University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

A study on image segmentation with convolutional neural networks

Text segmentation problem in manga

Pedro Henrique Barbosa de Almeida

Final Essay

mac 499 — Capstone Project

Supervisor: Prof. Dr. Nina S. T. Hirata

During this work, the author was supported by São Paulo Research Foundation (FAPESP), grant 2020/02891-3.

São Paulo
2020

Acknowledgements

"Ne me demandez pas qui je suis et ne me dites pas de rester le même"
— Michel Foucault

I would like to thank:

My parents, for always giving their best to provide me everything that I needed, for teaching me solid values of how to be a decent human, and for stand by me in every tough situation I faced in this journey.

My supervisor, for her supervision, attention, patience, didactic, empathy, and for opening doors in my academic journey.

My *fiancé*, for loving me, for being there for me in the hours of despair, for being my reviewer and critic.

My little brother, for being the motivation of my strength to overcome obstacles and of my desire to go beyond.

My college friends, for helping me in everything that I needed, for hearing me when I needed to let off steam.

My extra-college friends, for being my relief valve in these hard times.

IME-USP, for providing me the privilege of being in touch with highly skilled specialists and admirable professors that were inspirations to me.

FAPESP, for allowing me to dedicate myself exclusively to this project, through its grant.

Resumo

Pedro Henrique Barbosa de Almeida. **Um estudo sobre segmentação de imagens com redes convolucionais: *Problema de segmentação de texto em mangá***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2020.

O problema de segmentação de texto em mangás consiste em decompôr as imagens de páginas de mangá em regiões de "texto" ou "não texto". Essa tarefa de classificação pode ser modelada por uma função local, que será aprendida por um algoritmo de aprendizado supervisionado. Para aprender essa função, o algoritmo fará a minimização da diferença entre a saída esperada e a desejada, ajustando seus pesos. Neste trabalho, investigamos os impactos da variação de alguns parâmetros e aspectos de um modelo, chamado U-Net, em um contexto com poucos exemplos de treinamento. Para isso, construímos uma rede utilizando o framework Keras e a aplicamos no problema de segmentação de texto em mangás, a fim de medir seu desempenho. Dentre os resultados obtidos, notamos que um modelo com profundidade 3, normalização de entrada, com função de perda Generalized Dice Loss é robusto o suficiente para generalizar para diferentes títulos de mangás, mesmo quando é treinado em um título e testado em outro. Em resumo, obtivemos uma rede muito boa para esse problema, em termos de acurácia, precisão e revocação. Mesmo que não seja o modelo ótimo, poderá servir como uma referência para lidar com questões de pesquisa futuras.

Palavras-chave: Aprendizado profundo. Conjunto de treino pequeno. Mangá. Segmentação de texto.

Abstract

Pedro Henrique Barbosa de Almeida. **A study on image segmentation with convolutional neural networks: *Text segmentation problem in manga***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2020.

The text segmentation problem in mangas consists of decomposing images of manga's pages into "text" or "non-text" regions. This classification task can be modeled by a local function that will be learned by a supervised learning algorithm. To learn this function, the algorithm will perform a minimization of the difference between the expected and target outputs, adjusting its weights. In this work, we investigate the impacts of varying some parameters and aspects of a model, namely U-Net, in the context of few training examples. To achieve that, we built a network using the Keras framework and applied it to the text segmentation problem in mangas, to measure its performance. Among the results we obtained, we noticed that a network with depth 3, input normalization, and with generalized dice loss function is robust enough to generalize for different comic titles, even when it is trained on one title and tested on another. In summary, we have obtained a very good network for this problem, in terms of accuracy, precision, and recall. Even though it may not be the optimal model, it may serve as a reference to tackle further research questions.

Keywords: Deep learning. Small training data. Manga. Text segmentation.

List of Abbreviations

AH	AosugiruHaru
ANN	Artificial Neural Network
CCs	Connected components
CE	Cross-Entropy
CNN	Convolutional Neural Network
Concat	Concatenation
Conv	Convolution
EL	EvaLady
FCN	Fully-convolutional network
FL	Focal Loss
GDL	Generalized Dice Loss
JBF	JijiBabaFight
MSNN	MariaSamaNihaNaisyo
Std	Standard deviation
WCE	Weighted Cross-Entropy

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Manga refers to Japanese comics. A manga series is usually published in weekly or monthly released manga magazines, that includes an episode or chapter of each of the multiple manga series published in it. The chapters of a manga series are often compiled in book volumes, called *tankôbon*, containing a sequence of chapters published in the magazine.

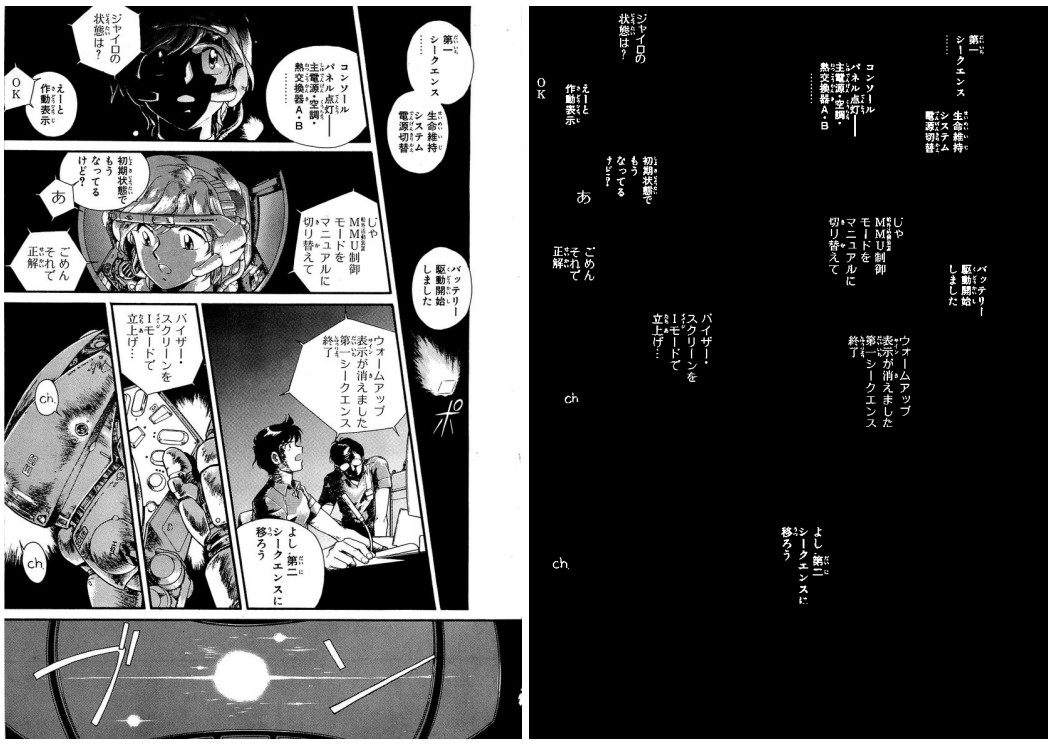
Mangas represent a large portion of the publishing industry in Japan. Some series are also translated and published in several international markets (such as in the United States or France). With increased digitalization, interest in the automatic processing of manga information has also been boosted. From a practical point of view, their automatic processing could speed up translation to multiple languages, enable better categorization based on content analysis (both textual and graphical) or the construction of a rich index, or even studies regarding cultural or social behaviors depicted in the stories.

The research community around manga image processing has been working on problems such as text detection, character (for instance, a person in the stories) detection, speech ballon detection, reading order detection, automatic inpainting, among others. Typically a scanned page or a photo of a manga page is processed individually.

Segmenting text from manga page images can be seen as an image segmentation problem. In image segmentation, we are interested in partitioning the image into regions such that each region carries semantic meaning. Typically, each region should correspond to objects, or parts of objects, to be further analyzed to extract information of interest.

The granularity of the segmentation may depend on the type of the desired information. In the text segmentation problem, we have the so-called binary segmentation problem, where the goal is to classify each pixel as being text or non-text. Figure 1.1 shows an example of a scanned page image and expected corresponding text segmentation.

Manually segmenting images is an unfeasible task for humans, since digital images can easily have more than millions of pixels. In this sense, a natural solution would be to design computer algorithms to perform the segmentation. However, designing a robust algorithm is challenging because typically there is great variability of fonts, scales, graphical elements, and orientations, as can be seen in Figure 1.2. Note that texts appear both in typeset fonts



(a) *Input* **(b)** *Target*

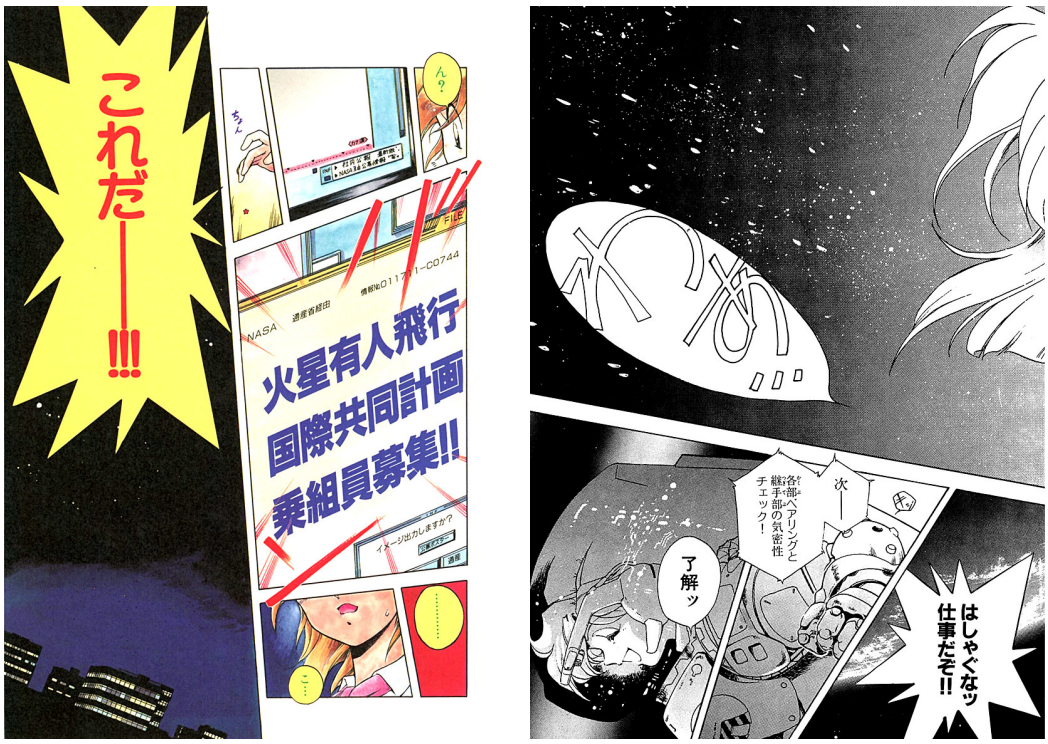
Figure 1.1: *Example of a manga page image and its corresponding text segmentation result (only typeset text). Page from the comic* EvaLady*©Miyone Shi from the Manga109 collection (*Matsui *et al., 2017; Aizawa et al., 2020).*

as well as in hand-drawn forms. Typeset fonts present different font sizes and styles.

In this scenario, machine learning-based techniques emerge as an interesting alternative. Since machine learning algorithms are data-driven, they could be trained to "learn" classification of pixels as text or non-text from training examples (such as the input-target pair of images shown in Fig. 1.1). If one needs to segment images with different characteristics, it suffices to redo the training with images from the new collection. Machine-learning-based approaches are widely used in image processing problems, not only in segmentation but also in classification, object detection, and other tasks.

For the text segmentation problem, a simple way to formulate the learning problem is to assume that the decision on whether a pixel is text or non-text can be done based on the analysis of a small patch of the input image centered on the pixel. More formally, this means that there is a function that can be locally applied, pixel by pixel. To learn this function, one has to decide what should be taken as the input (Coates *et al.*, 2011). It could be used the set of intensities of the image patch, taken in a specific order (for instance row by row), or a set of features extracted from this same patch. In any case, processing each of the pixels of an image individually in a sequential way may be computationally slow since images easily contain millions of pixels.

With the advance of computing resources and the availability of large volumes of data, there has been an impressive boost in machine learning techniques. In particular, it became possible to successfully train neural networks with a large number of parameters. During



(a) (b)

Figure 1.2: *Illustration of the variety of ways text appear in a manga. Page from the comic* EvaLady*©Miyone Shi*

this period, an architecture suitable for image processing, namely the Convolutional Neural Network (CNN), has emerged.

The first models were developed for the image classification task, where the goal is to predict which is object present in the image. Then, they have been extended for other image processing tasks such as segmentation. In segmentation, both input and output of the network are images. What makes CNNs particularly interesting for image segmentation is the fact that they can compute the predictions for every pixel at once, in parallel.

However, training of these networks requires a large amount of training data. In this work, we study a particular type of network, known as U-Net, and its application to the problem of text segmentation in manga page images. Although this is a widely used network for image segmentation, for a non-experienced user it is not obvious how to configure the hyperparameters of the network. It is also not very clear how much training data is necessary to achieve satisfactory results. Thus, we consider training scenarios with relatively little data (up to ten pages) and evaluate the effect of some of the settings.

Since an exhaustive exploration of possible configurations is unfeasible in a practical amount of time, we propose an incremental hyperparameter tuning scheme. At each step, we evaluate one or two aspects and choose the best values for them. In the subsequent steps, the values of the already evaluated hyperparameters are kept fixed, and new ones are assessed.

It is important to note that in this work we focus on understanding how the studied

hyperparameters affect learning and results, and not on building an optimal network that solves the aforementioned problem. Our experiments were carried using images from the Manga109 (Matsui *et al.*, 2017; Aizawa *et al.*, 2020) data set. We apply the final chosen configurations to four distinct manga series. Experimental results show that using few pages, sometimes less than ten, a good result can be achieved.

This text is organized as follows: in Chapter 2, we discuss some important concepts to the understanding of this study; in Chapter 3, we bring and comment on the projects decisions and the related works; following up, in Chapter 4, we present the experiments we have done towards finding good hyperparameters. Next, in Chapter 5, we present some results achieved with the best settings found so far; lastly, in Chapter 6, we elicit the conclusions we can take from the metrics observed.

Chapter 2

Background

In this chapter, we introduce some important concepts for the understanding of this study. We begin by defining what a digital image is, followed by the characterization of the segmentation problem, including examples of segmentation methods. We also introduce what are artificial neural networks and how convolutional neural networks differ from them. In sequence, we present the model investigated in this project, the U-Net. Ultimately, we discuss how these networks can be trained and evaluated, especially addressing the matter of loss functions and evaluation metrics.

2.1 Image segmentation

Images can be modeled as functions of the type $f : E \longrightarrow K$, where E represents the image definition domain (for instance a rectangular region in $\mathbb{Z}^2$) and K is the set of intensities associated to each point of the image. Typical intensity sets used are $K = \{0, 1\}$ for binary images, $K = \{0, 1, \dots, 255\}$ for grayscale images, and $K = \{(r, g, b) : r, g, b \in \{0, 1, \dots, 255\}\}$ for three-channel RGB color images. Images are computationally represented as matrices. In the RGB system, each image color is formed by the overlay of intensities from three different colors: red, green, and blue. Each one of these colors is represented by a matrix (or channel). The resulting colored image is then constituted of a stack of these three same-sized matrices, just like the schema shown in Figure 2.1, where each value at a matrix serves as the contribution of that color intensity to the final color.

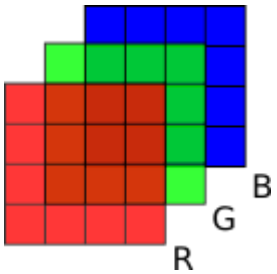


Figure 2.1: *RGB matrix representation.*

On the other hand, grayscale images are built of just one matrix, in which each value represents how "white" is the pixel color. In other words, if a pixel has value 0, its color is black; otherwise, if it is 255, then the color is white. Between 0 and 255, there are shades of gray.

Segmenting an image means partitioning the set of its points (often referred to as pixels) into several disjoint parts in which, usually, each part is associated with a semantic interpretation (Gonzalez, Woods, *et al.*, 2002). The granularity of the parts may vary according to further intended analysis. There are many methods for image segmentation such as SLIC (Achanta *et al.*, 2012), Watershed (Tarabalka *et al.*, 2010), connected components identification, binarization, among many others. For instance, in the context of manga images, a semantic unit computed by these algorithms could be a part of a letter, a letter, a group of letters, and so on. These basic units can be further grouped to generate a coarser segmentation, where each segment is related to a higher-level semantic interpretation. An example of a super-pixel level segmentation done by the SLIC algorithm is shown in Figure 2.2.

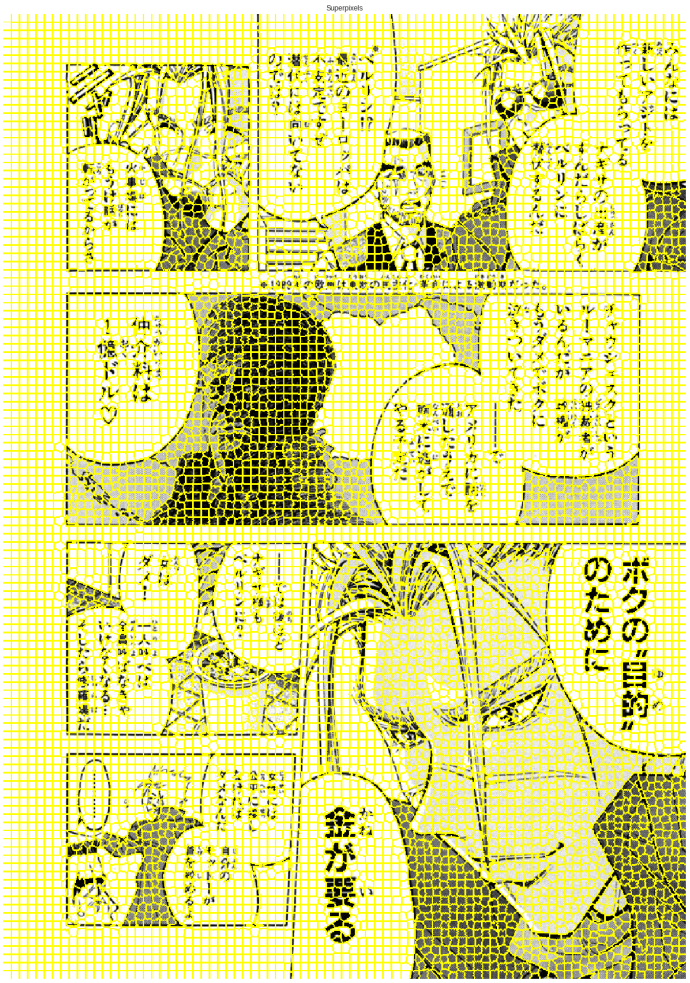


Figure 2.2: *Segmentation done by the SLIC algorithm. Original page from* Dollgun©*Deguchi Ryusei*

The segmentation problem can also be seen as a classification problem, where the classification task is to assign a semantic label to each of the many segments of an image.

Typically, one can consider individual pixel classification or classification of pixel groups previously computed (for instance using algorithms such as SLIC). For that, a representation for each of the segments should be computed to serve as input to the classifier. When classifying pixels, a straightforward representation for a pixel is to simply consider the set of pixel intensities of a small image patch centered at the pixel. In this case, the classifier would be a function that receives as input an image patch and outputs the semantic class label corresponding to the central pixel of the patch. An exaggerated schema of this approach is displayed in Figure 2.3. To compute the segmentation of an image, the classifier must be applied locally for every pixel in the image.

Figure 2.3: *Local function schema. Original page from* EvaLady*©Miyone Shi.*

This modeling allows us to formulate the image segmentation problem as a machine learning problem. The goal of the machine learning-based approach is to learn this local function from segmentation examples. As it will be further detailed in Section 2.4, this problem can be solved by fully convolutional networks that learn this local function implicitly and can compute the output values for all pixels in parallel. Fully convolutional networks have a set of parameters (weights) that are adjusted during training in such a way as to make the best mapping to the output image.

2.2 Neural Networks

An Artificial Neural Network (ANN) is an algorithm to learn to classify examples inspired on biological nervous systems (O'Shea and Nash, 2015). The basic unit in a Neural Network is a neuron. These units perform a dot product of the input values with the weights that are adjusted during training, plus a bias value. After that, a non-linearity, which is known as activation function, is applied. Mathematically, the output of a neuron is described in Equation 2.1, where σ is an activation function, $W \in \mathbb{R}^n$ is the weight vector, $b \in \mathbb{R}$ is the bias, and $x \in \mathbb{R}^n$ is the input vector. A scheme of a neuron with input of size $n = 3$ is shown in Figure 2.4.

$$f(x, W, b) = \sigma(W \cdot x + b) \tag{2.1}$$

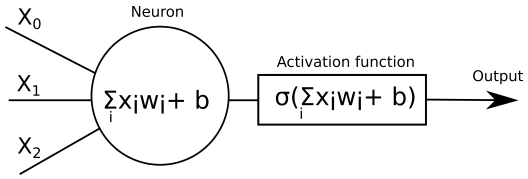


Figure 2.4: *Neuron schema.*

Neural Networks group neurons in layers. These layers, when they are between the input and the output, are called hidden layers. Each neuron in a layer, except in the input layer, is connected to all neurons of the previous layer. Thus, when building a network like this, the number of layers and the number of neurons in each layer are parameters that should be considered. See Figure 2.5 for an example. It is an architecture consisting of 16 nodes in the input layer, two hidden layers, and a final output layer with only one node.

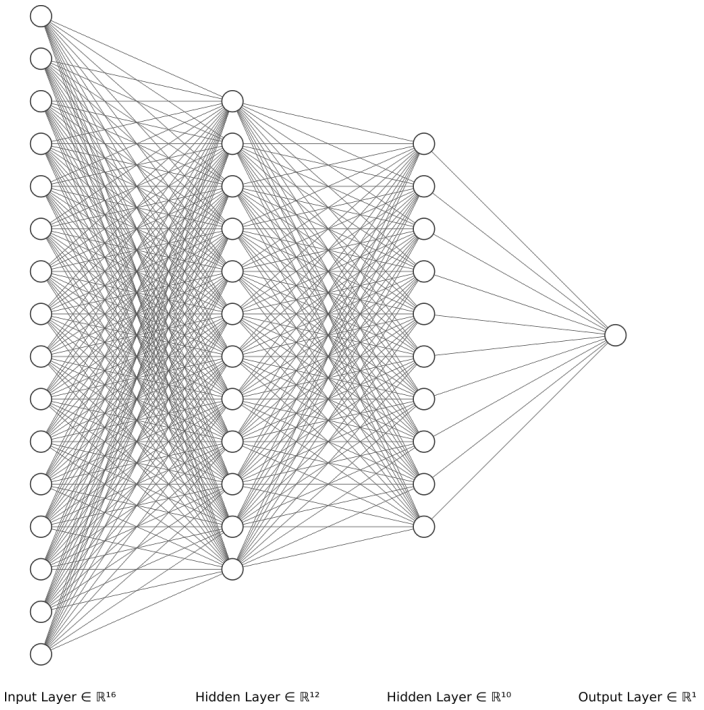


Figure 2.5: *Neural Network organization.*

The last layer is known as the output layer and it returns a prediction. It calculates the final output as a combination of all the previous neurons.

The weights are learned by minimizing a loss function. A loss function measures the discrepancy between the expected target output and the output predicted by the network. In other words, the minimization is performed by an algorithm called Gradient Descent that updates the weights so that the discrepancy is reduced. The weight update consists of subtracting a proportion of the loss function gradient (in this way, weights are “moved” in the direction of decreasing slope of the loss function). This proportion used in the update is determined by a small constant, called the learning rate.

The problem is that these networks do not scale well for images, if you consider each

pixel as a feature, because an image can easily reach more than a million pixels. In this case, Convolutional Neural Networks came to work around this issue. Although it still uses neurons, they are not connected to all the previous units, as we explain in Section 2.3.

2.3 Convolutional Neural Networks

A convolution is a linear operation between functions that calculates the integral of the intersection of these functions' shapes when one of the shapes is shifted through the domain. In digital image processing, we can depict this operation as the dot product[1] of a kernel (a function defined on a "window") and its intersection with a region of the image. More specifically, this kernel has its center slid throughout the image pixels, and the dot product is the result for the pixel on the kernel's center. A kernel is also called a "filter".

Figure 2.6 depicts the first step of a 3x3 (the kernel size) convolution. The center of the kernel (filter) is highlighted in light red. This center of the filter is positioned in the pixel of the input image that is pointed up in light blue. The output of this first step is the value emphasized in light green in the output image, denoted feature map in the figure. Note that the output feature map size is smaller than that of the input image. This is because the convolution operation can not be computed for the pixels in the border rows and columns of the input image.

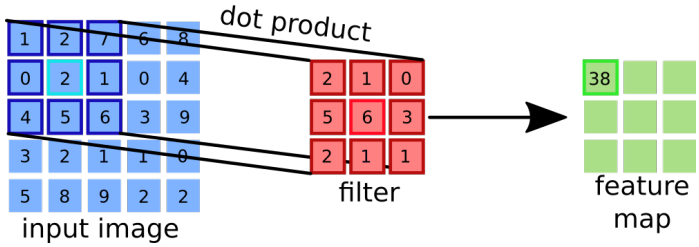


Figure 2.6: *Convolution schema.*

In a Convolutional Neural Network (CNN), there are usually four main layers: convolution, activation, sub-sampling, and fully connected layers. It is possible to check an example of this type of architecture in Figure 2.7.

The convolutional layer is where the aforementioned operation is implemented. In this layer, there are many stacked kernels (or filters) that convolve throughout the image. When the input consists of multiple channels (i.e., a volume), in standard implementations a filter consists of a set of kernels (one kernel for each input channel). The values of the kernel's matrix are the weights that will be learned during the optimization of the loss function. Each filter generates a feature map, that is a single-channel image (if the input

[1] It should be noted that the mathematical definition of convolution considers the transposed kernel. The dot product without transposition corresponds actually to what technically is known as cross-correlation (P. Soille, 2003, p.25). However, as, in the context of convolutional neural networks, these weights will be learned, often one does not worry whether they should be viewed as a transposed kernel or not.

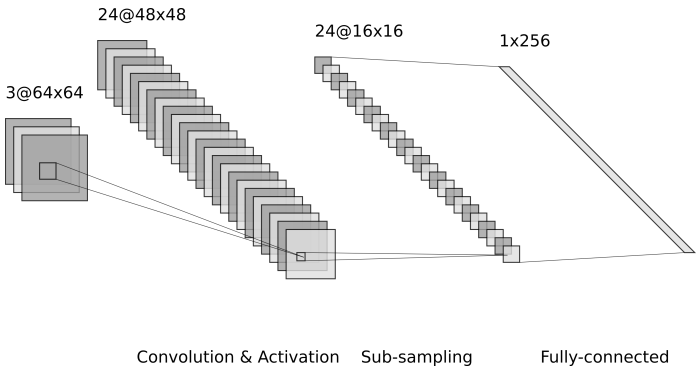


Figure 2.7: *Convolutional Neural Network organization.*

has multiple channels, then individual channel convolutions are summed up). Multiple filters generate a feature map volume. As already mentioned, the filters are not “connected" to all pixels at a step as it happens with standard neural networks. They are applied exactly as explained above (and as illustrated in Figure 2.6). In particular, this means that for convolutional layers the input size does not need to be fixed.

As every filter has its weights, it is commonly said that each filter learns to detect particular characteristics of the input images, like, colors, or vertical edges, diagonal edges, and so on. After the convolution is done, an activation function is usually applied pixel-wise on the feature map. ReLU is usually the activation used in this model (O’Shea and Nash, 2015).

The receptive field of any point in a feature map is the region of the input image that determines the value of that point in the map. For instance, if we apply a 3×3 convolution on an input image, the receptive field of any point in the resulting map is a 3×3 region. If we apply a second 3×3 convolution, now on the resulting feature map, the receptive field of any point in the new resulting map will be a 5×5 region. The receptive field of a network is the area that the network effectively "sees" to calculate the output value. The larger the number of convolutional layers, the larger will be the network’s receptive field.

In the sub-sampling layer, the volumes have their dimensions reduced. This operation can be characterized by a “window” that applies an aggregation function to the pixels it is hovering. The aggregation function can be a max operation (that returns the maximum value), a min operation, or even an average operation. Max-poolings are the most common in CNNs. Typically, each bock of 2×2 pixels (without superposition) is replaced with the aggregated value. In this case, both the number of rows and of the columns of the map are reduced in half, reducing the map to a quarter of its input size. This layer helps to reduce the number of learnable parameters that should be trained in the network.

Lastly, the fully-connected layers implement the network described in Section 2.2. The last feature map is flattened (converted to a 1D structure) to work as the input to the fully-connected part of the CNN. Although convolutions do not restrict the input image size, since a fixed size flattened map is expected in the fully-connected part of a CNN, the input of CNN must have a fixed size (unless some processing to reduce any input size image to a fixed feature map size is employed). At the end of a CNN, there is an output

layer. If CNNs are used for classification, the number of nodes in the output layer is equal to the number of classes, and they are usually followed by a softmax activation function. The softmax function guarantees that the sum of the scores of the output nodes will be 1, allowing us to interpret those scores as the likelihood probabilities of the object concerning each of the classes.

2.4 U-Net

CNNs, described in the previous section, are typically used in image classification tasks, where the input is an image and the target is a class label. In image segmentation tasks, however, the targets are also images. Therefore, some adaptations to CNNs were made to make them useful for segmentation tasks. The main idea is to use fully convolutional networks (Long *et al.*, 2015).

Fully convolutional networks (FCNs) (Long *et al.*, 2015) are CNNs without the fully-connected layers. This, in particular, means that the network can process images of any size. However, due to the pooling layers, the resulting map is smaller than the input image. To fix this, upsampling operations (implemented as convolutions with a particular shape kernel) are employed, in such a way that the resulting map, at the end of the network, has the same size as the input image. Segmentation is then viewed as a learning problem where the target are the individual pixel values in the target image.

After FCNs (Long *et al.*, 2015), many other fully-convolutional architectures have been proposed. Among them, one that became well-known and used in many segmentation problems is U-Net (Ronneberger *et al.*, 2015). This network is divided into two parts: the contracting path (in which the volumes have their dimensions reduced), and an expanding path (in which volumes have their dimensions increased), as illustrated in Figure 2.8.

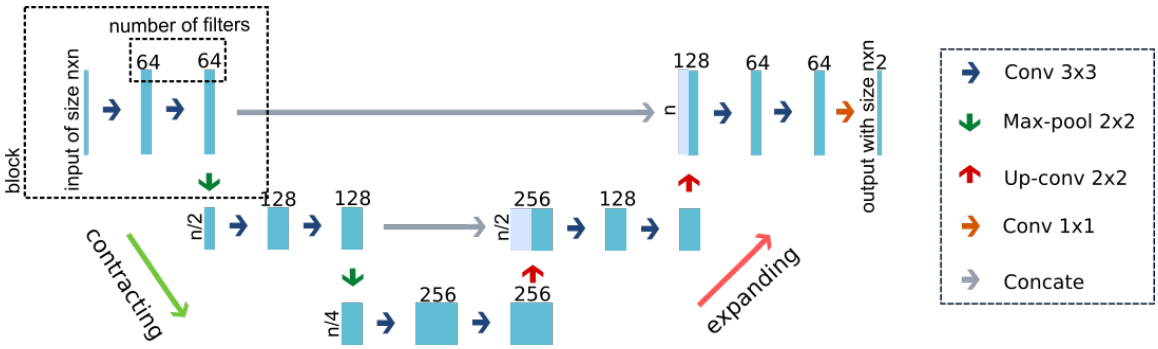


Figure 2.8: *U-Net diagram.*

In the contracting path, there might be one or more blocks of two convolutional layers (dashed rectangle highlighted in the left part of the diagram in Fig. 2.8), with ReLU activation and kernel size of 3x3, followed by a 2x2 max-pooling layer. After one block of this, the volume has its width and height halved. The number of blocks of two convolutions and one max-pooling determine what is called the depth of the U-Net. Moreover, the convolution immediately before the max-pooling has its filters concatenated to the filters of the first convolution after the up-convolution (further explained) at the same depth in the expanding path.

On the other hand, in the expanding path, there are blocks of a 2x2 up-convolution layer, which is an up-sampling succeeded by a 2x2 convolution, followed by two 3x3 convolutional layers. After one block of this, the volume has its width and height doubled. As previously mentioned, the convolution immediately after the up-convolution layer has its number of filters doubled by the concatenation with the filters from the last convolution of the block from the same depth in the contracting path.

Finally, the output image is calculated by a 1x1 convolutional layer with Softmax activation. The 1x1 convolutional layer is used to reduce a volume to a 2D map. As the volume is padded[2] before every convolution in such a way that its height and width remain the same before and after the processing, the output image has the same size as the input.

The concatenations (shown as gray color horizontal lines) are also called skip connections, and it is argued that they help the network to recover the correct position of the information (lost due to the pooling layers in the contracting path). This is one of the main differences of U-Nets when compared to the initially proposed FCNs. The name U-Net is due to the u-shaped way the architecture can be drawn. Often, the number of filters (f) at each convolution is calculated exponentially on the depth (d) of the network, i.e., $f = 2^{(5+d)}$. Note, however, that variations of the architecture concerning block characteristics, upsampling methods, among others, are present in the literature.

2.5 Neural network training and evaluation

So far we have described what neural networks are and how they work. In this section, we describe how their weights are adjusted based on training data. Once a network is trained, it is important to evaluate its performance. We describe some commonly used performance evaluation metrics.

2.5.1 Loss functions

As already hinted in Section 2.2, training of a neural network consists of adjusting the weights of the network based on the responses of the network for the training data. This adjusting is made iteratively using the backpropagation algorithm. At each iteration, a batch of training data is processed by the network, and predictions are generated for each example in the batch. Then, the loss between the predicted outputs and the expected target is computed and used to guide the weight updating. Every time the whole training set is processed, an epoch of the training process is completed. The training process usually is iterated until a predefined number of epochs is completed or until no decrease in the loss is observed.

The loss functions are the equations that measure how far or near are the predictions from the target outputs. Each loss function “sees" the differences in a particular way, thus, their outputs range in different intervals. Loss functions used in segmentation problems are mostly derived from the loss functions used in classification problems. In segmentation,

[2] In padding, a suitable number of rows and columns are added at the top, bottom, left and right sides of the maps, usually filled with value zero.

one can view each pixel in the target image as an individual classification target. This way, the loss for a segmentation target can be computed as an aggregated value of the classification loss computed to each of the target pixels in the target image, just as it is done in classification problems to examples in the training batches.

A challenge arises when we have a very imbalanced class distribution. For instance, in the text segmentation problem considered in this work, there are many more negative (i.e., non-text) than positive (i.e., text) pixels in the target image. For that reason, different loss functions try to minimize the effects of class imbalance, such as the ones described next.

We consider only binary classification, where 1 denotes the positive class and 0 denotes the negative class. Given a target class t_n (0 or 1), let p_n be the score predicted by the network for the positive class. Then, $1 - p_n$ is the score for the negative class. Let us also suppose the number of pixels in the target image is N.

The standard cost function is Cross-Entropy loss (CE, eq. 2.2):

$$CE = -\frac{1}{N}\sum_{n=1}^{N}(1 - t_n)log(1 - p_n) + t_n log(p_n)\,. \tag{2.2}$$

Note that when $t_n = 1$ and p_n agrees, we have $t_n log(p_n) = 0$ and $(1 - t_n)log(1 - p_n) = 0$, and then example n would contribute with zero to the cost. On the other hand, when $t_n = 1$ and p_n disagrees, for instance if $p_n \sim 0$, we would have $t_n log(p_n) \sim -\infty$ contributing a lot to the cost. When $t_n = 0$, similar reasoning applies. Note also that if there is large class imbalance, then in most of the times only one of the terms will contribute to the cost and therefore the training process will tend to force the predictions to be correct for examples of the predominant class, leading to a bias toward the most frequent class.

The Weighted Cross-Entropy loss (WCE, eq. 2.3) (Panchapagesan *et al.*, 2016) is a straightforward way to fix the imbalance problem of CE:

$$WCE = -\frac{1}{N}\sum_{n=1}^{N} w_0 \cdot (1 - t_n) \cdot log(1 - p_n) + w_1 \cdot t_n \cdot log(p_n)\,. \tag{2.3}$$

The main idea is to assign a larger weight to the less frequent class so that errors in classifying examples of that class will contribute more to the cost.

The focal loss (Lin *et al.*, 2017) is another attempt to reduce the effects of class imbalance:

$$FL = -\frac{1}{N}\sum_{n=1}^{N} \alpha t_n(1 - p_n)^\gamma log(p_n) + (1 - \alpha)(1 - t_n)(p_n)^\gamma log(1 - p_n)\,. \tag{2.4}$$

According to the authors (Lin *et al.*, 2017), α balances the importance of positive/negative examples, without differentiating easy/hard examples, and at the same time, the easy examples are down-weighted with a modulating factor $(1 - p_n)^\gamma$, with a focusing parameter γ.

The Generalized Dice loss (GDL, eq. 2.5) (Sudre *et al.*, 2017) is defined as:

$$GDL = 1 - 2\frac{w_0 \sum_n t_n \cdot p_n + w_1 \sum_n (1 - t_n) \cdot (1 - p_n)}{w_0 \sum_n (t_n + p_n) + w_1 \sum_n [(1 - t_n) + (1 - p_n)]}, \tag{2.5}$$

where weights w^0 and w^1 are usually calculated with the inverse of frequency from each class as $w^0 = \frac{1}{\left(\sum_n (1-t_n)\right)^2}$ and $w^1 = \frac{1}{\left(\sum_n t_n\right)^2}$.

2.5.2 Performance Evaluation metrics

While the cost function is used to guide the training process, for evaluating the performance, multiple metrics can be used. The prediction of the network is first converted to a class label. In binary classification problems, if p denotes the score output by the network for the positive class, a usual decision is:

$$\hat{y} = \begin{cases} 1, & \text{if } p > 0.5, \\ 0, & \text{otherwise.} \end{cases} \tag{2.6}$$

Given a set of examples, one can compare the predicted label $\hat{y}$ to the target label for each example and count true positives, true negatives, false positives, and false negatives denoted respectively as TP, TN, FP, and FN.

The accuracy (Sokolova *et al.*, 2006) expresses the proportion of correctly classified (positives and negatives) examples in the set. Precision (Davis and Goadrich, 2006) measures the proportion of correctly classified positive examples, taking into account the total of examples that were predicted as positives. In the meanwhile, recall (Davis and Goadrich, 2006) measures the percent of correctly classified positive examples, among truly positives examples present in the set. Their expressions are shown below:

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.7}$$

$$precision = \frac{TP}{TP + FP} \tag{2.8}$$

$$recall = \frac{TP}{TP + FN} \tag{2.9}$$

Chapter 3

Manga Text Segmentation

In this chapter, we briefly review some works on manga text segmentation. We then describe the datasets (manga series) we have used in the experimental part and the details of the study done in this work.

3.1 Some related works

To train a machine learning algorithm, training data is needed. For the problem considered in this work, for each manga page we need to build the target image (also called ground-truth), containing only the text components.

Bounding boxes are the most common ground-truth annotation available for text detection in mangas. It comprehends a bounding box and a class label. The bounding box is defined by four values: x_{min}, x_{max}, y_{min}, y_{max}, that represent a rectangle. For visualization purposes, these boxes are often overlaid on the image, with each class identified by a specific color. An example from Aramaki *et al.*, 2016 is shown in Figure 3.1. Note that both typeset and hand-drawn text are annotated.

Given ground-truth in the form of bounding boxes, a common idea explored for text detection is to build region proposals and then classify them as text or non-text. To build region proposals, Aramaki *et al.*, 2016 first classify connected components as text or non-text based on geometrical features and then group components classified as text into rectangular regions. Then these regions are classified as text or non-text using features extracted with pre-trained deep networks and the SVM algorithm.

Chu and Yu, 2018 employ object detection methods commonly used in generic computer vision tasks, and report results comparable or superior to the ones in Aramaki *et al.*, 2016. Region proposals are obtained using the selective search algorithm. Features extracted with CNNs are used for classification and also for regression (to learn the region offsets). Authors report that using this method, a similar performance to the ones reported in Aramaki *et al.*, 2016 were achieved, and even better performance was obtained using the well-known Faster-RCNN model.

In Piriyothinkul *et al.*, 2019, final text regions are built by first applying the Stroke

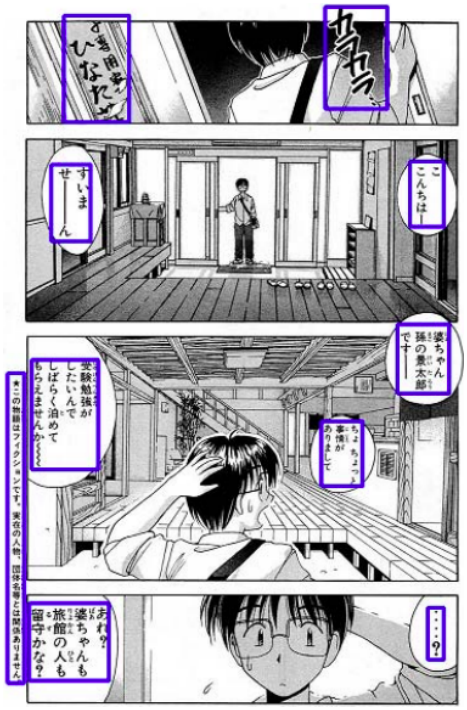


Figure 3.1: *Example of bounding box annotation used in* Aramaki *et al., 2016 as ground-truth.*

Width Transform on the input image to find letter candidates, then classifying them as a letter or non-letter using SVM, and finally grouping the components classified as letters in text regions based on heuristic criteria. Authors report better results than the ones in Aramaki *et al.*, 2016.

These works all use up to 100 pages for training and 100 pages for evaluation (although not the same pages) from 6 manga series in the Manga109 data set. They all follow the evaluation method of the first work (which is based on metrics that compute overlaps between the ground-truth boxes and predicted boxes).

Pixel-level segmentation is less explored (Hirata *et al.*, 2016; Ko and Cho, 2020). In this approach, each image pixel is mapped to a class. In Hirata *et al.*, 2016 a method for pixel-level segmentation modeled as an image transformation problem and computed based on a sliding window approach is presented, however without any quantitative evaluation. In Ko and Cho, 2020, a method for inpainting text region in mangas is proposed. This method includes segmentation of typeset as well as hand-drawn text as an intermediary step to help to remove the text before inpainting. For that, a U-Net-like architecture is used, with a total of 200 training, 57 validation, and 28 test images. Since they had no images annotated at the pixel level, according to authors, one part of the training data was generated relying on the available bounding box annotations and the other part was manually segmented. The performance was then evaluated in terms of the IoU metric since only bounding-box ground-truth was available. According to the authors, two loss functions, weighted cross-entropy, and Jaccard were tested and the best performance was obtained with the latter.

U-Net-like models are also used for speech balloon detection (Dubray and Laubrock,

2019), which is an effective way of detecting text when most of them are located within speech balloons.

3.2 Data used in the study

The comics used in this work are from the Manga109 data set, which compiles volumes from 109 series from Japanese artists that were published between 1970 and 2010 (Fujimoto *et al.*, 2016). Images in these collections correspond to comic pages, organized per volume, and they are available upon request according to instruction in the Manga109 site[1]. Most pages are in gray-scale and a few of them (typically the cover pages) are in color. The data used in this work has been downloaded a few years ago when annotations were not available for all volumes in the collection. On that occasion, target images – binary images with value 0 if the input pixel is not part of a text character, or 1 if the input pixel is part of a text character – have been prepared for some of the volumes or part of some volumes[2].

We selected four series that have target images: AosugiruHaru (AH), EvaLady (EL), JijiBabaFight (JBF), and MariaSamaNihaNaisyo (MSNN). Examples of input images from these comics are shown in Figure 3.2. Some additional information about each of the series can be checked in Table 3.1.

Series	Author	Age	Publisher	Target	Genre
AH	Okuda Momoko	2000's	Shueisha	lady	love romance
EL	Miyone Shi	1990's	Takeshobo	boy	science fiction
JBF	Nishikawa Shinji	1990's	Kodansha	boy	humor
MSNN	Konohana Akari	1990's	Shueisha	lady	love romance

Table 3.1: *Data about the four comics used in this work*

[1] http://www.manga109.org/en/download.html

[2] Target images were manually prepared by the supervisor of this work by first converting the input image to a binary image and then "erasing" non-zero pixels that are not part of text characters (Hirata *et al.*, 2016).

(a) *Page from* AosugiruHaru©*Okuda Momoko.*

(b) *Page from* EvaLady©*Miyone Shi.*

(c) *Page from* JijiBabaFight©*Nishikawa Shinji.*

(d) *Page from* MariaSamaNihaNaisyo©*Konohana Akari.*

Figure 3.2: *AosugiruHaru, EvaLady, JijiBabaFight, and MariaSamaNihaNaisyo are from Manga109 collection (Matsui et al., 2017; Aizawa et al., 2020). .*

The total number of available pages varies from volume to volume. From the total, we kept only those that have the corresponding target image. Some images were discarded because they had atypical text formatting (for example, summary, cover, and credits pages) or even because they did not have any text. RGB images were also excluded, keeping only the grayscale examples. The number of pages we considered by series is in Table 3.2.

Series	Number of pages
AH	105
EL	185
JBF	20
MSNN	192

Table 3.2: *Number of pages considered per series*

For each series, for the pages in Table 3.2, we computed the statistics regarding the number of pixels belonging to each of the two classes (text and non-text). Besides that, since each text character is, in general, a connected component (the maximal group of adjacent text pixels) in the target image, we also computed metrics regarding their area, height, and width, in terms of the number of pixels. The standard deviation (std) of the metrics on connected components (CCs) were taken over the averages per image. Those statistics can be seen in Table 3.3.

		Pixels		Connected components			
Series	**Value**	**"non-text"**	**"text"**	**#**	**Area**	**Height**	**Width**
AH	Average	961839.491	5750.510	81.471	77.799	12.676	11.451
	Std	4422.614	4422.614	65.271	38.304	3.223	3.228
EL	Average	951005.254	16584.746	191.914	132.274	13.578	12.187
	Std	16036.411	16036.411	189.274	188.759	5.645	4.859
JBF	Average	955582.100	12007.900	261.100	46.717	9.807	8.190
	Std	4365.898	4365.898	100.150	5.827	0.899	0.568
MSNN	Average	957618.492	9971.508	135.417	160.300	14.404	12.180
	Std	7277.946	7277.946	76.126	620.957	17.424	11.759

Table 3.3: *Statistics about the number of pixels and connected components (number, mean area, mean height, and mean width)*

From Table 3.3, we see that MSNN has the biggest CCs, while JBF the smallest, considering pixel area. CCs from JBF are also more alike since the standard deviations of all the metrics (area, height, and width) are the smallest. Differently, the metrics from MSNN are more diffuse, since this comic has the highest standard deviations. JBF is also the comic that has more CCs in the set, whilst EL, the lowest.

3.3 Proposed study

The choice of using U-Net was based on reports in the literature. When training neural networks, an important issue is to find the “right” configuration (network architecture and

hyperparameters). The "right" configuration is strongly related to the complexity of data (in terms of input-target relationship) and also the amount of available training data.

In some sense, we started the study assuming that U-Net would be able to produce good results based on reports about its successful training on related or similar tasks to the one considered in this work. For instance, as mentioned in Section 3.1, Ko and Cho, 2020 used U-Net as part of their solution for erasing text pixels before performing inpainting. However, they used around 200 images for training the U-Net.

Our interest, however, is to make use of few training pages. This would enable the quick construction of simple customized solutions that work very well for a specific series rather than a more complex model designed to work well on average on a large collection of series. Since we have found no similar works, i.e., one that is concerned with text segmentation in manga page images with U-net trained with few pages, we decided to focus on two aspects:

- An experimental procedure to configure the hyperparameters of U-Net concerning a specific series, and how well it can be tuned when using only a few training pages;
- Evaluation of the chosen configuration when trained on data of other series.

What is little training data? The first thing we considered is what would be a reasonable number that characterizes "few training pages". We fixed 10 as the maximum number of pages to be used in training.

Then, our next concern was to decide how to select 10 pages. Since pages present different densities of text data, and in general, the number of text pixels is much smaller than the number of non-text pixels, to reduce the impact of class imbalance one could select the 10 pages with the most text pixels. On the other hand, we could just select ten random pages or the ten first pages of the volume. We built three training sets. From each series volume, we chose (i) 10 pages with the most CCs in the target image (D_1) – CCs in the target image works as a proxy for the number of text characters in the page, (ii) 10 first pages (D_2), and (iii) 5 pages with the most CCs in the target image (D_3) – D_3 is a subset of D_1. Additionally, from the remaining pages in each volume, we randomly[3] selected, without replacement, 5 pages to be used as a validation set V_1 (used during training) and 5 pages as the test set V_2.

Thus, D_1 is designed in a way to privilege the presence of text, and D_2 is just the set of the first ten pages. Set D_3 was built to study the effects of a even smaller training set. The properties of these subsets are:

1. $D_3 \subset D_1$;
2. $V_1 \cap V_2 = \emptyset$;
3. $V_1 \cap D_1 = \emptyset$, $V_1 \cap D_2 = \emptyset$, $V_1 \cap D_3 = \emptyset$;
4. $V_2 \cap D_1 = \emptyset$, $V_2 \cap D_2 = \emptyset$, $V_2 \cap D_3 = \emptyset$.

[3] For random choices, we used 42 as a fixed random seed for numpy operations.

Sanity check: Before performing a systematic effort to find a network configuration that produces good results, we first verify if we can successfully train a default U-Net. That is, we would like to verify if we can make the loss decrease along with the training iterations. For that, we established a U-Net with depth three with default hyperparameters.

Network configuration: Selecting the right hyperparameters is a hard task since deep architectures present a wide miscellany of settings to tune. Moreover, each of these hyperparameters varies in a broad range, which makes a grid-search approach computationally unfeasible. An alternative approach is to evaluate hyperparameters one at a time, sequentially. In each step, one or two hyperparameters would be evaluated and the best choices found at each step would be, then, propagated to the next steps. This is the approach we chose for this study.

The ordering of hyperparameters to be tested was established based on a subjective evaluation of importance, according to our understanding.

1. U-Net depth: We started by adjusting the model capacity of overfitting the training data, to obtain a model of the "right size".
2. Data normalization: We also experimented with different data normalization strategies, together with depth variation. Normalization is known to affect convergence speed or sometimes be the detail that sets apart training that converges from the ones that do not.
3. Cost function: Given the high imbalance of classes in our data (even in D_1), the effect of distinct loss functions on performance is an interesting point to be investigated. For that, we consider some cost functions that were designed to reduce the impact of the imbalance by giving different weights for each class, such as Generalized Dice Loss, Weighted Cross-Entropy, and Focal Loss (see Section 2.5.1).

Roughly, the training is performed either on set D_1 or D_2 and the best hyperparameter values at each step are chosen based on the validation performance, computed on set V_1. Details are presented in Chapter 4.

The next question is to study whether the configuration optimized for one series would work equally well on other series. Thus, we train the configuration selected in the previous part on each of the series. In particular, we analyze their performance and also how they behave when they are tested on a series distinct to that one on which they have been trained. This comparison is interesting since each comic has its particular characteristics. Lastly, another aspect that we explore is to vary the number of training examples. Specifically, our interest is to investigate how the performance would vary as we diminish the training set size.

The experiments and results regarding the hyperparameter tuning (to select a "right configuration") were performed using EvaLady and are described in the next chapter. The experiments regarding the training of the selected network configuration on other series, as well as other types of evaluations, are presented in Chapter 5.

Chapter 4

Experiments on network configuration

In this chapter, we report the experiments and results regarding the network configuration tuning (see details in Section 3.3) performed for the EvaLady series. We start with an initial default configuration, used as a sanity check, and then in every subsequent section, we highlight the parameters that were evaluated. The sequence is as follows:

1. Training models with different depths;
2. Training models with different normalization techniques;
3. Training models with different loss functions;
4. Training models on different training subsets;

The experiments were done using Python as the programming language [1] and the Keras library[2]. During training, the loss evolution on the validation set V_1 was monitored and the weights at the moment the network achieved the least loss were saved. This is a common practice known as checkpointing. We also compute performance metrics on the test set V_2. However, it is important to emphasize that evaluations focus only on the loss computed on training and validation sets.

4.1 Default model performance

We initially set a default architecture and training hyperparameters: a U-Net architecture with depth 3 (the one presented in Fig. 2.8), Categorical Cross-Entropy as a loss function, learning rate 10^{-4}, Adam optimizer, batch size of one image, input patch size equals the full image, no data normalization, and 100 epochs of training. The learning curve with training set D_1 of EvaLady is shown in Figure 4.1.

[1] Python is an interpreted programming language, see more: https://www.python.org/

[2] Keras is the Python framework used to build the model. More info: https://keras.io/

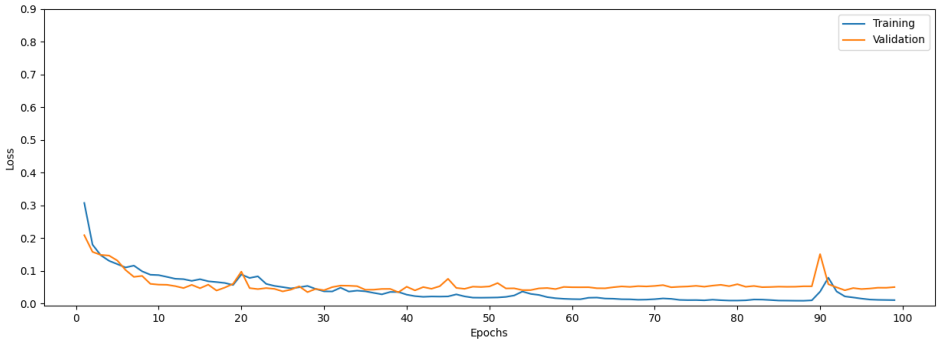


Figure 4.1: *Learning curve of the baseline model.*

As can be seen, the model starts to overfit around epoch 30, that is when the training loss continues decreasing while the validation curve keeps stable. It is also important to note that the training is converging, i.e., the weights are being optimized, as the loss is coming down and approximating to zero. This is an indication that this default configuration is being able to "learn".

4.2 Training models with different depths

We start tuning by evaluating the depth of the U-Net model. Other hyperparameters were kept. Depth is the parameter that determines the number of convolutions and max-pooling blocks in the contracting path (or, equivalently, the number of convolutions and up-sampling blocks in the expanding path). Strictly speaking, the deeper the network, the more parameters to learn, and greater it is its learning capacity. We have experimented with depths 1, 2, 3, 4, and 5. The behavior observed on the training and validation sets are shown in Figure 4.2. Additional details are presented in Table 4.1.

depth	seconds elapsed during training	min loss on training	min loss on V_1	accuracy on V_2	precision on V_2	recall on V_2
1	2361.793	0.088	0.079	0.988	0.433	0.193
2	10835.544	0.028	0.031	0.994	0.735	0.670
3	15516.197	0.008	0.034	0.993	0.753	0.578
4	19371.455	0.441	0.227	0.989	-	0.000
5	22481.266	0.441	0.227	0.989	-	0.000

Table 4.1: *Metrics of different depths with no normalization*

From the learning curves (Fig. 4.2) that converged, we see that depth 1 presented the worst result, while comparison between depth 2 and 3 is not easy. For depths 4 and 5, the curves did not come down, despite the slight fall in the very beginning. Out of curiosity, validation after the training (on V_2, Tab. 4.1) shows that depth 3 presented the best accuracy and precision. For finding out which is the "right network size", it is often useful to verify the training loss. If the training loss approaches zero, that means that the network has sufficient capacity. Based on the training loss curves, we see depth 3 is the one with the

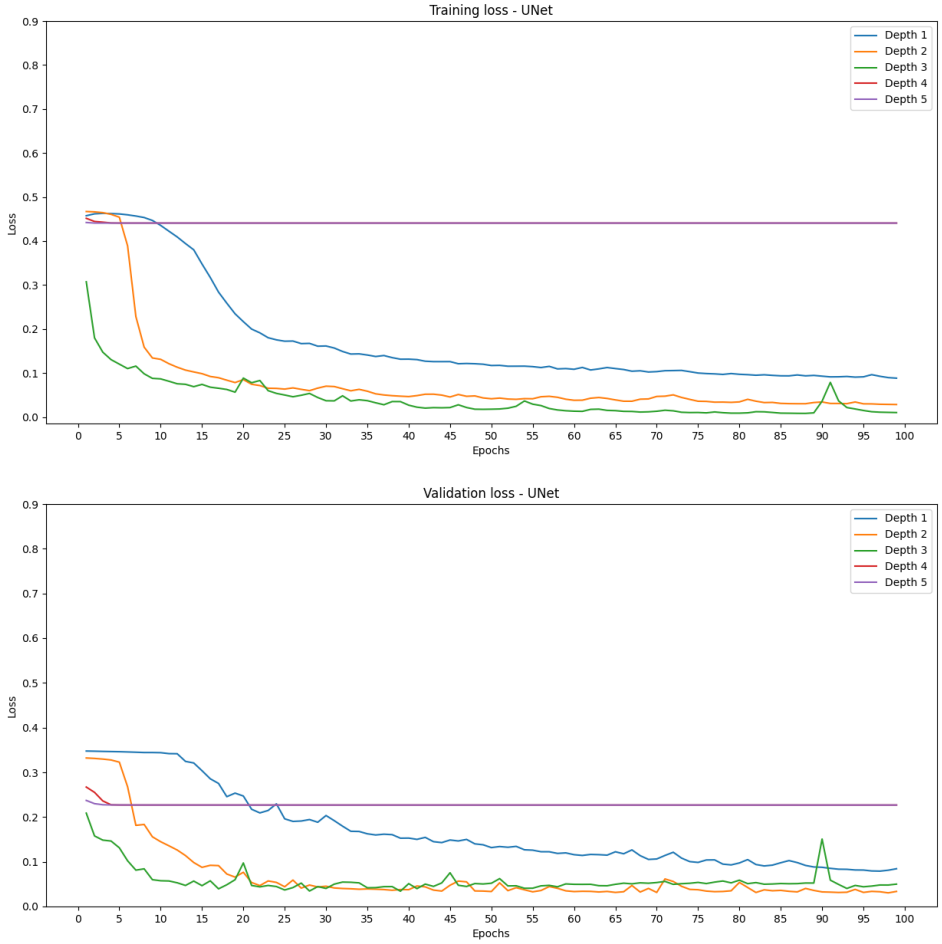


Figure 4.2: *Learning curves of depths with no normalization.*

best capacity. However, this is not conclusive because we were not able to successfully train networks with depth 4 and 5.

4.3 Training models with different normalization techniques

Following up, we tested different normalization schemes for the already mentioned five depths. Input normalization, batch normalization, and both normalizations combined were the tested configurations. It is expected that normalization will help the convergence of the learning curves.

4.3.1 Input normalization

With this normalization, every pixel intensity of the inputs is converted to the $[0, 1]$ interval. In order to do so, all values are divided by the maximum intensity, which is 255. Other parameters without exception were the same from the previous step. The learning curve for this case is shown in Figure 4.3.

In this scenario, every training converged. We can see that concerning training, a network with depth 1 presents a poor result, a network with depth 2 is better but also clearly worse than the others. Networks with depth 3, 4, and 5 present similar losses on training data. However the same is not observed on the validation set V_1. Moreover, even though the model with depth 5 had the best metrics on V_2 validation, it was 40% slower than depth 3 to train (Tab. 4.2). Similar to the case of depth 4, which is the second-best and is 21% slower than depth 3, in terms of time in seconds to learn. On the other side, depth 3 also achieved a 99% accuracy.

depth	seconds elapsed during training	min loss on training	min loss on V_1	accuracy on V_2	precision on V_2	recall on V_2
1	2298.055	0.084	0.055	0.989	0.484	0.020
2	10439.136	0.034	0.025	0.993	0.826	0.516
3	16006.453	0.009	0.035	0.990	0.932	0.090
4	19428.981	0.008	0.041	0.995	0.834	0.689
5	22557.089	0.006	0.029	0.997	0.941	0.763

Table 4.2: *Metrics of different depths with input normalization.*

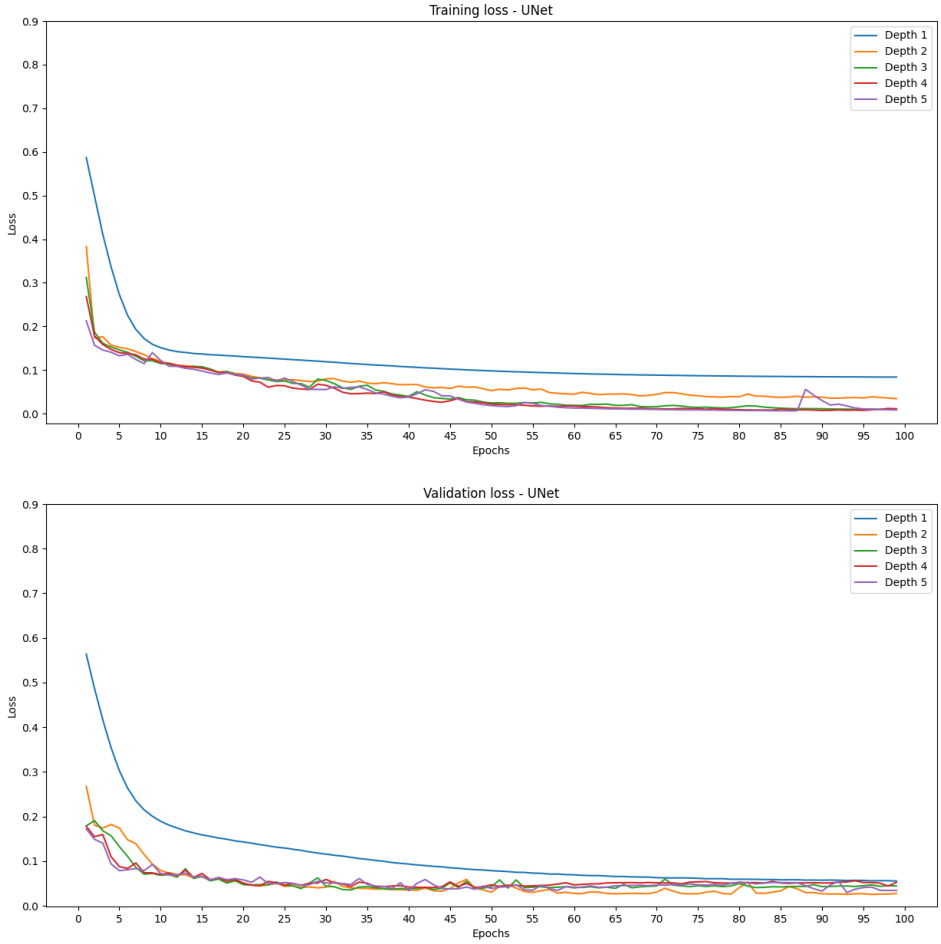


Figure 4.3: *Learning curves of depths with input normalization.*

4.3.2 Batch normalization

This normalization is done by a layer in Keras applied after every ReLU activation. The values from each filter are normalized by subtracting the mean and dividing by the standard deviation, like in Equation 4.1.

$$\text{intensity normalized} = \frac{intensity - mean}{std} \tag{4.1}$$

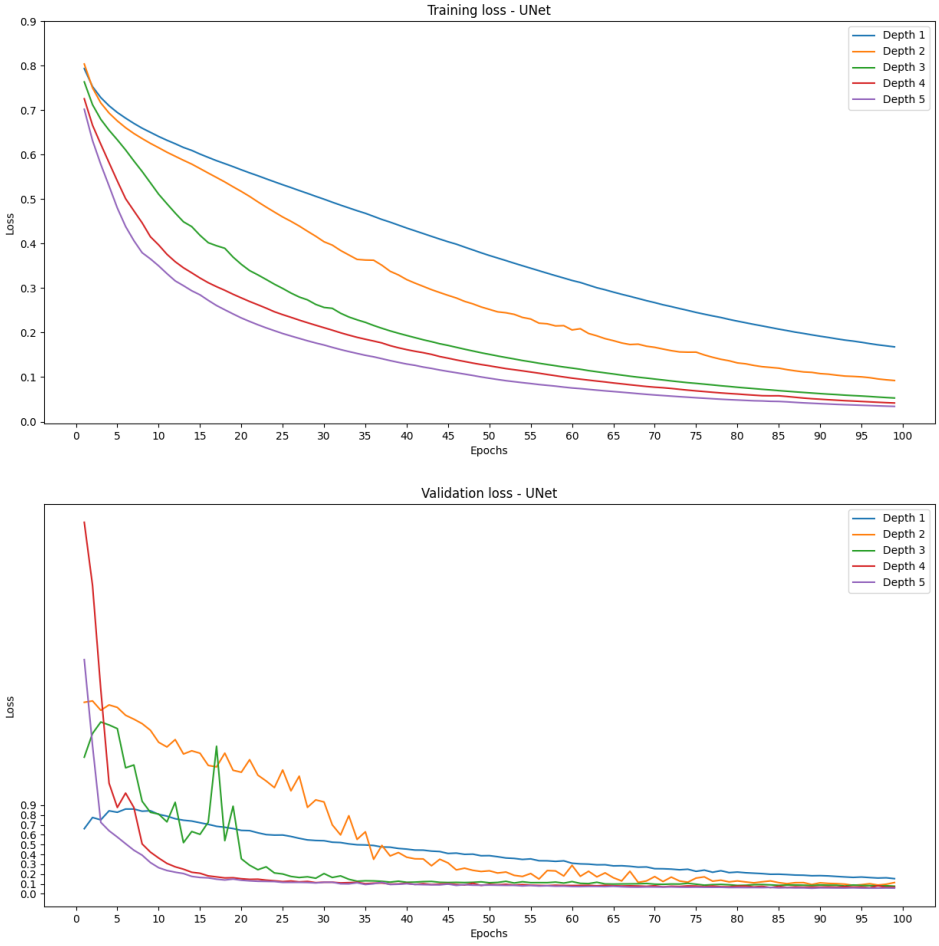


Figure 4.4: *Learning curves of depths with batch normalization.*

Again, all curves converged (Fig. 4.4). However, the loss decrease was slower. In this normalization scheme, we can see clearly that, the bigger the model, the lower the loss achieved during training. In this case, this behavior also happens in the validation on V_1 loss curve, i.e., bigger models also achieved lower V_1 loss.

On the other hand, training with batch normalization turned the learning very slower,

depth	seconds elapsed during training	min loss on training	min loss on V_1	accuracy on V_2	precision on V_2	recall on V_2
1	3531.379	0.168	0.154	0.988	0.456	0.475
2	15855.795	0.092	0.086	0.996	0.843	0.747
3	22294.151	0.053	0.077	0.997	0.864	0.857
4	26279.252	0.042	0.060	0.998	0.917	0.869
5	29429.710	0.034	0.057	0.997	0.912	0.797

Table 4.3: *Metrics of different depths with batch normalization.*

compared to normalization only on the inputs. To be precise, it was 53% slower for depth 1 and 30% for depth 5 (tabs. 4.2 and 4.3).

Another important fact to mention is that recall improved for every depth, while the precision and accuracy had no significant difference. This hints that the model was more penalized for guessing everything as "non-text", making it correctly classify more positive examples (Tab. 4.3).

4.3.3 Batch and input normalization

In this scenario, we tried to combine both normalizations previously mentioned to check if they could improve the metrics. The results can be checked in Figure 4.5 and Table 4.4.

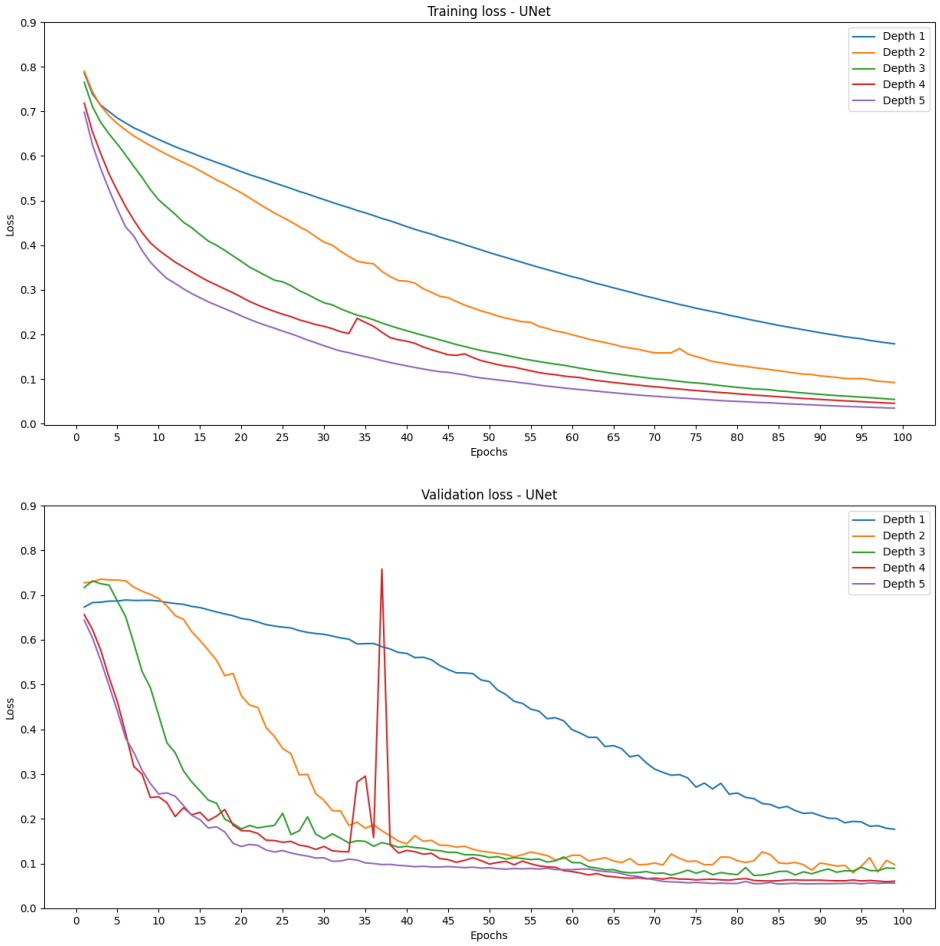


Figure 4.5: *Learning curves of depths with input and batch normalization.*

As we can see, the curves were very similar to the ones with only batch normalization, despite some oscillations. Even though precision improved for every depth, it was not a very wide increase. The other metrics did not change significantly, including the seconds elapsed to train.

4.3.4 Discussion

Based on the experiments with normalization, what we see in the training loss curves is consistent with what is expected: The larger the model capacity, the better its ability to

depth	seconds elapsed during training	min loss on training	min loss on V_1	accuracy on V_2	precision on V_2	recall on V_2
1	3345.867	0.179	0.177	0.987	0.450	0.485
2	15382.582	0.092	0.079	0.996	0.919	0.664
3	22022.136	0.054	0.073	0.997	0.926	0.836
4	26358.042	0.046	0.060	0.998	0.937	0.873
5	29570.680	0.035	0.054	0.997	0.920	0.752

Table 4.4: *Metrics of different depths with input and batch normalization.*

overfit (loss close to zero). When we analyze the validation loss, there is no clear winner. The only clear fact is that network with depth 1 is worse, and thus we should rule it out. The network with depth 2 sometimes present good validation results, but its training loss is more clearly separated from the other three. As concerning networks with depths 3, 4, and 5, they present similar validation loss. Considering the training loss curves, the validation loss curves, and the training time, depth 3 with input normalization could be an interesting choice.

4.4 Training models on different training subsets

As explained in Section 3.3, we made three subsets to use them as training sets: D_1, with the top 10 images with most CCs; D_2 with the first 10 pages from the comics; D_3 with the top 5 images with most CCs. We evaluate the model with depth 3 and input normalization when it is trained on these three subsets. This will allow us to understand how stable is training in this model when training data present different characteristics.

subset	seconds elapsed during training	min loss on training	min loss on V_1	accuracy on V_2	precision on V_2	recall on V_2
D1	15549.119	0.009	0.035	0.990	0.932	0.090
D2	15698.767	0.007	0.042	0.994	0.777	0.692
D3	8627.472	0.020	0.035	0.992	0.907	0.342

Table 4.5: *Metrics of different training subsets.*

Although training with $D3$, which has fewer examples, results in slightly worse performance, there is no significant difference between the subsets $D1$ and $D2$ (Fig. 4.6). In terms of validation loss, after 50 epochs, all the models achieve almost the same minimum loss on V_1. Furthermore, all models have achieved a 99% accuracy (Tab. 4.5). These results indicate that variations in the training data have no unexpected effects on the results.

4.5 Training models with different loss functions

The last hyperparameter we evaluate for defining the "right configuration" is the loss function. Specifically, this is much more related to improving the results concerning the task in question. We would like to segment as many text pixels as possible (that is,

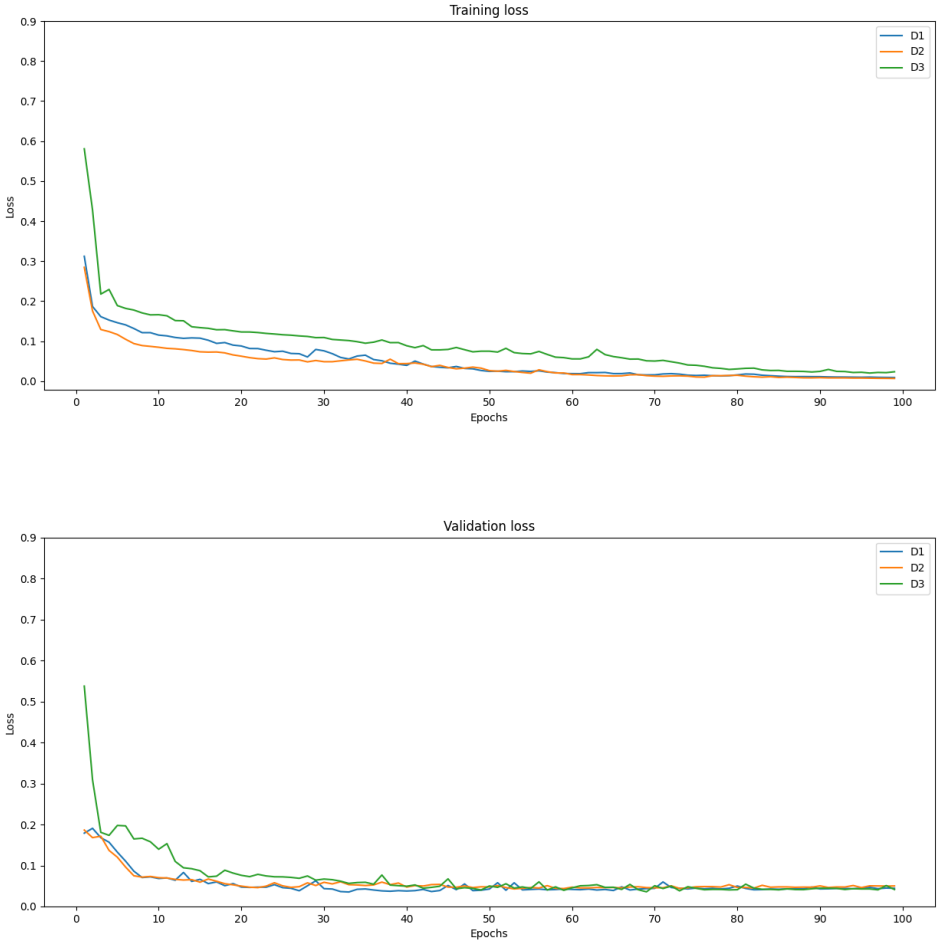


Figure 4.6: *Learning curves of different training subsets.*

maximize the true positives) and at the same time segment no pixels that are not part of the text (that is, maximize true negatives). In this work, we have experimented with the functions: Cross-Entropy (CE, the default), Weighted Cross-Entropy (WCE), Generalized Dice Loss (GDL), and Focal Loss (FL). As each metric ranges on its domain, in this Section, we chose not to display the learning curves of each loss function, as they are not directly comparable. However, some metrics on V_2, which are comparable, due to be on the same scale, are shown in Table 4.6.

For WCE, we set our weights to $w_0 = 0.45$ (for “non-text" examples) and $w_0 = 0.55$ (for “text" examples). For FL, our settings were $\gamma = 2$ and $\alpha = 0.5$.

loss function	**seconds elapsed during training**	**min loss on training**	**min loss on V_1**	**loss on V_2**	**accuracy on V_2**	**precision on V_2**	**recall on V_2**
CE	15549.119	0.009	0.035	0.028	0.990	0.932	0.090
WCE	15926.927	0.005	0.019	0.142	0.991	0.886	0.215
GDL	15729.872	0.065	0.263	0.156	0.997	0.880	0.823
FL	15591.891	0.001	0.002	0.033	0.994	0.778	0.635

Table 4.6: *Metrics of different loss functions.*

From Table 4.6, we see that the model trained with the GDL function has the best performance, with the highest accuracy and recall and very high precision. That means that, even though there is a high imbalance in the data set, the model can be generalized well on never-seen examples, and also that it does not tend to predict only the most frequent class. However, these results should be read with caution in the sense that a more exhaustive adjustment of the parameters of other cost functions may also lead to better performance. We performed no exhaustive adjustment for any of the cost functions.

We also notice that different loss functions do not affect significantly the seconds spent to train the network, as this metric is similar for all losses. The difference of time taken to train between the fastest and slowest is approximately 6 minutes.

Chapter 5

Results

Based on the experiments described in the previous chapter, we have defined a U-Net with depth 3, input normalization, and GDL loss function, with all other hyperparameters, kept with default values, as our configuration of choice. In this chapter, we present some evaluations with the chosen configuration.

5.1 Training the same model on different comics

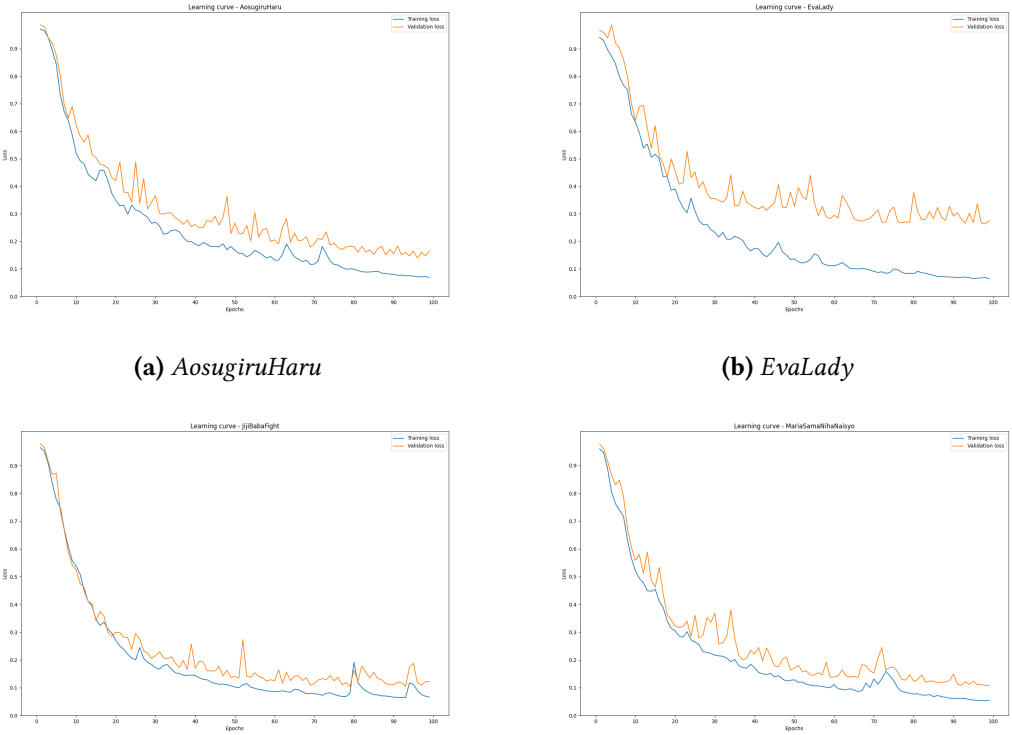
Following up, we tested the selected configuration on the four comics from Manga109. We trained the U-Net, individually, for each of the four comics. Learning curves are shown in Figure 5.1. Performance metrics are shown in Table 5.1.

comic	seconds elapsed during training	min loss on training	min loss on V_1	loss on V_2	accuracy on V_2	precision on V_2	recall on V_2
EL	15626.725	0.065	0.263	0.156	0.997	0.880	0.823
AH	15576.702	0.069	0.139	0.149	0.999	0.792	0.941
JBF	15606.121	0.064	0.104	0.091	0.998	0.890	0.943
MSNN	15660.370	0.053	0.108	0.276	0.996	0.656	0.820

Table 5.1: *Metrics of different comics.*

We see that JBF, which has smaller (in terms of area) and more CCs (in absolute number), has the best performances in terms of precision and recall on V_2. On the other hand, MSNN, which has the biggest CC's average area, has the worst performance in terms of precision and recall.

Figure 5.2 shows examples of the result for a test page from each series.



(a) *AosugiruHaru* **(b)** *EvaLady*

(c) *JijiBabaFight* **(d)** *MariaSamaNihaNaisyo*

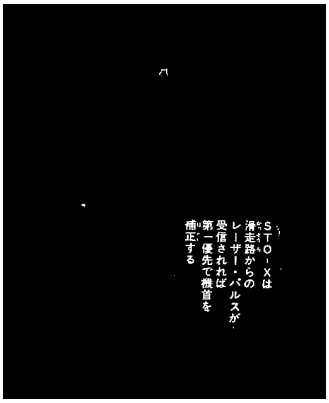
Figure 5.1: *Learning curves of different comics.*

5.2 Training models on a comic and evaluating on another

Each trained model was evaluated concerning its performance not only on the corresponding test set but also on the test set of the other three comics. Table 5.2 shows this cross-evaluation result. For instance, the top four rows of the table show the performance of the model trained on EvaLady (EL) for test sets of the four series. As it can be seen, the models best perform when trained and tested on the same comic.

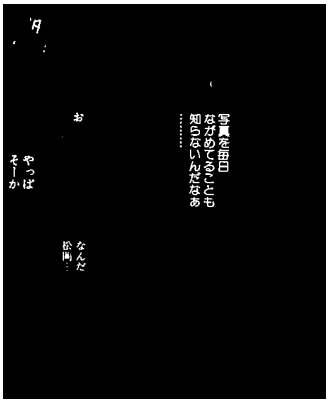
Aggregating the values per training comic, and averaging them by only considering when the models are evaluated on comics different than the used on training, we see that EL and MSNN have the highest mean accuracy; JBF has the highest mean precision; EL has the highest mean recall. It is also important to recall that JBF has the CCs with the smallest areas. Considering the ability to segment text, EL that presents the highest recall is the one that best generalizes on other volumes.

(a) *Input image* - EL©*Miyone Shi*



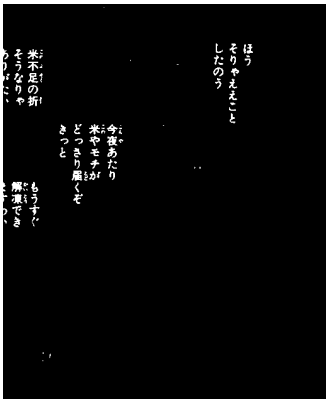
(b) *Output image - EL*

(c) *Input image* - AH©*Okuda Momoko*



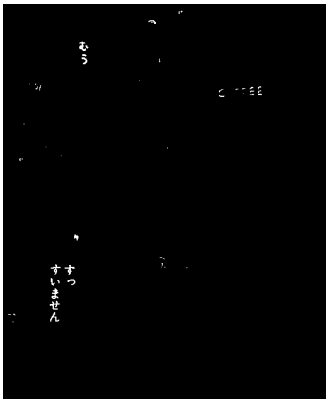
(d) *Output image - AH*

(e) *Input image* - JBF©*Nishikawa Shinji*



(f) *Output image - JBF*

(g) *Input image* - MSNN©*Konohana Akari*



(h) *Output image - MSNN*

Figure 5.2: *Example of predictions made by a network build in this study.*

Trained on	Tested on	Loss on V_2	Accuracy on V_2	Precision on V_2	Recall on V_2
EL	EL	0.089	0.996	0.857	0.813
EL	AH	0.207	0.998	0.735	0.821
EL	JBF	0.215	0.996	0.677	0.956
EL	MSNN	0.133	0.995	0.700	0.815
AH	EL	0.167	0.992	0.772	0.577
AH	AH	0.144	0.998	0.839	0.853
AH	JBF	0.157	0.997	0.814	0.883
AH	MSNN	0.152	0.994	0.738	0.660
JBF	EL	0.169	0.992	0.848	0.513
JBF	AH	0.202	0.998	0.867	0.702
JBF	JBF	0.091	0.998	0.890	0.943
JBF	MSNN	0.152	0.995	0.805	0.593
MSNN	EL	0.129	0.994	0.867	0.639
MSNN	AH	0.235	0.998	0.799	0.676
MSNN	JBF	0.228	0.996	0.745	0.810
MSNN	MSNN	0.097	0.996	0.844	0.786

Table 5.2: *Metrics of cross-evaluations.*

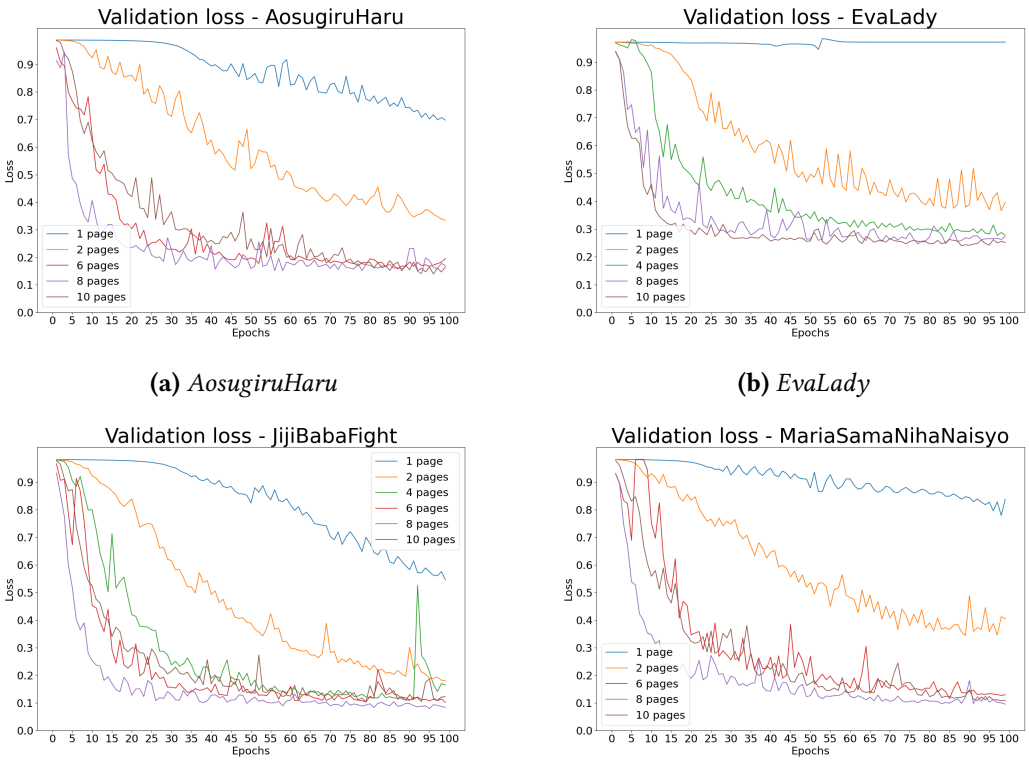
5.3 Training models with different number of examples

Lastly, we experimented with varying the number of training examples used to learn. We tested training with 1 to 10 images. In Figure 5.3, not all curves are plotted due to lack of convergence of the training. Probably this divergence happens because of a stochastic characteristic of the learning process. However, we could not isolate hypotheses to confirm that.

Considering the cases where the training converged and the loss decreased, we can see from Table 5.3 that the more examples, the higher are the metrics on V_2, as expected. Another fact observed is that sometimes a number of training examples allowed the model to converge on one comic, but not on another. This hints us the random behavior of the training. We suppressed the metrics from the experiments that did not converge.

comic	number of training examples	seconds elapsed during training	min loss on V_1	loss on V_2	accuracy on V_2	precision on V_2	recall on V_2
EL	1	2607.570	0.946	0.961	0.936	0.102	0.602
EL	2	4039.904	0.366	0.281	0.995	0.814	0.683
EL	3	5489.714	0.296	0.199	0.996	0.869	0.761
EL	4	6958.265	0.276	0.195	0.996	0.860	0.769
EL	8	12812.275	0.247	0.147	0.997	0.911	0.809
EL	10	15726.339	0.240	0.150	0.997	0.937	0.782
AH	1	2669.277	0.698	0.710	0.994	0.278	0.321
AH	2	4174.078	0.334	0.313	0.997	0.612	0.854
AH	6	9916.660	0.158	0.143	0.999	0.803	0.936
AH	7	11351.098	0.154	0.147	0.999	0.782	0.953
AH	8	12803.125	0.141	0.138	0.999	0.779	0.973
AH	9	14284.857	0.123	0.116	0.999	0.817	0.973
JBF	1	2613.289	0.546	0.580	0.990	0.449	0.424
JBF	2	4065.148	0.182	0.194	0.997	0.848	0.808
JBF	3	5524.634	0.107	0.106	0.998	0.885	0.924
JBF	4	6972.755	0.111	0.094	0.998	0.903	0.928
JBF	5	8444.949	0.091	0.069	0.999	0.926	0.947
JBF	6	9880.643	0.102	0.076	0.999	0.899	0.961
JBF	7	11358.774	0.111	0.109	0.998	0.887	0.913
JBF	8	12810.470	0.080	0.058	0.999	0.948	0.945
JBF	9	14264.388	0.074	0.060	0.999	0.924	0.964
MSNN	1	2583.022	0.779	0.872	0.970	0.080	0.346
MSNN	2	4033.580	0.344	0.501	0.992	0.421	0.660
MSNN	3	5495.866	0.168	0.329	0.995	0.618	0.751
MSNN	5	8386.843	0.128	0.424	0.993	0.459	0.779
MSNN	6	9855.683	0.116	0.264	0.997	0.730	0.750
MSNN	7	11366.824	0.096	0.211	0.997	0.787	0.797
MSNN	8	12756.922	0.096	0.230	0.997	0.723	0.829
MSNN	9	14272.052	0.094	0.231	0.997	0.734	0.810
MSNN	10	15705.678	0.094	0.198	0.997	0.801	0.806

Table 5.3: *Metrics of different numbers of training examples per each comic.*



(a) *AosugiruHaru*

(b) *EvaLady*

(c) *JijiBabaFight*

(d) *MariaSamaNihaNaisyo*

Figure 5.3: *Learning curves of different numbers of training examples per each comic.*

Chapter 6

Conclusion

In this work, we investigated the impact of varying some hyperparameters of U-Net and its training, in a context with few examples, for the typeset text segmentation problem in manga page images. We experimented with varying model depth, normalization techniques, loss functions, different training subsets, and different comics. We also conducted a cross-evaluation between models, training on a comic and assessing on another one.

We observe that a U-net of depth 3 is sufficient to efficiently generalize the problem of segmentation of text in mangas, achieving good metrics and not taking so long to train. Moreover, input normalization can help the convergence of the training, while the improvements brought by batch normalization do not compensate for the worsening in the convergence speed. Combining both normalization techniques did not bring any relevant improvements.

About training data, using the first 10 pages or the 10 pages with the most connected components (and training the model with input normalization) did not reveal any relevant difference. We also varied the number of training examples, from 1 to 10. Although not all training curves converged, probably due to a stochastic behavior, we observe that, in general, the more examples, the higher are the performance metrics, as expected. In general, using between 8 to 10 pages for training seems to lead to good results.

Regarding the loss functions, under the limited experiments done in our study, we observed that the generalized dice loss was able to best balance precision and recall in our application context where there is a high class imbalance.

When training the model on different comics, we notice a relationship between the average size of the connected components (CCs) and their precision and recall: JBF, which has smaller CCs, has the best precision and recall; whereas MSNN, which has the biggest CCs, has the worst performance in terms of the same metrics. We also tested a model trained on a comic on other comics, and in general, good performance was observed, indicating that what has been learned is at some degree transferable from one comic to another.

The experiments we did were not designed to optimize the performance, but to understand how some of the hyperparameters affect performance. In this sense, the optimizations

made were restricted to a few parameters. Taking into consideration the goals of this study, we successfully trained U-Nets for the manga text segmentation problem. We also showed that with few training data (up to 10 pages) it is possible to achieve precision and recall around 80% or more.

As future works, one of the issues is to understand why training does not converge in some cases. We suspect that it may be due to the batch size (in our case a full-page). Therefore smaller batch sizes should be tested. We also understand that the parameters of the loss functions, to account for class imbalance, are not optimally tuned yet. An interesting extension of this work is to also consider the segmentation of hand-drawn text in addition to typeset texts. Another issue that can be investigated is how the approaches based on pixel-level and bounding-box level annotations compare regarding the amount of training data and performance. We believe the U-net with depth three, with input normalization, GDL loss function, and 10 training pages, which we reached after the study done in this work, can be used as a baseline model in these future works.

References

[Achanta *et al.* 2012] Radhakrishna Achanta *et al.* "SLIC superpixels compared to state-of-the-art superpixel methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.11 (2012), pp. 2274–2282 (cit. on p. 6).

[Aizawa *et al.* 2020] Kiyoharu Aizawa *et al.* "Building a manga dataset "manga109" with annotations for multimedia applications". In: *IEEE MultiMedia* 27.2 (2020), pp. 8–18 (cit. on pp. 2, 4, 18).

[Aramaki *et al.* 2016] Y. Aramaki, Y. Matsui, T. Yamasaki, and K. Aizawa. "Text detection in manga by combining connected-component-based and region-based classifications". In: *IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 2901–2905 (cit. on pp. 15, 16).

[Chu and Yu 2018] W. Chu and C. Yu. "Text detection in manga by deep region proposal, classification, and regression". In: *IEEE Visual Communications and Image Processing (VCIP)*. 2018, pp. 1–4 (cit. on p. 15).

[Coates *et al.* 2011] Adam Coates *et al.* "Text detection and character recognition in scene images with unsupervised feature learning". In: *2011 International Conference on Document Analysis and Recognition*. IEEE. 2011, pp. 440–445 (cit. on p. 2).

[Davis and Goadrich 2006] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd International Conference on Machine Learning*. 2006, pp. 233–240 (cit. on p. 14).

[Dubray and Laubrock 2019] D. Dubray and J. Laubrock. "Deep CNN-based speech balloon detection and segmentation for comic books". In: *International Conference on Document Analysis and Recognition (ICDAR)*. 2019, pp. 1237–1243 (cit. on p. 16).

[Fujimoto *et al.* 2016] Azuma Fujimoto *et al.* "Manga109 dataset and creation of metadata". In: *Proceedings of the 1st International Workshop on Comics Analysis, Processing and Understanding*. 2016, pp. 1–5 (cit. on p. 17).

[Gonzalez, Woods, *et al.* 2002] Rafael C Gonzalez, Richard E Woods, *et al. Digital image processing*. 2002 (cit. on p. 6).

[Hirata *et al.* 2016] N. S. T. Hirata, I. S. Montagner, and R. Hirata Jr. "Comics image processing: learning to segment text". In: *Proc. of the 1st International Workshop on coMics ANalysis, Processing and Understanding*. ACM, 2016, 11:1–11:6 (cit. on pp. 16, 17).

[Ko and Cho 2020] U-Ram Ko and Hwan-Gue Cho. "SickZil-Machine: a deep learning based script text isolation system for comics translation". In: *Document Analysis Systems*. Ed. by Xiang Bai, Dimosthenis Karatzas, and Daniel Lopresti. Cham: Springer International Publishing, 2020, pp. 413–425 (cit. on pp. 16, 20).

[Lin *et al.* 2017] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal loss for dense object detection". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2980–2988 (cit. on p. 13).

[Long *et al.* 2015] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3431–3440 (cit. on p. 11).

[Matsui *et al.* 2017] Yusuke Matsui *et al.* "Sketch-based manga retrieval using Manga109 dataset". In: *Multimedia Tools and Applications* 76.20 (2017), pp. 21811–21838 (cit. on pp. 2, 4, 18).

[O'Shea and Nash 2015] Keiron O'Shea and Ryan Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015) (cit. on pp. 7, 10).

[P. Soille 2003] P. Soille. *Morphological Image Analysis. Principles and Applications*. 2nd. Berlin: Springer-Verlag, 2003 (cit. on p. 9).

[Panchapagesan *et al.* 2016] Sankaran Panchapagesan *et al.* "Multi-task learning and weighted cross-entropy for DNN-based keyword spotting". In: *Interspeech*. Vol. 9. 2016, pp. 760–764 (cit. on p. 13).

[Piriyothinkul *et al.* 2019] B. Piriyothinkul, K. Pasupa, and M. Sugimoto. "Detecting text in manga using stroke width transform". In: *International Conference on Knowledge and Smart Technology (KST)*. 2019, pp. 142–147 (cit. on p. 15).

[Ronneberger *et al.* 2015] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: convolutional networks for biomedical image segmentation". In: *International Conference on Medical Image Computing and Computer-assisted Intervention*. Springer. 2015, pp. 234–241 (cit. on p. 11).

[Sokolova *et al.* 2006] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. "Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation". In: *Australasian Joint Conference on Artificial Intelligence*. Springer. 2006, pp. 1015–1021 (cit. on p. 14).

[Sudre *et al.* 2017] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations". In: *Deep learning in medical image analysis and multimodal learning for clinical decision support*. Springer, 2017, pp. 240–248 (cit. on p. 14).

[Tarabalka *et al.* 2010] Yuliya Tarabalka, Jocelyn Chanussot, and Jon Atli Benediktsson. "Segmentation and classification of hyperspectral images using watershed transformation". In: *Pattern Recognition* 43.7 (2010), pp. 2367–2379 (cit. on p. 6).