TU WIEN Informatics

Reconstructing Production Data from Drawn Limited Animation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Felix Kugler
Matrikelnummer 01526144

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Joao Liborio Cardoso

Wien, 15. Oktober 2020

Felix Kugler	Michael Wimmer

TU WIEN Informatics

Reconstructing Production Data from Traditional Limited Animation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Felix Kugler
Registration Number 01526144

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Joao Liborio Cardoso

Vienna, 15th October, 2020

Felix Kugler	Michael Wimmer

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Felix Kugler

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. Oktober 2020

Felix Kugler

Danksagung

Danke an meine Betreuer, Joao Cardoso, der die Grundlage aller Datensätze in dieser Arbeit bereitgestellt hat, und mir jedes Mal geholfen hat, mich auf das Wesentliche zu konzentrieren, wenn ich mich in aussichtslosen Ideen verlaufen hatte, und Michael Wimmer für seine unerschütterliche Geduld.

Danke an Jarrett Martin für die Beratung und Einsichten, sowie für die Vernetzung mit den Talenten von *Tonari Animation*, ohne die diese Arbeit nicht möglich gewesen wäre. Dank auch an *Tonari Animation* und *OtakuVs* für ihre Hilfe und die Erlaubnis ihre Zeichnung für Illustrationen in dieser Arbeit zu verwenden.

Danke an Mario und Thomas, die mir viele Male dabei geholfen haben mich in der Bürokratie der Universität sowie den Straßen von Wien zurechtzufinden. Ich möchte auch Charlton und Luke danken, für die zahllosen inspirierenden Diskussionen und nützlichen Anlaufstellen.

Danke an meine Eltern, die mich über meine gesamte Bildung hinweg unterstützt haben.

Acknowledgements

Thank you to my supervisors, Joao Cardoso, who provided the basis of all the data used in this work, and also helped me get back on track whenever I got lost in ideas that did not go anywhere, and Michael Wimmer for his unwavering patience.

Thank you to Jarrett Martin for his consulting and insights, as well as providing the connections to the talents of *Tonari Animation*, without whom this work would have not been possible. Thanks to *Tonari Animation* and *OtakuVs* for helping us and giving us permission to use their art for the illustrations in this work.

Thank you to Mario and Thomas, who helped me many times with navigating the bureaucracy of the University as well as the streets of Vienna. I also want to thank Charlton and Luke for countless inspiring discussions and useful pointers to related works.

Thank you to my parents for supporting me throughout my education.

Kurzfassung

Die Produktion von traditioneller Animation geschieht in mehreren Schritten, in denen verschiedene Zwischenprodukte entstehen. Bildverarbeitungsverfahren und maschinelles Lernen könnten dafür benutzt werden, einige dieser zeitaufwendigen Schritte zu automatisieren. Lösungen des maschinellen Lernens erfordern jedoch große Mengen an Beispieldaten, welche für die Zwischenprodukte der Animation nicht erhältlich sind. Die finalen Produkte auf der anderen Seite sind in Form von Videoveröffentlichungen und online Streamingdiensten leicht verfügbar. Das Ziel dieser Arbeit ist es diese Kluft zu überbrücken, indem ein Werkzeug erstellt wird, welches durch maschinelles Lernen die Zwischenprodukte aus finalen Produktionen rekonstruiert. Diese rekonstruierten Zwischenprodukte können dann Grundlage von zukünftigen Arbeiten sein.

Einzelbilder traditioneller Animation bestehen aus Vordergrund- und Hintergrundelementen, welche in unterschiedlichen Arbeitsschritten produziert werden. Vordergrundelemente werden produziert in dem zuerst eine farbcodierte Strichzeichnung gezeichnet, welche dann ausgemalt und mit dem Hintergrund zusammengesetzt wird. In dieser Arbeit wird maschinelles Lernen dazu benutzt um zuerst Vordergrund- und Hintergrundelemente zu trennen und dann die farbcodierte Strichzeichnung der Vordergrundelemente zu reproduzieren.

Abstract

The creation of traditional animation is performed in multiple steps, creating various intermediary products. Image processing and machine learning could be used for the automation of some of these time-consuming steps to help animators and studios. However, machine-learning solutions require large amounts of example data, which are not available for the intermediary products of animation. On the other hand, final animation is more easily available through public datasets, video releases, and streaming services. This work aims to bridge this gap by creating a tool to predict intermediary products of animation from frames of the final video, using machine learning. The predicted production data can then be used for further research.

In particular, frames of traditional animation are made out of background and foreground elements, which are produced through different workflows. Foreground elements are created by first creating color-coded lineart. These are then colored and composited with the background. In this work, machine learning is used to "undo" these steps by separating a final frame into the foreground and background and recreating the lineart from the former.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Traditional animation is the process of creating movies and TV shows through a series of illustrations. Unlike 3D computer graphics animation or puppet animation, all frames of traditional animation are hand-drawn, allowing artists full freedom in how to pose and render objects and characters, giving the medium a unique look. This style of animation enjoys particular popularity in Japan. According to the Japanese Animation Association [MSR+19], the worldwide market for Japanese animation has been growing since 2009, reaching a market value of 2.2 trillion Yen (about 20 billion USD) in 2018.

Some tasks in the production of traditional animation, like designing characters, writing dialog, and storyboarding scenes, require creativity and problem-solving skills. Others are about executing repetitive tasks according to input from previous steps. The latter may include cleaning up line-art, drawing blends between line-art (inbetweens), and coloring line-art according to the colors selected by the character designer and color director. Automating the repetitive tasks with computers would free up artists to spend more time on other tasks, increasing quality or quantity. Traditional image processing methods, like edge-detectors (see Section 2.2), may be used to accomplish this. Traditional methods depend on carefully set thresholds and parameters, yet are frequently outperformed by machine-learning models, where all parameters are set automatically through a training process. Machine-learning methods, on the other hand, require datasets for training and benchmarking. This work aims to create a machine-learning model for the creation of such datasets from final frames of animation.

1.2 Problem Statement

In animation, *douga* is color-coded line-art that is created for each frame of a sequence as an intermediary result. It contains information on the shape and motion of moving

Figure 1.1: Example crop of a frame and corresponding douga. Lines that will appear in the final frame are drawn in black, edges between shades, highlights, and colors are drawn in blue, green, and red respectively. Filling is used to denote the different shades, highlights, and colors with different purple, yellow, and green tones respectively. The color-coding is not universal, as different productions use different colors. Art by Tonari Animation.

objects, and the shape and placement of highlights and shadows. Stationary parts of a scene are not drawn in douga. Figure 1.1 shows a section of a frame of animation and the corresponding douga. Which elements are drawn in a douga depends on the art-style, but they usually include lines that should be visible in the final frame and edges between different shades, highlights, and colors. See Section 2.1 for more information about animation production. Manually creating such a dataset would be time consuming and require trained professionals, as will be discussed in Section 1.3.

This work tackles the task of creating a machine-learning model to synthetically create douga from final frames of animated works. Synthetic datasets created this way can then be used to train future machine-learning models, similar to how synthetic line art datasets have already been used to train other models [IZZE17, Yon17]. The generated images can also be useful by themselves for teaching purposes or when studios need to re-create scenes for changes, but the original intermediary data is not available.

1.3 Challenges

Douga is generally not readily available and kept by studios for archiving purposes at best. It can take a trained professional 30 minutes or more to draw one such image, depending on the complexity of its content. There is no industry standard for the color-coding used in douga and different studios and even different productions within the same studio use different colors. Additionally, douga can be ambiguous or require scene understanding to understand lights and shadows in a scene correctly.

The industry-standard resolution for TV shows is 1920 by 1080 pixel, almost twice as large as the largest images used for existing machine-learning image-generation models (For example, Karras et al. [KALL18] use 1024 by 1024 pixels, most use fewer). The frequencies of elements in douga are imbalanced, with the most frequent element being about 800 times more prominent than the rarest. This is known to be challenging for existing models [LGG+18]. Lastly, douga contains small and thin objects (primarily ~2 pixels wide lines) that are frequently missed by existing models.

1.4 Goals and Contributions

The goal of this thesis is to create a machine-learning model capable of generating datasets of synthetic douga images from existing frames of animation. Our contributions are as follows:

- A proposed standardized schema to represent douga information suitable for automatic computer processing.
- A machine learning model which achieves higher prediction accuracy by dividing douga reconstruction into two segmentation problems, which can take advantage of different training data.

For the former, a set of line categories with corresponding color palette for representing the most common parts of douga is introduced. This allows consistency within the training data and the output. This palette is described in Section 5.1 and can be seen being used in Figure 1.1.

For the latter, two networks are combined into the final model. The first network deals with the separation of moving parts (called *cels*), which would be part of the douga, and stationary parts (called *backgrounds* and *books*), which would not be part of the douga. The second network deals with the generation of douga from these cels.

To train these networks, a small dataset (about two hundred images) of douga is created by professional animators and combined with a larger dataset (over five hundred images) of masks created by the student. While the professional dataset is used as the final metric of quality, the amateur dataset is used to provide additional input to the model. Different ways of combining the datasets are compared. An overview of this process can be seen in Figure 1.2.

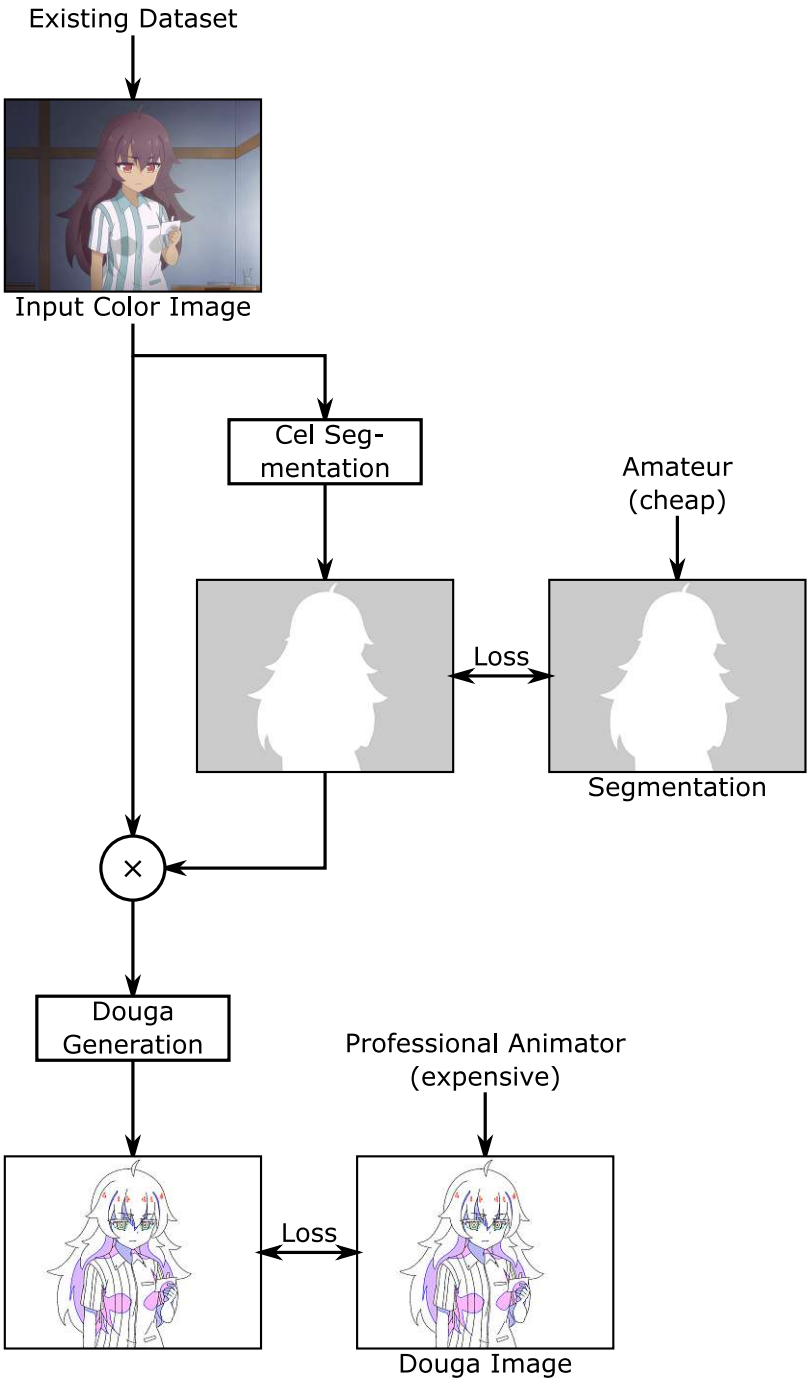


Figure 1.2: A visual overview of our work, which is divided into two sub-problems: Cel Segmentation is described in Chapter 4, Douga Generation in Chapter 5.

CHAPTER 2

Background

2.1 Animation Production

Animation is achieved by creating images and showing them in quick succession. Smooth motion requires generating dozens of images for each second. Traditional animation is the process of drawing these images by hand, either on paper or using a computer and a graphics tablet. Two different workflows have been established for this process. Western *full animation*, where most frames of video have a unique drawing, allowing for smooth motion [Whi09], and Japanese *limited animation*, where drawings are shown for multiple frames, sometimes several seconds, leading to "choppy" animation but reduced production costs [Li10, Mor15]. The latter is booming, having reached a record high in sales for the sixth consecutive year in 2018 [MSR+19].

Frames of traditional animation are split up into two parts: static elements and dynamic elements. The static elements are unchanged over the duration of a shot (typically several seconds) and may be re-used for many shots. Dynamic elements are characters and the objects they interact with, and they may change multiple times per second. Therefore, a more time-consuming workflow can be used for the generation of static elements, incorporating more details and more complex shading. On the other hand, moving elements, called *cels*, need an optimized workflow, incorporating less detailed drawings and simpler shading.

Before the advent of computers, moving elements were drawn on transparent sheets of celluloid. This allowed artists to reuse the background drawings for multiple frames. For each frame, the appropriate cel would be laid over the background and a photo would be taken. Computers allow artists to store and composite their drawings digitally, making the use of celluloid sheets obsolete. Nonetheless, the drawings of moving elements are still called cels, similar to how computer folders are still called folders because they serve the same function as their analog equivalent.

The generation of dynamic elements is split up into several steps, performed by different artists. Starting with the storyboards created by the director, the first and second key animations are created, which contain lineart for key-poses, extreme positions of movements. These are then cleaned up and extended with inbetweens, frames between the key poses to create consistent and fluid motion.

These inbetweens and cleaned up key animation, called *douga* (literally "moving image"), contain color-coded lineart. They contain both lines that should be solid in the final frame, as well as edges between differently colored areas, including edges between shades and highlights. Their purpose is to finalize the position and motion of objects, giving the coloring artist instructions on how to color the frame. Additionally to the line art, douga may also contain color-coded areas to specify which areas should be in shadow or light and which areas should have a different color than the areas around them. See Figure 1.1 for an example.

While one team is working on the moving parts of the animation, a separate team can work on the static parts. These are backgrounds and static foreground elements, called *books*, that can be laid over moving parts during compositing.

Afterward, these are colored and composited together with the backgrounds, books, and effects, like glow or blur. See Figure 2.1 for an example. An overview of the entire process is visualized in Figure 2.2. This work only focuses on the process after the second key. In this work, effects added during compositing are ignored.

2.2 Image Processing

The pipeline described is fundamentally a series of image processing steps and will be approached as such in this work. A short overview of the image processing operations that are referenced throughout this work is provided here.

Image Convolution Many local image operations can be represented as an image convolution. The discrete convolution of two images f and g is calculated as follows:

$$(f * g)_{x,y} = \sum_{i \in \mathbb{Z}} \sum_{j \in \mathbb{Z}} f_{i,j} \cdot g_{x+i,y+j}$$

f is called the *kernel* of the convolution. For most operations, the kernel is 0 for all points but a finite number around (0, 0). When $f_{x,y}$ and $g_{x,y}$ are scalars, the kernel can be represented as a matrix containing the values around the kernel center. If the kernel size is an even number, the position of the center is ambiguous and should be specified. For the purpose of this overview, this is ignored, and later sections will exclusively deal with kernels of an odd size.

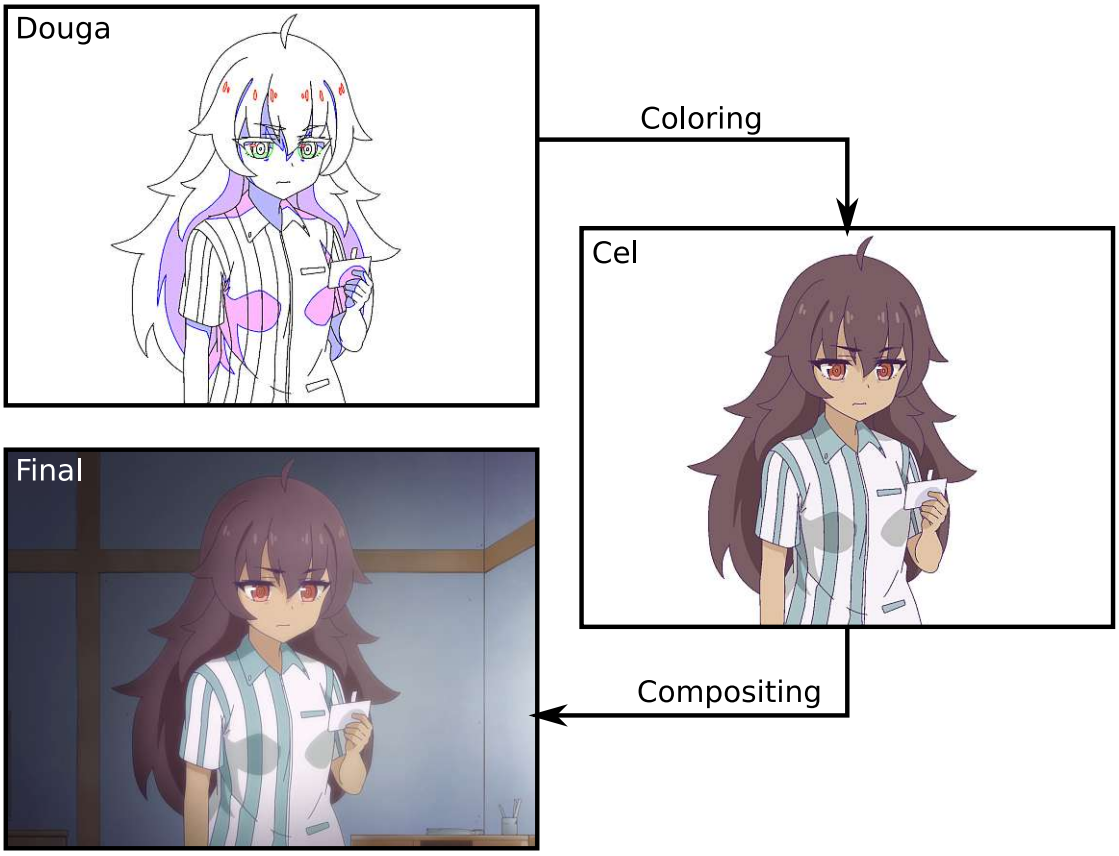


Figure 2.1: An example frame in its different stages. Art by Tonari Animation.

Image Gradient The gradient of an image is the change between neighboring pixel values. If the image is given as a function g mapping two coordinates to a real value, then the gradient can be expressed mathematically the following way:

$$\begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{dg}{dx} \\ \frac{dg}{dy} \end{bmatrix}$$

If the image is given as an array of pixel values, as is the case with raster images, the gradient can be approximated using a convolution operation:

$$g_x = \begin{bmatrix} -1 & 1 \end{bmatrix} * g$$

$$g_y = \begin{bmatrix} -1 \\ 1 \end{bmatrix} * g$$

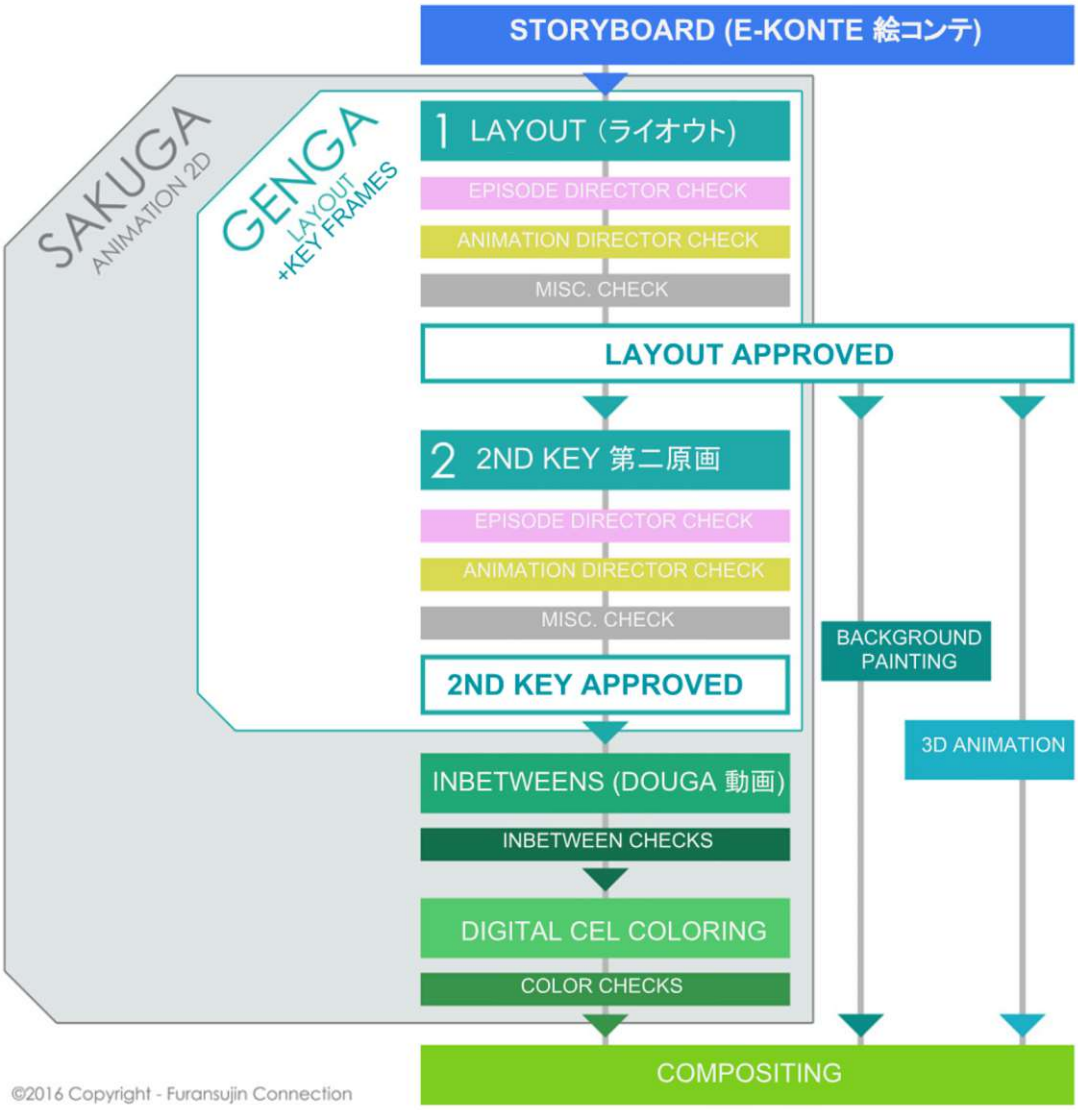


Figure 2.2: The standard 2D limited animation workflow. This work focuses on the task of 2D animation (*sakuga*) after the layout (*genga*) has been finalized. Translated from Furansujin Connection [Fur16].

These kernels have an even number of elements in one dimension. This results in the result being shifted by half a pixel compared to the input. To avoid this, a different set of kernels can be used:

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

A visualization of this operation can be seen in Figure 2.3b. Using the image gradient allows the computation of further image properties. One of which is the magnitude of an edge, which is calculated as follows:

$$\sqrt{g_x^2 + g_y^2}$$

Another is the edge angle, which is calculated as follows:

$$\tan^{-1}\left(\frac{g_y}{g_x}\right)$$

Sobel Operator Similar to the image gradient is the Sobel operator or Sobel-Feldman operator [Sob14]. Its only difference is that a smoothing filter is applied along the edge. Its kernels are as follows:

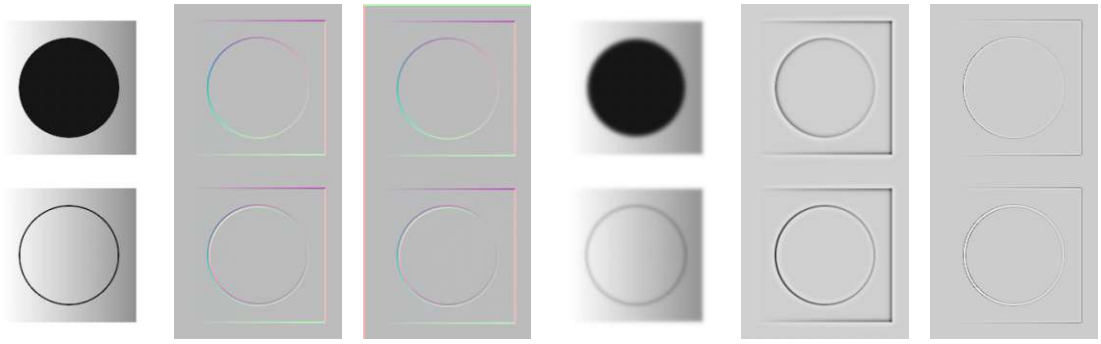
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The magnitude and angle of edges can be calculated the same way they are calculated from the image gradient. A visualization of this operation can be seen in Figure 2.3c. It is slightly softer than the image gradient but otherwise identical.

Gaussian Blur In order to remove noise and high-frequency features in an image, a common tool is to blur the image. A widely used method to blur images is to use the probability density function of the Gaussian distribution to create a convolution kernel. The Gaussian kernel is calculated as follows:

$$G_{x,y}(\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The variable σ here controls the radius of the blur. Larger values will remove more frequencies from the image. This kernel is non-zero for every x and y. It is unbounded. The kernel can be cut off at a distance, as values further away from the center contribute little to the result. The threshold depends on the application. A visualization of this operation can be seen in Figure 2.3d.



(a) Shapes and lines. (b) Image gradient. (c) Sobel operator. (d) Gaussian blur. (e) Difference of Gaussians. (f) Image laplacian.

Figure 2.3: Examples of the four edge filters.

Difference of Gaussians The difference between two Gaussian blurred versions of an image can be used as an edge-detector. It can be applied by applying the following convolution kernel:

$$\mathrm{DoG}_{x,y}(\sigma_1, \sigma_2) = G_{x,y}(\sigma_1) - G_{x,y}(\sigma_2)$$

Where σ_1 and σ_2 control the radii of the two Gaussians. A visualization of this operation can be seen in Figure 2.3e.

Image Laplacian The sum of the second derivative along both spatial axes is called the Laplacian. It can be used for edge-detection and can be expressed in terms of convolution. The discrete Laplacian kernel in 2D is defined as follows:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

A visualization of this operation can be seen in Figure 2.3f.

2.3 Convolutional Neural Networks

Machine learning solutions are able to automate processes by observing pairs of inputs and their expected outputs as examples. At first, a parameterized function $f_\Theta : X \to Y$ is constructed, where X represents the domain of inputs and Y represents the domain of outputs. Θ represents the parameters describing the mapping between inputs and outputs. In- and outputs are usually vectors. For image processing, the input could be a vector containing the red, green, and blue values for each pixel of the image. The output may be a classification, containing a probability for each class, or another image.

f is usually defined as a composition of linear and non-linear functions. Cybenko et al. [Cyb89] proves that, a function of the form

$$f_{W,V,a}(x) = W^T \cdot h(V \cdot x + a)$$

with $W \in \mathbb{R}^n$, $V \in \mathbb{R}^{l \times n}$, $a \in \mathbb{R}^l$, $x \in \mathbb{R}^n$ and h being an element-wise continuous sigmoidal function, called the *non-linearity* or *activation function*, can approximate any function arbitrarily well, independent of the choice of h given a sufficiently large l. The parameters W, V and a can be approximated numerically using gradient descent methods.

A function σ is *sigmoidal* if it is a one-dimensional, real-valued function that tends towards 0 if its parameter tends towards negative infinity, and 1 if its parameter tends towards positive infinity.

$$\lim_{x \to \infty} \sigma(x) = 1$$
$$\lim_{x \to -\infty} \sigma(x) = 0$$

Different choices of f have different advantages and disadvantages in terms of processing and memory requirements and rate of convergence. There is no general method for finding a good function, but different templates have been established empirically for different tasks.

Non-Linearities Many different functions have been proposed as non-linearities. The simplest and, according to Papers With Code [Pap20b], most widely used, is the *Rectified Linear Unit* (ReLU). It is defined as

$$\mathrm{ReLU}(x) = \max(0, x)$$

This function does not meet the requirements for Cybenko's proof, as it is not sigmoidal. However, the weighted sum of two translated ReLU functions can be:

$$\mathrm{ReLU}(x) - \mathrm{ReLU}(x - 1)$$

Convolution When dealing with image data, the in- and output vectors tend to have high dimensionality. A single frame of RGB Full HD video has over 6 million values. Performing general linear operations on such values is expensive and does not take advantage of the spatial consistency in image data. Discrete convolutions represent sparse linear transformations, suitable for images [LCBB97].

A convolution is local with regard to the spatial dimensions. Each output depends on a neighborhood of its corresponding inputs. It is also invariant with regards to spatial

translation. Translating the input causes the same translation in the output. The parameterization of a convolution operation, the *kernel*, describes the influence of each input in the neighborhood of the output.

For machine learning, the convolution is usually combined with pixel-wise matrix multiplication. For each spatial position in the kernel, a matrix is stored, that is applied to the input to create the summand for the output. The kernel is stored as a 4-dimensional array. Unlike the kernels in Section 2.2, these can not be represented as a single matrix.

The 2D convolution $y_{a,b} \in \mathbb{R}^m$, with $a \in \mathbb{Z}, b \in \mathbb{Z}$, of the input vectors $x_{a,b} \in \mathbb{R}^n$ with the kernel matrices $z_{i,j} \in \mathbb{R}^{m \times n}$ is calculated as follows:

$$y_{a,b} = \sum_{i \in \mathbb{Z}} \sum_{j \in \mathbb{Z}} z_{i,j} \cdot x_{a+i,b+j}$$

The only difference to the convolution used in image processing is that $\cdot$ denotes a vector-matrix multiplication. The output may have a different number of channels. A visualization of this operation can be seen in Figure 2.4a.

A variation of the convolution is the *strided convolution*. It combines the discrete convolution with an under-sampling of the input, reducing its size along the spatial dimensions. The strided convolution y of input x with strides $s \in \mathbb{N}_1, t \in \mathbb{N}_1$ is calculated as follows:

$$y_{a,b} = \sum_{i \in \mathbb{Z}} \sum_{j \in \mathbb{Z}} z_{i,j} \cdot x_{s \cdot a+i,t \cdot b+j}$$

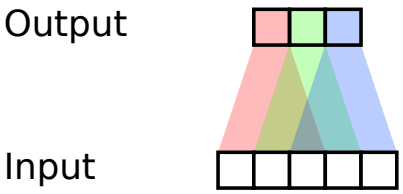
A visualization of the strided convolution can be seen in Figure 2.4b.

The *transposed convolution*, frequently misleadingly called *deconvolution*, is similar to the strided convolution. Instead of under-sampling, the input is over-sampled, increasing its size along the spatial dimensions. The transposed convolution with spatial strides s and t is calculated as follows:

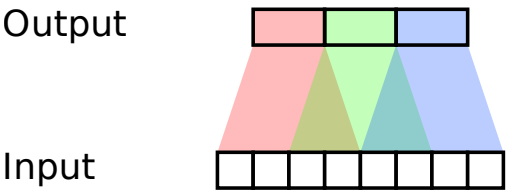
$$y_{a,b} = \sum_{i \in \mathbb{Z}} \sum_{j \in \mathbb{Z}} z_{(a \bmod s)+i \cdot s,(b \bmod t)+j \cdot t} \cdot x_{\lfloor \frac{a}{s} \rfloor + i, \lfloor \frac{b}{t} \rfloor + j}$$

If the kernel size is not divisible by the stride, the number of non-0 summands is not the same for each output pixel. This can lead to undesirable checker-board patterns in the results [ODO16]. Choosing the size of the kernel to be a multiple of the stride avoids this. A visualization of the transposed strided convolution can be seen in Figure 2.4c.

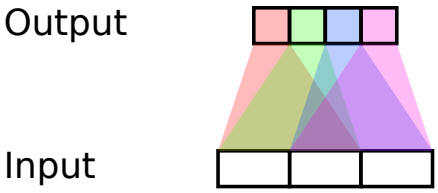
Lastly, a *dilated convolution* or *atrous convolution* [YK16] is a convolution where the input is under-sampled relative to the kernel, but not relative to the input. The dilated convolution of the input x with the kernel z and dilation rates $d \in \mathbb{N}_1$ and $e \in \mathbb{N}_1$ is calculated as follows:



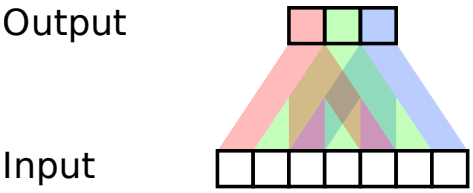
(a) Convolution with kernel size 3.



(b) Convolution with kernel size 4 and stride 2. The input is scaled down by a factor of 2.



(c) Transposed convolution with kernel size 4 and stride 2. The input is scaled up by a factor of 2.



(d) Convolution with kernel size 3 and dilation rate 2.

Figure 2.4: A visualization of the different types of convolutions.

$$y_{a,b} = \sum_{i \in \mathbb{Z}} \sum_{j \in \mathbb{Z}} z_{i,j} \cdot x_{a+i \cdot d, b+j \cdot e}$$

A visualization of the dilated convolution can be seen in Figure 2.4d.

An in-depth overview of convolutional operations in machine learning is given by Dumoulin et al. [DV16].

Pooling Like convolutions, pooling layers operate on rectangular regions of the input and apply a reduction operation on them. Like with convolutions, these regions may be overlapping and may be strided. Average pooling layers output the channel-wise average for each region. It is frequently used to downscale feature maps along spatial dimensions. Max pooling layers output the channel-wise maximum for each region. If each channel represents the presence of a feature, the max pooling layer removes information about the exact position of the feature without reducing its presence. Max pooling is used more frequently than average pooling, according to Papers with Code [Pap21a].

Depth Although Cybenko et al. [Cyb89] proves that a universal approximator is sufficient to model any function, it has been demonstrated, that a composition of several such approximators, called *deep* models, outperform individual approximators on many tasks [SLJ+15, KSH17]. Some models are hundreds to thousands of layers deep [HZRS16]. Other works suggest that increasing depth beyond a certain point (depending on task and

model) may decrease accuracy [HZRS16, SGS15]. The problem of decreasing accuracy in deep models can be mitigated by using residual learning.

Residual Learning He et al. [HZRS16] suggest replacing the learned functions in the model with functions of the following form:

$$h(x) = f(x) + x$$

Where f is a learned function. The result is, that not the mapping itself, but the *residual mapping* $f(x) = h(x) - x$ is learned. Their experiments suggest that the residual mapping tends to be easier to learn for gradient descent optimizers. This is also called adding a *skip connection* between x and $f(x)$, as information can "skip" f.

Classification Classification is the task of estimating which distribution an observed value is a result of. Given a set of distributions X_n for $n = 1, \ldots, N$ and a discrete distribution Y between 1 and N, the task of classification is to predict $y \sim Y$, given $x \sim X_y$. Equivalently, given two dependent variables $x \sim X$ and $y \sim Y$, with y discrete, the task is to estimate y given x. x is an observed value, and y is its classification. Because y is discrete, it is not possible to create a differentiable function that predicts it. As an alternative, one can create a model to predict probabilities for each possible value of y, given x. The function, that is estimated is the conditional probability:

$$f(x, y) = \mathrm{P}(Y = y | X = x)$$

When the conditional distribution of $\mathrm{P}(X = x | Y = y)$, called the *prior probability*, is known, $\mathrm{P}(Y = y | X = x)$, called the *posterior probability*, can be calculated using Bayes theorem:

$$\mathrm{P}(Y = y | X = x) = \frac{\mathrm{P}(Y = y)\mathrm{P}(X = x | Y = y)}{\mathrm{P}(X = x)}$$

Distributions Predicting the probability of each possible value of y amounts to predicting the parameters of a distribution. Depending on the domain of y, different families of distributions can be used. The learnable function is used to predict the parameters of this distribution.

The Bernoulli distribution is used when classifying between two classes, binary classification. It has one parameter p, representing the probability of a value being part of the first class. The probability of the value being part of the second class is $1 - p$.

Instead of modeling a function, that predicts probabilities directly, it is common to predict the *logits* of the probabilities. The logit of a probability p is defined as:

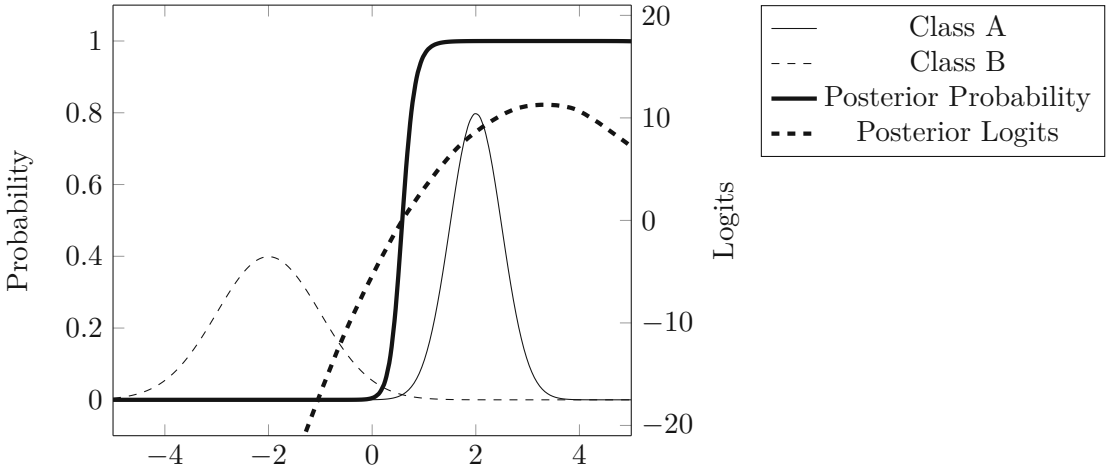


Figure 2.5: Difference between the posterior probability and the posterior logits. If the values of the classes follow normal distributions, the logits follow a quadratic function, which may be easier to learn than the probabilities.

$$\text{logit}(p) = \log(\frac{p}{1-p})$$

One of the motivating factors for this approach is the fact that, when classifying between two classes that can be described by the normal distributions with means μ_1 and μ_2 and standard deviations σ_1^2 and σ_2^2, the logits form a quadratic function. Formally, the posterior logits of the two distributions are:

$$\text{logit}(\text{P}(Y=1|X=x)) = -x^2\left(\frac{1}{2\sigma_1^2}-\frac{1}{2\sigma_2^2}\right)+x\left(\frac{\mu_1}{\sigma_1^2}-\frac{\mu_2}{\sigma_2^2}\right)-\frac{\mu_1^2}{2\sigma_1^2}+\frac{\mu_2^2}{2\sigma_2^2}-\log(\sigma_1)+\log(\sigma_2)$$

A graph of the posterior probabilities and their logits can be seen in Figure 2.5.

For classification with more than two classes, a categorical distribution is used. It has one parameter for each class, representing the probability of an element belonging to that class. As the probabilities need to add up to 1, it is common to normalize them by dividing each parameter by the sum of all parameters. When estimating logits, the probabilities can be calculated with the following expression:

$$P(Y=i) = \frac{e^{p_i}}{\sum_{j=1}^{n} e^{p_j}}$$

Where Y is the stochastic variable, $i \in \mathbb{N}$, p_i is the (unnormalized) logit of $P(Y=i)$ and n is the number of classes.

Multivariate distributions are multi-dimensional. They represent distributions of vectors, with each coordinate following a scalar-valued distribution. Images can be seen as multivariate distributions with each pixel following a distribution of colors or classes. The axes of the vectors can be independent or dependent. In the independent case, the distribution can be described by the parameters of the distribution of each axis. The probabilities of the stochastic variable $X = (X_1, \ldots, X_n)$ having the value $x = (x_1, \ldots, x_n)$ can be calculated as follow:

$$P(X = x) = \prod_{i=1}^{n} P(X_i = x_i)$$

If the axes are not independent the probability can be calculated the following way:

$$P(X = x) = \prod_{i=1}^{n} P(X_i = x_i | \bigwedge_{j=1}^{i-1} X_j = x_j)$$

It is the product of the conditional probabilities for each axis given all previous axes. The prediction of the parameters depends on all previous axis. This approach has been used for language models like GPT-3 [BMR+20] and the image generation model Pixel-RNN [OKK16].

Multivariate dependent distributions can be approximated as a mixture of independent distributions. This is called a *mixture model*. For this purpose, multiple independent distributions Y_j are predicted, as well as a scalar categorical distribution Z giving each of the multivariate distributions a probability. The probability is calculated as follows:

$$P(\hat{X} = x) = \sum_{j=1}^{m} P(Z = j) \cdot P(Y_j = x)$$

With increasing m, distributions can be approximated more accurately, but the number of parameters also increases.

Loss Function To estimate the parameterization of the function, it is necessary to define a differentiable measure for how "good" or "bad" a given parameterization is. This is done by defining a loss function l, that maps an example's true result from the dataset, and the model's prediction for the same input, to a real value. The smaller the result of this function, the better the parameterization.

To perform the least-squares approximation, the mean squared error (MSE) is used. The loss function is set to be the squared difference between the real values and the predicted values.

$$l_{\text{MSE}}(\hat{y}, y) = (\hat{y} - y)^2$$

When approximating distributions, a common loss function is the cross entropy. It represents the expected amount of data that is needed when compressing a set of symbols, following an (unknown) distribution p, using the approximate distribution q. It is minimal if $p = q$, that is if the approximate distribution is equivalent to the real distribution. Given a sample X of p, it can be estimated as follows:

$$\mathrm{CE}(X, q) = -\sum_{x \in X} \frac{1}{|X|} \log P_q(x)$$

Where $P_q(x)$ represents the probabilities or probability density of x according to the estimated distribution q. Taking the exponent of this equation and negating it leads to the following expression:

$$-e^{|X| \cdot \mathrm{CE}(p,q)} = \prod_{x \in X} P_q(x)$$

This expression is identical to the expression that is maximized for maximum likelihood estimation. Therefore, minimizing the cross-entropy is equivalent to maximizing the likelihood.

This leads to the following loss function:

$$l_{\mathrm{CE}}(\hat{Y}, y) = -\log P_{\hat{Y}}(y)$$

Where $P_{\hat{Y}}(y)$ is the predicted probability of y according to the model.

Optimizer f_Θ and l are chosen to be differentiable, allowing the use of gradient descent to approximate the parameters of the function. In its simplest form, the gradient of l over Θ is calculated and Θ is subtracted by the result, multiplied by a learning rate η.

$$\Theta_{t+1} = \Theta_t - \eta \nabla_\Theta \frac{1}{|X|} \sum_{x \in X} l(f_{\Theta_t}(x))$$

Instead of evaluating l for all examples, it is common to evaluate it over a random sample of examples. This method is called *stochastic gradient descent*. Several other variations exist. According to Papers with Code [Pap20a], the most widely used variation is Adam [KB17]. Instead of subtracting the gradient from the parameters directly, Adam keeps an exponential running average of the gradient and uses that to update the parameters. Additionally, they divide each dimension of the update by the square root of an exponential running average of their squares, to normalize updates. Dimensions of the parameter that tend to get small updates, have their updates scaled up, whereas dimensions that tend to get large updates have their updates scaled down. Its update rule is defined as follows:

$$m_0 = 0$$

$$v_0 = 0$$

$$g_t = \nabla_\Theta \sum_{x \in \hat{X}_t} l(f_{\Theta_t}(x))$$

$$m_{t+1} = \beta_1 \cdot m_t + (1 - \beta_1) \cdot g_t$$

$$v_{t+1} = \beta_2 \cdot v_t + (1 - \beta_2) \cdot g_t^2$$

$$\Theta_{t+1} = \Theta_t - \eta \cdot \frac{m_t}{1 - \beta_1^{t+1}} \Bigg/ \left(\sqrt{\frac{v_t}{1 - \beta_2^{t+1}}} + \epsilon \right)$$

With β_1 and β_2 being the factors for the exponential running averages, ϵ being a small value to avoid divisions by zero, and $\hat{X}_t$ being the random sample of the examples at step t.

Lipschitz Continuity To analyze the convergence of optimizers it is necessary to measure properties about the function that is to be minimized. One such property is the Lipschitz continuity and its corresponding Lipschitz constant. A real-valued function f is said to be Lipschitz continuous if the following equation is true for all pairs of points x_1, x_2 for some constant K:

$$|f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2|$$

The Lipschitz constant is the smallest value for K for which the equation holds. Every function with a bounded first derivative is Lipschitz continuous. Linear functions, all commonly used activation functions and compositions thereof are Lipschitz continuous.

Given a function f, with a Lipschitz continuous gradient with Lipschitz constant K, gradient descent with a learning rate smaller or equal to $\frac{1}{2K}$ converges to a local minimum. [Arm66]

Batching Adam and stochastic gradient descent only operate on samples of the entire training set at each step. The subset of the training set is called a *batch*. This reduces the memory and processing requirements per step. It allows choosing a trade-off between performing more, less accurate updates and fewer, more accurate updates. If the loss is the average over the samples in a batch, the resulting gradient is an unbiased estimate for the real gradient. As the sum is a linear operation, this is the case for the cross-entropy loss and mean squared error. The Adam optimizer, however, is biased, because the division of two unbiased estimates is generally not unbiased.

Parameter Initialization Gradient descent is a method for improving a given parameterization. But before it can be applied, it is necessary to create a starting point, initial values for all the parameters of the model. Initializing the weights with zero would lead to most of the gradients of the model being zero, and convergence is impossible. Choosing the weights too small causes activations, and therefore gradients, to tend towards zero with increasing depth. Choosing the weights too large causes activations and gradients to grow exponentially and to "explode" with increasing depth. Glorot et al. [GB10] notice that the speed of convergence hinges on the distribution of activations throughout the model. They suggest a method for initializing the parameters such, that the output of the linear operations in the model approximately has the same variance as its input. This method is called *Xavier initialization* or *Glorot initialization*.

Given a linear operation on the value $x \in \mathbb{R}^n$, parameterized by the matrix $W \in \mathbb{R}^{m \times n}$ as follows:

$$y = W \cdot x$$

They prove that the expected variance of y is equal to the expected variance of x if the following condition holds:

$$n \cdot Var(W) = 1$$

Further, it might be desirable to have an equal variance of the gradient for each layer. The variance of the gradients is stable for the following condition:

$$m \cdot Var(W) = 1$$

These two conditions cannot both be true if $m \neq n$. They suggest the following compromise:

$$Var(W) = \frac{2}{n+m}$$

When initializing W, values are sampled from a uniform or normal distribution with the desired variance.

Glorot et al. [GB10] do not take into account the properties of the activation functions. Using the most common activation function, ReLU, Xavier initialization has been found to perform poorly for networks with many (e.g. 30) layers [HZRS15, Kum17]. He et al. [HZRS15] suggests the following condition for the initialization of the weights:

$$\frac{1}{2} n \cdot Var(W) = 1$$

Batch Normalization Another solution to overcoming vanishing or exploding gradients is *batch normalization*. Glorot et al. [GB10] realize that it is beneficial to keep the variance of the activations stable and that weights can be chosen in a way to obtain that property. Ioffe et al. [IS15] provide a different solution to this problem. At each step, they calculate the mean and the variance across spatial dimensions of the activations within a batch and use these to normalize them to mean zero and variance one. Although created with the intent to normalize the gradients of the parameters, Santurkar et al. [STIM18] demonstrate that this normalization of the gradients is not necessarily desirable, nor is batch normalization effective at it. They link its effectiveness to a smoothing (reduction of the Lipschitz constant) of the loss function.

Generalization and Regularization The goal of approximating a function to perform the desired transformation on example data is to then be able to use this function on new data. A model that performs well on examples that were not used during training is said to *generalize* well. *Overfitting* is the problem of having a model that performs well on the training data, memorizing mistakes and outliers therein, while not learning general features of the data, that can be used on values outside the training data. Generally speaking, there could be any number of functions, that give the right answer for values of the training set, and arbitrary answers for values outside the training set. One method to evaluate this property is to split the dataset into two sets: The training set and the test set. The training set is used for estimating the parameters of the model. The test set is not used for parameter estimation or model design, but to estimate the accuracy of the model on potential future data points.

Instead of finding a single function that approximates the dataset, one can try to find the set of functions, that do. One approach to achieve this is to train many models separately and average their results [SHK+14]. This, however, can be prohibitively expensive for large models. Srivastava et al. [SHK+14] propose an approximation to this task. Instead of training several models separately, a single model is trained, with activations being set to zero with a certain chance. They call this method *dropout*. This can be seen as training 2^n different models with shared weights, where n is the number of activations or hidden units. For inference, instead of evaluating all 2^n models, they instead multiply each activation by the probability of it being set to zero during training.

Inference The ground truth in a classification task is discrete. Each element belongs to exactly one class. To create a differentiable classifier, it is common to not work with the classes themselves but with categorical distributions of classes. Instead of estimating the one true class, a probability for each class is estimated. To use the model, its output needs to be turned into discrete values. The most common way of doing this is to return the mode of the predicted categorical distribution. Each element is classified as its most likely class.

Evaluation After a model has been trained it can be evaluated to compare it with other models. Several measures have been suggested for different tasks. *Accuracy* is

the percentage of elements that are correctly classified by a classifier. When used in segmentation tasks, it is the number of pixels, that are correctly classified. Given the number of correctly classified elements TP and the total number of elements N, the accuracy is calculated as follows.

$$\text{accuracy} = \frac{\text{TP}}{\text{N}}$$

The *intersection over union (IoU)* or *Jaccard index* [FI+18] is a measure for the similarity between two sets and can also be used to calculate the similarity between a predicted segmentation and the true segmentation. Given the number of true positives TP, false positives FP and false negatives FN it is calculated as follows.

$$\text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}}$$

The above measures are useful when tasked with modeling a function through pairs of in- and outputs. For generative models, where the task is to predict a distribution from a sample of it, other metrics are required. Image generation tasks aim to produce images based on examples, in a way that they look similar to humans. Therefore, the quality metric should be more sensitive towards features that humans focus on, and indifferent towards features that humans do not perceive. The *Inceptions score (IS)* [SGZ+16] and *Fréchet Inception Distance (FID)* [HRU+17] use the conditional label distribution of an Inception v3 model [SVI+16], pre-trained on ImageNet [DDS+09], as an approximation to human vision. As humans are adept at differentiating the different classes in ImageNet, a model trained on doing so might, at least, focus on similar features. Studies with human participants indicate that both metrics correlate well with the similarity that humans assign to samples of distributions [SGZ+16, HRU+17].

CHAPTER 3

Related Work

The problem of reconstructing production data requires solving two sub-problems. As the final frames are compositions of images, created through different work-flows, segmentation can be used to split them up. Afterwards, edge-detection and image-to-image translation methods can be used to recreate the douga from the dynamic parts of the frame.

3.1 Synthetic Datasets

Existing machine learning solutions rely on the availability of training data in large quantities. For example, the ImageNet [DDS+09] dataset of photos of people, animals, and objects, contains over 14 million annotated images. Creating these datasets requires manual acquisition and annotation. For segmentation datasets, this requires between 20 minutes [BFC09] to 90 minutes [RVRK16] per image.

To speed up the process of collecting training examples, computer graphics can be used to create synthetic datasets automatically. This allows the generation of a virtually infinite number of examples. This method has been used to perform semantic segmentation and bounding box detection for urban street scenes [RVRK16, RSM+16, WU18], kitchen scenes [SBLL20], toys and dishware [HLWK18], and indoor scenes [MHLD17].

Richter et al. [RVRK16] use the computer game Grand Theft Auto V, which produces photo-realistic renderings of a fictional city, to automatically create a dataset and semantic segmentations for street scenes. Ros et al. [RSM+16] and Wrenninge et al. [WU18] create and render their own virtual worlds to generate a synthetic dataset.

Wrenninge et al. [WU18] evaluate the effect of training on synthetic datasets for image segmentation and object boundary detection in street scenes. They train a model on different synthetic datasets and calculate the model's accuracy on real data. They find that training on synthetic data alone tends to harm accuracy on real data, measuring a decrease in mean IoU from 68% to 45% in their best configuration. They also find that

realism of the synthetic data is important, measuring a variation in mean IoU between 45% and 22% depending on which synthetic dataset is being used. Lastly, they find that training on both real data and synthetic data improves mean IoU compared to training only on real data, even when synthetic-only training suggests that the synthetic dataset represents the ground truth poorly.

McCormac et al. [MHLD17] find that pre-training on a synthetic dataset of indoor scenes and then fine-tuning on real data improves mean IoU from 29% to 48% on the NYUv2 dataset [NSF12] and from 37% to 50% on the SUN RGB-D dataset [SLX15].

Although they are exclusively targeting photos of real scenes, these works suggest that synthetic examples, combined with real examples, improve the accuracy of classification models for various domains.

Li et al. [LLW17] use augmentation on a dataset of a smaller domain to train a model on a wider domain. Their goal is to detect lines in textured/hatched grayscale hand-drawn illustrations. For training they take a dataset of texture-less hand-drawn illustrations and assign a synthetic texture to each connected component in the image. The model is then trained to remove these textures, restoring the original lines.

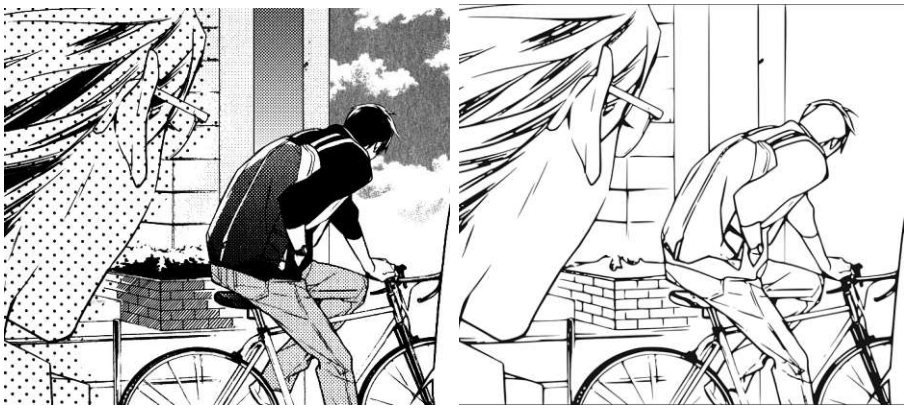
3.2 Segmentation

Image segmentation is the task of assigning a label to each pixel of an image. Existing research focuses on identifying object classes. There are datasets with labeled photos of everyday objects [EVGW+11] and street scenes [GLU12, COR+16] that are being used as benchmarks for segmentation models. The student is not aware of the existence of datasets or models for the segmentation of traditional animation, nor illustrations in general.

Both the task of identifying dynamic elements in an image and the task of identifying lines and edges can be interpreted as segmentation tasks. In the former case, each pixel would be classified as being produced by either the workflow for dynamic elements or the workflow for static elements. In the latter case, each pixel would be classified as either belonging to one of the line and edge classes or as not belonging to either.

Models Originally designed for medical applications, the U-Net architecture [RFB15] has been successfully used for image segmentation [IS18]. It consists of a sequence of strided convolutions, followed by a sequence of strided transposed convolutions. Skip connections are used between outputs of the same size. This structure allows the incorporation of both local and global information into the segmentation of a pixel. According to Papers with Code [Pap21b] the U-Net architecture is the most popular model for image-to-image translation and segmentation.

Long et al. [LSD15] take successful image classification architectures and investigate their use as segmentation models. Image classification models reduce a spatial image into a nonspatial output through strided convolutions and pooling layers, usually removing



(a) An illustration from a Manga.

(b) The line image generated from the input using Manga Line Extraction.

Figure 3.1: Example from the work by Li et al. [LLW17].

more and more spatial information every few layers. Long et al. counter this reduction by using bilinear upscaling on the outputs of the layers and adding skip connections between downscaling layers and the final output.

An alternative approach to translating classification models to segmentation models is presented by Chen et al. [CPSA17]. Instead of downscaling the image through pooling or strided convolutions, they increase the dilation rate of all layers following a pooling or strided convolution in the original model. This is equivalent to evaluating a classification network for each neighborhood in the input image. A visualization of how the strided convolutions in ResNet correspond to dilated convolutions in DeepLab can be seen in Figure 3.2.

Li et al. [LLW17] use a model similar to ResNet to extract lines from *Manga*, Japanese comics. These consist of hand-drawn grayscale illustrations with textures or hatching. The model performs a binary classification, identifying whether a pixel belonging to a line. An example of this work can be seen in Figure 3.1.

Peng et al. [PZY+17] find that incorporating a large neighborhood of pixels (25 by 25 pixels) into the classification of a pixel increases accuracy. As 2D convolutions with large kernels require both a large amount of time and memory, they split up 2D convolutions into pairs of 1D convolutions, along the horizontal and vertical axis. An example of a segmentation generated with this model can be seen in Figure 3.3.

Pre-Training ImageNet [DDS+09] is a dataset of photos and labels representing which objects are present in the image, without spatial information. This dataset represents the de facto standard benchmark for image classification tasks and has been well explored. Although it does not carry any spatial information, it can be used for training segmentation models. Before training a model on segmentation, a sub-graph of it is trained on ImageNet

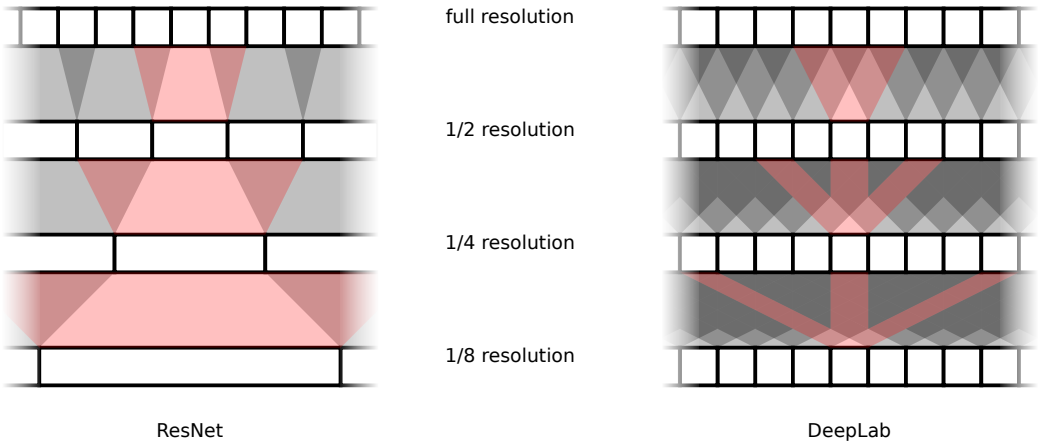
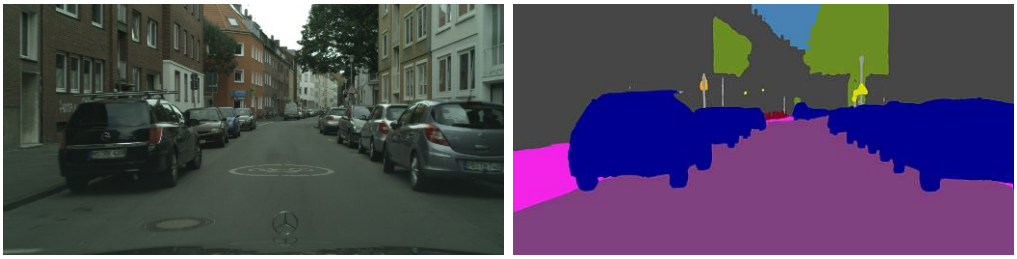


Figure 3.2: The scales of the ResNet [HZRS16] and DeepLab [CPSA17] models. While the former scales down the feature maps by half after each block, the latter increases the dilation rate by two after each block.



(a) An image of the Cityscapes dataset [COR+16] of street scenes.

(b) The semantic segmentation of the same image. The different classes of objects (cars, pavement, sky, buildings, etc.) are identified by the model and visualized with separate colors.

Figure 3.3: Example from the work by Peng et al. [PZY+17].

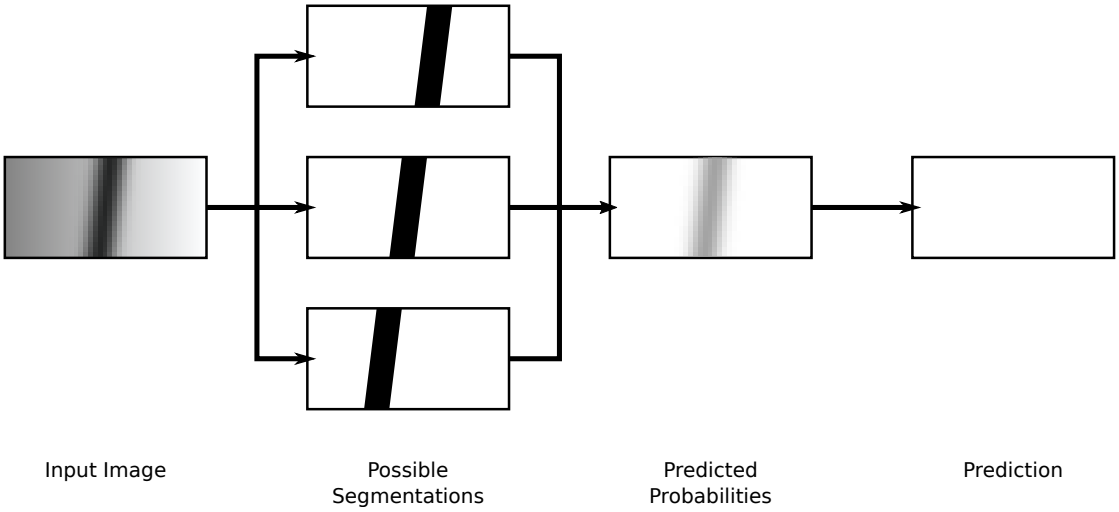


Figure 3.4: When the input is ambiguous, due to low contrast, blur or compression artifacts, giving each pixel a low chance of containing the class, minimizes the loss. As a result, no pixel reaches the threshold to be classified as the object, and the object is missing in the segmentation.

classification, allowing it to learn useful priors from a large dataset. This has shown to improve accuracy on some segmentation tasks [PZY+17] but hurt accuracy on others [ZGL+20].

ImageNet exclusively contains photos. It is unknown how well it performs on illustrations. The Danbooru dataset [Bra19] is a similar dataset for illustrations, containing information on the presence of object classes and user tags.

Loss Function The training set can be thought of as a sample of the distribution of pairs of final images and corresponding douga. The task is to model the conditional distribution of douga images given a final image. Ideally, there would be a single douga corresponding to a given final image, but due to inaccuracies in the model and ambiguous inputs, this is generally not the case.

Cross entropy minimization allows the prediction of the marginal distributions for each pixel. However, the marginal distributions do not represent the multivariate distribution of possible segmentations. As an example, suppose the input suggests there to be a thin vertical line-shaped object that is either at position a, b or c with equal probability. The prediction that would minimize the cross entropy loss would assign a probability of 33% to each pixel of the three possible lines. The probability of the object being present is below 50% for each pixel and, therefore, the model assumes that there is no object. See Figure 3.4 for a visualization.

For the task of representing edges between shapes, the existence of a line is more important than its exact position. Therefore, classification is not a suitable framework of

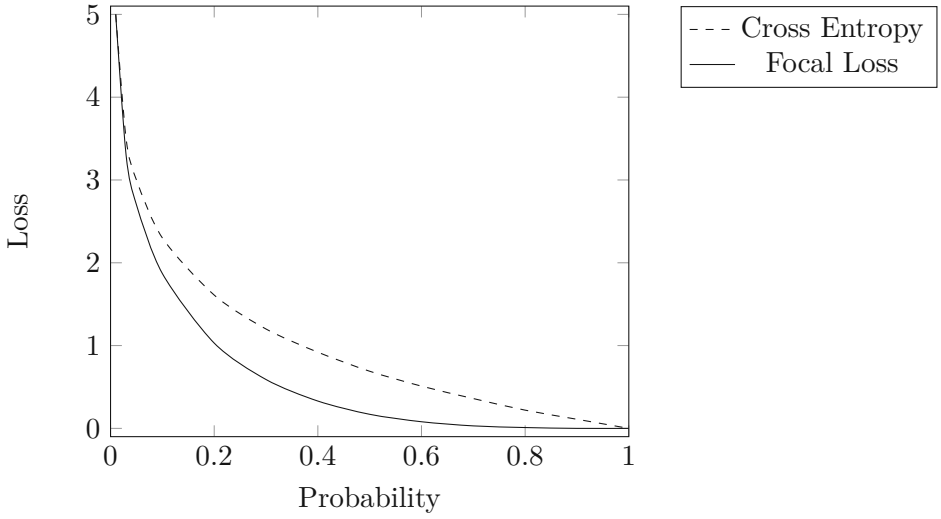


Figure 3.5: Cross entropy and focal loss with $\gamma = 2$ with regards to the probability given to the ground truth. Examples, for which the model is confident, have a lower loss with focal loss, than when using the cross entropy.

the problem. Instead, the problem can be framed as the task of *sampling* the distribution of possible segmentations. See Section 7 for an exploration of this solution..

As described in Section 2.3, cross entropy minimization leads to a maximum likelihood estimator, assuming infinite model capacity and training examples. Other training objectives have been suggested, to trade accuracy in some regions for accuracy in others.

Lin et al. [LGG+18] modify the cross entropy by a factor such that the weight of "easy" examples is less than that of "hard" examples. They call this *focal loss* If p_t is the probability the model gives to the true class, then the focal loss can be calculated as follows:

$$\mathrm{FL}(p_t) = (1 - p_t)^{\gamma} \cdot -\log(p_t)$$

With γ controlling how much to reduce weighting of accurate examples. At $\gamma = 0$, focal loss is equivalent to cross entropy. The authors suggest a value of $\gamma = 2$. A plot of both the cross entropy and the focal loss can be seen in Figure 3.5.

For an in-depth comparison of different loss functions see Shruti [Jad20].

Perturbation Szegedy et al. [SZS+14] notice that the prediction of classifiers is sensitive to changes barely visible to the human eye. They use the gradient of the model to calculate a change within a fixed range, that maximizes the prediction error. That is, given an example x, that the classifier can correctly classify, they calculate an example x' that is indistinguishable from x for a human, but causes the classifier to misclassify it.

One way to calculate this x' is to use gradient descent. If the classifier is locally linear, x' can be efficiently calculated as $x + \epsilon \cdot \text{sign}(\nabla_x l(\Theta, x, y))$, where ϵ is the amount of change applied to the example and $l(\Theta, x, y)$ is the loss for example x with ground truth y and learned weights Θ.

Goodfellow et al. [GSS15] demonstrate how perturbation can be used to improve generalization. To accomplish this, they apply perturbation to the training examples during training. This is a form of adversarial training.

3.3 Edge-Detection

Traditional edge-detection algorithms, including the Canny edge filter [Can86] and high-pass filters, were previously used for the task of creating line art from colored illustrations [IZZE17, Yon17]. However, these depend on a threshold parameter and tend to either not detect all edges or detect edges where there are none. They are also not able to distinguish between the different types of edges.

In the machine learning context, edge-detection can be either interpreted as an image segmentation task, where pixels need to be classified as either being part of an edge or not, or as an image to image translation process, where a model estimates a new color image from an existing one. While conceptually similar, the two interpretations lead to different methods.

Pang et al. [PLS+18] and Song et al. [SPS+18] solve inverse sketching and try to replicate the way humans create sketches. However, their goal is to replicate the behavior of an amateur, whereas this work focuses on line art generated by professional illustrators. Additionally, both works focus on creating sketches from photos.

Zhang [Zha17] uses machine learning to generate edge images for the use with automatic coloring solutions. However, these do not reflect the douga used during the production of traditional animation. It does generate colored line art, but they do not explain the meaning of the colors, and they do not appear to be useful for animators. Further, it is limited to images with a resolution of 512 by 512 pixel, whereas industry standard resolution is 1080 by 1920 pixel. Additionally, they train the model on static illustrations, which tend to have more detail and more complicated shading than illustrations made for animation. An example of this model can be seen in Figure 3.6.

3.4 Image Generation

Images can be represented as points in a space with a dimension for each pixel and channel. The generation of images is then the task of sampling a distribution in that space, learned from a sample of the real distribution. Several methods for this exist. Generative adversarial networks (GANs) [GPAM+14] and variational auto encoders (VAEs) [KW13] do not model the probability density function of the distribution explicitly but implicitly by learning a mapping (the generator or decoder respectively) from a known distribution

(a) A frame of a TV series.

(b) Line art predicted by SketchKeras.

(c) Crop of the real douga the frame was composed of.

(d) Same crop in the SketchKeras prediction. The resolution is less than a third.

Figure 3.6: Limitations of the line art generated by SketchKeras [Zha17]. Original web series frame and corresponding douga by Tonari Animation.

(most commonly a multivariate normal distribution) to the approximation of the target distribution. Recurrent models (RNNs) [CRC+20] and flow-based generative models [KD18] give an explicit solution to the probability density function of the underlying approximated distribution.

Additionally to, or instead of the known distribution, these models may also take additional inputs as conditional variables. This allows generating images of specific classes (e.g. cats or dogs) [BDS18] or transforming images from one class to another (e.g. horses to zebras) [IZZE17]. The task of generating plausible production data from final frames is a kind of image-to-image translation where the goal is to sample the distribution of potential intermediate images, given a final image.

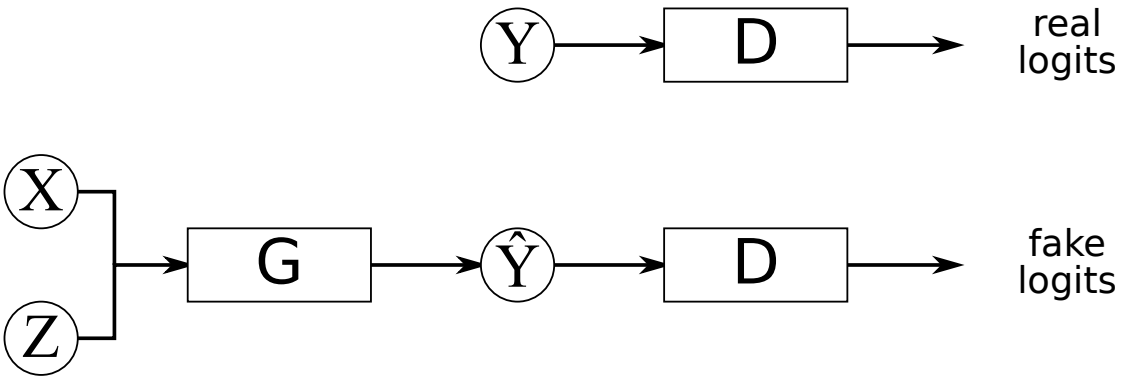


Figure 3.7: Overview of the Generative Adversarial Network. G and D are learned functions and represent the generator and the discriminator respectively. X represents the samples of the input data, Y the samples of the corresponding target data. Z is a known distribution, usually a normal distribution. $\hat{Y}$ represents the generated samples.

Generative Adversarial Networks Goodfellow et al. [GPAM+14] reformulate the problem of approximating a distribution as a two-player minimax game. The method jointly optimizes two models. One model, the generator, maps samples from a known distribution (e.g. a normal distribution) to the target distribution and the second model, the discriminator, differentiates between fake images created by the generator and the real training samples. By training a differentiable classifier on classifying between real and fake images, this classifier can be used to calculate a gradient from fake to real images, that the generator can be trained with. A visualization of the network structure is given in Figure 3.7.

In its minimax formulation, the loss for the generator is the negative loss of the discriminator. It can be written as follows:

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}(\log \mathrm{D}_{\theta_D}(y)) + \mathbb{E}(\log(1 - \mathrm{D}_{\theta_D}(\mathrm{G}_{\theta_G}(z))))$$

Where G and D represent the generator and the discriminator respectively, and $y \sim Y$ and $z \sim Z$. This formulation tends to provide insufficient gradient to G when D is accurate. An alternative proposed by the original author is to use a different loss function when updating the Generator. Instead of minimizing $\log(1 - \mathrm{D}(\mathrm{G}(z)))$, they minimize $-\log(\mathrm{D}(\mathrm{G}(z)))$. Fedus et al. [FRL+17] call the latter non-saturating, as the loss does not saturate for both generator and discriminator. For examples, that are easy to discriminate, the discriminator loss tends towards zero while growing linearly for the generator, and vice-versa. A graph of the loss of G for a saturating GAN and its non-saturating counter can be seen in Figure 3.8.

The training objective for the discriminator is to reduce the cross-entropy between its prediction and whether an input is real or fake. The training objective for the generator

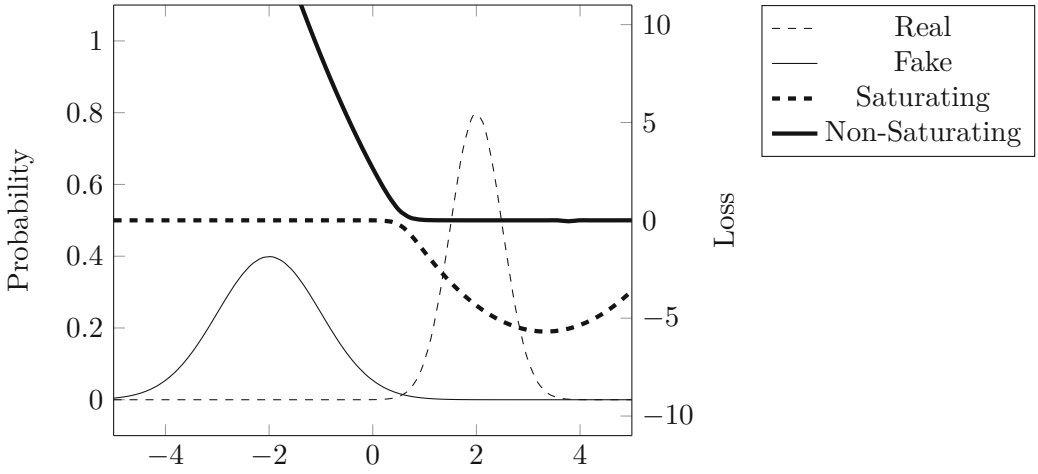


Figure 3.8: The generator loss for a saturating GAN and a non-saturating GAN, given two distributions for real and fake examples. The saturating GAN uses the negative D loss for the generator. This loss saturates at 0 for poor images. The alternative, the non-saturating GAN, uses a separate loss for G, that provides useful gradient for poor examples.

is to reduce the cross-entropy between the discriminator prediction and an image being real. This approach tends to be unstable as either the generator or the discriminator or both can get stuck in local minima and stop to provide useful gradients to each other.

Several workarounds have been proposed to address the instability of GAN training. Arjovsky et al. [ACB17] examine alternate loss functions. The original authors of the GAN approach have suggested the cross entropy as a measure for the difference between the target distribution and the generated distribution. Arjosky et al. instead use the *earth mover's distance* or *Wasserstein metric*. The Wasserstein distance between two distributions Y and $\hat{Y}$ is defined as follows:

$$W(Y, \hat{Y}) = \sup_{||f||_L \leq 1} \mathop{\mathbb{E}}_{y \sim Y}(f(y)) - \mathop{\mathbb{E}}_{\hat{y} \sim \hat{Y}}(f(\hat{y}))$$

Where f is a Lipschitz continuous functions with Lipschitz constant less or equal to 1. For training the generator, a multiplicative constant of f can be ignored. It is sufficient to compute $K \cdot W(Y, \hat{Y})$. Therefore it is enough to solve the following expression:

$$\max_{\theta} \mathop{\mathbb{E}}_{y \sim Y}(f_\theta(y)) - \mathop{\mathbb{E}}_{\hat{y} \sim \hat{Y}}(f_\theta(\hat{y}))$$

This can be approximated using gradient descent methods. f replaces the discriminator of the original GAN formulation. To constrain f to Lipschitz continuous functions, the absolute values of its weights are constrained to be less or equal some constant value.

To avoid the constraint on the weights of f, Gulrajani et al. [GAA+17] propose placing an additional loss on the Euclidean length of the gradient of f. The loss function that they use for the discriminator model is as follows:

$$\mathop{\mathbb{E}}_{y\sim Y}(f(y)) - \mathop{\mathbb{E}}_{\hat{y}\sim \hat{Y}}(f(\hat{y})) + \lambda \mathop{\mathbb{E}}_{y\sim Y, \hat{y}\sim \hat{Y}, a\sim \mathcal{U}(0,1)}((||\nabla f(y\cdot a + \hat{y}\cdot(1-a))||_2 - 1)^2)$$

The first part is identical to the original Wasserstein GAN. For the last part, a random interpolation between a real example and a fake example is selected, the length of the gradient of f at that point is calculated and its squared difference to the value 1 is added to the loss. Therefore, the summand is minimal if the gradient of f has a length of 1 in the convex hull of the real and fake examples.

Mao et al. [MLX+17] suggest yet another loss function. Instead of having the discriminator predict the logit probability of an image being real, they predict the probabilities itself. The loss function for the discriminator is as follows:

$$\frac{1}{2}\mathop{\mathbb{E}}_{y\sim Y}((D(y)-b)^2) + \frac{1}{2}\mathop{\mathbb{E}}_{z\sim Z}((D(G(z)-a)^2)$$

The loss of the generator is as follows:

$$\frac{1}{2}\mathop{\mathbb{E}}_{z\sim Z}((D(G(z)-c)^2)$$

If $a = 0$ and $b = c = 1$, the discriminator predicts probabilities between 0 and 1. Instead one might chose to predict probabilities scaled between -1 and 1.

Previous works have aimed at automating the coloring step. They require large collections of correlating pairs of colored frames and line art to accomplish this. By taking colored images and edge images created from the colored images, Isola et al. [IZZE17] and Yonetsuiji et al. [Yon17] are able to automate the process of coloring line art. Additionally, the tool by Yonetsuiji et al. allows the user to select colors for individual areas and the tool will complete the coloring. See Figures 3.9 and 3.10 for examples.

These works use the Canny edge filter [Can86] or similar traditional image processing methods to generate edge images and these do not reflect the line art used during the creation of traditional animation. Its quality hinges on the selection of a threshold parameter and the color-coding of douga is not taken into account.

Variational Auto Encoders Kingma et al. [KW13] use an auto encoder to find a mapping between a normal distribution and the target distribution. Two models, an encoder and a decoder are optimized jointly. The encoder maps an image to a normal distribution, represented by its mean and the diagonal of its covariance matrix. The decoder maps a sample from a normal distribution to an image.

(a) An edge image generated from a photo, using a traditional edge-detection filter.

(b) A color image generated from the edge image.

Figure 3.9: Example of the model by Isola et al. [IZZE17].

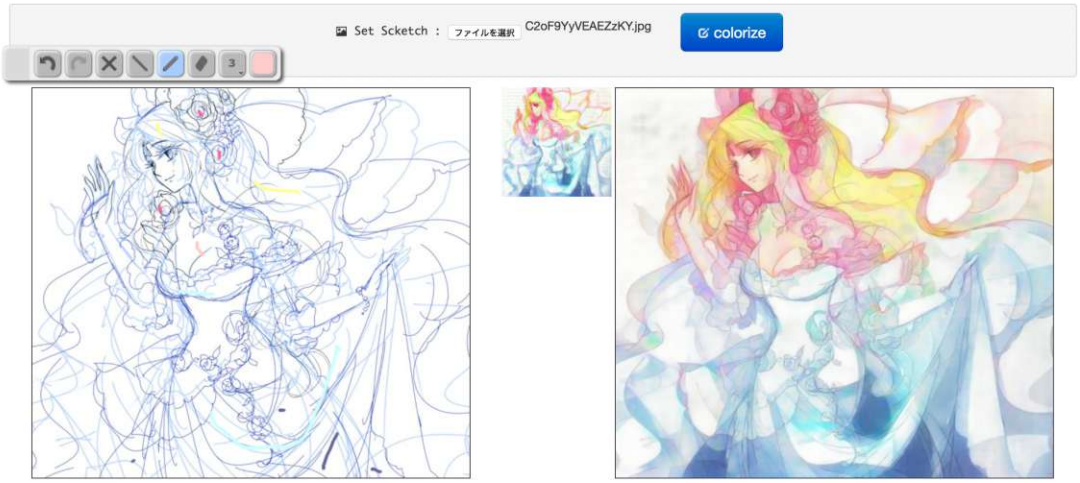


Figure 3.10: Screenshot of the interface of the Paints Chainer [Yon17] application.

The training objective is to minimize both the difference between an image and the result of encoding and decoding it, and the Kullback-Leibler divergence between an image's latent distribution and a standard normal distribution. The former means that for each image there is a latent point that is approximately decoded to it. The latter means that the latent distribution over all images approximates a standard normal distribution. The result is that images can be generated by decoding samples from the standard normal distribution. As the decoder only approximates the inverse of the encoder, results tend to be blurry.

Flow-Based Models The VAE can be improved by modelling the decoder as a function that can be inverted explicitly. Dinh et al. [DKB14] propose a set of invertible functions as the building blocks for their model. Kingma et al. [KD18] improve on it by introducing the invertible 1×1 convolution. Invertible differentiable functions (bijectors) allow explicitly calculating the probability density function of a known distribution transformed by the function. This allows minimizing the cross-entropy between samples and the approximated distribution directly. This solves the instability present in GANs and the blurriness present in VAEs. The target distribution can be sampled by decoding samples from the known distribution.

As the composition of bijective functions is itself a bijective function, models created this way are also bijective. If X is a distribution with probability density function P_X, the probability density function $P_{f(X)}$ of the transformed distribution $f(X)$ is calculated as follows:

$$\mathrm{P}_{f(X)}(y) = \mathrm{P}_X(f^{-1}(y)) \cdot \left|\det\left(\frac{d}{dy} f^{-1}(y)\right)\right|$$

This equation can be inserted into the cross entropy loss and simplified to attain the following:

$$l_{CE}(x) = -\log\left(\mathrm{P}_X\left(f^{-1}(y)\right)\right) - \log\left(\left|\det\left(\frac{d}{dy} f^{-1}(y)\right)\right|\right)$$

To evaluate it, the log probability of the distribution X, the inverse of f and the log determinant of its Jacobian need to be known. These can be efficiently calculated for all functions used in such models. The log probability of a unit normal distribution is given as the following quadratic function:

$$\log(\mathrm{P}_X(x)) = -x^2 \frac{1}{2} + \frac{1}{2} \log\left(\frac{1}{2\pi}\right)$$

Many common operations in machine learning are not bijective. Therefore, Kingma et al. [KD18] introduce three invertible operations to build their fully bijective model. An invertible 1x1 convolution, which performs a matrix multiplication on each pixel

of its input with learned weights. The weights are stored and optimized as their LU decomposition, speeding up the calculation of its inverse. The log determinant of its Jacobian is the log determinant of the weights matrix. An affine coupling layer, which performs a lifting operation on the input. When only performing additions, the log determinant of the Jacobian of a lifting scheme is always 0. And an invertible version of batch normalization, which just applies a denormalization in its inverse. The log determinant of its Jacobian is the sum of the logs of the scalar factors.

The *lifting scheme* allows them to incorporate arbitrary non-bijective functions into the model. Given a non-bijective functions g and h, which may be modelled by a convolutional neural network, the following function, that operations on two parts of the input vector separately, is bijective:

$$f(x_a, x_b) = (x_a \cdot g(x_b) + h(x_b), x_b)$$

Its inverse is as follows:

$$f^{-1}(x_a, x_b) = \left(\frac{x_a - h(x_b)}{g(x_b)}, x_b\right)$$

And its log Jacobian determinant as follows:

$$\log\left(\det\left(\frac{d}{dx} f(x_a, x_b)\right)\right) = \text{sum}(\log(|g(x_b)|))$$

Recurrent Neural Networks A series of elements $\mathbf{x} = (x_1, \ldots, x_N)$ can be predicted by finding a function $f(x, h)$ such that for a hidden state $\mathbf{h} = (h_1, \ldots, h_N)$, $f(x_i, h_i)$ predicts (x_{i+1}, h_{i+1}). This method has been used for the generation of text [SMH11] and hand-writing [Gra13], among others.

As each prediction depends on all previous evaluations of f, this method is not well suited for parallel hardware. An alternative is to have f not depend on the hidden state but, instead, a window of $\mathbf{x}$. This still requires sequential prediction during generation, but allows training to be parallelized per element. This method has been used for the generation of text [VSP+17, RNSS18].

A single element is predicted as a distribution over all possible symbols. For text, this is a categorical distribution over a set of characters or words, predicting a probability for each. When dealing with images, the set of symbols becomes a set of colors. This turns the task of generating images into a classification task, which is a well-studied class of problem.

Generally, RNNs operate on one-dimensional sequences of elements, whereas images are a two-dimensional plain of elements. Oord et al. [OKK16] address this by using a

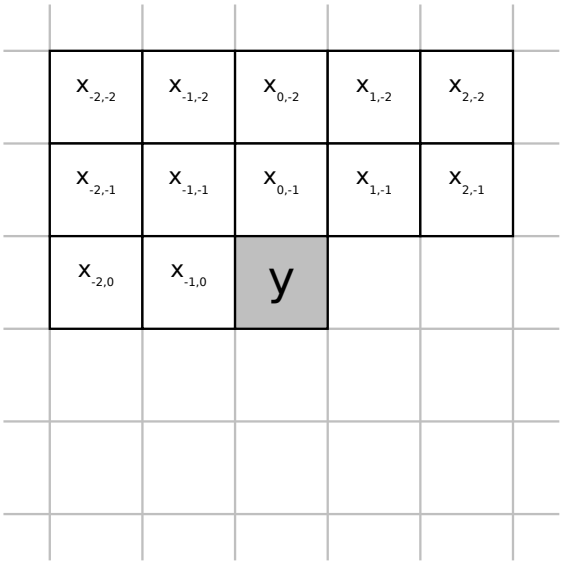


Figure 3.11: Oord et al. [OKK16] predict pixel of an image one by one, line by line. The pixel y is predicted by a rectangular window of previously predicted pixel x.

rectangular window, spanning a fixed number of rows and columns over the image, as the input to the learned function. See Figure 3.11.

To generate images, Chen et al. [CRC+20] re-purpose the GPT text generation model by Radford et al. [RNSS18] by replacing its alphabet of letters and letter combinations with 512 colors. Unlike the rectangular window used by Oord et al. [OKK16], they do not model spatial awareness directly. Pixel are predicted one by one, line by line, each depending on a window of previously predicted pixel.

Choromanski et al. [CLD+20] introduce an unbiased approximation to the attention layers used in these models that has a runtime of $O(n)$ instead of the original $O(n^2)$, with n being the size of the window that each prediction depends on. This method could also be use for image generation, but this has not been attempted yet to our knowledge.

3.5 Image Augmentation

Augmentation is the task of creating new training examples from existing ones by applying transformations to them. These include random color changes, scaling, rotation, flipping, and cropping. Additionally, it includes adding noise, following a distribution like the normal distribution, to images, and masking, where pseudo-random pixels of the image are painted over. Different shapes can be used for these masks, like rectangles with random size and position [ZZK+20] or samples from a dataset of shapes [LRS+18]. As the amount of available training data is orders of magnitude smaller than the ones required by existing successful image processing tasks, augmentation is a valuable tool. Zhao et al. [ZZC+20] examine the effect of augmentation during GAN training.

CHAPTER 4

Cel Segmentation

The goal of this work is to recreate douga. However, creating example douga requires professionals. A simpler but related problem is the task of segmenting cels from frames. Such segmentation maps contain a subset of the information of douga, but creating them is simple enough that an amateur can do it.

The segmentation maps for this work are stored as images where each pixel of a frame is colored either white, for pixels belonging to a cel, or black, for pixels belonging to the background or books. We manually create a dataset of such segmentation maps. A segmentation model is trained on this data. Different augmentation methods, model sizes, and loss functions are compared, and the best combination is used for the douga reconstruction in Chapter 5.

4.1 Dataset

To our knowledge, there is no easily available dataset of segmentation maps for frames of traditionally animated TV shows and movies available. Therefore, a dataset of segmentation maps is created manually from final color frames. Color frames are selected from an internal database of random animation data. Creating the segmentation maps takes about 10 minutes per image on average, with some complex images taking as much as 60 minutes.

Frame Selection Budget and time constraints make it necessary to be selective about which frames to use for examples. There are various methods of selecting examples, with different advantages and drawbacks. If the goal is to maximize the average accuracy over the entire dataset, examples should be created for a random sample of the dataset. If weighting the examples is an option, importance sampling can be used to achieve the same goal more efficiently. When the goal is to create a model that achieves high accuracy on an arbitrary subset of the dataset, one can limit it to certain features.

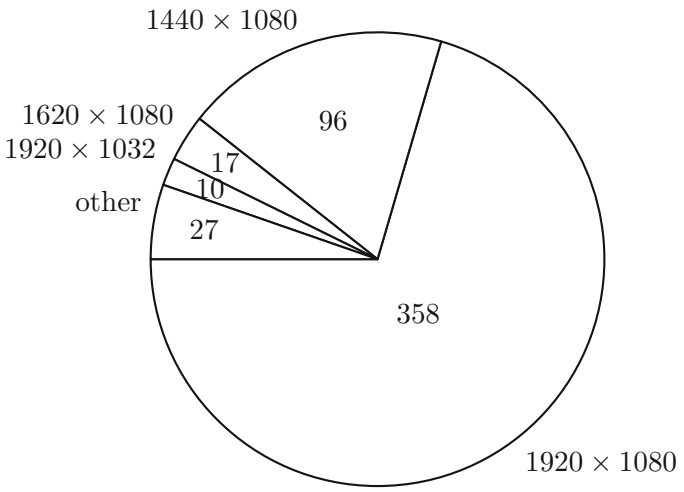


Figure 4.1: Distribution of image resolutions in the segmentation dataset. The majority of images have a resolution of 1920 × 1080 pixel. The numbers represent the number of images with the same resolution.

Not all examples are created equal. Examples the model is already performing well on provide little new information. Their loss is small, and such is their impact on the gradient. Examples can be selected iteratively, taking into account the model's performance on previously selected examples, by expanding the number of examples for classes the model is performing at poorly. This process can be repeated multiple times.

Another method is to select a wide range of different examples, increasing the chance of including "difficult" ones. Clustering algorithms, like k-means clustering, can be used to find a representative subset of the dataset. Each possible example's (squared) distance to an example in the subset is minimized.

A mix of these approaches is taken here. First, a small number of random color frames from the database are selected as examples using k-means clustering on edge images of the frames to identify visually distinct frames. The GCN model [PZY+17] is trained on this initial dataset. The trained model is then used to find new images for examples. The model is used to estimate the "difficulty" of additional frames. The model outputs a Bernoulli distribution for each pixel. This distribution's entropy can be used as an estimate for the model's confidence in the result. Then, additional frames of the highest confidence and of the lowest confidence are selected. The former contains already mostly correctly labeled frames that can be added to the dataset with little adjustment. The latter contains difficult frames for which segmentation maps need to be created manually. This process is repeated until a total of about 500 frames is selected.

Finally, the dataset is randomly partitioned into a training set and a test set. Throughout this work, the former is used for training and the latter is used for testing. A sample of the segmentation maps can be seen in Figure 4.4.

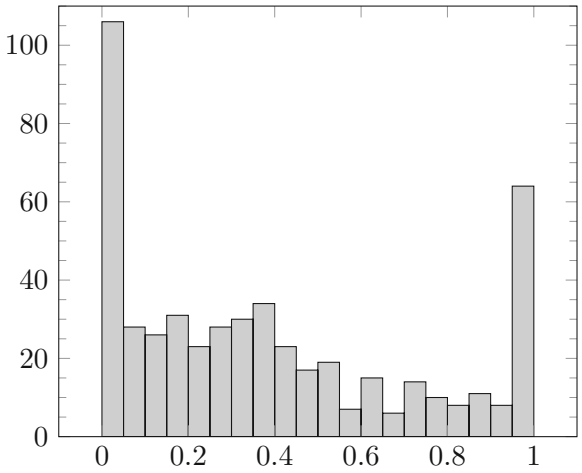


Figure 4.2: Histogram of the distribution of dynamic elements in the segmentation maps. A value of 1 represents frames that were created entirely by the workflow for moving elements. A value of 0 represents frames that only contain backgrounds.

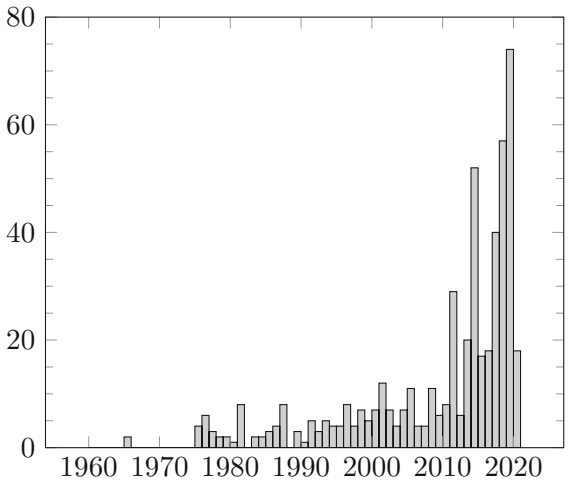


Figure 4.3: Histogram of the release years of the productions frames are segmented from.

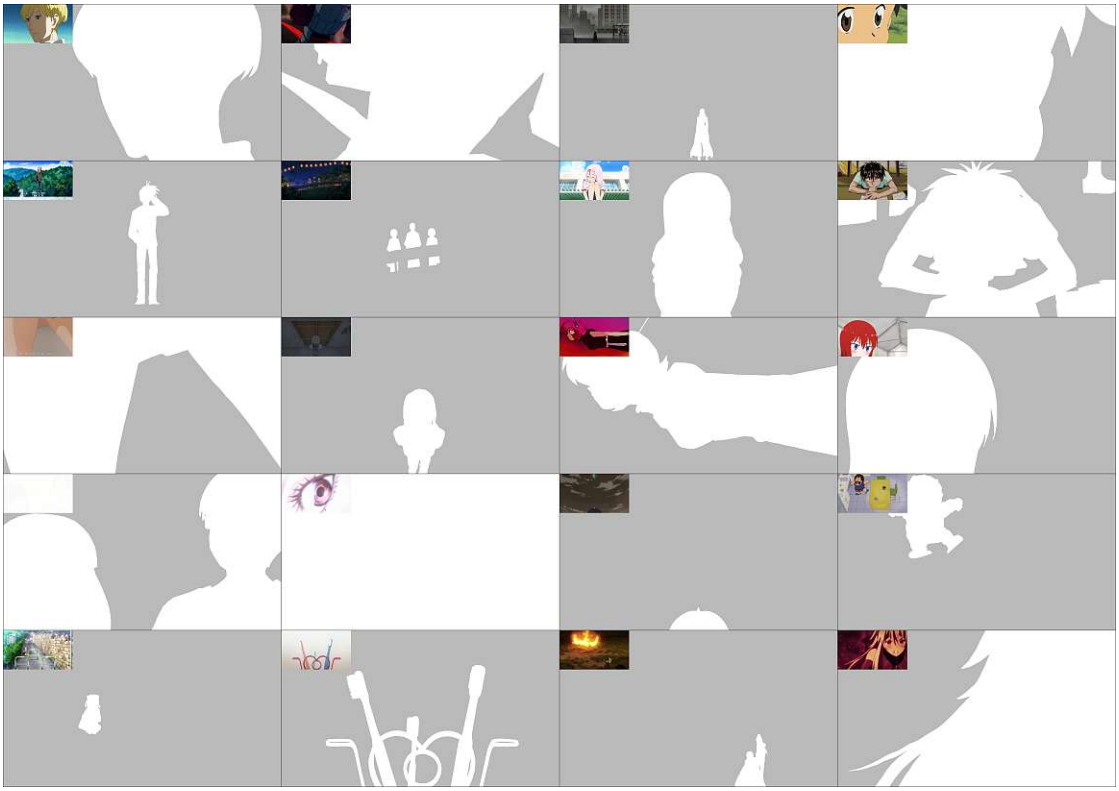


Figure 4.4: Examples of the manually predicted segmentation maps. The white areas represent pixels belonging to a cel and the gray areas represent pixels belonging to the background or books. The corresponding color image is visible in the corner. Color images by AIC, BONES, Group TAC, Kyoto Animation, MADHOUSE, P.A. Works, Production I.G, Shaft, Shin-Ei Animation, SILVER LINK., Studio 3Hz, Studio DEEN, Studio Gokumi, Toei Animation, Wit Studio and Xebec.

The resolution of the segmentation maps corresponds to the resolution of the original frames. The majority of the images have a resolution of 1920 by 1080 pixels, which corresponds to the now-standard 16:9 aspect ratio. Many older productions were made with an aspect ratio of 4:3 and later re-released at a resolution of 1440 by 1080 to fit modern displays. Figure 4.1 shows the distribution of image resolutions.

The dataset contains frames consisting entirely or mostly of moving elements, and frames with only a few small or no moving elements, and everything in between. A histogram of the distribution of moving elements can be seen in Figure 4.2. The oldest show that is segmented is from 1965. Most frames are taken from shows released in 2010 or later, which may be explained by the increase in the number of animation productions released in the past decade as well as the increase in online distribution [MSR+19]. The distribution of release years in the dataset can be seen in Figure 4.3.

4.2 Data Augmentation

In order to increase the amount of variation in the training examples, random transformations are applied to the images during training, effectively increasing the number of training examples.

Pixel-wise Transformation The examples' brightness and contrast are varied randomly following a normal distribution with a standard deviation of 25% and 50% respectively. Additionally, the colors' hue is rotated randomly following a normal distribution with a standard deviation of 45°.

Affine Transformations and Cropping During training, random scaling by a factor between 0.5 and 2 and random rotation by -10 to 10 degrees is applied to the images. All images are padded or cropped to a resolution of 1920×1080 pixel to allow for efficient batch processing. Additionally, frames are flipped horizontally with a chance of 50%.

Background Replacement During training, the frames' backgrounds are replaced with random other backgrounds with a certain probability. A separate dataset of frames containing only backgrounds is created to accomplish this. The existing handmade segmentation is used to composite the original frame with the randomly selected background. The probability of replacing the background is selected such that each replacement background has the same probability as each training example.

Texture Overlays Frames are randomly overlaid with textures of paper, wood, concrete, dirt, and metal with a certain chance. The texture is selected independently for foreground and background. Fifteen textures are collected from Texture.com [tex20]. The textures are normalized to a mean value of 1.0 as to not change the overall brightness or color of the images. Textures are tiled over the size of the image at a random offset. They are applied by multiplying them with the images. A sample of the textures can be seen in Figure 4.5.

Rectangular Erasure A randomly placed rectangular area is removed from the image. The size of the rectangle follows an uniform distribution between 0 and 50% of the corresponding dimension of the image and is selected independently for the two axes. Both the segmentation mask and the color image are set to zeroes within the rectangular region.

Hard Examples Early runs have shown that the model is accurate on some kinds of images, particularly close-up shots of character's heads, and inaccurate at extreme close-up shots of other body parts and shots of objects. As the former is much more common, the model may be incentivized to ignore other shots. To test this hypothesis, a separate training set containing only these "hard" examples is created. This dataset is a

Figure 4.5: The data is overlayed with various concrete, wood, paper, metal, and dirt textures. Textures are collected from Texture.com. [tex20].

subset of the whole dataset and has 73 images. Testing is still performed using the full test set.

4.3 Model

The model is based on an existing segmentation model by Peng et al. [PZY+17], the *Global Convolution Network* (GCN). It contains the blocks of the ResNet model [HZRS16] followed by convolutions. The convolutions are factorized into horizontal and vertical convolutions, allowing larger kernels (25 pixels) while keeping the computation and memory requirements low. The latter is their main contribution. They demonstrate the model on the PASCAL VOC dataset [EVGW+11] of photos of everyday objects.

The GCN model is made out of the ResNet model's residual blocks, global convolution blocks, and boundary refinement blocks (BR). A visualization of the GCN model can be seen in Figure 4.7.

4.4 Pre-Training

The residual blocks from the ResNet model are pre-trained on ImageNet classification. He et al. [HGD19] demonstrate that pre-training can speed up convergence and improve accuracy for small datasets. ImageNet features photos of everyday objects, people, and animals. On the other hand, this work focuses on stylized, non-photo-realistic illustrations.

Figure 4.6: Different augmentations of the same image. Affine transformations and contrast and brightness changes are applied to it. The background is replaced randomly. Rectangular regions are removed. Cel image by Tonari Animation, backgrounds by Shaft, Studio Chizu and Tonari Animation.

Features learned from ImageNet may not necessarily transfer well to the segmentation of traditional animation.

4.5 Loss Functions

Both binary cross-entropy and focal loss are used for training and compared. Only pixel-wise loss functions are examined. Future work may examine loss functions that consider entire images, like mixture models and adversarial models.

4.6 Training Procedure

The Adam optimizer is used for training with a learning rate of $1e-4$, linearly decreasing to $1e-5$ over 4,000 epochs. A batch size of 4 is used for every run. For the entire dataset, a single epoch encompasses 108 optimizer iterations. Training is stopped once the validation loss does not decrease for 100 epochs. All runs, except for the run with Resnet-101, are performed on four Nvidia GTX 2080Ti. The Resnet-101 run is performed on a single Nvidia V100 due to its increased memory requirements.

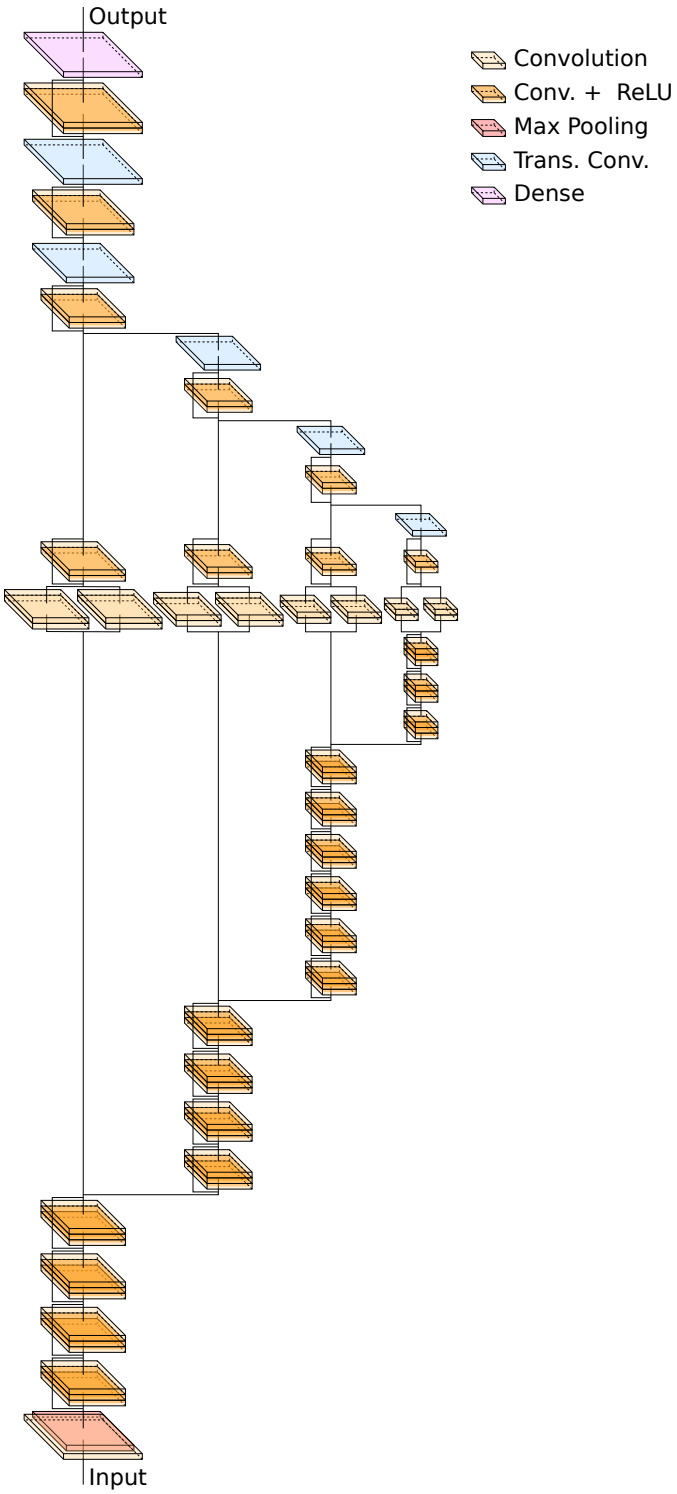


Figure 4.7: Visualization of the GCN model [PZY+17] as it is used for cel segmentation.

4.7 Implementation

The implementation is based on an existing implementation of the GCN architecture in PyTorch [PGM+19]. It is extended by several functionalities:

- It is extended to read color images from an existing animation database and read and encode segmentation maps and douga from our proposed representation.
- It is extended by both existing augmentation methods (brightness and contrast changes, affine transformations, rectangle erasure) and augmentation making use of the inherent layered nature of animation (background replacement and per-layer augmentation).
- The model itself is extended with dropout for regularization.
- It is extended by several loss functions: binary cross-entropy loss, binary focal loss, and adversarial loss.
- Programs for the visualizations and evaluations used throughout the works are implemented.

4.8 Evaluation

A comparison between the different experiments and the accuracies achieved with each can be seen in Table 4.2. For each run, both the highest accuracy achieved and the number of iterations to achieve it are presented.

The GCN model with a ResNet-50 backend pre-trained on ImageNet is used as a baseline. This baseline resembles the model that the authors of the GCN model Peng et al. [PZY+17] use. Variations to the model are compared to this model. This model achieves a validation accuracy of 90%. Accuracies are subject to noise. An accurate comparison requires future experiments.

- Using the shallower and smaller ResNet-34 over ResNet-50 as the backend for the GCN model degrades validation accuracy. Using ResNet-101 over ResNet-50 improves it. Due to the memory requirements of the ResNet-101 model, these runs are performed on different hardware. Therefore, the training time is not comparable with the other runs. It may be surprising that increasing model capacity does not harm generalization but even improves it. Using an even larger model may improve the accuracy further.
- Performing the background replacement improves validation accuracy from 90% to 92%. Training with textures diminishes the accuracy to 88%. If the model relies too much on texture borders, as the texture for the foreground and background are selected independently, adjusting the frequency at which textures are presented during training may improve accuracy.

True Class \ Prediction	Background or Book	Cel	Recall
Background or Book	**55.8%**	4.2%	96.9%
Cel	1.8%	**38.1%**	90.0%
Precision	92.9%	95.5%	

Table 4.1: Confusion matrix of the best model for cel segmentation on the validation dataset.

- After training only on hard examples, the model achieves a training accuracy only half a percentage point less than when training on all images. However, the validation accuracy, using the same validation images as for the other runs, decreases to 84%.
- Using Dropout or focal loss both slightly improve validation accuracy by half a percentage point.
- Performing rectangular erasure on both the color image and segmentation maps improves validation accuracy from 90% to 93%.

After these observations, a run is performed combining all variations that improved validation accuracy – namely, the larger ResNet-101 backend, focal loss, dropout, background augmentation, and rectangle erasure. Doing so leads to the highest validation accuracy of 93.9% and a test accuracy of 99.8%. This accuracy is achieved after 148 hours of training over 6 days. A plot of both the pixel-wise accuracy and the focal loss for both training and validation batches can be seen in Figures 4.8 and 4.9. As the training is subject to random augmentation, they are noisy. The data is approximately evenly sub-sampled for the plot. Segmentation maps created with this model can be seen in Figure 4.10.

To visualize which features the model is responding to, Mordvintsev et al. [ACM15] use gradient descent to optimize an input to the model towards particular activation. They call this *deep dream*. Their code is adapted to visualize the features of the second-to-last layer of the model. This visualization can be seen in Figure 4.11.

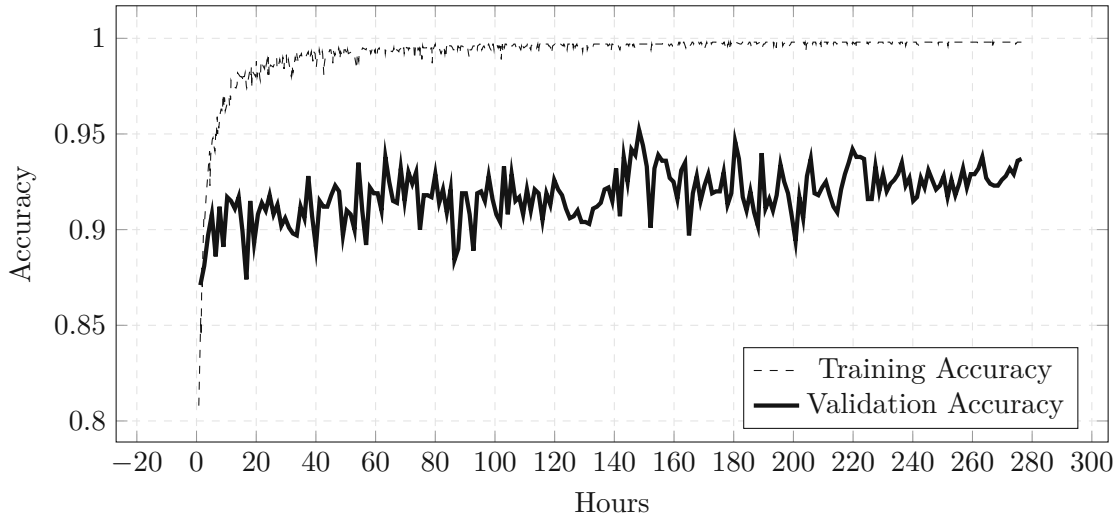


Figure 4.8: Accuracy graph of the final run.

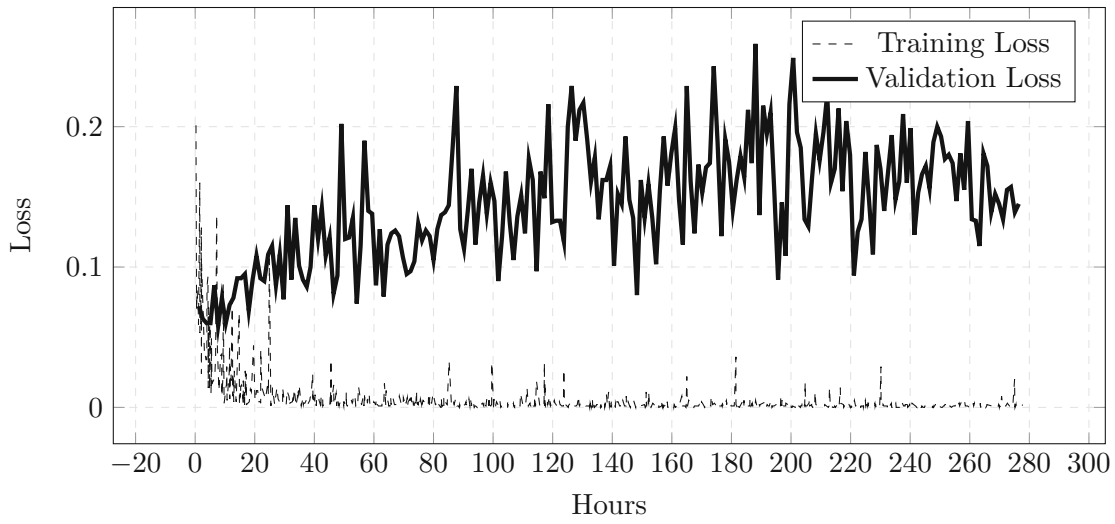
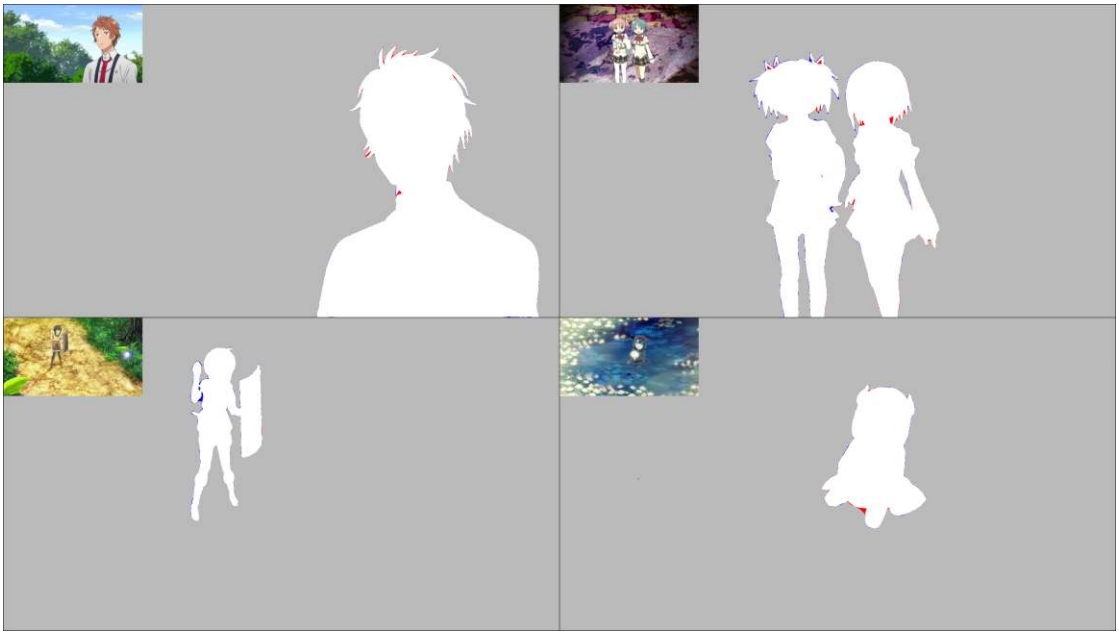
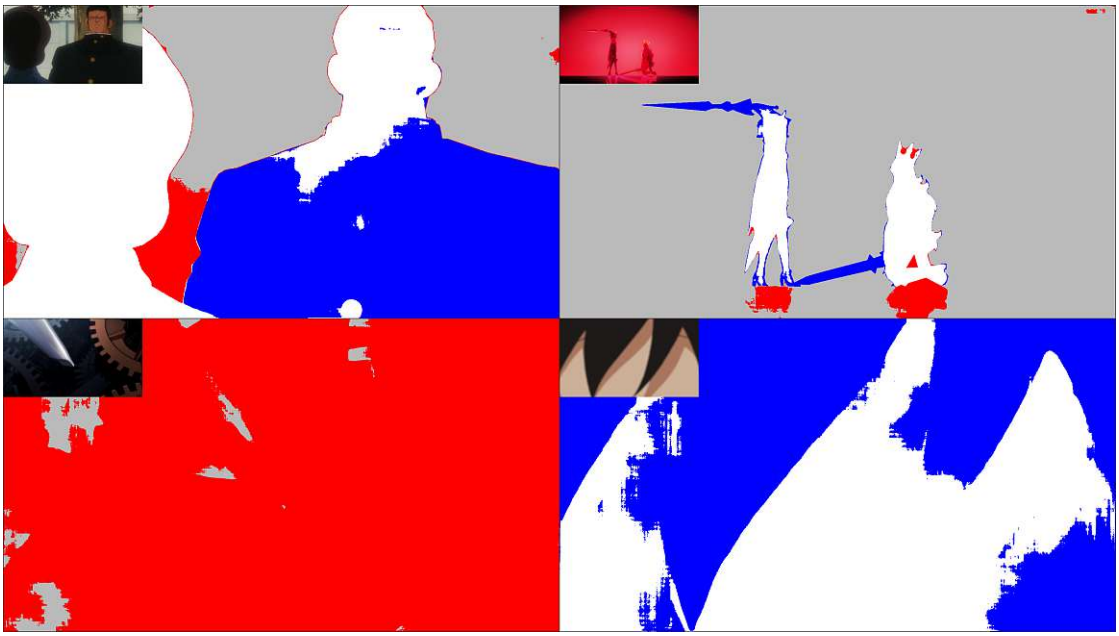


Figure 4.9: Loss graph of the final run.



(a) Medium closeup shots to long shots, as well as group shots, are accurately segmented.



(b) Low contrast and blurred shots, shots with CG elements, and extreme closeups are segmented incorrectly.

Figure 4.10: Examples of validation images segmented by the segmentation model. Gray and white color represent correctly labeled background and foreground, respectively. Red represents background elements that are incorrectly labeled as foreground elements. Blue represents foreground elements that are incorrectly labeled as background elements. The color input can be seen in the top left corner of each image. Color images by Group TAC, HORNETS, MADHOUSE, NAZ, P.A. Works, Satelight, Shaft, SILVER LINK. and Xebec.

Backend	Loss	Regularization	Data Augmentation	Best Train Accuracy	Best Validation Accuracy	Time to Best Validation Accuracy
ResNet-50 (Baseline)	CE	None	Affine, Brightness	99.6%	90.0%	34.4h
ResNet-34	CE	None	Affine, Brightness	99.5%	88.5%	20.5h
ResNet-101	CE	None	Affine, Brightness	99.6%	91.2%	64.1h[1]
ResNet-50	CE	None	Affine, Brightness, **Background**	99.6%	92.1%	25.9h
ResNet-50	CE	None	Affine, Brightness, **Background, Textures**	98.7%	87.9%	24.7h
ResNet-50	CE	None	Affine, Brightness, **Hard Only**	99.1%	83.9%	5.4h
ResNet-50	CE	**Dropout**	Affine, Brightness	99.6%	90.6%	20.6h
ResNet-50	**FL**	None	Affine, Brightness	99.6%	90.5%	35.6h
ResNet-50	CE	None	Affine, Brightness, **Rectangle Erasure**	99.7%	92.5%	2.7h
ResNet-101	**FL**	**Dropout**	Affine, Brightness, **Background, Rectangle Erasure**	99.8%	93.9%	148.3h[1]

Table 4.2: Comparison of different models and augmentation. The highest accuracy achieved over the test data and training data are presented; higher is better. The number of iterations after that accuracy is achieved is presented; lower is better.
[1] Not comparable with other timings, as it was run on different hardware.

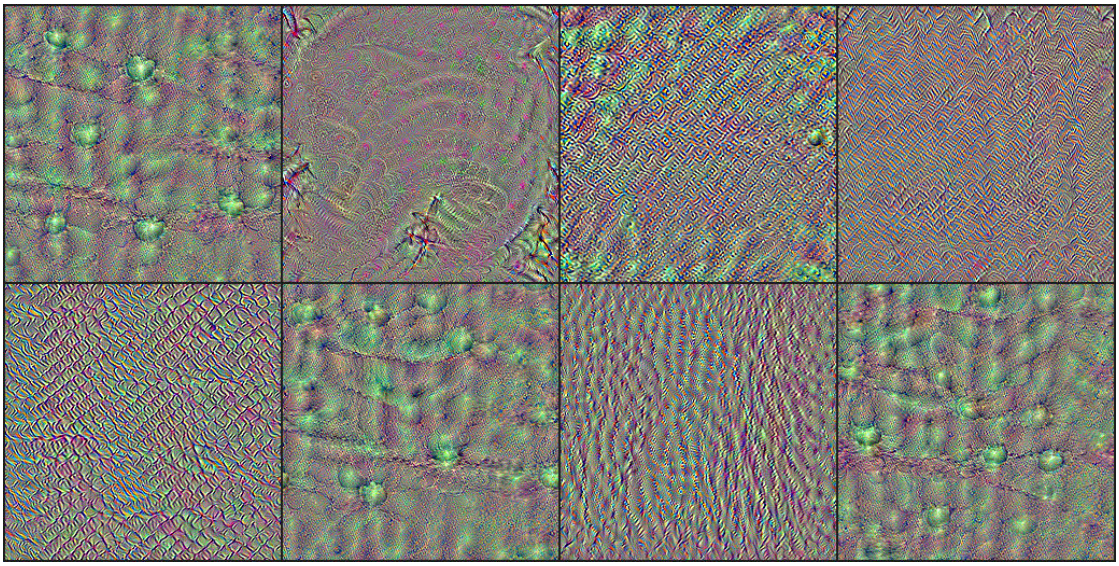


Figure 4.11: A visualization of the segmentation model created by maximizing the activation of the second to last layer on random noise. Each image represents a feature dimension in the layer and what general structure it is most sensitive towards.

CHAPTER 5

Douga Reconstruction

The second step to this work is to use the cel segmentation and additional data to reconstruct douga. Like the cel segmentation, this task involves acquiring a dataset, selecting augmentation methods, a model, and a loss function, and evaluating the results. As douga classes are more ambiguous than the classes of the previous task, both pixel-wise loss functions and adversarial loss functions are evaluated. Finally, the pre-trained segmentation model is incorporated into the douga generation to use the larger segmentation dataset.

5.1 Douga Schema

The purpose of douga is to specify and communicate the position, shape, and movement of objects in an animation. It contains information on the kind of color in each region of the frame (e.g., shades and highlights), but not the color itself. Douga differs between productions and studios in what classes of lines and regions are represented and how they are represented. For consistency within the dataset and the output of the model, a standard representation is required. Therefore, we propose a set of classes and a palette to represent them. Through discussions with professional animators, the most common and most useful classes were selected, as well as suitable colors for each class.

Classes The generated images assign each pixel one of ten different classes. Additionally, several classes are separated by instance. The instance information is not used in this work. The aim is to include all primary features present in real douga while still being easy to process automatically. Real douga often includes written notes to give further instructions to the color artists, or crosses to identify regions as transparent. These are intentionally left out in order to simplify processing. Automatic processing of notes may require OCR methods, which are beyond the scope of this work. For some frames in productions, multiple douga are used to create transparency and parallax effects during

compositing. These cases are ignored. Semi-transparent foreground elements are treated as part of the background, and all dynamic elements are included in a single douga. The colors are chosen such that the individual classes can be easily separated in code. An example of some encoded classes can be seen in Figure 1.1. A sample of the dataset can be seen in Figure 5.1.

The following classes are encoded:

Flat Area Areas that are invisible on the cel or contain flat coloring are colored white.

Visible Line Lines that will be drawn as lines in the final image are drawn as black and brown lines. These are used for outlines, edges on objects, and details.

Shading Line Lines that mark the edge between two shades in an area are drawn as dark purple lines. These are not visible as lines on the final image, but instruct the color artist to use a brighter color on one side and a darker one on the other, according to the lighting in the scene.

Color Edge Lines that only exist in the line-art to mark the edge between two differently colored areas are drawn as green and cyan lines. These are used for patterns on clothes, blush on faces, and other details.

Highlight Line Lines that represent the edges of highlights on surfaces are drawn as red lines. These serve a similar role as the Shading Lines.

Book Line Outlines of books, which are areas that will be occluded in the compositing step, are represented by gray lines. As they will be occluded by another layer later, it may not be necessary for the animators to draw the occluded areas.

Shading Fill Areas that are shaded darker than their surroundings are represented by different purples. This helps the color artists to identify which side of a Shading Line should be darker.

Color Fill Areas that are colored differently from their surroundings are represented by light green and cyan tones. The exact colors are not specified in the douga but by a style guide available to the color artists.

Highlight Fill Highlighted areas that are shaded brighter than their surroundings are represented by different light yellow tones. These serve a similar role as the Shading Fills.

Book Fill Parts of the cel that painted foreground elements will occlude after the compositing step are represented by different light grays. See Book Line.

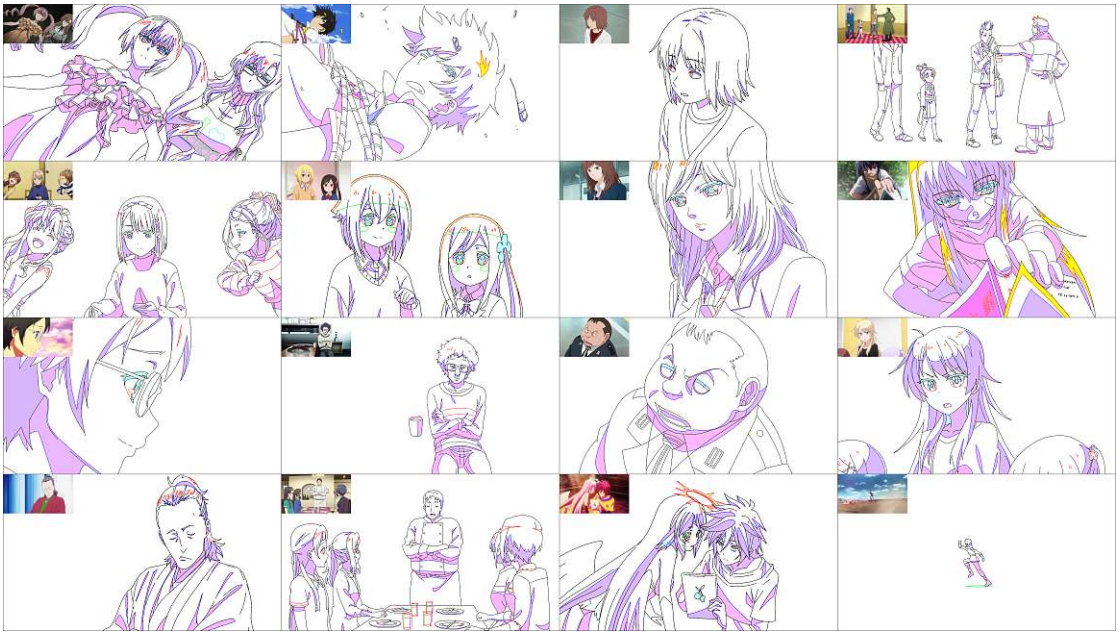


Figure 5.1: Interpretations of what the original douga could have looked like for each of the color images shown in the top-left corner. Douga drawn by Tonari Animation artists. Color images by A-1 Pictures, C2C, CloverWorks, diomedéa, Doga Kobo, J.C. Staff, MADHOUSE, P.A. Works, Production I.G, Toei Animation and White Fox.

5.2 Dataset

To train a model, a dataset of examples is required. Original douga is generally not readily available. After consultation with professional animators and directors, it became clear that a dataset would have to be created by professional hands. To create the training data for the model, professional animators were given colored frames from existing series and movies and asked to recreate the douga corresponding to these frames.

Frame Selection The same considerations as to the frame selection for cel segmentation, described in Section 4.1, apply when selecting this task's frames. The number of examples is limited to about 200 images. This is about half the size of the smallest dataset used by Isola et al. [IZZE17].

Again, a mix of different approaches is taken to select the frames used as examples. An initial set of 60 frames is chosen randomly, excluding frames with on-screen text. Early experiments and the experiments on cel segmentation show that the model performs better on certain types of shots, particularly medium shots and close-ups of humans. Therefore, it is decided to focus on these types of shots. This limitation is somewhat arbitrary. However, one may argue that, in practice, a tool that works reliably for a certain class of images is more useful than a tool that works unreliably for a broad class

of images.

Additionally to the initial 60 frames, a random sample of a thousand frames is taken from the internal database, and images not meeting the criteria are manually removed. These criteria are:

- Only frames showing human-like characters and no animals.
- No frames with interlacing artifacts or frames that are overly blurry. These are primarily common in old shows, ca. pre-2000. These are the result of the distribution, not necessarily the production of a show.
- No images with on-screen text. These are commonly found in title or credit sections. During production, these would be placed during compositing and are therefore not interesting for this work. Some releases also contain subtitles in the video stream.
- No overly dark or bright images, for some definition of "overly". During production, these would be drawn in neutral colors and then darkened or brightened during compositing and are therefore not interesting for this work.
- Only medium shots to close-ups.

Acquisition To create the training data, professional animators draw interpretations of what the selected frames' original douga might have looked like. These are drawn in drawing applications like Clip Studio Paint [Cel01] or Retas Studio [Cel08], following the agreed-upon color palette. A computer program then reads these images. Each pixel is individually compared to the colors in the palette, and the closest match is chosen to identify the class. An example of this process is shown in Figure 5.2.

5.3 Augmentation

The same augmentation as for the cel segmentation can be used. Brightness and contrast are randomly changed, rotation, scaling, cropping and flipping are applied.

5.4 Model

The GCN model [PZY+17], as described in Section 4.3 is used for douga generation as well. The number of outputs on the last layer is increased to ten, which corresponds to the number of pixel classes in our douga dataset. Additionally, the kernel size for the GCN blocks is increased from 25 to 49 pixels, as early experiments have shown this to improve the classification of shaded and highlighted areas. See Figure 5.6.

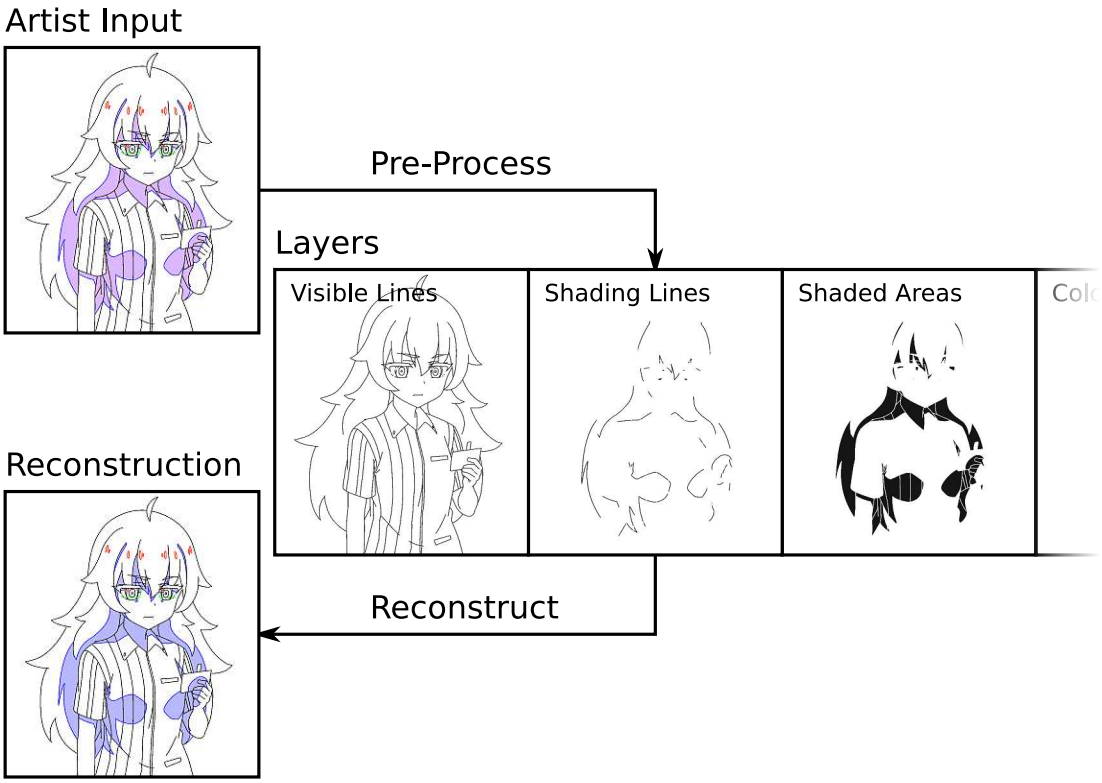


Figure 5.2: The process of reading the douga created by the artist. The first image shows the douga, as created by the artist and colored according to the proposed color palette. This image is decomposed into its classes, based on a list of colors. As multiple colors may represent the same class, a reconstruction from these classes is not necessarily identical to the input. Artwork by Tonari Animation.

5.5 Loss Function

Multiple loss functions are considered. Their main difference is whether they rate each pixel independently or the entire image as a whole. Each has its own advantages and disadvantages.

5.5.1 Pixel-wise Loss

The task of generating douga can be framed as a segmentation task. The model produces a probability for each class and pixel in the input. The most probable class of each pixel is selected, and the associated color is written to the output. The loss function is calculated from the probability given to the pixel's actual class.

As the loss only operates on individual pixels, spatial correlation is not taken into account. This leads to the problem visualized in Figure 3.4. A related problem is the problem of differentiating between multiple similar classes. For example, a pixel may get a 30%

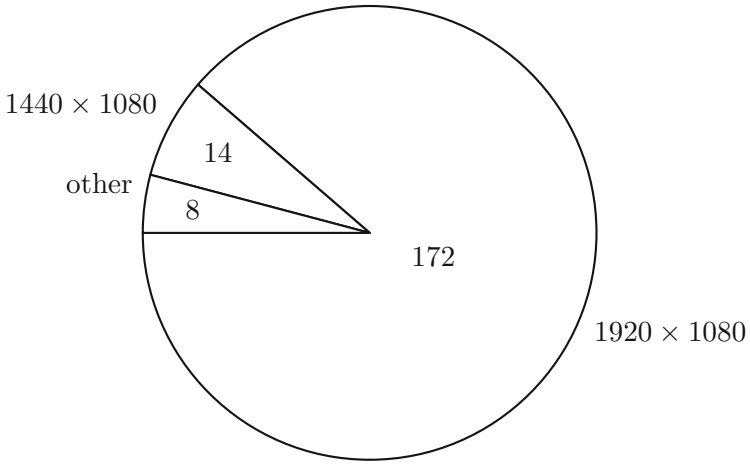


Figure 5.3: Distribution of image resolutions in the douga dataset. The majority of images have a resolution of 1920 × 1080 pixel. The numbers represent the number of images with the same resolution.

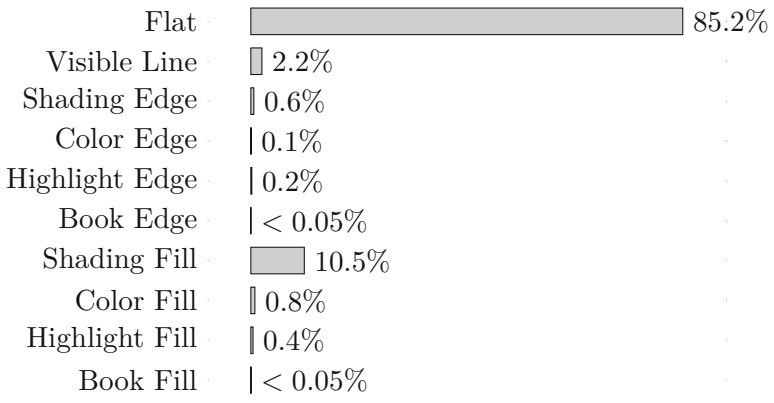


Figure 5.4: Histogram of the frequency of each class in the douga dataset.

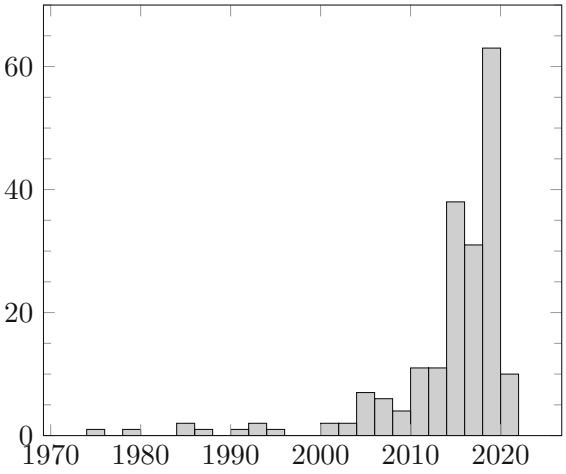


Figure 5.5: Histogram of the release years of the productions from which douga is predicted for.

Figure 5.6: The effect of increasing the kernel size of the GCN blocks. The leftmost image is the interpretation of what the original douga could have looked like by Tonari Animation artists. The center image shows the result of the GCN model trained with a kernel size of 25. The rightmost image shows the result of the GCN model trained with a kernel size of 49. Fewer pixels of the shaded areas (blue) are missed with the larger kernel. Color image by Shaft.

probability of being a solid line, a 30% probability of being a shading edge, and a 40% chance of being a flat area. While the pixel's chance of belonging to either line class is 60%, it will be classified as a flat area because neither of the line classes reaches a higher probability than the class for flat areas. This error is introduced through the application of the argmax function. The mode of the distribution does not necessarily represent plausible samples.

Our douga dataset is imbalanced, with the most common class being about 8,000 times more common than the rarest. The accuracy is not always the best measure for imbalanced datasets. A classifier that classifies everything as flat areas would achieve an accuracy of 85% while providing no information about a particular input at all. Solutions to this problem aim to change the weighting of the elements when calculating the loss.

One method is to give each class a weight relative to the inverse of its occurrence in

the dataset. In other words: missing elements of a rare class is given a higher loss than missing the same number of elements in a common class. An alternative solution is to discount high-confidence predictions, as is done with focal loss [LGG+18]. While the index over union is a popular measure to compare results, it cannot be used as a loss function directly, as it is not differentiable.

5.5.2 Adversarial Loss

An adversarial model can calculate a loss for the entire image instead of individual pixels. The model is split into two parts: the generator and the discriminator. The generator is given a color frame and suggests a douga that could correspond to the given color image. The discriminator is given douga from the ground truth and douga generated by the generator and is tasked with differentiating between them. Because the discriminator is a differentiable model, its gradient can be used to train the generator to trick the discriminator.

There are multiple different ways of encoding the douga. In the ground truth, each pixel is labeled with exactly one categorical class. The distribution of values is not continuous. During training, the generator's output needs to be continuous, or at least allow calculating a meaningful gradient. Several encodings are examined.

Three-Channel Image Stripping away the semantics of the different colors, douga is just a color image with three channels. GANs have been used successfully for the generation of color images.

Ten-Channel Image Douga can be encoded as an image with one channel for each class. A channel in a pixel is set to the number one if it holds the corresponding class and zero otherwise. Mapping such an image to the corresponding three-channel douga is a linear transformation. To retrieve the three-channel douga, one performs a pixel-wise left multiplication with a matrix containing the class colors as column vectors. A discriminator operating on three-channel images can be turned into one operating on ten-channel images by left multiplying its first linear transformation by said matrix.

Logits of a Categorical Distribution The previous method allows the generator to output any value for any channel. However, it is already known beforehand that real douga in its ten-channel encoding only contains values between zero and one, and the values sum up to one per pixel. The original output of the generator can be constrained to values adhering to these properties. One method of accomplishing this is to interpret the generator's outputs as the logits of a categorical distribution. The generator is extended by the softmax operation.

Argmax A problem with all previous encodings is that the generated douga can be easily identified by its continuous values. Whereas the ground truth only contains values exactly belonging to one class, the generated images may deviate from the exact colors or

contain mixtures of several classes. For example, a value in the generated image may be 1, whereas in the ground truth it is 0. Being continuous, the only way to move the output towards the correct value is to move through the values between them. But all of these are less frequent in the ground truth than either 0 and 1. A range of insurmountable loss values may separate the incorrect value (1) from the correct one (0), preventing gradient descent methods from convergence. When using the ten-channel encoding, this can be mitigated by performing argmax to retrieve a one-hot coding of the most prominent channel of a pixel. The argmax function is not differentiable, but one can approximate its gradient with the gradient of the softmax function.

Additionally to the encoded douga, the discriminator may be given the colored image to predict the conditional probability. Yonetsuiji et al. [Yon17] claim they get better results when not giving the discriminator access to the original input when performing automated coloring.

The 34 layer version of the ResNet model [HZRS16] is used as the discriminator. Both RMSprop and Adam are tried as optimizers, the learning rates $1 \cdot 10^{-4}$, $1 \cdot 10^{-5}$ and $1 \cdot 10^{-6}$ are tried. Both training only with the adversarial loss and training with a linear combination of adversarial and pixel-wise loss are performed. The non-saturating adversarial loss is used for all experiments. Training is attempted on the full images, images scaled down to half the size on both axes, and crops of the original images. Several combinations of these options are run for 4000 epochs. None of these lead to satisfactory results, frequently showing no correlation between the input and the output or being indistinguishable from noise. This may be the result of an insufficient number of iterations, an insufficient number of examples (200 compared to 400 to 1.2 million [IZZE17]), the image resolution being too high (0.5 to 2 million pixels compared to 66 thousand pixels [IZZE17], and 1 million pixels [KLA+20]), too small batch sizes (4 compared to 32 [KLA+20]) or insufficient network capacity for the given task in either the discriminator or the generator or both. Further experiments are deferred to future work. Example results of training with adversarial loss can be seen in Figure 5.7.

5.6 Incorporating Segmentation

The end goal of this work is to produce douga from colored images. While it is possible to train a single model on that task, the number of douga example images available is magnitudes smaller than the datasets that have been used for other image generation tasks. As a remedy, cel segmentation maps have been created. While these contain much less information than douga, they do contain some shared information and incorporating them into the douga generation training might improve douga generation.

Chapter 4 deals with the prediction of cel segmentation maps. The model used for this purpose already contains some of the information useful for douga generation. This section deals with methods of extending the douga generation model of this chapter with the segmentation model to simplify douga generation. The models can be combined to one larger model in multiple ways.

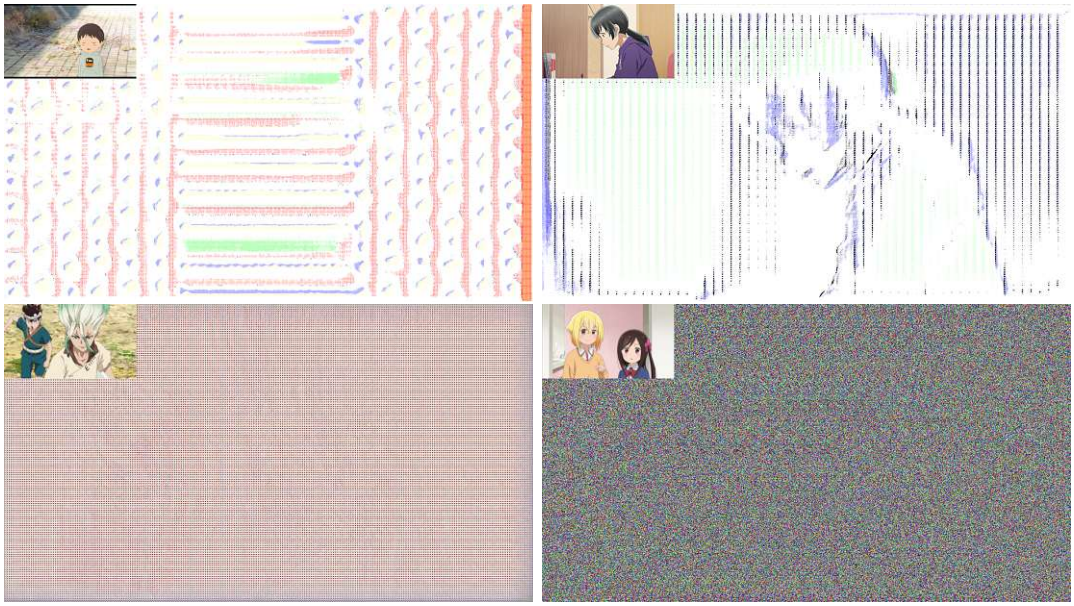


Figure 5.7: All the experiments with the adversarial loss result in images similar to these. Color images by C2C, feel., Studio Chizu and TMS Entertainment.

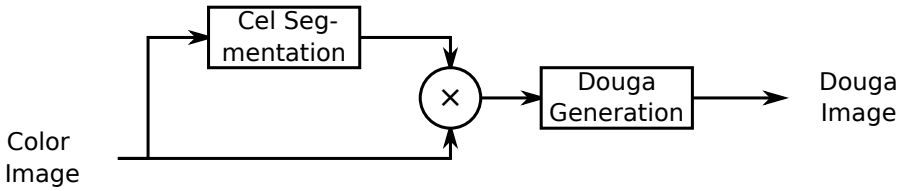
Synthetic Cel Model The segmentations predicted from the segmentation model can be used to remove the backgrounds from the color images, effectively "undoing" the compositing step of production. This allows the douga generation model to operate on approximations of the original cels alone, potentially simplifying the task. However, as the predicted segmentations contain visible errors, it may also make the task more difficult. Douga cannot be generated for parts that were erroneously classified as background and removed. Areas of the cel that are occluded by books (static foreground elements) cannot be reconstructed this way. Image inpainting methods (see [LRS+18]) may be used to reconstruct them, but this is beyond the scope of this work. See Figure 5.8a.

Concatenated Model This is similar to the Synthetic Cel Model. Instead of pre-separating the color image, the color image is extended with an additional channel containing the segmentation information. This allows the douga generation model to deal with errors in the segmentation, as the original image is still fully available. See Figure 5.8b.

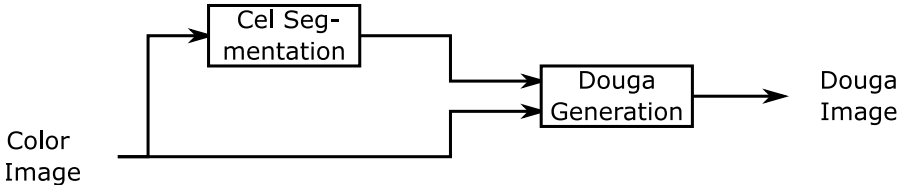
Manual Cel Model Hand-made segmentation maps can be used to first separate the color image into cel and background. This is used as a baseline to evaluate the effect of errors in the segmentation model. See Figure 5.8c.

Color Model The segmentation information can be ignored. Given enough model capacity and training examples, the douga generation model should be able to

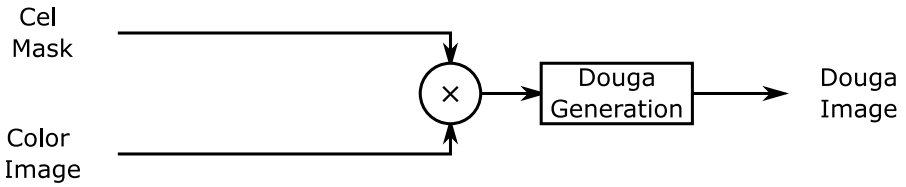
learn to distinguish dynamic and static elements in frames. This is only used as a baseline for comparison. See Figure 5.8d.



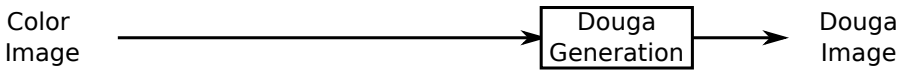
(a) One model to predict the foreground segmentation from color and a second to predict douga from the cel.



(b) One model to predict the foreground segmentation from color and a second to predict douga from the original color and the predicted foreground segmentation.



(c) Manually created cel masks to compare the effect of errors in the cel segmentation model on the douga generation. This is only used for comparison.



(d) One model predicting douga from color. This is used as a baseline.

Figure 5.8: Visualization of different methods of incorporating cel and douga data.

5.7 Post-Processing

Real douga is composed of lines of fixed width, yet our model is not constrained to images with this property. Even results with high per-pixel accuracy can look perceptually unappealing. Noise and uncertainty in the classification within a connected area of the image can lead to these areas being split up into multiple differently classified areas.

Class-weighting causes the model to avoid the misclassification of rare classes (e.g., lines) over the misclassification of frequent classes (e.g., filled areas) leading to an overestimation of the probability of a pixel belonging to a line. The result of this is that lines become thicker than they should be. Post-processing can be used to counter these artifacts.

The line width can be fixed using *skeletonization* [ZS84], a method to identify lines and erode them to a width of 1 pixel. Afterward, dilation can be used to increase the line width to 3 pixels. Skeletonization only operates on binary images. To apply it, the prediction is first reduced to two classes: areas and lines. Afterward, the unprocessed prediction is used to recover the original line classes.

To constrain each area to a single class, connected components are identified in the binary image used for skeletonization. For each detected component, the unprocessed predictions are averaged, and the class with the highest prediction is selected for the entire area. See Figure 5.9 for a visualization of the post-processing steps.

5.8 Evaluation

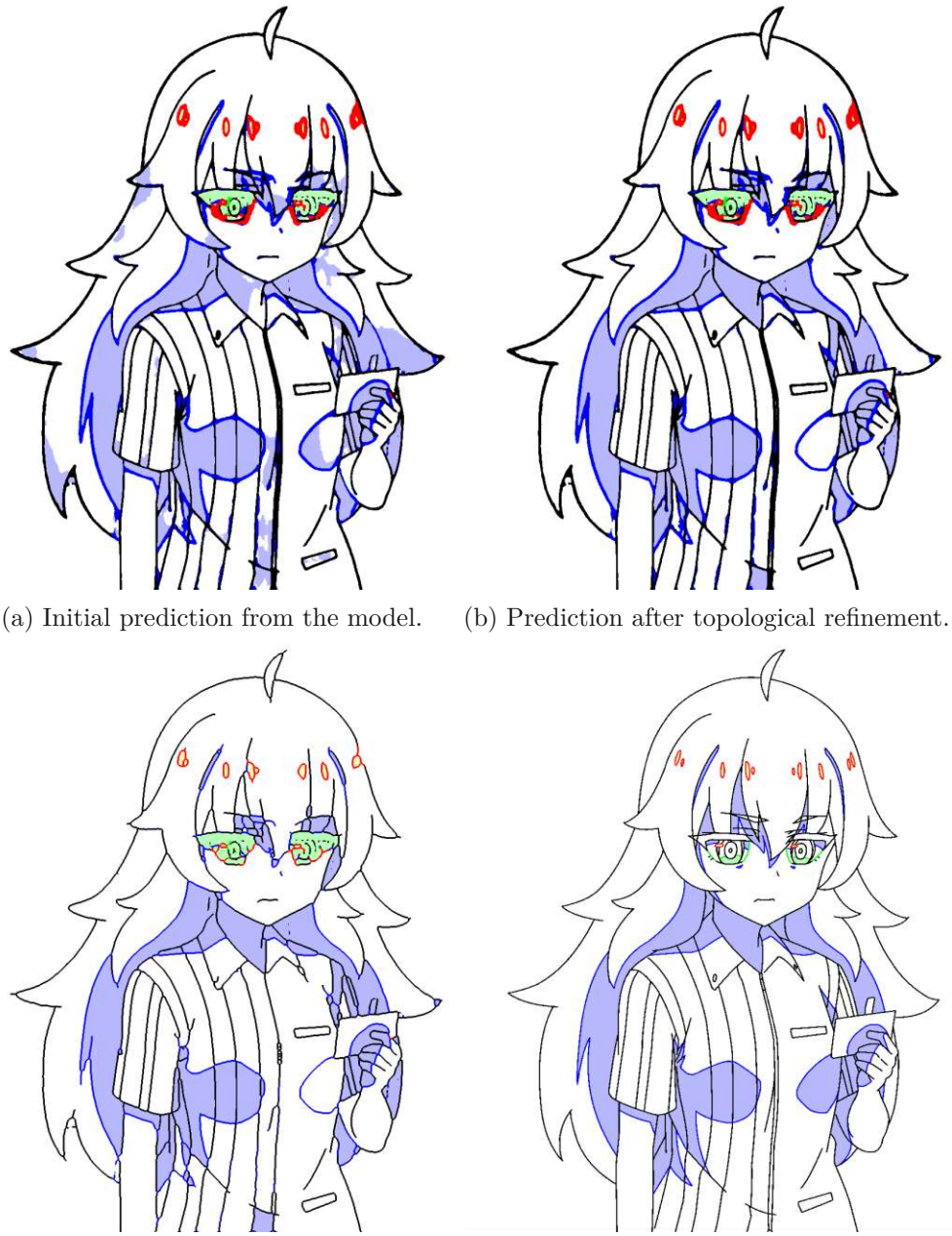
The models described in Section 5.6 are compared using the evaluation images from the dataset described in Section 5.2. These frames were not used during the training of the models. Per-pixel accuracies, intersection over union (IoU), as well as training times are compared. Finally, the per-pixel and per-class accuracies are compared with SketchKeras [Zha17] and Manga Line Extraction [LLW17].

Using the Color Model, a validation accuracy of 94.7% is achieved. The validation accuracy of the Synthetic Cel Model is slightly higher, with 95.4%. This suggests that incorporating the synthetic segmentation data does help with the douga generation, even though this data occasionally fails to remove backgrounds altogether or even removes parts of the image that should have douga. Using the Concatenated Model, the accuracy drops slightly to 95.2%.

New segmentation maps are hand-created for the Douga by the student to evaluate the effect of errors in the predicted cel segmentation maps. The cel segmentation model does not see these. The highest accuracy of 95.7% is achieved with the Manual Cel Model. The douga segmentation is improved by 0.4% when using hand-made segmentations over automatic cel segmentations. Further experiments are required to determine whether this is merely a result of noise. All results can be seen in Table 5.3.

The accuracy and loss values for both training and validation set for the Synthetic Cel Model can be seen in Figure 5.13 and Figure 5.14 respectively.

A confusion matrix for the Synthetic Cel Model is given in Table 5.1. Note that book edges and book fills are ignored as they make up less than 0.1% of the pixels. The recall of underrepresented classes is lower than that for others. The recall for all line classes is worse than their corresponding fill classes. This is a result of using accuracy as the training objective. Other training objectives may be considered in future works.



(a) Initial prediction from the model.

(b) Prediction after topological refinement.

(c) Prediction after skeletonization and topological refinement.

(d) Ground Truth.

Figure 5.9: Prediction for an image with and without connected component averaging on the model trained with weighted cross-entropy. Artwork by Tonari Animation.

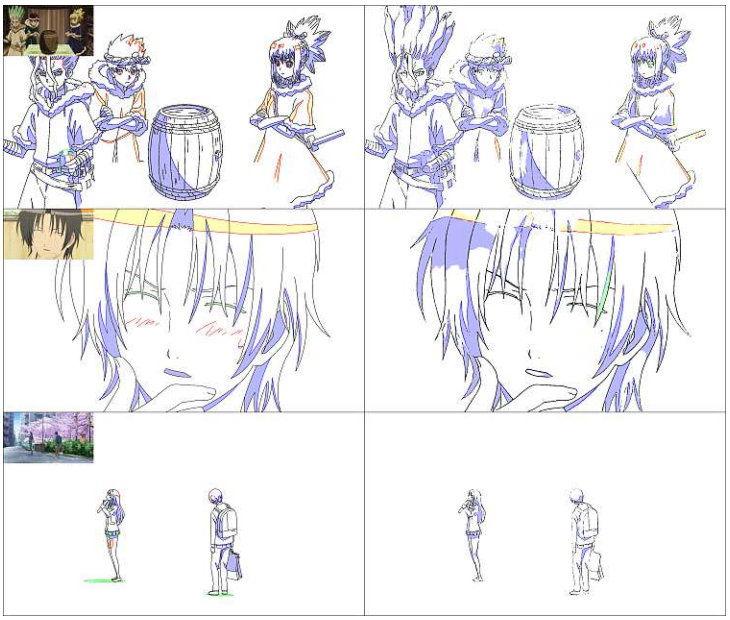
Weighting the classes in the loss calculation by the reciprocal of their frequency decreases the per-pixel accuracy from 95% to 91%. This is expected. One may expect the IoU to increase, as it tends to give more weight to in-frequent classes. However, this is not so. The IoU decreases from 33% to 29%. The lines in the predictions are visibly thicker. Table 5.2 shows the corresponding confusion matrix. For several classes the precision is higher, but the recall is lower.

Detecting connected areas and averaging the classification within them reduces the per-pixel accuracy from 95.4% to 89.4%. For areas that are already mostly labeled correctly, it increases the accuracy by spreading the correct class. On the other hand, it can also spread the incorrect class in partially mislabeled areas, increasing the error. Missing lines in the classification causes unrelated areas to be connected. Using this method with weighted cross entropy increases the accuracy from 89.4% to 91.3%.

Eyes are drawn with multiple differently shaded parts and differ between shows. They tend to be difficult for the model. Frequently they lack most details as in Figure 5.11, or contain incorrect lines.

The validation set created for this work is used for a comparison with SketchKeras [Zha17] and Manga Line Extraction [LLW17]. As these do not differentiate between line and fill classes, all line classes and all fill classes were combined into one. As SketchKeras and Manga Line Extraction output scalar images instead of classifications, a threshold is applied to the result. This threshold is chosen according to the frequencies of the two classes in the validation set. An accuracy of 77.7% and 95.6% is achieved on the validation dataset using SketchKeras and Manga Line Extraction respectively. Confusion matrices for both models are given in Table 5.5. Although Manga Line Extraction has a higher overall accuracy than SketchKeras, its recall for edges is less than half. The example image in Figure 5.12 shows about half the edges are missing in the Manga Line Extraction result.

(a) Good network predictions on the validation set.



(b) Sub-optimal network predictions on the validation set. Some shaded and highlighted areas are assigned incorrectly. Small details are missing.

Figure 5.10: Evaluation of the douga prediction model. Left shows predictions by professional animators and right by the model. Original color images by A-1 Pictures, Daume, Production I.G, TMS Entertainment and ZEXCS.

(a) Crop of a cel.

(b) Artist interpretation of what the douga for the cel might have looked like.

(c) Prediction of the model.

Figure 5.11: In some cases, it is ambiguous whether two adjacent regions should be classified as flat (white) and highlight (yellow) or shaded (blue) and flat (white). More context may be necessary to make this decision accurately. Color image by TMS Entertainment. Douga by Tonari Animation.

True Class \ Prediction	Flat Area	Visible Line	Shading Line	Color Edge	Highlight Line	Shading Fill	Color Fill	Highlight Fill	Recall
Flat Area	**86.6**	0.3	0.1	<0.1	<0.1	1.2	0.1	0.2	97.9
Visible Line	0.6	**1.3**	<0.1	<0.1	<0.1	0.2	<0.1	<0.1	60.4
Shading Line	0.2	<0.1	**0.2**	<0.1	<0.1	0.2	<0.1	<0.1	37.6
Color Edge	0.1	<0.1	<0.1	**<0.1**	<0.1	<0.1	<0.1	<0.1	1.3
Highlight Line	0.1	<0.1	<0.1	<0.1	**<0.1**	<0.1	<0.1	<0.1	16.7
Shading Fill	0.5	0.1	0.1	<0.1	<0.1	**7.1**	<0.1	<0.1	90.4
Color Fill	0.1	<0.1	<0.1	<0.1	<0.1	0.1	**0.1**	<0.1	23.4
Highlight Fill	0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	**0.3**	70.8
Precision	98.1	72.7	53.4	24.6	31.5	80.1	52.1	57.3	

Table 5.1: Confusion matrix of the Synthetic Cel Model on the professional dataset. All values are percentages. The book classes are omitted as they make up less than 0.1% of the pixels.

True Class \ Prediction	Flat Area	Visible Line	Shading Line	Color Edge	Highlight Line	Shading Fill	Color Fill	Highlight Fill	Recall
Flat Area	**82.2**	2.5	1.3	<0.1	0.4	1.8	0.1	0.1	93.0
Visible Line	<0.1	**1.9**	0.1	<0.1	<0.1	<0.1	<0.1	<0.1	90.8
Shading Line	<0.1	0.1	**0.5**	<0.1	<0.1	<0.1	<0.1	<0.1	85.4
Color Edge	<0.1	<0.1	<0.1	**<0.1**	<0.1	<0.1	<0.1	<0.1	7.3
Highlight Line	<0.1	<0.1	<0.1	<0.1	**0.1**	<0.1	<0.1	<0.1	53.4
Shading Fill	0.8	0.5	0.6	<0.1	<0.1	**5.9**	<0.1	<0.1	75.1
Color Fill	0.1	<0.1	<0.1	<0.1	<0.1	0.1	**0.1**	<0.1	26.6
Highlight Fill	0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	**0.2**	58.2
Precision	98.7	37.7	20.2	10.8	15.2	74.8	49.0	66.3	

Table 5.2: Confusion matrix when using weighted cross entropy. All values are percentages. The book classes are omitted as they make up less than 0.1% of the pixels.

Model	Best Train Accuracy	Best Validation Accuracy	Best Train IoU	Best Validation IoU	Time to Best Validation Accuracy
Color Model	**98.0%**	94.7%	**65.4%**	31.0%	59.2h
Synthetic Cel Model	97.5%	**95.4%**	55.3%	**32.6%**	36.1h
Concatenated Model	97.6%	95.3%	47.8%	32.3%	50.3h
Manual Cel Model	98.1%	95.7%	51.2%	32.6%	41.8h

Table 5.3: Comparison of different methods for incorporating segmentation maps into douga generation.

(a) SketchKeras (b) Manga Line Extraction

(c) Ours (d) Ground Truth

Figure 5.12: Comparison of different line segmentation models. Input image and ground truth by Tonari Animation.

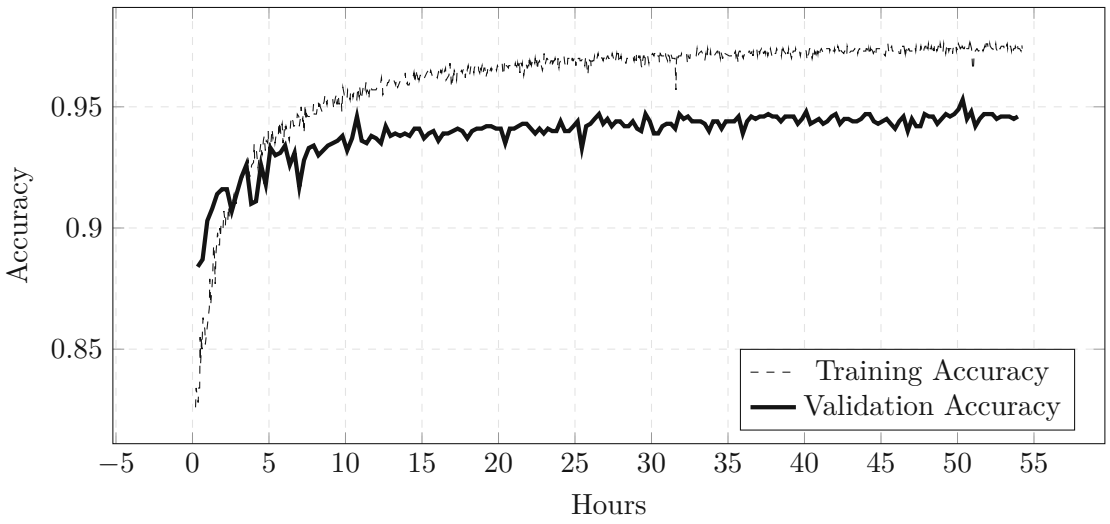


Figure 5.13: Accuracy graph for the training of the Synthetic Cel Model.

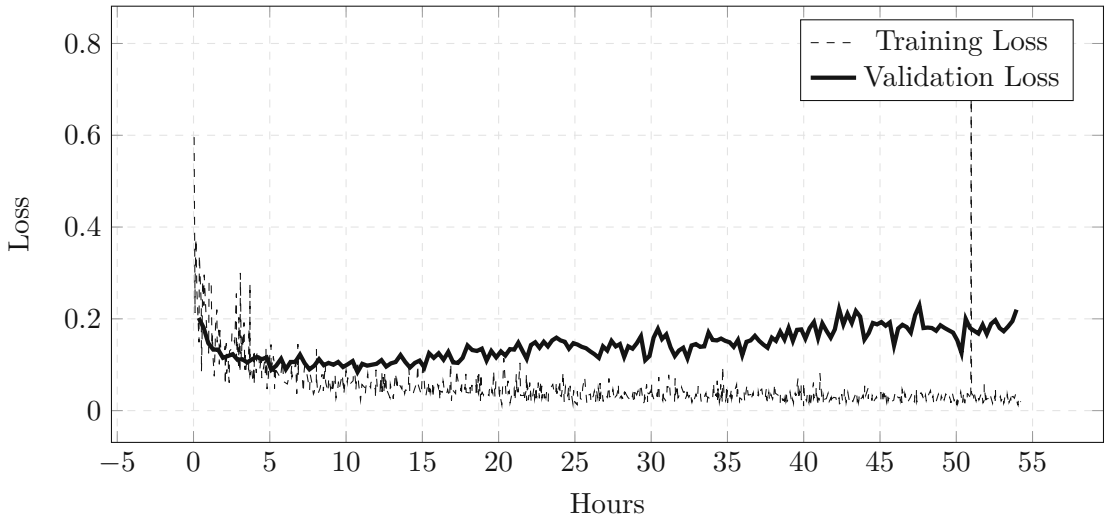


Figure 5.14: Loss graph for the training of the Synthetic Cel Model.

Loss	Best Train Accuracy	Best Validation Accuracy	Best Train IoU	Best Validation IoU	Time to Best Validation Accuracy
Weighted Cross Entropy	94.1%	91.4%	45.0%	30.0%	106.2h
Focal Loss	97.5%	**95.4%**	55.3%	**32.6%**	36.1h

Table 5.4: Comparison of different loss functions for douga generation.

	True Class \ Prediction	Edges	Areas	Recall
SketchKeras	Edges	**2.78%**	0.23%	92.35%
	Areas	22.05%	**74.95%**	77.27%
	Precision	11.20%	99.69%	
Manga Line Extraction	Edges	**0.98%**	2.02%	32.67%
	Areas	2.41%	**94.59%**	97.52%
	Precision	28.90%	97.91%	
Ours	Edges	**2.90%**	0.18%	94.27%
	Areas	5.70%	**91.23%**	94.12%
	Precision	34.72%	99.81%	

Table 5.5: Confusion matrices of SketchKeras [Zha17] and Manga Line Extraction [LLW17] on our douga dataset.

CHAPTER 6

Conclusion and Future Work

In this thesis, we use machine learning to reconstruct douga from color frames of animation. We present a class-based representation of douga, evaluate the suitability of segmentation models for the reconstruction of douga from color frames and propose a method for incorporating cel segmentation maps into the training procedure, which contain less information but are easier to create. Finally we compare our results with two existing line detection models on our dataset.

Two datasets are created and organized: A dataset of segmentation masks and a dataset of douga corresponding to existing color images. These datasets are used to perform cel segmentation on frames of traditional animation and the reproduction of douga, an intermediary product of the production process.

Different methods of augmentation and regularization are compared. Using the segmentation information to randomly replace the backgrounds and perform independent augmentation on both background and cel improves the cel segmentation accuracy (92.1% vs. 90.0%). So does rectangle erasure (92.5% vs. 90.0%). Regularizing the model through dropout improves the validation accuracy (90.6% vs. 90.0%). Increasing the model's size from the ResNet-50 to the ResNet-101 model improves cel segmentation accuracy (91.2% vs. 90.0%).

Different loss metrics are compared. For the douga generation from cels, using focal loss over cross-entropy loss improves the accuracy (90.5% vs. 90.0%). Focal loss was not attempted for the cel segmentation from final frames, as its classes are well balanced.

We find that using predicted segmentation masks improves the douga generation compared to using the douga dataset alone (from 94.7% to 95.4%), even though the generated segmentation masks have visible inaccuracies. The accuracy does not reach the same level as when using manually created segmentation masks (95.7%). The accuracy with predicted segmentation masks is closer to the accuracy with manually created segmentation masks

than they are to the accuracy without any segmentation masks. Further experiments are necessary to determine the significance of these differences.

Edge cases in the visual description of douga are identified. For scenes with multiple shading levels, it is ambiguous whether two adjacent areas should be interpreted as a neutral and a highlighted area or as a shaded and a neutral area. Scene understanding regarding the placement of objects and light sources or knowledge about character designs may be necessary to differentiate between these cases accurately.

The predictions of the model contain visible artefacts, including visibly wider lines and inconsistent classification of connected areas. Using topological analysis of the result and skeletonization, these problems can be alleviated. Although the result may look visibly closer to the ground truth, it does not have a higher per-pixel accuracy.

Our work outperforms both SketchKeras [Zha17] (77.7%) and Manga Line Extraction [LLW17] (95.6%) on our validation dataset. Additionally, our model is capable of differentiating between different line and area classes, whereas the other models are not.

6.1 Future Work

Problems arising from the non-uniqueness of the solutions (see Figure 5.11) may be solvable through generative models, including auto-regressive models like PixelRNN [OKK16] and adversarial models like Pix2Pix [IZZE17], as well as mixture models. Instead of providing a single solution, they can provide a sample of possible solutions. This may be beneficial to both cel segmentation as well as douga generation.

With about 200 images, the douga dataset is comparatively small. To our knowledge, the smallest dataset previously successfully used for image-to-image translation had twice as many images [IZZE17] and many are orders of magnitude larger. Expanding the dataset may help to improve the results.

Increasing the model's size from the ResNet-50 to the ResNet-101 model improves the accuracy for cel segmentation. Increasing it further, both in width and in depth, may improve the accuracy. This may be true for both cel segmentation and douga generation. Due to hardware constraints, no attempts are made on larger models.

Augmentation is used to alleviate the impact of the small size of the dataset. Further methods of augmentation may be explored in the future. Instead of merely erasing rectangles from the images during training, more complicated shapes may be masked [LRS+18]. Further exploration of augmentation with texture overlays may be beneficial.

List of Tables

Index

Bibliography

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[ACM15] Mordvintsev Alexander, Olah Christopher, and Tyka Mike. Inceptionism: Going deeper into neural networks. `https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`, 2015. [Online; accessed 25-February-2021].

[Arm66] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

[BDS18] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[BFC09] Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88 – 97, 2009. Video-based Object and Event Analysis.

[BMR+20] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[Bra19] Gwern Branwen. Danbooru2019: A large-scale crowdsourced and tagged anime illustration dataset. 2019.

[Can86] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[Cel01] Celsys. Clip studio paint, 2001.

[Cel08] Celsys. Retas studio, 2008.

[CLD+20] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2020.

[COR+16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.

[CPSA17] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.

[CRC+20] Mark Chen, Alec Radford, Rewon Child, Jeff Wu, Heewoo Jun, Prafulla Dhariwal, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.

[DKB14] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[DV16] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[EVGW+11] M Everingham, L Van Gool, CKI Williams, J Winn, and A Zisserman. The pascal visual object classes challenge 2012 (voc2012) results (2012). `http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html`, 2011. [Online; accessed 2020-August-19].

[FI+18] Sam Fletcher, Md Zahidul Islam, et al. Comparing sets of patterns with the jaccard index. *Australasian Journal of Information Systems*, 22, 2018.

[FRL+17] William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M Dai, Shakir Mohamed, and Ian Goodfellow. Many paths to equilibrium: GANs do not need to decrease a divergence at every step. *arXiv preprint arXiv:1710.08446*, 2017.

[Fur16] Furansujin Connection. Les étapes de fabrication. `http://www.furansujinconnection.com/les-etapes-de-fabrication/`, 2016. [Online; accessed 2020-August-19].

[GAA+17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein GANs. *Advances in neural information processing systems*, 30:5767–5777, 2017.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[Gra13] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[HGD19] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking ImageNet pre-training. In *Proceedings of the IEEE international conference on computer vision*, pages 4918–4927, 2019.

[HLWK18] Stefan Hinterstoisser, Vincent Lepetit, Paul Wohlhart, and Kurt Konolige. On pre-trained image features and synthetic images for deep learning. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[HRU+17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[IS18] Vladimir Iglovikov and Alexey Shvets. TernausNet: U-Net with VGG11 encoder pre-trained on ImageNet for image segmentation. *arXiv preprint arXiv:1801.05746*, 2018.

[IZZE17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[Jad20] S. Jadon. A survey of loss functions for semantic segmentation. In *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–7, 2020.

[KALL18] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[KD18] Durk P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in neural information processing systems*, pages 10215–10224, 2018.

[KLA+20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[Kum17] Siddharth Krishna Kumar. On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*, 2017.

[KW13] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[LCBB97] Y. Le Cunn, L. Bottou, and Y. Bengio. Reading checks with multilayer graph transforming networks. *Speech and Image Processing Services Research AT&T Lab*, pages 1–4, 1997.

[LGG+18] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[Li10] Xiaoshan Li. Limited animation: History and artistic techniques. In *2010 IEEE 11th International Conference on Computer-Aided Industrial Design & Conceptual Design 1*, volume 1, pages 807–810. IEEE, 2010.

[LLW17] Chengze Li, Xueting Liu, and Tien-Tsin Wong. Deep extraction of manga structural lines. *ACM Transactions on Graphics (SIGGRAPH 2017 issue)*, 36(4):117:1–117:12, July 2017.

[LRS+18] Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 85–100, 2018.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.

[MHLD17] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J. Davison. Scenenet rgb-d: Can 5m synthetic images beat generic ImageNet pre-training on indoor segmentation? In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[MLX+17] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2017.

[Mor15] Tomohiro Morisawa. Managing the unmanageable: Emotional labour and creative hierarchy in the japanese animation industry. *Ethnography*, 16(2):262–284, 2015.

[MSR+19] Hiromichi Masuda, Tadashi Sudo, Kazuo Rikukawa, Yuji Mori, Naofumi Ito, Yasuo Kameyama, and Megumi Onouchi. Anime Industry Report. Technical report, The Association of Japanese Animations, 2019.

[NSF12] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[ODO16] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.

[OKK16] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[Pap20a] Papers With Code. Adam explained. `https://paperswithcode.com/method/adam`, 2020. [Online; accessed 08-November-2020].

[Pap20b] Papers With Code. Relu explained. `https://paperswithcode.com/method/relu`, 2020. [Online; accessed 08-November-2020].

[Pap21a] Papers With Code. Max pooling explained. https://paperswithcode.com/method/max-pooling, 2021. [Online; accessed 08-January-2021].

[Pap21b] Papers With Code. U-Net explained. https://paperswithcode.com/method/u-net, 2021. [Online; accessed 08-January-2021].

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[PLS+18] Kaiyue Pang, Da Li, Jifei Song, Yi-Zhe Song, Tao Xiang, and Timothy M Hospedales. Deep factorised inverse-sketching. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 36–52, 2018.

[PZY+17] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters–improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2017.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation, 2015.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

[RSM+16] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[RVRK16] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 102–118, Cham, 2016. Springer International Publishing.

[SBLL20] R. Sagues-Tanco, L. Benages-Pardo, G. López-Nicolás, and S. Llorente. Fast synthetic dataset for kitchen object segmentation in deep learning. *IEEE Access*, 8:220496–220506, 2020.

[SGS15] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[SGZ+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *arXiv preprint arXiv:1606.03498*, 2016.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[SLX15] Shuran Song, Samuel P. Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015.

[SMH11] Ilya Sutskever, James Martens, and Geoffrey E. Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

[Sob14] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[SPS+18] Jifei Song, Kaiyue Pang, Yi-Zhe Song, Tao Xiang, and Timothy M. Hospedales. Learning to sketch with shortcut cycle consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 801–810, 2018.

[STIM18] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[SZS+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

[tex20] textures.com. Textures for 3D, graphic design and Photoshop! `https://www.textures.com/`, 2020. [Online; accessed 2020-October-15].

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[Whi09] Tony White. *How to make animated films: Tony White's complete masterclass on the traditional principles of animation.* Focal Press, New York; London, England, 2009.

[WU18] Magnus Wrenninge and Jonas Unger. Synscapes: A photorealistic synthetic dataset for street scene parsing, 2018.

[YK16] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions, 2016.

[Yon17] Taizan Yonetsuji. Paints chainer. `github.com/pfnet/Paintschainer`, 2017. [Online; accessed 21-April-2020].

[ZGL+20] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin D. Cubuk, and Quoc V. Le. Rethinking pre-training and self-training. *arXiv preprint arXiv:2006.06882*, 2020.

[Zha17] Lvmin Zhang. sketchkeras. `https://github.com/lllyasviel/sketchKeras`, 2017. [Online; accessed 21-April-2020].

[ZS84] T. Y. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27(3):236–239, March 1984.

[ZZC+20] Zhengli Zhao, Zizhao Zhang, Ting Chen, Sameer Singh, and Han Zhang. Image augmentations for GAN training, 2020.

[ZZK+20] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):13001–13008, Apr. 2020.