ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Matematica

THE TOONOUT MODEL AND MATHEMATICAL THEORY BEHIND IT

Tesi di Laurea in
Probabilistic Methods for Machine Learning

Relatore:
Chiar.mo Prof.
GIOVANNI PAOLINI

Presentata da:
MATTEO MURATORI

Anno Accademico 2024-2025

Questa volta sapevo che
potevo farcela perché
l'avevo già fatto . . .
J.K.ROWLING

Introduzione

La rapida evoluzione che sta avendo l'intelligenza artificiale negli ultimi anni ha portato a nuovi modi di utilizzare questo strumento, che fino a poco tempo fa sembravano inverosimili. In questa tesi ci concentriamo sull'utilizzo del Machine Learning applicato alle immagini e nello specifico alla generazione e alla segmentazione di queste. Nel Capitolo 1 viene trattata una parte di teoria matematica/probabilistica riguardante i *Diffusion Models*, ovvero una classe specifica di modelli facenti parte dello stato dell'arte per la generazione di immagini. Questi modelli si rivelano particolarmente interessanti da un punto di vista teorico, poiché vengono addestrati con l'obiettivo di imparare la distribuzione di probabilità di ottenere una certa immagine ove è stato aggiunto rumore, a partire da un'immagine priva di esso. Dopodiché, questa distribuzione viene invertita in modo che il modello, a partire da un'immagine totalmente rumorosa, sia capace di produrre un'immagine verosimile, assomigliante a quelle su cui viene effettuato l'addestramento. Nel Capitolo 2, ci si sposta sull'argomento della segmentazione binaria di immagini, volta a individuare l'elemento principale rappresentato e a separarlo dallo sfondo. Questo tipo di elaborazione presenta diverse possibili applicazioni tra cui la rimozione dello sfondo al fine di ottenere la sagoma dell'oggetto principale, la realtà aumentata, la guida autonoma, oppure l'individuazione di patologie nelle immagini di organi. Anche questo argomento viene inizialmente trattato da un punto di vista teorico, descrivendo le reti neurali che permettono di mettere in pratica questa segmentazione e parlando della teoria matematica utilizzata per definire le principali metriche che permettono di valutare la qualità dei risultati ottenuti. Infine, ci si sposta su un progetto pratico realizzato in collaborazione con l'azienda *Kartoon*, che utilizza gli strumenti introdotti in maniera teorica nei primi due capitoli. Questo progetto consiste nell'implementazione di un modello open-source per la rimozione degli sfondi dalle immagini rappresentanti personaggi o oggetti in stile anime. Infatti, seppure i modelli esistenti per la rimozione degli sfondi al giorno d'oggi ottengano solitamente risultati strabilianti, questo non vale quando le immagini sottoposte al modello presentano uno stile particolare e sono, di conseguenza, differenti dalla distribuzione di dati utilizzata per addestrare la rete neurale. Per raggiungere l'obiettivo del progetto, abbiamo creato un dataset personalizzato e fatto fine-tuning sul modello open-source esistente *BiRefNet*, al fine di assicurarci che anche le immagini in stile anime facessero parte dei dati su cui la rete neurale è addestrata. Inoltre, abbiamo anche contribuito presentando una metrica implementata da noi per valutare la qualità della rimozione dello sfondo ottenuta, che nei casi su cui lavoriamo fornisce valutazioni più puntuali rispetto alle metriche esistenti. La descrizione completa di questo progetto, nonché i risultati ottenuti, sono presentati nel Capitolo 3.

Contents

Chapter 1

Diffusion Models

The first topic we address is *Diffusion Models*, a class of Machine Learning models suited for image generation. These models obtain new images starting from a total noise image and denoising it step-by-step. We are going to focus on the mathematical theory upon which they rely.

1.1 Introduction

In recent years, the rapid expansion of Machine Learning has led to new ways in which human users can be assisted by computers during their daily life. On this topic, one of the new frontiers that has emerged is the automatic generation of new content. Generative models learn the underlying patterns and structures of their training data and employ them to produce new data based on the input, which often comes in the form of natural language prompts. In this chapter we will discuss image generation and in particular *Diffusion Models*: an interesting category of these models from a theoretical mathematical and probabilistic perspective. This class of models often performs well for image generation task and for this reason has been identified as the state-of-the-art for many applications in these years.

Diffusion Models (sometimes called, more completely, *Denoising Diffusion Probabilistic Models (DDPMs)*) [41, 2] are based on the central idea to

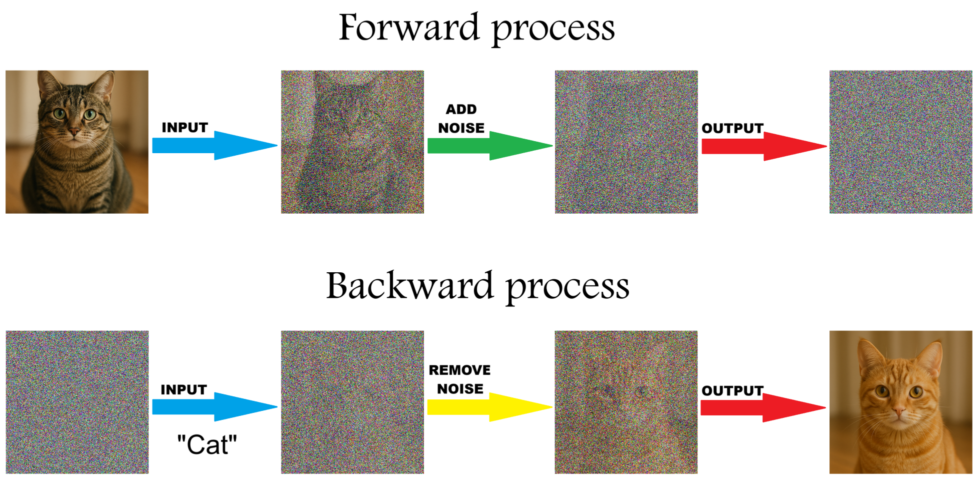

Figure 1.1: Example summarizing how *Diffusion Models* work: during the *forward process* the original image is progressively transformed into a Gaussian noise image; the *backward* process generates a new image starting from a pure noise image and a textual prompt (in this case `cat`), denoising step-by-step.

take each training image and progressively add noise to it, using a multi-step process (*forward process*) to transform it into a sample from a *Gaussian distribution* (an overview about *Gaussian random variables* is provided in Section 1.2.1). A neural network is, subsequently, trained to invert the process: starting from a Gaussian noise image, the quantity of noise portrayed in the image is predicted and then removed step-by-step, until a meaningful image is obtained (*backward process*). We will go in depth with the mathematical details of both the phases in the next sections. *DDPMs* present some important advantages, compared to other image generation techniques (as *Variational Autoencoders* or *Generative Adversarial Networks*): they are easy to train, they work well if distributed to parallel hardware and they avoid instability challenges that often arise using different methodologies, while the quality is at least the same or even better. Conversely, the main limitation of *Diffusion Models* is the high computational cost required, due to the multi-step *forward process*.

1.2 Forward Encoder

1.2.1 Gaussian random variables

As the *forward process* deals with *Gaussian random variables* [27] to represent the conditional distributions of the images after adding the *Gaussian noise*, some theoretical details about these random variables are included in this section. The following formally defines a *Gaussian random variable.*

Definition 1. *Let X be a random variable with values in $\mathbb{R}^d$, $\mu \in \mathbb{R}^d$ and $C \in \mathbb{R}^{d\times d}$ symmetric and positive definite; X is a Gaussian random variable with mean μ and covariance matrix C, and we denote it by $X \sim N(\mu, C)$, if X has density function*

$$\gamma(x) = \frac{1}{\sqrt{(2\pi)^d \det(C)}} e^{-\frac{1}{2}C^{-1}(x-\mu)\cdot(x-\mu)}.$$

This definition can be given in a different equivalent formulation, that allows to extend the class to *positive semi-definite* matrices.

Definition 2. *Let X be a random variable with values in $\mathbb{R}^d$, $\mu \in \mathbb{R}^d$ and $C \in \mathbb{R}^{d\times d}$ symmetric and semi-positively defined; X is a Gaussian random variable with mean μ and covariance matrix C, and we denote $X \sim N(\mu, C)$, if X has characteristic function*

$$\varphi(x) = e^{i\mu\cdot x - \frac{1}{2}Cx\cdot x}.$$

This second formulation allows to easily get a useful result related to the affine transformation of a *Gaussian random variable.*

Proposition 1. *Let $X \sim N(\mu, C)$ be a Gaussian random variable, with $\mu \in \mathbb{R}^d$ and $C \in \mathbb{R}^{d\times d}$ symmetric and semi-positively defined. Consider $A \in \mathbb{R}^{n\times d}$ and $b \in \mathbb{R}^n$; then $AX + b \sim N(A\mu + b, ACA^T)$.*

Proof. Consider φ_{AX+b} the characteristic function of $AX + b$ and φ_X the characteristic function of X. From a result on the characteristic function of an affine transformation of random variables, we have that

$$\begin{aligned}\varphi_{AX+b}(x) &= e^{ix\cdot b}\varphi_X(A^Tx)\\ &= e^{ix\cdot b}e^{i\mu\cdot A^Tx-\frac{1}{2}CA^Tx\cdot A^Tx}\\ &= e^{i(A\mu+b)\cdot x-\frac{1}{2}ACA^Tx\cdot x},\end{aligned}$$

and the statement follows. $\square$

This proposition has as a direct consequence the distribution of the linear combination of two independent *Gaussian random vectors*.

Corollary 1. *Let $X_1 \sim N(\mu_1, C_1)$ and $X_2 \sim N(\mu_2, C_2)$ be two independent Gaussian random variables in $\mathbb{R}^d$, and $\alpha_1, \alpha_2 \in \mathbb{R}$. Then, the linear combination $\alpha_1X_1 + \alpha_2X_2 \sim N(\alpha_1\mu_1 + \alpha_2\mu_2, \alpha_1^2C_1 + \alpha_2^2C_2)$.*

Proof. Denote by φ_1 and φ_2 the characteristic functions of C_1 and C_2, respectively. From a result about the characteristic functions of random vectors with independent components, the vector $\left(\begin{smallmatrix} X_1 & X_2 \end{smallmatrix}\right)$ has characteristic function

$$\begin{aligned}\varphi_{\left(\begin{smallmatrix} X_1 & X_2 \end{smallmatrix}\right)}(x_1, x_2) &= \varphi_{X_1}(x_1)\varphi_{X_2}(x_2)\\ &= e^{i\mu_1\cdot x_1-\frac{1}{2}C_1x_1\cdot x_1}e^{i\mu_2\cdot x_2-\frac{1}{2}C_2x_2\cdot x_2}\\ &= e^{i\left(\begin{smallmatrix} x_1 & x_2 \end{smallmatrix}\right)\left(\begin{smallmatrix} \mu_1 \\ \mu_2 \end{smallmatrix}\right)-\frac{1}{2}\left(\begin{smallmatrix} x_1 & x_2 \end{smallmatrix}\right)\left(\begin{smallmatrix} C_1 & 0 \\ 0 & C_2 \end{smallmatrix}\right)\left(\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right)}.\end{aligned}$$

From Definition 2, it follows that

$$\left(\begin{smallmatrix} X_1 & X_2 \end{smallmatrix}\right) \sim N(\left(\begin{smallmatrix} \mu_1 \\ \mu_2 \end{smallmatrix}\right), \left(\begin{smallmatrix} C_1 & 0 \\ 0 & C_2 \end{smallmatrix}\right)).$$

Let $I \in \mathbb{R}^d$ be the d-dimensional identity matrix; we can write that

$$\alpha_1X_1 + \alpha_2X_2 = \left(\begin{smallmatrix} \alpha_1 I & \alpha_2 I \end{smallmatrix}\right)\left(\begin{smallmatrix} X_1 \\ X_2 \end{smallmatrix}\right)$$

We now apply Proposition 1 to get the final result

$$\begin{aligned}\alpha_1X_1 + \alpha_2X_2 &\sim N(\left(\begin{smallmatrix} \alpha_1 I & \alpha_2 I \end{smallmatrix}\right)\left(\begin{smallmatrix} \mu_1 \\ \mu_2 \end{smallmatrix}\right), \left(\begin{smallmatrix} \alpha_1 I & \alpha_2 I \end{smallmatrix}\right)\left(\begin{smallmatrix} C_1 & 0 \\ 0 & C_2 \end{smallmatrix}\right)\left(\begin{smallmatrix} \alpha_1 I \\ \alpha_2 I \end{smallmatrix}\right))\\ &= N(\alpha_1\mu_1 + \alpha_2\mu_2, \alpha_1^2C_1 + \alpha_2^2C_2).\end{aligned}$$

$\square$

1.2.2 Forward distribution

After this brief overview to underline some properties of the *Gaussian random variables*, we specifically focus on the mathematical theory behind *Diffusion Models*. In the following, we will consider all the images to have the same size, and in particular to be d-dimensional vectors with real values, where $d = h \times w \times c$ is the product between the images height (h), width (w) and number of channels (c), namely the number of values necessary to univocally define the images. Consider $X \in \mathbb{R}^d$ a random image from a training set, and suppose to blend it with a Gaussian noise independent for each pixel $\epsilon_1 \sim N(\mathbf{0}, I)$, where $\mathbf{0}$ stands for the zero vector in $\mathbb{R}^d$ and $I \in \mathbb{R}^{d \times d}$ is the d-dimensional identity matrix. We obtain a noise-corrupted image Z_1 defined in the following way:

$$Z_1 = \sqrt{1-\beta_1}X + \sqrt{\beta_1}\epsilon_1,$$

where $\beta_1 \in (0,1)$ is a coefficient to set the variance. Using the results explained in Section 1.2.1, we are able to explicitly write the conditional distribution of Z_1 given X

$$q(Z_1|X) = N(\sqrt{1-\beta_1}X, \beta_1 I).$$

This process is then repeated to obtain increasingly noisy images $Z_2, Z_3 \dots Z_T$. Analogously, for each $t \in \{2, 3, \dots, T\}$ we set

$$Z_t = \sqrt{1-\beta_t}Z_{t-1} + \sqrt{\beta_t}\epsilon_t,$$

where $\epsilon_t \sim N(\mathbf{0}, I)$ again and the values $\beta_t \in (0,1)$ are empirically chosen and typically set such that $\beta_1 < \beta_2 < \dots < \beta_T$, to ensure that the variance increases throughout the steps. This yields

$$q(Z_t|Z_{t-1}) = N(\sqrt{1-\beta_t}Z_{t-1}, \beta_t I), \tag{1.1}$$

which we refer to as the *forward distribution*, and we notice that the process $\{X, Z_1, \dots, Z_T\}$ can be viewed as a *Markov chain* [47].

1.2.3 Diffusion kernel

After defining the process $\{X, Z_1, \dots, Z_T\}$ and explicitly writing its *forward distribution*, we can delve deeper into some probabilistic properties. By simultaneously exploiting the combination of the *Markov property* and the *chain rule*, for each $t \in \{1, 2, \dots, T\}$ we obtain the conditional joint distribution of the variables $Z_1, Z_2, \dots, Z_t$:

$$q(Z_1, Z_2, \dots, Z_t|X) = q(Z_1|X) \prod_{\tau=2}^{t} q(Z_\tau|Z_{\tau-1}).$$

Computing the marginal distribution of Z_t, we get

$$q(Z_t|X) = N(\sqrt{\alpha_t}X, (1-\alpha_t)I), \tag{1.2}$$

where we denote

$$\alpha_t := \prod_{\tau=1}^{t}(1-\beta_\tau).$$

We call the conditional distribution of Z_t given X the *diffusion kernel.* The choice of the coefficients $\beta_1 < \beta_2 < \cdots < \beta_T$ ensures that

$$\lim_{T\to+\infty} \alpha_T = 0$$

. As a direct consequence, after many steps all the information in the corrupted image is lost and it becomes indistinguishable from Gaussian noise, because in the limit $T \to +\infty$

$$q(Z_T|X) = N(\mathbf{0}, I).$$

Since the right term is independent of X, the conditional distribution of Z_T given X coincides with the distribution of Z_T:

$$q(Z_T) = N(\mathbf{0}, I).$$

1.2.4 Reverse distribution

Given that the goal of *Diffusion Models* is to learn to undo the *forward process*, it is natural to consider the *reverse distribution*, namely the reverse of the *forward distribution.* This density can be expressed using *Bayes' formula*

$$q(Z_{t-1}|Z_t) = \frac{q(Z_t|Z_{t-1})q(Z_{t-1})}{q(Z_t)}. \tag{1.3}$$

The distribution $q(Z_{t-1})$ can be written as

$$q(Z_{t-1}) = \int q(Z_{t-1}|X)q(X)dX.$$

The factor $q(Z_{t-1}|X)$ is acquainted from (1.2), but $q(X)$ is unknown. It could be approximated using a batch of samples from the training set, but the result would be a complicated distribution expressed as a combination of Gaussians. To solve the issue, it is sufficient to consider the conditional version of the *reverse distribution* given X. In total analogy as before, we write

$$q(Z_{t-1}|Z_t, X) = \frac{q(Z_t|Z_{t-1}, X)q(Z_{t-1}|X)}{q(Z_t|X)}, \tag{1.4}$$

and the *Markov property* ensures that

$$\frac{q(Z_t|Z_{t-1}, X)q(Z_{t-1}|X)}{q(Z_t|X)} = \frac{q(Z_t|Z_{t-1})q(Z_{t-1}|X)}{q(Z_t|X)}.$$

As a consequence, the *conditional reverse distribution* $q(Z_{t-1}|Z_t, X)$ can be expressed as the product of the *forward distribution* $q(Z_t|Z_{t-1})$, given by (1.1), and the *diffusion kernel* $q(Z_{t-1}|X)$, given by (1.2). The denominator can be ignored, because it is constant in Z_{t-1}. Since the two factors of the product are *Gaussian densities*, the result is that the *conditional reverse distribution* is Gaussian too, and its mean and variance are identified with the technique of *completing the square*, obtaining the following:

$$q(Z_{t-1}|Z_t, X) = N(m_t(X, Z_t), \sigma_t^2 I), \tag{1.5}$$

where

$$m_t(X, Z_t) = \frac{(1-\alpha_{t-1})\sqrt{1-\beta_t}Z_t + \sqrt{\alpha_{t-1}}\beta_t X}{1-\alpha_t}, \quad \sigma_t^2 = \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}.$$

1.3 Reverse Decoder

1.3.1 Reverse distribution approximation

Since we have established that the *reverse distribution* is intractable, we aim to train a deep neural network to approximate it. In particular we denote by $p(Z_{t-1}|Z_t, w)$ this approximation, where w are learnable parameters. Once the model is trained, we can generate Z_T and transform it to a sample of the distribution $q(X)$, through sequential denoising steps.

If we set the variance $\beta_t \ll 1$, then the distribution $q(Z_{t-1}|Z_t)$ will be approximately a Gaussian over Z_{t-1}. In fact, (1.3) shows that $q(Z_{t-1}|Z_t)$ depends on Z_{t-1} through the product between $q(Z_t|Z_{t-1})$ and $q(Z_{t-1})$. If $q(Z_t|Z_{t-1})$ is a sufficiently narrow Gaussian then all the product mass will be concentrated in one region, which means $q(Z_{t-1}|Z_t)$ is approximately a Gaussian. The reverse distribution can therefore be modeled as

$$p(Z_{t-1}|Z_t, w) = N(\mu(Z_t, w, t), \beta_t I), \tag{1.6}$$

where $\mu(Z_t, w, t)$ is a deep neural network with learnable parameters w. A common architecture choice for μ is the *U-net*, frequently used for image processing. The main limitation of setting a low variance is that the denoising process will require a larger number of steps to achieve the totally denoised image. The reverse denoising process $\{Z_T, Z_{T-1}, \dots, X\}$ takes the form of a *Markov chain*, and applying the *Markov property* after the *chain rule*, the joint distribution is the product of the conditional marginals:

$$p(X, Z_1, \dots, Z_T|w) = p(Z_T|w)\left(\prod_{t=2}^{T} p(Z_{t-1}|Z_t, w)\right) p(X|Z_1, w),$$

where $p(Z_T|w)$ is assumed to be $N(\mathbf{0}, I)$ for any choice of w, and therefore it can be more simply denoted by $p(Z_T)$.

1.3.2 Objective function

To train the neural network μ, an objective function is required. The obvious choice would be the *likelihood function* which, given a training sample

X, is computed by

$$p(X|w) = \int \cdots \int p(X, Z_1, \ldots, Z_T|w)\, dZ_1 \ldots\, dZ_T.$$

These integrals are intractable because they require integrating over the complex neural network function. To solve the issue, instead of maximizing the likelihood, we aim to maximize a lower bound of the *log likelihood function*. To do this we require the definition of a measure of the similarity between two density functions: the *Kullback-Leibler divergence*.

Definition 3. *Consider f and g two density functions. The Kullback-Leibler divergence between f and g is defined as*

$$KL(f(x)||g(x)) = -\int f(x) \ln\left(\frac{g(x)}{f(x)}\right)\, dx.$$

Denoting $Z := (Z_1, \ldots, Z_T)$ and applying the *chain rule* to get $p(X, Z|w) = p(X|w)p(Z|X, w)$, the following computation shows that the *log likelihood* can be written as the sum of two different integral terms:

$$\begin{aligned}\ln p(X|w) &= \int q(Z|X) \ln p(X|w)\, dZ \\ &= \int q(Z|X)[\ln\left(\frac{p(X|w)p(Z|X,w)}{q(Z|X)}\right) - \ln\left(\frac{p(Z|X,w)}{q(Z|X)}\right)]\, dZ \\ &= L(w) + KL(q(Z)||p(Z|X,w)),\end{aligned}$$

where

$$L(w) = \int q(Z|X) \ln\left(\frac{p(X,Z|w)}{q(Z|X)}\right)\, dZ.$$

From the property $KL(\cdot||\cdot) \geq 0$, we get $\ln p(X|w) \geq L(w)$ and for this reason $L(w)$ is called *Evidence Lower Bound (ELBO)*. This is the term maximized during the neural network training process. The *ELBO* term can be rewritten in a different way. Introducing the notation

$$\mathbb{E}_{Z|X}[\,\cdot\,] = \int q(Z_1|X) \prod_{t=2}^{T} q(Z_t|Z_{t-1})[\,\cdot\,]\, dZ,$$

we find that

$$\begin{aligned} L(w) &= \mathbb{E}_{Z|X}\left[\ln \frac{p(Z_T)\big(\prod_{t=2}^{T} p(Z_{t-1}|Z_t,w)\big)p(X|Z_1,w)}{q(Z_1|X)\prod_{t=2}^{T} q(Z_t|Z_{t-1},X)}\right] \\ &= \mathbb{E}_{Z|X}\left[\ln p(Z_T) + \sum_{t=2}^{T}\ln\left(\frac{p(Z_{t-1}|Z_t,w)}{q(Z_t|Z_{t-1},X)}\right) - \ln q(Z_1|X) + \ln p(X|Z_1,w)\right]. \end{aligned}$$

The first and the third term are independent of w and therefore they can be ignored. The fourth term can be estimated using *Monte Carlo* method, sampling $Z_1^{(l)} \sim N(\sqrt{1-\beta_1}X, \beta_1 I)$ a sufficiently high number L of times, and considering that

$$\mathbb{E}_{Z|X}[p(X|Z_1,w)] \simeq \frac{1}{L}\sum_{l=1}^{L}\ln p(X|Z_1^{(l)},w).$$

While dealing with the second term an issue arises. Theoretically, it could be evaluated with *Monte Carlo* in the same way as we used for the fourth term, but since pairs of sampled values Z_{t-1} and Z_t would be required, the estimation variance would be extremely large, and as a consequence a too high number of samples would be necessary. To bypass the issue, we expand the second term using (1.4) as it follows:

$$\ln\frac{p(Z_{t-1}|Z_t,w)}{q(Z_t|Z_{t-1},X)} = \ln\frac{p(Z_{t-1}|Z_t,w)}{q(Z_{t-1}|Z_t,X)} + \ln\frac{q(Z_{t-1}|X)}{q(Z_t|X)}.$$

Ignoring, once again, the term independent of w, we obtain the following reformulation of the *ELBO*:

$$L(w) = \mathbb{E}_{Z|X}\left[\sum_{t=2}^{T}\ln\frac{p(Z_{t-1}|Z_t,w)}{q(Z_{t-1}|Z_t,X)} + \ln p(X|Z_1,w)\right],$$

and integrating out all the terms independent from the integration variable, it can be rewritten as

$$\begin{aligned} L(w) = &\int q(Z_1|X)\ln p(X|Z_1,w)\,dZ_1 - \\ &-\int \sum_{t=2}^{T} KL(q(Z_{t-1}|Z_t,X)||p(Z_{t-1}|Z_t,w))q(Z_t|X)\,dZ_t. \end{aligned}$$

The two integral terms are named, respectively, *reconstruction term* and *consistency term.* The latter term involves the *Kullback-Leibler divergence* of two Gaussian distributions, therefore it can be explicitly determined in closed form, and the result is

$$KL(q(Z_{t-1}|Z_t, X)||p(Z_{t-1}|Z_t, w)) = \frac{1}{2\beta_t}||m_t(X, Z_t) - \mu(Z_t, w, t)||^2 + \text{const}, \tag{1.7}$$

where in the constant term are contained all the terms independent of the weights. At this point, to compute the integral of the *Kullback-Leibler divergence*, it is sufficient to apply *Monte Carlo* once again, sampling Z_t from the distribution given by (1.2).

1.3.3 Noise prediction

A more convenient technique to employ diffusion consists of predicting the noise added to the original image at each step, instead of the denoised image. Exploiting the *diffusion kernel* (1.2), we find that X can be expressed in the following way:

$$X = \frac{1}{\sqrt{\alpha_t}} Z_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}} \bar{\epsilon}_t,$$

with $\bar{\epsilon}_t \sim N(\mathbf{0}, I)$ representing the total noise added step-by-step to the original image X to find Z_t. Substituting the last expression into the definition of $m_t(X, Z_t)$, we get

$$m_t(X, Z_t) = \frac{1}{\sqrt{1-\beta_t}} \left(Z_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} \bar{\epsilon}_t \right).$$

Instead of using a neural network $\mu(Z_t, w, t)$ to predict the image at the previous step, we exploit a new neural network $g(Z_t, w, t)$ that has the purpose to predict the total noise added to the image X to generate the image Z_t. In this case, substituting in the last equality the sampled noise $\bar{\epsilon}_t$ with the predicted noise $g(Z_t, w, t)$ and considering (1.5), the result of the right side can be viewed as a prediction of the previous-step image Z_{t-1}. Therefore, the two neural networks are related by the equality

$$\mu_t(Z_t, w, t) = \frac{1}{\sqrt{1-\beta_t}} \left(Z_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} g(Z_t, w, t) \right).$$

Substituting in (1.7) $m_t(X, Z_t)$ and $\mu(Z_t, w, t)$ with the two expressions just mentioned, we can write the *Kullback-Leibler* divergence between that two distributions in a new way:

$$\begin{aligned} &KL(q(Z_{t-1}|Z_t, X)||p(Z_{t-1}|Z_t, w)) = \\ =&\frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}||g(Z_t, w, t) - \bar{\epsilon}_t||^2 + \text{const} \\ =&\frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}||g(\sqrt{\alpha_t}X + \sqrt{1-\alpha_t}\bar{\epsilon}_t, w, t) - \bar{\epsilon}_t||^2 + \text{const}. \end{aligned} \tag{1.8}$$

Considering (1.6) with $t = 1$ and computing the logarithm, the integrand of the *reconstruction term* becomes

$$\ln p(X|Z_1, w) = -\frac{1}{2\beta_1}||X - \mu(Z_1, w, 1)||^2 + \text{const}.$$

Leveraging the relation between μ and g and the relation between Z_1 and X given by (1.1) the latter can be written

$$\ln p(X|Z_1, w) = -\frac{1}{2(1-\beta_1)}||g(Z_1, w, 1) - \epsilon_1||^2 + \text{const},$$

which exactly coincides with the expressions of the *consistency term*. Empirical experiments showed that performance increases if the factor $\beta_t/2(1-\alpha_t)(1-\beta_t)$ is omitted. All considered, we can write the *ELBO* objective function in the following final form:

$$L(w) = -\sum_{t=1}^{T} ||g(\sqrt{\alpha_t}X + \sqrt{1-\alpha_t}\bar{\epsilon}_t, w, t) - \bar{\epsilon}_t||^2. \tag{1.9}$$

This function represents the difference between the actual noise $\bar{\epsilon}_t$ and the predicted noise g. To evaluate the *ELBO*, for each training point X, we sample $\bar{\epsilon}_t$ and add it to the original image to get a sample of the noisy image. Then it becomes the input of the network g to return a prediction of that noise. To maximize the function L with stochastic gradient descent, for each dataset image X the gradient of L is computed only for a random step t, rather than for each step. All this procedure automatically provides data augmentation, because to each image at each training step is applied a new sampled noise.

1.3.4 Samples generation

Once the training process is completed, we can generate new images as we aimed. This procedure starts by sampling a pure noise image Z_T from the distribution $p(Z_T)$, that is then denoised step-by-step. For each step, the image Z_t is given as input to the network, to compute the prediction $g(Z_t, w, t)$. From this prediction, it is possible to evaluate $\mu(Z_t, w, t)$ because of the relation between the two networks explained in the last section. Eventually, a sample of the predicted image is generated, using the modeled distribution in (1.6). In particular, sampling $\epsilon \sim N(\mathbf{0}, I)$, we get

$$Z_{t-1} = \mu(Z_t, w, t) + \sqrt{\beta_t}\epsilon.$$

In the last step, after computing $\mu(Z_1, w, 1)$, this value is taken directly as the prediction of X without adding any noise, because the final output should be noise-free.

1.4 Guided diffusion

1.4.1 Classifier guidance

So far, we have discussed *Diffusion Models* as a way to generate new samples from a distribution $p(X|w)$ learned using training samples $X_1, X_2, \dots, X_N$. In real applications, it often happens that a generic sample of X is useless, because the user requires specific choices for the content to generate. To address this issue, guidance is introduced into the *Diffusion Models*. In particular, the distribution learned in this case is $p(X|C, w)$, where C is the *conditioning variable*, namely a class label or a textual description of the desired content of the final image.

The most natural approach to achieve guidance, would be to add the parameter C as an additional input for the neural network, considering $g(Z, C, w, t)$ and training the model using pairs (X_n, C_n). This approach has the limitation that it is impossible to control how much weight the neural network gives to the guidance. An alternative way to solve the issue is termed *classifier guidance*. To explain the idea behind this approach we need to mention a general property of the distributions.

Proposition 2. *A probability density function in $\mathbb{R}^n$ can be univocally determined by the gradient of its logarithm.*

Proof. Consider two generic probability density functions $f(x)$ and $g(x)$ and suppose that

$$\nabla \ln f(x) = \nabla \ln g(x).$$

By the definition of the density functions it is know that

$$\int f(x)\, dx = \int g(x)\, dx = 1.$$

Using a basic property of the gradient and, subsequently, computing the exponential of both terms we obtain

$$\nabla \ln f(x) = \nabla \ln g(x) \iff \ln f(x) = \ln g(x) + c \iff f(x) = kg(x),$$

where k and c are constants. At this point we conclude using the fact that

$$\int kg(x)\, dx = \int f(x)\, dx = 1 = \int g(x)\, dx \implies k = 1 \implies f(x) = g(x).$$

$\square$

Now, suppose a classifier $p(C|X)$ is trained. Following the statement of the proposition, we can work with the gradient of the logarithm of the distributions. Consider that

$$\begin{aligned} \nabla_X \ln p(X|C) &= \nabla_X \ln \frac{p(C|X)p(X)}{p(C)} \\ &= \nabla_X \ln p(X) + \nabla_X \ln p(C|X). \end{aligned} \tag{1.10}$$

In the last equality we omitted the term $\nabla_X \ln p(C)$, because $\ln p(C)$ is independent of X and therefore its gradient is equal to zero. The weight of the classifier term in influencing this computation can be controlled by adding a hyperparameter λ called the *guidance scale*. If the network is trained to learn the function

$$s(X, C, \lambda) = \nabla_X \ln p(X) + \lambda \nabla_X \ln p(C|X),$$

called *score function*, it learns a model of the distribution $p(X|C)$, which is what is needed to sample new images with the guidance. For $\lambda \gg 1$ the model is encouraged to follow the conditioning label, but this penalizes sample diversity because the model prefers samples where the classifier works well.

This *classifier guidance* comes with several limitations. First, a classifier must be separately trained, leading to high computational costs; moreover, this classifier needs to work well with noisy images, and not solely with clean images. For this reason, standard classifiers are not well-suited for this task.

1.4.2 Classifier-free guidance

There exists another guidance approach, which does not require a trained classifier. If we substitute the term $\nabla_X \ln p(C|X)$ in the *score function* formula using (1.10), we get the following expression:

$$s(X, C, \lambda) = (1 - \lambda)\nabla_X \ln p(X) + \lambda \nabla_X \ln p(X|C).$$

If we choose $\lambda < 1$ the network actively penalizes samples that do not care about the conditioning information. Furthermore, $p(X|C)$ and $p(X)$ can be modeled using the same neural network: during the training, for a random subset of the training points ($\sim 20\%$ of the total), the conditioning variable C is ignored, and these samples are used to approximate the unconditional $p(X)$ instead of $p(X|C)$. This *classifier-free guidance* approach solves the issues of the *classifier guidance*, and generally performs better.

1.4.3 Tasks

The guidance with the diffusion model can be put into practice in some different ways, depending on the task. In the simplest case, the conditioning variable is a label chosen from a discrete finite number of possible classes. However, this approach is not sufficient when we want to give the freedom to the user to choose the conditioning input not only between predefined instances. Examples of tasks where this freedom is needed are *prompt-to-image* generation, where the user types a textual prompt describing what should be represented in the image, and *image-to-image* generation, where the input is an image related in any way to the image to be generated. An example of an important application of *image-to-image* generation is super-resolution, where a low-resolution image is given to the model to generate the same image with higher resolution. Other examples are inpainting, style transfer, colourization, video generation, etc.

Prompt-to-image guidance requires incorporating large language models techniques to encode the input prompt and use it to influence the generation. Its internal representation is concatenated with the input to the denoising network, and cross-attention mechanisms are employed to attend to the representation of the text sequence. An important example of *prompt-to-image diffusion model* is *Stable Diffusion XL* [28], which is a classifier-free guidance model. We used *Stable Diffusion XL* to generate images for our background removal project, described in Chapter 3.

Chapter 2

Dichotomous Image Segmentation

After studying in Chapter 1 the theory behind *Diffusion Models*, a computer vision task that can be performed on generated images is presented. The topic of this chapter is *Dichotomous Image Segmentation*: a task that aims to accurately segment detailed objects from natural images.

2.1 Task presentation

In the past few years, Artificial Intelligence development led to improvements in existing tasks and created new ones. Nowadays, one of the AI's most useful and studied applications is computer vision. This Machine Learning field demands highly accurate outputs to endorse intensive human-machine interaction and realistic virtual experiences. Image segmentation is the task of highlighting pixels of the exact area where each element represented in the image is located. Segmentation plays a fundamental role in enabling computers to perceive and understand the environment and the objects represented in analyzed images. As can be observed in Figure 2.1, compared to *Image Classification* [17] and *Object Detection* [37], it provide more accurate descriptions of the displayed items. This advantage is exploited in a

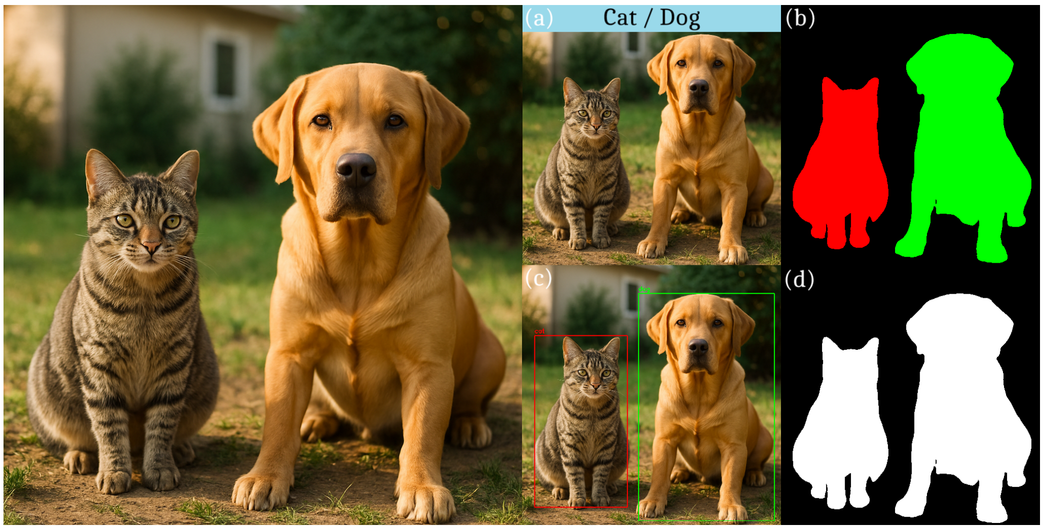

Figure 2.1: Comparison among computer vision tasks: the left-half picture is the original image, at the right we have outputs produced after applying: *Image Classification* (a), *Multi-class Segmentation* (b), *Object Detection* (c), *Dichotomous Image Segmentation* (d).

wide range of technologies, such as autonomous driving, augmented reality, image processing, etc. These applications can be distinguished in two different classes, based on how strongly they affect the real world. The "light" applications are relatively tolerant of the segmentation inaccuracies and failures because these issues only lead to more adjustment work and time costs. While, in the "heavy" applications, those defects are more likely to cause serious consequences such as damage to objects requiring huge repair costs, or injuries sometimes fatal for humans or animals. Logically, this second class of applications requires very accurate and strong models. Currently, most segmentation models are still less used in those "heavy" applications because their reliability is not above the right threshold yet. This fact blocks the segmentation techniques from playing fundamental roles in broader applications.

The goal of addressing these two mentioned classes in a general framework is achieved through *Dichotomous Image Segmentation (DIS)*, a task which aims to precisely segment detailed objects from the provided images. An

RGB image is given as input to the model, and the output consists of a single channel image having the same resolution as the input one. The output pixels values are included between 0 and 255 and assigned by the model according to the following criteria: each pixel is assigned the white color (value 0) if considered part of the main foreground object(s) of the image, while it is assigned the black color (value 255) if considered part of the rest of the image. Intermediate values are also possible, for example to provide a pixel gradient around the foreground item(s) edge or to characterize transparent segmented objects (such as glasses, ice, etc.).

Dichotomous Image Segmentation can also operate as a smart strategy for implementing *Multi-class Segmentation* [21]. This task aims to label all the pixels in an image, sometimes representing a complex scenario or containing many different objects. As a standard technique, the pre-defined multiple categories are encoded as one-hot vectors. Nonetheless, the one-hot representation of the classes becomes memory-prohibitive when the number of categories is huge, and this holds especially for high-resolution images. This is a waste of memory capacity, as some input images only contain objects from a few categories. A hack to this issue consists of a two-step solution using *Object Detection* combined with *Dichotomous Image Segmentation*: a bounding box and category of a certain object can be predicted first. The segmentation process can then be conducted in a dichotomous way within the bounding box region by producing a single channel probability map.

In the following, some baseline architectures used for image processing are presented. Although designed for image tasks, these models are still insufficient to perform well with *Dichotomous Image Segmentation.* After that, the state-of-the-art approaches that rely on these baselines to build deep segmentation models are analyzed in depth. For each of them, a description of the corresponding model architecture is provided, and their advantages are reported.

2.2 Baseline architectures

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [25] is a type of neural network used to solve difficult image-driven pattern recognition tasks that learns features via filter (or *kernel*) optimization. *CNNs* are comprised of three types of layers. These are *convolutional layers, pooling layers* and *fully-connected layers*. When these layers are stacked, a *CNN* architecture is formed.

In the *convolutional layers*, the parameters are the values of some learnable *kernels*. These *kernels* are usually small in spatial dimensionality, but spreads along the entirety of the depth of the input. When the data hits a *convolutional layer*, each filter is convolved across the spatial dimensionality of the input. In this way, each *kernel* produce a 2D *activation map* for each channel of the input. The *activation maps* corresponding to each image channel are then summed together and the result is an output with a certain spatial dimension and unitary depth. Stacking the *activation maps* for all filters along the depth dimension forms the full output volume of the *convolutional layer*. During the 2D convolution between a certain channel of an image and the corresponding channel of a *kernel*, the latter is aligned at the top-left of the image and the element-wise product between the *kernel* parameters and the image channel values is computed. The results are summed together to obtain the first pixel of the produced *activation map*. After that, the kernel slides over the image to compute the other activation map values.

Pooling layers aim to gradually reduce the dimensionality of the representation, and thus further reduce the number of parameters and the computational complexity of the model. These layers take an activation map as input and scale its dimensionalities, often applying *maximum* or *average* functions through 2×2 *kernels* with stride 2, which means that the *kernel* will skip one pixel for every sliding over the image. In this case, the activation map will be scaled to the 25% of the original size, while the depth will not vary.

After several *convolutional* and *pooling layers*, the final classification is done via *fully connected layers*. Here, neurons have connections to all activations in the previous layer. Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset (addition of a learned or fixed bias vector).

Convolutional Neural Networks are designed to emulate the behavior of a visual cortex. Cortical neurons respond to stimuli only in a certain area of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. In *CNNs* stacking many layers leads to nonlinear filters that become increasingly responsive to a larger region of pixel space so that the network first creates representations of small parts of the input, then from them assembles representations of larger areas.

2.2.2 U-Net

The typical use of *Convolutional Neural Networks* is on classification tasks, where the output to an image is a single class label. However, *CNNs* usage can be extended to more tasks, for example segmentation, where the output is a label for each pixel of the input. To achieve this, we have to consider a network with a structure different from the typical one.

U-Net [39] is an architecture where a usual convolutional structure to contract the input is supplemented by upsampling operators to extend the feature maps to the original image size. Hence, these layers increase the resolution of the output. High resolution features from the contracting path are combined with the upsampled output. A successive convolution layer can then learn to assemble a more precise output based on this information. In the upsampling part we have also a large number of feature channels, which allow the network to propagate context information to higher resolution layers. The expansive path is symmetric to the contracting path, and yields a U-shaped architecture. The network does not have any fully connected layers. The contracting path follows the typical architecture of *CNNs*. At each downsampling step, it consists of the application of two 3×3 convolutions one after the other, each followed by a *Rectified Linear Unit (ReLU)* activation and a 2×2 max pooling operation with stride 2 for downsampling. The number of feature channels is doubled at every step. The expansive path consists of the repetition at each level of the feature map upsampling followed by a 2×2 convolution that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3×3 convolutions, each followed by a *ReLU*. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1×1 convolution is applied to map each 64-component feature vector to the desired number of classes. In total the *U-Net* has 23 convolutional layers.

2.2.3 U^2-Net

U^2-Net [36] is a two-level nested U-structure. On the bottom level, it is constituted by a *ReSidual U-block (RSU)*, which is able to extract multi-scale features without impacting the resolution; on the top level, there is a U-Net like structure, where each stage is filled by an *RSU* block. *RSU* block consists of three components:

1. a convolutional input layer, which transforms the feature map given as input x having height H, width W and number of channels C_{in}, to an intermediate map $F_1(x)$ with C_{out} channels. This component is responsible for local feature extraction.

2. a U-Net like symmetric encoder-decoder structure with height L, taking the feature map returned as output of the input layer $F_1(x)$ and extracting and encoding the multi-scale contextual information $U(F_1(x))$. Larger L means more pooling operations, larger receptive fields and more detailed local and global features. The multi-scale features are extracted from downsampled feature maps and encoded into high resolution feature maps by progressive upsampling, concatenating and

convoluting. This process avoids the loss of fine details that typically occurs while directly upsampling with large scales.

3. a residual connection which combines local and multi-scale features by the summation: $F_1(x) + U(F_1(x))$.

This U^2-*Net* design allows the network to extract features from multiple scales directly from each residual block. Thanks to the U-structure, the amount of computation is reasonable because most operations are calculated on the downsampled feature maps.

2.3 State-of-the-art approaches

2.3.1 Intermediate supervision

In general, deep segmentation models can fit the training sets very well, but not reaching good enough performance on the test sets. This is explained by two main reasons. On one hand, the "distributions" of the training, validation, and testing sets can differ substantially, leading to performance degradation on the test sets. On the other hand, having the required strong capabilities of feature representations to ensure sufficient performances means more accurately fitting the training set, and as a consequence a higher probability of overfitting. For this reason, stronger representative capabilities and lower risks of overfitting are usually in conflict with each other and models are often struggling to find the right balance between them. To alleviate overfitting, the *dense supervision* (or *deep supervision*) [18] is one of the most common techniques, which imposes ground truth supervisions on the outputs of the deep intermediate layers. On the cons, transforming the deep intermediate features (multi-channel) into an output (single-channel) in *Dichotomous Image Segmentation* is a dimension reduction operation, and this avoids improving significantly the performance.

An approach capable to solve this issue is *intermediate supervision* [35]. It follows the dense supervision idea but develops a simple yet more effective supervision strategy to directly enforce the supervisions on high-dimensional intermediate deep features in addition to the side outputs. On this topic, we present *IS-Net*, an architecture developed to perform *DIS* using *intermediate supervision*. This network consists of a *ground truth encoder* and an *image segmentation component*. The *ground truth encoder* is designed to encode the GT masks into high-dimensional spaces and then used to provide *intermediate supervision* on the *image segmentation component*. Meanwhile, this second component is expected to be capable of capturing fine structures in large size inputs with affordable memory and time costs. This architecture is schematized in Figure 2.2.

Let $I \in \mathbb{R}^{H\times W\times 3}$ and $G \in \mathbb{R}^{H\times W}$ denote respectively the input image and the corresponding ground truth mask, F the *IS-Net* model, θ the learnable parameters and D the number of intermediate feature maps we want to provide. The first training is done on the *ground truth encoder* in a self-supervised way. This process corresponds to solve the minimum problem:

$$\arg\min_{\theta_{gt}} \sum_{d=1}^{D} BCE(F_{gt}(\theta_{gt}, G)_d, G),$$

where BCE is the *Binary Cross-Entropy loss* (which is explained in details in Section 2.4), F_{gt} stands for the GT encoder model and θ_{gt} for the GT encoder learnable weights. After this first phase, θ_{gt} are frozen for generating the high-dimensional intermediate deep features used as intermediate supervision, which are denoted by $\{f_d^G\}_{d=1}^D$ where $f_d^G := F_{gt}^-(\theta_{gt}, G)_d$ and F_{gt}^- stands for F_{gt} without the last convolution layers used for generating the probability maps. Analogously, let us consider F_{sg} as the segmentation model, θ_{sg} its learnable weights, F_{sg}^- the same model without the final convolution layers, and $\{f_d^I\}_{d=1}^6$, where $f_d^I := F_{sg}^-(\theta_{sg}, I)_d$. The intermediate supervision is performed using the following loss:

$$L_{fs} = \sum_{d=1}^{D} \lambda_d^{fs} \|f_d^I - f_d^G\|^2,$$

where λ_d^{fs} are the weights of each loss component. Moreover, we have the

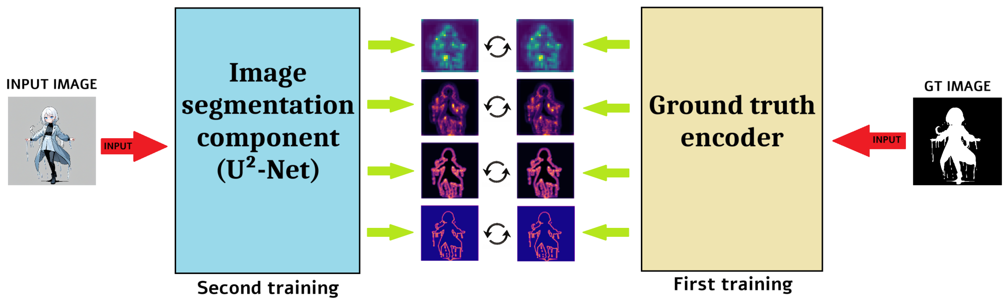

Figure 2.2: Scheme representing the architecture of *IS-Net* model.

classical GT supervision by the loss

$$L_{sg} = \sum_{d=1}^{D} \lambda_d^{sg} BCE(F_{sg}(\theta_{sg}, I)_d, G),$$

where λ_d^{sg} denotes weight for each side output loss. The training process of the segmentation model can be formulated as the optimization problem

$$\arg\min_{\theta_{sg}} (L_{fs} + L_{sg}).$$

The *image segmentation component* is implemented as a U^2-*Net*, described in Section 2.2.3. In fact, this architecture provides strong performance in capturing fine structures.

2.3.2 Frequency priors

An approach sometimes used in computer vision tasks to improve performance consists of using knowledge in the frequency domain during image processing [32, 19]. In particular, as the parts of the image with sharp contrast changes correspond to high frequency, we can rely on the fact that frequency prior provides valuable cues to precisely localize fine-grained object edges, which is sometimes difficult in the RGB domain. As a consequence of these considerations, performing *Dichotomous Image Segmentation* utilizing frequency information seems worth exploring.

FP-DIS [54] is the state-of-the-art model that, using the *frequency priors*, can capture very fine details of the input image. Qualitative and quantitative

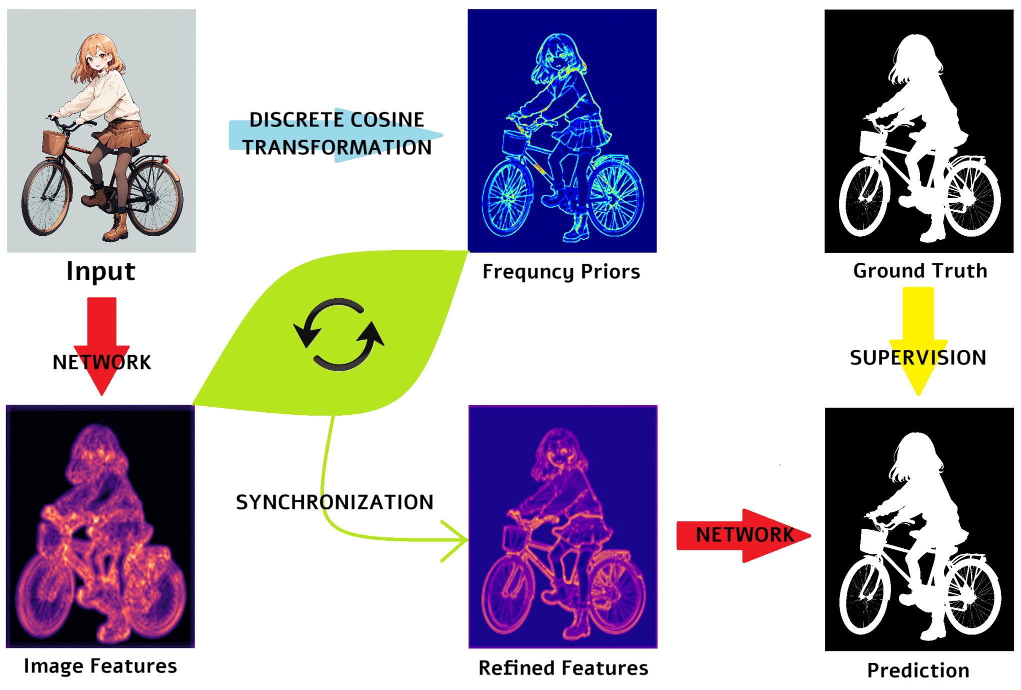

Figure 2.3: Conceptual scheme explaining how *FP-DIS* model works.

experiments determined the high quality of the predictions made by this network. The model, as shown in Figure 2.3, first catches multi-scale features with a *pyramid feature extractor*. The feature *harmonization module* aims to harmonize features having similar semantics and different scales. Meanwhile, the input image is processed by the *frequency prior generator* to compute frequency priors, which are embedded into the harmonized multi-scale features using the *frequency prior embedding module*. Going more in depth with the details, we explain how each module precisely works:

- *pyramid feature extractor:* Convolution neural networks are widely adopted for common vision tasks, getting satisfactory performance. However, in the DIS task, the size of the input is often large and the target objects are fully detailed. Shallow networks are not sufficiently deep to work well with this type of input, because they are in general more focused on local information. To deepen the feature extractor, we use the vision transformer with long-distance modeling capability, in addition to the convolution layers. Specifically, we have *ResNet-50* [13] as the CNN-based backbone to extract multiscale features $\{X_i\}_{i=1}^4 \in \mathbb{R}^{\frac{H}{2^i} \times \frac{W}{2^i} \times C_i}$ from the input image $I \in \mathbb{R}^{H \times W \times 3}$, where H and W denote respectively the height and width, and $C_i \in \{256, 512, 1024, 2048\}$ is the number of channels. The feature X_4 is downsampled by a convolution layer and fed into a transformer block [38] to get X_5 and this operation is repeated on X_5 to obtain X_6. The channels of X_5 and X_6 are 256, but to facilitate the subsequent step, we convert the channels of all the features $\{X_i\}_i^6$ to 96.

- *frequency prior generator:* to generate *frequency priors*, the *discrete cosine transformation (DCT)* [8] is first used to transform the image I into the frequency domain, getting the frequency distribution map M:

  $$M = DCT(I).$$

  DCT is a linear and invertible function. Taking as domain three-dimensional $\mathbb{R}$-tensors with fixed sizes N_1, N_2 and N_3 (as our input image is) its rigorous definition is the following:

  $$DCT : \mathbb{R}^{N_1 \times N_2 \times N_3} \to \mathbb{R}^{N_1 \times N_2 \times N_3}$$

  $$DCT(T)[k_1, k_2, k_3] = \alpha_{N_1}[k_1] \cdot \alpha_{N_2}[k_2] \cdot \alpha_{N_3}[k_3] \cdot \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \sum_{n_3=0}^{N_3-1} T[n_1, n_2, n_3] \cdot$$

  $$\cdot \cos\left(\frac{\pi(2n_1+1)k_1}{2N_1}\right) \cdot \cos\left(\frac{\pi(2n_2+1)k_2}{2N_2}\right) \cdot \cos\left(\frac{\pi(2n_3+1)k_3}{2N_3}\right),$$

  $$\text{where} \quad \alpha_{N_i}[k_i] = \sqrt{\frac{1}{N_i}} \cdot \begin{cases} 1 & \text{if } k_i = 0 \\ \sqrt{2} & \text{otherwise,} \end{cases}$$

  $$\text{for each} \quad k_i \in \{0, 1, \ldots, N_i - 1\}, \quad i \in \{1, 2, 3\}, \quad T \in \mathbb{R}^{N_1 \times N_2 \times N_3}.$$

  After applying this function, the fixed filter and the learnable filters extract different frequency components. In particular, the fixed filter divides the frequency information into different bands (low frequency, medium frequency, high frequency, and all frequency), while the learnable filters provide more information. Eventually, the frequency priors are generated using the outputs from learnable and fixed filters, respectively F_l and F_f, and the inverse function of the *discrete cosine transformation*, denoted by *iDCT*:

$$X_{fp} = iDCT(M \odot (F_f + \sigma(F_l))),$$

where σ is a function to restrict the output in the interval $[-1, 1]$, while $\odot$ stands for the *Hadamard Product* [24].

- *feature harmonization module:* the features generated by the pyramid structure are widely heterogeneous because they contain different structural and semantic information. In fact, the shallow layers capture abundant information, while the deeper layers extract features with more semantics. The fusion of these multi-scale features would drive the model to focus on both the information types. Taken two outputs from the *pyramid feature extractor*, denoted by X_i and X_{i+1}, they are upsampled on the depth dimension, which is multiplied by N times. Then, what obtained is split into N groups that are subsequently divided into subgroups. The harmonization is performed both intra-group and inter-group. This is reached making use of harmonization components, namely gate units consisting of a concatenation among specific functions necessary for filtering. The output harmonized features are passed through a convolution layer for smoothness, and then added to the inputs X_i and X_{i+1} to get as final output of the model X'_i and X'_{i+1}.

- *frequency prior embedding module:* direct fusion between frequency priors and image features leads to inferior performance due to the semantic gap between the frequency and image domains. For this reason the embedding procedure is not straightforward. There are two cascaded components involved in frequency embedding. The first one is responsible for embedding the frequency priors into the features $\{X'_i\}_{i=1}^{6}$ given as input one at time. For each $i \in \{1, 2, \ldots, 6\}$, X'_i and the frequency priors are aggregated and then passed through filtering and other functions. Then, the second frequency embedding component takes as input the output of the first component and the last output produced by the *frequency prior embedding module* itself. This allows for the transfer from deep semantics to shallow semantics. After that X'_i have been fed into the module for every $i \in \{1, 2, \ldots, 6\}$, the final output is upsampled to the original image size.

Denoting by G the ground truth map and P the model prediction, the supervision is performed minimizing the loss function

$$L(P, G) = BCE^w(P, G) + IoU^w(P, G).$$

BCE^w and IoU^w are weighted version of, respectively *Binary Cross-Entropy* and *Intersection Over Union* (better explained in Section 2.4) [45], where the more a specific pixel is located in an error-prone area (borders or holes), the more it is heavily weighted, to ensure the network focusing on difficult details.

2.3.3 Multi-view aggregation

During the design stage of a *DIS* model, balancing in the best way the semantic dispersion of high-resolution targets in the small receptive field and the loss of fine-grained details in the large receptive field is the main challenge. Moreover, the task is nowadays applied on images with higher and higher resolution and this results in a relatively small receptive field, obstructing the network's ability to catch essential global semantics. A solution used to address these problems consists of using multi-resolution inputs, but given that the high-resolution image itself embeds all the information contained in the low-resolution image, this method may lead to repetitive computation and as a consequence to a lower model speed. Since real-time applications require very fast image processing, we aim to avoid issues of this type.

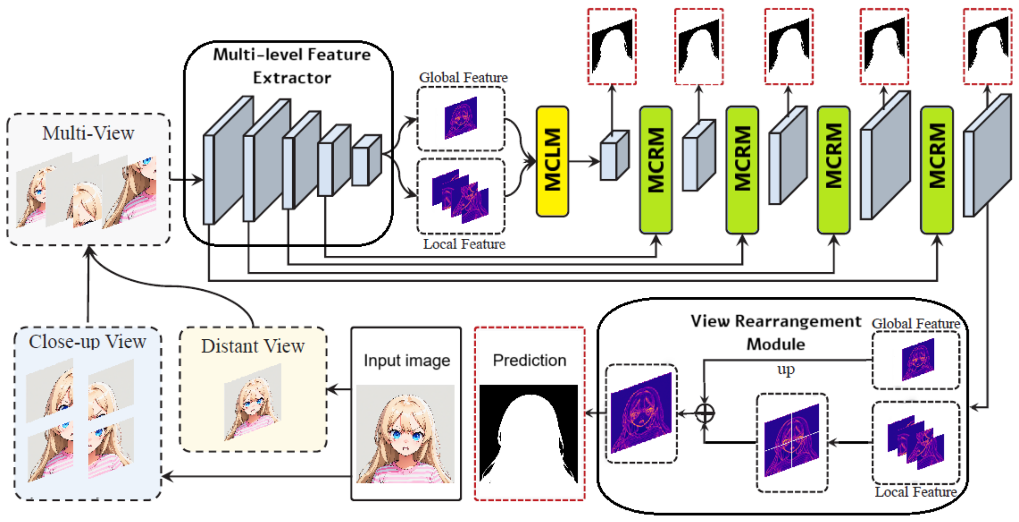

Figure 2.4: Scheme representing the *MVANet* workflow, from the given input image, to the prediction mask.

Finding a valuable solution, motivates the design of a new approach inspired by the human visual system that captures interesting details by observing them from multiple views. In particular, it consists of splitting the high-resolution input images from the original view into the far-view images used for global information and close-up view images containing local details.

Multi-View Aggregation Network (MVANet) [50] is a framework compatible with global and local signals, implemented following the idea of the aforementioned approach. It obtains global semantics and local features in parallel according to the characteristics of different patches, and manages the feature fusion of the distant view and close-up view into a single stream. This network provides strong visual interactions across multiple views, leading features to represent highly detailed structures. As illustrated in Figure 2.4, the input $I \in \mathbb{R}^{B\times3\times H\times W}$ (a batch of images) is rescaled to get a low-resolution version $P_0 \in \mathbb{R}^{B\times3\times h\times w}$, which represents the distant view. In addition, I is cropped into several non-overlapping patches $\{P_m\}_{m=1}^{M}$. Each of them is a specific close-up view representing a fine-grained texture. P_0 and $\{P_m\}_{m=1}^{M}$ together make up the multi-view patch sequence, which is fed into the *Multi-level Feature Extractor* where the multi-level feature maps $\{E_i\}_{i=1}^{5}$ are generated. Each of them embeds information about both the far and detailed representations. The one from the highest level (E_5) is partitioned along the batch dimension into global and local features. They serve as input for the *Multi-view Complementary Localization Module (MCLM)* to extract information about objects' positions in the global view. *MCLM* then filters out wrong information from the close-up view. After this stage, the updated feature maps are joined along the batch dimension to have a unique feature map $D_5 \in \mathbb{R}^{B\times 3\times \frac{h}{32}\times\frac{w}{32}}$, which is sent to the decoder. A *Multi-view Complementary Refinement Module (MCRM)* is inserted in each decoding stage. These modules dynamically add missing details to the global representation using local information. Moreover, shallow features are added layer by layer in the upsampling path in the decoder. After decoding, both global and local features pass through a *View Rearrangement Module*, where the positional and semantic information from the global view is merged with the detailed information from the patches, into a unified whole. At this point, we retrieve D_1' whose shape is the shape of the original image divided M times. To obtain the final output, D_1' is not directly upsampled: shallow features are incorporated as low-level information to further enhance the quality of the final result.

Looking at Figure 2.4, one can notice that each layer's output of the decoder and the final prediction are supervised. Specifically, the decoder output loss consists of three components: L_l, L_g and L_a, respectively for the assembled local representation, the global representation, and the token attention map in the *MCRM*. The final output loss is denoted by L_f instead. These components employ the combination of the *Binary Cross-Entropy (BCE)* loss and the *weighted Intersection over Union (IoU^w)*:

$$L_i = BCE + IoU^w, \quad \text{where } i \in \{l, g, a, f\}.$$

The total loss is computed in the following way:

$$L = L_f + \sum_{j=1}^{5}(L_l^j + \lambda_g L_g^j + \lambda_a L_a^j),$$

where the index j serves to indicate which decoder layer the component refers to and λ_g and λ_a are weights to reflect the relevance of each component.

Experiments on the popular *DIS-5K* [35] dataset suggest that *MVANet* significantly outperforms state-of-the-art methods in both accuracy and speed.

2.4 Bilateral Reference Network

2.4.1 Approach

In Section 2.3, we listed and explained the state-of-the-art approaches for *DIS*, presenting the corresponding architectures. Although these models achieved favorable results, improvements are still necessary on particular types of image, for example images containing thin structures or camouflaged objects.

Bilateral Reference Network (BiRefNet) [53] is an innovative model, recently developed with the objective of improving *DIS* on cases where state-of-the-art models perform poorly. Its realization is inspired by other segmentation works [52, 15], which found that fine and non-salient features in image objects can be well extracted by computing gradient features through derivative operations on the original image. At the same time, if any area displays color and texture very similar to the background, the gradient features are probably not enough. For this reason, *BiRefNet* introduces a *bilateral reference* mechanism. In particular, the original image is cropped into patches that are fed into the decoder at different stages and merged with the original features. This phase is known as *inward reference*, and has the goal of supplementing high-resolution information. At the same time, each decoder stage generates a gradient map, which is used to perform gradient supervision. This mechanism is named *outward reference*, and has the purpose to draw attention to areas displaying dense details. We chose *BiRefNet* as the baseline model in Chapter 3, for our background removal project.

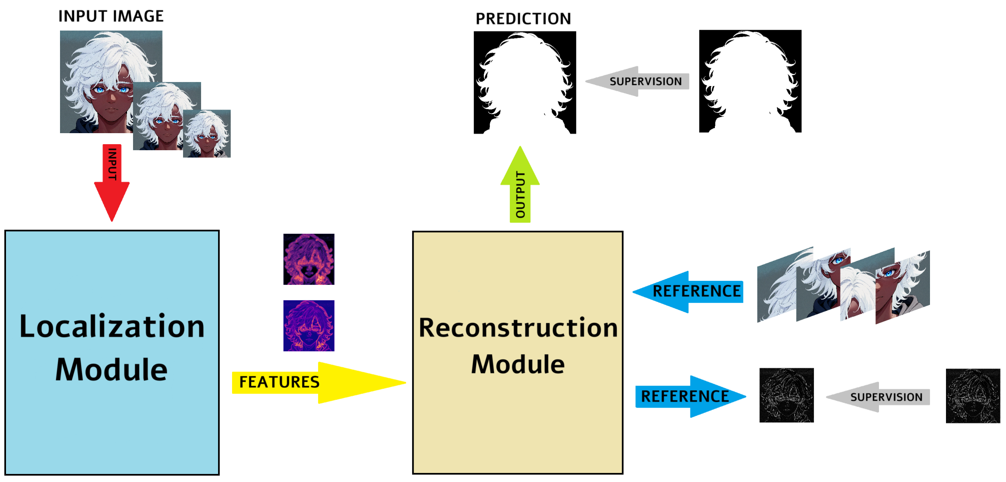

Figure 2.5: Visual explanation of *BiRefNet* modules and *inward reference* and *outward reference*.

2.4.2 Architecture

The *BiRefNet* architecture is divided into two main components: a *localization module* and a *reconstruction module* which serve, respectively, as encoder and decoder. A batch of images $I \in \mathbb{R}^{N\times3\times H\times W}$ is given as input to the *localization module*, and passes through a transformer encoder [20] which extracts features for each stage $\{F_i^e\}_{i=1}^3$, with decreasing resolution $\frac{H}{k} \times \frac{W}{k}$ where $k \in \{4, 8, 16\}$. These features are transferred to the same resolution decoder stages using lateral connections (1×1 convolution layers). Moreover, they are stacked and concatenated in the last encoder block, generating F^e. This feature is then passed to a classification module, to get a semantic representation useful for object localization. Simultaneously, F^e is fed into an ASPP module [5], to both enlarge the receptive field and focus on local features, getting F^d in output to be passed to the *reconstruction module*.

This module has a *BiRef block* at each stage. The one at the lowest level takes as input the feature map F^e and returns F_d^3. The upper ones take as input $F_i^{d+} := F_i^d + F_i^l$, where $i \in \{1, 2, 3\}$ is the corresponding level, F_i^d is the previous stage output and F_l^i is generated by the same level encoder stage and is provided through lateral connections. The output returned by each *BiRef block* is F^d_{i-1}, and the upper one F^d_1 is summed to F^l_1 to get F^{d+}_1. This one passes through a 1×1 convolutional layer, to obtain the final output prediction mask $M \in \mathbb{R}^{N \times 3 \times H \times W}$. Meanwhile, each block generates a prediction M_i, with stage by stage increasing resolution, that serves as ulterior supervision. Inside each *BiRef block* we have the *bilateral reference* mechanism. In fact, the input image I is cropped into patches $\{P_k\}^n_{k=1}$ in an adaptive way: each patch will have the same resolution of the block input F^{d+}_i. These patches are used as *inward reference*, to add high-resolution representation of the details to the encoded features. The patches are then concatenated with F^{d+}_i and fed into the *reconstruction block*, where deformable convolutions [9] are applied to extract features with receptive fields of various scales. After these operations, the generated map $F^{d'}_i$ is used to compute a gradient map prediction G_i (*outward reference*). The latter is supervised by G^m_i, computed multiplying point-wise the gradient map of the input image, and the predicted segmentation mask M_i. We need this multiplication to discard the useless gradient information generated by background noise, obtaining an attention map focusing on foreground parts reacher in details. With this procedure we obtain features sensitive to the gradient, which are multiplied to $F^{d'}_i$ to get F^d_i.

2.4.3 Losses

Relying solely on a loss keeping track of the difference between prediction and ground truth in a pixel by pixel way, is not sufficient to achieve great results on the fine-grained structures. Following the idea of a previous work [33], the objective function in *BiRefNet* is defined as a weighted combination of different losses:

$$\begin{aligned} L &= L_{\text{pixel}} + L_{\text{region}} + L_{\text{boundary}} + L_{\text{semantic}} \\ &= \lambda_1 BCE + \lambda_2 IoU + \lambda_3 SSIM + \lambda_4 CE, \end{aligned} \tag{2.1}$$

where $\lambda_1, \lambda_2, \lambda_3$, and λ_4 are, respectively, set to 30, 0.5, 10, and 5 to keep all the losses in the same range when starting the training. Denoting by M the model prediction and G the ground truth map, the complete definitions of the losses are reported below:

- *Cross-Entropy (CE)* [3]: semantic-aware loss, which is used to correctly classify each pixel:

$$CE = -\sum_{(i,j)} \frac{1}{N} \sum_{c=1}^{N} y_c(i,j) \log(p_c(i,j)),$$

  where N is the number of classes, $y_c(i,j)$ states whether class label c is the correct classification for pixel (i,j) and $p_c(i,j)$ denotes the predicted probability that pixel (i,j) is of class c.

- *Binary Cross-Entropy (BCE):* it is a version of CE used when the number of classes is 2. In our case, it serves for pixel-level supervision for the generation of binary maps:

$$BCE(M,G) = -\sum_{(i,j)} \left[G(i,j)\log(M(i,j)) + (1-G(i,j))\log(1-M(i,j))\right],$$

  where $G(i,j)$ and $M(i,j)$ denote the value of the ground truth and predicted maps, respectively, at pixel (i,j). Since this loss is computed pixel-wise, it does not consider the labels of the neighborhood and it weights all the pixels equally. Significantly erroneous predictions produce large BCE loss, thus the models trained with this loss suppress rapidly these errors, guaranteeing a relatively good local optimum. However, boundaries and fine structures often remain blurred.

- *Intersection over Union (IoU)* [23]: region-aware supervision for the enhancement of binary map predictions:

$$IoU(M,G) = 1 - \frac{\sum_{(i,j)} M(i,j)G(i,j)}{\sum_{(i,j)} \left[M(i,j) + G(i,j) - M(i,j)G(i,j)\right]}.$$

Larger areas contribute more to the IoU score. Moreover, as a consequence of the formula, the large foreground regions impact more on the score. Hence, models trained with this loss are able to produce relatively homogeneous and confident probabilities for these regions. On the contrary, it is frequent to find false positives in those predictions.

- *Structural Similarity Index Measure (SSIM)* [44]: boundary-aware supervision. Given $y = \{y_j : j = 1, ..., N^2\}$ and $x = \{x_j : j = 1, ..., N^2\}$ representing the pixel values of two corresponding $N \times N$ patches derived from G and M, respectively. $SSIM(x, y)$ is defined as

$$SSIM(x,y) = 1 - \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{2.2}$$

  where μ_x, μ_y and σ_x, σ_y are the means and standard deviations of x and y, respectively. The notation σ_{xy} stands for their covariance, and C_1 and C_2 are constants used to avoid division by zero. The $SSIM$ final score is calculated averaging the values found using each pixel as the center of the patch. Since this loss takes into account a local neighborhood of each pixel, those errors on the boundary affect $SSIM$ computation while using patches centered in the whole pixel region around.

To take advantage of the above losses, we combine them together to formulate the hybrid loss L as in (2.1). CE is needed to add semantic supervision, BCE is used to get the prediction closer to the ground truth in an image-level smooth way, while IoU is employed to put more focus on the foreground. $SSIM$ serves to encourage the prediction to respect the structure of the original image, as well as further push the backgrounds predictions to zero.

2.4.4 Training strategies

Training models on high-resolution data leads to high costs in terms of time and computational resources. For this reason, it is natural to look for training strategies that allow us to make the training as short as possible. Experiments highlighted that a first incomplete version of *BiRefNet* model converges relatively quickly in the localization of targets and the segmentation of rough structures. However, the performance in segmenting fine parts is still increasing after 400 epochs. Though long training can easily achieve great results in terms of both structure and edges, it consumes too much computation; supervision using intermediate predictions in the decoder levels (as described in Section 2.4.2), can dramatically accelerate the learning process on segmenting fine details and enable the model to achieve similar performance as before but with only 30% training epochs. We also found that changing the loss computation at some epoch, including only region-level losses, can easily improve the binarization of predicted results and metric scores. Finally, context feature fusion and image pyramid inputs on the backbone are implemented, and these two modifications to the backbone achieved an ulterior improvement.

2.4.5 Results and applications

Quantitative and qualitative evaluations show that *BiRefNet* outperforms *DIS* models introduced in Section 2.3. In fact, comparing the segmentation of slim shapes and curve edges made by different state-of-the-art networks, *BiRefNet* is the model that achieved the best performance. Moreover, it demonstrated a great ability to identify the precise location in the picture of the right target and to correctly segment camouflaged objects.

Due to this power in precisely segmenting very fine structures, *BiRefNet* seems to be the right model to be used for some *DIS* applications where previous approaches were not sufficient to reach results satisfying enough to be implemented for real use cases. For example, anomaly detection (such as cracks in the wall) and complex object background removal are two applications where this model could be employed.

2.5 Metrics

After analyzing state-of-the-art approaches to put into practice *Dichotomous Image Segmentation*, we need some metrics to evaluate the quality of the obtained results. In the following, we list the main metrics that are used nowadays for this task, explaining their structure, their advantages and their limitations.

2.5.1 F-measure

F-measure [29] is one of the most applied metrics to image segmentation tasks, but also to completely different machine learning areas (for example natural language processing, document classification, etc.) and in general to every binary classification problem. In fact, suppose we have a collection of items, and each of them can be labeled as positive or negative. At the end of the classification, items can be distinguished into four categories: *True Positives (TP), False Positives (FP), True Negatives (TN)*, and *False Negatives (FN)*. At this point, we consider the following ratios:

$$\text{recall} = \frac{TP}{TP + FN}, \quad \text{precision} = \frac{TP}{TP + FP}.$$

F-measure is defined as the weighted harmonic mean between *precision* and *recall*:

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} = \frac{(1 + \beta^2)TP}{(1 + \beta^2)TP + \beta^2 FN + FP},$$

where $\beta \in \mathbb{R}^+$ is a parameter to weight the importance of *precision* with respect to *recall.* The obtained score aims to be an overall measure of classification "effectiveness", taking into account both *precision* and *recall* in a unique score. The harmonic mean is chosen to combine these quantities because it is the operation commonly used to average ratios. To apply *F-measure* to *DIS* both the ground truth and the predicted map are binarized, choosing a threshold value and setting all the pixels above that value as foreground (white) and the pixels below that value as background (black). White pixels are considered positives and black pixels are considered negatives, and the final score is computed using the aforementioned formulas.

Although the definition of *F-measure* appears to be representative of prediction quality, some issues arise when applying it to specific cases. In particular *F-measure* relies on two wrong assumptions. First, each pixel is considered to be independent of each other but, in practical applications, a sparse error is considered better than an error concentrated in a specific area, in case of an equal amount of wrong predictions; second, errors' location is not taken into account while computing the score, but wrong predictions close to the foreground boundary are in general less visible than in the other areas.

These considerations motivate to analyze a more sophisticated version of the metric, called *Weighted F-measure* [22]. This version employs a strategy to make up for the lack of attention to pixel correlation and error location. In fact, consider $g \in \{0,1\}^{N^2}$ and $p \in [0,1]^{N^2}$ as the column stack representation of, respectively, the binarized ground truth and the non-binarized map to be evaluated. Let $z := |g - p|$ be the pixel-by-pixel error vector. We also define a matrix $A \in \mathbb{R}^{N^2 \times N^2}$ that captures the dependencies between pixels:

$$A(i,j) = \begin{cases} 1 & \text{if } i = j \\ \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{d(i,j)^2}{2\sigma^2}} & \text{if } g(i) = g(j) = 1 \\ 0 & \text{otherwise,} \end{cases}$$

where σ is a parameter to get quantities in the right range, and $d(i,j)$ is the Euclidean distance between pixels i and j. This matrix assigns to each couple of pixels a correlation coefficient from the interval [0,1]. In particular, the correlation between each pixel and itself is set to 1; two different foreground pixels receive a gaussian coefficient in the range $(0,1)$, decreasing as the distance between them increases; correlation between a foreground and a background pixel, or two different background pixels, is set to 0. Moreover, a weight vector $b \in \mathbb{R}^{N^2}$ to keep track of each pixel location is defined in the following way:

$$b(i) = \begin{cases} 1 & \text{if } g(i) = 1 \\ 2 - e^{\alpha\delta(i)} & \text{otherwise,} \end{cases}$$

where $\delta(i) := \min_{g(j)=1} d(i,j)$, while α is a negative constant. Note that $b(i) \in [1,2)$ for each i, and the closer a pixel is to the foreground, the lower its weight. Having A and b, a new version of the error vector can be computed, taking into account both the pixel correlation and the pixel location: $z^w := \min(z, Az) \odot b$, where $min()$ is intended as an element-wise function and ensures the error for each pixel cannot increase. At this point, we define the following quantities as a weighted non-binary generalization of *TP, FP, TN* and *FN*:

$$\begin{aligned} TP^w &= (1 - z^w) \odot g & TN^w &= (1 - z^w) \odot (1 - g) \\ FP^w &= z^w \odot (1 - g) & FN^w &= z^w \odot g \end{aligned}$$

The final *Weighted F-measure* is computed in total analogy with the classical *F-measure*, using this just defined quantities:

$$F_\beta^w = (1+\beta^2)\frac{\text{precision}^w \cdot \text{recall}^w}{\beta^2 \cdot \text{precision}^w + \text{recall}^w} = \frac{(1+\beta^2)TP^w}{(1+\beta^2)TP^w + \beta^2 FN^w + FP^w}.$$

2.5.2 S-measure

S-measure [11] was conceived to measure the structural similarity between the foreground objects in the ground truth and the prediction. In fact, vision studies concluded that the human visual system is highly sensitive to similarity in structures. As a consequence, a metric based on this type of similarity could return scores more aligned to the human thought.

To get the correct result, *S-measure* computes two different terms and then combines them together in a weighted average. The first term is called *region-aware* and aims to first catch local information and then gather it into global information. The image is split into four parts of equal size, and each of them recursively into four parts again and so on until the desired patch number K is reached. After that for each patch $k \in \{1, 2, \ldots, K\}$ the $SSIM_k$ score is computed, using (2.2) with both the constants set to 0. The *region-aware* score is obtained with a weighted sum of the local scores:

$$S_r = \sum_{k=0}^{K} w_k (1 - SSIM_k),$$

where w_k are weights proportional to the percentage of foreground pixels contained in patch k. Meanwhile, *object-aware* term considers the distributions of foreground and background regions and compares them between the ground truth and the map to be evaluated. The idea behind *object-aware* score is that the ground truth usually has uniform distributions in both the background and foreground regions. So, we aim to assign high value to a map with salient object uniformly detected. Let X be a random variable; in probability theory the ratio $\sigma_X / \mathbb{E}[X]$ is a dispersion measure of the probability distribution of X. Relying on what just mentioned, this index applied to the map to be evaluated can be a good dissimilarity marker with respect to the ground truth. Accordingly, the dissimilarity coefficient is defined as

$$D_{FG} = \frac{(\mu_{x_{FG}})^2 + (\mu_{y_{FG}})^2}{2\mu_{x_{FG}}\mu_{y_{FG}}} + \lambda \frac{\sigma_{x_{FG}}}{\mu_{x_{FG}}},$$

where x_{FG} and y_{FG} stand for the foreground part of, respectively, the map to be evaluated and the ground truth and $\lambda \in \mathbb{R}^+$ balances the two addenda. The first term of the sum is employed to compare the two foreground pixels average. The other quantities are defined in a straightforward fashion:

$$D_{BG} = \frac{(\mu_{x_{BG}})^2 + (\mu_{y_{BG}})^2}{2\mu_{x_{BG}}\mu_{y_{BG}}} + \lambda \frac{\sigma_{x_{BG}}}{\mu_{x_{BG}}}, \quad O_{FG} = \frac{1}{D_{FG}}, \quad D_{FG} = \frac{1}{D_{BG}}.$$

The *object-aware* score is computed considering β the ratio of foreground area to total area in the ground truth image, and employing the sum

$$S_o = \beta O_{FG} + (1 - \beta) O_{BG}.$$

The final *S-measure* is a weighted average between the *region-aware* and *object-aware* terms

$$S = \alpha S_o + (1 - \alpha) S_r,$$

with $\alpha \in [0, 1]$ chosen to give the right importance to the two components.

2.5.3 E-measure

Although *S-measure* achieves excellent performance in *DIS* evaluation, it is designed for non-binary maps. Applying this measure to binary maps, leads to poor results. *E-measure* [10] is a different metric tailored for binary maps evaluation. It combines local-pixel values and image-level mean value, matching information jointly. Consider a binary foreground map F. Denoting by A the matrix having the same dimension as F and all the elements equal to 1, we can define the *bias matrix* φ_F as the distance between each pixel and the global average

$$\varphi_F = F - \mu_F \cdot A.$$

Denoting by $G \in \{0, 1\}^{h\times w}$ the ground truth map and $P \in \{0, 1\}^{h\times w}$ the prediction to be evaluated, we compute the *bias matrix* with $F \in \{G, P\}$. Therefore, an *alignment matrix* between φ_G and φ_P is computed in the following way:

$$\xi = \frac{2\varphi_G \odot \varphi_P}{\varphi_G \odot \varphi_G + \varphi_P \odot \varphi_P},$$

where the fraction is intended to be element-wise. For each pixel (i, j) we get an alignment value $\xi(i, j) \in [-1, 1]$ that is positive when $G(i, j)$ and $P(i, j)$ are aligned with respect to the global image mean, otherwise it is negative. Moreover, $\xi(i, j) = 1$ if and only if $\varphi_G(i, j) = \varphi_P(i, j)$, and $\xi(i, j) = -1$ if and only if $\varphi_G(i, j) = -\varphi_P(i, j)$. At this point, each element of the matrix ξ is rescaled to the range [0,1] using the following convex function element-wise:

$$\phi(i, j) = f(\xi(i, j)), \quad \text{where } f(x) = \frac{1}{4}(1 + x)^2,$$

for each $i \in \{1, 2, \ldots, h\}$ and $j \in \{1, 2, \ldots, w\}$. In this way a very low score (below ~ 0.3) is assigned to pixels not aligned between G and P, and higher scores are reserved to aligned pixels. The final *E-measure* is the pixel-by-pixel average of ϕ:

$$E = \frac{1}{wh} \sum_{j=1}^{w} \sum_{i=1}^{h} \phi(i, j).$$

2.5.4 MAE and MSE

Mean Absolute Error (MAE) and *Mean Squared Error (MSE)* [31] are classical metrics that are often used for the evaluation process in every regression task. Both of these metrics measure the error element-by-element (pixel-by-pixel in *DIS* context), and they return an average where all of them are equally weighted. Denoting by $G \in \mathbb{R}^{h \times w}$ the ground truth image and $P \in \mathbb{R}^{h \times w}$ the image to be evaluated, the mathematical definitions of the two metrics are the following:

$$MAE(G, P) = \frac{1}{hw} \sum_{j=1}^{w} \sum_{i=1}^{h} |G(i, j) - P(i, j)|,$$

$$MSE(G, P) = \frac{1}{hw} \sum_{j=1}^{w} \sum_{i=1}^{h} (G(i, j) - P(i, j))^2.$$

The main difference between the two scores consists in the way large and strict errors are weighted. In fact, the square power returns even larger numbers if applied to large numbers and even smaller numbers if applied to small numbers. For this reason, MSE further accentuates extreme values, with respect to MAE.

2.5.5 Boundary Intersection over Union

Another famous segmentation metric is *Intersection over Union*, which considers the foreground region of two binarized maps and returns the ratio of the intersection of those regions to the union, as the name suggests. Despite this metric is one of the most common, it is not reliable enough when applied to large objects containing fine details. In fact, a theoretical analysis of the metric structure highlights that when scaling up the representation of an object, the number of boundary pixels grows linearly while the number of interior pixels grows quadratically. The consequence is that *Intersection over Union* computed on larger objects tolerates a larger number of misclassified pixels for each unit of contour length.

Boundary Intersection over Union (BIoU) [6] is a different version of the metric designed to avoid this problem. Consider G_{FG} the foreground region of the ground truth, P_{FG} the map to be evaluated, and $d \in \mathbb{N}$. First, the images region G_d and P_d distant less than d pixels from the foreground boundary are intersected with the foreground region of the corresponding images; then, the score is computed applying the *Intersection over Union* formula with these new regions:

$$BIoU(G, P) = \frac{|(G_d) \cap (G_{FG})|}{|(P_d) \cup (P_{FG})|}.$$

The parameter d controls the sensitivity of the measure. With d large enough $BIoU$ becomes equivalent to IoU, otherwise internal pixels sufficiently distant from the boundary are ignored and the score becomes more sensitive to the small boundary inaccuracies. The risk of a too small d is to over-penalize small errors, which in some case are harmless and due to imperfect annotations.

Boundary Intersection over Union can be overall considered a valuable metric to highlight small boundary errors on large objects that some of the other metrics are designed to ignore. On the other hand, a metric defect arises while using it in some special cases where the prediction is labeled as perfect although a complete area is missing (for example, $BIoU$ returns 1 comparing an image representing a circle of radius r and an image representing a circular crown of internal radius $r - d$ and external radius r).

2.5.6 Mean Boundary Accuracy

Another metric that cares only about the boundary area is *Mean Boundary Accuracy (MBA)* [1]. This metric keeps track of the pixel accuracy in the area within a certain pixel distance to the boundary, averaging the scores obtained using different distance thresholds. Consider the ground truth $G \in$

$\{0,1\}^{h \times w}$ and a corresponding prediction $P \in \{0,1\}^{h \times w}$, both subjected to a binarization process. First, 5 radii with uniformly spaced interval in $[3, \frac{w+h}{300}]$ are sampled; denoting them by $\{r_i = 3 + \frac{i}{4}(\frac{w+h}{300} - 3) \mid i \in \{0,1,2,3,4\}\}$, for each one the region of interest is identified applying to the ground truth foreground boundary a disk kernel of radius r_i. Subsequently, the number a_{r_i} of aligned pixels between P and G in the region of interest is counted, as well as the total number t_{r_i} of pixels inside that region. The partial scores and the final MBA are computed with the following:

$$s_{r_i} = \frac{a_{r_i}}{t_{r_i}}, \qquad MBA(G,P) = \frac{1}{5}\sum_{i=1}^{5} s_{r_i}.$$

2.5.7 Human Correction Effort

Metrics we listed so far evaluate the quality of a prediction by relying on the mathematical or perceptual differences distinguishing the image from the ground truth. This is not always the correct conceptual idea, for example in case of real applications where the predicted foreground mask is then subjected to a human correction process. In these cases the fundamental point is not whether a foreground map looks like the ground truth or not: a good prediction is characterized by a low number of needed human correction operations to get exactly the right segmentation.

Human Correction Effort (HCE) [35] is a metric that approximates the number of mouse clicks required to correct the segmentation mask. Specifically, during this correction process two operation types are conducted:

1. point selection along target boundaries to formulate polygons, used when a wrong region is surrounded (even if partially) by a correct region labeled in the same way;

2. region selection, used when a wrong region is completely surrounded by a differently labeled region.

Each of these operations corresponds to a human operator mouse click. The final *HCE* score is supposed to be representative about the human effort required to correct the segmentation mask.

Chapter 3

ToonOut: Background Removal Model for Anime style

In this chapter we are going to study a concrete application to the Dichotomous Image Segmentation task, discussed in Chapter 2. In particular we present *ToonOut*: a fine-tuned background removal model specialized in anime character images. This work has been developed by us, in collaboration with the company *Kartoon*.

3.1 Motivation and goals

In Chapter 2 Dichotomous Image Segmentation (DIS) task has been deeply studied, going over all the state-of-the-art approaches and highlighting how recent advancements in machine learning have led to significant progress so that results are often fabulous. As mentioned, DIS is commonly used for background removal, isolating the salient foreground object for downstream applications like image editing, story creation, or logo design. While current state-of-the-art background removal models excel at processing realistic images, they often struggle with data far from the distribution they are trained on. For this reason, using those models on specific domains like non-realistic style images (hand-drawn images, comics images, or anime-style images) in

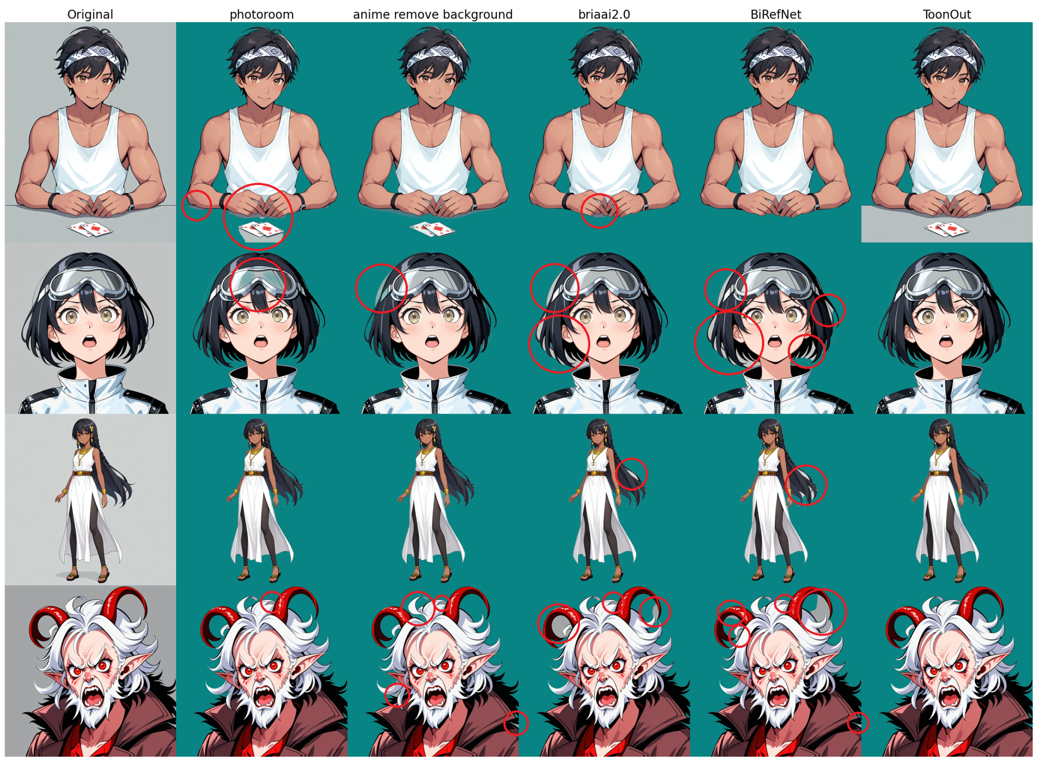

Figure 3.1: Closed source [14] and open source models ([40], [4], [53]) are inadequate on our anime characters test set. We fine-tune BiRefNet [53] and the resulting model *ToonOut* yields better outputs for our use-case.

general leads to poor and disappointing results.

The Kartoon company developed the *toongether* [42] application, a mobile application designed to allow users to create their own anime comic-style stories, and share them with the community. In fact, *toongether* gives access to a wide number of different characters, backgrounds and items. Users can compose their story by simply dragging characters and items silhouettes over the background and typing dialog text into speech bubbles. When a new character or item image is loaded into the application, it undergoes a background removal process, to get the needed silhouette. Since these are anime-style images, state-of-the-arts DIS models' performance [53, 4, 14] are degraded (see Figure 3.1), particularly in handling complex features like hair and transparent elements. Even models specifically designed for anime [16,

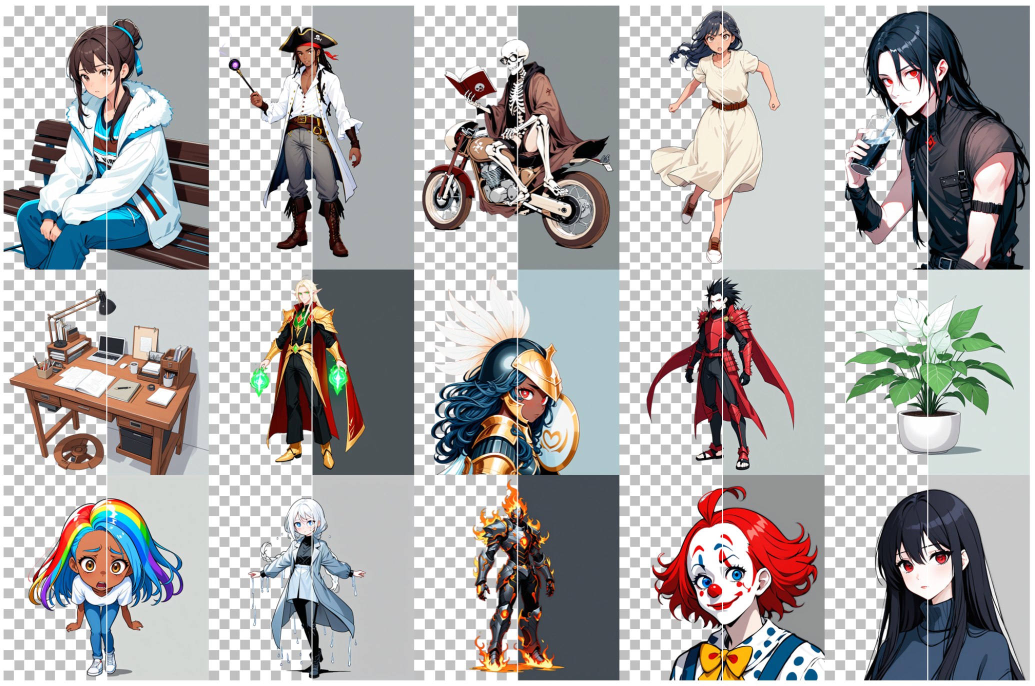

Figure 3.2: Examples of predictions made by *ToonOut* on images contained in our test datasets.
Our datasets cover a variety of characters in challenging poses, their interactions with items, and standalone objects.

40] often yield unsatisfactory results.

This challenge motivates our work to improve background removal for images of anime characters and items. We aim to select a popular DIS model and enhance its capabilities on anime content. The goal of the project consists in contributing in the following way: (1) gathering a high-quality custom dataset of anime images depicting characters and items; (2) fine-tuning the popular open-source BiRefNet model on this dataset, demonstrating performance that matches the best closed-source models; and (3) introduce a new metric, *Pixel Accuracy*, to evaluate fine-grained performance of DIS models.

We are open-sourcing the code, the fine-tuned model weights, as well as the dataset at: `https://github.com/MatteoKartoon/BiRefNet`.

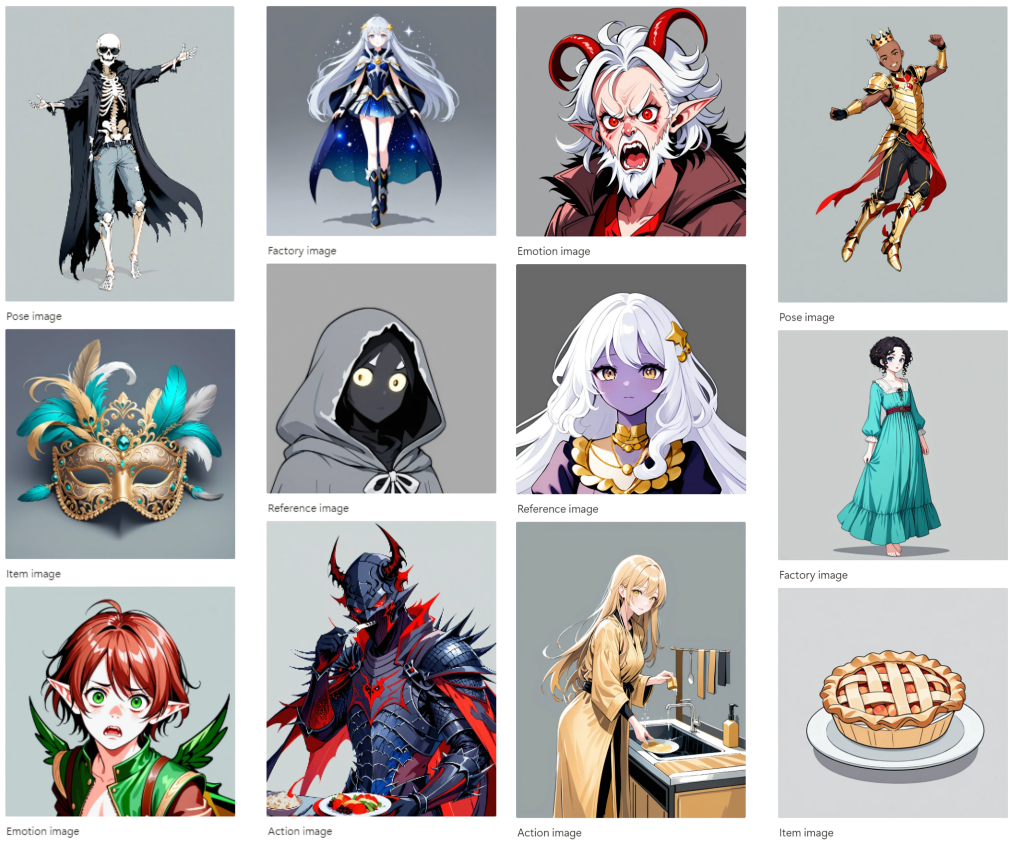

Figure 3.3: Example of images contained in the dataset, covering different characters, poses and items.

3.2 Custom dataset

3.2.1 Data sourcing

Our dataset consists of 1228 images broken down into `train / validation / test` with an `80% / 10% / 10%` ratio (see Table 3.1). For each image, we provide both the original RGB image and the corresponding pixel-level ground truth mask. This consists of a grayscale image where black pixels represent the background, white pixels represent the foreground, and intermediate gray values indicate partially transparent pixels (useful, in particular, to provide a transparency gradient around character edges).

We designed our dataset to meet the following essential criteria:

- *Domain coverage:* The dataset should comprehensively represent the target domain, encompassing diverse anime-style content including both character portraits and object illustrations. To ensure broad generalization, we collected images featuring varied character designs, poses, viewing angles, and activities, grouped in various sub-datasets.

- *Data quality:* All images maintain high resolution (minimum 1024×1024 pixels) and visual realism to preserve fine-grained details essential for accurate segmentation; the annotations are top-quality and highly-precise as well.

- *Sample diversity:* To prevent overfitting and ensure robust evaluation metrics, the dataset presents high diversity. Each image represents a unique combination of character, pose, and context, so that we verify that the trained model doesn't over-fit the `train` & `validation` distribution.

To get images for the dataset we followed the same procedure used at *Kartoon* when new contents are needed for the *toongether* application, namely generating images using a diffusion model (see Chapter 1 for a detailed explanation about these models). In particular, we used the anime-specialized checkpoint *Yamer's Anime* [48] of *Stable Diffusion XL* [28, 2]. Characters are defined by three reference pictures, and a prompt that is broken down into segments (a different segment for each character body or outfit detail). We then used different workflows and hyperparameters to generate the different sub-datasets explained in Section 3.2.2. Here are the main common building blocks:

- We applied a different *IP-adapter* [49] for the face and the body, a composition *ControlNet* [51] for pose, and a *Concept Sliders* [12] for emotions.

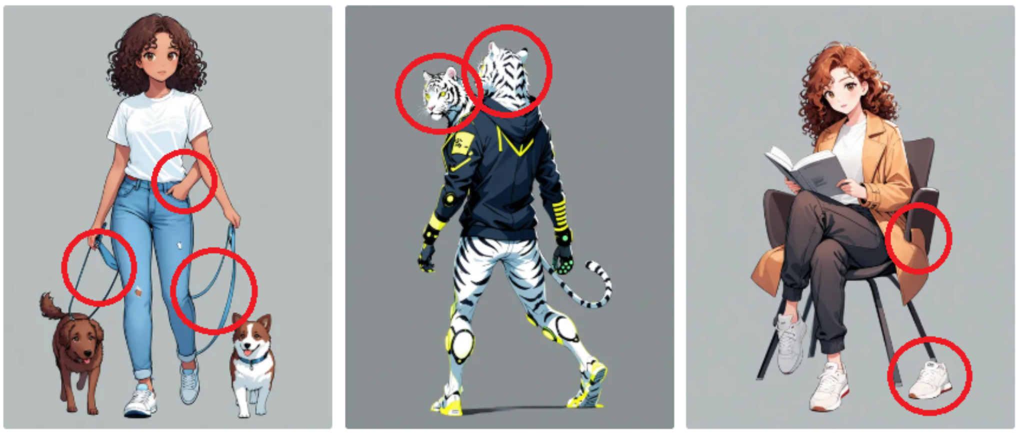

Figure 3.4: Examples of discarded images: in correspondence of red circles generation issues can be observed, leading these images looking not realistic.

- We used an *upscaler* called *RealESRGAN* [43] to go from 1024×1024 resolution to 2048×2048.

- After the *upscaler* we do a refiner pass (image-to-image diffusion) to increase image quality.

Gathering images operating this generation procedure using a diffusion model is clearly convenient. First, it allows us to get a wide batch of images really quickly. Moreover, we are obtaining images with exactly the same style of those used in the *toongether* application, and this should let our final fine-tuned model perform well in this environment, that will be the first one where our model will be used. After the generation phase, the obtained images were rigorously filtered to ensure high quality: to ensure our dataset matching the essential criteria we listed, we discarded images with anatomical inconsistencies, unclear foreground-background boundaries, and artifacts that would result in visually unappealing segmentation masks. Some examples of discarded images are provided in Figure 3.4.

Following the filtering phase, we evaluated the remaining images using the baseline model *BiRefNet* to identify challenging cases where the baseline model performs poorly. We prioritized these difficult examples for dataset inclusion, as they represent the most valuable training samples for improving model performance. To maintain dataset balance, we also included up to 20% of images where the baseline *BiRefNet* model already performs well.

The images chosen to be included in the dataset were annotated, to produce the ground truth mask. The annotation was made keeping in mind that the main objective of this model is to be adopted in the *toongether* application. This means that the background removed image should be suitable to be used in a comic story as a character silhouette. For this reason, all the elements shown in the image were kept as part of the foreground (desks, bicycles, pets, etc.). The unique component discarded was the character shadow that, if shown in the image, was annotated as part of the background.

3.2.2 Dataset Composition

To ensure comprehensive domain coverage and training diversity, we structured our data collection around six distinct image categories, each targeting specific visual characteristics and segmentation challenges.

Dataset	**number of images**				**total %**
	train	**validation**	**test**	**total**	
Reference	58	7	8	73	5.9%
Emotion	201	25	26	252	20.5%
Pose	199	25	25	249	20.3%
Factory	324	41	41	406	33.1%
Action	112	14	15	141	11.4%
Items	85	11	11	107	8.7%
Total	**979**	**123**	**126**	**1228**	**100%**

Table 3.1: Break-down of dataset composition

A couple of examples for each category are provided in Figure 3.3. The distribution and characteristics of each of these dataset categories are as follows (see Table 3.1):

- *Reference (neutral face portraits):* High-quality character portraits, with neutral emotion;
- *Emotion (close-up portraits):* Character close-ups expressing various emotions (joy, anger, sadness, etc.). These images are a challenge for baseline models due to fine facial details and hair complexity;
- *Pose (full-body characters in motion):* Full-body character representations in diverse poses (standing, jumping, running, etc.). Baseline model performance is generally better for these samples;
- *Factory (full-body idle characters):* Full-body character representations in idle stance. Similarly to pose, the baseline performance is good;
- *Action (characters with items):* Characters engaged in activities (cooking, gaming, working) that include environmental objects. These samples represent the most challenging segmentation scenarios due to complex foreground-background interactions;
- *Items:* Standalone object illustrations (vehicles, food items, tools, etc.). Baseline model performance varies significantly based on object complexity and boundary definition.

3.3 Fine-tuning process

3.3.1 Model choice

For our fine-tuning approach, we selected the popular and high-performing *Bilateral Reference Network (BiRefNet)* [53] as our base model. *BiRefNet*'s DIS architecture incorporates several design principles well-suited to our segmentation objectives. To summarize what was mentioned in Section 2.4, *BiRefNet*'s bilateral reference mechanism employs dual supervision strategies: auxiliary gradient supervision [52] to enhance detail preservation in fine-grained regions, complemented by ground truth supervision, particularly useful in regions where foreground elements resemble the background in terms of color and texture. In particular, *BiRefNet*'s architecture comprises two primary components: a *Localization Module* and a *Reconstruction Module*, counting a total of 221 millions of parameters. *Localization Module* leverages global semantic information for object localization, while *Reconstruction Module* performs the segmentation reconstruction process using hierarchical image patches as source reference [50] and gradient maps as target reference.

This dual supervision enables the model to maintain awareness of both local details and global context. This aligns with our goals (motivating our choice), that the model must handle varying scales, from fine details of character hair to full-body items and bodies.

3.3.2 Fine-tuning experiment

As a starting point for the model fine-tuning process, we cloned the GitHub *BiRefNet* repository, containing the training code. The most important structural change we made to that code was the addition of validation loss computation after a default number of iterations. Moreover, we implemented the *Weights&Biases* [7] logs, to allow for monitoring and keeping a record of training runs and real-time plotting of the losses, the metrics, or other useful information. After that, we started with the first training experiments by resuming from a checkpoint saved after 244 training epochs, provided by the repository's author. We initially conducted experiments using only a small subset of the final dataset and gradually incorporated the remaining parts as they were sourced and annotated. To achieve the best possible training performance we had to conduct trials with numerous combinations of hyperparameters. We identified the best setting as the following:

- *loss weights:* our loss function is computed as a weighted sum among *SSIM loss, MAE loss, and IoU loss* [53, 31, 34]; an explanation of the

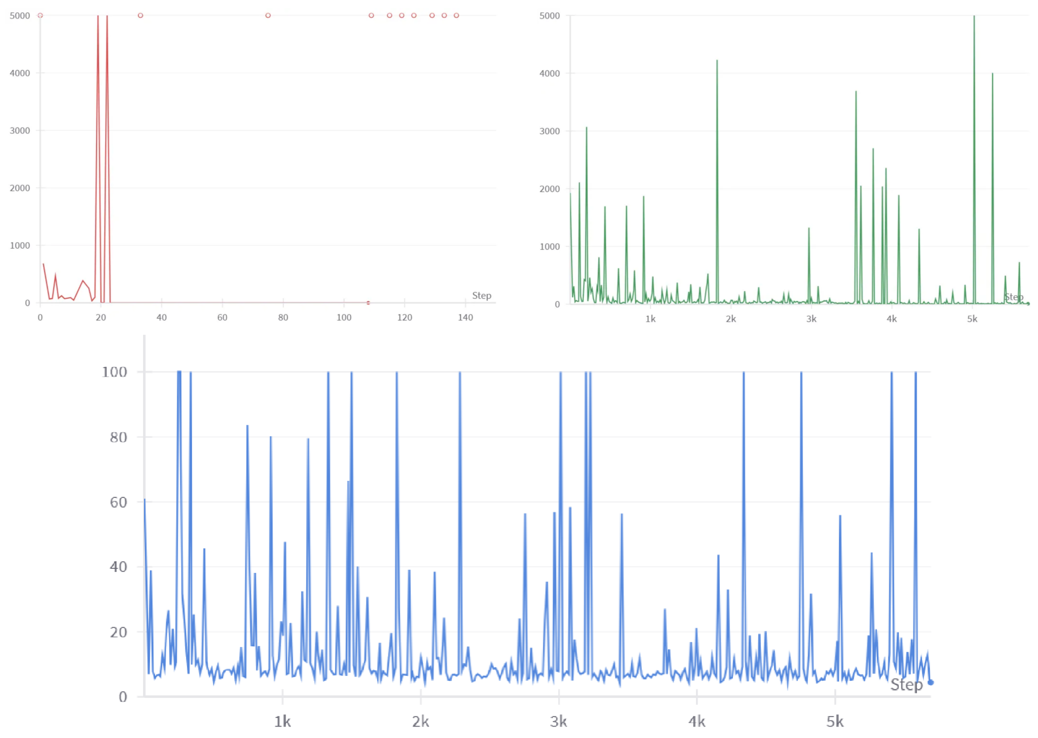

Figure 3.5: Comparison between gradient norm plots about different trainings: the top-left one is the initial setting, using *floating point 16*; the top-right behavior is obtained after switching to *brain float 16*; the bottom one is the final one, after gradient clipping to 100.

formulas behind these losses is provided in Chapter 2.

$$L = \lambda_1 L_{SSIM} + \lambda_2 L_{MAE} + \lambda_3 L_{IoU} \tag{3.1}$$

After trying several combinations, we decided on the right weights to use during the model fine-tuning, which are $\lambda_1 = 10,\ \lambda_2 = 90,\ \lambda_3 = 0.25$;

- *gradient supervision loss:* we have a loss function for the gradient supervision too, and we opted for *BCE loss with logits* [30], considered more stable than computing the *BCE loss* after passing the output to a *sigmoid* function.
- *number of epochs:* we identified the appropriate number of epochs as

100. This is a very high number considering that the model was already trained for 244 epochs, but we perform this long training while saving a checkpoint for each epoch. After the training ended, we looked for the best checkpoint by evaluating each of them on the validation set, using the metrics we will explain in the next section. The checkpoint we chose as our final model was the one related to epoch 290, so we trained for 46 epochs.

- *GPUs:* we trained using in parallel two *GeForce RTX 4090* GPUs.

- *batch size:* The largest batch size our hardware allowed to avoid out-of-memory issues was 2, which became our chosen setting.

- *learning rate*: it was set to 1×10^{-5}, which is the maximum value possible to prevent the training from diverging. Moreover, to make the training loss continue decreasing after a large number of epochs, we implemented learning rate decay, every 20 epochs with a multiplying factor of 0.5.

- *mixed precision*: it was set by default to *floating point 16*, but during our first training runs we observed the *gradient norm* oscillating and subsequently becoming *NaN* at some point. This unstable behavior was resolved by switching to *brain float 16* [46], a format more suited for machine learning.

- *gradient clipping*: after the stabilization just mentioned, we obtained a stable *gradient norm* plot. Despite this, sometimes individual spikes appeared, indicating extreme values for certain iterations and a corresponding drop in performance appeared looking at the training loss plot. To address this issue we decided to introduce *gradient clipping*, setting the maximum accepted gradient norm value to 100. The different gradient norm behaviors can be compared in Figure 3.5.

model	dataset	# images	Pixel Accuracy	Boundary IoU	Weighted F-measure
Photoroom			98.9%	90.9%	99.3%
BiRefNet	reference	8	96,6%	82.7%	98.5%
ToonOut [ours]			**99.8%**	**95.5%**	**99.7%**
Photoroom			99.7%	95.0%	99.6%
BiRefNet	emotion	26	97.7%	87.4%	98.7%
ToonOut [ours]			**100.0%**	**96.9%**	**99.8%**
Photoroom			**99.9%**	**96.7%**	**99.5%**
BiRefNet	pose	25	99.1%	94.2%	99.1%
ToonOut [ours]			**99.8%**	96.4%	**99.5%**
Photoroom			**99.9%**	**96.8%**	**99.5%**
BiRefNet	factory	41	98.7%	93.0%	98.9%
ToonOut [ours]			**99.8%**	96.0%	**99.4%**
Photoroom			96.3%	91.5%	98.6%
BiRefNet	action	15	76.8%	69.4%	91.2%
ToonOut [ours]			**99.0%**	**93.1%**	**99.3%**
Photoroom			**98.3%**	**94.3%**	**98.6%**
BiRefNet	items	11	92.2%	92.2%	97.2%
ToonOut [ours]			96.6%	92.5%	97.8%
Photoroom			99.2%	95.2%	**99.3%**
BiRefNet	overall	126	95.3%	88.5%	97.8%
ToonOut [ours]			**99.5%**	**95.6%**	**99.4%**

Table 3.2: Performance on the sub-datasets released. *Items* remain challenging as they are a small part of the overall dataset.

3.4 Results evaluation

3.4.1 Metrics overview

In Section 2.5 metrics that are currently used to evaluate the performance of Dichotomous Image Segmentation models are studied in detail. Now we have to measure the quality of our *ToonOut* model, so we are going to select appropriate metrics to assess the performance of background removal in anime character images.

Empirically, two primary categories of model prediction errors can be observed:

- *Coarse-grained errors*: Large regions may be incorrectly classified; common examples are: between a character's ponytail and head, between limbs and torso, in the folds of the clothes, or the character's shadow;

- *Fine-grained errors*: Small-scale inaccuracies, such as hair strands, fingers, detailed object contours.

Accordingly, evaluation measures for Dichotomous Image Segmentation fall into two categories:

- *Coarse-grained metrics* judge errors at the pixel level, so small boundary mistakes barely change the score. We considered: *S-measure* [11], *E-measure* [10], *F-measure*, *Mean Absolute Error (MAE)* and *Mean Squared Error (MSE)*;

- *Fine-grained metrics* focus on pixels near object edges, rewarding crisp outlines. We looked at: *Boundary Intersection over Union (BIoU)* [6], *Human Correction Effort (HCE)* [35], *Weighted F-measure (WF)* [22], and *Mean Boundary Accuracy (MBA)* [1].

To select the most suitable evaluation metrics, we conducted experiments on the validation set. Going through the predictions made by different models on validation images and corresponding scores, we immediately individuated relevant examples where all the *coarse-grained metrics* assigned a definitely incorrect score. That is why, as a consequence of how these metrics are structured, in the predictions where a number slightly different from the GT value is assigned to a wide portion of the image, the returned score is relatively high. In the cases where this difference is visible for human eyes, the human judge the prediction as certainly incorrect, in contrast with the metric score. This consideration let us conclude that no one of the *coarse-grained metrics* was the right one to evaluate this type of images. The second phase of this exploration consisted in analyzing the *fine-grained metrics* behavior. To conduct the experiment, we compared in the following way predictions made by *Photoroom*, *Briaai2.0*, and a first version of *ToonOut* trained only on few images. We individuated a 10 images subset of the validation set including cases we considered interesting for metrics evaluation. For each of those images, we first established a human-judged ranking of model outputs, then observed how often each metric aligned with that preference. We got the following results:

- *BIoU:* 1 error
- *MBA:* 5 errors
- *WF:* 1 error
- *HCE:* 4 errors

As you can see, the *Boundary IoU* and the *Weighted F-measure* demonstrated the strongest concordance with our visual assessments of boundary quality.

3.4.2 Pixel Accuracy

We introduce *Pixel Accuracy (PA)* with the goal of finding a region-oriented metric that match with our visual rankings. This metric is designed by us, meant to return a score depending on the fraction of correctly labeled pixels, as specified in equation (3.2). This motivates the choice of the name *Pixel Accuracy.*

A pixel is counted as correct when the absolute difference δ between prediction and ground-truth alpha values is $\delta \leq 10$, which is visually indistinguishable. To find the right δ value, we conducted several trials, by iteratively changing the α value of an image area and identifying the maximum difference that remained visually imperceptible. To avoid penalizing harmless few-pixel-wide edge artifacts around the character and object boundaries, we erode the error mask once using the `cv2.erode()` function [26], and a full-of-ones 5×5 matrix as the kernel. The *PA* score is then computed using the formula

$$PA = \left(1 - \frac{\text{Number of Incorrect Pixels}}{\text{Total Number of GT Foreground Pixels}}\right)^2 \qquad (3.2)$$

Model	Open code & weights	Open data	Performance over our test set (126 images)						
			Pixel Accuracy	Boundary IoU	Weig. F-measure	E-measure	S-measure	MAE	F-measure
Photoroom	✗	✗	99.2%	95.2%	**99.3%**	99.2%	98.7%	**0.04**	**99.2%**
Briaai2.0	✓	✗	97.8%	92.4%	98.8%	98.8%	97.9%	0.08	98.7%
BiRefNet	✓	✓	95.3%	88.5%	97.8%	97.9%	96.9%	0.15	98.0%
ToonOut [ours]	✓	✓	**99.5%**	**95.6%**	**99.4%**	**99.4%**	**98.9%**	**0.03**	**99.2%**

Table 3.3: On our test sets, we manage to slightly outperform Photoroom's state-of-the-art remove-background model.

where "GT Foreground Pixels" denotes the pixels with $\alpha > 128$ in the ground truth mask. This computation allows for taking into account the width of foreground area in the ground truth image. In fact, in the case of the same number of incorrect pixels, an error is more evident if it occurs in an image where the foreground area is less wide. This holds because the larger the character represented in the image, the less of the body portion the error affects. The exponentiation to the power of two is performed to prevent the obtained scores from being clustered in a small range close to one, better distinguishing them from each other.

Testing our new metric on the validation set following the same procedure described in the previous section (comparing its rankings with those made by human on the same 10 images), *PA* aligned with every human ranking. We can conclude that *Pixel Accuracy* performs as intended, and thus it will be one of the metrics used for results evaluation.

3.4.3 Evaluating model performance

After selecting the optimal checkpoint based on the validation set, we evaluate the performance of our *ToonOut* model on the introduced test sets.

This evaluation follows three key metrics, chosen after experiments described in Section 3.4.1: *Pixel Accuracy (PA)*, *Boundary Intersection over Union (BIoU)*, and *Weighted F-measure (WF)*. We benchmarked *ToonOut* against two prominent models: the closed-source high-performing *Photoroom* [14] and two open-source baselines, *Briaai2.0* [4] and *BiRefNet* [53].

The aggregated results, presented in Table 3.3, indicate a clear trend:

all the evaluation metrics scores reported agree that *ToonOut* bridges the gap between the open-source models and *Photoroom*, overall reducing the *PA error rate* from 2.2% / 4.7% to 0.5%, and the *BIoU error rate* from 7.6% / 11.5% to 4.4%. Looking across individual datasets (Table 3.2), the performance is slightly worse for the *items* category, where the images are few (only 107 images) and quite varied. Also for *pose* and *factory* images *ToonOut* does not achieve scores as high as *Photoroom*, but a great improvement with respect to baseline *BiRefNet* is clearly shown. Overall, we can observe the high performance of *ToonOut*'s capabilities in anime character background removal, particularly in the challenging datasets *action* and *emotion*.

To conclude, we can establish metrics show that *ToonOut* effectively narrows the performance disparity between the open-source baselines and the closed-source state-of-the-art. This is what we aimed for while starting this project. Great examples, where *ToonOut* model makes the best prediction among the models selected for the comparison, are shown in Figure 3.1. Gathering a relatively small dataset, entirely dedicated to anime-style data distribution, was enough to "specialize" the baseline *BiRefNet* on this type of images. We wish this observation to serve as inspiration for future works similar to ours. In fact, it is desirable that the strength of fine-tuning on a particular domain holds in general, and not only for anime style. Moreover, our contribution on metrics should be valuable in other domains as well.

Conclusions

In this thesis we treated some theoretical topics, such as the theory behind *Diffusion Models*, the state-of-the-art models to solve *Dichotomous Image Segmentation* and the metrics used to evaluate *DIS* performance. Moreover, we discussed the practical project in which we realized the *ToonOut* model, describing both the process and the results, as well as a new metric implemented by us, well-suited for *DIS* evaluation. A possible future development for this work could be the enlargement of the dataset to understand whether more data can improve *ToonOut*'s performance. Moreover, by now the model is fine-tuned only on characters and items, but new classes of images could be required for the *toongether* [42] application, for example animals. In this case, it would be great to include samples of these types in the dataset. Eventually, we know that Machine Learning is always evolving rapidly and things become outdated in a short time. For this reason, it is plausible that new *DIS* models will be released soon, improving the state of the art. In this case it would be interesting to repeat the project using the same dataset and these new models as a baseline, with the purpose of further improving the results.

Bibliography

[1] Ho Kei Cheng (HKUST) et al. "CascadePSP: Toward Class-Agnostic and Very High-Resolution Segmentation via Global and Local Refinement". In: (2020). DOI: `https://arxiv.org/pdf/2005.02551`.

[2] Chris Bishop. *Deep Learning*. Springer, 2024. ISBN: 978-3-031-45467-7.

[3] Pieter-Tjerk de Boer et al. "A Tutorial on the Cross-Entropy Method". In: (2005). DOI: `https://people.smp.uq.edu.au/DirkKroese/ps/aortut.pdf`.

[4] briaai. *RMBG-2.0*. URL: `https://huggingface.co/briaai/RMBG-2.0`.

[5] Liang-Chieh Chen et al. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: (2018). DOI: `https://arxiv.org/pdf/1802.02611`.

[6] Bowen Cheng et al. "Boundary IoU: Improving Object-Centric Image Segmentation Evaluation". In: (2021). DOI: `https://arxiv.org/pdf/2103.16562`.

[7] CoreWeave. *Weights&Biases*. URL: `https://wandb.ai/site/`.

[8] V. A. Coutinho, R. J. Cintra, and F. M. Bayer. "Low-complexity Multidimensional DCT Approximations". In: (2023). DOI: `https://arxiv.org/pdf/2306.11724`.

[9] Jifeng Dai et al. "Deformable Convolutional Networks". In: (2017). DOI: `https://arxiv.org/pdf/1703.06211`.

[10] Deng-Ping Fan et al. “Enhanced-alignment Measure for Binary Foreground Map Evaluation”. In: (2018). DOI: https://arxiv.org/pdf/1805.10421.

[11] Deng-Ping Fan et al. “Structure-measure: A New Way to Evaluate Foreground Maps”. In: (2017). DOI: https://arxiv.org/pdf/1708.00786.

[12] Rohit Gandikota et al. “Concept Sliders: LoRA Adaptors for Precise Control in Diffusion Models”. In: (2023). DOI: https://arxiv.org/pdf/2311.12092.

[13] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: (2015). DOI: https://arxiv.org/pdf/1512.03385.

[14] Photoroom Inc. *Photoroom*. URL: https://www.photoroom.com/it.

[15] Ge-Peng Ji et al. “Deep Gradient Learning for Efficient Camouflaged Object Detection”. In: (2022). DOI: https://arxiv.org/pdf/2205.12853.

[16] Komiko. *AI Background Removal*. URL: https://komiko.app/background-removal.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: (2012). DOI: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[18] Chen-Yu Lee et al. “Deeply-Supervised Nets”. In: (2014). DOI: https://arxiv.org/pdf/1409.5185.

[19] Junxuan Li, Shaodi You, and Antonio Robles-Kelly. “A Frequency Domain Neural Network for Fast Image Super-resolution”. In: (2017). DOI: https://arxiv.org/pdf/1712.03037.

[20] Ze Liu et al. “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”. In: (2021). DOI: https://arxiv.org/pdf/2103.14030.

[21] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: (2015). DOI: https://arxiv.org/pdf/1411.4038.

[22] Ran Margolin, Lihi Zelnik-Manor, and Ayellet Tal. “How to Evaluate Foreground Maps?” In: (2014). DOI: https://arxiv.org/pdf/1708.00786.

[23] Gellert Mattyus, Wenjie Luo, and Raquel Urtasun. “DeepRoadMapper: Extracting Road Topology from Aerial Images”. In: (2017). DOI: https://openaccess.thecvf.com/content_ICCV_2017/papers/Mattyus_DeepRoadMapper_Extracting_Road_ICCV_2017_paper.pdf.

[24] Elizabeth Million. “The Hadamard Product”. In: (2007). DOI: http://buzzard.ups.edu/courses/2007spring/projects/million-paper.pdf.

[25] Keiron O’Shea and Ryan Nash. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: (2015). DOI: https://arxiv.org/pdf/1511.08458.

[26] OpenCV. *erode*. URL: https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html.

[27] Andrea Pascucci. *Teoria della Probabilitá*. Springer, 2020. ISBN: 978-88-470-3999-5.

[28] Dustin Podell et al. “SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis”. In: (2023). DOI: https://arxiv.org/pdf/2307.01952.

[29] David M. W. Powers. “What the F-measure doesn’t measure: Features, Flaws, Fallacies and Fixes”. In: (2015). DOI: https://arxiv.org/pdf/1503.06410.

[30] PyTorch. *BCEWithLogitsLoss*. URL: https://docs.pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html.

[31] Jun Qi et al. "On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression". In: (2020). DOI: https://arxiv.org/pdf/2008.07281.

[32] Yuyang Qian et al. "Thinking in Frequency: Face Forgery Detection by Mining Frequency-aware Clues". In: (2020). DOI: https://arxiv.org/pdf/2007.09355.

[33] Xuebin Qin et al. "BASNet: Boundary-Aware Salient Object Detection". In: (2019). DOI: https://openaccess.thecvf.com/content_CVPR_2019/papers/Qin_BASNet_Boundary-Aware_Salient_Object_Detection_CVPR_2019_paper.pdf.

[34] Xuebin Qin et al. "Boundary-Aware Segmentation Network for Mobile and Web Applications". In: (2021), pp. 6–7. DOI: https://arxiv.org/pdf/2101.04704.

[35] Xuebin Qin et al. "Highly Accurate Dichotomous Image Segmentation". In: (2022). DOI: https://arxiv.org/pdf/2203.03041.

[36] Xuebin Qin et al. "U2-Net: Going Deeper with Nested U-Structure for Salient Object Detection". In: (2022). DOI: https://arxiv.org/pdf/2005.09007.

[37] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: (2016). DOI: https://arxiv.org/pdf/1506.01497.

[38] Sucheng Ren et al. "Shunted Self-Attention via Multi-Scale Token Aggregation". In: (2022). DOI: https://arxiv.org/pdf/2111.15193.

[39] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: (2015). DOI: https://arxiv.org/pdf/1505.04597.

[40] skytnt. *Anime Remove Background.* URL: https://huggingface.co/spaces/skytnt/anime-remove-background.

[41] Jascha Sohl-Dickstein et al. “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”. In: (2015). DOI: `https://arxiv.org/pdf/1503.03585`.

[42] toongether.ai. *Toongether: Comic Creation.* URL: `https://toongether.ai`.

[43] Xintao Wang et al. “Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data”. In: (2021). DOI: `https://arxiv.org/pdf/2107.10833`.

[44] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. “Multi-scale Structural Similarity for Image Quality Assessment”. In: (2017). DOI: `https://www.cns.nyu.edu/pub/eero/wang03b.pdf`.

[45] Jun Wei, Shuhui Wang, and Qingming Huang. “F3Net: Fusion, Feedback and Focus for Salient Object Detection”. In: (2019). DOI: `https://arxiv.org/pdf/1911.11445`.

[46] wikipedia. *bfloat16 floating-point format.* URL: `https://en.wikipedia.org/wiki/Bfloat16_floating-point_format`.

[47] wikipedia. *Markov chain.* URL: `https://en.wikipedia.org/wiki/Markov_chain`.

[48] Yamer. *SDXL Yamer’s Anime.* URL: `https://civitai.com/models/76489`.

[49] Hu Ye et al. “IP-Adapter: Text Compatible Image Prompt Adapter for Text-to-Image Diffusion Models”. In: (2023). DOI: `https://arxiv.org/pdf/2308.06721`.

[50] Qian Yu et al. “Multi-view Aggregation Network for Dichotomous Image Segmentation”. In: (2024). DOI: `https://arxiv.org/pdf/2404.07445`.

[51] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. “Adding Conditional Control to Text-to-Image Diffusion Models”. In: (2023). DOI: `https://arxiv.org/pdf/2302.05543`.

[52] Zhao Zhang et al. “Gradient-Induced Co-Saliency Detection”. In: (2020). DOI: https://arxiv.org/pdf/2004.13364.

[53] Peng Zheng et al. “Bilateral Reference for High-Resolution Dichotomous Image Segmentation”. In: (2024). DOI: https://arxiv.org/pdf/2401.03407.

[54] Yan Zhou et al. “Dichotomous Image Segmentation with Frequency Priors”. In: (2023). DOI: https://www.ijcai.org/proceedings/2023/0202.pdf.

Ringraziamenti

Un sentito grazie al prof. Giovanni Paolini, relatore di questa tesi, per l'attenta supervisione e tutti i preziosi consigli.

Ci tengo anche a ringraziare l'azienda *Kartoon* per la grande opportunità di tirocinio, e in particolare Joël, Thomas, Cédric, Samir e Gabriel senza i quali sarebbe stato impossibile portare a termine questo lavoro.

Per quanto riguarda i ringraziamenti personali confermo tutto quello che ho scritto sulla tesi triennale riguardo ai miei amici, mio fratello i miei genitori e le mie nonne.