Optimizing Perceived Aesthetics of UIs Using Metric Guided Generative Pipelines

Bachelor Thesis - Informatik B.Sc.

Moritz Wörmann
ScaDS.AI, Leipzig University

1. Referee: Dr. Patrick Ebel

Leipzig, 15.08.2024

Declaration

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann. Ich versichere, dass das elektronische Exemplar mit den gedruckten Exemplaren übereinstimmt.

Leipzig, August 16, 2024

Moritz Wörmann

Abstract

User Interfaces (UIs) and their design are an integral part of the User Experience (UX) of mobile applications and the preference of users to choose one app over another. While it is an important part of mobile application development, it is often a complex and time-consuming task for the developer to come up with an aesthetic UI design. Therefore, automating or assisting this task is a relevant topic that has been researched extensively.

This thesis explores the metric-guided generation of UIs to ensure that the generated UIs conforms to users' expectations in terms of aesthetics. Different generative processes are explored that are guided by a Convolutional Neural Network (CNN) that predicts how users will rate aesthetics. This predicted metric is used as a score which is optimized in a gradient descent pipeline. A key problem is the optimizer "learning" weaknesses in the metric predictor that increase the metric without meaningfully changing the actual UI. Despite exploring many different generative processes and testing a large number of hyperparameters, this problem could not be overcome.

Contents

List of Figures

List of Acronyms

AI	Artificial Intelligence
BLIP	Bootstrapping Language-Image Pre-training
CNN	Convolutional Neural Network
DM	Diffusion Model
ELBO	Evidence Lower Bound
FID	Fréchet Inception Distance
GAN	Generative Adversarial Network
GUI	Graphical User Interface
IoU	Intersection over Union
LLM	Large Language Model
MSE	Mean Squared Error
NN	Neural Network
OS	Operating System
PDF	Probability Density Function
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SD	Stable Diffusion
UI	User Interface
UX	User Experience
VAE	Variational Auto Encoder

Chapter 1

Introduction

In recent years smartphones have become increasingly indispensable for almost everyone. Most interactions with these smartphones happen through apps. Today, the Operating System (OS) Android has a market share of around 80% of the global smartphone market [4]. Accordingly, most applications are retrieved or downloaded via the Google Play Store, which currently holds around 2.5m different apps [5]. Users can interact with the applications using Graphical User Interfaces (GUIs). While common design principles, such as Material Design [1], vastly different designs still exist. Users usually prefer an application with an appealing design over an application with an inferior UI design, even when its usability is rated lower [6].

1.1 Motivation

UI design is one of the most important factors for a user's decision to choose one application over another since it is the primary way a user interacts with the application and the UX is heavily influenced by this [7]. UI Design consists of multiple different components, which include color scheme, font choices, layout, and size of UI elements. Utilizing an aesthetically pleasing user interface for the application while simultaneously setting it apart from similar applications is therefore an important factor in the application being successful and adopted by a large user base. Although common design guidelines, especially OS-specific ones, exist, developers might want to use a more customized UI to stand out across all applications.

Through UI design tools like Figma [2], it gets easier to create UIs for apps and websites. However, designing visually pleasing UIs still proves to be a complicated task, especially since these metrics are highly subjective [8]. This challenge becomes even more significant when considering the impact initial impressions of a UI can have on the users' perception and on the willingness to stay on the website or the mobile app [9]. Automating the task of creating UI or providing assistance through automatic algorithms is therefore a worthwhile topic. An end-to-end process of creating UIs, or at least optimizing existing ones, can reduce time and effort.

The UI design process is divided by the Design Council into four separate sections: Discover, Define, Develop, Deliver [10]. This division is also discussed in the next sections. While all of the steps are time-consuming, this research will primarily focus on the "Develop" section, since the layout and design choices are made mostly during this stage.

There are numerous ways to assist developers and UI designers in this process. This thesis focuses on the task of deciding on a layout for all UI components. The choice of this subproblem is advantageous as the amount and type (Buttons, Text boxes, etc.) ofUI

[1]https://m3.material.io
[2]https://www.figma.com/

components is usually already predetermined and dictated by the functionality. Other design components, such as font and color, are often constrained by corporate design or similar guidelines.

As previously discussed, Artificial Intelligence (AI)-assisted UI design can provide meaningful help in the design process. Thus, different processes and generative techniques for automated UI design have been researched [2, 11] in this area. While these approaches leverage datasets consisting of existing UIs to train generative models, this generation can be considered unguided, meaning that some initial requirements can be specified (e.g. amount and type of input elements), but the final generation is not refined in any way. This thesis focuses on an iterative improvement of the generated UI with a focus on perceived aesthetics instead of technical metrics such as alignment measures. With this approach it is ensured that the final generated UI actually conforms to user expectations in terms of aesthetics.

1.2 Problem Definition

The main objective of this thesis is to create a methodology that creates a layout for existing UI elements that is aesthetically pleasing. This could either be used directly when no prior layout exists or as a mechanism to provide alternative designs if a layout already exists. What users consider aesthetically pleasing should be predicted by a model, trained on annotated screenshots of UIs. To optimize a given UI, which is stored in a latent representation from which it can be rendered and graded by the aesthetics predictor, the predicted score is optimized via a gradient descent pipeline.

Another challenge explored by this thesis is the segmentation of existing user interfaces if no prior segmentation exists. After a UI has been segmented, it can then be passed to the aforementioned process to create an optimal layout. This part is therefore also an integral part of the central problem discussed, however, initially only pre-segmented UI datasets will be used.

Additionally, the task of (automatically) predicting how users perceive UIs is explored as the central grading mechanism used to guide the layout generation process. Different existing approaches for this task are compared for their applicability in this use case.

1.3 Contributions

With this thesis, we present a nouveau approach to metric-guided UI generation. This is done in an experiment setup that ensures the generated UIs conforms to users' UX expectations. Additionally, we contribute a fine-tuned Stable Diffusion (SD) model which can be used to generate images of UIs based on a text prompt[3] The dataset used to train this model, which contains text-image pairs is also made available.[4] Furthermore, we present a VAE architecture used to create a latent representation of UI layouts. The code for this thesis is publicly available.[5]

[3]https://huggingface.co/mowoe/stable-diffusion-v1-4-rico-blip-large-conditioned
[4]https://huggingface.co/datasets/mowoe/rico-captions-blip-large-conditioned
[5]https://github.com/mowoe/bachelorthesis

1.4 Results

Different model architectures for predicting the perceived aesthetics of UIs have been explored. While all of them showed promising accuracy results during training and evaluation, this did not translate to the actual use case of them in this thesis, which is the usage as a metric for an optimizer. The optimizer quickly “learned” weaknesses present in the predictor models and exploited them which caused the predicted score to increase rapidly without changes in the generated UI reflecting or justifying the higher score. This can be considered to be an accidental adversarial attack against the predictor models.

1.5 Structure

The thesis is structured in the following way. Section 2 provides a theoretical background for the techniques used that allows the reader to understand the proposed concepts. Chapter 3 discusses research efforts related to the one presented in this thesis. After that Chapter 4 goes over the performed experiments and showcases the results individually. Finally Chapter 5 summarizes the key results of this thesis and gives an outlook into future work.

Chapter 2

Background

2.1 Machine Learning

This chapter provides the reader with background information on a selection of machine learning concepts used throughout this thesis. While the scope of all important concepts is too large to include them in this thesis, this chapter aims to focus on the most important concepts.

2.1.1 Gradient Descent

This section provides the reader with the necessary background information about gradient descent, which is central to the metric-guided optimization part of this thesis.

Gradient Descent is one of the integral building blocks of modern machine learning[12]. It allows models to learn from data and be optimized with regard to a cost function, such as a loss, through an iterative process. Most classic machine learning models can be written as some equation containing parameters and variables. The goal is to find a set of parameters for this equation such that the cost function is minimized across the training data. While the entire model can be viewed as a single equation, the same holds true for the cost function. If we consider the very basic prediction problem that is defined by a dataset created with $f(x) = 2x$, we can define a model with one parameter θ and one variable x as

$$m(x) = \theta \times x \tag{2.1}$$

If we want to optimize the parameter θ, we can use the common loss function Mean Squared Error (MSE), which measures, as the name suggests, the mean squared deviation from each datapoint in the training set:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \tag{2.2}$$

with n being the size of our training dataset, y_i being the real value ($y_i = 2 \times x$ in this case) and $\hat{y}_i$ being the predicted value ($\hat{y}_i = \theta \times x$ in this case). The next step is to view MSE not as a function of $\hat{y}_i$ but of θ, which is done by substituting $\hat{y}_i$ with $m(x)$:

$$\text{MSE}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (\theta \times x_i)) \tag{2.3}$$

The core idea behind gradient descent is to differentiate this parameterized loss function with regards to the trainable parameter [12]. With this differentiation, the gradient at the current parameter point is retrieved. This point is part of whats considered to be the "loss landscape" in which the optimization happens. If the direction (or rather the sign) of the gradient is inverted, the direction in which the parameter needs to be modified to *decrease* the loss is retrieved [12].

In the case of a model/equation which is dependant on multiple parameters, the loss function needs to be partially differentiated with regards to each parameter individually.

After the gradient (i.e. the direction), in which the parameter(s) needs to be modified, in order to decrease the loss, is retrieved, the optimization can be performed. The optimization step is described as

$$\theta' = \theta - \alpha \frac{\partial\, \mathrm{MSE}(\theta)}{\partial\, \theta} \tag{2.4}$$

with α being the *learning rate*. This additional training (hyper-)parameter controls the size of the modification to the parameters and is a crucial component of the training process. If the learning rate is chosen too small, the model might not converge fast enough and if it is chosen too large, the optimal parameters are possibly never found, as they are "skipped over" [12].

In practice, the loss calculation and optimization is not always performed for each training sample individually, but for a number of examples at once. This is considered Batch Gradient Descent (BGD).

Besides these basic optimization algorithms, there are a number of other more complex algorithms, which all rely more or less on these basic constructs. These include

- Adam [13]
- Stochastic Gradient Descent (SGD) [14]
- AdamOnLion [15]

This optimization process can not only be used for training model parameters, but also for optimizing the result of some generative process with regards to a metric, such as optimizing the prompt for generating an image such that the image conforms to some predefined metric, e.g. amount of the color blue.

2.1.2 Differentiable Image Translation

As is described in Section 2.1.1, to optimize a parameter with regards to a metric, the calculation of the metric on the basis of the parameter is differentiated. The implication of this is that the entire computation must actually be differentiable. For most operations, this is relatively straightforward, although not always as canonical as in a simple multiplication (e.g., in the case of clamping). A central operation, which is used in several stages of this thesis, that is however not differentiable, is the placing of smaller image segments on a larger canvas. Mathematically, this operation can be seen as indexing a matrix and overwriting the (pixel) values already present there. However, this is not differentiable with regards to the index itself, which is the parameter to be optimized.

To avoid this problem, the process of affine transformation is used throughout this thesis. The main concept behind this transformation is the use of a 3×3 matrix, called the affine matrix, to control the transformation of an image. One of the key properties of the affine transformation, is that parallel lines in the original image will always stay parallel in the resulting image. The transformation consists of a linear transformation and a translation (shifting each point by a vector).

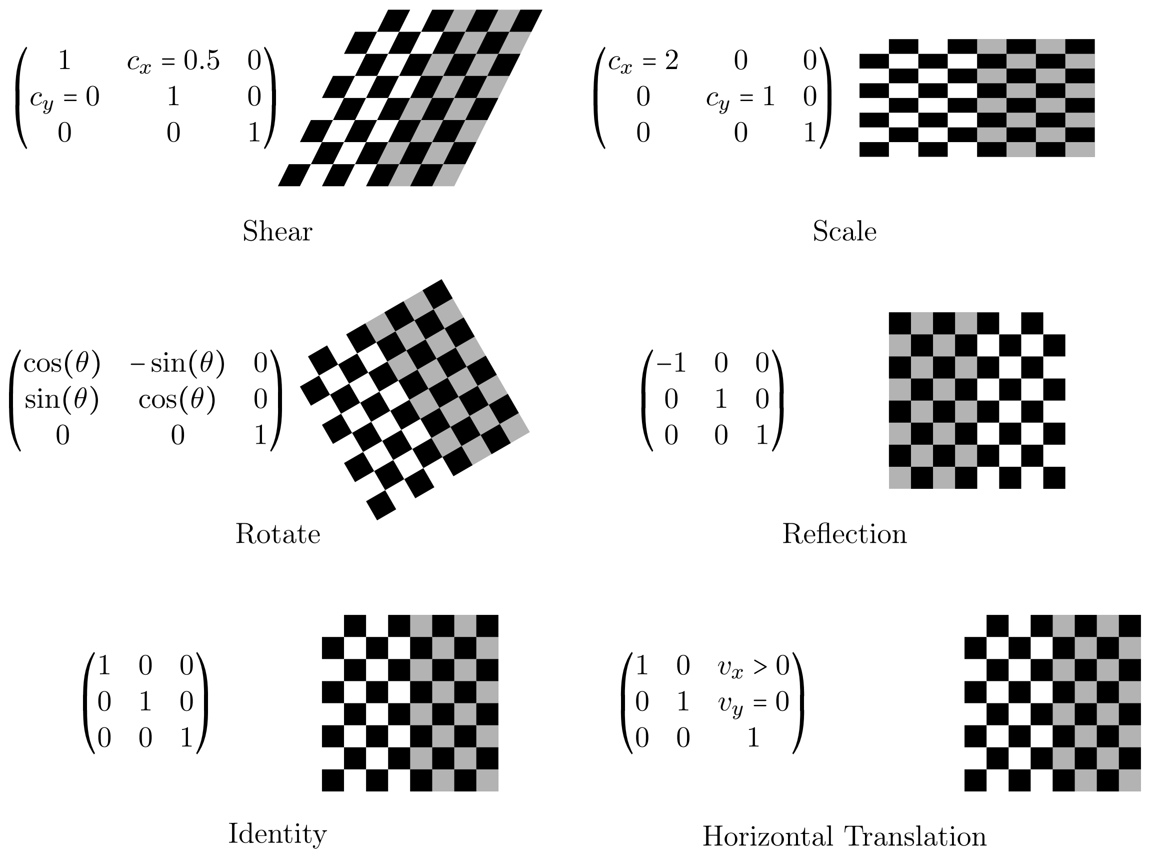

$\begin{pmatrix} 1 & c_x = 0.5 & 0 \\ c_y = 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Shear
$\begin{pmatrix} c_x = 2 & 0 & 0 \\ 0 & c_y = 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Scale
$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Rotate
$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Reflection
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Identity
$\begin{pmatrix} 1 & 0 & v_x > 0 \\ 0 & 1 & v_y = 0 \\ 0 & 0 & 1 \end{pmatrix}$
Horizontal Translation

Figure 2.1: Standard Transformations With Affine Matrices

Formally, the transformation can be written as matrix multiplication with an augmented matrix (linear transformation) and vector (translation):

$$M = \left(\begin{array}{cc|c} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ 0 & 0 & 1 \end{array}\right) \tag{2.5}$$

which is equivalent to

$$y = Ax + b \tag{2.6}$$

With this setup, a variety of different standard transformations can be created: For this thesis, the horizontal translation is the most important one. With this translation, the placement of image segments on a bigger canvas can be realized while keeping differentiability. Both v_y and v_x can be used as parameters for moving the segment on a blank canvas around. When applying the translation for multiple segments individually and then stacking the resulting images on top of each other, multiple segments (e.g. UI elements) can be arranged on a canvas.

2.2 Convolutional Neural Nets

This section provides the reader with high-level concepts related to CNNs and briefly discusses the necessary background information to understand their usage in this thesis. CNNs are primarily used throughout this thesis as predictors for various metrics for which a given UI is optimized for. The models output is subsequently used as a cost/loss function used in the gradient descent process.

Originally presented in LeCun et al. [16], CNNs are a class of Neural Networks (NNs) which have been modeled after the visual cortex of animals and are designed to learn spatial hierarchies in images. Although the concept of CNNs dates back to the 1990s, the emergence of powerful GPU architectures and larger datasets have helped CNNs to gain popularity in most vision-related machine learning tasks. After AlexNet was presented in 2012 by Krizhevsky et al. [17], which achieved a new state-of-the-art score in the popular image recognition challenge "ImageNet Large Scale Visual Recognition Challenge" (ILSVRC) [18], CNNs have become the de-facto standard for object detection tasks.

2.2.1 Neural Nets

To understand the concept of CNNs, a basic understanding of NNs is necessary. In essence, a NN consists of multiple matrices, called "layers", which are made up of a fixed number of trainable weights. These layers are then chained together and an input value, often also a matrix, is multiplied with each layer sequentially [19]. This process can be viewed as if each weight in a layer was a singular entity, sometimes called a "Neuron", which has a connection to each Neuron in the following layer. This connection is the trainable parameter, which is just a factor that the output of the neuron gets multiplied with, before it is passed to the next layer. In each neuron, all incoming "signals" are then just summed up. What is taking NNs beyond just being a more complex linear calculation, are activation functions [19]. These activation functions introduce nonlinearity into the NN when they are added to the output of a layer. While a lot of different activation functions exist, one of the more common ones is the sigmoid function [20], which maps all x values into the interval $[0, 1]$, which can be seen in Figure 2.2. This nonlinearity

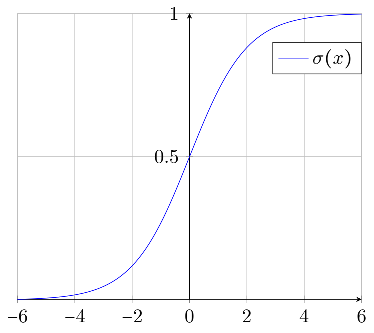


Figure 2.2: Sigmoid function

is what allows NNs to learn complex non-linear relationships in data. Additionally to these simple NNs, larger models, called Deep Neural Networks, with essentially the same architecture are used for more complex problems [19].

2.2.2 Architecture

Instead of the common linear layers (sometimes referred to as "fully connected layers"), of which common NNs are constructed, CNNs consist of convolutional layers. These layers consist of a learnable filter, known as the "kernel", which revolves over the input image, during which the dot product of each individual pixel, along with the pixels around it, with the kernel is computed and passed to the next layer. Through this process, a feature map is created which is able to capture high-level spacial data, like edges, in images. Similarly to conventional NNs, activation functions are used to introduce nonlinearity into the model. For CNNs, the most commonly used function is Rectified Linear Unit (ReLU):

$$\text{ReLU} = \max(0, x) \tag{2.7}$$

Since most CNNs are used for classification tasks, the final layer in such models is often a softmax layer, which converts the raw values into probabilities, representing the likelihood of each class.

2.2.3 Common Convolutional Neural Net Architectures

Since the inception of AlexNet [17], multiple more advanced model architectures have been presented. These include:

ResNet: Presented in He et al. [21], ResNet (short for "Residual Networks") introduced the concept of residual layers, which create connection skipping layers, enabling the training of very deep networks, normally suffering from vanishing gradients.

VGG-Net: Originally presented in Simonyan and Zisserman [22], VGG-Net pioneered the usage of 3×3 kernel sizes throughout the network and achieved state of the art scores in common object detection challenges, such as ILSVRC [18].

GoogLeNet: Presented in Szegedy et al. [23], GoogLeNet, codenamed "Inception", introduced the concept of chaining multiple different filter sizes in order to learn multi-scale features.

By removing the final layer in these models, which produce the classification logits, and replacing it with a new layer that only has one output value, these pre-trained models can be fine-tuned for a different metric, such as aesthetics. This is the primary way CNNs are used in this thesis.

2.3 Diffusion Models

This section provides the reader with the concept of Diffusion Models (DMs) and necessary background information to understand the associated concepts. As DMs are not integral to the key results of this thesis, only a brief overview will be given, allowing the reader to have a general intuition for what diffusion is.

2.3.1 Introduction to Diffusion Models

DMs, as introduced originally by Sohl-Dickstein et al. [24], are a category of generative models that use two Markov chains to transform a data distribution, like Gaussian Noise, into a target data distribution. This is done in two distinct phases where noise is first subsequently added to the data and after that a denoising process, which has been trained on removing the noise again, removes it again. This denoising process has been optimized during the training phase by generating random noise, overlaying it on top of an existing image and training a model to predict this "noise residual" which has to be removed from the combined image in order to retrieve the original image again. A reconstruction loss is used to compare the original image and the image with the noise residual removed. With this two-phased process, DMs can generate data fitting to the desired target distribution.

2.3.2 Architecture

As previously described, DMs generate data by first destroying structure in existing data and then trying to reverse this process by removing noise [24]. Both of these processes can be thought of as trainable markov chains [25].

By keeping the steps in the noise adding process T relatively small, the otherwise complex task of predicting the previous data structure in the reverse process becomes manageable [24].

Generally speaking, the forward markov chain is called the "noise scheduler" and the reverse process is called the "denoiser".

2.3.2.1 Noise Scheduler

The noise scheduler is the forward process of a diffusion model and what controls the noise addition in each step. During training of the model, this part of the model learns the amount of noise added to the image in each step.

Mathematically speaking, the noise scheduler controls the variance of gaussian noise added in each timestep [25]:

$$x_t = \sqrt{\overline{\alpha_t}} * x_0 + \sqrt{1 - \overline{\alpha_t}} * \epsilon_t \tag{2.8}$$

with x_0 being the original image, x_t the image with added noise up to timestep t and $\overline{\alpha_t}$ the cumulative product of α_s up to timestep t. The most important part is e_t, the gaussian noise sampled at timestep t with variance β_t. The core parameter of the noise scheduler is the variance β_t, defining the noise scheduling in each step with

$$\beta_t = 1 - \frac{\alpha_t}{\alpha_{t-1}} \tag{2.9}$$

Different noise scheduling strategies exist, which include linear or cosine trajectories for the noise variance in each step [26].

2.3.2.2 Denoiser

The denoising process is effectively the inverse of the noise scheduler. It consists of iteratively removing the noise again, which was added in the previous steps. To do this, the noise residual is predicted in each step and then removed from the image:

$$x_{t-1} = \frac{x_t - \sqrt{\beta_t} * \epsilon_\theta(x_t, t)}{\sqrt{\overline{\alpha_{t-1}}}} \tag{2.10}$$

with $\epsilon_\theta(x_t, t)$ being the prediction of the noise residual for a timestep t. This step is done until $t = 0$ and the final image is retrieved again.

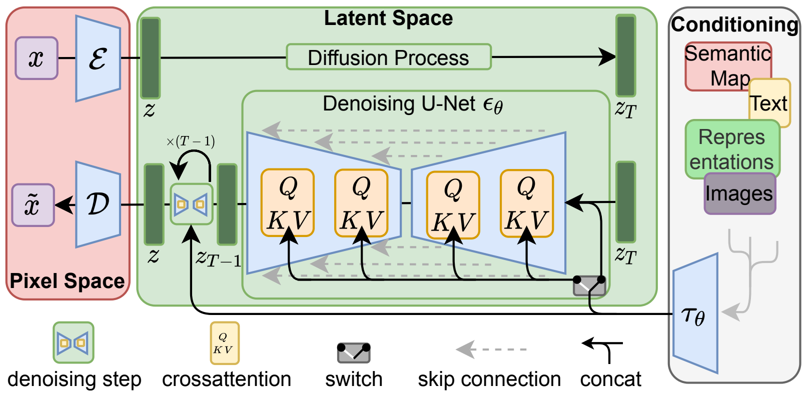


Figure 2.3: StableDiffusion architecture, reproduced from [1]

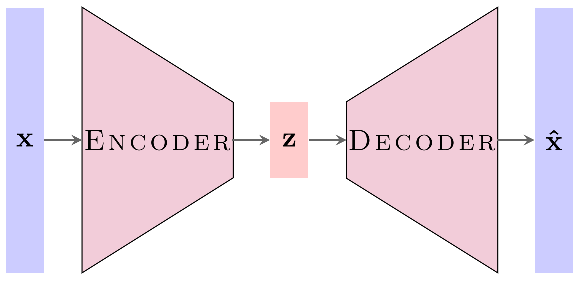


Figure 2.4: Conventional Auto Encoder

2.3.3 Stable Diffusion

Presented in Rombach et al. [1], StableDiffusion is one of the most widely known diffusion model architectures. While its architecture, shown in Figure 2.3, contains multiple additional elements and is more complex, it still relies on the basic concepts described in Section 2.3.2. StableDiffusion will be the primary model used in experiments leveraging DMs throughout this thesis.

2.4 Variational Auto Encoders

This section presents VAEs and provides the reader with the necessary background information to understand how VAEs work and how they are leveraged in this thesis.

As the name implies, VAEs are similar to conventional Autoencoders. The primary goal is to translate some data X into a latent space and then reconstruct X from this latent space, usually denoted as $\hat{X}$. In most Autoencoder applications, the latent space z is fixed and defined prior to training. Conventional Autoencoders can be seperated into two parts: (1) the encoder and (2) the decoder. Both of these modules are NNs which subsequently translate the input data into a latent space. The decoder can be viewed as the inverse of the encoder, both figuratively and in a technical sense. However, it is worth to note that they do not share the same weights. Conventional Autoencoders are usually used for graphical data like images. Hence, the model architecture mostly consists of convolutional layers.

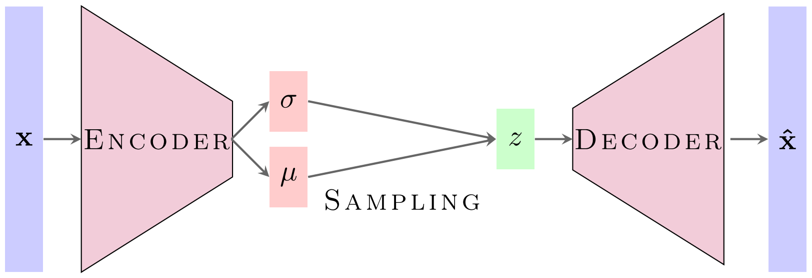


Figure 2.5: Variational Auto Encoder

Figure 2.4 illustrates the encoder and decoder part as well as the input data X and decoded data $\hat{X}$. The delta between X and $\hat{X}$ is considered the reconstruction loss during training and the main metric to be optimized for. While this model architecture is well suited for the task of mere reducing the dimensionality of input data, it has a major drawback when trying to create new data, i.e. sample from the latent space. Since the Autoencoder has been trained on specific training data, generating new data is not something it was optimised for. If the latent space is sufficiently large, it might even lead to overfitting [27], which, for this model architecture, means in the case of images that each training example gets a discrete point assigned in the latent space, effectively creating a dictionary, rather than an encoder. This causes the latent space to consist of some discrete data points corresponding to the training data but without any meaningful data in between. This data in between is generally regarded as "noise" since it does not produce any meaningful data when passed to the decoder.

This is where the VAE shows large performance improvements over a conventional Autoencoder [28]. The key difference between the two models is the lack of a fixed latent space and the usage of a distribution instead. While the VAE architecture also uses the two-part encoder and decoder architecture, the two elements are not directly connected. Instead, the encoder has two outputs: (1) a mean μ and (2) a standard deviation σ. These describe a distribution inside of the latent space, the dimensionality of which is again predefined prior to training. During training, a datapoint is passed to the encoder, which produces the parameters for a gaussian distribution. A data point from this distribution is then sampled and passed to the decoder, after which the reconstruction loss is calculated. This causes the VAE to be inherently nondeterministic. Additionally to the expected capabilities, encoding and decoding with minimal reconstruction loss, the VAE also learns the distribution of all possible z vectors, that is, all vectors that generate plausible data points when passed to the decoder. This distribution is considered to be the prior distribution $p_\theta(z)$. The encoder subsequently learns the posterior distribution $p_\theta(z|x)$ and the decoder learns the likelihood distribution $p_\theta(x|z)$. To generate a new datapoint, a vector z^* is sampled from $p_\theta(z)$, and a decoded x is created via the likelihood distribution $x = p_\theta(x|z = z^*)$. Since the *actual* parameter θ is unknown, only an approximation ϕ with distribution $q_\phi(z|x)$ can be reached during training. To get closer to the actual parameter during training, the Kullback Leibler Divergence would be used. However, this is not computable since the real parameter is still unknown. Instead, the *Evidence Lower Bound (ELBO)* function is used as a loss function [29]. It is comprised of (1) the reconstruction loss and (2) the KL Divergence. To reiterate, while the actual KL divergence would be:

$$D_{KL}(q_0||p_0) = E_{q\theta}[log\frac{q_\theta(z|x)}{p_\theta(z|x)}] \tag{2.11}$$

Instead, the ELBO function is used [30]:

$$\text{ELBO} = E_{q\theta}[log\, p_0(z|x)] - E_{q\theta}[log\frac{q_\theta(z|x)}{p_\theta(z)}] \tag{2.12}$$

With the first part being the reconstruction loss:

$$L_{\text{Recon}} = E_{q\theta}[log\, p_0(z|x)] \tag{2.13}$$

During training, the distribution of z is assumed to be $N(0,1)$, which additionally keeps the latent space from getting too complex, avoiding overfitting.

The key difference between conventional autoencoders and VAE leads to an additional challenge during the training phase. Since the encoder does not produce a latent vector directly, but rather the parameters to a distribution, the reconstruction would be non-deterministic, due to the sampling. This would lead to the process not being differentiable, making the training of the VAE impossible. To combat this, the training phase employs a method commonly referred to as the "Reparameterization Trick" [29, 30]. Instead of sampling z via

$$z \sim N(\mu, \sigma) \tag{2.14}$$

the following trick is used:

$$z = \mu + \sigma \odot \epsilon, \text{ where } \epsilon \sim N(0,1) \tag{2.15}$$

This leads z to still be random and additionally conform to the distribution. Finally, the sampling process is differentiable with regards to μ and σ.

Chapter 3

Related Work

3.1 Prediction of Perceived Aesthetics

Predicting how users perceive the aesthetics of UIs has been the subject of multiple research efforts [31, 32]. de Souza Lima et al. [32] presented a CNN based on fine-tuned versions of RESNET, originally presented in He et al. [33], which can reliably predict how users will perceive the aesthetics of a user interface. The data that the model in de Souza Lima et al. [32] was trained on consists of parts of the RICO [34] dataset and a newly created dataset which contains screenshots of applications created with MITs AppInventor.[1]

A similar approach was presented in Leiva et al. [31], where different CNN architectures are compared. Fine-tuned versions of the popular RESNET [33] and DENSENET [35] architectures and a custom CNN model architecture were evaluated, with the custom architecture outperforming the fine-tuned models. The custom-architecture-based model, which was ultimately chosen as the best-performing one, was modified to include factors such as gender and age into its prediction to improve its performance further. The dataset used to train the model is part of the LabInTheWild [36] project, which conducts large-scale online experiments, where users answer different questions to create large datasets. The original objective of the experiment was to allow users to compare their own tastes with other parts of the world.[2] As the dataset created during this experiment contains labels (score 1-9) of roughly 400 websites and 32k participants and is therefore fitting for usage as training data in this task.

Both of the showcased approaches presented in Leiva et al. [31] and de Souza Lima et al. [32] provide models which are beneficial for a number of different research areas. In this research, they are primarily used as grading mechanisms for generated UIs.

3.2 UI Generation

In a recent literature review in the area of AI assisted UI generation by Lu et al. [37], the work in this research area is divided in the four sections of the double-diamond model, proposed by the Design Council [10] (Figure 3.1).

Since the main focus of this research is on the "Develop" phase of the Double Diamond process, the emphasis is on related work in this specific phase.

A selection of previous research efforts in the area of automated UI generation, which can be divided into Generative Adversarial Network (GAN)- and DM-based is showcased in the following sections.

[1] https://appinventor.mit.edu

[2] https://www.labinthewild.org/studies/aesthetics/

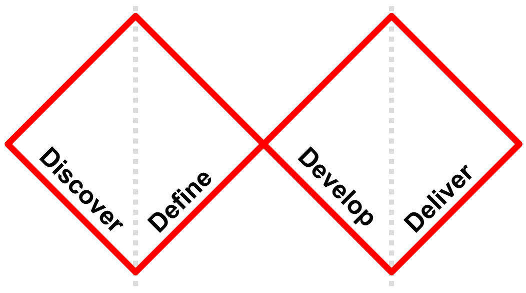


Figure 3.1: The Double Diamond Model as proposed by the Design Council

3.2.1 Generative Adversarial Network Based UI Generation

Since the emergence of GANs [38] and their use for image generation has been showcased, various attempts have been made to use the results for UI generation [11]. In LayoutGAN, Li et al. [11] proposed a novel GAN which models relationships between graphic elements of UI to synthesize new UIs. The model produces a wireframe-aligned output which can then be optimized using a CNN based discriminator.

3.2.2 Diffusion-Based UI Generation

Since Diffusion models gained traction after their initial presented applicability in image generation in Rombach et al. [1], multiple research efforts have gone into exploring their capabilities in UI generation [2, 39]. While the increase in popularity of DMs has largely been due to the emergence of image-creating models such as in Rombach et al. [1], most approaches leveraging diffusion for UI generation are using a discrete diffusion process, which, as the name suggests, does not map data into a continuous distribution, like image data, but rather a discrete feature space. Such a space consists of different properties, such as UI element type, position, and size. This approach is presented in Zhang et al. [2] and Inoue et al. [39]. The approach by Zhang et al. [2], called LayoutDiffusion uses a feature vector to represent a single UI. This feature vector contains sizing and position information as well as the category for each individual element in the UI. It is then used in a forward corruption and backward denoising process like it is common for DMs. The discrete DM is trained on the RICO [34] dataset. Some selected examples of the unconditional layout generation can be seen in Figure 3.2.

3.3 Non-Subjective Metrics

Besides the previously described metric, aesthetics, which relies purely on human interrogation for labeling UI examples, there are a number of non-subjective metrics that can be computed purely with the known UI layout. The two most widely known metrics are Intersection over Union (IoU) and alignment, which have been used by similar research efforts, such as in Zhang et al. [2]. Both of these metrics are calculated pairwise for two UI elements and then averaged across the entire UI. These metrics are often used as benchmarks for automatic UI generation since they are straightforward to calculate for new data, and their respective "normal" expected values are known for large datasets like RICO [2, 39, 40]. While these metrics are usually easy to calculate and optimize for, they

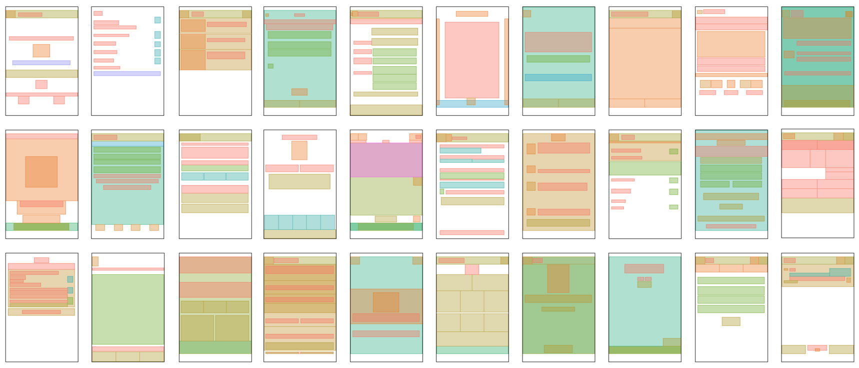


Figure 3.2: Unconditional Generation by LayoutDiffusion, reproduced from [2]

do not always align with the actual perceived UX [40]. Due to this, it is desirable to optimize directly for perceived aesthetics. However, this is usually harder to do, since for non-subjective metrics, the task can be broken down to solve a simple equation, but for perceived aesthetics a robust prediction model as well as a capable optimizer are necessary.

3.4 UI Datasets

While multiple different UI datasets, such as WebUI [41], MUD [42] and Screen2Words [43], have been presented in the past, this thesis will largely focus on the RICO dataset [34], due its size and the amount of metadata included in the dataset.

3.4.1 RICO Dataset

The RICO dataset [34, 44] is one of the largest datasets in the UI space. Deka et al. [34] proposed a mining infrastructure that can automatically download applications from one of the largest app marketplaces for the Android OS, Google's *Play Store*.[3] This agent then runs various simulations of user interactions with the applications. This process generates a large dataset of user interface traces and layouts. The advantage of this research is that the dataset is automated and therefore continuously growing. While the original research is already seven years old, its results are still used in a number of modern different research fields and approaches, like traditional UI generation [45] and even Large Language Model (LLM) assisted generation [46], as well as UI design choices studies [47].

As the presented automated agent has direct access to the application's source code, allowing for a precise understanding of the code and markup, the dataset not only provides graphical elements of the applications but also a hierarchical definition of the UI elements. Thus, the dataset can also be used for tasks that require segmented UIs. The dataset differs from other UI segmentation datasets, such as in the WebUIProject [48], as no precision is lost due to imperfect models since the agent has direct access to the hierarchical segmentation. As a result, the dataset is fitting for tasks where one objective is the rearrangement of UI elements.

[3] https://play.google.com

Additionally, Deka et al. [34] presented an autoencoder that transforms a layout of a user interface into a 64-dimensional latent space, from which a similar UI can be retrieved again, when "reversing" the autoencoder.

3.4.2 Annotations for RICO

de Souza Lima et al. [32] presented a NN based on a fine-tuned version of RESNET, which was originally presented in He et al. [33], which can reliably predict perceived aesthetics of UIs. The presented research includes both a new dataset and a model trained on it. The dataset consists of Screenshots obtained by the researchers and screenshots previously presented in RICO [34]. Both the model and the dataset are useful resources when considering automated layout generation, especially when using automated grading and evaluation.

Chapter 4

Methods & Results

This section provides the reader with four different experiments designed to answer the stated research questions. The last three experiments are similar and directly built on top of each other. All of the experiments involve

4.1 Diffusion-Based UI Generation

As generative models like StableDiffusion [1] have become ubiquitous, the challenge to create images from prompts that satisfy the users' intention has become increasingly more visible. This, among advances in other generative AI fields, has led to a separate field known as "prompt engineering". Past research has shown that this process can be partly automated by leveraging a metric representing the users' objective. Such a metric may be aesthetics. By gradually improving the input to a generative model, *the prompt*, with regard to the metric, represented by a predictor model, the generated image gets closer to an image that satisfies the metric. This process can be considered a generic gradient descent pipeline [49].

The objective of this thesis is to improve existing UIs, guided by a metric representing perceived aesthetics. This problem can be considered a special instance of the aforementioned, already-researched problem. The difference is that the objective is to improve an existing image and not create an entirely new image. Generative models like StableDiffusion not only support the *Text2Image* task, in which a new image is created based on random noise but also the *Image2Image* task, in which a new image is created based on some initial latents that are retrieved from an existing image. DMs such as StableDiffusion do not operate, directly on the pixel space but rather an intermediate and latent representation, called latents. The Text2Image pipeline of any SD-based model can easily be adapted to optimize an existing UI.

In this experiment, we develop a custom generative model based on the StableDiffusion [1] architecture and use its *Img2Img* pipeline to optimize a screenshot of an existing image, guided by an aesthetics predictor. This experiment can be divided into **4** tasks: (1) dataset creation (2) model training (3) metric selection and (4) evaluation.

4.1.1 Dataset Creation

Text2Image models primarily learn the connection between text tokens and images by using a model like CLIP [50, 51, 52], which maps both in the same latent space. This model ensures a close cosine similarity between two vectors corresponding to close semantic meaning. Therefore, a dataset containing pairs of text and images is necessary for training such a model.

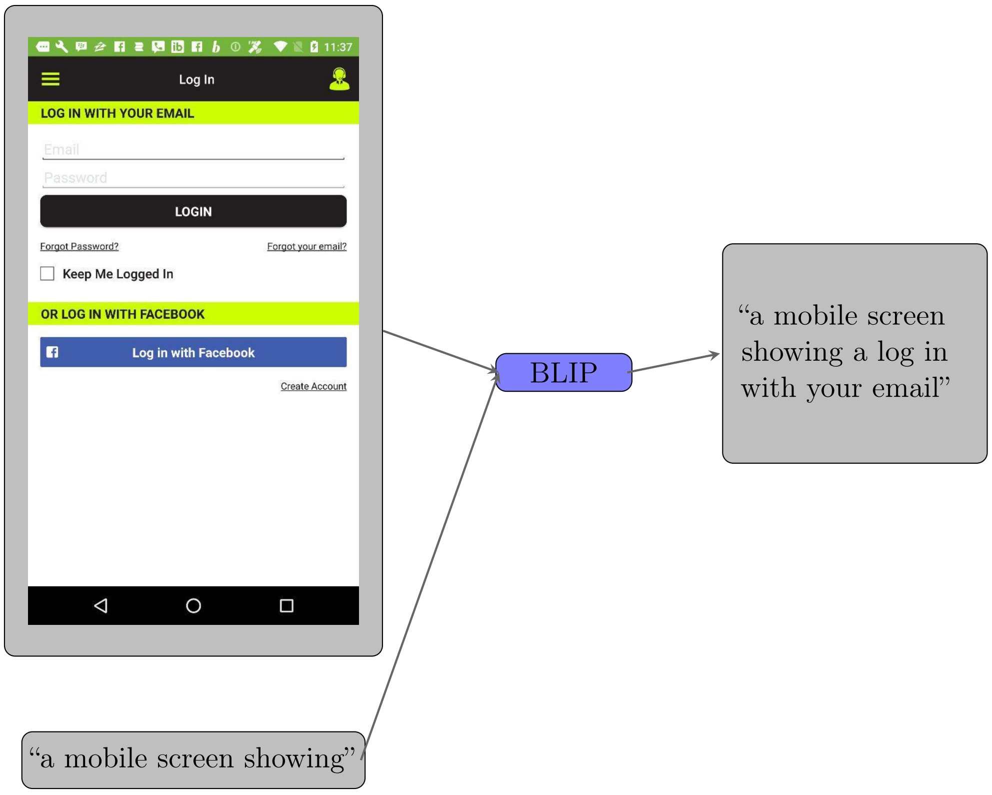


Figure 4.1: Conditioned captioning with BLIP

Due to its performance and amount of past research, StableDiffusion [1] was chosen as the *Text2Image* model for this experiment. It is not feasible to train the entire model “from scratch” specifically for this task, as the amount of parameters is too large. Instead, a process known as fine-tuning, originally presented for SD in Ruiz et al. [53] is used to both leverage the existing model weights but also adapt a model to a specific task, in this case generating UIs. The fine-tuning methodology consists, in essence, of retraining the U-Net component of SD. However, while the dataset does not have to be as big as the original dataset used to train the model (which has 5 billion examples [54]), it still surpasses what a single human could reasonably annotate.

As all of the larger UI datasets, such as RICO [34], WebUI [41], MUD [42] and Screen2Words [43] were created for a different task and do not contain any text annotations, we use an automatic image captioning approach to create the dataset. Specifically, the image-text retrieval model BLIP, originally presented by Li et al. [55] is used to create conditioned captions for images from the RICO dataset presented in Deka et al. [34], which contains screenshots from various android apps. BLIP allows for conditioned captioning where an initial string is given and the model completes it. In this case, the initial string chosen was *“a mobile screen showing”*. Through this process, we created a dataset of 66k examples. We have made the dataset publicly available.[1]

[1]https://huggingface.co/datasets/mowoe/rico-captions-blip-large-conditioned

4.1.2 Model Training

With the created dataset, the model *Stable Diffusion V1.4* [1] was finetuned using the described process. We have made the model publicly available.[2] The trained model can be used for simple *Text2Image* applications. An example of the prompt-to-image capabilities can be seen in Figure 4.2.

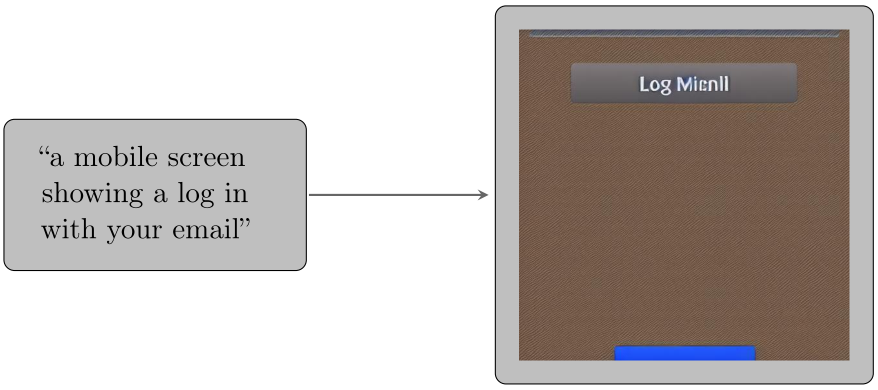


Figure 4.2: Generated image with fine-tuned model

4.1.3 Metric Selection

The primary target of this experiment and thesis is to optimize existing UIs with regards to some metric. To allow for automatic optimization of some parameters through a gradient-descent process, the metric calculation needs to be fully differentiable. As such, a human feedback mechanism is not feasible. Instead, a predictor model trained on perceived aesthetics is used. In this case, the model originally presented in Leiva et al. [31] is used.

4.1.4 Evaluation Setup

To optimize a given UI, the *Img2Img* functionality of the fine-tuned SD model is used. The UI is represented as a screenshot and encoded into latents via the SD internal VAE. Additionally, the *Img2Img* functionality takes a prompt which will be used by the SD model to generate data on top of the initial latents. This prompt can be considered to be the trainable parameter in the gradient descent optimization process. To initialize this parameter it is key to choose a value that modifies the image only minimally to ensure the optimization process will in fact start at the initial given UI. As it is not feasible to search for this optimal initial value for every new UI, an approximation will be used. The initial prompt will be approximated by interrogating the BLIP model, similar to how it was done for the dataset creation, showing in Figure 4.3. After the initial captioning, the retrieved prompt is considered to be a trainable parameter and together with the previously presented metric, an optimization via gradient descent is performed.

[2]https://huggingface.co/mowoe/stable-diffusion-v1-4-rico-blip-large-conditioned

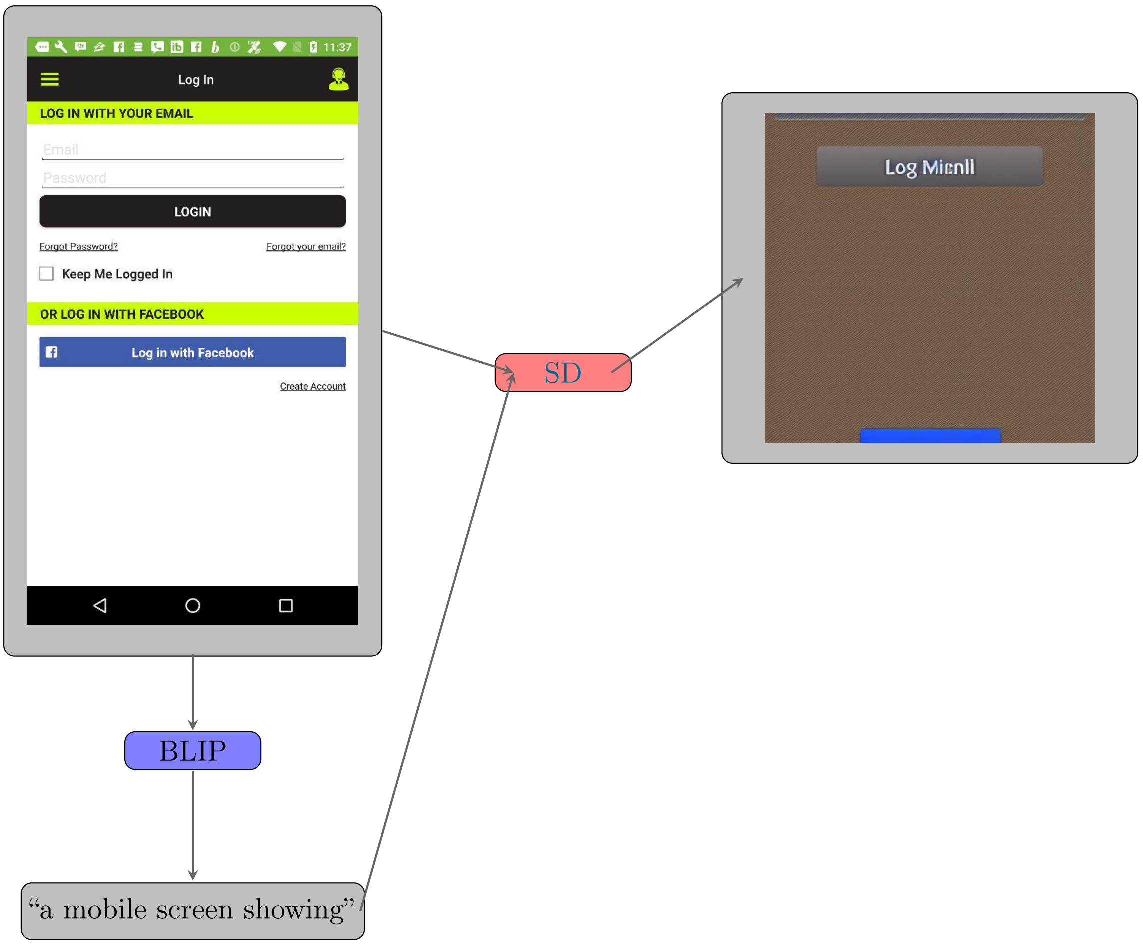


Figure 4.3: Optimization process with automated initial prompt captioning

4.1.5 Evaluation

We executed the optimization process described in Section 4.1.4 for multiple examples, however only one is discussed here as an example. The results can be seen in Figure 4.4. There are multiple issues visible in this example. First, the image from the initial step is already not a real UI anymore, since the text and logos are distorted and the color scheme seems to be not matching. This means that after the first optimization step, the generated UI is already not representing the initial image anymore. Due to the nature of this optimization process, it is unlikely, that the generated image will get closer to the initial image again. This is evidently suboptimal for an optimization problem since the original image does not get optimized, but rather a completely new image is generated. While this might be expected for a low-grade DMs, state-of-the-art models, like SD, usually achieve a performance, that the generated image is visually close to the original image. While the other issue is that the final image (from step 49) does not represent a real UI anymore, this is more likely to be able to be improved by different models, optimization techniques, and other hyperparameters. A common metric for image generation models is the Fréchet Inception Distance (FID). Essentially this metric compares the distribution of generated images with a ground truth distribution, which is the distribution of the "real" images, which are real UIs in this case [56]. Without having to calculate the actual FID, we know that it is below what is considered to be a good value since the generated images only very faintly represent a real UI, thus the two distributions *have* to be far apart. Due to these described shortcomings, it was decided to not continue the approach of diffusion-based image generation.

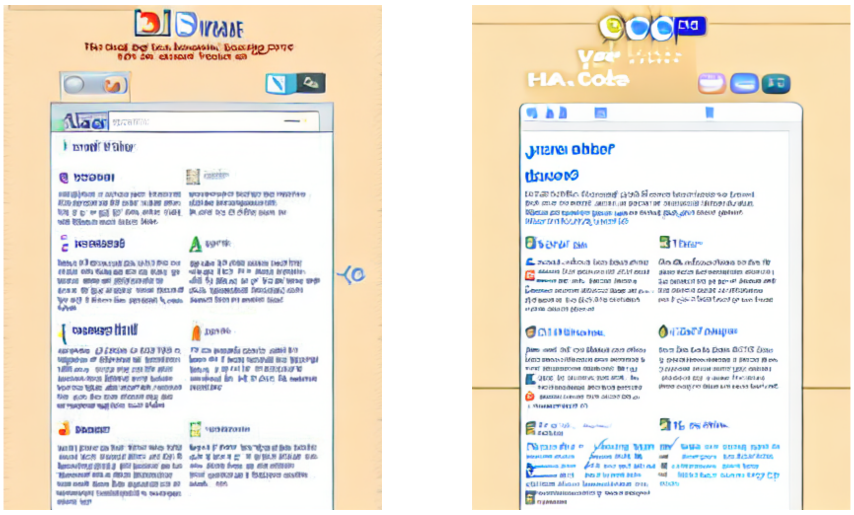


Figure 4.4: Optimization Step 1 and 49

4.2 Optimizing Layout via Latent Space Representation

In order to decrease the complexity of the optimization problem, the space, or rather its complexity, in which a given UI is optimized, has to be decreased. Since the optimization directly in pixel space, like it is done in Section 4.1, gives a potential optimizer a space possibly too large to meaningfully navigate through, optimization in a more semantic approach could provide benefits. A more semantic approach being the optimizable parameters actually having meaning on their own, e.g. position, size, and color, as opposed to pixels, which don't hold any information on their own. This experiment solely focuses on optimizing the layout of a UI. The goal is to create a process that receives a number of UI elements (buttons, text inputs, images, etc.) and outputs a finished layout on a blank canvas for these elements. The process again leverages a gradient descent style optimization process which uses a prediction model as the metric to be optimized for.

4.2.1 Data

The main source of data for this experiment is the RICO [34] dataset. It contains screenshots of mobile application UIs along with semantic information about the elements they consist of. This includes information about at which position the elements are to be found. This allows for an automatic segmentation of the user interface and therefore the deconstruction of a given UI into its components. The base data for this experiment consists of segmented screenshots, i.e. smaller images that contain only one UI element. During optimization, these smaller images will be placed onto a blank canvas and moved around according to the optimization of some metric.

4.2.2 Metric

As the main source of data in this experiment is mobile UIs, an aesthetics predictor specifically designed for mobile UIs is necessary. In this case, the metric and predictor presented in de Souza Lima et al. [32] was chosen. This metric has been created by conducting a user study in which 2k screenshots of mobile UIs were rated for their aesthetics on a scale of 1-5, with 5 being the highest rating. The rating for the UIs is then averaged across all participants for one UI and the resulting value is then considered to be the score for a UI. Additionally, de Souza Lima et al. [32] also presented a predictor model specifically for this dataset. While multiple different model architectures were explored, the best-performing model was found to be a finetuned version of ResNet50 [33]. The fine-tuned model is modified to only have a singular output logit instead of the full output layout. This final layer is the only part that we specifically trained.

4.2.3 Evaluation Setup

To optimize a given UI, the positions of individual UI elements, represented as a vector of their respective x and y coordinates, are considered as trainable parameters. Together with the previously described metric, a gradient-descent style optimization process is used to optimize the parameter with regard to the metric. Since the model responsible for predicting the metric is trained on screenshots of UIs, the given UI needs to be reassembled in each training step. Since the optimization process needs to be differentiable from start to finish, this also applies to the UI reassembly. Arranging smaller images on a larger canvas can be done using regular matrix indexing when considering the images as matrices. However, this process is not differentiable with regard to the index itself, which is the trainable parameter in this case. To circumvent this, an affine transformation is used to ensure differentiability. An affine transformation uses a 3×3 matrix, called

the affine matrix, which contains the parameters for the transformation. Originally, this process is used to perform transformations such as shearing, rotating, and scaling. However, through correctly chosen parameters, the transformation can be used to place a smaller image onto a bigger canvas and control the position in it. As far as we are aware, this technique has not been presented before. Through this process, the transformation is applied to each UI element (now an image) individually, creating a number of images with the dimensions of the target canvas size. To fully reassemble the UI, the images can be summed up and clamped down to the maximum values for pixels. This process ensures that the final metric, the predicted aesthetic score, is differentiable with respect to the parameters controlling the positions, in this case, the affine matrices.
For the final evaluation, two different approaches were used and compared:

Initialization through initial positions: Since the RICO dataset is made up of real mobile UIs, the positional parameters for each element are already known. In this approach, these known parameters were used to initialize the trainable parameters. This can be considered as the task of optimizing an already existing user interface with regard to its layout.

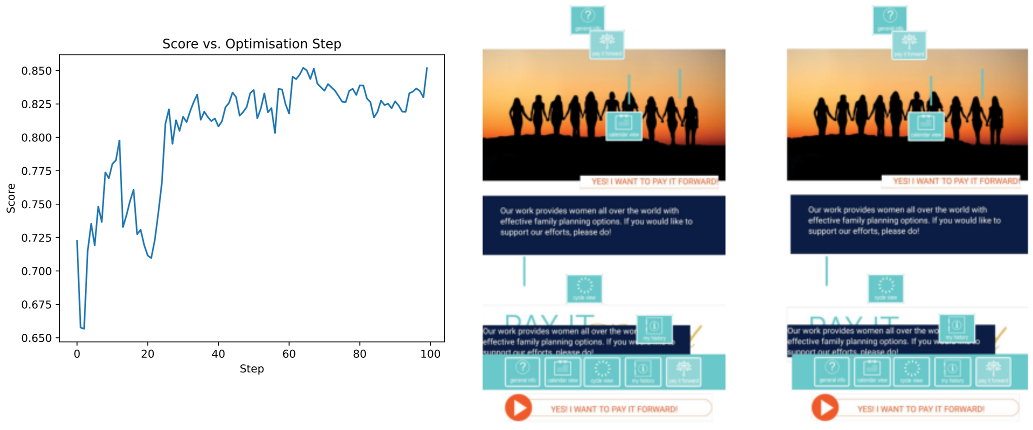
Random initialization: Alternatively, the trainable parameters can also be initialized with random values. This approach is closer to the objective of creating a UI without any prior information or an already existing layout. While this does provide a more generalized approach to the problem, random initialization poses difficulties when elements get initialized on top of each other. In this case, there may not be a clear gradient out of this state since the hidden element is not visible to the aesthetics predictor. To circumvent this issue, we modified the random generator to only generate positions where no other element is already placed.

4.2.4 Evaluation

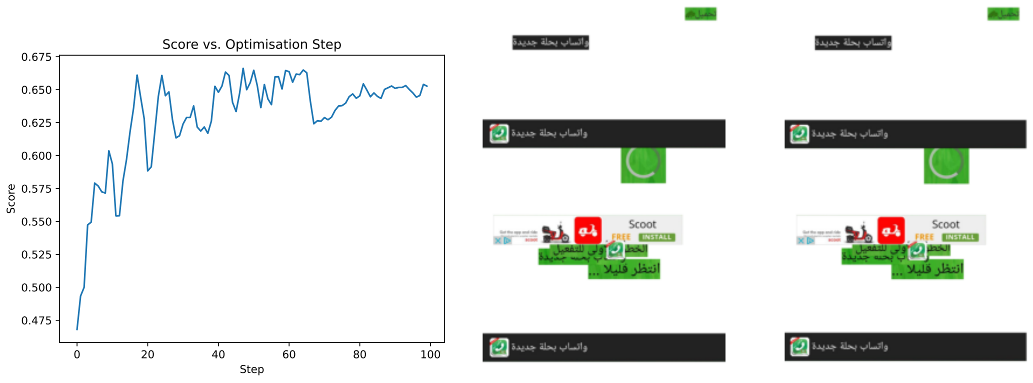
While both approaches described in Section 4.2.3 have been tested, the results are very similar. While the optimization and score themselves show an improvement over the course of the optimization, the actual images do not show any significant changes to the human eye. We randomly selected 20 examples to inspect manually, all of which had the described issue. This is visible for three selected examples in Figure 4.5.

While it is challenging to find out the exact reason for this behavior, two issues may be the most prevalent ones. First, the prediction model for the metric is susceptible to an accidental adversarial attack, sometimes also referred to as “Specification Gaming” or “Reward Gaming” [57]. In this situation, the optimizer exploits weaknesses in the model to increase the score with only minuscule changes.

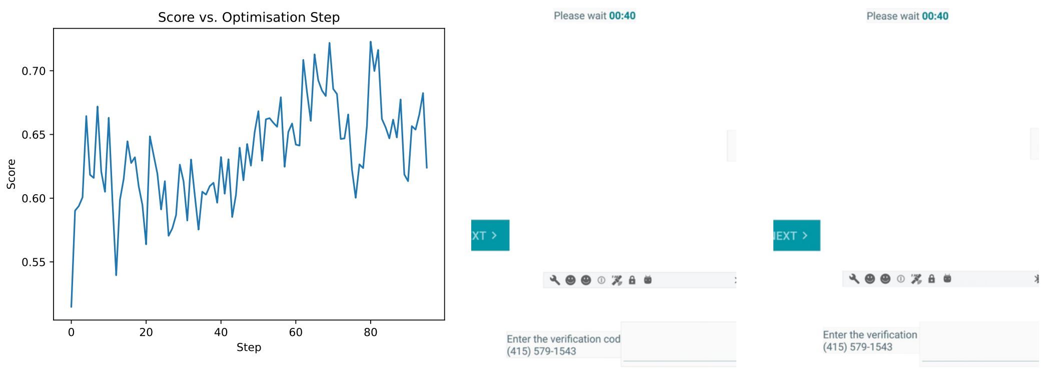
Adversarial attacks have been a long-documented phenomenon [58]. The most common type of adversarial attack involves some CNN with a classification task. In the attack, the model is “tricked” into predicting a different class than what would be correct for the image. This is done by overlaying a very specific filter on top of the image, called the perturbation, which has been specifically engineered for a specific class and image. While adversarial attacks are almost always undesired, the described common attack involves an attacker with (at least crafted) malicious intents, i.e. fooling a harmful content detection model [58]. In this case, there are no malicious intents, but the setup used in the experiment is similar to the one that would be used for crafting an adversarial task. This setup usually consists of an image in which the raw pixel values get optimized to



Score progression and initial vs. best image score (0.852 at i=64)



Score progression and initial vs. best image score (0.667 at i=47)



Score progression and initial vs. best image score (0.73 at i=65)

Figure 4.5: Multiple optimization examples

increase a given score (e.g. a class score in a classification task) without regard for actual meaningful changes. While in our setup, only the position of elements is changed and not directly the RGB values, this is still similar, since the optimizer has the ability to change individual pixel values, even if that entails a change in other areas too.

Second, the optimizer is only trained on real UIs and has never seen "fake" (i.e. random) UIs. This means that the space in which these broken/fake UIs lie is neither aesthetic nor non-aesthetic but rather "undefined" and the predictor model has trouble assigning correct values to this space. The shown examples represent a selection with hyperparameters that were deemed to be the best-performing ones. While a different set of hyperparameters as well as the amount of iterations during optimization might alter the results slightly, there is no indication that these would alleviate the described challenges. The learning rate (currently $lr = 0.05$) should not be any lower since the changes would be even smaller and not any higher since the change in loss is already meaningful enough. The second example in Figure 4.5 has a delta of around 0.2 between the start and end of the optimization, which would correspond to a full additional point given by each user on the 1-5 rating scale of the dataset. As there is virtually no real visible change in the UI, this decrease is incorrect.

4.3 VAE Latent Space

Since the predictor model used in Section 4.2.2 has been specifically trained to predict how users would rate aesthetics of a UI, it lacks the capabilities to predict aesthetics for images that do not represent UIs. The task the classifier is trained on can also be described as

$$p(im_{aesth}|im \overset{represents}{=} \text{UI}) = ? \tag{4.1}$$

but it is unknown how the classifier performs on a task like

$$p(im_{aesth}|im \overset{represents}{\neq} \text{UI}) = ? \tag{4.2}$$

To alleviate the challenges associated with this uncertainty, it would be desirable to ensure only images of actual UIs will be created during the optimization process. In this case, the optimizer would not run into an issue in which an image is optimized for a perfect score with regards to the aesthetics predictor, but the resulting image does not represent a UI, but rather something that might be described as an accidental adversarial attack. A common way to ensure that generated examples are kept to come from a target distribution is to use VAEs as an intermediate step. This is because the VAE learns the distribution of the data during training and can be used to check whether or not a latent vector is still part of the learned distribution.

4.3.1 Data

For this experiment, the latent space is not the positional vector from Section 4.2, but the latent vector from a VAE which has been trained to represent/encode a positional vector. The input vector X of the VAE consists of the relative positions of the 5 largest UI elements, which are not clickable, and the 2 largest UI elements which are clickable. Fixating this input size ensures no padding is needed in case a specific UI has a different number of elements. We decided to avoid any padding since this significantly complicates the VAE implementation. The amounts of elements were chosen to ensure the dataset is kept reasonably big without having to discard too many examples (we used 90% as a threshold) because of too few elements in the respective categories.

4.3.2 VAE Architecture

For the architecture of the VAE, we use a structure presented by Subramanian [59] with normal fully connected layers instead of the convolutional ones, since x is numerical data and not images. Figure 4.6 shows the final image of the VAE.

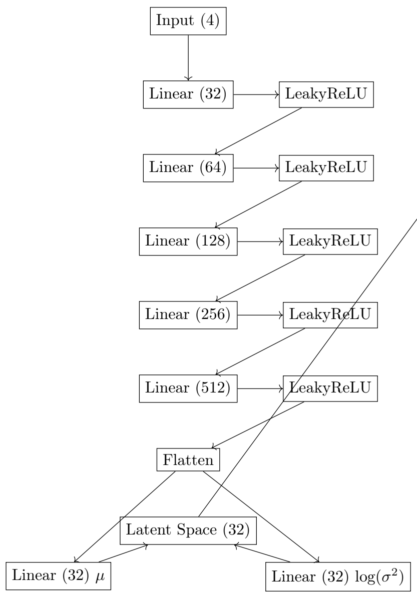


Figure 4.6: VAE Architecture

4.3.3 Evaluation Setup

To optimize a given UI with the approach described in this experiment, the setup is as follows: First, the vector x is extracted from the UI. Since the main source of training examples for this experiment is the RICO dataset, this data just needs to be transformed, as it is already part of the dataset, albeit in a different representation. After that, this vector is passed to the VAE and the resulting latent vector is used as a trainable parameter in a gradient descent pipeline. Additionally to the metric described in Section 4.3.3.1, which, inverted, is considered to be the loss, a second loss is used. This second loss is the Probability Density Function (PDF) of the multivariate normal distribution that has been learned by the VAEs. Due to the challenges described in Section 2.4, this distribution is assumed to have the parameters $N(0, 1)$. The PDF acts as a loss to penalize latent space vectors which are unlikely to come out of the trained distribution. This is intended to help the optimization process to achieve the goal of only creating “real” UIs. Both of the losses are weighted together to create one uniform loss.

4.3.3.1 Metric

Similar to the experiment described in Section 4.2.2, the metric used in this experiment is again the model and metric presented in de Souza Lima et al. [32].

4.3.4 Evaluation

The evaluation of this experiment is, again, presented with one specific example. First, one UI is selected out of the RICO dataset. This image can be seen in Figure 4.7a. It is worth noting, that this image is already a reconstructed image from segments, conforming to the constraints described in Section 4.3.1 (5 largest non-clickable elements, 2 largest clickable elements). This image is then passed to the VAE and the latent vector is optimized. During the optimization, some increase in the (predicted) aesthetics score is observable. However, the original goal of this experiment to only generate "real" UIs could not be reached. This issue is already visible after the initial translation of the image into the latent space and reconstruction from there to the image, as seen in Figure 4.7b.

The generated image does not represent a realistic looking UI and the VAE seemingly predicted the same position for all UI elements, which is not desirable. This means the VAE failed to properly learn the distribution of the data and the reconstruction loss of the VAE is too high for this task. We were not able to circumvent the issue, either by e.g. using a different set of hyperparameters, which were selected using a heuristic process, or by changing the VAE architecture. As this issue is common in VAEs applications, extensive research into mitigation techniques has been done, such as by Asperti and Trentin [60], whose work focuses on balancing the Kullback Leibler distance with the reconstruction loss. These two metrics are somewhat in conflict with each other and have to be balanced carefully. Asperti and Trentin [60] present a novel approach to calculate the optimal balance between the two metrics. While we explored a selection of the mentioned techniques presented by Asperti and Trentin [60], some room for additional exploration is still present.

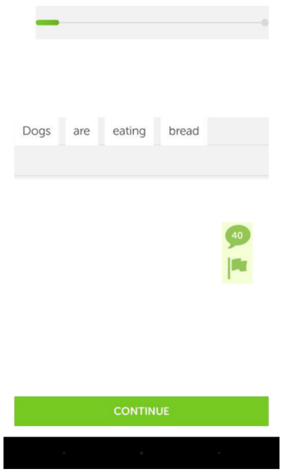
4.4 Positions as Latent Space: Intersection over Union and Alignment

This final experiment is designed to prove the validity of the gradient descent process on UI images while using positioning as the optimizable parameter. As one possible problem of the previous experiment is the metric signal being too weak, we aim to prove that with a different signal, the process itself is still viable. Instead of using a subjective metric like aesthetics, IoU and alignment are used. However, these metrics are still predicted by a CNN to ensure the same optimization process as in the previous experiments is used.

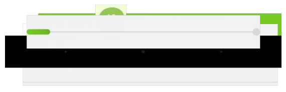
4.4.1 Metric

The metric used is a combination of IoU and alignment.

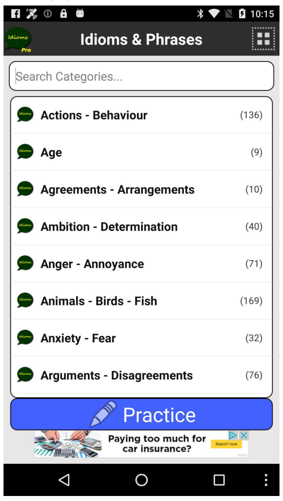
$$\text{Avg IoU} = \frac{1}{N}\sum_{i=1}^{N}\frac{|A_i \cap B_i|}{|A_i \cup B_i|}$$



(a) Initial Image



(b) Reconstructed image from VAE latent space



10:15

Idioms & Phrases

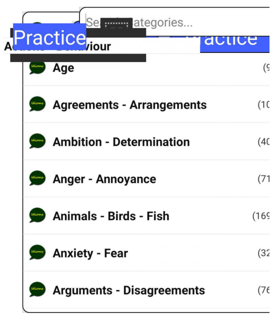
Search Categories...

Actions - Behaviour (136)
Age (9)
Agreements - Arrangements (10)
Ambition - Determination (40)
Anger - Annoyance (71)
Animals - Birds - Fish (169)
Anxiety - Fear (32)
Arguments - Disagreements (76)

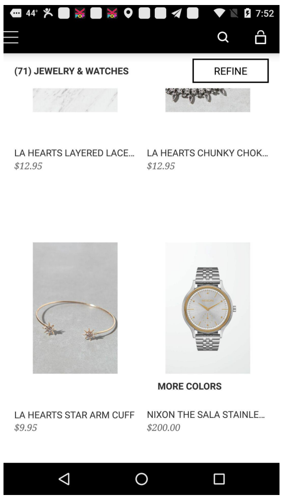
Practice

Paying too much for car insurance?

(c) Initial Image



(d) Reconstructed image from VAE latent space



7:52

(71) JEWELRY & WATCHES REFINE

LA HEARTS LAYERED LACE... $12.95

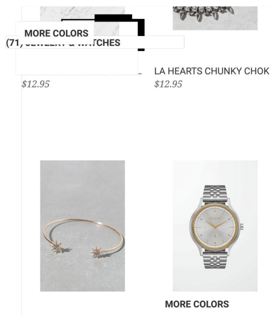
LA HEARTS CHUNKY CHOK... $12.95

LA HEARTS STAR ARM CUFF $9.95

MORE COLORS

NIXON THE SALA STAINLE... $200.00

(e) Initial Image



(f) Reconstructed image from VAE latent space

Figure 4.7: Initial and reconstructed image

with $|A_i \cap B_i|$ being the area of intersection of two elements and $|A_i \cup B_i|$ being the sum of their bounding boxes. To calculate the average alignment, the following formula is used:

$$\text{Avg Alignment} = \frac{1}{N}\sum_{i=1}^{N}\sum_{b=1}^{N} min(|i_{left} - b_{left}|, |i_{middle} - b_{middle}|, |i_{right} - b_{right}|)$$

It is used to calculate the average minimum distance between the most left, most right, or middle coordinates (on the x-axis) between a pair of two UI elements.

Since both of these metrics are calculated for two individual UI elements, the averages for all pairwise values are used for an entire UI. The data on which the predictor CNN model is trained is again based on the RICO dataset. In a preliminary step, average IoU and alignment are calculated for each example in the RICO dataset. As the range for average IoU and alignment is relatively large, instead of the raw values, normalized values in the form of min-max scaling have been used.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{4.3}$$

A difference from the prior experiments is that these metrics are considered to be better when they are lower, so no inversion for use as a loss is necessary.

4.4.1.1 Model Architecture

Since the input data for the model are images, a CNN is used as a model. The full model architecture is visualized in Figure 4.8. It consists of two convolutional layers with a kernel size of 3×3. The final fully connected layer has two outputs since both of the described metrics are predicted separately.

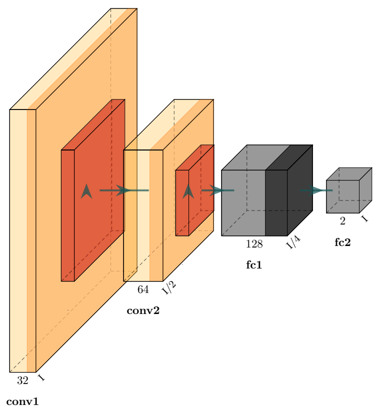
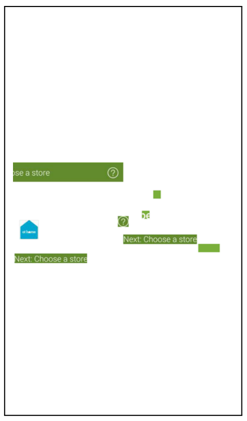


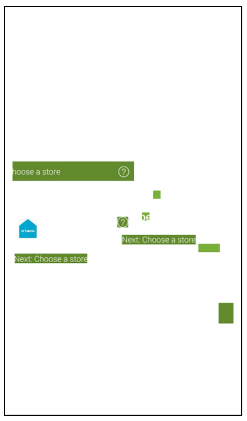
Figure 4.8: CNN Architecture, generated with [3]

4.4.2 Evaluation Setup

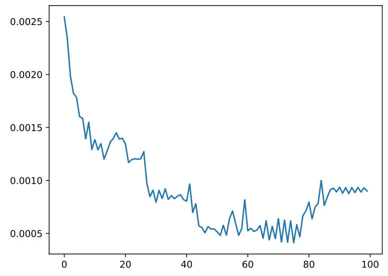
To optimize a given UI, the same setup described in Section 4.2 is used. The only difference to this experiment is the different predictor model. In this case, only the random initial layout is chosen, since the RICO dataset mostly contains examples that already have a



(a) Initial Random Layout



(b) Layout after 100 iterations



(c) Loss progression

Figure 4.9: Optimzation progress

low IoU and a relatively good alignment. To ensure the UI elements are not overlapping in a way that makes them invisible to the classifier, elements are only randomly placed in empty spaces.

4.4.3 Evaluation

Prior to the UI optimization, the CNN is trained over the entire RICO dataset in 10 epochs, with the final MSE being at 0.00063. Similar to the previous experiment, the evaluation is done with one UI as an example. Initially, the layout is random, which can be seen in Figure 4.9a. After that, the optimization cycle is done for 100 steps. The loss progression can be seen in Figure 4.9c. Again, the loss seems to decrease, however, the changes in the resulting image are only minimal, visible in Figure 4.9a. While some UI elements seem to move closer to the middle of the image, the changes are very minimal. The loss decrease also seems to be larger than what would be expected for the amount of changes. If the "real" metric were to be computed, the potential for a lower loss through further changes to the layout would be large. This indicates that the same issue occurs, that was already described in Section 4.2.4, i.e. accidental adversarial attacks. We explored multiple different hyperparameter configurations, however we found no configuration, that improved the results significantly. Other mitigation approaches exist, such as augmenting the input [61] and ensembling models [62], which remain to be explored.

Chapter 5

Final Evaluation & Discussion

In this thesis, we explored different approaches and techniques to approach the topic of metric-guided optimization of existing UIs. We aimed to provide a proof of concept for automatically optimizing existing UIs to provide designers with a tool assisting in the complex task of creating new UIs and better UXs. We wanted to automatically optimize for aesthetics which a model predicts. For this, we explored multiple vastly different approaches. We first used a DM to create and improve UIs on a direct pixel basis. After that, we simplified the latent space and directly optimized the element layout in a latent representation. Additionally, a VAE was evaluated for its ability to restrict the generation. Finally, we used more technical metrics that can directly be calculated and used them to try to prove the validity of the optimization method. During the exploration of these techniques, we faced several challenges mostly related to the automatic optimization of a model-predicted metric. A key problem is the idea of aesthetics and what that encompasses. The goal was to optimize directly for aesthetics since this ensures a good UX without having to define which technical measures, such as alignment, influence UX to what degree. While some research has been done in predicting the perceived aesthetics for UIs [31, 32], using these predictor models as a scoring metric for automatic UI generation is a new concept. Thus, the presented performance metrics in previous work do not necessarily translate well to this task. Specifically, the occurrence of "accidental" adversarial attacks, also called reward gaming, when using these models is a major problem to this research. Pang et al. [63] explored a similar problem in human annotation-based Reinforcement Learning (RL) for conditional text generation. While reward gaming in the context of RL has been discussed [64], this is not the case for the metric-guided generation of UIs. Both Pang et al. [63] and Amodei et al. [64] trace reward gaming back to shortcomings in the metric, either via mislabeling of the data or an unintended bias in the data selected for annotation. While these two issues may also play a role in the aforementioned issue, we suspect a more technical issue with the way the CNNs predictor models work. As previously described, the issue closely resembles an adversarial attack, albeit not an intentional one. Therefore another mitigation strategy would be previously discussed hardening techniques against (intentional) adversarial attacks. Ranjan et al. [65] presented the concept of Compact Convolution which hardens CNNs against adversarial attacks. Utilizing this technique in the aesthetics predictor model would be a topic to explore.

Finally, no complete working proof of concept could be established. However, the space of hyperparameters, different optimization techniques, metrics, and optimizable parameters is larger than what a single thesis could encompass. Thus, it is too soon to say that this approach is flawed by design. We aim to motivate further research into this topic, to clarify whether the showcased approach is viable. This additional research might include exploring different latent spaces, metrics, or even datasets. Modern DM, such as discrete diffusion, remains yet to be used for the described metric-based optimization approach.

The experiments already performed should help others navigate the large space of different parameters previously described. Thus our research should provide a meaningful addition to the resources available in UI generation research.

To motivate further research on this topic, we published all code and related models used in this thesis.[1]

[1]https://github.com/mowoe/bachelorthesis

Bibliography

[1] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” 2021.

[2] J. Zhang, J. Guo, S. Sun, J.-G. Lou, and D. Zhang, “Layoutdiffusion: Improving graphic layout generation by discrete diffusion probabilistic models,” 2023.

[3] H. Iqbal, “Harisiqbal88/plotneuralnet v1.0.0,” 2018. [Online]. Available: https://doi.org/10.5281/zenodo.2526396

[4] IDC. (2024, Apr.) Marktanteil von Android* am absatz von smartphones weltweit vom 1. quartal 2009 bis zum 1. quartal 2024 [graph]. 15. April, 2024.

[5] AppBrain. (2024, Jan.) Number of available applications in the Google Play Store from december 2009 to december 2023 [graph]. Statista.

[6] B. S. Chaparro and C. Phillips, “Visual appeal vs. usability: Which one influences user perceptions of a website more?” 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:5933030

[7] D. Greunen, A. Van, A. Van der Merwe, and P. Kotzé, “Factors influencing bpm tools: The influence on user experience and user interfaces,” *International Journal of Computing and ICT Research*, vol. 4, 11 2010.

[8] C. G. von Wangenheim, J. V. A. Porto, J. C. R. Hauck, and A. F. Borgatto, “Do we agree on user interface aesthetics of android apps?” 2018.

[9] M. Douneva, R. Jaron, and M. T. Thielsch, “Effects of Different Website Designs on First Impressions, Aesthetic Judgements and Memory Performance after Short Presentation,” *Interacting with Computers*, vol. 28, no. 4, pp. 552–567, 06 2016.

[10] D. Council, “The Double Diamond,” https://www.designcouncil.org.uk/our-resources/the-double-diamond/, this work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.

[11] J. Li, J. Yang, A. Hertzmann, J. Zhang, and T. Xu, “Layoutgan: Synthesizing graphic layouts with vector-wireframe adversarial networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 7, pp. 2388–2399, 2021.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[13] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[14] S. Ruder, "An overview of gradient descent optimization algorithms," 2017.

[15] P. Tillet, H. Kung, and D. Cox, "Triton: an intermediate language and compiler for tiled neural network computations," *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019.

[16] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[19] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015. [Online]. Available: https://doi.org/10.1016/j.neunet.2014.09.003

[20] S. Narayan, "The generalized sigmoid activation function: Competitive supervised learning," *Information Sciences*, vol. 99, no. 1, pp. 69–82, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025596002009

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 1–9.

[24] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2256–2265.

[25] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[26] T. Chen, "On the importance of noise scheduling for diffusion models," 2023.

[27] J. Mounayer, S. Rodriguez, C. Ghnatios, C. Farhat, and F. Chinesta, "Rank reduction autoencoders – enhancing interpolation on nonlinear manifolds," 2024.

[28] Q. Xu, Z. Wu, Y. Yang, and L. Zhang, "The difference learning of hidden layer between autoencoder and variational autoencoder," in *2017 29th Chinese Control And Decision Conference (CCDC)*, 2017, pp. 4801–4804.

[29] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[30] L. Weng, "From autoencoder to beta-vae," *lilianweng.github.io*, 2018. [Online]. Available: https://lilianweng.github.io/posts/2018-08-12-vae/

[31] L. A. Leiva, M. Shiripour, and A. Oulasvirta, "Modeling how different user groups perceive webpage aesthetics," *Universal Access in the Information Society*, vol. 22, no. 4, pp. 1417–1424, Nov 2023.

[32] A. L. de Souza Lima, O. P. H. R. Martins, C. G. von Wangenheim, A. von Wangenheim, A. F. Borgatto, and J. C. Hauck, "Automated assessment of visual aesthetics of android user interfaces with deep learning," in *Proceedings of the 21st Brazilian Symposium on Human Factors in Computing Systems*, 2022, pp. 1–11.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[34] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proceedings of the 30th Annual Symposium on User Interface Software and Technology*, ser. UIST '17, 2017.

[35] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[36] K. Reinecke and K. Z. Gajos, "Labinthewild: Conducting large-scale online experiments with uncompensated samples," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1364–1378.

[37] Y. Lu, Y. Yang, Q. Zhao, C. Zhang, and T. J.-J. Li, "Ai assistance for ux: A literature review through human-centered ai," 2024.

[38] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[39] N. Inoue, K. Kikuchi, E. Simo-Serra, M. Otani, and K. Yamaguchi, “LayoutDM: Discrete Diffusion Model for Controllable Layout Generation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 10 167–10 176.

[40] K. Kikuchi, E. Simo-Serra, M. Otani, and K. Yamaguchi, “Constrained graphic layout generation via latent optimization,” in *MM ’21: ACM Multimedia Conference, Virtual Event, China, October 20 - 24, 2021*, H. T. Shen, Y. Zhuang, J. R. Smith, Y. Yang, P. César, F. Metze, and B. Prabhakaran, Eds. ACM, 2021, pp. 88–96. [Online]. Available: https://doi.org/10.1145/3474085.3475497

[41] J. Wu, S. Wang, S. Shen, Y.-H. Peng, J. Nichols, and J. Bigham, “Webui: A dataset for enhancing visual ui understanding with web semantics,” *ACM Conference on Human Factors in Computing Systems (CHI)*, 2023.

[42] S. Feng, S. Ma, H. Wang, D. Kong, and C. Chen, “Mud: Towards a large-scale and noise-filtered ui dataset for modern style ui modeling,” 2024. [Online]. Available: https://arxiv.org/abs/2405.07090

[43] B. Wang, G. Li, X. Zhou, Z. Chen, T. Grossman, and Y. Li, “Screen2words: Automatic mobile ui summarization with multimodal learning,” 2021. [Online]. Available: https://arxiv.org/abs/2108.03353

[44] T. F. Liu, M. Craft, J. Situ, E. Yumer, R. Mech, and R. Kumar, “Learning design semantics for mobile apps,” in *The 31st Annual ACM Symposium on User Interface Software and Technology*, ser. UIST ’18. New York, NY, USA: ACM, 2018, pp. 569–579.

[45] B. Wang, G. Li, and Y. Li, “Enabling conversational interaction with mobile ui using large language models,” in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’23. New York, NY, USA: Association for Computing Machinery, 2023.

[46] W. Feng, W. Zhu, T.-J. Fu, V. Jampani, A. Akula, X. He, S. Basu, X. E. Wang, and W. Y. Wang, “Layoutgpt: Compositional visual planning and generation with large language models,” in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 18 225–18 250.

[47] A. Mathur, G. Acar, M. J. Friedman, E. Lucherini, J. Mayer, M. Chetty, and A. Narayanan, “Dark patterns at scale: Findings from a crawl of 11k shopping websites,” *Proc. ACM Hum.-Comput. Interact.*, vol. 3, no. CSCW, nov 2019.

[48] WebUIProject, “Ui screenshots dataset,” https://universe.roboflow.com/webuiproject/ui-screenshots, sep 2023, visited on 2024-08-09. [Online]. Available: https://universe.roboflow.com/webuiproject/ui-screenshots

[49] N. Deckers, J. Peters, and M. Potthast, “Manipulating embeddings of stable diffusion prompts,” 2023.

[50] G. Ilharco, M. Wortsman, R. Wightman, C. Gordon, N. Carlini, R. Taori, A. Dave, V. Shankar, H. Namkoong, J. Miller, H. Hajishirzi, A. Farhadi, and L. Schmidt, "Openclip," Jul. 2021, if you use this software, please cite it as below.

[51] M. Cherti, R. Beaumont, R. Wightman, M. Wortsman, G. Ilharco, C. Gordon, C. Schuhmann, L. Schmidt, and J. Jitsev, "Reproducible scaling laws for contrastive language-image learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 2818–2829.

[52] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *ICML*, 2021.

[53] N. Ruiz, Y. Li, V. Jampani, Y. Pritch, M. Rubinstein, and K. Aberman, "Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation," 2023.

[54] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman, P. Schramowski, S. Kundurthy, K. Crowson, L. Schmidt, R. Kaczmarczyk, and J. Jitsev, "LAION-5B: an open large-scale dataset for training next generation image-text models," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/a1859debfb3b59d094f3504d5ebb6c25-Abstract-Datasets_and_Benchmarks.html

[55] J. Li, D. Li, C. Xiong, and S. Hoi, "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *ICML*, 2022.

[56] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 6626–6637. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html

[57] J. Skalse, N. Howe, D. Krasheninnikov, and D. Krueger, "Defining and characterizing reward gaming," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 9460–9471. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/3d719fee332caa23d5038b8a90e81796-Paper-Conference.pdf

[58] J. C. Costa, T. Roxo, H. Proença, and P. R. M. Inácio, "How deep learning sees the world: A survey on adversarial attacks amp; defenses," *IEEE Access*, vol. 12, p. 61113–61136, 2024. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2024.3395118

[59] A. Subramanian, "Pytorch-vae," https://github.com/AntixK/PyTorch-VAE, 2020.

[60] A. Asperti and M. Trentin, "Balancing reconstruction error and kullback-leibler divergence in variational autoencoders," *IEEE Access*, vol. 8, pp. 199 440–199 448, 2020.

[61] P. Qiu, Q. Wang, D. Wang, Y. Lyu, Z. Lu, and G. Qu, "Mitigating adversarial attacks for deep neural networks by input deformation and augmentation," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 157–162.

[62] L. Qendro and C. Mascolo, "Towards adversarial robustness with early exit ensembles," in *2022 44th Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC)*, 2022, pp. 313–316.

[63] R. Y. Pang, V. Padmakumar, T. Sellam, A. P. Parikh, and H. He, "Reward gaming in conditional text generation," in *Annual Meeting of the Association for Computational Linguistics*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:253553557

[64] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *ArXiv*, vol. abs/1606.06565, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:10242377

[65] R. Ranjan, S. Sankaranarayanan, C. D. Castillo, and R. Chellappa, "Improving network robustness against adversarial attacks with compact convolution," *CoRR*, vol. abs/1712.00699, 2017. [Online]. Available: http://arxiv.org/abs/1712.00699