Towards End-to-end Semantically aware 2D Plot Tactile Generation

by

Mohammad Heydari

A thesis proposal submitted to Carleton University in partial fulfillment of the requirements for the degree of

Master of Computer Science

in

School of Computer Science

Carleton University

Ottawa, Ontario

Abstract

An estimated 1.5 million Canadians identify as having vision loss, reaching up to 253 million people worldwide. The most effective way of delivering visual information to people with visual disabilities is through tactile graphics. Tactile graphics have textured or raised surface that makes them understandable by touching. To generate tactile graphics, a designer usually uses software such as CorelDraw to draw a grayscale image of the subject which will then be printed by an embosser.

The goal of this thesis is to automate the process of converting 2D plots including Bézier curves, scatter plots, polygons, and bar charts to tactile format. Ultimately, our model can be used as an add-on to speed up the translation process that tactile designers execute. The end user of this model can be tactile designers not people with visual disability themselves, since implementing an end-to-end pipeline for tactile generation without the need to a designer is beyond the scope of this thesis. We defined the problem as an image-to-image translation task where the source domain belongs to 2D plots and the target domain is the tactile equivalent of the input plot. The proposed method is based on the pix2pix architecture which is one of the seminal works on paired image-to-image translation. The proposed models use UNet++ as the generator. We also propose to use gradient penalty and perceptual loss to further enhance the results.

To achieve editable outputs, we propose two approaches. One aims to generate RGB outputs. The other aims to generate multichannel outputs where each channel is associated with a component of the 2D plot. We evaluate the proposed models quantitatively and qualitatively. For RGB outputs we use foreground MSE, background MSE, precision, and recall. On the other hand, we use pixel accuracy, Dice coefficient, and Jaccard index to evaluate our channelwise model.

On the combined category of Bézier curves, scatter plots, and polygons, the proposed channelwise method enhanced the pixel accuracy of the base model from 0.58 to 0.98. It also improved the Dice coefficient from 0.03 to 0.28, and Jaccard index from 0.01 to 0.21. Likewise, for the bar charts, the proposed channelwise method enhanced the pixel accuracy of the base model from 0.82 to 0.98. It enhanced the Dice coefficient from 0.06 to 0.31, and Jaccard index from 0.03 to 0.28.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An estimated 1.5 million Canadians identify as having vision loss, reaching up to 253 million people worldwide. Those affected still need to be able to use products such as documents, forms, books, and messages during their everyday lives. Therefore, banking, healthcare, and all levels of government services should make their products accessible to clients with visual impairment. The demand is more crucial in education, where textbook publishers, students, teachers, and institutions regularly require accessible reading material in a timely manner [1].

Traditionally, the Braille writing system and later on automatic text-to-speech systems have been helpful but unfortunately, they don't work on images and graphics. The most effective way of conveying visual information to those people with visual impairment is through tactile graphics having textured or raised surface. Like braille, tactile graphics can be made with an embosser, and their content can help users understand information by touch. Figure 1.1 shows examples of images of 2D plots and their tactile equivalent.

With advances in AI and Computer Vision in recent years, the goal of this work is

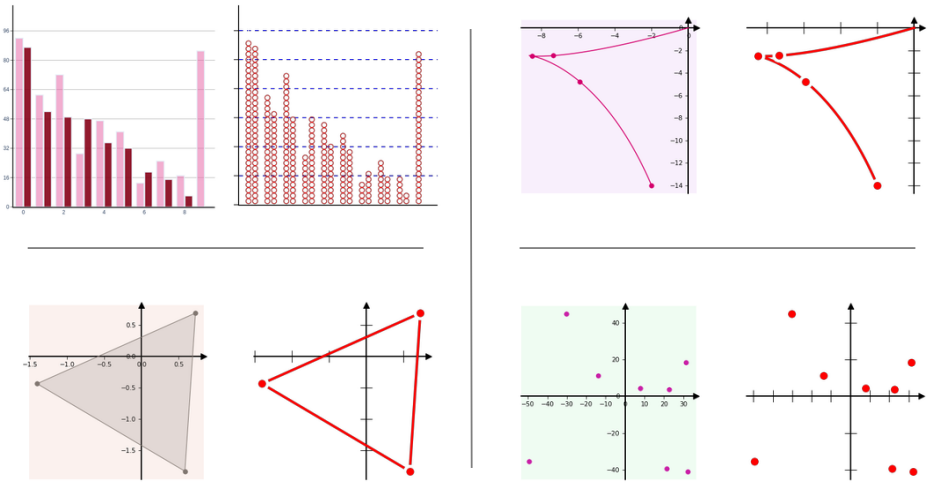

Figure 1.1: An example of every category covered in this thesis.

to design a pipeline using deep learning concepts that empowers designers to translate certain types of images to tactile graphics more quickly and efficiently.

1.1 Problem Statement

The overall goal of this project is to develop a machine learning-based system that automatically converts RGB images to tactile friendly format that needs less modification by the designers before being fed to the embosser. To create the tactile representation of an image, tactile designers use software such as Corel Draw, Adobe Illustrator, and Microsoft Office (Word and PowerPoint) to manually draw a tactile image. This is a time-consuming and expensive task. In many scenarios such as in education, timely access to available formats for exams and assignments is crucial. As a result, it is more efficient to rely on an automated pipeline that performs the

drawing and let the designers customize and post-process the output. Ultimately, the model proposed in this thesis can be used as an AI-based add-on that helps tactile designers to draw tactile images faster.

There are various categories of images the translation of which should be addressed. Among those categories, plots and charts were selected as the starting point of our research. Since plots and charts still include broad subcategories, in this thesis, we will focus only on particular types of them, namely 2D curves, polygons, scatter plots, and bar charts. Examples of such images and their tactile equivalent are shown in Figure 1.1.

For 2D curve images, as shown in Figure 1.2, text areas should be removed. Text parts will be replaced with Braille as a manual post-process step. Converting text to Braille is out of the scope of this thesis. In the tactile image, the background should be removed. Whenever various objects in the plot intersect, the intersection should be marked with a circle, as shown in Figure 1.2. The thickness of the curves and tick lines on the axis should be corrected.

Likewise, for bar charts, text and background should be removed. The axes and tick lines should be corrected. In addition, it is common to represent each bar with a stack of circles as shown in Figure 1.3.

Several attempts have been made to automate the tactile creation process [2, 3, 4]. However, they are all based on programming a hand-crafted procedure that requires human intervention at every intermediate step [3, 5], or the traditional image processing and pattern recognition approaches for detecting lines and circles followed by handcrafted rules to interpret the objects [3]. Such approaches cannot take advantage of all information in a unified way and do not scale to larger datasets. Consequently, without being able to employ those models on an identical dataset and setup, it is

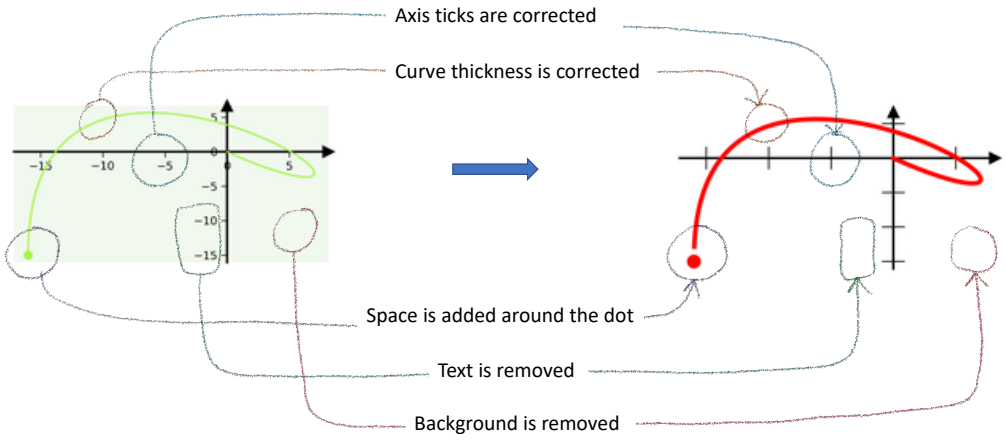

Figure 1.2: A detailed schema of the translation process for Bézier curves.

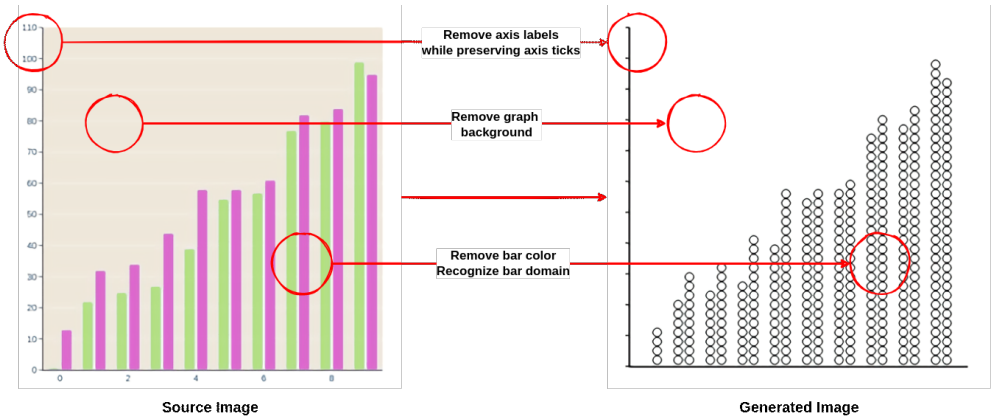

Figure 1.3: A detailed schema of the translation process for bar charts.

hard to compare the performance of those models.

1.2 Contributions

The proposed project will fill the knowledge gap that exists in the current approaches by employing advanced machine learning techniques. We realize the goal of the project by proposing an image-to-tactile translation system in a deep learning framework. The goal is to translate an image from the source domain to a target domain where the source domain includes 2D curve images with a wide range of styles, whereas the target domain has a particular style that follows the guidelines set for designing tactile. See Figure 1.4.

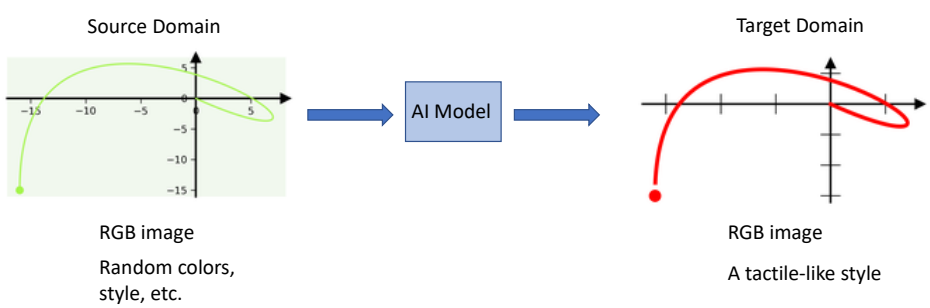

Figure 1.4: A block diagram resembling our desired model.

We also created a dataset of synthesized images and tactile graphics for each of the above categories. The collection includes 10000 image pairs for each type of 2D

plots including 2D Bézier curves, bar charts, polygons, and scatter plots. Each pair consists of an RGB image and its equivalent tactile image. For tactile graphics, we considered two main formats: 1) an RGB image of the tactile. 2) a custom channelwise representation of the tactile, where each channel contains a component of the tactile. For example, channel 1 contains the axes, channel 2 contains the grid lines, and channel 3 contains the curve. The former is more compatible with the existing image-to-image translation methods where input and output are RGB images. The latter is more convenient for manually editing the resulting tactile by a tactile designer because it gives direct access to each component of the tactile.

The proposed approaches for converting 2D curve images to tactile graphics are based on a Generative Adversarial Network [6], which consists of two main blocks: a generator and a discriminator. The generator network consists of an auto-encoder that generates tactile graphics. The discriminator network forces the generator to synthesize output similar to real output.

A tactile image is a grayscale image where black is usually used for lines, white is for background and there could be a few grayscale levels too. The darker the shade of gray, the more the area is raised when tactile is printed. A typical tactile usually has only 8 distinct grayscale levels because finer levels will be too hard to distinguish by touch. To achieve editable outputs, we generate outputs that are more suitable to be used with third-party SVG generators. SVG is an Extensible Markup Language (XML)-based vector image format. SVG files can be edited in most drawing tools such as Corel Draw. There are third-party tools for converging images to SVG. Notably, Potrace is an open-source software for converting images to a vector image format such as SVG. Corel Draw also has a tool that allows converting to SVG. From our experience, such tools are more reliable when applied to colored images. This may

be attributed to the fact that in most cases pixels that should be associated with an object in SVG format have a similar color that is distinct from other objects' colors.

In this thesis, we propose two approaches for converting 2D curve images to tactile. The first approach aims to generate RGB tactile graphics where each component of a 2D curve is shown with a unique color. As discussed above, such an output requires third-party tools to achieve editable output. The second approach aims to create a multi-channel output where each channel contains a component of the 2D curve. Therefore, each component can be accessed individually and modified by a tactile designer.

1.3 Thesis Structure

After an introduction in the first chapter, we will elaborate on related topics to our solution in the second chapter, starting from Artificial Neural Networks concepts such as Perceptron, and Convolutional Neural Networks (CNNs) to Conditional Generative Adversarial Networks (cGANs) mentioning layers and submodules used to build that architecture. Consequently, We provide a few definitions of image-to-image translation tasks. The third chapter is dedicated to the literature review. We will discuss some of the related tasks and existing solutions that have been inspiring or used in our proposed method. We start with general solutions for image-to-image translation tasks, continue with UNet variations [7], which were a game changer in medical semantic segmentation, and end the chapter with two conditional GANs. Pix2Pix [6] uses UNet as the generator, and SemanticGAN [8], which is a more recent model that inspired us to further refine our approach. The model proposed in [8] is one of the first few generative models optimized to generate channelwise outputs to solve semantic

segmentation tasks.

In the fourth chapter, we will introduce 2DPlotsTactile-GAN, our proposed method to address channelwise tactile generation for 2D plots. We dedicate the fifth chapter to the exploration of our dataset in more detail and demonstrating the results generated by the proposed models. In Chapter 6, conclusions, achievements, and limitations of the proposed models are discussed. We will end the thesis by leaving some ideas for future work.

Chapter 2

Background

In this chapter, we plan to delve into the key concepts that are essential to know before we develop a solution to the problem of channelwise tactile generation. We start with CNNs that raised hopes about deep learning through the improvement that they brought about in Image Classification accuracy. We put more emphasis on the blocks that are commonly used in a CNN rather than introducing various CNN architectures and their advantages since the blocks are vastly used in the submodules of our proposed model.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are computing systems naturally inspired by biological neural networks that are the constructive unit of animals' brains. Their forms and complexity have evolved on the basis of brain plausibility or simply through direct experiments and what tends to work out.

2.1.1 Perceptron

A perceptron is the smallest computation unit in an ANN, designed to mathematically model a neuron which is the fundamental unit of the brain and nervous system. The neuron only transmits nervous signals passing a certain threshold to the next cells, motivating scientists to perform binary classification using a model following the same behavior. More precisely, a perceptron computes a weighted sum of its inputs, passing the result to a non-linear activation function that acts like a trigger and using the result as the output or $o = f\left(\sum_i w_i x_i\right)$. Figure 2.1 shows how the model of a perceptron is inspired by natural neurons. Perceptron was able to mimic simple logical functions and classifiers.

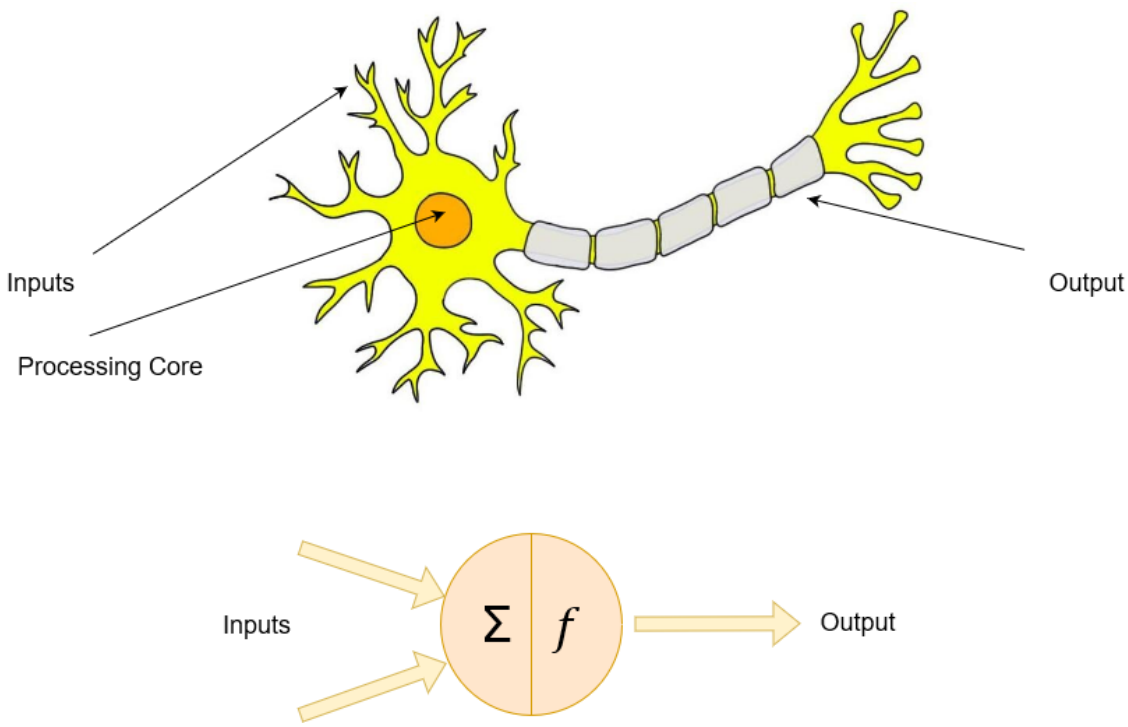

Figure 2.1: The similarity of a neuron to a perceptron.

2.1.2 Activation Functions

Activation functions are the components that add nonlinearity to the CNN architecture. ReLU variations are widely used in CNNs, allowing us to stack more layers and design deeper architectures that can solve more complex optimization problems. Using ReLU needs less computation comparably and prevents gradient vanishing since its derivative does not tend to zero as Figure 2.2 suggests. The derivative of an activation function is very important because of its role in backpropagation. We can realize the significance of ReLU by comparing the behavior of its derivative to the other well-known activation functions like Sigmoid and Tanh, where the derivative tends to zero when we are located on the extremums of each function, affecting the learning capability of the submodule.

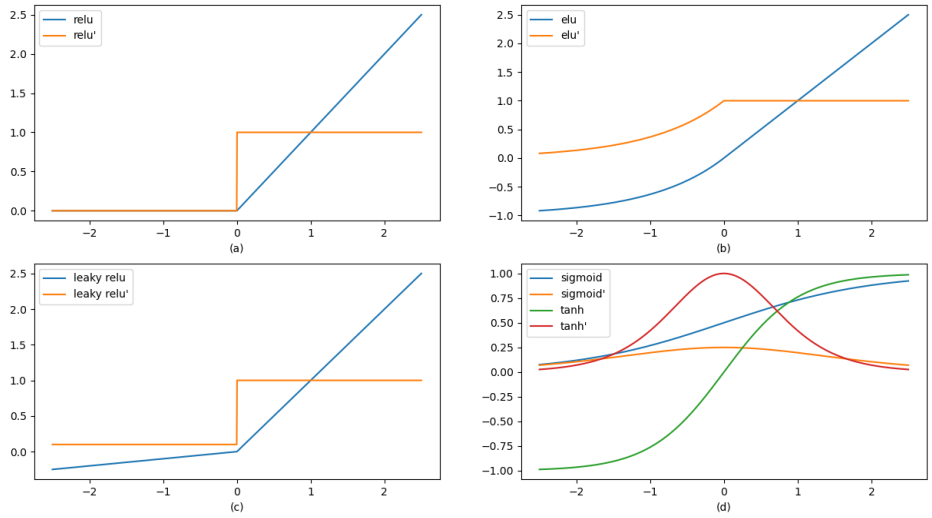

Figure 2.2: ReLU and its derivative have been plotted on (a). To add more flexibility there are other variations of ReLU such as Elu and LeakyReLU plotted on (b) and (c) respectively. They try to address the problem of dying ReLU in its negative region by replacing the term corresponding to that part. However, ReLU is still widely used due to its simplicity.

2.1.3 Multi-layer Perceptron

With advances in computation power and the emergence of more activation functions, multi-layer perceptron (MLP) was introduced as a structured block of perceptrons. The architectures which have hidden layers, i.e. any number of intermediate layers between the input and the output layers, brought about the notion of deep learning being widely used. These models were capable of performing feature extraction and classification at the same time. This aspect makes it easier to apply deep learning models to new problems; however, a larger black box can raise concerns about the interpretability of the model. The training of MLPs follows the backpropagation algorithm, which updates weights proportional to the derivative of the error with respect to each weight calculated using the chain rule.

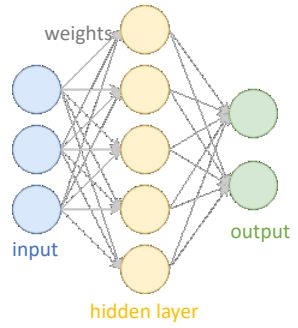

Figure 2.3: The schema of a simple MLP with one hidden layer.

2.1.4 Convolutional Neural Networks

This category of artificial neural networks (ANN) was established after showing promising classification performance on the ImageNet [9] dataset. Since AlexNet [10] which was the architecture that outperformed non-Deep Learning methods for the first time in 2012, CNNs have evolved every year. However, the components of each architecture have been almost the same and, we want to introduce those that were the most important to us.

2.1.4.1 Convolution Layers

MLPs are not capable of tracking spatial dependencies efficiently. On the other hand, convolution layers have an intuitive way of connecting learnable parameters together, called parameter sharing, which makes them more capable of learning local correlations compared to fully connected layers. The weights of these layers are grouped as multi-dimensional kernels (filters) that move across the input computing a weighted sum of what they "see." These kernels can represent image processing crafted filters such as horizontal or vertical edge detection, more flexibly. In Figure 2.4, you can find how the output of a convolution kernel should be calculated.

There are several hyperparameters in a convolution layer that we should fix in the design process including:

- The shape of the kernel that we show by k.
- The number of pixels that we use for padding is shown by p. This parameter helps us avoid the outputs with too small shapes due to multiple convolutions.
- The stride of the convolution is shown by s, controlling the step size the kernel takes during the convolution.

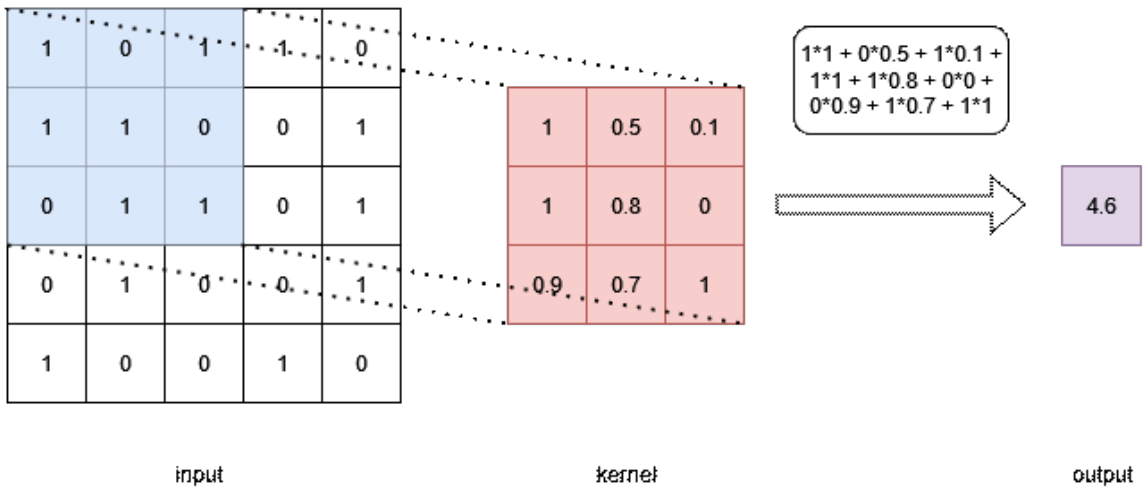

Figure 2.4: The process of calculating a convolution.

- The number of kernels shown by d determines the depth (number of channels) of the output.

Consequently, the shape of the output shown by o follows the equation below for the input with the shape of i.

$$o = \frac{i - k + 2p}{s} + 1 \tag{2.1}$$

2.1.4.2 Fully-connected Layers

By aligning multiple perceptrons as layers and intuitively connecting every perceptron in consecutive layers, we will form fully-connected layers. Fully-connected layers showed great capability in feature extraction and supervised learning of categorical data. However, they are not sensitive enough to spatial features nor robust enough against geometrical transformations such as rotation, scaling, and shifting. As a result, they are widely used (if ever) in the last layers of CNN architectures, where we deal with the classification task rather than feature extraction.

2.1.4.3 Transposed Convolution Layers

In a group of CNN architectures, such as encoder-decoders, there is a contraction part that tries to represent the input in a new vector space and a dilation part that tries to build up the output from that vector space. That's where the need for upsampling arises. As the name may suggest, transposed convolution layer is a module that performs the inverse function of a convolution layer. By setting the hyperparameters, we can achieve upsampling by a transposed convolution layer. Figure 2.5

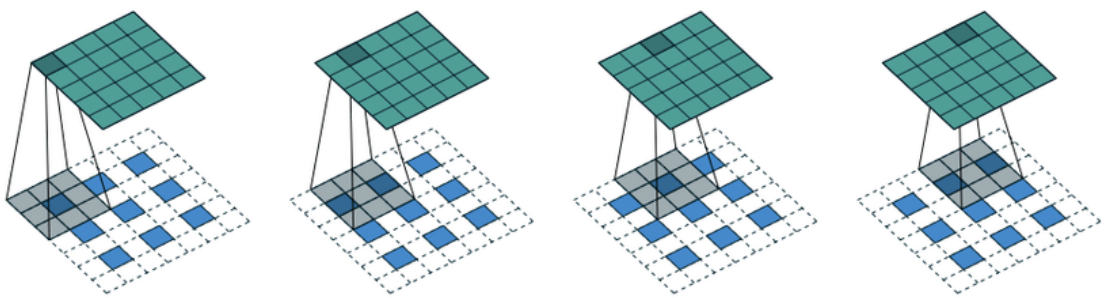

Figure 2.5: The transpose of convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5, k = 3, s = 2$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 3×3 input (with 1 zero inserted between inputs) padded with a 1×1 border of zeros using unit steps ($i' = 3, \bar{i}' = 5, k' = k, s' = 1$ and $p' = 1$). [11]

2.1.4.4 Normalization Approaches

One of the operations that we may need in our deep learning architecture is normalization across different pivots. Regardless of the kind of normalization, it is performed by subtracting the mean from the input and then dividing it by the standard deviation denoted as $\hat{x}_i = \frac{1}{\sigma_i}(x_i - \mu_i)$.

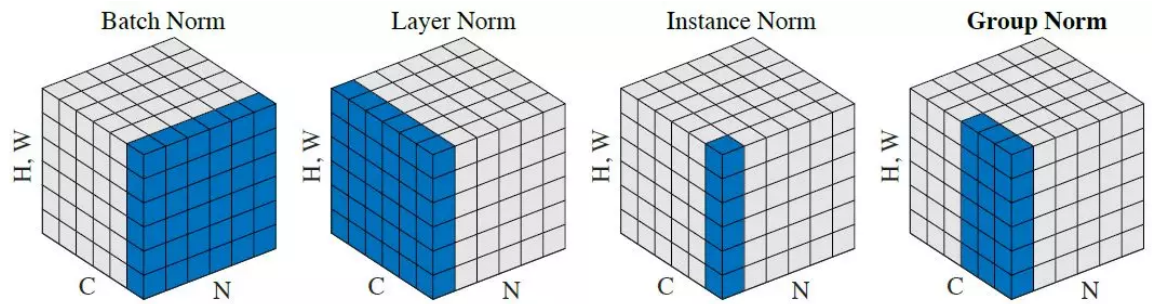

Figure 2.6: Normalization methods. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels. [12]

2.1.4.5 Aggregation Layers

Aggregation is an inevitable part of many CNN architectures. Intuitively, convolution layers might slightly reduce the input dimensions by width and height, but largely increase their depth. As a result, we need complementary mechanisms to control the number of trainable parameters, that indicates how huge and resource intensive our model will be. There are non-trainable layers to reduce the height and width of the input named after the operation that they perform to aggregate values like Max Pooling or Average Pooling which are drawn in Figure 2.7.

2.1.5 Recurrent Neural Networks

Recurrence adds memory to the ANN. Recurrent Neural Networks (RNNs) were introduced to learn temporal dependencies and process sequential data more efficiently. For instance, CNNs consider each sample independent from the others, but such an assumption will not be valid for movie frames, rain forecast, or stock market analysis. The most trivial group of RNNs is formed by adding a link from the output of a node to its input and making the next output of the model dependent on every input fed

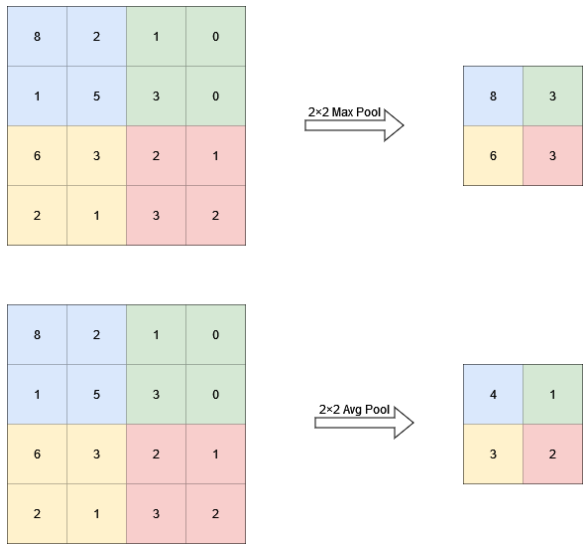

Figure 2.7: The difference between Average Pooling and Max pooling.

to it according to the equation 2.2.

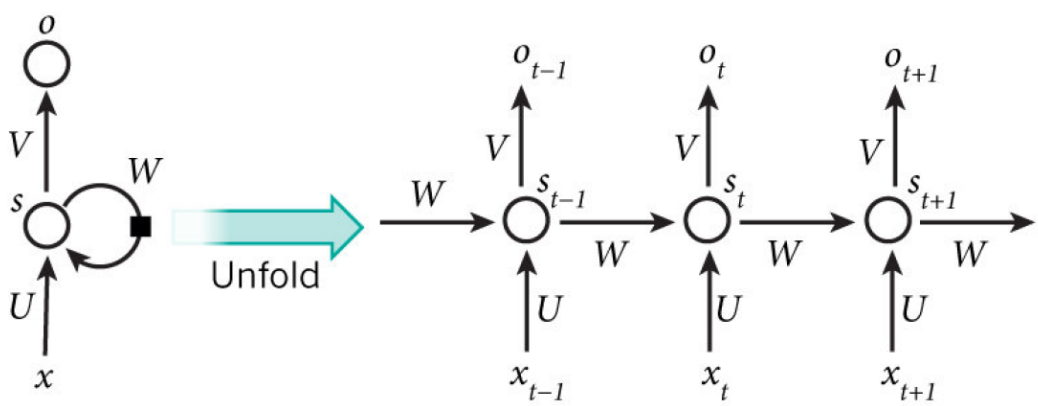

Figure 2.8: Unfolding an RNN node explains how it codes temporal dependencies.

$$s_t = \sigma\left(Ux_t + Ws_{t-1}\right) \tag{2.2a}$$

$$h_t = softmax\left(Vs_t\right) \tag{2.2b}$$

By training RNNs using the backpropagation algorithm, we would find an exponential dependency on the derivative of the activation function, since we have to multiply it for each time point in the expanded model. As a result, the training would even take so long by gradient vanishing or, divergence due to gradient explosion. As a remedy, one can clip the gradient to keep it in the desired range or use gated RNNs that have exclusive gradient paths to secure training stability and learn long-term temporal dependencies such as LSTM and GRU.

2.1.5.1 LSTM

Long short-term memory or (LSTM) [13] is a more advanced form of RNNs that is able to learn arbitrary long-term dependencies by utilizing a constant error carousel (CEC) that is a feedback loop enabling the LSTM to learn long-term relationships while reducing the risks of gradient vanishing. LSTM architecture was developed to have a forget gate that allows for continual prediction. Figure 2.9 shows the process taking place inside an LSTM cell and equations 2.3 describe it mathematically where $\odot$ represents element-wise matrix multiplication.

$$z_t \;=\; tanh\left(W_z\left[h_{t-1},\; x_t\right]\right) \tag{2.3a}$$

$$i_t,\; f_t,\; o_t \;=\; \sigma\left(W_{i,f,o}\left[h_{t-1},\; x_t\right]\right) \tag{2.3b}$$

$$s_t \;=\; z_t \odot i_t \;+\; s_{t-1} \odot f_t \tag{2.3c}$$

$$h_t \;=\; tanh\left(s_t\right) \odot o_t \tag{2.3d}$$

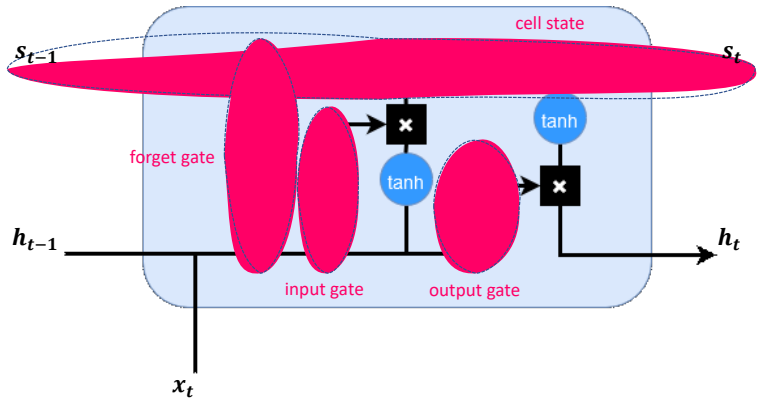

Figure 2.9: The position of input gate, output gate, forget gate, cell state, and the way that they are connected in an LSTM cell.

2.1.5.2 ConvLSTM

To mix the learning capabilities of RNNs and CNNs, we are provided with ConvLSTM which codes spatio-temporal features available in videos or other dependent series of images. Processing such data using LSTM units has the same limitation of feeding images to a fully-connected network. The redundancy in the way that the nodes are connected will prevent the model to keep track of spatial dependencies. Therefore, we have ConvLSTM [14] layers simply by replacing dense trainable weight $W_{i,f,o,z}$ with convolutional kernels. It's worthy to note that a simplified version of the cell proposed in [14] is widely used and mathematically formulated as equation 2.4 which $*$ indicates convolution operation.

$$z_t \;=\; tanh\,(W_z * [h_{t-1},\; x_t]) \tag{2.4a}$$

$$i_t,\; f_t,\; o_t \;=\; \sigma\,(W_{i,f,o} * [h_{t-1},\; x_t]) \tag{2.4b}$$

$$s_t \;=\; z_t \odot i_t \;+\; s_{t-1} \odot f_t \tag{2.4c}$$

$$h_t \;=\; tanh\,(s_t) \odot o_t \tag{2.4d}$$

2.1.6 Autoencoder Architecture

To investigate related works to this research and elaborate on the proposed method, it is crucial to study autoencoder (AE) [15] architecture. (AE) is applicable to image denoising, anomaly detection, and super-resolution. This architecture technically consists of a contracting CNN, termed the encoder, and an expanding CNN, called the decoder. They are connected through a common vector (code), which is the output of the encoder and the input of the decoder at the same time. The encoder tries to learn a vector space where the samples are distinguishable based on a loss function, while the decoder is learning how to reconstruct the initial input of the whole model having the code.

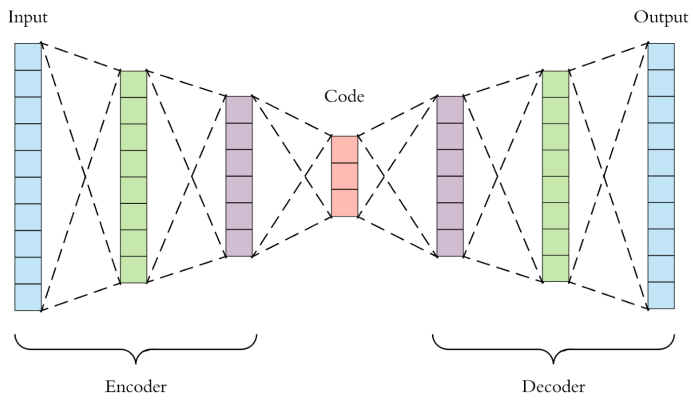

Figure 2.10: The relation of the encoder, decoder, and the code in an autoencoder.

2.1.7 Generative Adversarial Networks

A generative adversarial network (GAN) [16] is an innovative framework for training ANNs that defines its loss function based on competition with another ANN. Figure 2.11 shows a basic schema of how the main modules of a GAN, named generator and discriminator, interact.

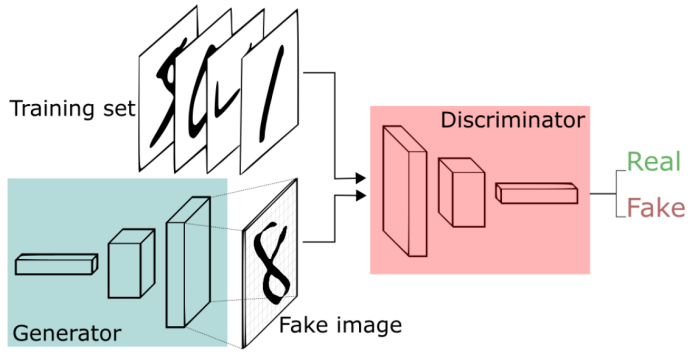

Figure 2.11: The relation of the generator and discriminator in a GAN.

As you may notice, in the original idea, the generator does not need any input, producing fake samples from random noise. As the training goes by, the generator learns to produce more realistic outputs or estimate the probabilistic distribution of the dataset while the discriminator learns to tell apart fake inputs from real ones.

Using GANs, complicated domain-specific data augmentation is feasible, as well as solving problems like image-to-image translation that demand a generative approach.

2.1.8 Conditional GAN

Conditional GANs theoretically emerged as a solution to mode collapse happening frequently in GANs. Mode collapse stands for the situation when the generator tries to trick the discriminator by only generating limited types of samples in the real dataset. For instance, the generator may find it efficient to generate only eights in the scenario of learning handwritten digits, since we do not have any direct control over its input. Alternatively, conditional GANs enable us to specify the class or type of generated output by feeding data to the generator instead of random noise. One way of achieving conditional GANs is by combining the notion of AE with GANs, and hinting to the generator about the type of images it should cover by feeding images to it.

2.2 Image-to-Image Translation Tasks

Image-to-image translation is a computer vision task in which the input and output of the model are images, unlike other tasks such as image classification. The goal of image-to-image translation is to maintain the actual components of meaning from the source image while taking the style features from another domain. For instance, converting all the horses in a landscape to zebras, night scenes to morning, or rainy weather to sunny, could be a perfect example of image-to-image translation.

Semantic Segmentation

Semantic segmentation is a computer vision task of labeling an image based on the object for which each pixel is responsible. Figure 2.12 shows an exaggerated example of how we expect a semantic segmentation model to generate labels for different objects in an image. Each square in the output is an enlarged version of a pixel. Having the expected format of the input and output, we can naively stack convolution layers to match the dimensions, but more intuitively, we can leverage the dimensionality reduction ability of autoencoders. One of the key differences between a semantic segmentation scenario and many other image-to-image translation tasks is in the methods we use to evaluate the model. In the semantic segmentation problem, we care about the pixel classification performed by the model. As a result, common evaluation metrics are formed based on the pixel values of the generated output. On the other hand, we have metrics that assess the overall quality of generated outputs, such as the Fréchet inception distance (FID) or inception score (IS), that compare the distributions of generated output and the ground truth, in general.

Image Inpainting

Image inpainting is a task where image-to-image translation meets image reconstruction. It is an important computer vision task that can extend to object removal, image restoration, and manipulation. Autoencoders and GANs are two well-known tools to address image inpainting. Figure 2.13 shows two examples of image inpainting.

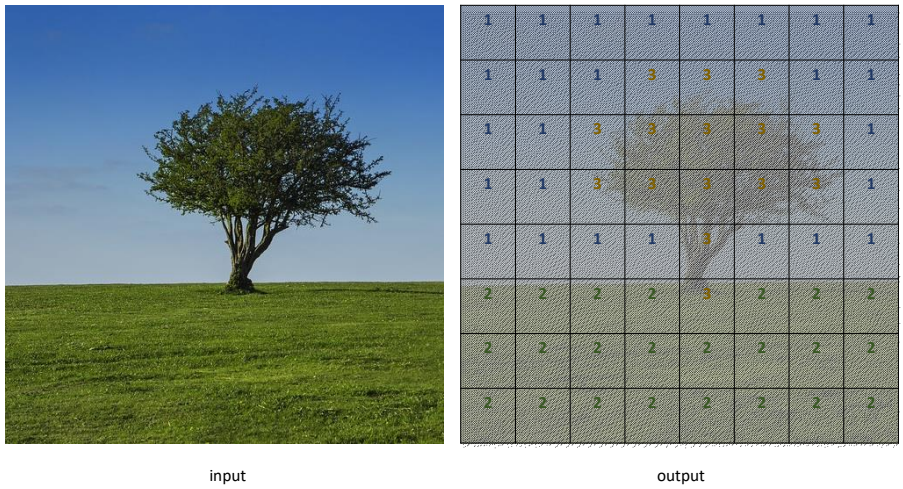

Figure 2.12: An example of semantic segmentation showing sky with label 1, grass with label 2, and tree with label 3.

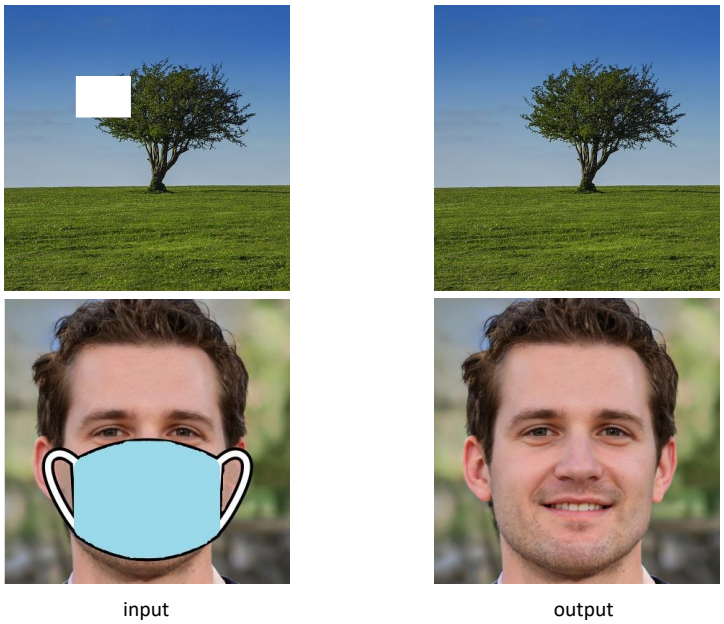

Figure 2.13: Examples of image inpainting (a) inpainting patches and (b) inpainting objects.

Style Transfer

Style transfer is a specific task of alternating the theme of an image based on the style of another image. For instance, we might like to know how a certain painter would

draw a scene. The important job for the model in this scenario is to tell content and style apart, then adapt the style while the content is ideally unchanged. Research on style transfer has produced many generative models that optimize multistyle transfer [17], model attention, and per-object style transfer.

Sketch Generation

Sketch generation is a subcategory of style transfer where the style comes from the pencil sketches of a painter. The difference between sketch generation and style transfer is the fact that sketches are usually simpler than the actual subject. By drawing a sketch, we aim to summarize the content and provide a minimal representation, ignoring inconsiderable details.

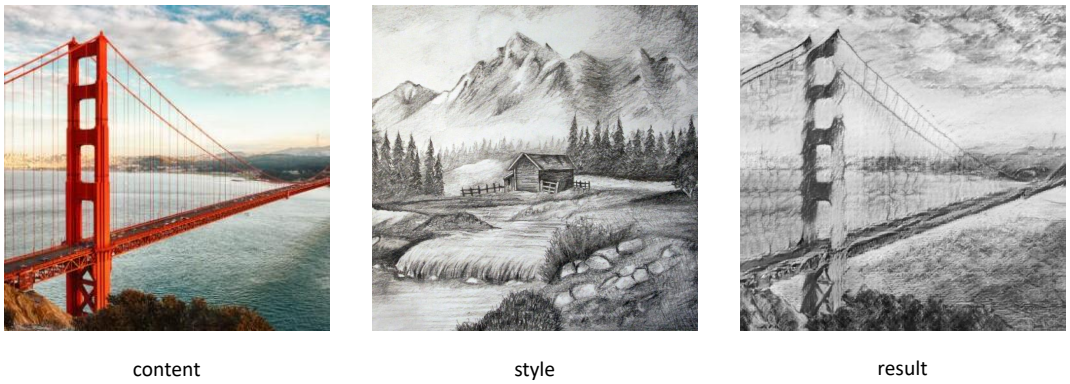

Figure 2.14: The sketch of the Golden Gate bridge drawn by DeepAI fast style transfer model.

Chapter 3

Literature Review

In this chapter, we will discuss papers and models relevant to our problem. Non-machine learning approaches have not yet performed better than manually drawing tactile graphics. As we mentioned in the first chapter, right now, the designers use CorelDraw and other drawing tools to create tactile images for people with visual disabilities. All of the solutions to problems similar to ours, fall under the umbrella of image-to-image translation. Image-to-image translation approaches can be categorized as binary domain and multi-domain. Further, similar to many other machine learning tasks we can take supervised, semi-supervised, or unsupervised approaches to address these categories. There is rich literature on image-to-image translation due to its significance in computer vision, and we aim to find our way through to build the ground for image to tactile translation which is, to our knowledge, rarely explored. Pix2Pix [6] among supervised approaches, and CycleGAN [18] among unsupervised approaches have notably set the groundwork for other GAN-based image-to-image translation solutions. More specifically, image-to-image translation has been applied in semantic image synthesis [19], [20], [21], [22], [23], image segmentation [24], [25],

[26], style transfer [18], [27], [28], [29], [30], image inpainting [31], [32], [33], [34], [35], 3D pose estimation [36], , [37], image/video colorization [38], [39], [40], [41], [42], [43], image super-resolution [44], [45], domain adaptation [46], [47], [48], cartoon generation [49], [50], [51], [52], [53], [54] and image registration [55]. As we move forward in this chapter, we discuss more relevant models in more depth.

One of the unique applications of image-to-image translation that facilitated the development of our model was the conversion of the rasterized floorplan to vector graphics addressed by Raster-to-vector [56]. Typically, existing approaches before [56] highly relied on performing preprocessing on floorplan's components, and made the translation possible by adopting low-level image processing such as edge detection. On the other hand, the authors in [56] tried to detect key components of the floorplan and reconstructed them with a new style into a vector representation. Approximately, we tried to map this point of view to image to tactile translation, and define the objective around detecting key objects in a 2D plot and then, translating them to tactile domain.

Image to sketch translation is also relevant to our problem. Both sketches and tactile graphics reflect human comprehension of an image. While drawing a portrait, an artist decides how to maintain simplicity without losing the identity of the subject. Similarly, a tactile designer aims to only transfer the most informative parts of the image to tactile. The GAN model proposed in [57] addressed one to many sketch generation. To ensure identity preservation and sketch quality, the GAN proposed in [58] uses one generator and two discriminators. [59] uses Cycle-GAN [18] to address bidirectional image to sketch conversion and a feature autoencoder to refine the synthesized results. [60] also addressed bidirectional conversion of images and sketches. Composition-Aided Generative Adversarial Network (CA-GAN) was

proposed in [61]. In that model, paired inputs consisting of a face photo/sketch and the corresponding pixel-wise face labeling mask are utilized to generate the portrait. Moreover, [62] proposed an Identity-Aware Cycle Generative Adversarial Network (IACycleGAN) that concentrated on both face photo-sketch synthesis and recognition. The DeepFaceDrawing model [63], can convert hand-drawn face drawings to actual pictures. In that model, key facial features were learned to be embedded in features, and a network was used to map the embedded features to the real photo. A face image synthesis system called DeepFacePencil [64] is based on hand-drawn sketches. This model makes use of the spatial attention pooling (SAP) module and the dual generator training technique.

In the following, we will discuss the some of most relevant prior work in more detail.

3.1 UNet

Medical image segmentation, which is a specific application of semantic segmentation, led to the introduction of a model called UNet. Due to the lack of numerous annotated samples in medical image datasets, a model was required to make better use of labeled data. In the paper proposing UNet [7], the authors have come up with a model that uses a contracting path to capture context and an expanding path to rebuild precise localization similar to autoencoders with the addition of a new path called skip connections that hints the model with piecewise reconstruction of the output by directly concatenation of corresponding feature maps of the contracting path to those of expanding path. Usually, medical image datasets have a few samples with quite high resolutions. The authors in [7] used this phenomenon as a data augmentation

tool by cropping each image into several training samples. Moreover, they partially perform segmentation on test samples and fix the final results by concatenating related outputs in the same order. Originally, the architecture of UNet does not guarantee that the output will have the same size as the input. As a result, they had to crop the feature maps from the contracting path before performing concatenation.

3.1.1 UNet++

An ablation analysis on UNet showed that the depth of the model can act as a hyperparameter that varies the performance of the model based on the dataset and the whole scenario. To alleviate confusion, the authors of UNet++ [65] proposed a model that was an ensemble of UNets with four different depths values and shared their encoder as far as the depth allowed. They denoted the new model as $UNet^e$ and argued why this model needed deeply supervised training in order to converge. On the other hand, they tested another model with the same layout and set the connection among adjacent nodes instead to form UNet+. This model did not depend on deeply supervised training. Finally, they combined both connection sets in UNet+ and $UNet^e$ to form UNet++. The later architecture addressed the limitations of UNet by leveraging an encoder with variant depth and a more flexible feature fusion scheme by adding more skip connections between intermediate nodes with different depths instead of forcing the decoder to use only the feature maps of the encoder with the same depth. UNet++ tends to generate more precise segmentation maps without being dependent on deeply supervised training. They also replaced transpose convolutions with simple interpolations on the expanding path to restrain the number of learning weights, prevent overfitting, and speed up convergence.

3.1.2 BCDUNet

A more recent variation of UNets called BCDUNet [66] set a new state-of-the-art for medical semantic segmentation on multiple datasets. The improvement was made by adding memory to skip connections using ConvLSTM layers. This approach can help the model with the data augmentation scheme used in the original paper and enhance the learning capacity of the model by adding nonlinearity to the skip connections. UNet does not track the dependencies of the input patches that belong to the same sample, so BCDUNet might take advantage of those patches more intelligently. Furthermore, BCDUNet uses dense convolution [67] in the layer connecting the expanding path to the contracting path. Dense convolution intuitively reuses feature maps from preceding layers in a feed-forward manner leading to more parameter sharing and better gradient flow.

3.1.3 Discussion

UNet can be considered the first significant deep learning model to address problems where the output is not necessarily a label. Unlike previous methods, UNet breaks the trade-off between localization accuracy and the use of context. All of the variations show great learning potential. However, the instinct of an optimization problem depends on the loss function as well. Despite UNet variations can address semantic segmentation tasks efficiently, there is a need for a more complex loss function that can point out the desired and irrelevant output types more differently. The CNN model will basically provide blurry results if we only force it to optimize a loss function based on the Euclidean distance between predicted and ground truth pixels which is highly biased to the average of all possible outputs.

3.2 Perceptual Loss

Per-pixel loss terms are unable to generate sharp images or enable the model to learn deep contextual features. It is observed by visualizing the feature maps of fully trained CNNs that a hierarchy exists between the level of abstraction in the features and the depth of the feature map. In other words, shallow layers tend to be triggered by simple features such as edges and colors while the deeper layers react to wheels, ears, hair, etc. based on the categories that exist in the dataset. Based on the same statement, Johnson et al. proposed a more efficient loss term called perceptual loss [68]. The main idea is that a trained classifier with rich feature maps should have similar perceptions from similar images. For instance, the feature maps will react more similarly to two images of dogs than to an image of a dog and another image of a cat. Therefore, we can penalize our model to be trained based on a loss function that compares the distance between the feature maps of the ground truth and those of the generated output. In their paper, they used the feature maps of a VGG16 pretrained on ImageNet [9] as the loss network to evaluate the quality of the output of the generative model. They argued that their approach can be useful in both style transfer and super-resolution by the fact that the loss term can successfully separate style and content objectives. The authors have considered the Euclidean distance between feature maps to form feature reconstruction loss, and a complementary style reconstruction term should take care of colors, textures, and common patterns. This approach was followed by adding more shallow feature maps to style reconstruction loss than feature reconstruction loss. For style reconstruction loss the authors suggested the Euclidean distance of Gram matrix formed by the feature maps instead of directly using them. Gram matrix G_j^{ϕ} is a square matrix with the size of $C_j \times C_j$ where C_j is the number of channels in the jth feature map and the matrix itself is proportional

to the uncentered covariance of the feature maps $G_j^\phi \propto \Phi\Phi^T$. The main reason to do so is that the style reconstruction loss would be well-defined even when $\hat{y}$ and y had different sizes since their Gram matrices would both have the same shape.

3.2.1 Adversarial Perceptual Loss

Adversarial perceptual loss has been coined in perceptual adversarial networks (PAN) [69]. The model has shown promising results on image-to-image translation tasks such as image deraining, image inpainting, object edges to photos, semantic labels to scenes, etc. The authors have intuitively replaced the loss network with the discriminator of a GAN and used its feature maps alongside the training procedure. As a result, the generator would try to trick the discriminator perceptually, having access to its different feature maps instead of a prediction label or map. The model leverages two loss terms including adversarial loss and perceptual loss. The perceptual loss term was calculated using the same approach as feature reconstruction error, directly by calculating the Euclidean distance. For the sake of context, we only mentioned the loss function in this part and left more details about the architecture to another section particularly related to Pix2Pix.

3.2.2 Discussion

The original perceptual loss could be used in any architecture as a full loss function having feature reconstruction loss and style reconstruction loss. They tend to show more flexibility and stability in training due to the use of an independent network, which is fully trained to give feedback to the actual model. However, PAN uses perceptual loss as a partial loss term from a network that is actively being trained.

3.3 Pix2Pix

Pix2Pix [6] is a general-purpose architecture to address image-to-image problems using conditional GANs. The authors argue that GANs learn a loss that adapts to the data instead of requiring a problem-specific loss function. The model consists of a UNet [7] generator and a PatchGAN as the discriminator. PatchGAN generates a label matrix the elements of which are responsible for how realistically each patch of the generated outputs looks from the receptive field of the discriminator. The authors argued that using a combination of the L1 norm between the ground truth and generated output along with the adversarial loss term could balance the attention between fine and coarse details, respectively. Although the model was originally tested on a "label maps to images task" specifically using labels to facade dataset, it was not initially used as an image to segmentation map solution.

3.4 SemanticGAN

A solution to semantic segmentation using generative models was developed SemanticGAN [8]. The model is a modified version of a general-purpose GAN called StyleGAN2 [70]. Initially, StyleGAN2 aims to separate high-level attributes, such as identity and pose in human faces, from low-level aspects such as hair, eye color, and so on with unsupervised learning. The reason why the authors used this model as their base model was that they wanted to address scenarios with limited labeled data. They aimed to reduce human annotation efforts for different datasets by using a model that can achieve strong generalization ability. The SemanticGAN generator produces a pair of RGB image and segmentation map through different channels showing how different channels of the output can be optimized using separate loss terms. There are

two discriminators in the model denoted as D_m and D_r. D_m takes a concatenation of images and label maps, giving feedback on how correctly the segmentation map is generated. This part of the network needs annotated data. However, D_r only takes images and determines whether they are real or fake. Consequently, the whole model can make use of unlabeled data too and perform semi-supervised training.

Chapter 4

Proposed Method

In this chapter, we will describe the proposed models and the different stages we went through to evolve the final configuration. We propose two models. The first model is designed to generate RGB tactile images where each component of the input 2D plot is shown by a unique color. We will refer to this model as 2DPlotsTactile-GAN. The model should detect and remove text, and background, recognize different components of the input plot, correct the thickness of the lines and axis ticks and show different components of the 2D plot in different colors. The motivation for developing this model was to provide the capability to edit the output by converting it into the SVG format using available third-party tools such as Potrace and CorelDraw. Although tactile is a grayscale image, generating a colored tactile helps with a more reliable RGB to SVG conversion where the resulting SVG file can be edited by tactile designers using software such as CorelDraw.

The second model that we developed aims to generate multi-channel output images where each component of the input 2D plot is encoded in a separate channel. For example, channel 1 contains the axes, channel 2 contains the grid lines, and channel 3

contains the curve. We will refer to this model as 2DPlotsTactile-GAN-Channelwise. The advantage of the 2DPlotsTactile-GAN-Channelwise model is that it provides direct access to the individual components of the generated tactile which in turn facilitates post-hoc modifications by a tactile designer without any additional third-party tools.

4.1 2DPlotsTactile-GAN

We build our model based on Pix2Pix [6] which set the ground for many other supervised GAN-based image-to-image translation models. To begin with, we formulate our problem as a typical image-to-image translation task similar to other applications that Pix2Pix had proven to be successful [6]. The input of the model is an RGB image and likewise, the output is another RGB image, expected to have a specific style based on tactile standards. Through extensive experiments, we took their PatchGAN discriminator, coupled it with various generators, and examined additional loss terms to achieve the desired model behavior. In the following, we will describe different components of the proposed 2DPlotsTactile-GAN model.

4.1.1 Generator

UNet variations are mostly used for semantic segmentation tasks where any of the input or output images are grayscale. We reimplemented BCDUNet [66] from scratch to fit it to the rest of our model and also changed the UNet++ [65] input and output shapes to generate RGB images. In our implementation, we have the freedom to set the number of input channels and output channels to arbitrary numbers. Figure 4.1 shows the schema of the modified UNet and table 4.1 shows the output shape of each stage. We replaced max pooling layers and covered its functionality with convolutional

layers. In our generator, down-conv is a convolutional layer with a kernel size of 4 and step size of 2, and another convolutional layer with a kernel size of 3 and step size of 1. Unlike the original model, the output is the same as the input. Out-conv, however, is only responsible for changing the number of channels at the output. The output layer of the UNet++ and BCDUNet models also were modified to match the desired output size.

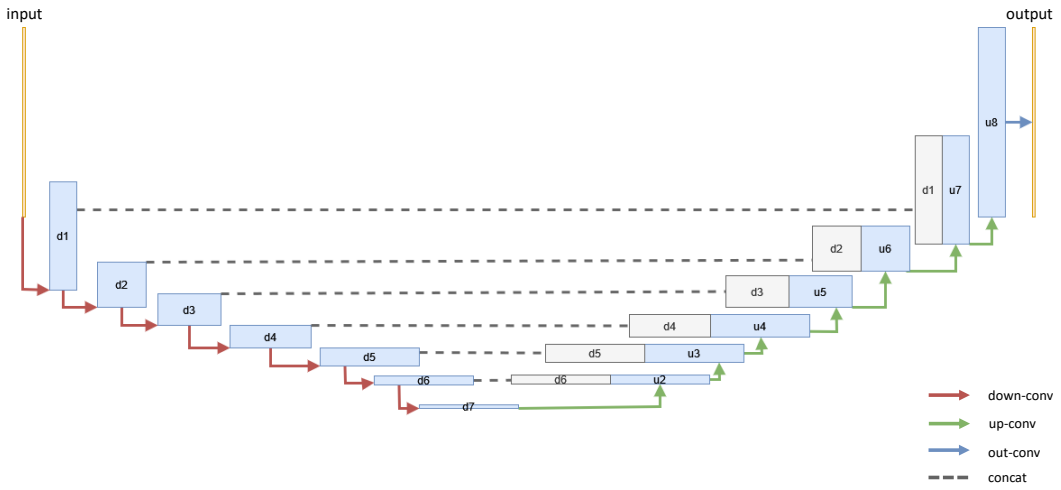

Figure 4.1: The schema of the modified UNet such that it supports RGB inputs and outputs.

4.1.2 Loss Function

We improved the original Pix2Pix loss function that only includes a distance-base loss term (L1 loss) and the GAN loss by adding a perceptual loss term and gradient penalty. Consequently, the loss function of the proposed model has four terms which will be described in the following.

Down-conv	Output shape	Up-conv	Output shape
d1	$128 \times 128 \times 64$	u2	$4 \times 4 \times 512$
d2	$64 \times 64 \times 128$	u3	$8 \times 8 \times 512$
d3	$32 \times 32 \times 256$	u4	$16 \times 16 \times 512$
d4	$16 \times 16 \times 512$	u5	$32 \times 32 \times 256$
d5	$8 \times 8 \times 512$	u6	$64 \times 64 \times 128$
d6	$4 \times 4 \times 512$	u7	$128 \times 128 \times 64$
d7	$2 \times 2 \times 512$	u8	$256 \times 256 \times 64$

Table 4.1: Modified UNet output shapes

4.1.2.1 Distance-based loss

This loss term is responsible for penalizing the generator directly by comparing the generated output and the ground truth. We used the L1 loss for this purpose, as it is mentioned in [6], to maintain the low-frequency correctness and fidelity of the model. Denoting the input as x, and the ground truth as z, we will have:

$$L_{L1}(G) = \sum_{x,z} |z - G(x)| \tag{4.1}$$

4.1.2.2 GAN Loss Term

This loss term defines adversarial training between the generator and the discriminator. We denote the criterion for this term as f_c, and the patch scores for the real data and the generated data as y and $\hat{y}$, respectively. We denote a matrix of ones with the size of i as J_i and a matrix of zeros with the size of i as O_i. As a result, $J_{|y|}$ is a matrix of one with the size that corresponds to the matrix of y. The criterion can be any of the binary cross-entropy, mean squared error, or Wasserstein loss. We examined all these options and the results are discussed in Appendix B. We denoted the generator with

G, and the discriminator with D.

$$L_{GAN}(D) = \sum_{x,z} \left(f_c \left(D(x,z),\ J_{|y|} \right) \ + \ f_c \left(D(x, G(x)),\ O_{|\hat{y}|} \right) \right) \tag{4.2a}$$

$$L_{GAN}(G) = \sum_{x,z} \left(f_c \left(D(x, G(x)),\ J_{|\hat{y}|} \right) \right) \tag{4.2b}$$

4.1.2.3 Gradient Penalty Term

In this method, we employed the gradient penalty approach as [71] to regulate the feedback in the training process. Without gradient penalty, the discriminator would keep receiving strong feedback during training, leaving the generator far behind, and this phenomenon could hinder the convergence of the training process. To calculate the gradient penalty term, we need a random interpolation of the generated output $G(x)$ and its corresponding ground truth z. If we denote the interpolation with $\hat{z} = \alpha \odot z + (J_{|z|} - \alpha) \odot G(x)$, where the elements of matrix α are random values satisfying $0 < \alpha_{ij} < 1$, we can calculate gradient penalty using equation 4.3.

$$L_{gp}(G, D) = \sum_{x,z} \left(\|\nabla_{\hat{z}} D(x, \hat{z})\|_2 - 1 \right)^2 \tag{4.3}$$

4.1.2.4 Perceptual Loss Term

The generator has a much more difficult task to perform than the discriminator in our problem. Consequently, we also provided the generator with more hints about its performance using a perceptual loss term. Originally in [68], there are two perceptual loss terms referred to as feature reconstruction loss and style reconstruction loss. In our method, we only used the first term employing the feature maps from a VGG16

pretrained on ImageNet [9]. Finally, this loss term was calculated using equation 4.4 where each weighted VGG feature map is represented by ϕ_κ.

$$L_{per}\left(G\right) = \sum_{\kappa \in K} \sum_{x,z} \|\phi_\kappa\left(G\left(x\right)\right) \; - \; \phi_\kappa\left(z\right)\|_2^2 \tag{4.4}$$

In general, these loss terms will form our optimization problem or minimax game formulated as equation 4.5. we denoted the scalar for each loss term with λ.

$$G^* = arg \underset{G}{min} \; \underset{D}{max} \; L_{GAN}\left(G,\; D\right) + \lambda_a L_{L1}\left(G\right) + \lambda_{gp} L_{GP}\left(G,\; D\right) + \lambda_{per} L_{per}\left(G\right) \tag{4.5}$$

In the following, we will describe the details of the second proposed model.

4.2 2DPlotsTactile-GAN-Channelwise

After successfully generating results with 2DPlotsTactile-GAN and the observations that occurred on the effectiveness of the changes to the base model, we considered a more complex scenario where the output has multiple channels where each channel contains one of the components of a 2D plot. We updated our dataset accordingly and proposed a channelwise version of the proposed method which is referred to as 2DPlotsTactile-GAN-Channelwise. In the following, we will describe different components of the proposed 2DPlotsTactile-GAN-Channelwise model.

4.2.1 Generator

The generator in 2DPlotsTactile-GAN-Channelwise deals with channelwise output. To be more precise, we would like our model to generate multiple grayscale outputs all at once. There are different methods to arrange our desired output in multiple

channels based on the tricks we found in SemanticGAN [8]. We only compared UNet and UNet++ generators in this scenario, since the adversarial training of BCDUNet failed to converge in the first version of our model. Figure 4.2 shows the architecture of UNet++ and PatchGAN that we used as the generator, and the discriminator respectively.

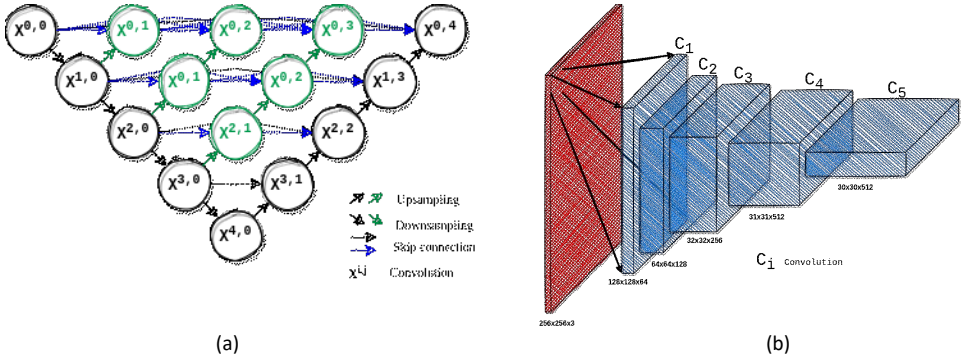

Figure 4.2: The schema of the proposed 2DPlotsTactile-GAN-Channelwise model which is based on the UNet++ and PatchGAN for the generator and discriminator, respectively. Models are modified to support RGB inputs and generate outputs that correspond to the channel size of ground truth.

4.2.2 Loss Function

The loss function of this model is very similar to the proposed 2DPlotsTactile-GAN model with some updated loss terms. In the following, we describe the proposed loss terms and highlight the changes we made to the 2DPlotsTactile-GAN loss.

4.2.2.1 Distance-based Loss

This loss term is the same as the last version since we still needed the L1 term to keep track of fine details.

4.2.2.2 GAN Loss Term

We added hinge loss as a candidate of f_c to our GAN loss term, but the structure is still the same.

4.2.2.3 Gradient Penalty Term

By making the generation task more complex, the rivalry between the generator and discriminator gets more uneven. There are several ways to bound the update speed of the discriminator. For instance, we can update discriminator weights every several epochs or with a smaller learning rate but we found a modified version of gradient penalty more helpful. We found it more efficient to penalize the discriminator for intense changes that come from strong gradient signals every few epochs. The advantage of this option is that we do not slow down the training process to ensure convergence. Instead, we let the training go on and intervene regularly to make sure that the discriminator is on track. To do so, we create a random matrix α again with the new condition of $0.5 < \alpha_{ij} < 1$. Now, our interpolation $\hat{z} = \alpha \odot z + (J_{|z|} - \alpha) \odot G(x)$ will always be more similar to the ground truth than the generated output. As a result, it is reasonable to expect that the interpolation should not cause a huge gradient norm when fed to the discriminator. We emphasize the difference by changing $L_{gp}(G,\ D)$

to $L_{gp}(D)$ in equation 4.6.

$$L_{gp}(D) = \sum_{x,z} \left(\|\nabla_{\hat{z}} D(x, \hat{z})\|_2 - 1\right)^2 \tag{4.6}$$

4.2.2.4 Perceptual Loss Term

VGG is a powerful image classifier that has rich feature maps. However, it is pretrained on a dataset that does not have much in common with our dataset. Also, we mentioned that discriminator training goes fast in our scenario. As suggested in [69], we use the feature maps of our discriminator instead of a third model such as VGG16. We only used the perceptual adversarial loss as a loss term for the generator, contrary to [69] which uses it as a new approach to train the entire model. This way, we have more relevant feature maps, and we do not have to load an extra model, saving more memory and computation when training. We still committed to feature reconstruction loss but we found the absolute difference more helpful than the second norm used in eq. 4.4. The perceptual adversarial loss term is denoted as 4.7 and ϕ_κ represents selected feature maps of the discriminator immediately after passing the activation function.

$$L_{per}(G) = \sum_{\kappa \in K} \sum_{x,z} \left|\phi_\kappa(G(x)) \; - \; \phi_\kappa(z)\right| \tag{4.7}$$

4.3 Summary

We propose our model as two working versions. The first version translates RGB images to tactile graphics also in RGB format. We compared UNet, UNet++, and BCDUNet as the generator of our model. In the loss function, in addition to the

adversarial loss term and L1 loss, we proposed to use gradient penalty and perceptual loss. As will be discussed in Chapter 5, the new terms help with the training process and improve the results.

The second version of our model generates channelwise output that performs image inpainting and object reconstruction when various components of a 2D plot occlude each other. For the generator, we examined a modified version of UNet and UNet++. The model based on BCDUNet did not converge in adversarial training. The loss term includes a more general form of adversarial loss and L1 loss, a modified gradient penalty term that is applied to the discriminator every few epochs to help with the convergence of the model, and an adversarial perceptual loss term that guides the generator by giving it information about the feature maps of the discriminator. The results of the proposed methods will be provided in the next chapter.

Chapter 5

Experimental Results

In this chapter, we learn about the experiments performed using our models. First, we introduce the dataset that we created for the translation of a 2D plot into the tactile format. Then, we describe the metrics we used for evaluating the performance of different models. Afterward, we explain the environment setup and training regime for our experiments, and finally, we provide an ablation analysis to demonstrate the effectiveness of different components of the proposed models.

5.1 Dataset

With recent advances in deep learning, we can include more tasks inside a single pipeline, without caring about feature extraction manually or intensive preprocessing or postprocessing. However, to make use of many of those models and solutions, we need a sufficiently large number of training samples. Creating such a dataset from an existing publication is time-consuming. Instead, we took a different route and created a synthesized dataset of 2D plots including Bézier curves, scatter plots, polygons, and bar charts. Due to the similarities between the first three categories, we trained a

model on those categories and trained a separate model for bar charts. We created a dataset of 5000 samples for training the first model and created a dataset of 5000 bar charts for training the second model. The structure of both models is the same. For each experiment, we took 10% of the data as testing set. In other words, there are 4500 samples for training and 500 unseen samples for testing. Basically, there were three approaches to represent the tactile outputs of the model.

1. We can generate RGB images having a channel responsible for each of the red, green, and blue colors. To measure how much the model is paying attention to different components of the input (comprehension), we can assign a color to each component of the output, or define keypoints (e.g. at intersections). This is a common approach in face or body pose estimation. Having such clues would also facilitate the evaluation of our model since we can define performance measures based on the keypoint recognition rate of the model. Figure 5.1 shows examples of the Bézier curves that we generated using this approach.

2. If we want to address our problem more similarly to semantic segmentation, we can represent the output as label maps. In RGB representation, we inevitably prioritize some components over others. More specifically, we can observe that the content comes over the axes and they are all together on top of gridlines. Finally, the white pixels of the background have the least priority. If we move all these white pixels to the first channel, black pixels to the second channel, blue pixels to the third channel, and red pixels to the fourth channel, we reach a new representation that can help us formulate our problem as a pixel-wise classification as in a standard semantic segmentation problem. In other words, in this representation, each pixel is a 4D vector one-hot encoded based on the

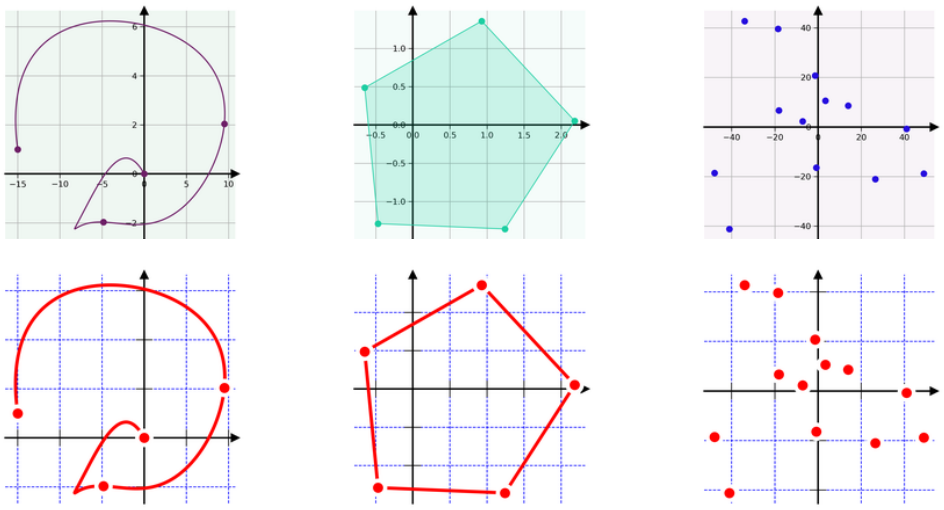

Figure 5.1: Examples of image pairs in our first approach of data representation. In the target domain (i.e. tactile) we assign black to the axes, blue to grid lines, and red to the content. We also highlighted the intersections on the curve with a specific marker. The first row is the input RGB image. The second row is the tactile equivalent in RGB format.

label of that pixel. For instance (1, 0, 0, 0) represents a background pixel and (0, 0, 1, 0) indicates a grid line pixel. Since we maintained the priority of components, we will face only a single occurrence of 1 in each vector. Figure 5.2 shows examples of this representation. The advantage of this representation is that, similar to semantic segmentation, we can use channelwise cross-entropy to compare generated output to the ground truth. However, the downside is that since each pixel belongs to exactly one class, if we remove a channel with higher priority (e.g. the content), axis and grid lines will have discontinuities. Figure 5.3 shows how we could face discontinuities using this approach. This phenomenon still makes us dependent on postprocessing generated output.

Another problem with this representation is the class imbalance since we assign

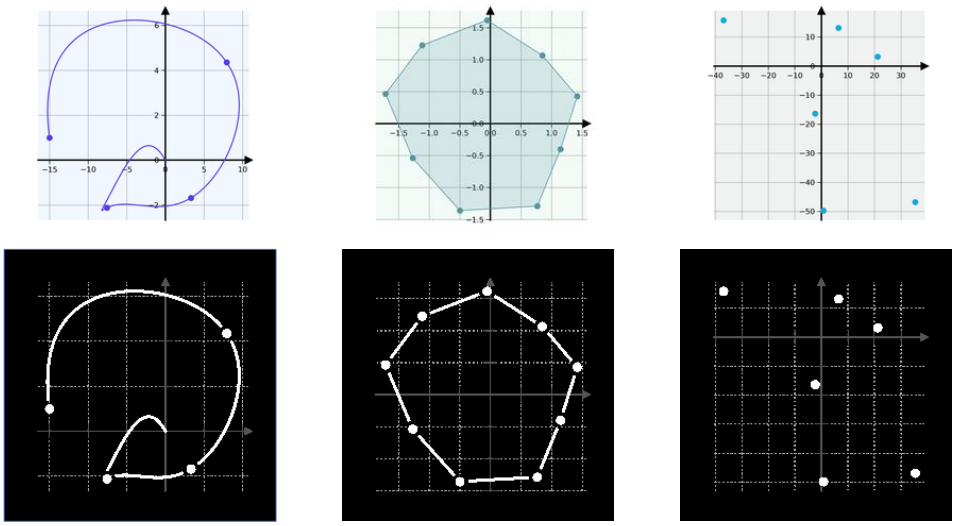

Figure 5.2: Three examples for the second representation. The top row belongs to the source domain, and the bottom row shows tactile images as segmentation maps. In this example, the background is labeled as 0, axes as 1, gridlines as 2, and the content as 3 which is the brightest. This is the way that we store data but the way that we decide to feed it to the model could be channelwise.

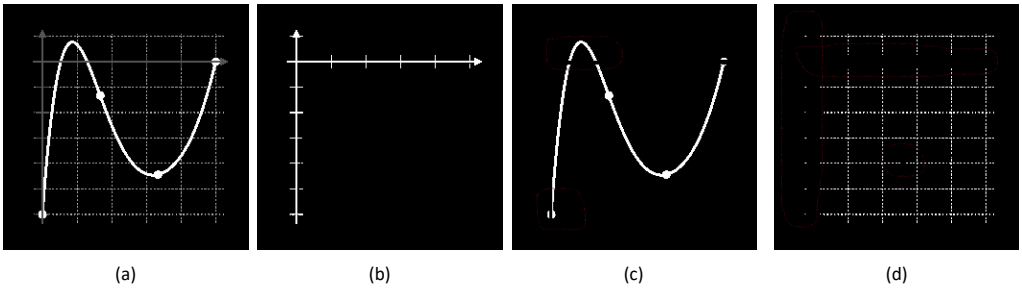

Figure 5.3: An example of discontinuities caused by representation 2. (a) is the assembled visualization of channels, and (b) shows the axes which look normal. However, there are discontinuities in content (c) and gridlines (d). The discontinuities are marked with thin red lines.

an actual class to the background, and the background covers a significant part of the image. Moreover, the visual quality of tactile images is reduced in this approach because we cannot use intermediate brightness intensities due to the one-hot vector representation of each pixel. Because of these issues, we eventually abandoned this representation and instead used the third approach, which will be described next.

3. We can make better use of channels by assigning each of them to one of the components. This way, there will be no 4D vectors for each pixel. However, we can add reconstructed components and eliminate component priorities. The model can treat each output channel as an independent image needed to be generated as we saw in [8]. Moreover, since we can generate grayscale images in each channel, the quality limitation will be resolved and the impact of class imbalance will be mitigated. Figure 5.4 shows examples of the approach. In this approach, unlike the previous approach, we do not assign a class label to each pixel. There is no class label. Instead, there are independent channels, where each channel is a grayscale image associated with a component of the 2D plots. Note that in this approach, the model is doing object reconstruction and inpainting, and because of that, there is no discontinuity even at locations where different components occlude each other in the input image. For example, if we look at the channel that represents the grid lines, there is no discontinuity.

5.1.1 Visualization

Ultimately, we used the third approach above to generate channelwise outputs where each channel is responsible for a component of the 2D plot. This is for the purpose

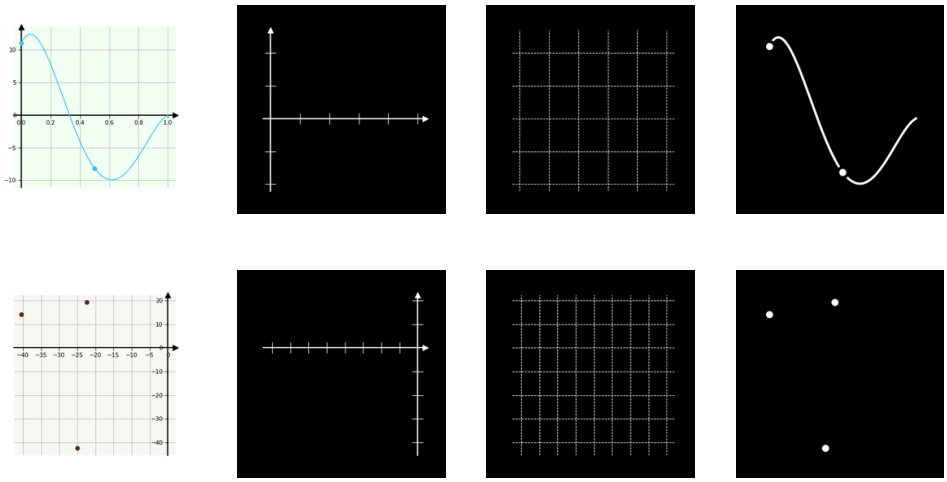

Figure 5.4: Two examples for the third representation. The left column shows the source domain. The other three columns show axes, gridlines, and content channels, respectively. Note that in this approach, the model is performing object reconstruction and inpainting and because of that there is no discontinuity even at locations where different components occlude each other in the input image.

of post-hoc modification by tactile designers. However, for the sake of a compact visualization throughout the thesis, we merge the channels, and assign different colors to different components, and display the channels in a single RGB image. Figure 5.5 shows some examples of how we again aggregated these objects into one image. Note that this is only for visualization. 2DPlotsTactile-GAN-channelwise indeed generates outputs with individual channels as defined in the third approach above.

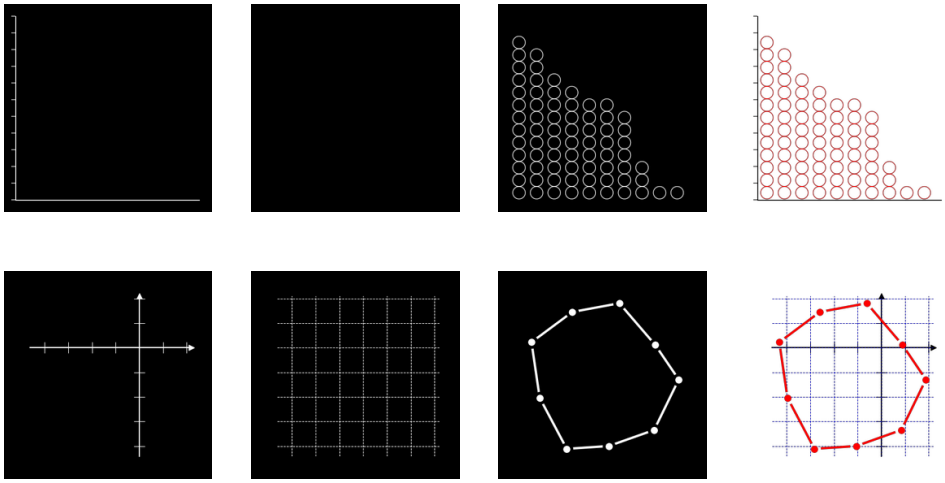

Figure 5.5: The process of merging components of a tactile image for visualization purposes. The hierarchy is maintained by plotting axes over gridlines and then pasting the content on top of them. This representation enables us to compare 2DPlotsTactile-GAN-channelwise outputs with those of 2DPlotsTactile-GAN.

5.2 2DPlotsTactile-GAN Results

5.2.1 Evaluation Metrics for RGB Tactiles

To compare the models and evaluate their results, we used quantitative and qualitative measures. There may be slight flaws in the results that we cannot perceive with our vision, so the numbers can become useful. But, basically, we need to generate images without artifacts, having certain visual standards to be useful for the designers and speed up their translation process, so we also need qualitative measures. For 2DPlotsTactile-GAN, we had to come up with creative measures to assess it quantitatively. Also, it should be noted that our samples consist mostly of white pixels related to the background (more than 82% on average) that do not carry so much information. In the following, we first describe the metrics we used for quantitative

evaluation. Then we explain our qualitative study.

5.2.1.1 Quantitative Measures for RGB Tactiles

We will use four metrics to quantitatively evaluate 2DPlotsTactile-GAN. Foreground MSE is defined as pixel-wise mean squared error between the foreground of the generated image and that of the ground truth. To isolate the foreground region, we created the binary masks depicted in Figure 5.6.

Background MSE is defined as pixel-wise mean squared error between the background of the generated image and that of the ground truth. We were able to ensure that the model did not add any artifacts to the generated output, using background MSE. We used the binary complement of the masks in Figure 5.6 to calculate this metric.

In addition to the above pixel-wise metrics, to further emphasize the intersection markers, we took them as the positive class, and calculated precision and recall. Precision is defined as $\frac{tp}{tp+fp}$ where tp is the number of intersections that are correctly generated, and fp is the number of redundant intersections generated by the model. The recall is defined as $\frac{tp}{tp+fn}$ where fn is the number of intersections that the model failed to generate. To facilitate the process of detecting intersection points for evaluation purposes, when creating the synthesized dataset, in the tactile domain, we colored the intersection markers with a color that is opposite of the random color of the curve. For instance, if the curve was brown (#A52A2A) the intersection color would be cyan (#5AD5D5). Table 5.1 shows our quantitative results, attained by the metrics above.

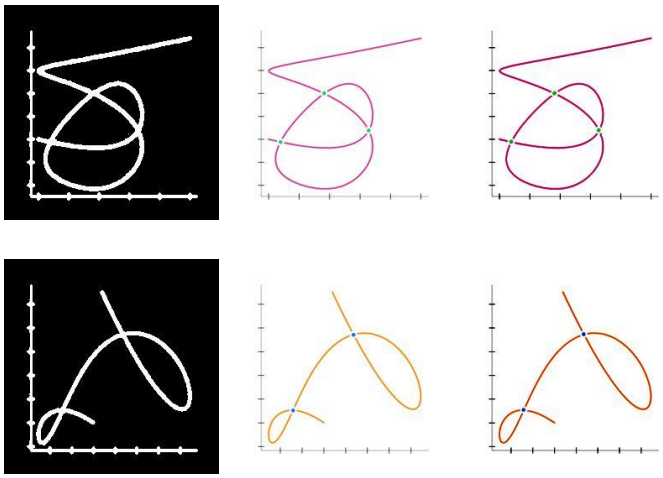

Figure 5.6: Generated binary masks (the left column) to extract foreground. The middle column shows the ground truth and the right column shows the generated output of 2DPlotsTactile-GAN.

5.2.1.2 Qualitative Measures for RGB Tactiles

We put emphasis on qualitative measures as well as quantitative measures because ultimately, visual quality has the highest priority for the designers. For qualitative comparison, we designed a web application that performs anonymous voting. We provided the human annotators with assorted images of the same sample generated by different models and asked them to rank them from best to worst visual quality compared to the ground truth. Then we calculate a score for each model by averaging the rank of the tactile graphics generated by that model. Results are averaged over annotations made by two human annotators on a subset 100 out of 500 randomly selected test images for each model. Figure 5.7, shows our web application used for qualitative analysis. We explained to the web application users that our criteria for visual quality includes less background and foreground artifact, accurate position and

color translation, and style consistency comparing to the ground truth. However, we did not bring any priority or weight to their judgement and asked them to rank the results in a way that seems natural and fair to them.

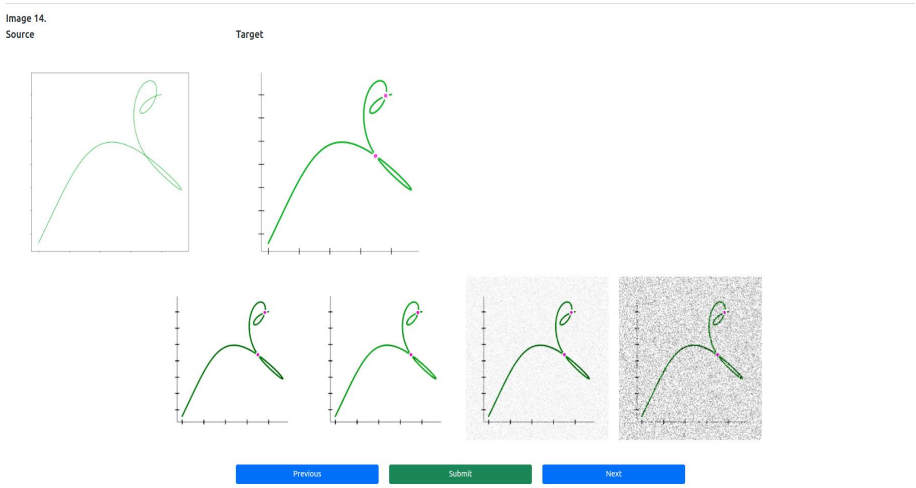

Figure 5.7: The web application used to evaluate models qualitatively. Annotators rank the tactiles generated by different models based on their similarity to the ground-truth.

5.2.2 Ablation Analysis

We took Pix2Pix as our base model. The generator was replaced with UNet++, applied gradient penalty, and finally added perceptual loss term. It's noteworthy that in this scenario, UNet performs suitably and the epochs take less time compared to UNet++. On the other hand, UNet++ produces a smaller model that uses less storage. In the last two columns, we report precision and recall based on the intersections detected by the model in a test set with 500 samples.

Method	Foreground MSE	Background MSE	Precision	Recall
Base model (Pix2Pix)	3.778e-3	4.074e-2	0.988	0.970
Base model, UNet++	3.736e-3	4.045e-2	0.994	0.982
Base model, UNet++, GP	3.732e-3	**4.017e-2**	**1.000**	**0.988**
Base model, UNet++, GP, Perceptual	**3.715e-3**	4.033e-2	**1.000**	**0.988**

Table 5.1: 2DPlotsTactile-GAN performance on Bézier curves, scatter plots, and polygons

	Base model (Pix2Pix)	Base model, UNet++	Base model, UNet++, GP	Base model, UNet++, GP, Perceptual
Avg rank (1-4)	2.58	2.40	2.56	2.50

Table 5.2: 2DPlotsTactile-GAN average ranks.

Figure 5.8 shows the generated outputs of four different models given the same Bézier curve. We used a test of 500 such samples for evaluation.

Based on Table 5.1 and the visual inspection of the generated outputs, we conclude that the proposed modifications reduce color mismatch and artifacts. They also outperform the base model in detecting intersections that shows itself specifically by higher precision and recall. Using the web application, we found out that changing the generator from UNet to UNet++ improves the average rank from 2.58 to 2.40 (lower is better). In many cases, annotators found it difficult to distinguish the difference between the outputs, generated by adding gradient penalty and perceptual loss term to the loss function. On average those models were ranked 2.56 and 2.50 respectively as shown in Table 5.2. In the following section, we will provide the results for generating channelwise tactiles.

5.3 2DPlotsTactile-GAN-Channelwise Results

5.3.1 Evaluation Metrics for Channelwise Tactiles

Similar to RGB tactiles, we considered both quantitative and qualitative measures to evaluate channelwise tactiles. Due to the different structures of RGB and channelwise

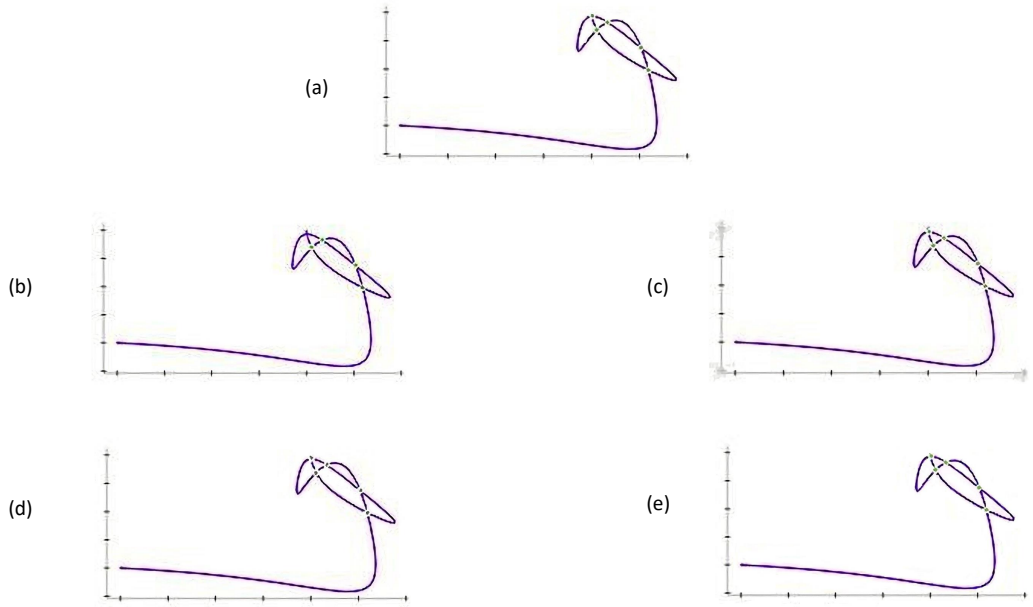

Figure 5.8: RGB tactiles generated by four different models. (a) the ground truth, (b) the generated output where the generator is unable to fully detect all intersections, (c) the generated output with artifacts and the wrong intersection color, (d) the generated output with the wrong intersection color, (e) the generated output from 2DPlotsTactile-GAN.

tactile formats, we had come up with metrics suitable for the channelwise tactiles. In the following, we will elaborate on the metrics that we used, and the reason why we found those metrics more informative and helpful.

5.3.1.1 Quantitative Measures for Channelwise tactiles

In the evaluation of RGB tactiles, we used metrics based on the pixel values i.e. color of pixels. However, in the channelwise format, channels are associated with various components of the 2D plots. Each channel contains one component independently, which means our evaluation will not be affected by occluding components. As a result, the channelwise approach helped us evaluate the models more naturally, and use quantitative metrics that better align with the human's perception of tactiles. We use Pixel Accuracy, Dice Coefficient (F1 score), and Jaccard Index (IoU), which are popular measures in semantic segmentation. More precisely, we calculate these metrics for each channel separately and report the average as the score of an instance. In each channel, if we denote pixels that are truly recognized as foreground with *tp*, redundant pixels that are added by the model (artifact pixels) with *fp*, pixels that the model missed to generate (missing foreground pixels) with *fn* and pixels that are correctly assigned to the background with *tn*, the formulation of each evaluation metric is defined as follows:

$$pixel\,accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{5.1}$$

$$Dice\,coefficient = \frac{2tp}{2tp + fp + fn} \tag{5.2}$$

$$Jaccard\,index\ =\ \frac{tp}{tp\ +\ fp\ +\ fn} \tag{5.3}$$

5.3.1.2 Qualitative measures for channelwise tactiles

Similar to RGB tactile generation, we took 100 randomly selected samples from 500 samples in the test set. Then, we used the same web application and asked two users to rank the models without knowing that which output is related to which model. The results confirmed that 2DPlotsTactile-GAN-Channelwise generates higher quality tactiles and performs significantly better than the base model. Table 5.5 shows the details of our evaluation.

5.3.2 Ablation Analysis

In this section, through an ablation study, we analyze the effectiveness of various components of the proposed method compared to the base Pix2pix model. We first start by changing the generator, the reason being that the original Pix2Pix model tends to generate fair results when we were working on RGB output, but it somehow failed to keep up as we moved to the more difficult task of generating a channelwise output. We refer to this model as the "Base model, UNet++". Then, we added gradient penalty loss term to improve training stability and finally added perceptual loss term to further improve the output quality. After the first model began to generate meaningful results, we stopped training other models to keep them equally trained. Consequently, other models might perform better given more time to train. Figures 5.9 and 5.10 show the results generated by the checkpoints that we considered for ablation analysis. They represent 2D plots and bar charts respectively. Also, we trained the models on 2D plots first, then we performed hyperparameter tuning using our training

set of 4500 image pairs, and finally used the exact same set of hyperparameters for bar charts to make sure of their validity. Although, the quality of the generated tactiles for 2D plots is generally better than bar charts. However, the translation task is initially harder on bar charts since the source domain and the target domain have more differences. Ultimately, we have to evaluate the robustness and generalization capability of our models by testing them on real-world data. To be more clear, we assumed the samples that are not synthesized by us as real-world data.

Method	Pixel Accuracy	Dice Coefficient	Jaccard Index
Base model (Pix2Pix)	0.58	0.03	0.01
Base model, UNet++	0.83	0.07	0.04
Base model, UNet++, GP	0.98	0.24	0.17
Base model, UNet++, GP, Perceptual	**0.98**	**0.28**	**0.21**

Table 5.3: 2DPlotsTactile-GAN-Channelwise performance on Bézier curves, scatter plots, and polygons

Method	Pixel Accuracy	Dice Coefficient	Jaccard Index
Base model (Pix2Pix)	0.82	0.06	0.03
Base model, UNet++	0.86	0.05	0.02
Base model, UNet++, GP	0.94	0.15	0.10
Base model, UNet++, GP, Perceptual	**0.98**	**0.31**	**0.28**

Table 5.4: 2DPlotsTactile-GAN-Channelwise performance on bar charts

	Base model (Pix2Pix)	Base model, UNet++	Base model, UNet++, GP	Base model, UNet++, GP, Perceptual
Avg rank (1-4)	3.72	2.96	2.24	1.08

Table 5.5: 2DPlotsTactile-GAN-Channelwise average ranks.

We also plotted the loss function through the training process. Figure 5.11 shows the trend in loss function for each variation of the proposed method. In the base model, the generator loss shows that it lacks learning capability since the magnitude saturated at a large value after a few epochs. By changing the generator to UNet++, we observed that the generator loss stopped at a lower magnitude. However, there

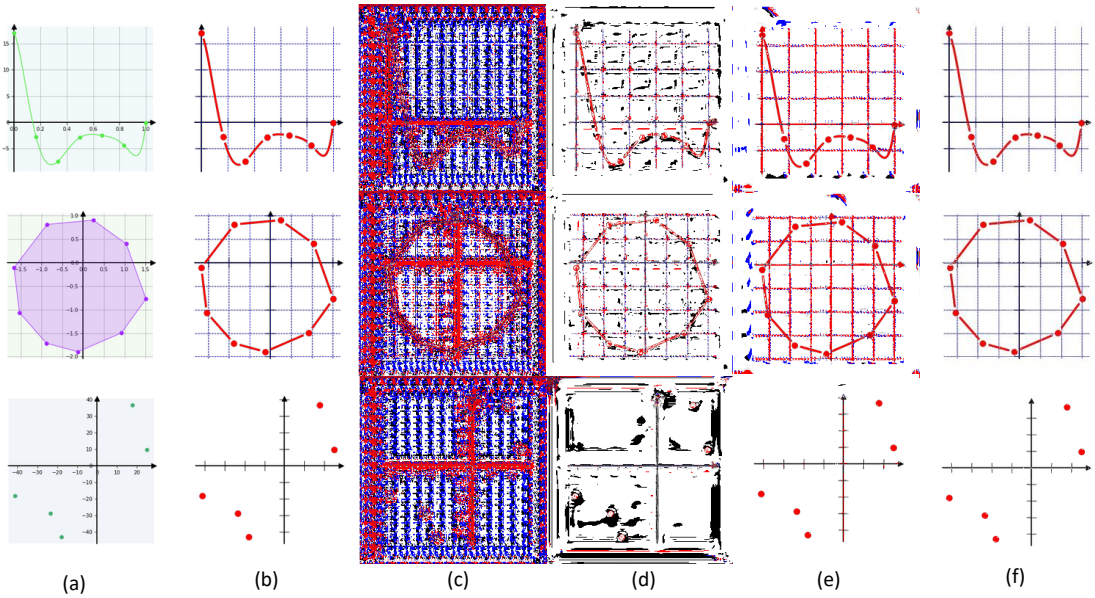

Figure 5.9: Examples of 2D plot tactiles generated by various channelwise models when the best-performing model converges. Columns from left to right represent (a): input, (b): ground truth, (c): base model, (d): base model with UNet++, (e): base model with UNet++, and GP, (f): based model with UNet++, GP, and perceptual loss.

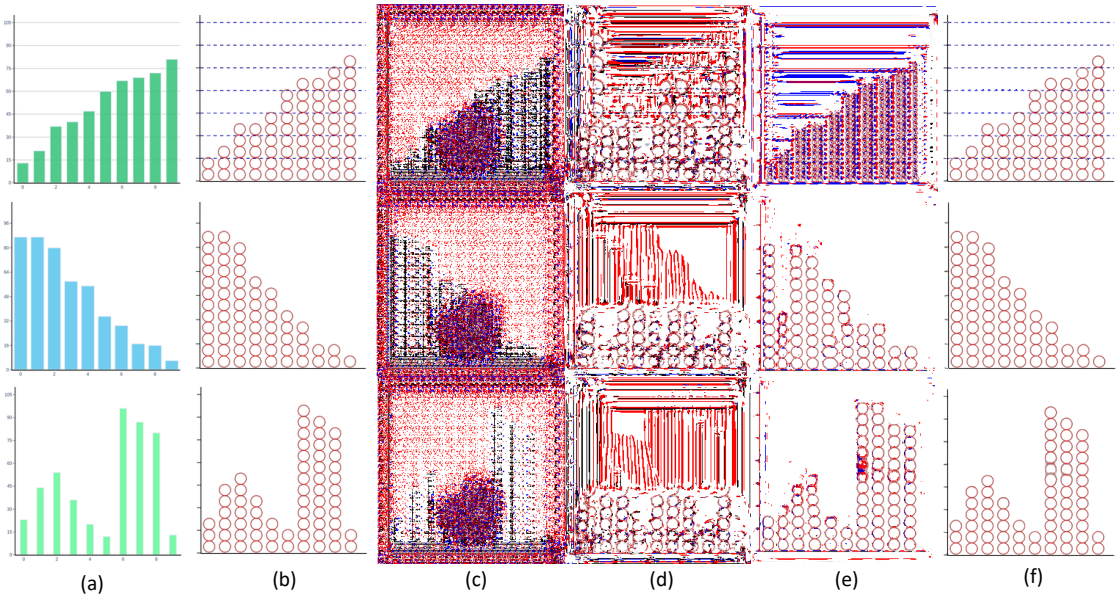

Figure 5.10: Examples of bar chart tactile graphics generated by various channelwise models when the best-performing model converges. Columns from left to right represent (a): input, (b): ground truth, (c): base model, (d): base model with UNet++, (e): base model with UNet++, and GP, (f): based model with UNet++, GP, and perceptual loss.

are more fluctuations in the loss values. By using gradient penalty, we realized that adversarial training leads to a lower generator loss and the stability is improved. In the end, adding perceptual loss slightly improved the training loss trends.

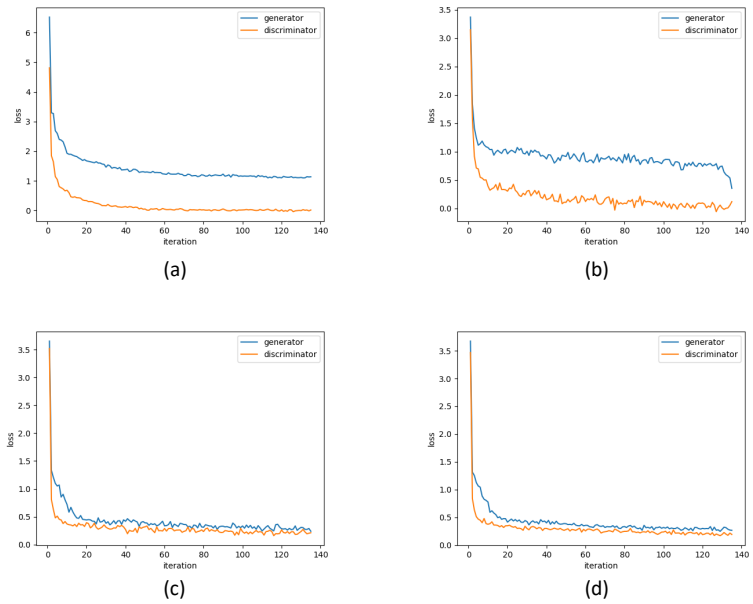

Figure 5.11: Loss variation through the training process. Generator loss is plotted with blue, and discriminator loss with orange. The plot shows the training for 135 epochs. (a) is related to the base model. (b) is related to replacing the generator with UNet++. (c) is related to the model with UNet++ generator, and gradient penalty. Finally, (d) is related to the addition of the perceptual loss term to the last model.

The proposed 2DPlotsTactile-GAN-Channelwise model outperforms the base model in pixel accuracy, Dice coefficient, and Jaccard index. The base model fails to generate desirable channelwise tactiles. It can be seen that the proposed modifications to the based model, including the use of UNet++ as the generator and the addition of gradient penalty, and perceptual loss, significantly improves the results. Figures 5.9 and 5.10 show that our proposed model can reach quality results while the outputs of the base model severely suffer from artifacts. UNet++ increased the learning

capability of the model. However, without using gradient penalty the model was unable to properly distinguish different classes and often grid lines, and some parts of the axes were confused with the content. Finally, by using perceptual loss, we could make the model more sensitive to fine details.

5.4 Real-world Scenarios

To test the robustness of our proposed model on real-world test samples, we surfed Google images to find 2d curves, scatter plots, polygons, and bar charts. Figure 5.12 shows the images generated by 2DPlotsTactile-GAN, and Figure 5.13 shows those generated by 2DPlotsTactile-GAN-Channelwise. Over all, there is more room for improvement in generalization, that can be achieved by adding more diversity to the training samples. We observed that 2DPlotsTactile-GAN performs better than 2DPlotsTactile-GAN-Channelwise on generalization having clearer results with less artifact, and more accuracy classifying each component. Comparing different categories, 2DPlotsTactile-GAN-Channelwise finds it harder to generalize bar charts than other categories. This can be due to the fact that the translation task is initially more difficult on bar charts.

5.5 Summary

In this chapter, we introduced the approaches that we take to populate our dataset. We stated that we used 10% of the samples (500 image pairs) as the test set and avoided our model to be trained on them. Then we showcased the results of the proposed 2DPlotsTactile-GAN which generates RGB tactiles, and 2DPlotsTactile-GAN-channelwise which generates channelwise tactiles where each channel is associated

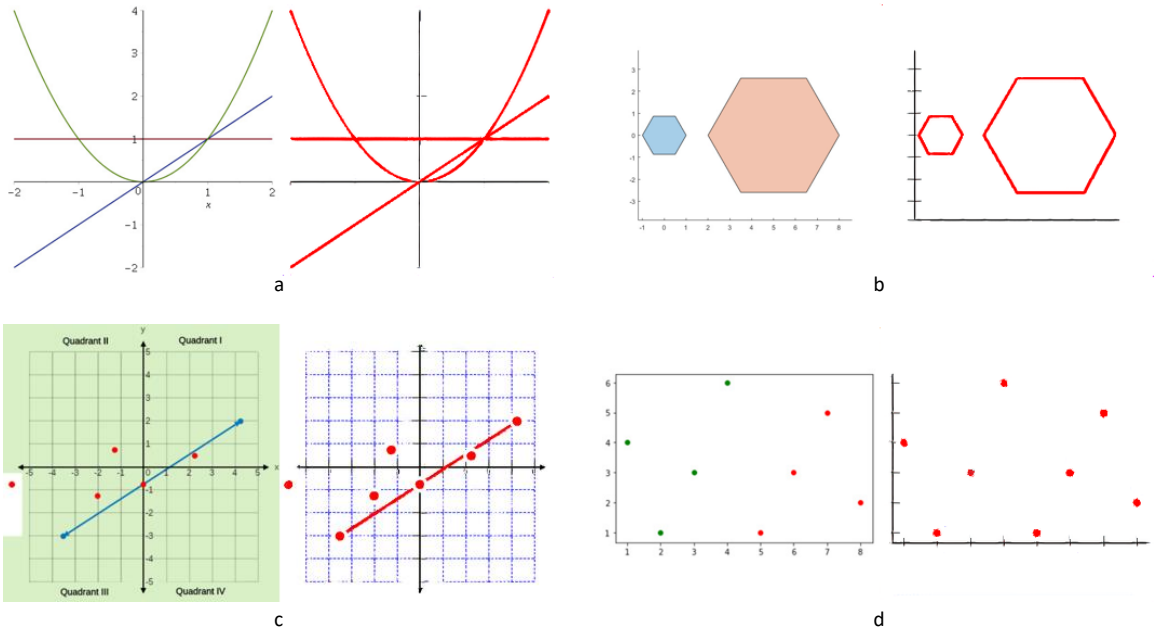

Figure 5.12: 2DPlotsTactile-GAN performance in real-world scenarios. (a) includes multiple curves which is new to the model, and the model could detect them all as the content. (b) includes multiple polygons that was not covered in our training set. (c) includes a 2d curve and scatters with different color, on an intense colored background but the model successfully converted the components. (d) includes scatters with different colors which was not implemented in our dataset, but the model successfully detected them.

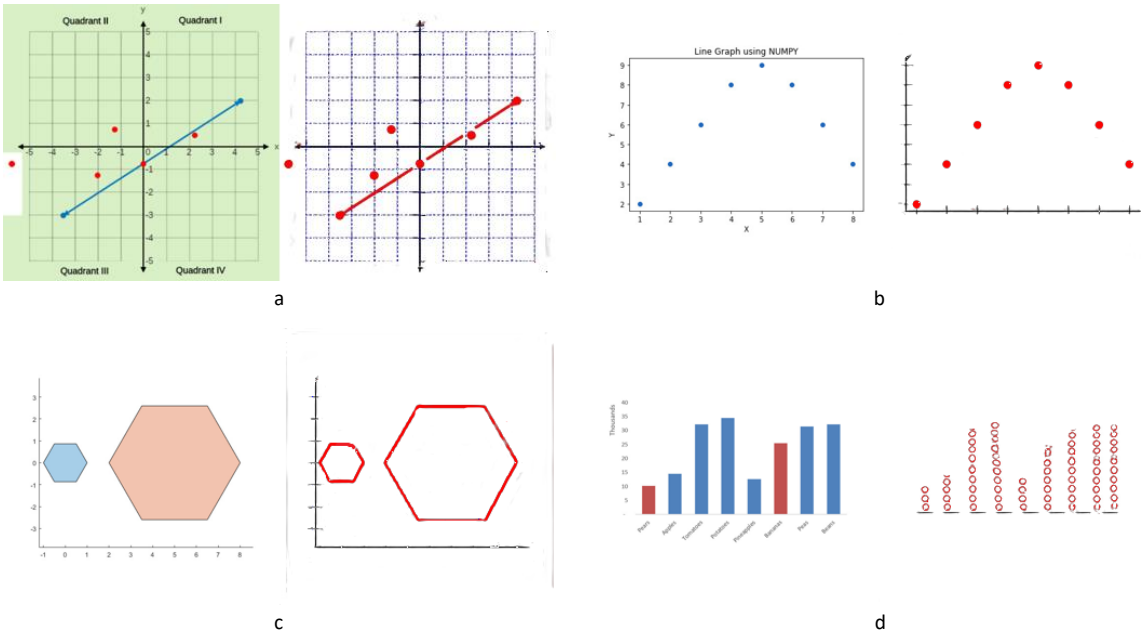

Figure 5.13: 2DPlotsTactile-GAN-Channelwise performance in real-world scenarios. (a), (b), and (c) shows that with channelwise outputs, the model can still keep up with RGB model in generalization. (d) shows a bar chart with a style different from our dataset samples, but the model is able to detect the bars and draw them as stacked circles.

with a component of the 2D plot. We described the metric used for the quantitative evaluation of RGB tactiles and channelwise tactiles. We also described the web application developed for qualitative evaluation, the criteria, and results of that evaluation for RGB and channelwise tactiles. The result of the visual inspections through our web application available in Tables 5.2 and 5.5 showed a better average rank after adding each modification to the base model i.e. the base model took the last rank, and 2DPlotsTactile-GAN-Channelwise took the first place. We also demonstrated the effectiveness of each component added to the base model using the results of quantitative metrics in tables 5.3, and 5.4. The task of generating channelwise tactiles is more challenging because the network has to learn to reconstruct occluding components through an implicit inpainting process. The proposed method, in particular, outperformed the base model by a large margin in that task. We also explained our hyperparameter tuning process, tuning the hyperparameters on 2d plots test set and validate them on bar charts test set. We elaborated on the details of hyperparameter tuning in Appendix B.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we explored the problem of generating tactile graphics using deep generative models. We focused on 2D curves, polygons, scatter plots and bar charts. There are still more similar categories to be addressed such as pie charts, maps, and molecule diagrams. Initially, tactile is a grayscale image. However, for editable results, we needed to be able to access individual components of the generated 2D plots. for example, axes, grid lines, and the curve. We proposed two approaches to achieve this. The first approach was based on using third-party software such as Potrace and CorelDraw for converting tactile images to SVG format which can be edited in software such as CorelDraw. To enhance the process, we trained our model to generate outputs where each component of the 2D plot has a unique color. This guides the third-party software to identify the components in the desired way.

In the second approach, the output has multiple channels, each is associated with a component of the 2D plot (axes, gridlines, content, etc.). While the first approach

was consistent with typical image-to-image translation where the output is a grayscale of the colored image, it required third-party software to achieve editable results. On the other hand, the second approach eliminated the need for third-party software because it provides direct access to different components.

Both of the proposed models are based on the pix2pix model. Through extensive experiments, we arrived at the proposed models. In summary, we changed the generator of the pix2pix to UNet++ and updated its loss function by adding perceptual loss and gradient penalty terms.

Initially, we proposed 2DPlotsTactile-GAN, trained it in the RGB target domain, and evaluated its results using quantitative and qualitative metrics. Quantitative metrics consisted of foreground MSE, background MSE, precision, and recall. We also used a Web application as a means of qualitative evaluation. We noticed that all variations could often generate acceptable results, though our proposed model was able to outperform the base model by a small margin.

On the combined category of Bézier curves, scatter plots, and polygons, the proposed channelwise method improved the Pixel Accuracy of the base model from 0.58 to 0.98. It improved the Dice Coefficient from 0.03 to 0.28, and Jaccard Index from 0.01 to 0.21. Similarly, on bar charts, the proposed channelwise method improved the Pixel Accuracy of the base model from 0.82 to 0.98. It improved the Dice Coefficient from 0.06 to 0.31, and Jaccard Index from 0.03 to 0.28.

6.2 Limitations and Future Work

In this thesis we focused on 2D curves, polygons, scatter plots and bar charts. There are certainly more image categories to convert to tactile format. Bar charts need a

more advanced level of abstraction learned by the model to be translated into tactile graphics compared to other 2D plots since the bars should completely change their appearance to be drawn as circles. Similarly, there may be other categories that may be more challenging for our model to translate to a tactile format. There is still a potential to add more diversity to the samples that we included in those certain type of plots. In real-world scenarios, the source images themselves may be grayscale or have artifacts and noise due to the scanning process. Hence, we need a model that is more robust to these layouts. An effective way to implement such robust models is to add more variations of the same categories that we focused on, like by modifying the components of the source domain, and add samples with different source image quality by introducing random noise as an image augmentation scheme.

In this thesis, an input image was translated into a single output image. Compared to RGB images, a tactile has much less capacity to contain information because otherwise the resulting tactile would be too crowded and difficult to understand by touch. For more complex categories of images, such as maps, the common practice is to convert the input to multiple tactiles. For example, one for showing routes, one for showing cities, one for rivers, etc. Archiving this using the existing approaches is challenging.

Finally, another limitation of our current approach is the inability to generalize to new categories. Training a separate model for each category is not sample efficient. On the other hand, it is not clear how much more we can vary our training dataset. The model may forget what it has learned from three categories if we expand it to work on ten categories. Addressing these issues can be a good direction for future work.

Appendices

Appendix A

Dataset Inspection

In this part, we provide more details about our dataset and data augmentation. There are a few standards that we maintained in our tactile format.

- The line width of the axes and content is set to 2pt.
- Axes should have in-out tick lines with a length of 15-30 pt.
- Grid lines should be plotted with dashed lines.
- Scatters should have a size of 35-50 pt having a white margin around them to point them out.
- Bar charts should be plotted using circles stacked on top of each other.

To prevent overfitting, and make the model robust to unseen samples we included three aspect ratios and centered the result regardless, to achieve squared images to be fed to our model. The candidates for the aspect ratio were 1:1 with 50%, 1:2 with 25%, and 2:1 with 25% shares. We drew gridlines for 40% of the samples in every dataset and also 50% of the Bézier curves had at least one intersection, and we collect

them as a subset to perform our experiments on. In case of a train/test split, we took 10% of each dataset as unseen data and reported the performance of our model based on that. In our data loader, we perform a horizontal flip with a chance of 50%, shift with a limit of 10%, scale with a limit of 20%, rotate with a limit of 15 degrees, and partial occlusion with a chance of 50%.

Appendix B

Experiment Settings

We developed our dataset generation script in Python using Numpy, Matplotlib, and Plotly. The model itself was developed using PyTorch in Anaconda environment. We also ran debugging and small-scale training on an Nvidia RTX-3060 GPU and performed mass training on Nvidia V100l provided by Compute Canada. There were a couple of hyperparameters that we tuned globally before performing the ablation analysis.

B.1 Hyperparameter Tuning

In the following, we describe each hyperparameter and its effect.

- Batch size: Controls the number of samples in each batch. In case of resources, a lower batch size will cause each epoch to take more time, and a higher batch size will increase memory consumption. In the original Pix2Pix paper, the authors insisted on setting the batch size to 1. However, we found that a batch size of 4 performs better. Since our approach separates different objects into

different channels, we should prevent the model from being dependent on the percentage of each object existing in the dataset. For example, since only 40% of the samples contain gridlines, setting the batch size to 1 will lead to the model losing generalization ability and fluctuations in the learning curve.

- Total iterations: Controls the total number of epochs for an experiment. We set it to 135 to avoid the code from taking more than a day for each run. We also have the option to load the result of a previous experiment and extend the learning process.

- Learning rate: Regulates the step size at each epoch while minimizing the loss function. We set the learning rate to 0.002 globally. We employed multistep learning rate decay with another hyperparameter for epochs with a constant learning rate. We trained the model for 25 epochs with a constant learning rate. Then we took 10 checkpoint epochs to apply the learning rate decay. After each checkpoint, we multiplied the learning rate by 0.8 and consequently ended with a learning rate which is $0.8^{10} \approx 0.1$ of the initial learning rate.

- Label smoothing: We only performed label smoothing on the positive label of the discriminator which is used when the network is fed with real images [72]. To do so, we replaced the matrix of ones with a normal distribution around 1 that is clipped for values greater than 1.

- L1 scalar: Regulates the value of the L1 loss term in the loss function. Since L1 loss values are comparably smaller by an order of magnitude, we used 5 for this hyperparameter.

- GP scalar: Regulates gradient penalty with other loss terms. Since GP values

can grow to significantly larger values than the other terms, we used 0.1 for this hyperparameter.

- Perceptual scalar: Regulates the absolute value of the perceptual loss relative to other loss terms. Since we do not intend to give strong feedback to the generator, and the perceptual loss term only aims to make the adversarial training more balanced for the generator and the discriminator, we used 0.2 for this hyperparameter.

- Perceptual weights: We used four feature maps from the discriminator in our perceptual loss and based on our observations demonstrated in figure B.1 we took [0, 0.1, 0.3, 0.6] to calculate the weighted sum of the perceptual loss of the feature maps.

- GAN loss: we implemented least-square error, cross-entropy, Wasserstein, and hinge loss in our-GAN loss term and took least-square as the default. cross-entropy is only applicable when we tried to treat the problem as a pixel classification and assigned a 4D vector of probabilities to each pixel. Wasserstein and hinge loss did not make any improvements either.

- Discriminator regulation frequency: This parameter controls how often we use gradient penalty to slow down the training of the discriminator. By default, we apply GP regularization every 4 epochs.

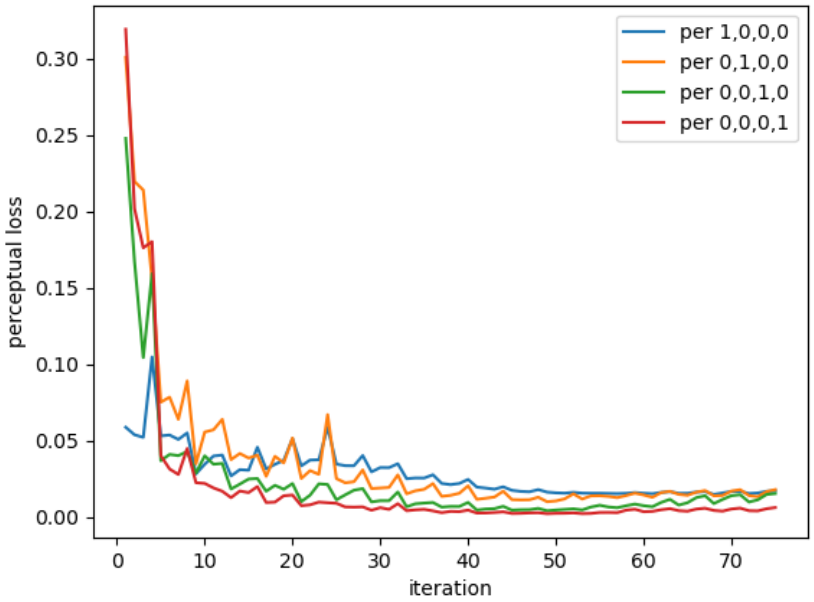

Figure B.1: The individual effect of each of four feature maps extracted from the discriminator, on the perceptual loss term. We trained similar models using only one of the feature maps at a time. The legends show the index of the feature map that we used ([1,0,0,0] for the first, and [0,0,0,1] for the fourth feature map). Eventually, we set the weights based on the effectiveness of each feature map in training.

Bibliography

[1] Joseph Mathieu. Communicating visual information to the vision impaired. `https://newsroom.carleton.ca/story/communicating-visual-information-vision-impaired`. Published: 2022-04-27.

[2] Richard E Ladner, Melody Y Ivory, Rajesh Rao, Sheryl Burgstahler, Dan Comden, Sangyun Hahn, Matthew Renzelmann, Satria Krisnandi, Mahalakshmi Ramasamy, Beverly Slabosky, et al. Automating tactile graphics translation. In *Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, pages 150–157, 2005.

[3] Jianjun Chen and Noboru Takagi. A pattern recognition method for automating tactile graphics translation from hand-drawn maps. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 4173–4178. IEEE, 2013.

[4] Radim Štampach and Eva Mulíčková. Automated generation of tactile maps. *Journal of Maps*, 12(sup1):532–540, 2016.

[5] Tetsuya Watanabe, Toshimitsu Yamaguchi, Satoko Koda, and Kazunori Minatani. Tactile map automated creation system using openstreetmap. In *International Conference on Computers for Handicapped Persons*, pages 42–49. Springer, 2014.

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

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[8] Daiqing Li, Junlin Yang, Karsten Kreis, Antonio Torralba, and Sanja Fidler. Semantic segmentation with generative models: Semi-supervised learning and strong out-of-domain generalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8300–8311, 2021.

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

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

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

[12] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28, 2015.

[15] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[17] Zixuan Huang, Jinghuai Zhang, and Jing Liao. Style mixer: Semantic-aware multi-style transfer network. In *Computer Graphics Forum*, volume 38, pages 469–480. Wiley Online Library, 2019.

[18] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[19] Krishna Regmi and Ali Borji. Cross-view image synthesis using conditional gans. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 3501–3510, 2018.

[20] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2337–2346, 2019.

[21] Peihao Zhu, Rameen Abdal, Yipeng Qin, and Peter Wonka. Sean: Image synthesis with semantic region-adaptive normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5104–5113, 2020.

[22] Cheng-Han Lee, Ziwei Liu, Lingyun Wu, and Ping Luo. Maskgan: Towards diverse and interactive facial image manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5549–5558, 2020.

[23] Hao Tang, Dan Xu, Yan Yan, Philip HS Torr, and Nicu Sebe. Local class-specific and global image-level generative adversarial networks for semantic-guided scene generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7870–7879, 2020.

[24] Qianye Yang, Nannan Li, Zixu Zhao, Xingyu Fan, Eric I Chang, Yan Xu, et al. Mri cross-modality neuroimage-to-neuroimage translation. *arXiv preprint arXiv:1801.06940*, 2018.

[25] Xi Guo, Zhicheng Wang, Qin Yang, Weifeng Lv, Xianglong Liu, Qiong Wu, and Jian Huang. Gan-based virtual-to-real image translation for urban scene semantic segmentation. *Neurocomputing*, 394:127–135, 2020.

[26] Rui Li, Wenming Cao, Qianfen Jiao, Si Wu, and Hau-San Wong. Simplified unsupervised image translation for semantic segmentation adaptation. *Pattern Recognition*, 105:107343, 2020.

[27] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *International conference on machine learning*, pages 1857–1865. PMLR, 2017.

[28] Zili Yi, Hao Zhang, Ping Tan, and Minglun Gong. Dualgan: Unsupervised dual learning for image-to-image translation. In *Proceedings of the IEEE international conference on computer vision*, pages 2849–2857, 2017.

[29] Youssef Alami Mejjati, Christian Richardt, James Tompkin, Darren Cosker, and Kwang In Kim. Unsupervised attention-guided image-to-image translation. *Advances in neural information processing systems*, 31, 2018.

[30] Matteo Tomei, Marcella Cornia, Lorenzo Baraldi, and Rita Cucchiara. Art2real: Unfolding the reality of artworks via semantically-aware image-to-image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5849–5859, 2019.

[31] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[32] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. *Advances in neural information processing systems*, 30, 2017.

[33] Yuhang Song, Chao Yang, Zhe Lin, Xiaofeng Liu, Qin Huang, Hao Li, and C-C Jay Kuo. Contextual-based image inpainting: Infer, match, and translate. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[34] Ming-Yu Liu, Xun Huang, Arun Mallya, Tero Karras, Timo Aila, Jaakko Lehtinen, and Jan Kautz. Few-shot unsupervised image-to-image translation. In *Proceedings*

of the IEEE/CVF international conference on computer vision, pages 10551–10560, 2019.

[35] Lei Zhao, Qihang Mo, Sihuan Lin, Zhizhong Wang, Zhiwen Zuo, Haibo Chen, Wei Xing, and Dongming Lu. Uctgan: Diverse image inpainting based on unsupervised cross-space translation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5741–5750, 2020.

[36] Hsiao-Yu Fish Tung, Adam W Harley, William Seto, and Katerina Fragkiadaki. Adversarial inverse graphics networks: Learning 2d-to-3d lifting and image-to-image translation from unpaired supervision. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4354–4362, 2017.

[37] Bowen Li, Xiaojuan Qi, Thomas Lukasiewicz, and Philip HS Torr. Manigan: Text-guided image manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7880–7889, 2020.

[38] Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S Lin, Tianhe Yu, and Alexei A Efros. Real-time user-guided image colorization with learned deep priors. *arXiv preprint arXiv:1705.02999*, 2017.

[39] Patricia L Suárez, Angel D Sappa, and Boris X Vintimilla. Infrared image colorization based on a triplet dcgan architecture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 18–23, 2017.

[40] Mingming He, Dongdong Chen, Jing Liao, Pedro V Sander, and Lu Yuan. Deep exemplar-based colorization. *ACM Transactions on Graphics (TOG)*, 37(4):1–16, 2018.

[41] Bo Zhang, Mingming He, Jing Liao, Pedro V Sander, Lu Yuan, Amine Bermak, and Dong Chen. Deep exemplar-based video colorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8052–8061, 2019.

[42] Zhongyou Xu, Tingting Wang, Faming Fang, Yun Sheng, and Guixu Zhang. Stylization-based architecture for fast deep exemplar colorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9363–9372, 2020.

[43] Junsoo Lee, Eungyeup Kim, Yunsung Lee, Dongjun Kim, Jaehyuk Chang, and Jaegul Choo. Reference-based sketch image colorization using augmented-self reference and dense semantic correspondence. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5801–5810, 2020.

[44] Yuan Yuan, Siyuan Liu, Jiawei Zhang, Yongbing Zhang, Chao Dong, and Liang Lin. Unsupervised image super-resolution using cycle-in-cycle generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 701–710, 2018.

[45] Yongbing Zhang, Siyuan Liu, Chao Dong, Xinfeng Zhang, and Yuan Yuan. Multiple cycle-in-cycle generative adversarial networks for unsupervised image super-resolution. *IEEE transactions on Image Processing*, 29:1101–1112, 2019.

[46] Zak Murez, Soheil Kolouri, David Kriegman, Ravi Ramamoorthi, and Kyungnam Kim. Image to image translation for domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4500–4509, 2018.

[47] Jinming Cao, Oren Katzir, Peng Jiang, Dani Lischinski, Danny Cohen-Or, Changhe Tu, and Yangyan Li. Dida: Disentangled synthesis for domain adaptation. *arXiv preprint arXiv:1805.08019*, 2018.

[48] Alexander H Liu, Yen-Cheng Liu, Yu-Ying Yeh, and Yu-Chiang Frank Wang. A unified feature disentangler for multi-domain image translation and manipulation. *Advances in neural information processing systems*, 31, 2018.

[49] Yichun Shi, Debayan Deb, and Anil K Jain. Warpgan: Automatic caricature generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10762–10771, 2019.

[50] Maciej Pęśko, Adam Svystun, Paweł Andruszkiewicz, Przemysław Rokita, and Tomasz Trzciński. Comixify: Transform video into comics. *Fundamenta Informaticae*, 168(2-4):311–333, 2019.

[51] Ziqiang Zheng, Chao Wang, Zhibin Yu, Nan Wang, Haiyong Zheng, and Bing Zheng. Unpaired photo-to-caricature translation on faces in the wild. *Neurocomputing*, 355:71–81, 2019.

[52] Yang Chen, Yu-Kun Lai, and Yong-Jin Liu. Cartoongan: Generative adversarial networks for photo cartoonization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9465–9474, 2018.

[53] Junho Kim, Minjae Kim, Hyeonwoo Kang, and Kwanghee Lee. U-gat-it: Unsupervised generative attentional networks with adaptive layer-instance normalization for image-to-image translation. *arXiv preprint arXiv:1907.10830*, 2019.

[54] Xinrui Wang and Jinze Yu. Learning to cartoonize using white-box cartoon representations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8090–8099, 2020.

[55] Moab Arar, Yiftach Ginger, Dov Danon, Amit H. Bermano, and Daniel Cohen-Or. Unsupervised multi-modal image registration via geometry preserving image-to-image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[56] Chen Liu, Jiajun Wu, Pushmeet Kohli, and Yasutaka Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2195–2203, 2017.

[57] Uche Osahor, Hadi Kazemi, Ali Dabouei, and Nasser Nasrabadi. Quality guided sketch-to-photo image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 820–821, 2020.

[58] Ye Lin, Shenggui Ling, Keren Fu, and Peng Cheng. An identity-preserved model for face sketch-photo synthesis. *IEEE Signal Processing Letters*, 27:1095–1099, 2020.

[59] Jieying Zheng, Wanru Song, Yahong Wu, Ran Xu, and Feng Liu. Feature encoder guided generative adversarial network for face photo-sketch synthesis. *IEEE Access*, 7:154971–154985, 2019.

[60] Mingrui Zhu, Jie Li, Nannan Wang, and Xinbo Gao. A deep collaborative framework for face photo–sketch synthesis. *IEEE transactions on neural networks and learning systems*, 30(10):3096–3108, 2019.

[61] Jun Yu, Xingxin Xu, Fei Gao, Shengjie Shi, Meng Wang, Dacheng Tao, and Qingming Huang. Toward realistic face photo–sketch synthesis via composition-aided gans. *IEEE transactions on cybernetics*, 51(9):4350–4362, 2020.

[62] Yuke Fang, Weihong Deng, Junping Du, and Jiani Hu. Identity-aware cyclegan for face photo-sketch synthesis and recognition. *Pattern Recognition*, 102:107249, 2020.

[63] Shu-Yu Chen, Wanchao Su, Lin Gao, Shihong Xia, and Hongbo Fu. Deep generation of face images from sketches. *arXiv preprint arXiv:2006.01047*, 2020.

[64] Yuhang Li, Xuejin Chen, Binxin Yang, Zihan Chen, Zhihua Cheng, and Zheng-Jun Zha. Deepfacepencil: Creating face images from freehand sketches. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 991–999, 2020.

[65] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 3–11. Springer, 2018.

[66] Reza Azad, Maryam Asadi-Aghbolaghi, Mahmood Fathy, and Sergio Escalera. Bi-directional convlstm u-net with densley connected convolutions. In *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pages 0–0, 2019.

[67] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[68] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[69] Chaoyue Wang, Chang Xu, Chaohui Wang, and Dacheng Tao. Perceptual adversarial networks for image-to-image transformation. *IEEE Transactions on Image Processing*, 27(8):4066–4079, 2018.

[70] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020.

[71] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[72] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.