CRAFTING THE WEIGHTS OF A CONVOLUTIONAL NEURAL NETWORK TO MAKE A DRAWING

By

ERIK ULBERG

CARNEGIE MELLON UNIVERSITY

School of Architecture

College of Fine Arts

Thesis

Submitted in Partial Fulfillment of the requirements for the degree of

Master of Science in Computational Design

TITLE:

Crafting the Weights of a Convolutional Neural Network to Make a Drawing

AUTHOR:

Erik Ulberg

ACCEPTED BY ADVISORY COMMITTEE:

[redacted] May 11, 2020

Daniel Cardoso Llach, Principal Advisor — Date

[redacted] May 11th, 2020

Daragh Byrne, Advisor — Date

[redacted] 5/12/2020

Golan Levin, Advisor — Date

ACKNOWLEDGMENTS

I'd like to thank Daniel Cardoso Llach for being the chair of my advising committee. His feedback always kept me critical of technical artifacts and focused on the humans involved. Daragh Byrne was a consistent force in shaping this thesis through all of its transformations. His advice was invaluable for clarifying what I was making while at the same time keeping the work true to myself. Golan Levin, whose presence at CMU first drew my interest to the school, was an excellent teacher and supportive mentor who pushed me to make stuff. He also connected me to Kyle McDonald who helped me to understand my drawing algorithm.

Many other faculty assisted along the way. Molly Wright Steenson gave me a broader view of AI in society and was a patient and thoughtful guide while I searched for a thesis that fit my background and interests. Eunsu Kang provided further exposure to machine learning art and a community of students trying to figure out what it was about. I also thank Tai Sing Lee for nurturing curiosity and exploration among his students. His support of my creative projects allowed me to deepen my understanding of computational perception and its connection to art.

I thank my peers from the Computational Design program, especially Vincent Mai, Yi-Chin Lee, Ian Friedman, Yixiao Fu, Yaxin Hu, and Emek Erdolu who participated in the Code Club reading group and made time out of their busy schedules for other fun activities.

I'm grateful to my mother, Marilyn Watkins, and brother, Carl Ulberg, for their enduring support through all my explorations. Also, a special shout out goes to my non-biological brother Patrick Williams for his technical mentoring related to this project.

Finally, I'd like to dedicate this thesis to my loving partner Claire who made sure I took breaks, listened to my endless musings, and assisted as a test subject for presentations and editor for drafts. I couldn't have stayed sane or enjoyed the process without you!

CRAFTING THE WEIGHTS OF A CONVOLUTIONAL NEURAL NETWORK TO MAKE A DRAWING

Abstract

by Erik Ulberg, Masters
Carnegie Mellon University
May 2020

A growing number of visual artists use convolutional neural networks (CNNs) in their practice. While CNNs show promise as a form of representation in art, the lack of interpretability of CNNs limits creative control to high level decisions around datasets, algorithms, and hyperparameters. As an alternative, the field of computer vision presents a more immediate paradigm of control through the hand-crafting of convolutional kernels. This thesis investigates the hand-crafted approach as an additional creative lever for artists working with CNNs. It reimagines network weights as a continuous, spatial, and computational material supporting direct human interaction. Two experimental tools are proposed: one for parametrically generating first layer kernels and the other for editing multiple layers. These tools attempt to transform the hand-crafting of features into "crafting" in a truer sense by bringing CNNs and visual materials into a close feedback loop. The author extensively engaged with these tools and this serves as a case study that examines the affordances of hand-crafted CNNs. The results suggest that hand-crafted CNNs can be a viable form of representation for artists seeking to build simple, bespoke feature detectors, but that more complex CNNs would likely require a hybrid approach integrating data-driven methods.

TABLE OF CONTENTS

Chapter One

Introduction

> *...to talk of one's internal model of the world is to talk of a representation, clearly. But it is not a fixed, coherent representation, the way a representation on a sheet of paper may be thought of as fixed and coherent. It takes very little introspection to discover that the pictures we conjure up in our heads are anything but complete.*
>
> Harold Cohen

1.1 Forms of Representation

When I sit down to make art, I have a vision in mind. There is a *thing* that I want to exist in the world. My vision for it may not be clear, but I have an intuition. AI art pioneer Harold Cohen referred to this intuition as the "volatile" mental image (Cohen, 1982). For me, art making is putting this ephemeral internal idea into an external representation. I don't expect others to see what I see, and usually they don't. But, occasionally, that compelling internal *thing* becomes a compelling external *thing*. External transmission is an act of clarification for myself and a method of sharing with others.

These creative concepts do not exist in a vacuum. They emerge out of connections I make between inspirational experiences and forms of representation that I have available. In that way, the tools I use and the forms of representation that they enable shape how I think

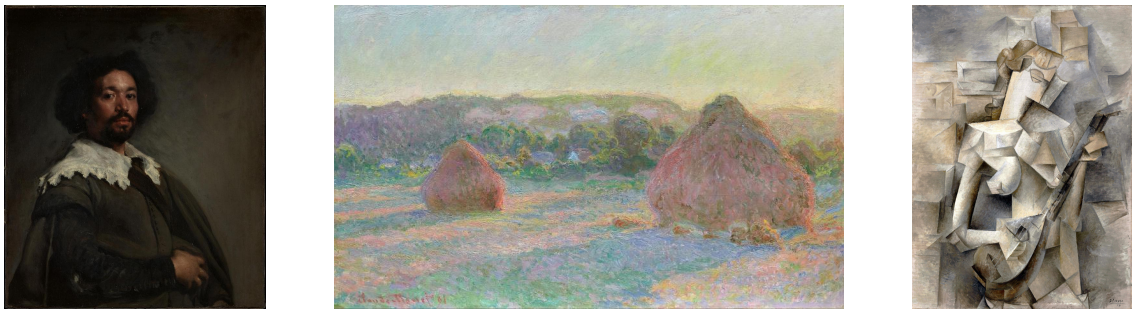
about the world. A new form of representation can transform a creative practice. Two of my favorite artists made mid-career transitions after being inspired by new ways of working that ended up defining their legacies. Conceptual artist Theo Jansen discovered the kinetic possibilities of yellow PVC pipes using an evolutionary algorithm in 1990 (Jansen, 2020). Ever since then, we have been treated to his Strandbeest sculptures populating the beaches of the Netherlands. Cohen was introduced to programming in 1968 after a successful stint as an abstract painter (McCorduck, 1991). AARON, his AI painting program, was born shortly thereafter. New forms of representation cause new connections between internal concepts and potential realizations.

(a) (b)

Figure 1.1 (a) The Strandbeest *Sabulosa Cutis* (Jansen, 1994). (b) *82P2* (Cohen and AARON, 1981).

Artists have always sought out novel forms for their unique affordances for externalizing ideas. The tools and methods used in image-based artwork have morphed over time to transmit nature realistically, capture momentary sensations, combine multiple views of an object, or convey other aspects of the essence of images.

This search has expanded past the confines of specific artifacts to rule-based artwork, which exists beyond a fixed, physical form. Artists specify a visual concept through symbolic logic that generates a universe of possible outputs. Conceptual artist Sol LeWitt described his

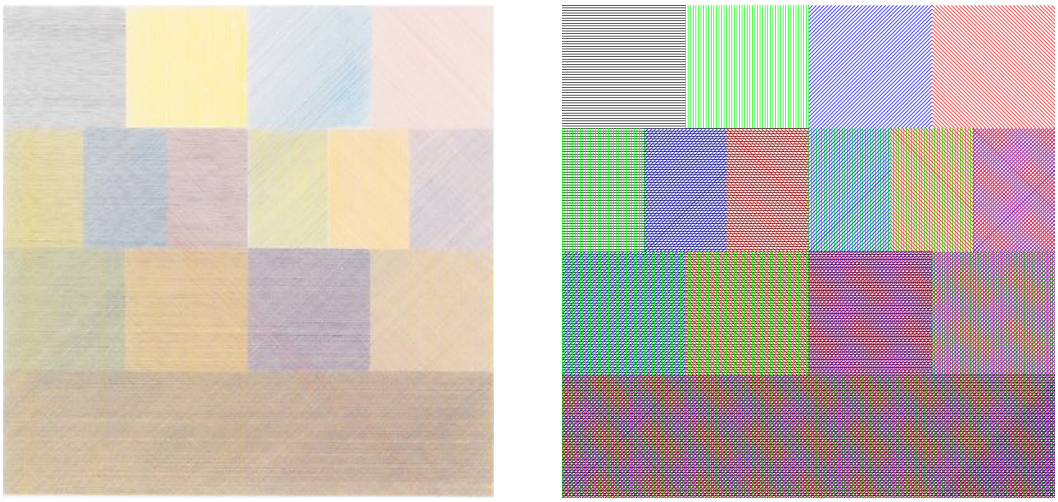


(a) (b) (c)

Figure 1.2 (a) A naturalistic painting (Velázquez, 1650). (b) A painting of a momentary impression (Monet, 1890). (c) A painting capturing multiple angles (Picasso, 1910).

wall drawings with diagrams and rules such as "A Wall is divided into four horizontal parts. In the top row are four equal divisions..." (Reas, 2004; LeWitt, 1970b). Theorists George Stiny and James Gips brought a mathematical formalization they called *shape grammars* to production rules for design (Stiny and Gips, 1971). Shape grammars eschew words and use the shapes themselves to define rules. Artworks that exist in these forms can be executed by anyone, or by any thing. For example, computational artist Casey Reas ported LeWitt's rules to a creative coding platform in 2004 (Reas, 2004). Combined with computation, rule-based approaches have the potential to dynamically yield many artworks.

Early forms of AI art operated on the assumption that building up enough of these rules within a computer program would result in artistic agents that mimicked human cognition. Cohen's drawing machine AARON is an iconic example of these so-called "expert systems." Each of AARON's hundreds of small functions were interpretable, but once these functions were arranged within many layers of feedback the results were highly unpredictable (Cohen, 1979). Much has changed since AARON's creation in the early 1970's, but the drawings and paintings produced by that system still loom large. In my opinion, they stand as some of the most satisfying artworks made with AI, or for that matter, computation in general.



(a) (b)

Figure 1.3 (a) *Untitled, from Composite Series* (LeWitt, 1970b). (b) The drawing executed by a computer (Reas, 2004).

1.2 Machine Learning Art

More recently, machine learning has enabled the "automatic" encoding of rules within a computer by inferring them from examples (I use quotes here because there are many hidden efforts in these systems that often go unnoticed). Instead of directly describing an artwork with symbolic logic, artists use a data-driven approach to generate it computationally. They gather images or other data and train the system to mimic the input in interesting ways.

Machine learning art often involves leveraging a convolutional neural network (CNN) as the primary method of encoding visual concepts. A CNN is a specific type of neural network that translates between raw pixels and abstract information by applying filters as a sliding window across an image. CNNs as a form of representation for art making are the main focus of this thesis.

(a) (b)

Figure 1.4 (a) Cover art from *The Book of GANesis* (Sarin, 2019a). (b) *Neural Glitch* (Klingemann, 2018).

1.3 The Limits of Convolutional Neural Networks

Standard applications of machine learning have demonstrated that CNNs are a robust form of representation. CNNs can encode highly complex and abstract visual concepts. As a result, this technology is making inroads in many commercial settings such as self-driving cars and medical diagnostics (Hu, 2019; Metz, 2019). Given the litany of recent successes using CNNs to work with images, there is clearly a latent potential in this medium to act as a form of representation for artists.

However, the manner in which CNNs encode visual concepts is not well understood. Deep networks record information using tens of millions of weights. Thus, direct manipulation seems to be a fool's errand and we leave the networks to be trained by algorithms in a data-driven process. When artists employ CNNs, control is mostly limited to trial and error with different datasets, algorithms, and hyperparameters.

But with limited direct control, how will mastery emerge in the realm of machine learning art? One critic mused that the field is completely reliant on novelty and is a race to see

who can be the first to use each new algorithm or dataset. The author argued that neural networks have no aesthetic potential and are only suitable for use in an art context as a critique of commercial practices (Offert, 2019). Before we declare neural networks a pictorial dead-end, I think we should re-examine our approach.

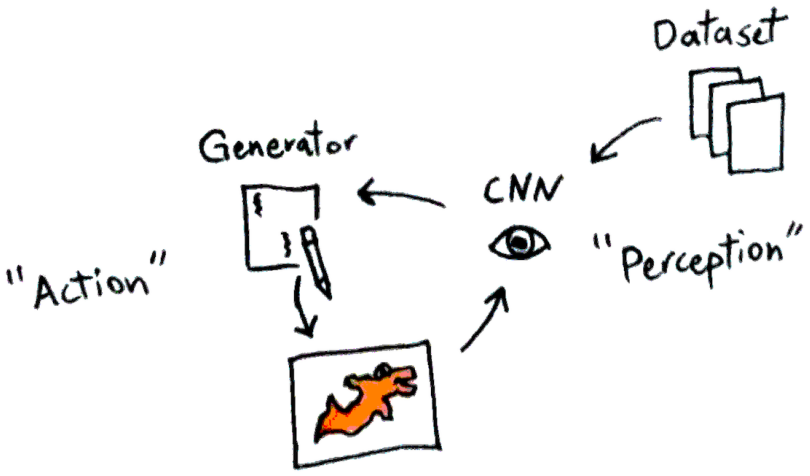


Figure 1.5 Most visual art made with machine learning can be boiled down to this setup.

This is not to overlook the existing examples of strong work in machine learning art. Self-proclaimed "folk AI artist" Helena Sarin incorporates CNNs within a technique she calls "NeuroBricolage" (Sarin, 2018; Sarin, 2019b). Using datasets of her own paintings and photographs, she generates textures and images with CNNs and then post-processes them by hand, custom code, or additional neural networks to produce digital collages. Sarin is one of a number of machine learning artists using alternative approaches such as custom datasets, smaller networks, or creatively-structured outputs (Ridler, 2017; Akten, 2017; Barrot and Barrat, 2019). While these avenues hold promise, this thesis takes a different approach that may yield exciting results of its own or simply provide a better intuition of how CNNs can represent visual concepts within art.

1.4 Thesis Vision

This thesis attempts direct manipulation of the internal weights of CNNs to make art. **It does not use machine learning**. There is no gradient descent or training of weights. Instead, this project adopts a data structure from machine learning, the convolutional neural network, and uses human reasoning to shape them. While machine learning art happens on the outside of CNNs, this thesis goes inside.

My approach was born out of a desire to mix insights from AARON's expert AI with current methods using CNNs for art making. It is an experiment to flesh out an idea. Prominent AI researcher Christopher Olah asks the provocative question: "What if we treated individual neurons, even individual weights, as being worthy of serious investigation?" (Olah et al., 2020b). I wanted to see if the right tool could transform CNNs into a continuous, spatial, and computational material capable of being directly sculpted by a human.

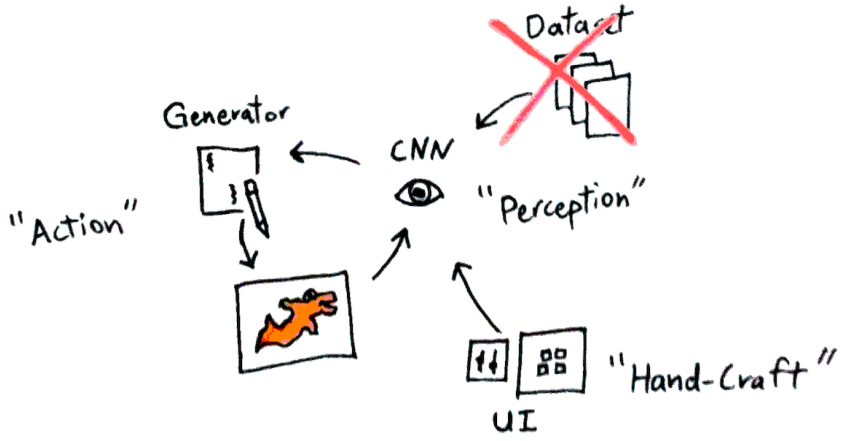


Figure 1.6 This thesis replaces the data-driven approach with direct manipulation.

This project is an attempt to enable engagement with CNNs as a craft. It borrows the notion of "hand-crafted" weights from the field of computer vision, where the term is widely used to describe feature detectors designed by humans instead of automatically calculated from data. These hand-crafted features are usually restricted to the first layer of CNNs due

to the lack of interpretability of networks. However, recent research into representations and interactive visualizations suggests the possibility of pushing our comprehension into the realm of shallow networks (Olah et al., 2020a).

While hand-crafted features typically cater to generic image recognition tasks, this thesis re-imagines hand-crafting as a creative lever for artists working with machine learning. The tools demonstrated in this project put editable convolutional kernels in direct communication with visual materials, thus transforming the process into "crafting" in a truer sense.

This brings me to my research question:
*What are the **affordances** and **uses cases** of hand-crafted convolutional neural networks as a form of representation for art-making?*

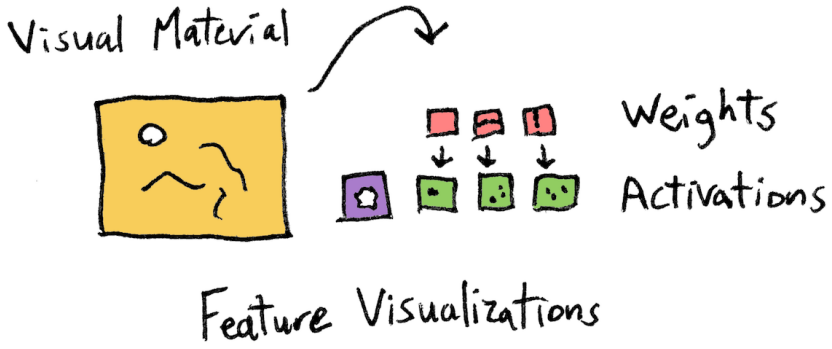


Figure 1.7 The tools in this thesis are meant to make visible and support the manipulation of weights in a CNN with respect to a particular visual material.

This document attempts to partially demystify the magic of the so-called "black box" of CNNs. It demonstrates a potential way for artists to achieve specific control over CNNs and clarifies the underlying aesthetic value proposition of CNNs for art-making. Overall, the goal of this project is to uncover what layers of convolution can do for artists as a form of representation.

Computational Design is fundamentally a tool-making enterprise that critically exam-

ines the role of technical approaches applied to creative practice. In that vein, this thesis consists of building two interactivate visualization tools and then using them to explore the affordances of hand-crafting CNNs. Producing interactive artifacts helps move an idea past speculation. Technical probes clarify questions and provide material with which to reason about a theoretical premise.

Personally, my use of software as a probe was inspired by architectural researcher Nicholas Negroponte's development of URBAN5. URBAN5 was a software system for urban design that examined the "desirability and feasibility of conversing with a machine about an environmental design program." This highly experimental project was never functional and "inexhaustibly printed garbage," but served as the primary research material to better understand the theoretical premise of collaborating with an AI agent within a specific context (Negroponte, 1970). The process of realizing a system, regardless of whether it fully works, helps to uncover questions, challenges, and potential futures. Harold Cohen echoed this sentiment when explaining his development of AARON, describing it as a way for him to "pin" down aesthetic principles (McCorduck, 1991). Hypothesizing about an experience cannot substitute for a confrontation with the actual substance.

I should mention that the system I am building is not meant to automate art. Cohen said that it is a misconception that artists need easy tools. In fact, they need tools that are "difficult to use — not impossible, but difficult." That way, they "stimulate a sufficient level of creative performance, which does not happen with tools that are easy to use." (McCorduck, 1991) I do not see the future of machine learning as replacing the labor of human artists, but as a mode of communication allowing the computer to become a richer medium.

1.5 Summary of Thesis

This thesis is divided into chapters. First, the background chapter traces my journey from finding inspiration in the aesthetics of AARON, to art making with CNNs, to the experimen-

tal method employed in this thesis: hand-crafting convolutional kernels. The main topics it covers are AI art, CNNs, interactive visualizations of CNNs, and hand-crafting features in CNNs. Following that, the methods chapter explains my research approach and the tools I built. The next chapter covers the results and discussion of the research investigations. It examines the affordances related to the tools and then explores the practice of hand-crafting CNNs through drawings, compositions, and conceptual artwork. Following these investigations, the conclusion summarizes the findings of the thesis and describes potential future work.

Chapter Two

Background

> *Human art-making behavior is characterized by the artist's awareness of the work in progress.*
>
> Harold Cohen

This chapter begins by examining the aesthetic contributions of Harold Cohen's AARON through its historical development and a partial reconstruction of its functionality. I connect this to CNNs and then, after briefly explaining the basics of convolution, I review projects that use CNNs for image making. Next, I discuss interpretations of the inner workings of CNNs and interactive visualizations that support these interpretations. Finally, I describe methods of hand-crafting weights in CNNs.

2.1 Understanding the Aesthetics of AARON

I find the aesthetic qualities of AARON's output to be uniquely exceptional among AI art. This led me to examine the details of its creation and implementation to help inform my own approach to making art with AI tools.

Cohen's key insight was that art making programs should focus on process rather than output. He theorized that human drawings are interesting mainly because they are made by humans (Cohen, 1976). To make machine drawings interesting, they should be produced in a

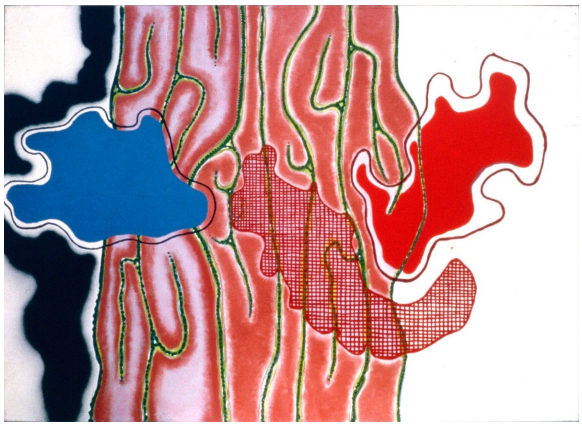
Figure 2.1 *Athlete Series* (Cohen and AARON, 1986).

similar manner. AARON did not necessarily match "the 'facts' of the human system," but it mimicked the "fluently changing pattern of decision-making which characterizes the practice of art" (Cohen, 1973). The decision-making used by AARON was infused by Cohen as part of a deep relationship with the system that was nurtured over time. This led Cohen to refer to it as an "expert's system" rather than an expert system (Cohen, 1988). The emphasis being on the machine as an extension of himself rather than as a replacement for his labor. Cohen saw his act of programming AARON as its own form of art-making (McCorduck, 1991). To better understand the origin of AARON's aesthetic, we should examine its historical development and the perspective of its expert.

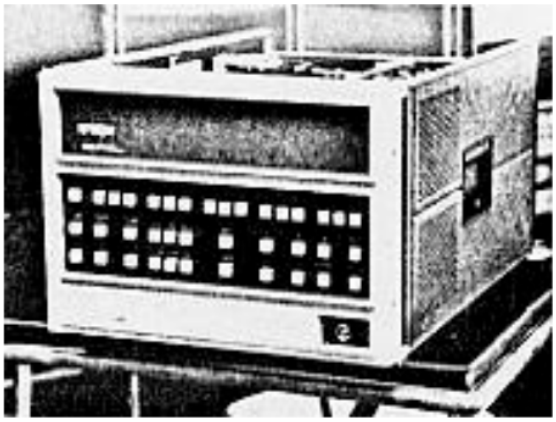
The Development of AARON

Cohen brought a unique background to computational art. He had a successful career as a painter before AARON. He was named one of five artists to represent Great Britain in the Venice Biennale in 1966. Shortly after, Cohen moved to the United States to teach art at UC San Diego, where he was introduced to programming by one of his students. (McCorduck, 1991)

In 1968, Cohen wrote his first computer program (McCorduck, 1991). His early programs were concerned with the division of space and human perception of open versus closed forms (Cohen, 1976). Cohen had a strong theoretical and aesthetic underpinning for his work carrying influences from artificial intelligence, computational design, and conceptual art.



(a)



(b)

Figure 2.2 (a) *Search* (Cohen, 1964). (b) A Hewlett Packard 2100 A, one of the early systems Cohen mentions in his work (Cohen, 1974).

Cohen began AARON by writing the simplest program capable of creating evocative shapes. Its basic function was to differentiate between figure and ground, open and closed forms, and insideness and outsideness (Cohen, 1976). As a painter, Cohen had an interest in symbols and the grammar of shapes. Through AARON, he was searching for a universal language of visual primitives, finding inspiration in glyph paintings on rocks he encountered

on hikes in the Southwest (Cohen, 1979). Cohen was also influenced by the idea of grammars. He explicitly talks about left and right hand rules for his program, echoing contemporary ideas found in the shape grammars of Stiny and Gips. Although, their grammars existed in an abstract world of shapes while AARON's production rules manipulated a pixel-like representation of the world (Cohen, 1976; Stiny and Gips, 1971). In addition, Cohen seems to have drawn from contemporary themes in conceptual art. AARON was a very literal extension of Sol LeWitt's declaration "the idea becomes a machine that makes the art" (LeWitt, 1967).

Shortly after beginning work on AARON, Cohen developed relationships with the nascent AI community. He spent two years at Stanford's AI Laboratory working with Edward Feigenbaum, a pioneer in AI expert systems (Grimes, 2016). Cohen developed a view of computers as analogous to the human brain. He shared the common conception of the era that a computer could model human thinking by stringing enough if-statements together (McCorduck, 1991). Cohen had read economist and AI pioneer Herb Simon's Sciences of the Artificial and believed in the usefulness of computers as a rigorous test bed for understanding human intelligence (McCorduck, 1991; Simon, 1969). He claims to have come to these conclusions on his own, but they were, at the very least, reinforced through his exposure to these groups (McCorduck, 1991).

Over time, AARON grew organically. Cohen always focused on the experience of the viewer in relation to the piece and its construction. He exhibited his work with the robot actively producing the pieces, thus allowing visitors to watch the process of decision making unfold in front of them. His initial intention was for viewers to make interpretations based on the shapes. After he added explicit representations, the focus shifted to relationships among shapes (McCorduck, 1991). AARON continued to evolve until Cohen's death in 2016.

Cohen's project investigated the human art making process. Fundamental to his approach is the idea of mimicking how humans make art. He did not explicitly focus on outcomes, but process. His concern was not merely the transformation of images, but their generation

Figure 2.3 "Harold Cohen with a painting machine at the Computer Museum in Boston in 1995." 1995.

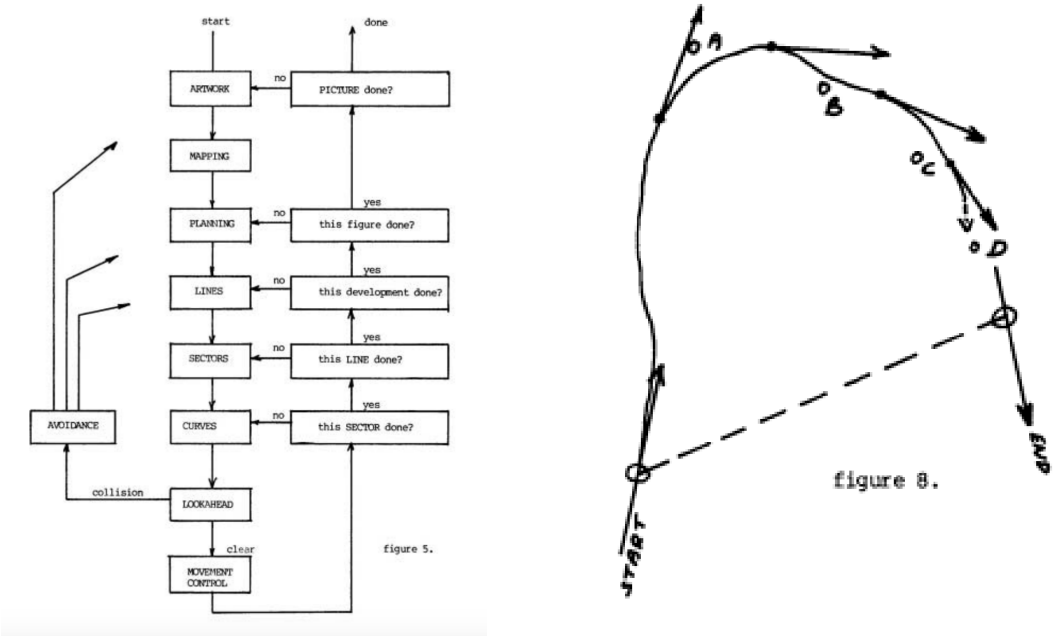
(McCorduck, 1991). Cohen was challenging what he saw as a serious deficit in computer art of the time, which was that the systems generally functioned as "picture-processors." He drew a distinction between AARON and the simple feedback in programs like Conway's "Game of Life." (Cohen, 1979)

In AARON, there are feedback loops at every level from the composition of the artwork down to the movement control of the drawing robot. According to Cohen, an early version of the main program had three hundred "micro-productions" that each handled an "action-atom" (Cohen, 1979). While each function might be understandable, it was not possible to predict the outcome of these feedback loops.

Reconstruction of the Hand-Drawn Aesthetic of AARON

To investigate Cohen's approach of using layers of feedback, I decided to reconstruct the bottom levels of AARON to see if I could mimic the hand-drawn aesthetic of the original. Source code is not available, so I used a close reading of Cohen's descriptions of the system to

recreate it. Cohen states that drawing even a single line involved twenty or thirty productions spanning at least three levels. At the most basic level, AARON navigated by controlling the speeds of two wheels, using their ratio to draw arcs of different sizes (Cohen, 1979). It used a model based on freehand drawing and did no pre-planning of any kind (McCorduck, 1991). In addition, Cohen added an element of randomness to imitate "arthritic joints" (Cohen, 1976). I incorporated these details and others to rebuild portions of AARON's architecture corresponding to the movement control, lookahead, sectors, lines, and planning portions of the architecture using p5.js and JavaScript. I also added a basic composition function to make the program draw multiple shapes.



(a) (b)

Figure 2.4 The hierarchy of feedback in the architecture of AARON and a diagram showing the line drawing algorithm in AARON (Cohen, 1979).

Upon running the program, the resulting shapes from the reconstruction loosely approximate the hand-drawn aesthetic of the shape outlines in AARON. They lack the in-fills and shadows, but the quality of line is similar. The complex feedback between a hierarchy of

layers yields forms that are unpredictable, but with a subtle, logical structure. This is similar to the stochastic, but orderly, forms found in nature. I believe Cohen's focus on the process of drawing the line rather than the final goal, along with small amounts of randomness, is the source of the organic feel.

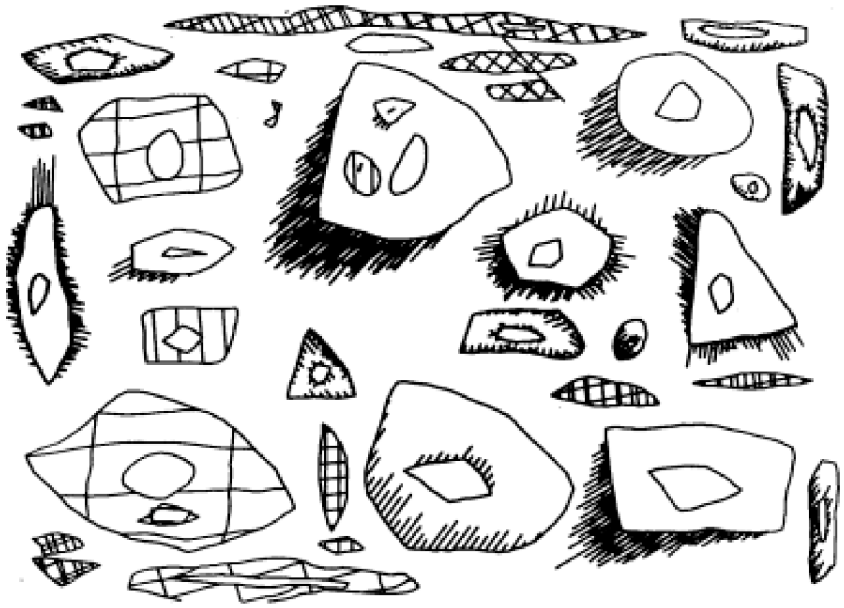
A surprising finding while constructing this was the strong effect of the temporal aspect on the impression of the system. The reconstruction adjusts its heading to correct its course to the signposts along its way. It also uses a "lookahead" function to avoid obstacles. Seeing a program make decisions in real-time allows humans to relate, in some small way, to the machine's "thinking" process. This causes the impression of intelligence within the system.

Cohen's approach of using an expert system and focusing on cognitive behavior seems to run counter to much of the machine learning art we see today. Machine learning art is data-driven. These systems are based on existing examples of artwork rather than the human process used to create them. By being limited to a set of results, it appears that data-driven machine learning art has more in common with the "picture-processors" Cohen was trying to avoid than AARON.

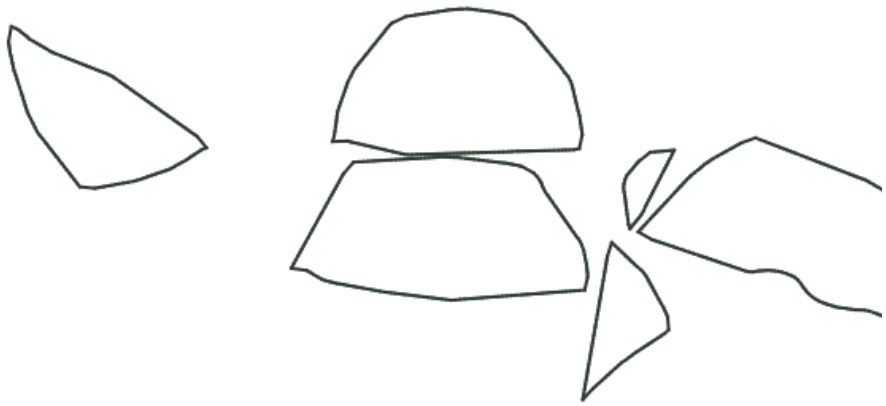
Most visual art made with machine learning relies on the use of a CNN to encode images. The rest of the background will examine CNNs, their use in art, and the potential of hand-crafting them as a way to avoid falling into the "picture-processor" trap.

2.2 Convolution Neural Networks

This project focuses on how CNNs can be used to encode visual concepts for art making. CNNs have their origin in the visual nervous system of mammals and have been around in some form or another since Fukushima's neocognitron. In this seminal work, the author created a neural network using convolution to recognize patterns independently of their position. He also organized the layers in pairs to model the simple and complex cells of our visual system that first detect patterns and then combine them to add invariance. (Fukushima,



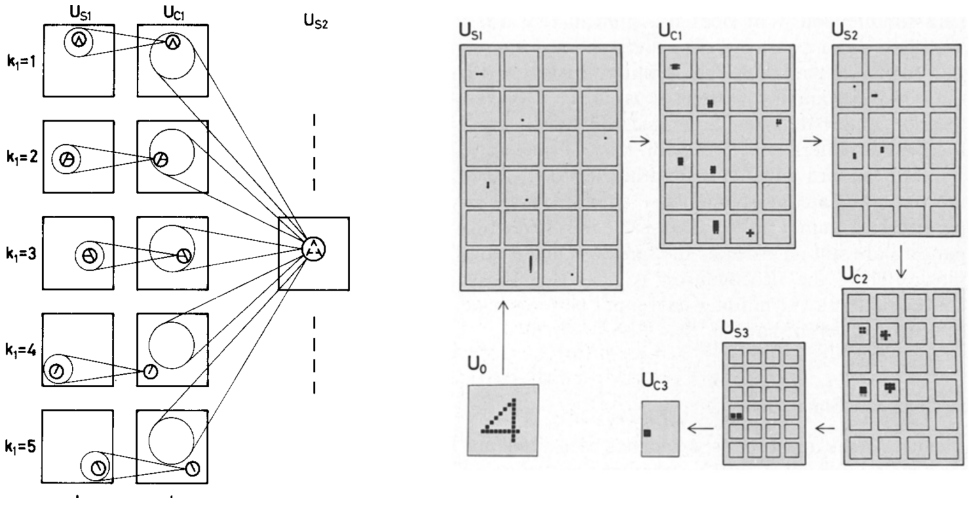
(a)



(b)

Figure 2.5 (a) Early shapes produced by AARON (Cohen, 1979). (b) The output of the reconstruction built by the author.

1980) After the neocognitron, algorithmic improvements and computing power slowly built up. In 2012, a breakthrough performance on ImageNet, the widely used benchmark for classification accuracy on natural images, propelled CNNs into widespread use (Hadji and Wildes, 2018).



(a) (b)

Figure 2.6 (a) The image on the left shows how the neocognitron contained pairs of layers to select for structures and then invariance. (b) The diagram on the right shows the activation outputs as the neocognitron selects for higher level features to recognize the number "4." (Fukushima, 1980)

CNNs are generally useful for translating an image from pixel values to higher order features. They do this through a process called convolution in which kernels are applied as a sliding window across an image. After these kernels are convolved, each pixel location is, in effect, given a more meaningful name such as "vertical line" or "corner facing down and right." This ability to translate from raw pixels to higher order constructs makes CNNs useful as a crude form of perception for art-making programs. Before describing more about how CNNs function as a form of representation, I will next discuss how they have been used to make art.

(Note: A "kernel" is a 2D matrix of weights. A "filter" is a stack of kernels, with one

kernel for each input channel.)

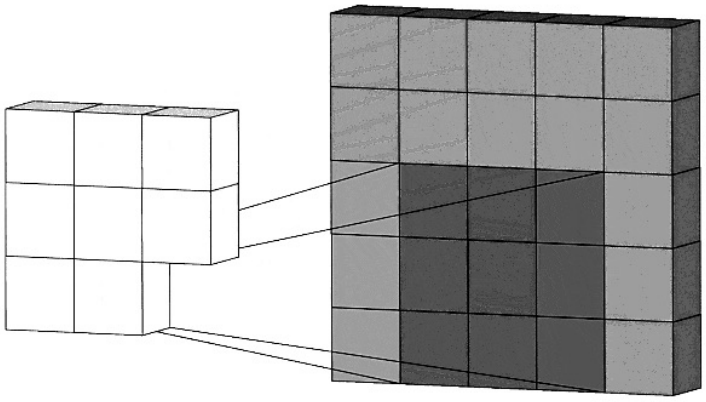


Figure 2.7 The process of convolution (Shafkat, 2018).

2.3 Image-Making with Convolutional Neural Networks

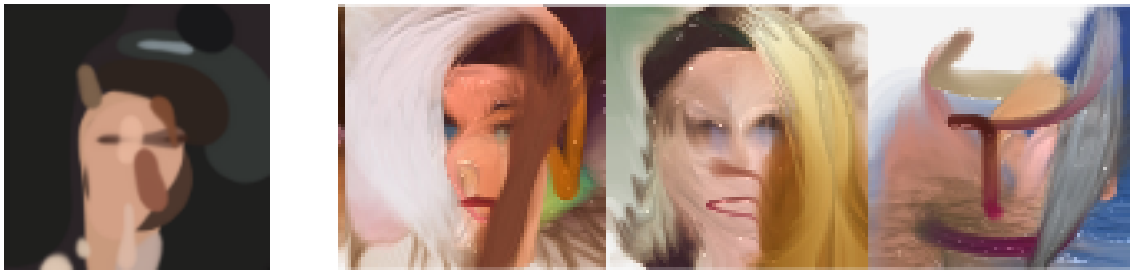
CNNs can be used in art making in a variety of ways. The most common approaches are to use them as a reward function for a generative algorithm or to integrate them into a co-creative drawing canvas for feature detection or image synthesis. One author describes these paradigms as "Heroic AI" (the AI does all the work) and "Collaborative AI" (the AI "supports, challenges, and provokes" human creativity) (d'Inverno and McCormack, 2015). Not all examples fall neatly into one of these two categories, but it helps to clarify the field.

Heroic AI

Generative adversarial networks (GANs) are a popular method for generating art with CNNs. The adversarial setup of competing networks was first proposed by Ian Goodfellow in 2014 (Goodfellow et al., 2014). In this arrangement, a generative CNN is trained as an adversary against a discriminative CNN that has been trained on a dataset of images. The generative CNN learns to produce fake images that trick the discriminator (thus using it somewhat like a reward function) while the discriminator simultaneously learns to distinguish between fake

and real images. Eventually, the generative CNN can become capable of producing images that are similar to the original dataset, but different. GANs make no assumptions about the underlying structures in the dataset. The network infers structure from pixel patterns. While impressive examples exist, there is little control over the structural relationships that GANs identify within the visual concepts. The outputs can look like a melted version of the original. Many artists leverage this surreal effect to extract interesting textures or serendipitous results.

Other approaches have leveraged the GAN setup, but swapped out the generative CNN for a reinforcement learning agent that paints through a stroke-based structure (Huang, Heng, and Zhou, 2019; Mellor et al., 2019). In a GAN, the generative CNN learns to make a pixel image in a one-shot process. By using strokes, systems can engage in a step-by-step feedback loop that uses a CNN as a form of perception.



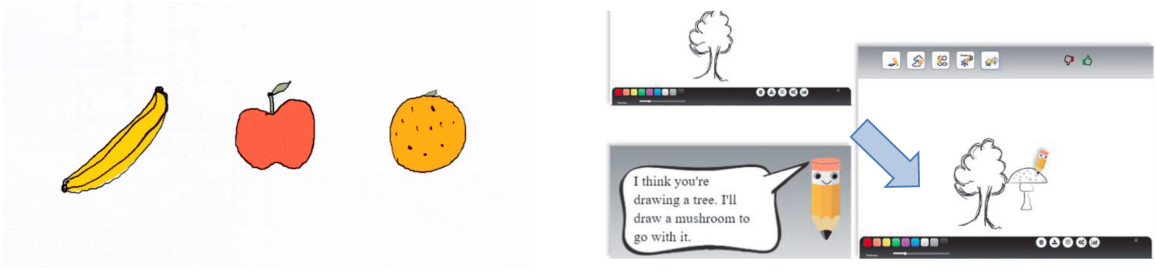
(a) (b)

Figure 2.8 (a) "Learning to Paint" (Huang, Heng, and Zhou, 2019). (b) SPIRAL (Mellor et al., 2019).

Collaborative AI

CNNs have also been used in art making by integrating them into digital canvases. A recent example of this is a project entitled Suggestive Drawing which integrates multiple AI bots into a canvas for feature detection and image synthesis. The author uses a CNN to label sketched objects and to fill those objects with appropriate or interesting textures (Alonso,

2017). Similarly, the Sketch Apprentice project uses a CNN to recognize human drawing input and respond fittingly. As an example, the Sketch Apprentice can recognize a user-drawn tree and decide to draw a mushroom next to it (Davis et al., 2016). Both the Sketch Apprentice and Suggestive Drawing projects leverage the ability of CNNs to dynamically respond to messy input and to serve as an artificial co-creator.



(a) (b)

Figure 2.9 (a) The Suggestive Drawing agent correctly identifies the objects and fills them with an appropriate color (Alonso, 2017). (b) The co-creative agent knows that mushrooms usually live next to trees (Davis et al., 2016).

Another important approach to consider when working with machine learning and line drawings is recurrent neural networks (RNNs). RNNs are well matched to sequential data. Line drawings are created through a series of strokes, which RNNs can encode more efficiently than pixel-based CNNs. Projects like Sketch-RNN have leveraged sequential stroke data to impressive effect (Ha and Eck, 2017). While RNNs can readily perceive the steps involved in creating a visual concept, they lack the ability to perceive an overall visual impression of an image, as does a CNN.

While the creation of art has a strong sequential aspect, the resulting image must also be able to stand on its own as a visual impression. This would seem to require the involvement of something more like a CNN than an RNN. There has been some research into combining the strengths of the vector-based interpretation of RNNs with the pixel-based interpretation of CNNs, but it is not well examined and is beyond the scope of the tools presented here (Li

et al., 2018). This project limits itself to investigating CNNs as a form of representation.

Critical Art Inquiries

A number of artists have built generative systems that critically interrogate the visual information encoded within CNNs. Tom White created a stroke-based, evolutionary drawing algorithm to generate images using popular classifiers (White, 2018). In his project, Perception Engines, the algorithm continually adjusts images to better match the visual concept contained within the CNNs. His work highlights the potential of CNNs to act as a form of perception and to drive a creative process. Mario Klingemann probed the nature of representation within CNNs by manually altering the internal weights of a GAN (Klingemann, 2018). His results retain some of the appearance of the original output, but with haunting effects. Klingemann's work reveals the delicate balance of weights within the network and shows how even small changes can have a dramatic impact on the learned structures.

Figure 2.10 *Perception Engines* by Tom White (2018).

In generla, CNNs as a form of representation seem to have a connection to themes from Surrealist art. Artist Philipp Schmitt relates machine learning art to *frottage*, a technique developed by Surrealist Max Ernst that involves rubbing a drawing tool on a piece of paper pressed against an object to produce a texture. Frottage results in objective forms by leveraging a reality outside of the subjective control of the artist (Schmitt, 2018; Ernst et al., 1969). In addition, the architectural similarities between CNNs and human perception suggest the importance of Gestalt theory as a lens of interpretation for CNN-based art. Gestalt theory is the formal study of the phenomena of our visual impression switching

between individual objects, background, and texture. When humans experience gestalt, their impression of a whole image becomes more than the sum of its parts. Likewise, CNNs do not simply calculate a sum of pixel values. These networks find meaning through complex connections between parts.

The next section will touch on current research into the nature and interpretation of these connections.

2.4 Representing Visual Concepts Within Convolutional Neural Networks

CNNs are able to robustly encode visual concepts, but the process of how this happens is not well understood. The prevailing wisdom is that each layer of a network contains a higher order abstraction than the previous layer (Olah, Mordvintsev, and Schubert, 2017). This may or may not be the case, but it is at least a useful starting point for thinking about how networks learn (Greff, Srivastava, and Schmidhuber, 2017). In the first layer, filters function as templates. At this point, they are easily understood because they visually correspond to the pattern that activates them. The filters can have some tolerance, but they match a narrow range of pixel arrangements. After multiple layers, the filters no longer correspond to templates. The filters match different scales, rotations, and individual expressions of an object and thus defy explanation through a single image. Various visualization techniques have yielded some insight into what they encode, but they are still somewhat of a mystery (Olah, Mordvintsev, and Schubert, 2017; Olah et al., 2020b).

One approach to deciphering visual concepts within CNNs is using K-means clustering. In this technique, images are clustered by their activation patterns to see if images near each other in activation space are semantically related. Wang et. al used this technique on a set of natural images and found that the interior layers of CNNs learn semantically related

parts, as well as groupings of images with no obvious correspondence. This suggests that networks can be built by dividing visual concepts into parts. It also shows that networks learn representations that defy simple human comprehension. In addition, the authors found that clusters corresponded to a "dense" description of the object. In other words, the intermediate parts learned by the network had significant overlap. Finally, the authors found that the visual concepts they discovered were mostly represented through a distribution of activations rather than the activation of a single filter. (Wang et al., 2016)

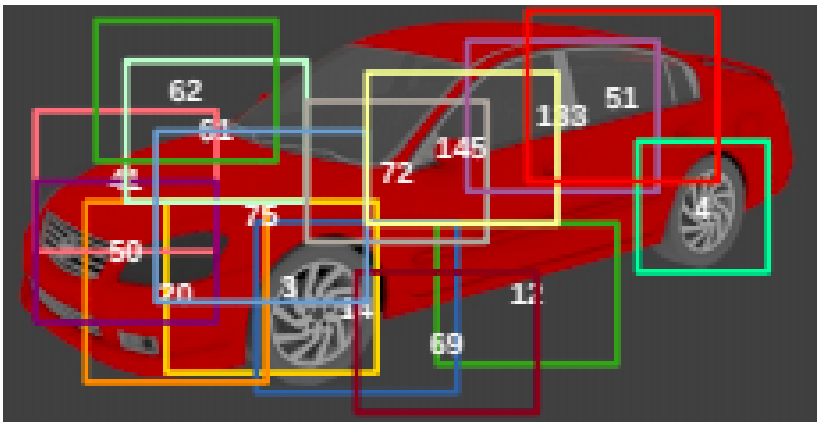


Figure 2.11 A "dense" description of the object (Wang et al., 2016).

In a simple world, each visual concept would correspond to a single filter acting as a "grandmother cell" (the theoretical neuron in our brain that gets excited when we see our grandmother). In practice, CNNs do not heavily utilize this type of cell. While the work by Wang et al. demonstrates the complexity of CNNs, it tells us that their internal weights can divide a recognition task into smaller, comprehensible visual concepts. It also shows us that these parts often overlap and that it is possible (even though machines rarely do it) to represent them with single filters.

While Wang et al. used natural images, this thesis focuses on line drawings. To see how their findings would hold up on line drawings, I repeated their K-means clustering method on Sketch-A-Net, a state-of-the-art line drawing CNN with eight convolutional layers (Yu et al., 2017). I found that the network slowly built up representations. As expected, the

first layer visual concepts corresponded to the kernels. The second layer began adding corners, curved corners, texture fields, and dots, while passing along filters matching single lines. The third layer started showing edges of objects (as opposed to lines), pipes (pairs of lines), and a greater variety of textures. The fourth and fifth layers continued to add the features beginning to emerge in the third layer, but began to define more textures. Layers six through eight all had receptive fields that spanned the entire image, so they divided the images essentially by type. Although, layers six and seven had interesting combinations of types such as scissor/planes and table/cows.

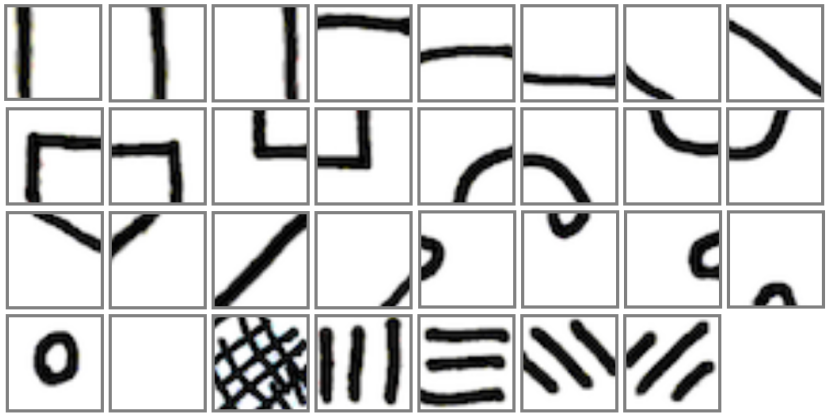


Figure 2.12 Artistic renderings of visual concepts found in Sketch-A-Net through K-means clustering by the author.

My interpretation of the basic structures within a CNN was validated in recent work by Olah et al. examining the first several layers of a CNN trained on natural images. They also found that the early layers build from Gabor filters to lines, curves, corners, textures, and pipes (which they refer to as "double boundaries") (Olah et al., 2020a).

Ultimately, there is no single image that can represent what activates a given filter within a CNN. The interactions are complex. Even if we find a particular activation pattern that maps to a set of images, we would still have trouble understanding the weights that are involved in creating that mapping. Regardless, it is important to analyze which visual concepts are learned at which layers in order to know what is possible to build.

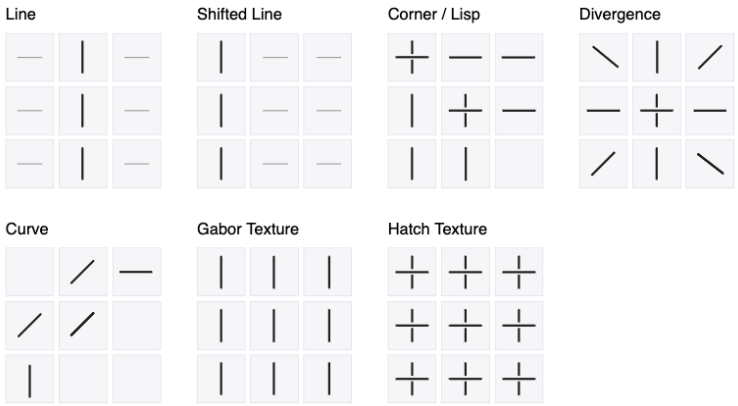


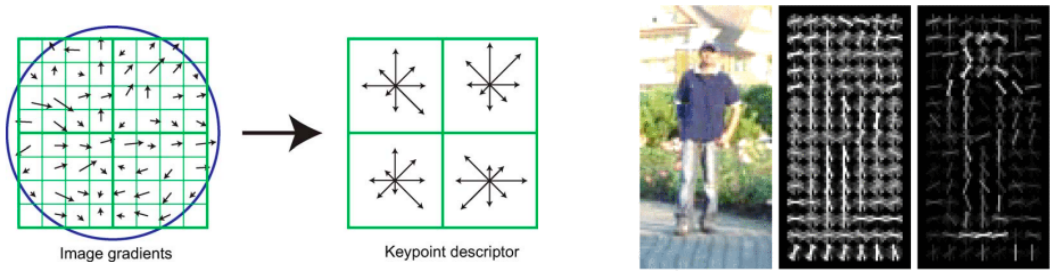
Figure 2.13 Apparent structures in the second convolutional layer of InceptionV1 (Olah et al., 2020a).

This thesis attempts to shape CNNs in human interpretable ways. Instead of focusing on what machine learning algorithms find most useful or efficient, we can use insights from how those algorithms shape networks as possible strategies to inform direct manipulation. Direct manipulation, also called hand-crafting, has serious limitations, but it can be used effectively in the first layer of CNNs and recent research has opened up the possibility of pushing that effectiveness deeper.

2.5 Hand-Crafting Convolutional Neural Networks

The term hand-crafted is widely used in the field of computer vision to describe an approach to image processing where feature detectors are designed by humans instead of automatically calculated from data. Two popular methods of hand-crafting are HOG and SIFT (Lowe, 2004; Dalal and Triggs, 2005).

A common alternative to hand-crafted features which has arisen in the past several years is to learn features through machine learning. This has led to a number of studies examining the comparative advantages of hand-crafted and learned features on standard image datasets. Some studies found that learned features outperform hand-crafted alternatives (Antipov et



(a) (b)

Figure 2.14 (a) SIFT (Scale Invariant Feature Transform) (Lowe, 2004). (b) HOG (Histogram of Gradients) (Dalal and Triggs, 2005).

al., 2015; Tiwari, 2020). Others have found contradictory results with more sophisticated hand-crafting algorithms (Schonberger et al., 2017). Even when hand-crafted features are used, they are typically used as the first layer of a neural network in combination with learned features (Majtner, Yildirim-Yayilgan, and Hardeberg, 2016; Nanni, Ghidoni, and Brahnam, 2017; Chherawala, Roy, and Cheriet, 2013). Although, hand-crafted Gabor kernels with learned parameters functioned well at multiple layers in one study (Zhang et al., 2020). Machine learning has displaced most hand-crafting of networks, but there still seems to be some benefit to using expert knowledge.

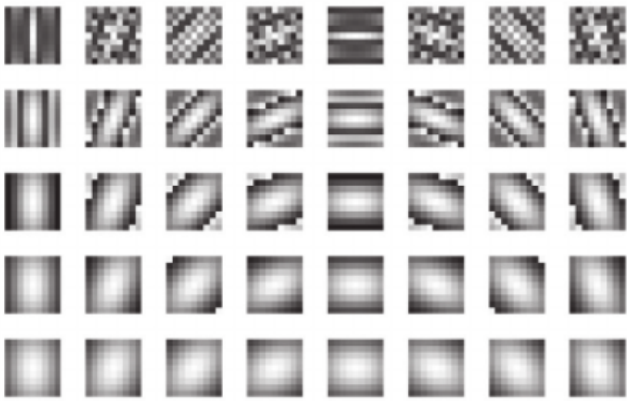


Figure 2.15 Gabor filters for a layer of a CNN designed through learned parameters (Zhang et al., 2020).

While HOG and SIFT are not specific to a certain type of image, some hand-crafted features are more "crafted" in the traditional sense. An early example of the use of the term hand-craft comes from a report on outdoor navigation in 1991. Thorpe and Kanade describe their hand-crafted stop sign detector as follows, "We use the red color as a cue to sign location, then look for color edges, then use a variety of techniques to fit the octagonal shape." (Thorpe and Kanade, 1991) This hand-crafted feature detector is specific to a certain scenario. It uses expert knowledge of a subject to make an educated hypothesis about the salient features of an image of a stop sign. The authors' close attention to material aligns with the use of the term craft outside of computer vision.

In other fields, the term craft is associated with workmanship and is used to imply a subconscious dexterity in the actions of a practitioner. Woodworker and design theorist David Pye contrasts design and workmanship as the difference between what can and cannot be "conveyed in words and by drawing" (Pye, 1968). There is a special form of expert judgement at play through the development process. Hand-craft also invokes the classical association with the hand of the worker that can both sense and act. Architect and digital designer Malcolm McCullough describes the hand as "part effector and part probe." It acts on a material while sensing important details at the same time. The hand serves as the source of embodied knowledge and skill central to craftsmanship. This is not to say that craft is restricted to workers using their hands. McCullough elaborates that the perception of craft is relative to "the degree of personal participation, more than any degree of independence from machine technology." (McCullough, 1998) Craft can be present in technology and computational workflows as long as the user is exercising personal design decisions. Computer vision practitioners who bring their expert knowledge and react to their observations meet this mark. The focus of this thesis is leveraging expert knowledge (an artist's) to hand-craft networks with respect to a specific material (a line drawing).

Chapter Three

Methods

> *You can speculate on how things work, and painters usually do. But everybody has his own ideas, painters and critics alike; everyone's in charge of his own fantasy — that means you can't pin anything down because you don't have rigorous testing procedures.*
>
> Harold Cohen

3.1 Overview

The tools I built are called the Kernel Tuner and the Network Builder. They assist with the direct manipulation of convolutional kernels and provide interactive visualizations of the effect of those kernels when applied to a particular visual material (in this case, black and white line drawings). The Kernel Tuner parametrically creates groups of kernels for detecting features on the raw pixels of the canvas. The Network Builder attempts to support the more ambitious goal of hand-crafting multiple layers of a CNN. The technical aspects and features of the Kernel Tuner and Network Builder are described in detail in the Tools chapter. Additional information on supporting code can be found in the appendix.

The tools for this thesis, much like URBAN5 and AARON, were not developed with an eye towards public release. While something of the sort could happen one day, this thesis is an experiment to flesh out whether directly manipulating CNNs is feasible or desirable.

Before doing this research, I knew that I could make reasonable kernels with the Kernel Tuner (based on its similarity to research in computer vision), but I did not know if the ability to fine tune those kernels with respect to a specific visual material was desirable for art making. On the other hand, beyond desirability, I questioned the basic feasibility of hand-crafting multiple layers of a CNN with the Network Builder.

This research contains two connected investigations in which I used myself as a case study. One covers the affordances of the tools and the second offers potential use cases.

First, I explore the affordances of the tools for hand-crafting shallow CNNs. By "affordances," I mean the ways in which the features provided by the tool transform my ability to directly interact with a CNN. This part of the research focuses on the feasibility of using these systems. I had to find out if the tools provided affordances to make the difficult task of crafting CNNs slightly less difficult.

For the Kernel Tuner, I describe the process of creating a set of kernels for detecting line ends, document the steps taken, and explain how I used the tool to inform my actions. To analyze the results, I discuss the resulting line end detector and compare the hand-crafted kernels to learned versions from a CNN for classifying line drawings.

For the Network Builder, I give an account of my construction of basic networks to detect a line at a flexible vertical position and a box of flexible size. Again, I describe the steps taken along the way and my decision making process. Then, I relate a specific example of how I used the features of the Network Builder to solve a particular problem I was having with the box network. I analyze the results and offer theoretical explanations of some of the phenomena and challenges encountered during the process.

The second investigation tests the desirability of hand-crafting CNNs by examining potential use cases. To do this, I incorporated the output of the tool into my own art making practice. I describe my initial views on the relative strengths of the networks I built and how I attempted to leverage those strengths when designing multiple art pieces. Then, I reflect on how the aesthetic outputs compare to other computational approaches for making art.

3.2 Tools

This section gives a detailed look at the tools used in this thesis: the Kernel Tuner, the Network Builder, and their supporting code.

3.2.1 Kernel Tuner

The Kernel Tuner is a parametric tool for crafting the first layer of kernels in a CNN to extract basic features from line drawings. Taking inspiration from the practice of hand-crafting in computer vision, it puts the visual material of a line drawing in direct communication with the design of convolutional kernels. Typically, we leave it to the machine to infer the weights of a CNN from training data. However, the first layer almost always ends up looking like a type of kernel called a Gabor filter (Olah et al., 2020b). If we know what we want the result to be, why not make the kernels ourselves? This is the motivating consideration behind hand-crafting. By crafting these kernels ourselves, we ensure their even distribution and match them to human-comprehensible visual concepts.

I use the Kernel Tuner to generate kernels, observe how they interact with a canvas through visualizations, and then use those observations to guide further updates to the kernels. The kernels can then be exported from the tool and inserted as fixed, pre-trained weights in larger networks.

Features

Parametric Control: Users can adjust the size, types and number of rotations of the kernels. In addition, they can choose the width of the positive activation and the radius of the 2D Gaussian function applied to the kernels.

Drawing Input: The drawing space allows making or erasing lines, as well as rotating the canvas, to check the robustness of the kernels to position and rotation.

Interactive Visualizations: To provide insight into the performance of the kernels, the

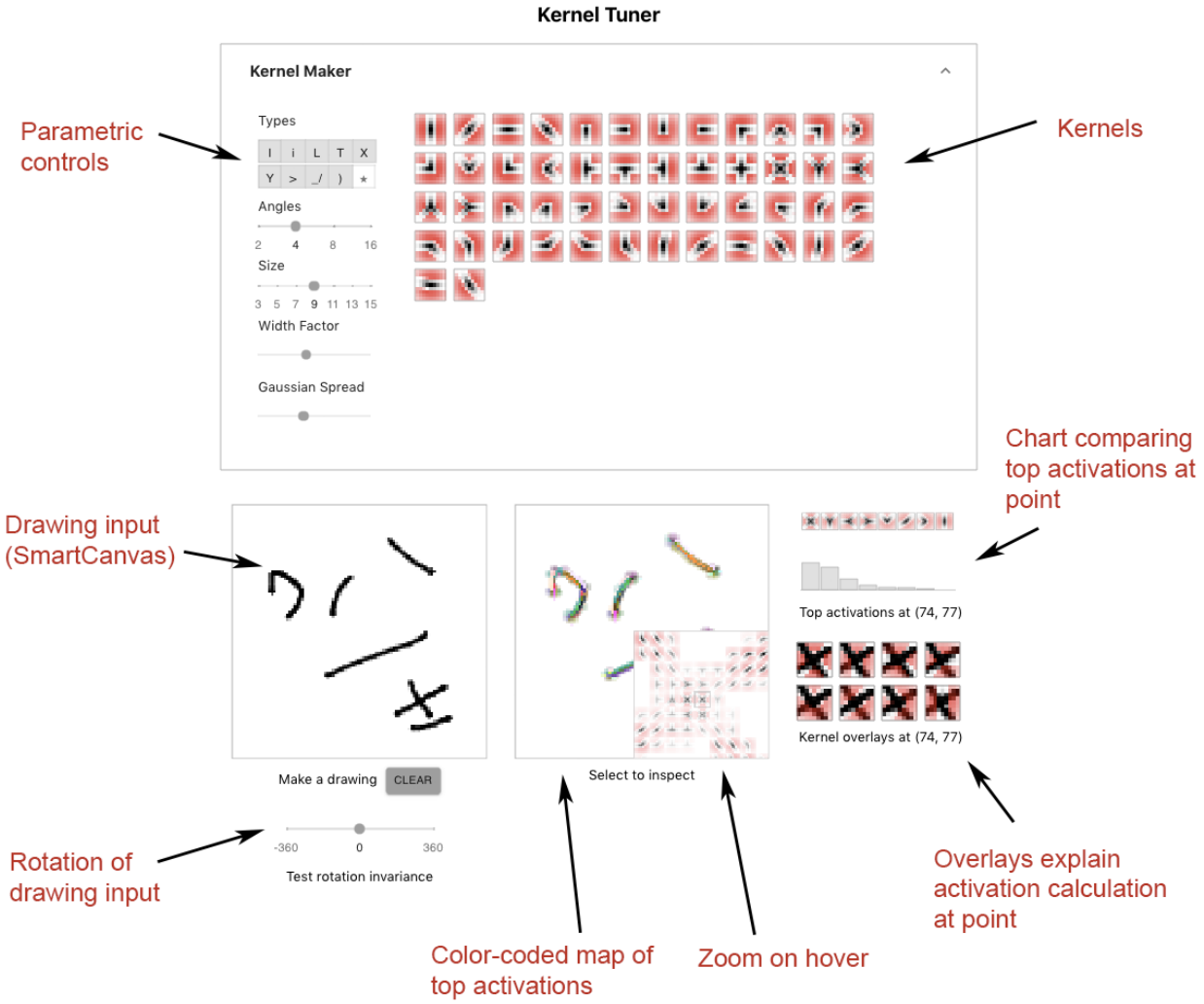
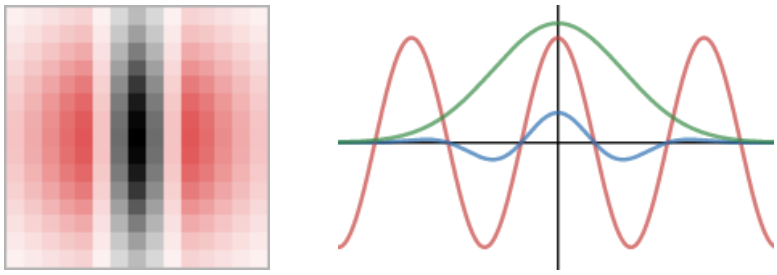


Figure 3.1 A screenshot of the Kernel Tuner.

tool includes three charts. The first chart shows a color-coded map of the top activations by location. It can zoom in on hover and allows selection to control the other two charts. The second chart is a bar graph comparing activations by kernel at a given location. The final chart shows the kernels as overlays at the selected location to allow the user to understand how the activations are being calculated.

Construction of Kernels

The Kernel Tuner uses Gabor filters as the basis of its approach. Previous research has demonstrated the effectiveness of Gabor filters as the kernels of the first layer (Yu et al., 2017). Gabor filters are kernels created using a Gaussian function applied to a sine wave (Mehrotra, Namuduri, and Ranganathan, 1992). They are useful for edge detection (or in this case, line detection) and provide similar performance to the receptive fields in our visual system (Jones and Palmer, 1987).

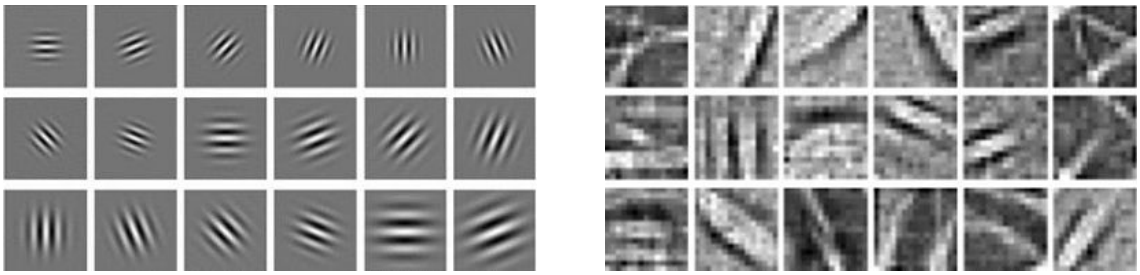


(a) (b)

Figure 3.2 (a) A vertical Gabor filter from the Kernel Tuner. (b) The construction of a 1D Gabor filter (Mehrotra, Namuduri, and Ranganathan, 1992). The filter line (blue) combines a sine wave (red) and a Gaussian/normal distribution (green).

The patterns for the kernels produced by the Kernel Tuner are not precisely Gabor filters. They differ in a number of ways. Typically, Gabor filters are applied at various sizes. This tool is designed to work with line drawings of a consistent stroke width and therefore provides a single size. In addition, this tool produces filters with three ridges (one positive flanked by two negative). Gabor filters usually have multiple ridges to match textures in addition to lines. For simplicity's sake, this project ignores textures and focuses on detecting line features.

The other important difference between the output of the Kernel Tuner and Gabor filters is the range of features matched. Instead of being limited to lines of different orientations, the kernels produced by this tool can match a range of feature types. Normally, networks



(a) (b)

Figure 3.3 (a) A bank of Gabor filters at different sizes and rotations to match edges/lines of different widths and textures. (b) Sample of kernels from the first layer of Sketch-A-Net, a state-of-the-art line drawing CNN. (Yu et al., 2017)

take their time to build up these representations (Olah et al., 2020a).

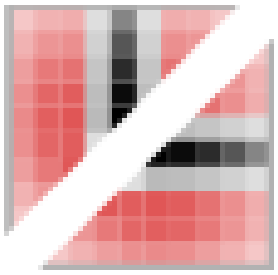


Figure 3.4 Each kernel is a Frankenstein of Gabor filters corresponding to the lines involved. The "L" corner is made by stitching together two Gabor filters at an angle.

The kernel types chosen span the range of possibilities from one, two, or three lines passing through the center of the kernel. There are usually kernels matching lines that do not pass through their center in machine-trained kernels (Yu et al., 2017; Olah et al., 2020a). I chose to exclude these offset filters to reduce the size of the network. The goal of the Kernel Tuner is to efficiently extract the most salient, "template"-like information available. The layers built on top of these kernels can further push the representations as need be.

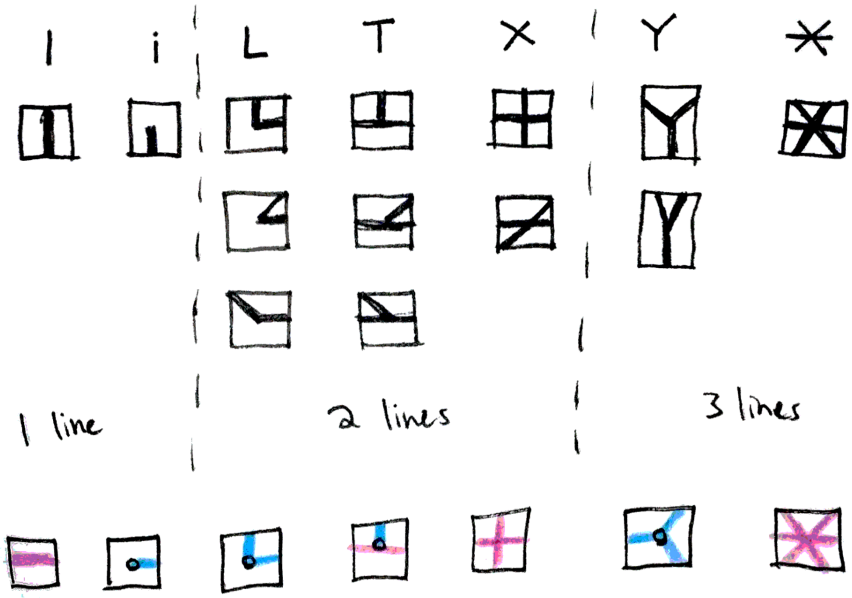


Figure 3.5 The one, two, and three line constructions considered.

3.2.2 Network Builder

The second tool is called the Network Builder and assists with building a CNN with multiple layers of convolution and pooling. Like the Kernel Tuner, the goal of the tool is to support the creation of these kernels with close attention to a particular visual material. To that end, the Network Builder provides a drawing area and a visualization of a network with respect to that drawing. Drawings can be executed by a human or through a generative algorithm. As lines are drawn or changes are made to kernels the interactive visualization of the network updates in real time.

Features

Drawing Input: The network builder uses a similar drawing input to the Kernel Tuner. However, this drawing input also includes an "AutoDraw" feature that uses a drawing algorithm to render the network.

Edit Network Shape: The first step to building a network is defining a network shape. The interface currently supports convolutional and max pooling layers. In order to simplify

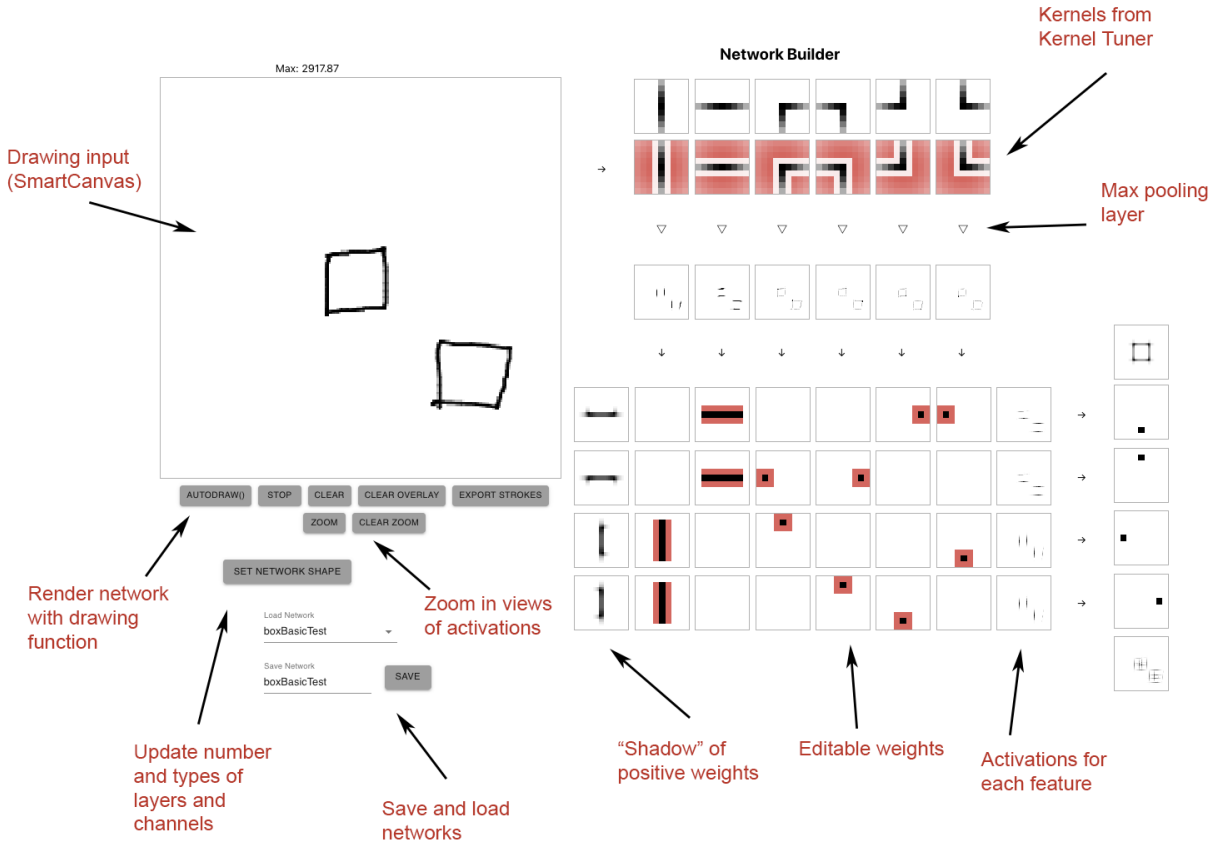


Figure 3.6 A screenshot of the Network Builder.

working with the network, I decided to follow a consistent pattern. Therefore, I set the strides on the convolutions to one, the bias to zero, and the kernel size to nine-by-nine. This size allows for enough space to pad the activations to account for flexibility, to prevent errors at the pixel boundaries and to add negative weights outside of the positive areas. Size seven or less would often not leave enough room.

Interactive "Waterfall" Visualization: The network is displayed as a series of steps down and to the right, like a waterfall. The output shape of each convolutional layer is the shape of the input for the next layer. Visualizing a CNN as a staircase shape works well when it is small. Typically, CNN visualizations focus on activations instead of kernels and are too large to display in this manner. The waterfall visualization is a novel method of

display of a CNN (as far as the author knows).

At the top of the waterfall are the kernels imported from the Kernel Tuner. Each of these kernels is actually a filter made up of one kernel. Every filter in a convolutional layer has a "shadow" of its positive weights displayed above or to the left of the filter. The current activations resulting from the drawing area are displayed down or to the right for each output channel of a layer. In terms of channels, the output shape of a convolutional layer is the input shape of the next layer. As a result, the convolutional filters rotate the output ninety degrees. Pool layers are indicated by triangles and feed directly to the next layer without rotation since they have the same input and output shape for channels.

Edit Kernels: Kernels are edited through a pop-up overlay. The overlay has convenience functions for reflection, shifting, and rotation. Many of the kernels end up as transformations of other kernels in the network and these functions make copying and pasting much more efficient.

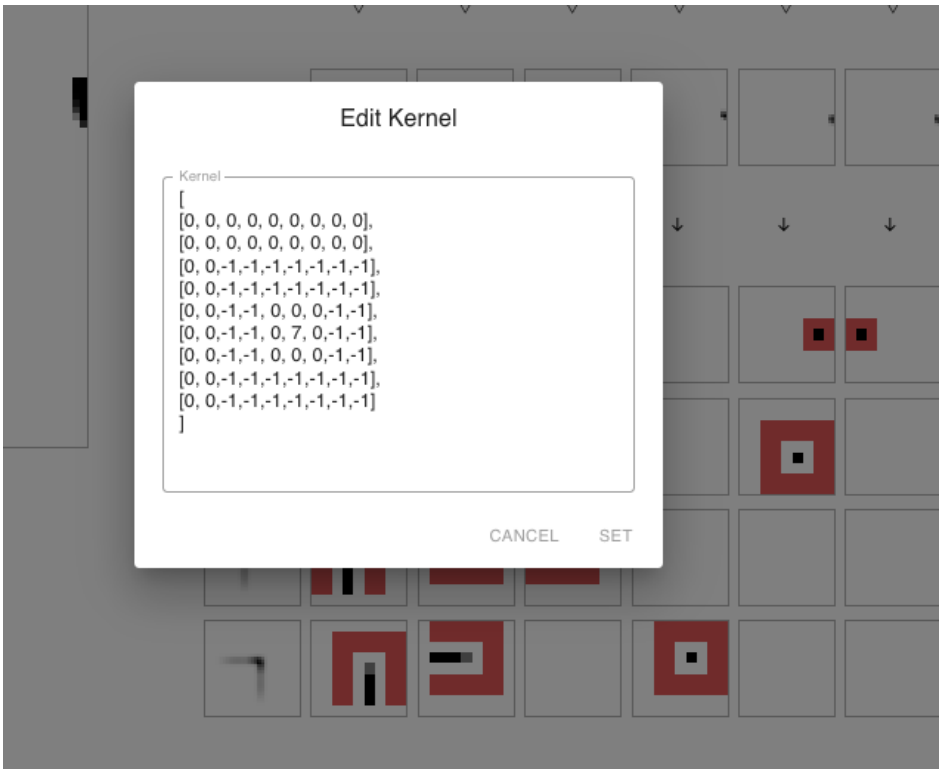


Figure 3.7 A pop-up overlay for editing a kernel.

Zoom Activations: The interface provides a zoom function that allows the user to select an area of the drawing to focus on. All the activation displays in the waterfall zoom in to that area. In addition, the user can select a channel of activation outputs in the waterfall and see a matrix of the exact values for that filter.

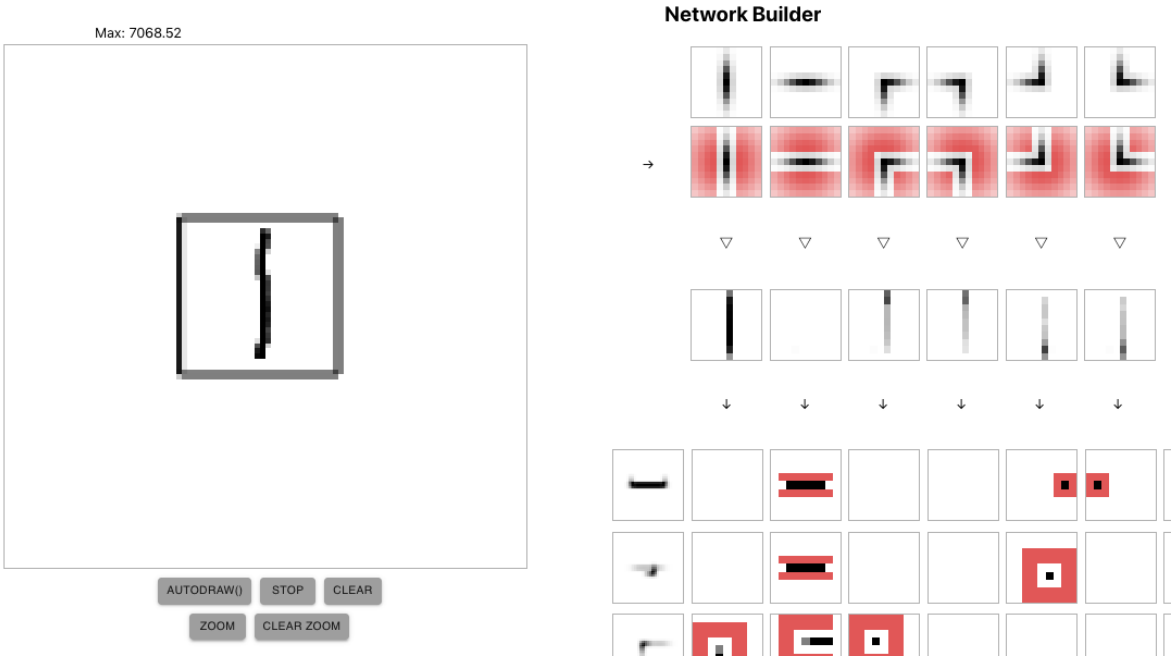


Figure 3.8 The zoom selection in the drawing area scopes all the activation outputs.

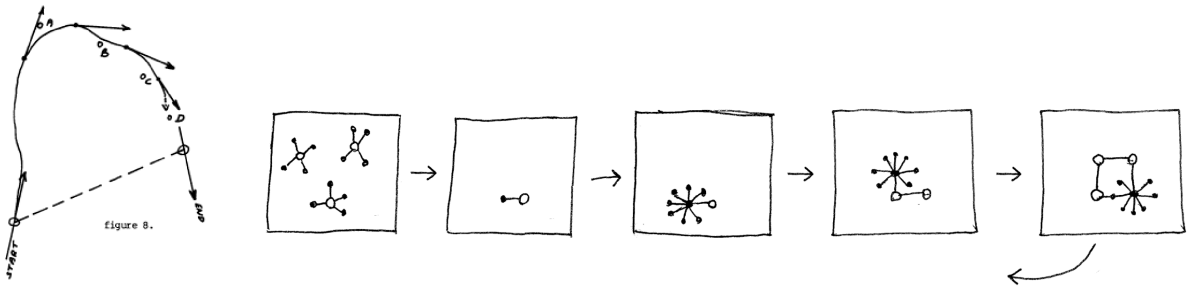
Save and Load: When the user is satisfied with a network, they can save it to the browser's localStorage for later use.

3.2.3 Supporting Code

AutoDraw Function

The most important visualization tool (and the artistic output of the system) is the Auto-Draw function. It is an agent-based line drawing algorithm that attempts to maximize a layer and filter at a given location. This function is meant to be a simple and generic way of using a CNN as a reward function. It has required tuning and adjustments to coordinate

action with the CNNs, but its core functionality has remained the same throughout. At a high level, the AutoDraw function is a brute force algorithm with two subroutines: "start new line" and "continue current line." The algorithm tests out line segments and picks the best one. It tries to continue from its current position before jumping to a new starting location. The algorithm only draws lines that improve the activation "score" and after a certain number of failures to find an option above a certain score threshold, it exits the program.



(a) (b)

Figure 3.9 (a) Inspiration for the drawing algorithm comes from AARON, by Harold Cohen (Cohen, 1979). (b) A flow chart showing the typical steps taken by the line drawing algorithm in this thesis.

The exact nature of the reward function that determines the scores for the line drawing algorithm can be altered. For testing a single visual concept, the algorithm maximizes the activation at a given location. When drawing a composition, the reward function needs to dynamically balance activations for different visual concepts at various locations.

The main idea behind the drawing algorithm is that "the vision of time is broad, but when you pass through it, time becomes a narrow door" (Herbert, 1965). At the beginning of a drawing, there are many possibilities. As the drawing is executed, the set of possible completions becomes smaller and smaller. The algorithm and network must be designed such that the algorithm does not draw itself into any dead ends. It should progress towards its goal broadly at first. Then, towards the end, it should follow the limited path that completes the line. The complications of satisfying this requirement are mainly left to the CNN.

Figure 3.10 The line drawing function is an agent-based algorithm that prefers to draw continuous lines to maximize the activation of the CNN, but can start new ones as well.

Ultimately, any algorithm that explores a design space in a suitably rich way can be paired with a discriminative CNN to produce images. I chose to use a simple, brute force line drawing algorithm because it was easy to implement and required no training. This allowed me to immediately evaluate changes to a network. Generative networks require extensive training and evolutionary algorithms can require many generations to find suitable outputs. My approach allowed me to quickly iterate on different sets of weights in the Network Builder. Although, it also made designing the discriminative CNNs more difficult.

SmartCanvas

Crucial to the performance and function of the AutoDraw algorithm is the special code backing the drawing canvas to support efficiently processing incremental updates. I call this feature the SmartCanvas. Normally, CNNs are used on fixed datasets of unrelated images. Large datasets of distinct images are fed in for training and inference. Therefore, it makes sense to recalculate all the activations for each image. In this project, I wanted to test thousands of small updates of a few pixels to the same image. When the same image is fed into the network over and over again with minor adjustments, the network has already calculated most of the activations in the system. The influence of a change to a few pixels spreads out in the network, but leaves most of the activations untouched. The SmartCanvas greatly speeds up calculating activations for small updates by caching these values. More

information on the SmartCanvas can be found in the appendix.

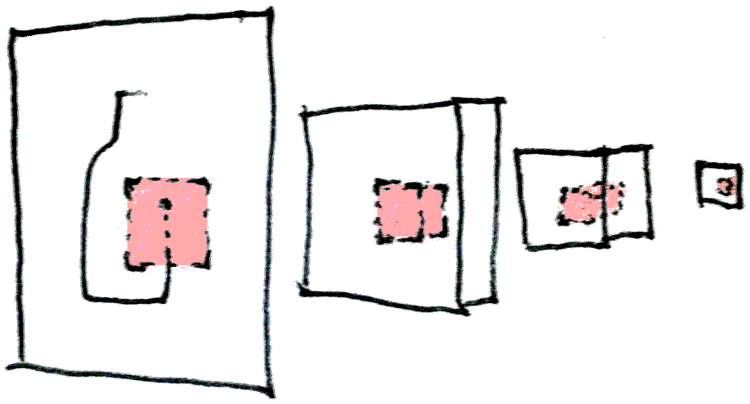


Figure 3.11 The illustration shows the mapping of affected activations through the CNN (marked in red) after a small update.

Chapter Four

Results

4.1 Affordances

This section examines the affordances of hand-crafted CNNs as enabled by the Kernel Tuner and Network Builder.

4.1.1 Using the Kernel Tuner

The Kernel Tuner parametrically produces a set of kernels that detect features related to line drawings. These convolutional kernels allow art making systems to react dynamically to input without having to track every mark made to the canvas. This section will examine the affordances provided by the Kernel Tuner with respect to a single layer of kernels through a case study describing my construction of a dynamic line end detector.

Crafting a Line End Detector

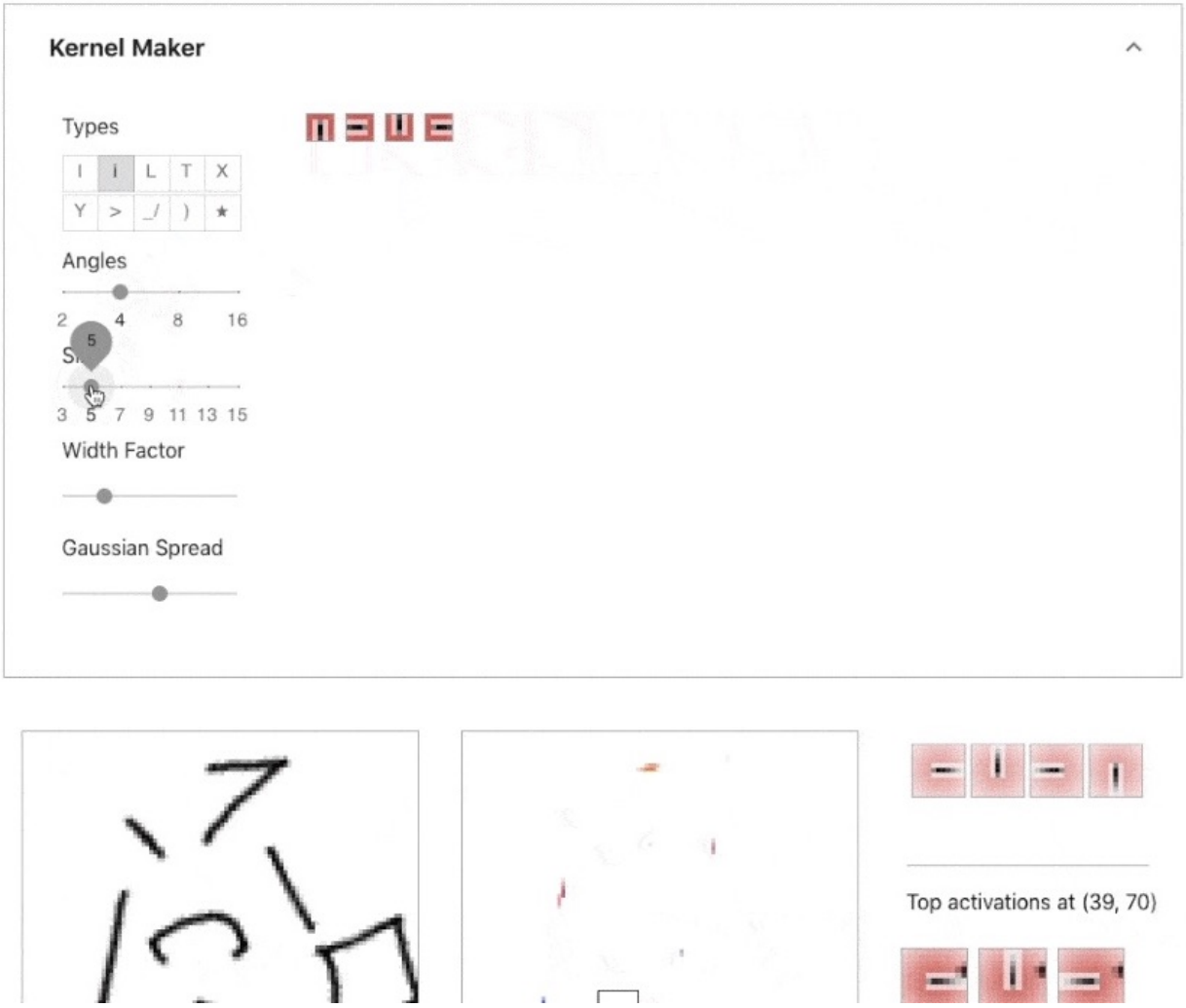
I used the Kernel Tuner to find a set of kernels for detecting line ends. The line drawing system I built prefers to draw continuous lines rather than many short, separate segments. To facilitate this, it needed to be aware of line ends when deciding where to start new lines. It could memorize every line drawn to the canvas, but sometimes the canvas might not start blank or the lines could be altered through erasure or intersection. I wanted the machine to

Figure 4.1 A line drawing with line ends detected using a CNN.

be dynamic and react to whatever was placed in front of it. I felt the drawing system should rely on its impression of the canvas as much as possible. Therefore, a line end detecting CNN seemed like the best solution.

The detector I was building did not have to be perfect (there could be some false positives or false negatives), but I wanted it to be fast. My goal when approaching the Kernel Tuner for this session was to detect the terminal end points of lines of a particular stroke weight with as few and as small of kernels as possible. Fewer and smaller kernels would minimize the calculations performed during convolution and thus speed up detection.

I started by selecting the kernel type corresponding to line ends at four rotations at size five-by-five. This produced four kernels to apply to the drawing area.



(a)

(b)

Figure 4.2 (a) Selecting the type, rotations, and size of the kernels. (b) The four kernels currently active.

With those kernels in hand, I compared them to a drawing I made composed of random marks. I rotated the drawing to determine how well the feature detectors reacted to the visual material. As seen here, the activations were hyperactive and matched all parts of the lines.

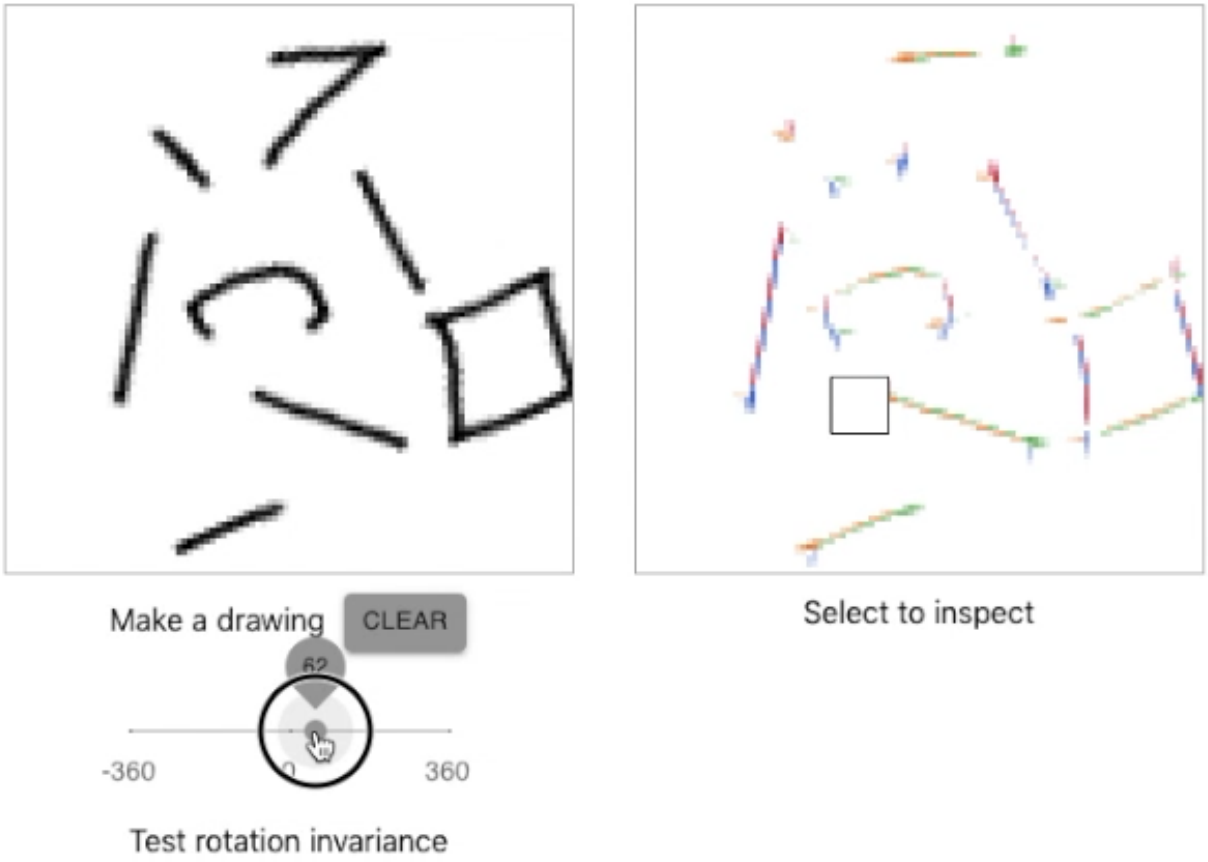


Figure 4.3 Adjusting the rotation of the input drawing. On the right, the color-coded map shows strong activity all along the lines, not just at the line ends.

I then used the color-coded chart to closely examine the top activations.

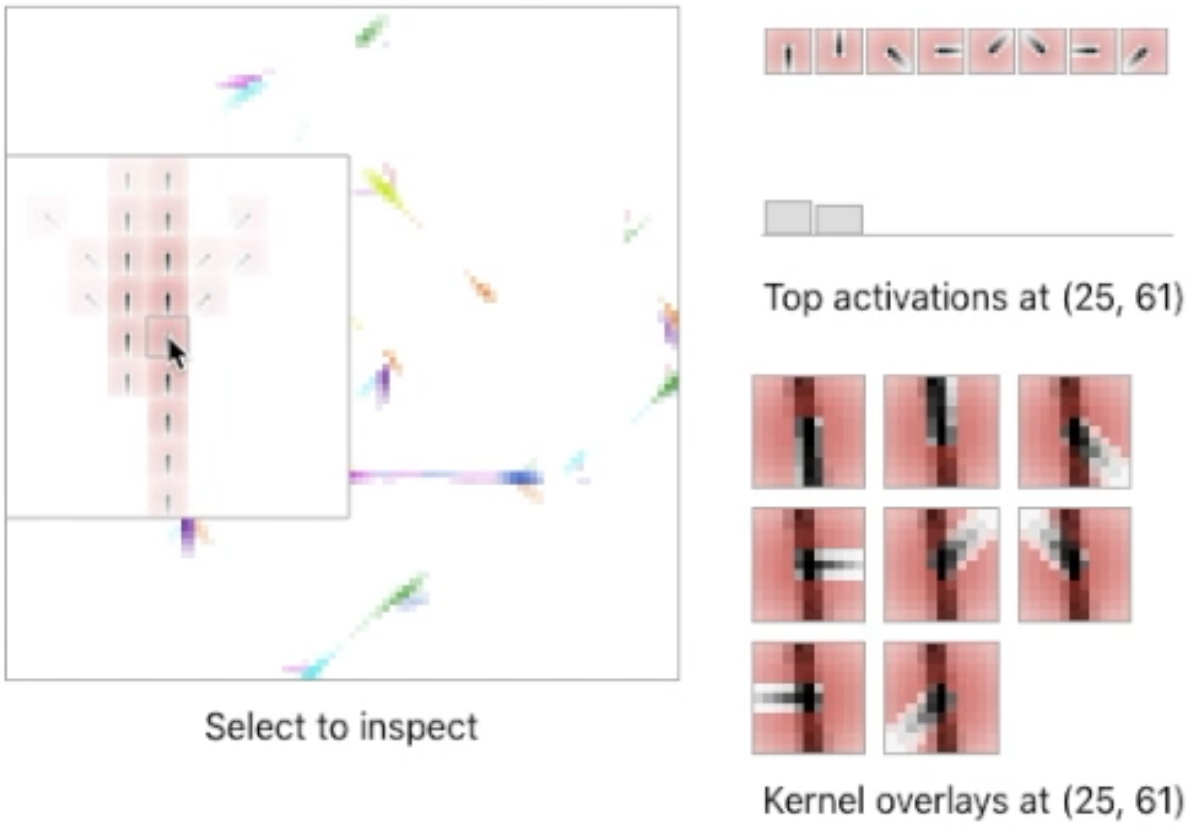
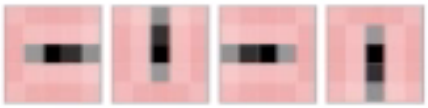


Figure 4.4 Navigating the color-coded map by hovering and selecting a point to further inspect.

I tried adjusting the width of the positive portion of the kernels and the 2D Gaussian function to tune the kernels to the line drawing. The activation charts updated in real time as I adjusted the parameters allowing me to see the results of my changes in a tight feedback loop.

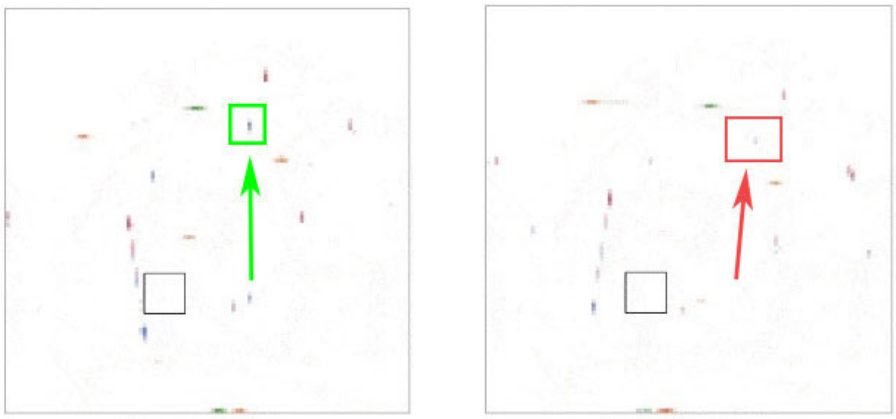
(a)



(b)

Figure 4.5 (a) Adjusting the spread of the kernels. (b) The four kernels currently active.

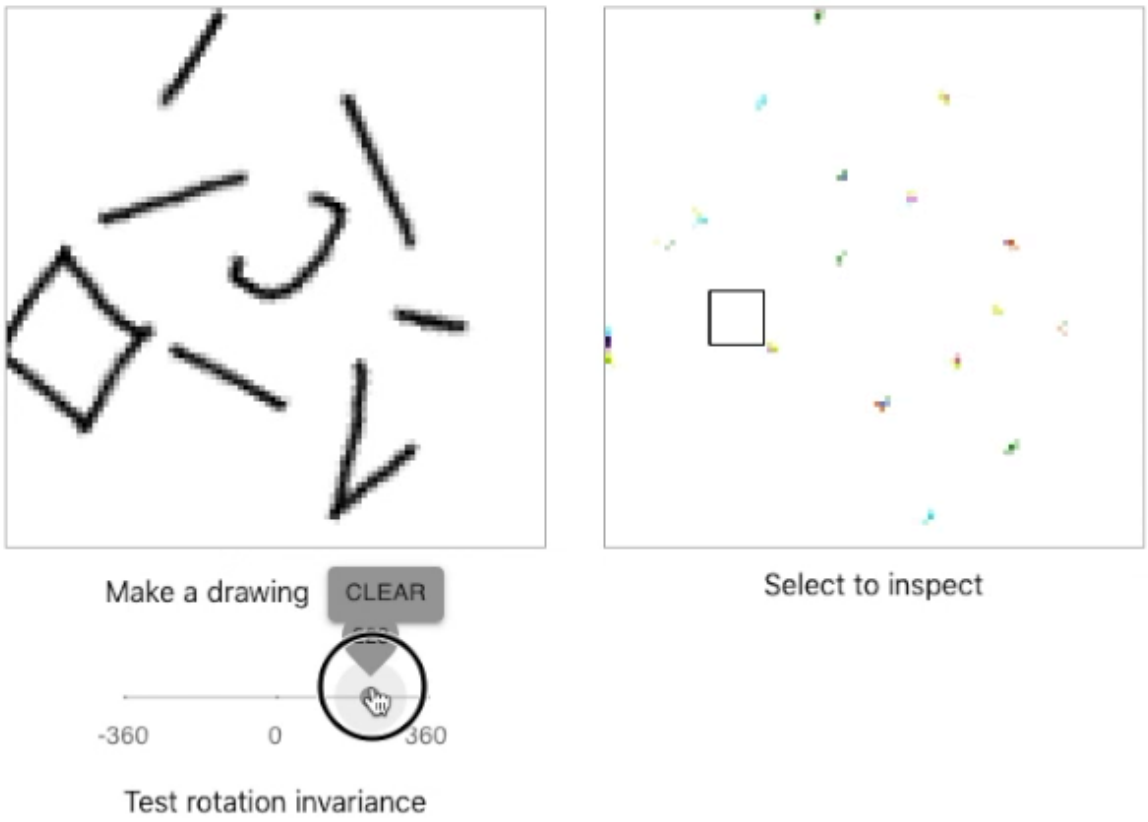
After adjusting the parameters the best I could, I was concerned that some line ends that should be showing up were disappearing at certain angles. It seemed that four rotations was not accurate enough.



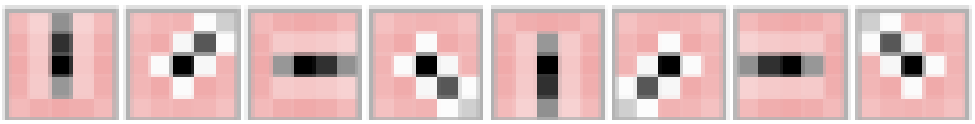
(a) (b)

Figure 4.6 (a) The color coded chart on the left shows a strong match to a line end. (b) The activation becomes faint to the point of disappearing at another rotation.

So I went back and set the tuner to provide eight rotations while keeping the kernel size at five. I also updated the width and spread parameters.



(a)



(b)

Figure 4.7 (a) Adjusting the rotation of the input drawing. (b) The eight kernels currently active.

Now, the color-coded activations seemed balanced and active at every rotation. I could use the kernels to create a line end detector. I exported the kernels and integrated them into the line drawing system.

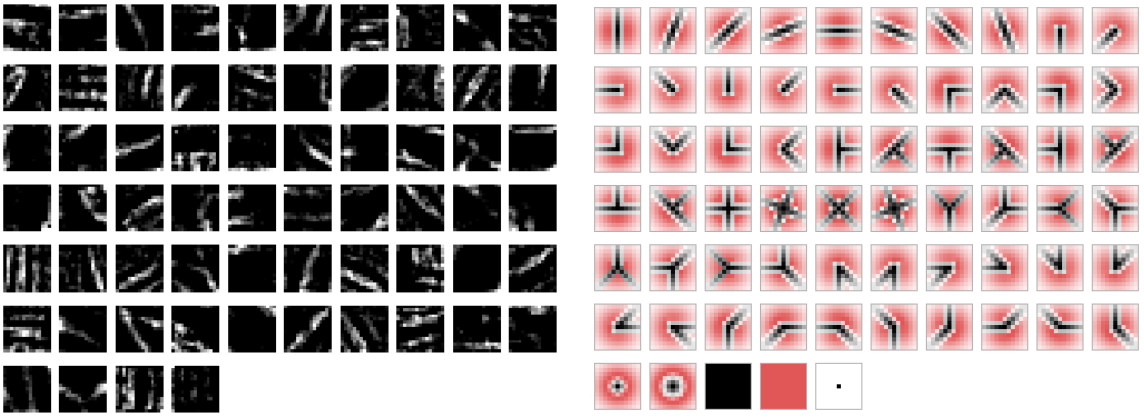
(a) (b) (c)

Figure 4.8 With the line end detector, the canvas was able to dynamically track line ends quickly and fairly accurately as I made updates (line ends highlighted in red).

4.1.2 Discussion of the Kernel Tuner

Throughout this process, the Kernel Tuner allowed me to finely adjust kernels to correspond to the visual material of the line drawing. I chose the minimum set of kernels I needed and I was able to determine my own balance of speed and accuracy through my decisions around kernel type, number of rotations, and size. It is important to note that the eight small kernels that I produced are not a one-size fits all solution for detecting line ends in images. Instead, they are a solution for my particular project that draws black lines against a white canvas at a stroke weight of 1.7px using the p5.js renderer.

The Kernel Tuner creates kernels that are well distributed and human-comprehensible. When we leave training to the machine (as seen in Figure 4.9), the filters formed are messy, offset, and incomplete. It is unclear how they are distributed or whether they are overfit to the data. The kernels produced by this tool are evenly spaced by angle and correspond to visual concepts with human-interpretable names such as "lines," "corners," and "intersections." Crafting the kernels ourselves allows us to manage over-fitting with a balanced approach.



(a) (b)

Figure 4.9 (a) The first layer kernels in Sketch-A-Net (Yu et al., 2017). (b) Kernels produced with the Kernel Tuner matching lines, line ends, corners, T-intersections, X-intersections, Y-intersections, points, open corners, dots, fields, and individual pixels.

One concern with generating the filters by hand is that we could easily miss features that are useful for classification. Patterns spread across vast troves of data can be easier to detect for computers cranking out billions of calculations per second than for a human thumbing through images one-by-one. This potential downside explains why most researchers use hand-crafting in conjunction with learned kernels or parameters. Still, hand-crafting has the advantage of restricting the search space for the machine.

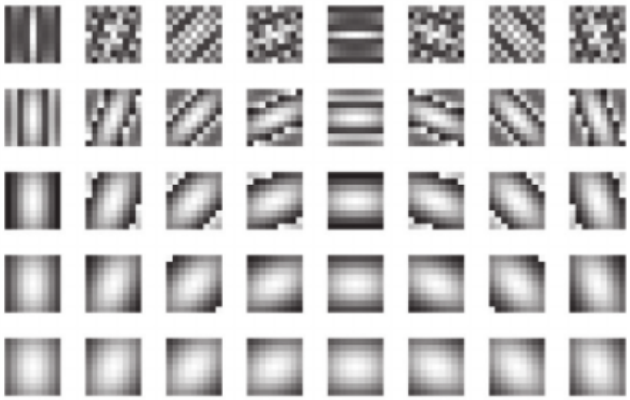


Figure 4.10 Gabor filters produced with learned parameters for the first layer of a CNN (Zhang et al., 2020).

While machine learning has its strengths, humans have difficulty evaluating the kernels produced. We tolerate this lack of interpretability of CNNs because we assume that the mystery is connected to their power. There is a commonly held belief that networks understand concepts that we cannot grasp and that a tradeoff must be made between accuracy and interpretability. This attitude may work for metric driven settings such as diagnosing cancer, but it seems inappropriate for artists seeking creative control. One must understand a material to mold it. The Kernel Tuner allows for a human to take the lead, but perhaps the ideal approach uses human guidance to refine the search space and then uses example data to balance the exact parameters.

4.1.3 Using the Network Builder

This section covers my use of the Network Builder as a probe into hand-crafting multiple layers of CNNs. The results are divided into three sections. I start with a simple example of encoding a line at a flexible vertical position. Next, I describe the process of encoding a flexibly sized box. Then, I close with a specific example of how I used the tool to guide my adjustments to a CNN.

Encoding a Flexible Line

Before discussing more complex shapes, I will start with a network for drawing a line at a flexible vertical position. If we want to have a single line of a certain length, we can start with a kernel matching a horizontal line.

(a) (b)

Figure 4.11 (a) A network consisting of only the horizontal line filter. (b) Its output with the AutoDraw function.

Then, we add a layer that extends that line. The negative margin on the first layer kernel encodes a single, connected line. As long as the negative margin is larger than the pooling size, there should be no problem. However, there is also no flexibility.

(a) (b)

Figure 4.12 (a) A network with a pooling layer and another convolutional layer that extends the horizontal line along. (b) Its output with the AutoDraw function.

The complications to this scheme arise when we start to introduce flexibility. What if we want a single line, but we want it to have a flexible vertical location? We could try vertically extending the positive activations in the kernel in the second layer. Unfortunately, the output of this network does not match our desired criteria. We may have encoded flexibility (as demonstrated by the lines appearing at multiple levels), but we now have more lines than we wanted (one).

(a) (b)

Figure 4.13 (a) A network with a flexible vertical activation in the second layer. (b) Its output with the AutoDraw function.

It turns out we cannot just add a wider receptive field. To select for a single line at a flexible location, we have to use one layer to define a long singular line and another to add vertical flexibility. The negative margin in the preceding layer prevents multiple lines from emerging in the generative drawing. Now, the output matches our desired criteria.

(a) (b)

Figure 4.14 (a) A network adding a larger negative margin to go with the larger vertical positive activation. (b) Its output with the AutoDraw function.

Encoding a Box

Next, we will move on to a shape that has multiple, overlapping constraints. Let us imagine I want to draw a box. What is a box? I might say a box is a rectangle of flexible size, formed by a single continuous line with no loose ends and square corners that are approximately in line with each other. Maybe they look something like the following...

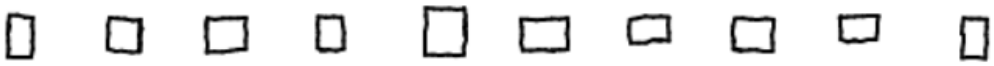


Figure 4.15 A variety of boxes of flexible size drawn with a single continuous line.

To encode this within a CNN I started with six kernels produced by the Kernel Tuner. These kernels shown below serve as the first layer. I used these, plus a pooling layer, as the foundation for all the networks in this example.

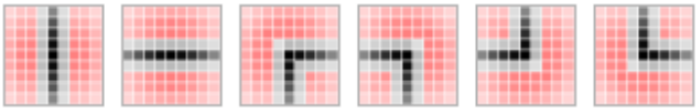


Figure 4.16 The kernels of the first layer of a box CNN.

With that base in hand, I first tried matching pairs of vertical and horizontal lines. I wanted to add size flexibility, so I tried different widths for the receptive areas.

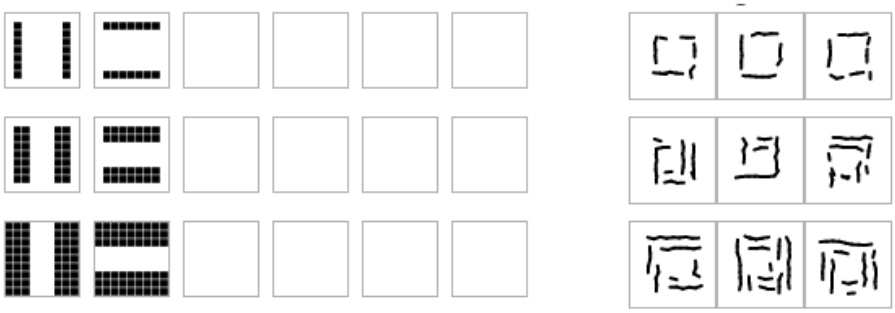


Figure 4.17 Attempts nos. 1-3 matching vertical and horizontal lines with different widths (Left) and some example output (Right).

I also wanted the shape to have a continuous line. These boxes were too broken up (just like we encountered with the preceding line example), so I added positive receptive areas for the corners at different widths.

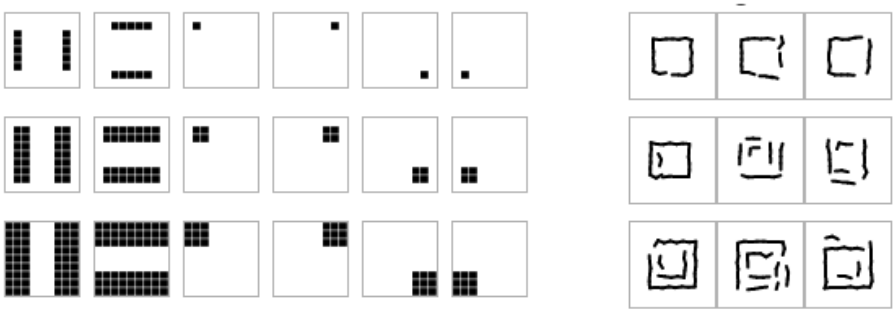


Figure 4.18 Attempts nos. 4-6 added corner receptors at different widths.

These boxes had fewer extra lines. The receptive field that is only one pixel wide is starting to look like a box, but it is not flexible.

I decided to add another layer to try to add flexibility while maintaining the single line restriction. Now, I would start with a layer that recognized edges consisting of a wall and two corners. I would have four of these filters and combine them in the next layer to make a box. The idea was to select for features at one layer and add flexibility at the next. Again, I tested different widths.

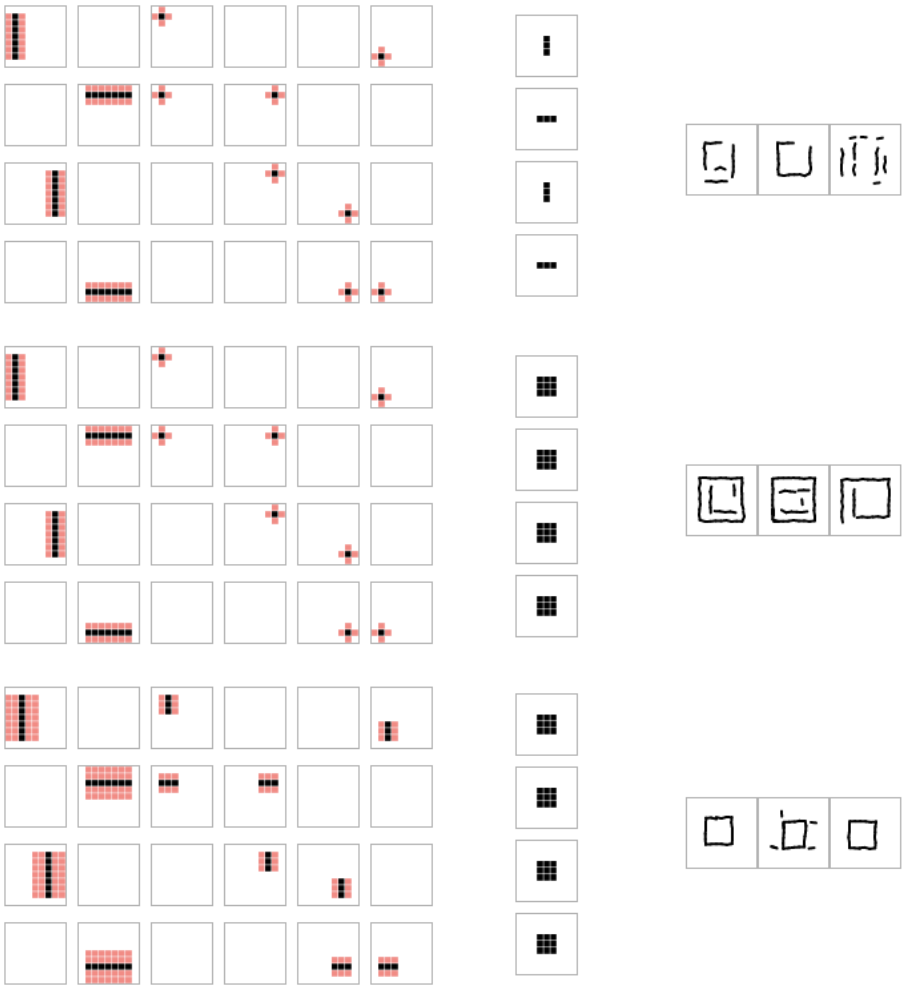


Figure 4.19 Attempts nos. 7-9 with an additional layer along with their results. The networks have different widths on the negative and positive margins.

I found that sufficient thickness at both layers yielded the best result. However, the corners still needed to be cleaned up. To do this, I added a negative backstop behind each edge for the other type of edge. The result was a flexible, clean looking box.

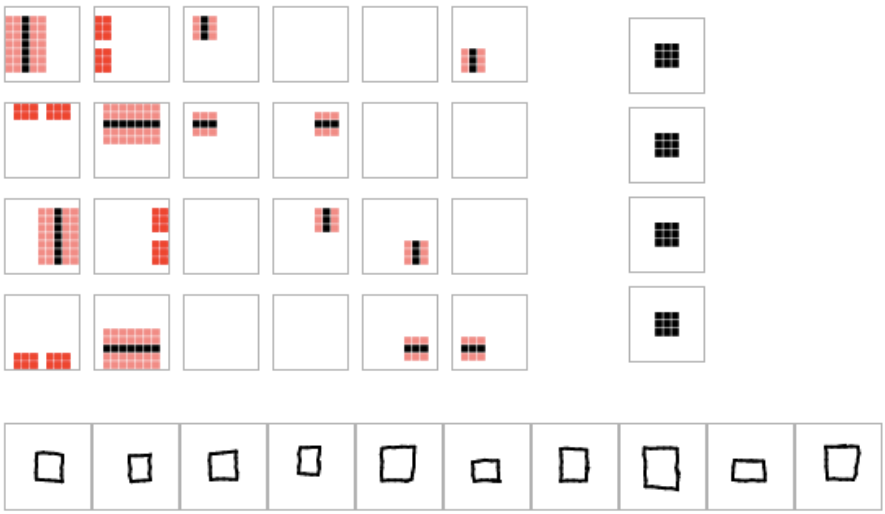


Figure 4.20 A network that yields a flexibly shaped box with a single clean line.

Problem Solving with the Network Builder

The above example showed the steps I took to encode a box. This section will illustrate how the Network Builder was used to make some of those decisions.

In this scenario, the AutoDraw function has gotten stuck at a corner while drawing a box. When I encountered this issue, I paused the drawing and used the tool to investigate. The goal was for the activation scores to go up as the corner was completed. Since the AutoDraw function was stuck, I could assume that was not the case at this point.

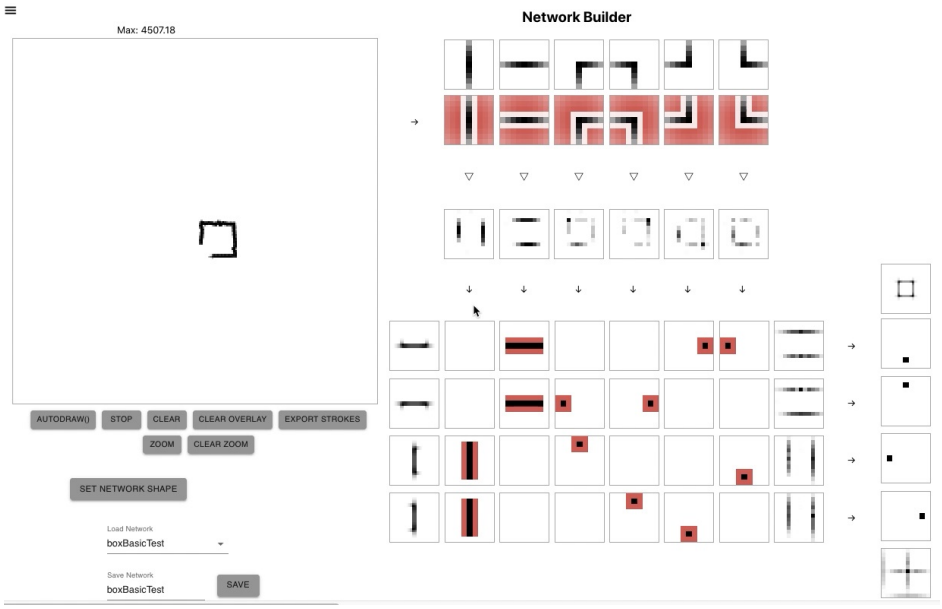
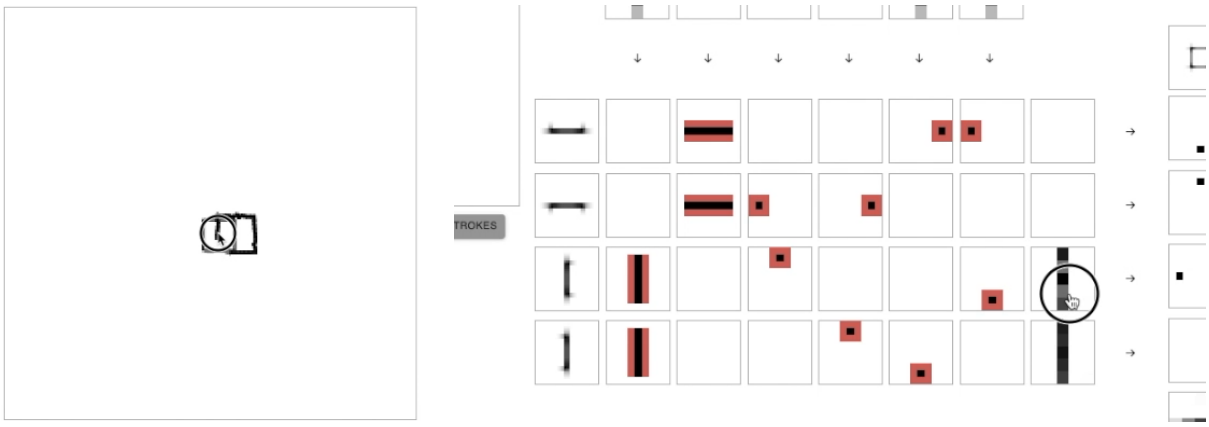


Figure 4.21 The AutoDraw function has gotten stuck while drawing a box.

To begin unpacking the issue, I first used the zoom function to scope all the activations to the area around the box. The issue appeared to be at the corner shared by the left and bottom edges. The color-coded displays of the scores are helpful for high level observations, but for close observation the actual scores are more precise. So, I selected the activation display of the left edge to see the heads up display of its activation scores.



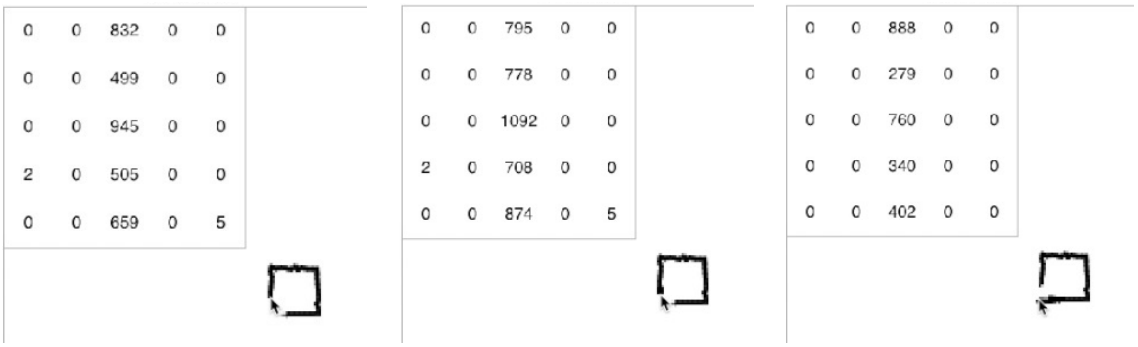
(a) (b)

0	0	832	0	0
0	0	499	0	0
0	0	945	0	0
2	0	505	0	0
0	0	659	0	5

(c)

Figure 4.22 (a) Zooming the activation scores. (b) Selecting the activations to show in the heads up display. (c) The heads up display of activation scores. We care about the '945' score at the center.

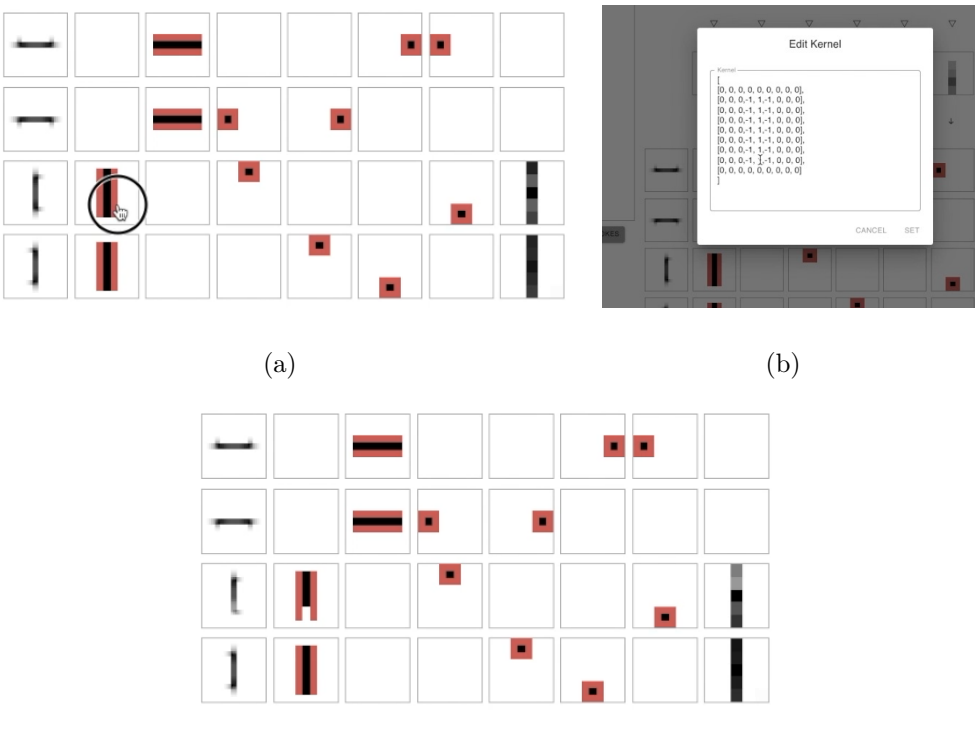
Next, I drew and erased on the canvas to observe the effect on the scores. Continuing the vertical line downwards increased the activation score, so that was not the problem. At this point, I knew it must be an issue with the horizontal line. I tried extending the bottom edge towards the incomplete corner and confirmed the score did indeed go down.



(a) (b) (c)

Figure 4.23 (a) The original state when the problem was encountered. (b) Extending the vertical line raised the activation score as desired. (c) Extending the horizontal line decreased the score.

At this point, I hypothesized that the issue was with the weights in the left edge. I thought that the vertical line detector on the left edge was conflicting with the horizontal line as it approached since the vertical line detector has a negative margin on its right side. Therefore, I removed the positive weights at the bottom of the vertical line portion to reduce its effect at the corner.

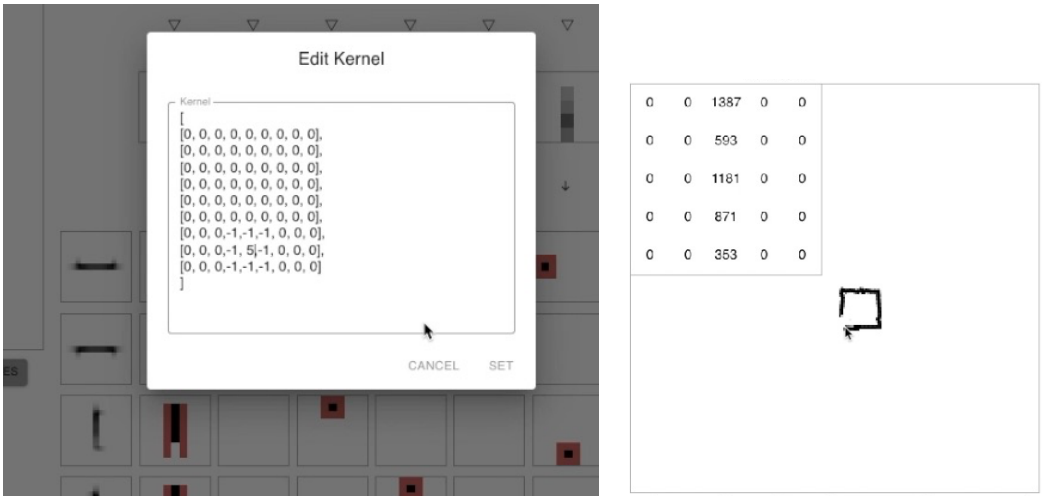


(a) (b)

(c)

Figure 4.24 (a) Selecting the kernel to edit. (b) Using the edit kernel overlay. (c) The result after editing.

After making this edit and performing another drawing test, the decrease in score persisted. My next idea was to change the bottom-left corner detector. By increasing the positive connection in the corner detector kernel, the system would know to emphasize it more.



(a) (b)

Figure 4.25 (a) Increasing the positive score of the bottom corner of the left edge from one to five. (b) After adjusting the corner, the score increased as the horizontal line was completed.

After adjusting the corner, the activation score increased as the horizontal line was completed. With this update, the expectation was that the AutoDraw function would avoid getting stuck at corners. I proceeded to update the other corner detector kernels to match.

4.1.4 Discussion of the Network Builder

Through these explorations, I demonstrated the use of the Network Builder to hand-craft shallow CNNs to encode abstract visual concepts. The CNNs acted as a form of representation that guided a generic drawing algorithm to render a variety of basic shapes.

The shapes produced were abstract. By that, I mean they maintained their key structural elements while spanning a diversity of possible shapes. The alternative would have been

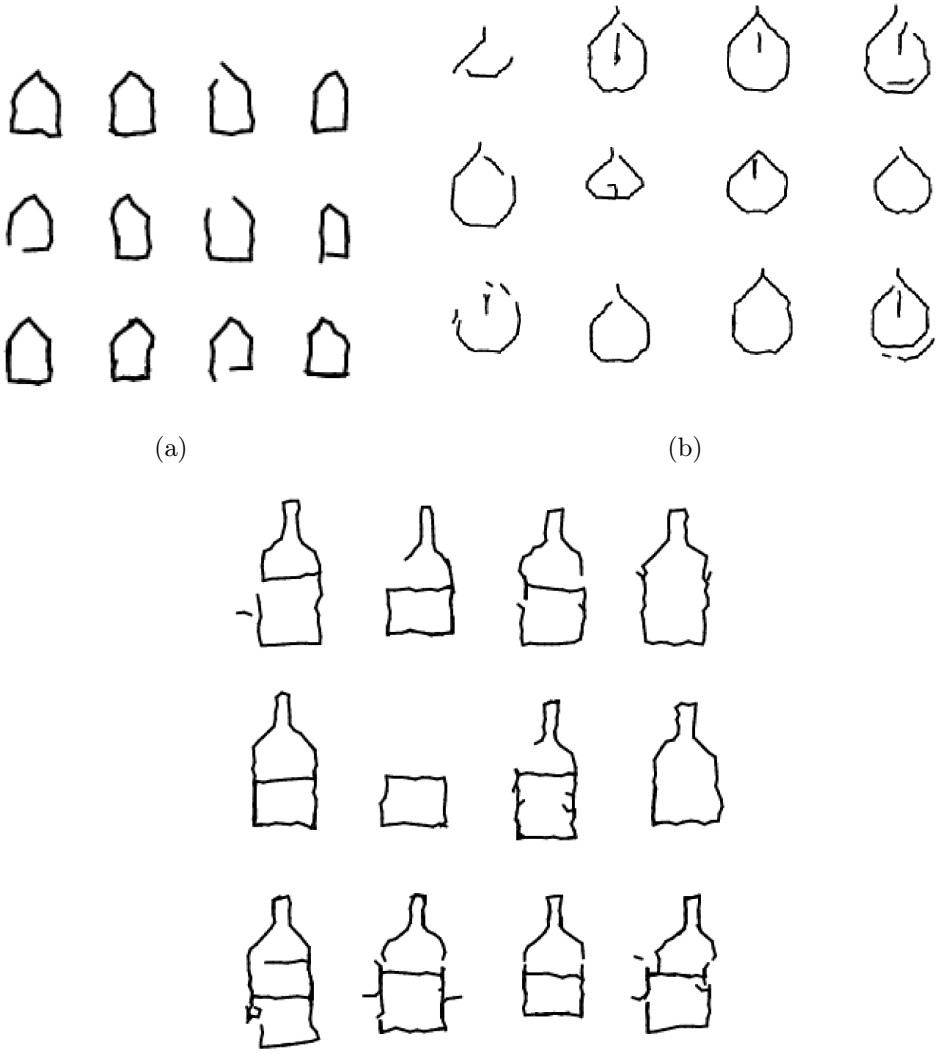
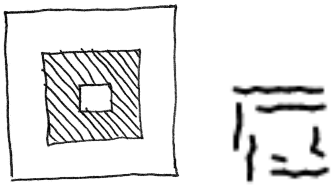


Figure 4.26 Example outputs from the AutoDraw function using hand-crafted networks encoding (a) houses, (b) leaves, and (c) bottles.

what I call fuzzy templates. Fuzzy templates are kernels that match a specific shape and can only allow for flexibility through padding (thus the fuzziness). This coarse manner of accommodating variance results in a loss of structure.

As an example of a fuzzy template, imagine a kernel with a thick square of positive activations. This fuzzy template would match a thick square, a square with squiggly lines, rectangles of all sorts, and multiple rectangles. It would map to a large design space, but

in an undiscerning way. The visual concepts encoded within a CNN have this ability to be flexible, but with greater structure. A network with multiple layers maps to designs in a way that could not be matched by a fuzzy template. Attempting to use a fuzzy template to match them would incorporate a whole host of designs that violate the desired structure of their design. For example, using the thick square to encode a box would look something like my early attempts at building networks that had unconnected lines.



(a) (b)

Figure 4.27 (a) A fuzzy template for a box. (b) A hypothetical result for the fuzzy template.

The ability to encode flexible structure emerges through multiple layers of convolution. The two rows of images below illustrate deconstructing and reassembling an image of the word "TIXY" with one and multiple layers to demonstrate the superior discernment of multiple layers. The individual pixels have about the same variance in both sets, but the ones on the bottom maintain the structures crucial to the visual concept. This happens through slowly building up flexibility layer by layer.

Using the Network Builder also demonstrated some of the challenges facing a practitioner when hand-crafting CNNs. It seems that the complexity of balancing weights within and between kernels may be impractical for anything beyond trivial concepts. Even encoding a flexible line proved challenging. A straight line has many properties that only become clear upon rigorous examination. For example, lines have a length, stroke width, connectedness, and straightness. There are many possibilities that could theoretically match a straight line. When we implement a line with rules we have to be very explicit.

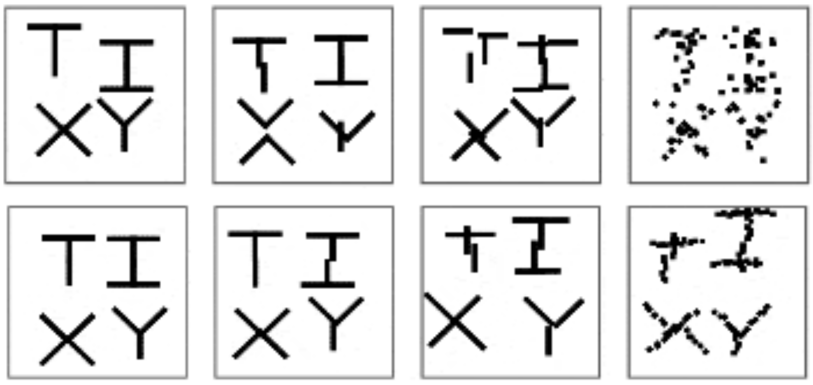


Figure 4.28 These two rows show the word "TIXY" deconstructed at different levels (by letter, letter part, line segment, and pixel) and reassembled. The top row adds variance at a single level while the bottom row adds variance at multiple layers.

Figure 4.29 Different possibilities in the design space of "a straight line."

The exploration of the box further demonstrated the delicate interplay between different kernels. The weights of the kernels, when coupled with the drawing algorithm, create different selections of a design space. A design space is many dimensional, which makes it difficult to represent. Building these networks requires a great deal of labor to find a balance. One key issue is related to the experimental setup: the line drawing algorithm requires the CNN to act as a perfect compass through the design space as it draws an object. The network has to intelligently play the role of an ever increasing (formally referred to as "monotonically increasing") reward function that guides the drawing in the right direction.

This leads to issues when partial structures create activations that compete with each other. Many shapes share parts of their structures with other shapes. A line looks like the start of a corner at first, but the corner activation evaporates when the line continues

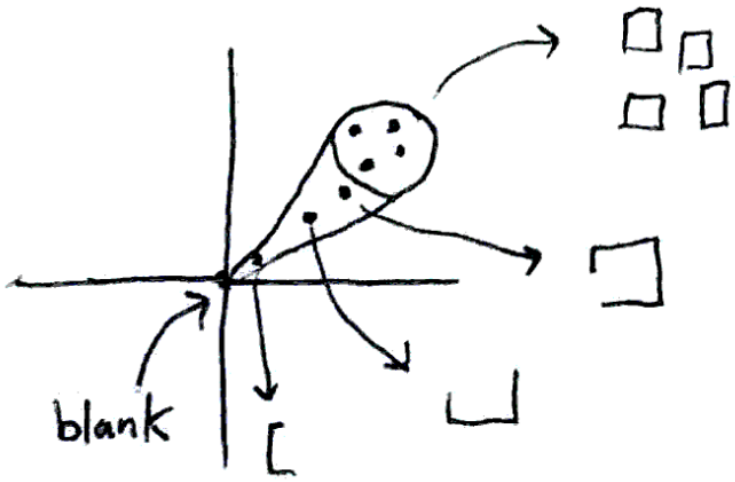
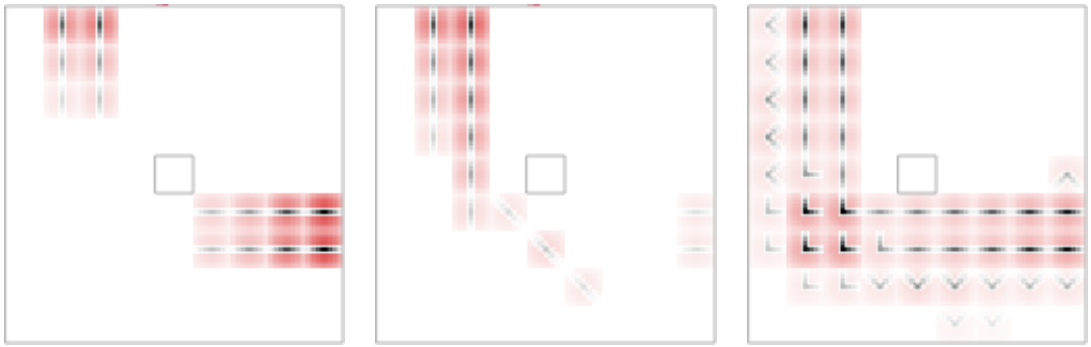


Figure 4.30 The network must remap the design space such that a blank canvas is at the origin and every segment that is added increases the activation slightly until it reaches the final goal. It also must avoid rewarding dead ends or moving past the desired design space.

straight. On the other hand, completing the corner would cause the straight line activation to disappear. This tradeoff seems desirable, but implementing this system allowed me to see that the effects are more complicated.

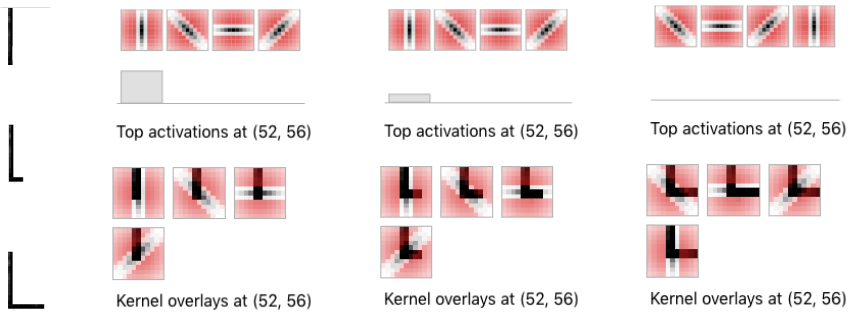


(a) (b) (c)

Figure 4.31 As the corner is made, vertical line activations are destroyed and horizontal line activations only happen strongly several pixels into the turn.

For example, we might assume that a completed corner would be a combination of ver-

tical and horizontal lines meeting at an intersection. In practice, there is a buffer at the intersection. The corner itself does not strongly activate either. In addition, it activates strongly for one as a line is drawn, but once the turn is made, that activation at the intersection disappears. This phenomena has led to constant issues with the line drawing algorithm getting stuck from negative activations while turning.



(a) (b) (c) (d)

Figure 4.32 (a) Three stages of drawing a corner. (b-d) The activations of line filters at the point where the corner happens. As the corner is drawn, the corner loses significant activation for the vertical line and gains only a microscopic amount of activation for a horizontal line.

The phenomena of competing activations that cancel each other out also raises the question of what types of kernels are needed. With only straight line kernels, there is a gap in activations at corners. If we add kernels that match a corner, we can better identify this point of interest and correctly guide the drawing algorithm. This approach could be taken further by adding offset corners and lines such that we could perfectly identify every possible situation, but this would add complexity. As it is now, there are only kernels for centered lines.

The problem of competing activations also suggests that a more sophisticated line drawing algorithm could reduce the labor involved in designing the CNN portion of the system. Other algorithms, such as reinforcement learning, incorporate delayed gratification to allow systems to discover future rewards. This helps them persevere through short term losses, but they

are also difficult and time consuming to train. Hidden steps in the reward function make the design space more difficult to navigate. The current line drawing system has the benefit of simplicity.

Overall, using the Network Builder helped to uncover some of the challenges with hand-crafting CNNs. While there were minor successes, the results indicate that crafting more complex visual concepts will likely require better tools or the incorporation of data-driven methods.

4.2 Artistic Experiments

This section explores the use of hand-crafted convolutional kernels arranged in one or more layers for art making. First, I describe the design and testing of the drawing system. Then, I relate the conceptual development of several artistic explorations. The chapter concludes with a discussion of the output in relation to other modes of art making.

4.2.1 Drawing a Shape

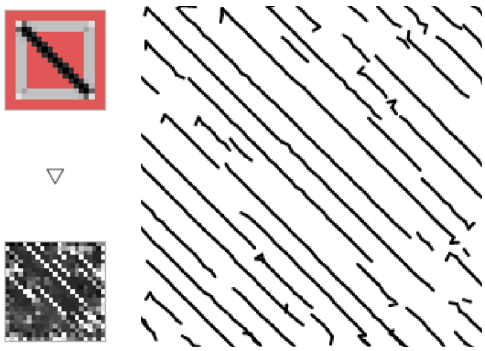
The drawing algorithm uses a shallow CNN as its guiding function. It draws new lines or continues drawing the line at its current position. The algorithm prefers to start new lines at line ends (which it detects using the line end detector). The segments it adds either make a mark or erase. The result is a drawing system that reacts to its situation without any memory of how the drawing was made (except the drawing agent's current location).

I provided the drawing system with a series of random drawings to see how it would react. The bottles produced incorporated elements of what was on the canvas and successfully moved the image towards the shape of a bottle in a rough-hewn and abstract manner.

Figure 4.33 The top row images are random drawings by me. The bottom row images show the same images after running the bottle network.

4.2.2 Drawing a Composition

After testing the bottle shape, I turned my attention to building compositions with it. To create artwork with the drawing system I needed to find a way to put multiple instances of the bottle on the same canvas. During development the networks acted as reward functions for the drawing algorithm at a particular location, but for rendering I scaled them to the size of the canvas. Before using bottles, I tested out making compositions with simple networks. My hope was that the system would draw versions of a single feature all over the canvas. Instead, the algorithm ended up drawing only one part of the feature.

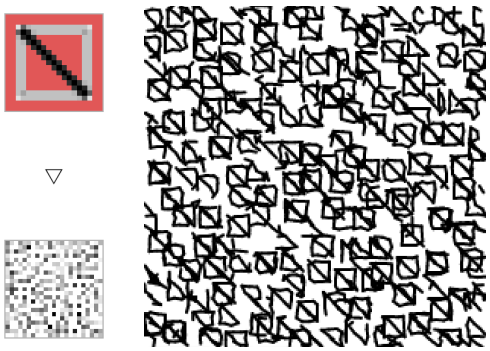


(a) (b)

Figure 4.34 (a) The red square with a black diagonal is a kernel (black/gray is positive, red is negative). The square below it shows the activation scores from the composition that was drawn. (b) The composition drawn by the system.

At any given point, the system's strongest motivation was to draw a diagonal line since that had the strongest weight in the kernel. As it drew, each diagonal line began to activate the diagonal line for a kernel in the neighboring position and the line ended up continuing to the edge of the canvas.

To remedy this, I decided to work towards exceptional individual scores rather than maximizing the sum of scores. I created a wrapper algorithm that focused selective attention on one location at a time. Its goal was to improve the score at a location until it could do no better and then move on. After this change, the drawing system drew complete versions of the target feature.



(a) (b)

Figure 4.35 The reward function has been given an attention state such that the drawing algorithm focuses on maximizing activation at one location at a time. The result was completed individuals.

4.2.3 Conceptual Work

Once I had the ability to render bottles in compositions, I began experimenting with creating artistic images. My hope was to create physical artifacts suitable for decorating a wall. The CNNs used in these examples act as feature detectors and reward functions. Each system chooses locations based on where the CNN detects line ends and high activations. Once it has chosen a location, it uses the CNN to test potential changes to the canvas.

My goal with the following conceptual work was to leverage the CNNs ability to dynamically respond to input. One strength of the drawing system was that it did not have to know how the image got to the state it was in. It could work with human input or other drawing systems as long as the marks on the page were somehow meaningful to it.

Multiple Drawing Systems Interacting

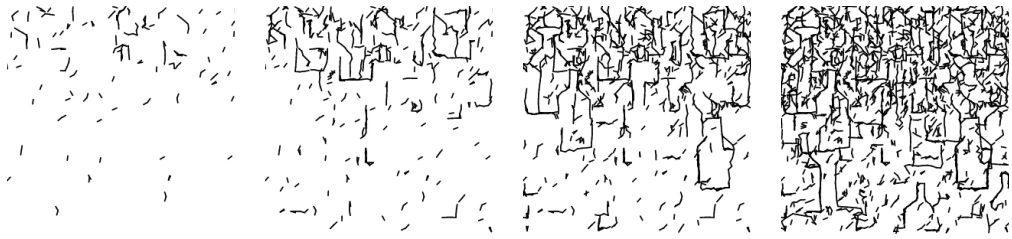
First, I made artworks where multiple versions of the drawing system interacted on the same canvas. I considered using CNNs with different visual concepts, but I realized that the CNNs needed to share structures in order for rich interactions to emerge. I wanted them to compete and build off of each other. A single network shares structures within and between

layers. The neuron representing the top of a bottle could build off of the neuron representing a corner from the layer below it. My plan was for the higher layers to absorb and build off of the lower layers. The neurons would randomly cast their focus around the canvas and concentrate on maximizing activations that were already partially developed.

I imagined all the drawing systems corresponding to different neurons simultaneously entering the scene in waves. The lower layers would create promising building blocks for the next layer. The image would start out as primitive fragments and would begin to form higher and higher orders of image "life forms." The process would unfold like life forming from amino acids in a primordial soup or like a wave crashing on the beach with foam made up of increasingly elemental pieces of the visual concept.

(a) (b) (c) (d)

Figure 4.36 A box emerging from the primitive building blocks of lines, corners, and edges.



(a) (b) (c) (d)

Figure 4.37 A wave of bottles that gets a little crowded.

I also tried waves expanding in concentric circles from the center and waves that spanned the whole canvas, tinting the latter with watercolor.

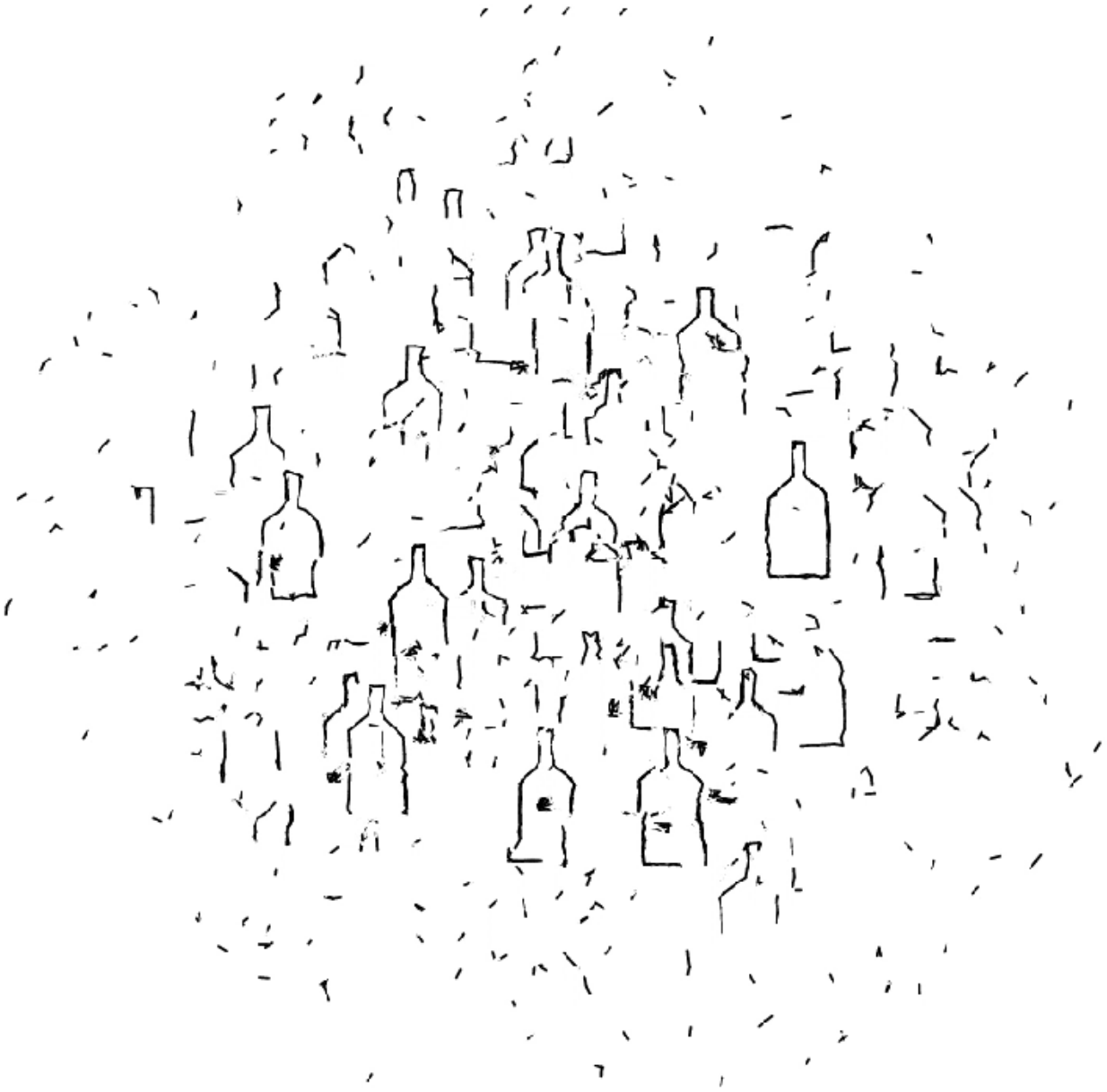


Figure 4.38 Waves of bottle parts growing from the center.

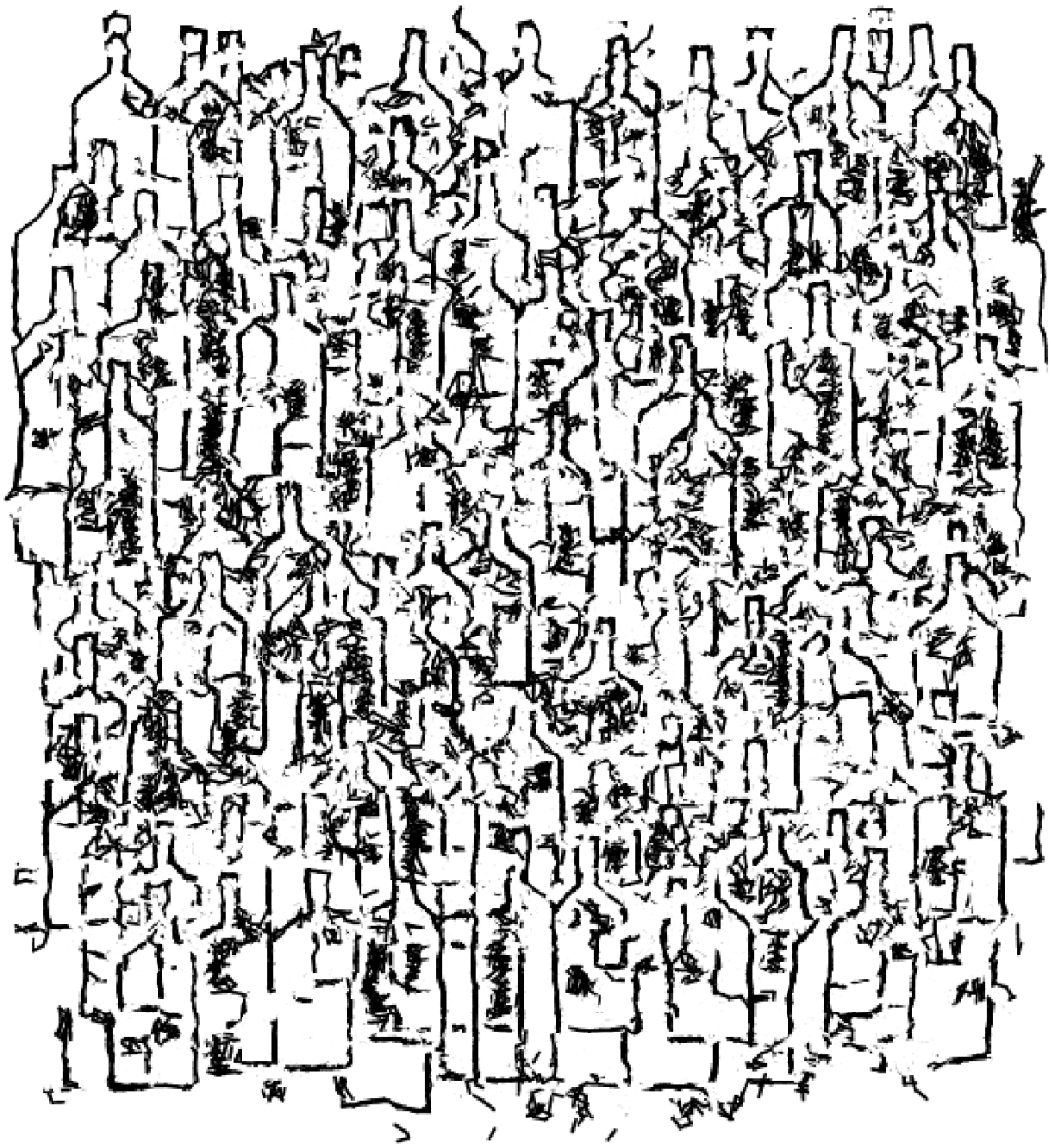


Figure 4.39 Bottles rendered to the whole canvas, resulting in a wall paper-like appearance.

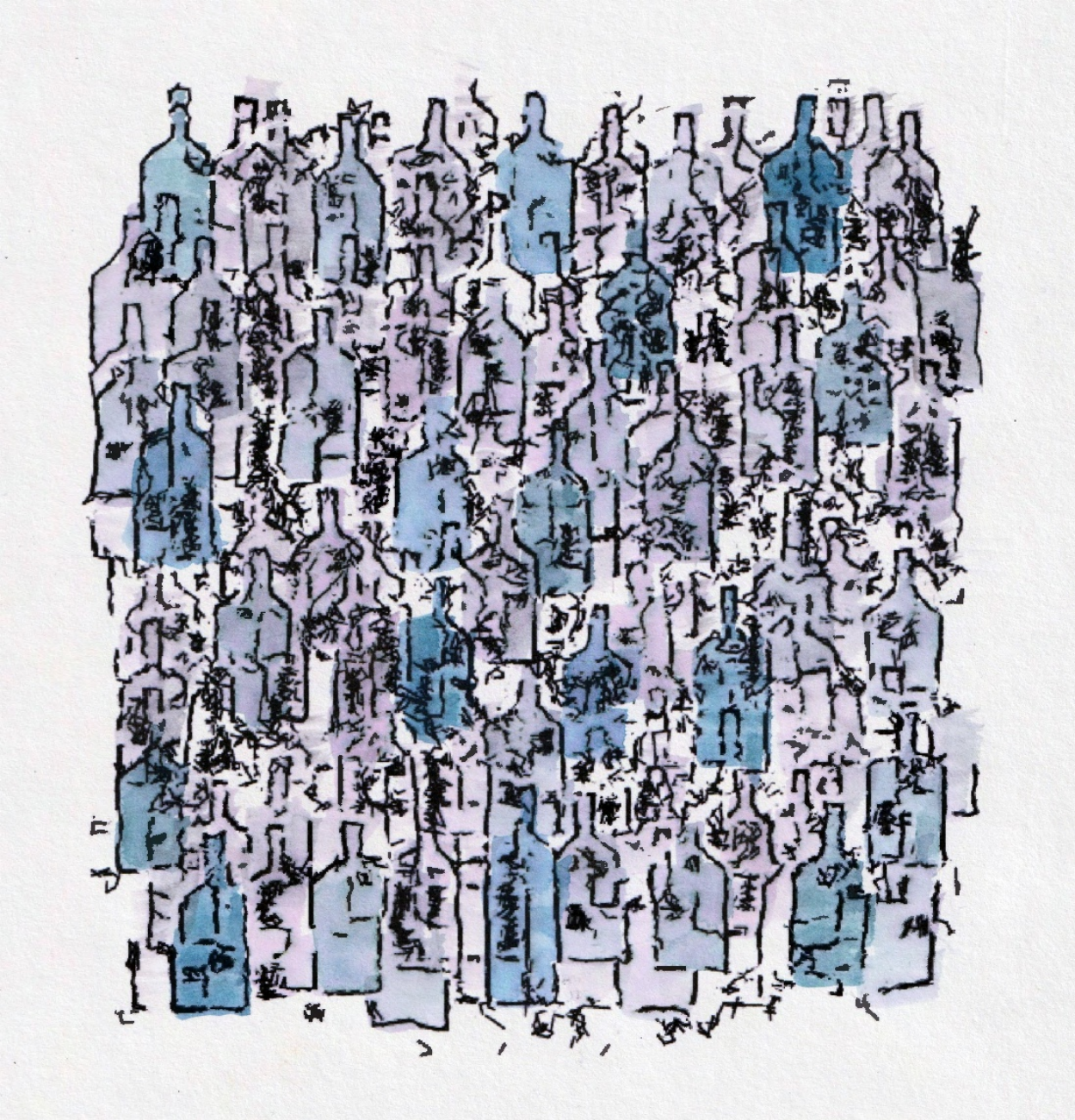


Figure 4.40 Hand-painted version no. 1.

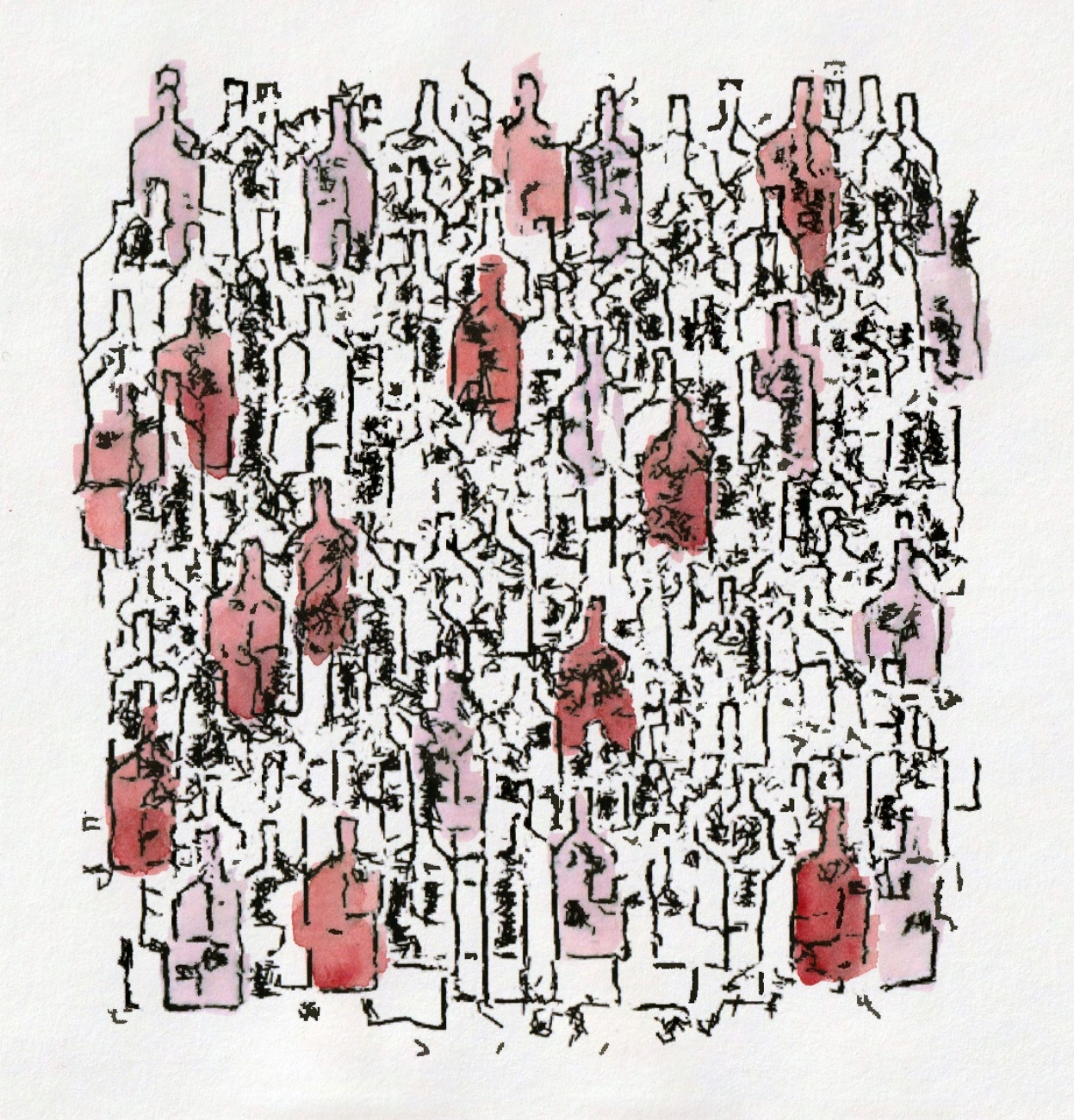
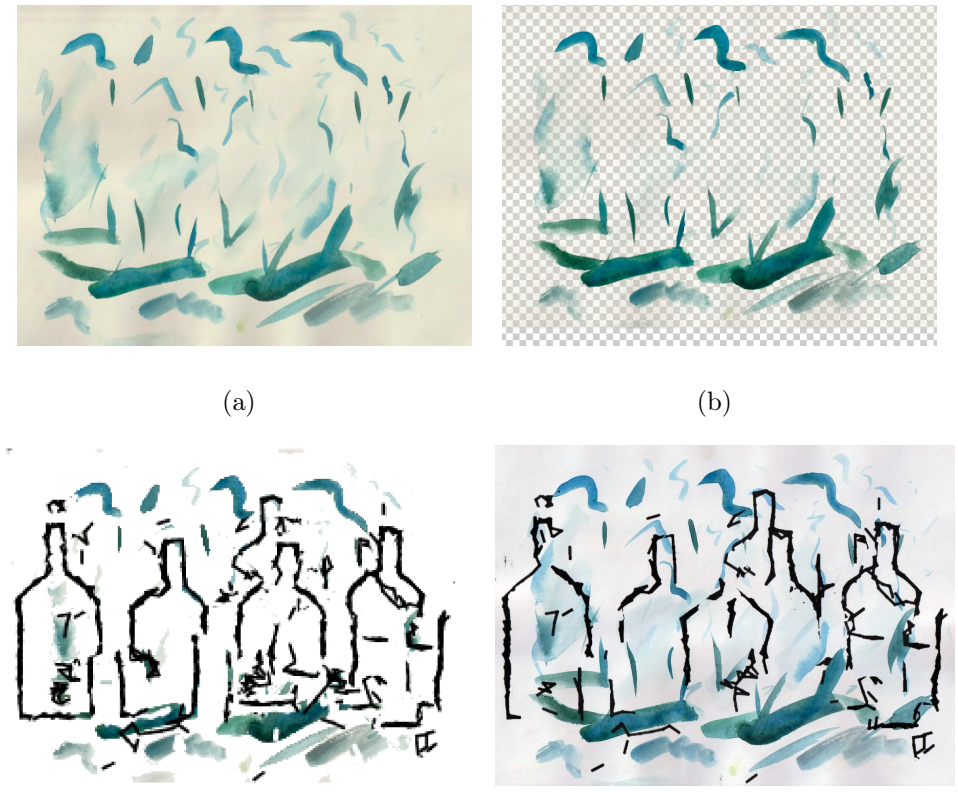


Figure 4.41 Hand-painted version no. 2.

Reacting to Human Input

I also wanted to leverage the dynamic ability of the system to respond to human input. So, I painted suggestive shapes, scanned them into the computer, performed basic image processing, and then ran the drawing system on the input. The idea was to use the network to release me from the need to give specific definition to the paintings. By adding specific structures after the fact, I would be able to focus on laying down the paint loosely and expressively without worrying. Trying to make a watercolor look like something can get in the way of the beauty of the watercolor pigment.



(a) (b)

(c) (d)

Figure 4.42 The original watercolor image (a) was scanned into the system (b). The bottle network was run on it (c) and then the marks were printed onto the original painting (d).

The results were not as strong as the previous experiments. In addition, it did not free me from worrying about what I was painting. My goal had been to stay loose and expressive

while laying the down paint, but I had trouble finding a balance between suggestion and openness. The system was cantankerous about responding to the images I made, probably because washes of watercolor are not a part of its vocabulary. I had decided to use paint to delineate a clear distinction between the digital canvas and real life through its stochastic detail. I thought that the image would still have enough for the drawing system to work with, but I struggled to find a style of painting stroke that it could work with. In retrospect, I should have stuck to black and white lines.

To test with black and white lines, I searched for existing artwork by other artists to use as the prompt for the system. My thought was the existing composition in the artwork would serve as scaffolding for the bottle drawing network. I tried a few different pieces, but I liked the outcome from a Jackson Pollock canvas the best.

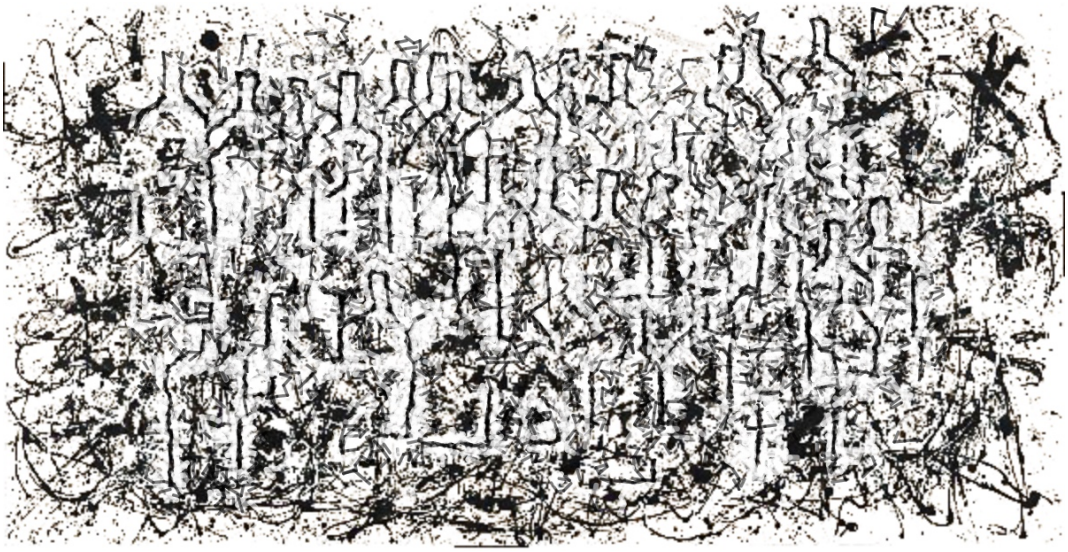


Figure 4.43 The result of running the bottle network on Jackson Pollock's *Autumn Rhythm (Number 30)* (Pollock, 1950).

Figure 4.44 The same image with Pollock's painting removed.

4.2.4 Discussion of the Artistic Experiments

These explorations demonstrate a few of the potential ways hand-crafted CNNs can be used for art making. Through this process my focus was on producing art, but the results helped me to examine the question: What does it mean to use a CNN as a form of representation for art making?

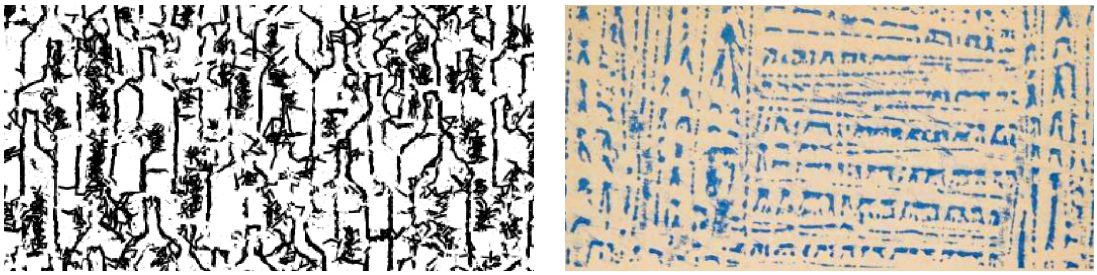
Comparing Gestalt and Frottage with Convolutional Neural Networks

CNNs, whether formed by hand or through data, seem to bear relation to both gestalt and frottage from Surreal art. Gestalt is a popular theme with Surrealist artists, and can be seen reflected in the piece in Figure 4.45 by Paul Klee. In Klee's work, the eye moves around, tracing the suggestive lines and catching short impressions that fade between texture and form. Likewise, the artistic outputs of this thesis stimulated my subconscious in particular ways. For example, the concentric circles created with the bottle network triggered an optical effect. The bits of structure caused my eye to be active, shifting back and forth between texture and individual elements in a search for completeness.

Figure 4.45 *Rich Harbour (Picture of a Journey)* (Klee, 1938)

In addition, the appearance of my outputs closely corresponded to artworks produced

through frottage. However, Max Ernst used frottage to free himself from the pressures of the blank canvas and to use the texture as an "optical provocateur." (Ernst et al., 1969) In my explorations, I too sought a release from the fussiness of representation, but I did not use the output as inspiration for free association. Instead, one could argue that the drawing system used my input as its jumping off point. I added structure indirectly through the CNN after loosely making a painting. My technique was related in terms of goals, but was the opposite with respect to procedure.



(a) (b)

Figure 4.46 Excerpt from a drawing with the bottle network bears a visual similarity to the Surrealist technique of frottage (Ernst et al., 1969).

Similarly, Philipp Schmitt draws a distinction between machine learning art and frottage in that machine learning art often seeks representation while frottage traditionally dissolves it (Schmitt, 2018). While this might be the case, it is important to highlight that CNNs focus on a general form of representation. They strike a balance of dissolving specific details while maintaining crucial structures. For me, this ability to mimic human gestalt by handling messy visual impressions is the most exciting part of CNNs. Computers have a tendency towards exactness, but a CNN's robustness to variance allows it to operate in a more organic realm.

The Rough-Hewn Aesthetic of Convolutional Neural Networks

The bottle images I produced can be described as rough-hewn in comparison to what one might generate with a traditional, rule-based approach. If we think about a piece of wood shaped by a hand axe versus a mill, I created networks with the equivalent of the axe. Rough-hewn is often used in a negative sense to mean lacking refinement. However, it can also mean a material that has been shaped as necessary for its function, but not frivolously smoothed.

Personally, I enjoy this aesthetic. According to art historian Pierre Francastel, the "rough sketch" has been popular since Impressionism. He notes that while many observers mistake a rough piece as incomplete or unfinished, it can also be the result of an "extreme precision" in a process that "simply responded to dictates that differed from those of classical art." Through this lens, a drawing system guided by a CNN can be seen as following the "dictates" of the visual concept that the CNN encodes. The system perceives a gestalt of the canvas through the CNN and retains structures it considers crucial while discarding others. It distills the visual concept to its core features. Francastel sees this distillation as the basis for transcending imitation and progressing into the realm of true creation. (Francastel and Cherry, 2000)

So, are the networks in this thesis rough-hewn in the positive sense that Francastel was talking about with the rough sketch or are they simply inefficient?

On the one hand, my method did not feel like "extreme precision." It was slow and difficult and the tools did not quite give me a clear view of the impact of my choices. There was still an element of guess and check and I had to accept what the network was willing to give me. But, ultimately, the weights were still directly chosen by me. I was able to inject at least some of my own dictates. The tools did not fully convert me into a craftsman of network weights, but it feels like I was at least pointed in the right direction.

Chapter Five

Conclusion

5.1 Summary

This thesis has proposed two tools for hand-crafting small CNNs for art making purposes: the Kernel Tuner and the Network Builder. The purpose of these tools was to make working with kernel weights a seamless experience by providing information directly at the point of action.

Before reading this, an artist might have assumed that hand-crafting could only be used unintentionally by randomly assigning weights to make glitch art. My contribution with this thesis was demonstrating two tools that gave me slightly more visibility when working with inner weights. I described my experience with the tools and the tightly woven feedback afforded by simultaneous interaction with a canvas and a visualization of a CNN. In addition, I provided examples of hand-crafted networks used to make intentional pictures and discussed how they related to other art forms.

While the process was far from seamless, the exercise of designing and using the tools provided fertile ground for thinking about representations within CNNs and the aesthetic value proposition of layers of convolution.

5.1.1 Convolutional Neural Networks as a Representation

What does it mean to use a CNN in the production of art? AI art always replaces some aspect of the human art making process. There are many models for this process, but, in my opinion, the high level phases are planning, action, and judgment. These phases are interwoven and repeat, but those are the main parts. An artist has some sort of plan (although it may be vague or partial). They execute part of that plan. Then, they view the result and make some decision about how to proceed.

A CNN replaces part of the judging portion. In that sense, it is view-obsessed. This stands in contrast to rule-based approaches or RNNs that focus on the planning step. CNNs may not explicitly make decisions, but they control how the system perceives the state of the image. Perception of a situation deeply informs any decisions made with respect to it. CNNs can be used in art as an imitation of a portion of human perception. The question then becomes: is this imitation creative? Is it something an artist wants?

Harold Cohen believed the deficit in computer art arose not from imitation in of itself, but from imitation of results. He criticized "picture-processors" and modeled AARON after the elementary activities of human art making. (Cohen, 1973) His goal was an imitation of processes. Typical uses of CNNs for art-making depend on datasets of images. The results of these processes seem to be limited to geometric transformations of the original images (Todorov, 2019). Whether those images are produced by the artist themselves or come from a public dataset, the system is imitating results, not processes. Thus, they would not satisfy Cohen's taste.

It is clear that the artwork presented in this thesis does not imitate results. There is no dataset to imitate. However, it does imitate part of the human process. It replaces a portion of the judgement involved in art making with a mechanical version of perception historically modelled after human biology (Fukushima, 1980). A possible summary of my results would be that I created a drawing system that pushes a rough-hewn image towards a bottle image

containing parts I determined to be crucial. However, this assessment paints an overly rosy picture of the usability of hand-crafted CNNs by implying I was able to "determine" these crucial parts. In fact, I found it to be a slow and laborious process that required me to accept what the network was willing to give me. So, is hand-crafting a CNN worthwhile?

5.1.2 The Economy of Hand-Crafted Convolutional Neural Networks

I have always felt that artwork is beautiful if the process used to create it is beautiful. This process oriented view aligns with Cohen's. Additionally, I prefer some elegance or economy of input when producing an artwork. Simple and quick is usually best. No one likes fussiness. I am not saying that artwork should be easy, but I subscribe to the popular notion that an artwork should not be more complex than it needs to be.

The bottle images produced in this thesis are relatively economic in terms of the complexity of the CNN used to make them. Compared to the millions of weights in most CNNs, they are tiny and comprehensible. On the other hand, when comparing the amount of labor that went into hand-crafting versus what an artist might use to build a network in a data-driven fashion, it was much less economic. Although, this calculation might even out if we consider the hidden labors that go into producing and collecting datasets.

We could also compare the bottle drawing system to rule-based artwork. When Sol LeWitt sold a wall drawing, a signed document containing the rules would accompany any transfer of the drawings (LeWitt, 1970b). On the next page, I have reproduced the weights of the bottle network and a set of LeWitt's rules. Both easily fit on the page, but LeWitt's rules are easier for a human to read and execute.

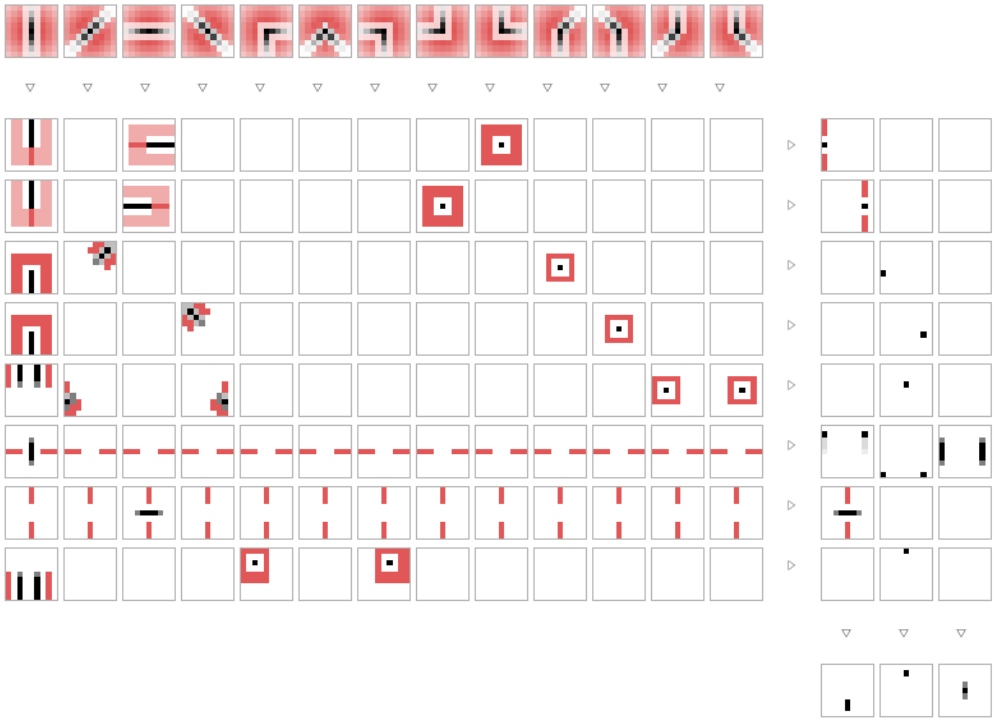


Figure 5.1 The weights of the bottle network.

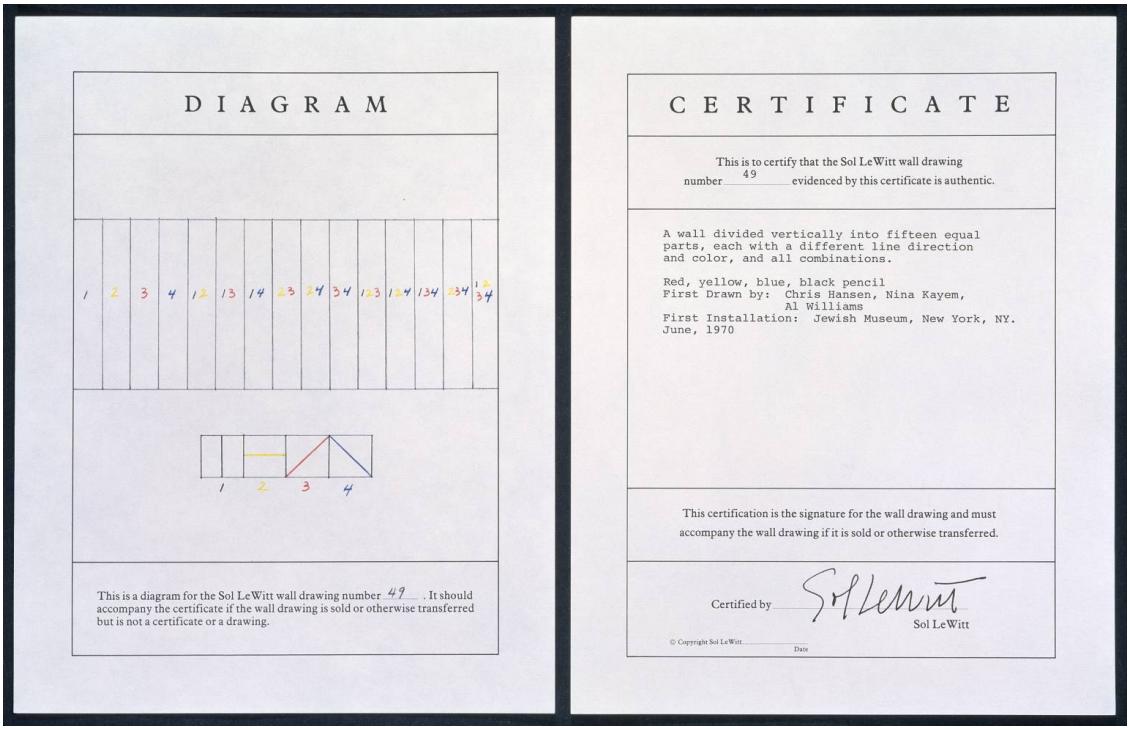


Figure 5.2 Rules for *A Wall Divided Vertically into Fifteen Equal Parts, Each with a Different Line Direction and Colour, and All Combinations* (LeWitt, 1970a).

Finally, there is the comparison to traditional forms of art making. One could say that the CNNs in this thesis are much less complex than the collection of atoms in a physical object. However, if we use a more common notion of complexity, such as which feels simpler, the hand-crafted CNNs would lose out. I could much more easily have sat down and made drawings with a pen and paper.

Yet, experimenting with CNNs as a form of representation pushed me to think about images in new ways. When I see an image, I tend to think of it as made up of distinct parts. With neural networks, these atomic units fade into a noisy collections of signals. A leaf is not a "leaf" unit. Instead, it is a collection of visual frequencies (many of which we do not have names for) communicating that it is oval, pointy, green, shiny, veined, etc. This way of thinking about images is built into CNNs. It shaped my process while working with the tools in this thesis, and it will now be in the back of my mind whenever I make images going forward.

5.2 Future Work

The investigations in this thesis suggested possible paths forward for using hand-crafted CNNs within dynamic art making systems.

First off, I believe that the tools in this thesis could be expanded upon to support the creation of low level feature detectors for creative coding platforms. Robust systems exist for detecting features such as hands, faces, animals, or custom objects, but most of them are oriented towards high level features (*ml5js* 2020; Webster, 2019; Fiebrink, 2009; Lundgren, 2020). Low level feature detectors might not be as sophisticated, but they could be combined with layers of rules to create rich and highly complex behavior. Like other CNN-based detectors, their strength would be their robustness with respect to organic visual materials. Thus, they could support artists working with randomness or stochastic data from the natural world.

In addition, future research could improve the process of hand-crafting CNNs through a hybrid method incorporating both direct manipulation and data-driven tuning. The major challenge encountered within this project was the delicate interdependencies between network weights. While hand-crafting allows us to integrate our expert knowledge of a visual material, we can no longer leverage example images. Combining human-designed features with learned features would leverage both human cleverness and computational power. The human could set up a general architecture and add obvious features to jump start the data-driven process which would in turn handle tuning weights and filling in gaps in the representation. This method is already common practice in most computer vision research using hand-crafting, but I believe it could be pushed further with new tools and applied to art making. By using a hybrid method, humans could communicate artistic concepts to a machine through what I see as the ideal mode of human-computer interaction: directly through hand-crafting and indirectly through examples.

5.3 Final Thoughts

The tools that artists use and the forms of representation that they enable shape how artists perceive and react to a work in progress. Many machine learning artists maintain a certain remoteness to the networks and datasets that they use. Their tools focus on generic image recognition tasks and only offer coarse-grained creative control. Interaction mediated by prepackaged feature detectors makes it difficult for an artist to develop an intimate relationship with their material. Digital designer Malcolm McCullough advises us that craft is most compelling when we move past the manipulation of symbols to "continuous operations on a workable medium" (McCullough, 1998). The tools in this thesis re-imagine network weights as a workable medium of a continuous nature. With further development, this form of representation could prove a rich mode of communication between artists, computation, and visual materials.

REFERENCES

Akten, M. (2017). *Learning to See*. URL: http://www.memo.tv/portfolio/learning-to-see/.

Alonso, N. M. (2017). "Suggestive Drawing Among Human and Artificial Intelligences". MA thesis. Harvard University Graduate School of Design.

Antipov, G. et al. (2015). "Learned vs. Hand-Crafted Features for Pedestrian Gender Recognition". In: *Proceedings of the 23rd ACM international conference on Multimedia*. MM '15. Brisbane, Australia: Association for Computing Machinery, pp. 1263–1266.

Barrot, R. and R. Barrat (2019). *BARRAT / BARROT, INFINITE SKULLS*. URL: http://avant-galerie.com/Infinite-Skulls-Barrat-Barrot-press-release.pdf.

Cavigelli, L. and L. Benini (2019). "CBinfer: Exploiting frame-to-frame locality for faster convolutional network inference on video streams". In: *IEEE Transactions on Circuits and Systems for Video Technology*.

Chherawala, Y., P. P. Roy, and M. Cheriet (2013). "Feature Design for Offline Arabic Handwriting Recognition: Handcrafted vs Automated?" In: *2013 12th International Conference on Document Analysis and Recognition*, pp. 290–294.

Cohen, H. (1964). "Search". In: *The New York Times*. URL: https://www.nytimes.com/slideshow/2016/05/09/obituaries/harold-cohens-assisted-artistry.html (visited on 04/25/2020).

— (1973). "Parallel to perception: some notes on the problem of machine-generated art". In: *Computer Studies* 4.3/4. URL: http://www.aaronshome.com/aaron/publications/paralleltoperception.pdf.

— (1974). "On purpose: an enquiry into the possible roles of the computer in art". In: *Studio International* 187.962, pp. 9–16.

— (1976). *The material of symbols*. University of Nevada. URL: http://www.aaronshome.com/aaron/publications/matofsym.pdf.

— (1979). "What is an Image?" In: *Proceedings of the 6th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'79. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 1028–1057. URL: http://www.aaronshome.com/aaron/publications/whatisanimage.pdf (visited on 10/06/2019).

Cohen, H. (1982). "How to make a drawing". In: *talk given to the Science Colloquium, National Bureau of Standards, Washington DC*. Vol. 17.

— (1988). "How to Draw Three People in a Botanical Garden." In: *AAAI*. Vol. 89, pp. 846–855.

Cohen, H. and AARON (1981). *82P2*. URL: https://www.nytimes.com/slideshow/2016/05/09/obituaries/harold-cohens-assisted-artistry.html (visited on 04/25/2020).

— (1986). "Athlete Series". In: *The New York Times*. URL: https://www.nytimes.com/slideshow/2016/05/09/obituaries/harold-cohens-assisted-artistry.html (visited on 04/25/2020).

Dalal, N. and B. Triggs (2005). "Histograms of oriented gradients for human detection". In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. IEEE, pp. 886–893.

Davis, N. M. et al. (2016). "Co-creative drawing agent with object recognition". In: *Twelfth artificial intelligence and interactive digital entertainment conference*.

d'Inverno, M. and J. McCormack (2015). "Heroic versus collaborative AI for the arts". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Ernst, M. et al. (1969). *Max Ernst*. New York: H. N. Abrams.

Fiebrink, R. (2009). *Wekinator | Software for real-time, interactive machine learning*. URL: http://www.wekinator.org/ (visited on 09/19/2019).

Francastel, P. and R. Cherry (2000). "Art & technology". In: *Trans. Randall Cherry. New York: Zone Books*.

Fukushima, K. (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4, pp. 193–202.

Goodfellow, I. et al. (2014). "Generative adversarial nets". In: *Advances in neural information processing systems*, pp. 2672–2680.

Greff, K., R. K. Srivastava, and J. Schmidhuber (2017). "Highway and Residual Networks learn Unrolled Iterative Estimation". In: *arXiv:1612.07771 [cs]*.

Grimes, W. (2016). *Harold Cohen, a Pioneer of Computer-Generated Art, Dies at 87 - The New York Times*. URL: https://www.nytimes.com/2016/05/07/arts/design/harold-cohen-a-pioneer-of-computer-generated-art-dies-at-87.html (visited on 10/06/2019).

Ha, D. and D. Eck (2017). "A Neural Representation of Sketch Drawings". In: *CoRR* abs/1704.03477.

Hadji, I. and R. P. Wildes (2018). "What do we understand about convolutional networks?" In: *arXiv preprint arXiv:1803.08834*.

"Harold Cohen with a painting machine at the Computer Museum in Boston in 1995." (1995). In: *The New York Times*. URL: https://www.nytimes.com/slideshow/2016/05/09/obituaries/harold-cohens-assisted-artistry.html (visited on 04/25/2020).

Herbert, F. (1965). *Dune*.

Hu, W. (2019). "Driverless Cars Arrive in New York City". In: *The New York Times*. URL: https://www.nytimes.com/2019/08/06/nyregion/driverless-cars-new-york-city.html (visited on 04/14/2020).

Huang, Z., W. Heng, and S. Zhou (2019). "Learning to Paint With Model-based Deep Reinforcement Learning". In: *arXiv:1903.04411 [cs]*.

Jansen, T. (2020). *Explains – Strandbeest*. URL: https://www.strandbeest.com/explains (visited on 04/24/2020).

— (1994). *Sabulosa Cutis*. URL: https://www.strandbeest.com/genealogy.

Jones, J. P. and L. A. Palmer (1987). "An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex". In: *Journal of Neurophysiology* 58.6, pp. 1233–1258.

Klee, P. (1938). *Rich Harbour (Picture of a Journey)*.

Klingemann, M. (2018). *Neural Glitch*. URL: http://underdestruction.com/2018/10/28/neural-glitch/.

LeWitt, S. (1967). "Paragraphs on conceptual art". In: *Artforum* 5.10, pp. 79–83.

— (1970a). *A Wall Divided Vertically into Fifteen Equal Parts, Each with a Different Line Direction and Colour, and All Combinations*.

— (1970b). *Untitled, from Composite Series*.

Li, L. et al. (2018). "Sketch-R2CNN: An Attentive Network for Vector Sketch Recognition". In: *arXiv:1811.08170 [cs]*.

Lowe, D. G. (2004). "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2, pp. 91–110.

Lundgren, E. (2020). *tracking.js*. URL: https://trackingjs.com/ (visited on 04/24/2020).

Majtner, T., S. Yildirim-Yayilgan, and J. Y. Hardeberg (2016). "Combining deep learning and hand-crafted features for skin lesion classification". In: *2016 Sixth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pp. 1–6.

McCorduck, P. (1991). *Aaron's code: meta-art, artificial intelligence, and the work of Harold Cohen*. Macmillan.

McCullough, M. (1998). *Abstracting craft: The practiced digital hand*. MIT press.

Mehrotra, R., K. R. Namuduri, and N. Ranganathan (1992). "Gabor filter-based edge detection". In: *Pattern recognition* 25.12, pp. 1479–1494.

Mellor, J. F. J. et al. (2019). "Unsupervised Doodling and Painting with Improved SPIRAL". In: *arXiv:1910.01007 [cs, stat]*.

Metz, C. (2019). "A.I. Shows Promise Assisting Physicians". In: *The New York Times*. URL: https://www.nytimes.com/2019/02/11/health/artificial-intelligence-medical-diagnosis.html (visited on 04/14/2020).

ml5js (2020). URL: https://ml5js.org/ (visited on 04/24/2020).

Monet, C. (1890). *Stacks of Wheat (End of Summer)*.

Nanni, L., S. Ghidoni, and S. Brahnam (2017). "Handcrafted vs. non-handcrafted features for computer vision classification". In: *Pattern Recognition* 71, pp. 158–172. (Visited on 04/07/2020).

Negroponte, N. (1970). *The Architecture Machine*. M.I.T. Press.

Offert, F. (2019). *The Past, Present, and Future of AI Art*. URL: https://thegradient.pub/the-past-present-and-future-of-ai-art/ (visited on 09/10/2019).

Olah, C., A. Mordvintsev, and L. Schubert (2017). "Feature Visualization". In: *Distill* 2.11, e7. URL: https://distill.pub/2017/feature-visualization (visited on 05/02/2019).

Olah, C. et al. (2020a). "An Overview of Early Vision in InceptionV1". In: *Distill* 5.4, e00024.002. URL: https://distill.pub/2020/circuits/early-vision (visited on 04/24/2020).

— (2020b). "Zoom In: An Introduction to Circuits". In: *Distill* 5.3, e00024.001. URL: https://distill.pub/2020/circuits/zoom-in (visited on 04/14/2020).

Picasso, P. (1910). *Girl with a Mandolin (Fanny Tellier)*.

Platform and environment | TensorFlow.js (2020). URL: https://www.tensorflow.org/js/guide/platform_environment (visited on 04/25/2020).

Pollock, J. (1950). *Autumn Rhythm (Number 30)*.

Pye, D. (1968). *The nature and art of workmanship*. University Press Cambridge.

Reas, C. (2004). *{Software} Structures. Comments - Wall Drawing #85*. URL: https://artport.whitney.org/commissions/softwarestructures2016/_85/comments.html (visited on 04/14/2020).

Ridler, A. (2017). *Fall of the House of Usher*. URL: http://annaridler.com/fall-of-the-house-of-usher.

Sarin, H. (2018). *#neuralBricolage: An Independent Artist's Guide to AI Artwork That Doesn't Require a Fortune*. URL: https://www.artnome.com/news/2018/11/14/helena-sarin-why-bigger-isnt-always-better-with-gans-and-ai-art (visited on 09/10/2019).

— (2019a). *A Book of GANesis*. Divine Comedy in Tangled Representations.

— (2019b). *Twitter post*. URL: https://twitter.com/glagolista/status/1177752980558417920.

Schmitt, P. (2018). "Augmented imagination: machine learning art as automatism". In: *Plot (s), the Design Studies Journal* 5, pp. 25–32.

Schonberger, J. L. et al. (2017). "Comparative Evaluation of Hand-Crafted and Learned Local Features". In: pp. 1482–1491.

Shafkat, I. (2018). *Intuitively Understanding Convolutions for Deep Learning*. URL: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1 (visited on 02/07/2020).

Simon, H. (1969). *The Sciences of the Artificial*.

Stiny, G. and J. Gips (1971). "'Shape Grammars and the Generative Specification of Painting and Sculpture'". In: vol. 71, pp. 1460–1465.

Thorpe, C. E. and T. Kanade (1991). *Second Annual Report for Perception for Outdoor Navigation*. Carnegie Mellon University, The Robotics Institute.

Tiwari, S. (2020). *A Comparative Study of Deep Learning Models With Handcraft Features and Non-Handcraft Features for Automatic Plant Species Identification*. article. (Visited on 04/07/2020).

Todorov, P. (2019). "A Game of Dice: Machine Learning and the Question Concerning Art". In: *arXiv:1904.01957 [cs]*.

Velázquez, D. (1650). *Juan de Pareja*.

Wang, J. et al. (2016). "Unsupervised learning of object semantic parts from internal states of CNNs by population encoding". In: *arXiv:1511.06855 [cs]*.

Webster, B. (2019). *Teachable Machine Tutorial: Bananameter*. URL: https://medium.com/@warronbebster/teachable-machine-tutorial-bananameter-4bfffa765866? (visited on 12/11/2019).

White, T. (2018). *Perception Engines*. URL: https://medium.com/artists-and-machine-intelligence/perception-engines-8a46bc598d57 (visited on 03/27/2020).

Xu, M. et al. (2017). "Accelerating Convolutional Neural Networks for Continuous Mobile Vision via Cache Reuse". In: *arXiv preprint arXiv:1712.01670*.

Yu, Q. et al. (2017). "Sketch-a-Net: A Deep Neural Network that Beats Humans". In: *International Journal of Computer Vision* 122.3, pp. 411–425.

Zhang, Y. et al. (2020). "AGCNN: Adaptive Gabor Convolutional Neural Networks with Receptive Fields for Vein Biometric Recognition". In: *Concurrency and Computation: Practice and Experience*, e5697.

APPENDIX

Appendix A

A.1 The SmartCanvas

This project uses CNNs differently than most. Crucial to the performance of the AutoDraw function is a network that can efficiently handle many small updates to the canvas. To accomplish this, I built a wrapper called the SmartCanvas that only updates activations within the scope of a given change. While CNNs are interconnected, a single (x,y) location in the hypercolumn output at any given layer is only connected to an area corresponding to the kernel size in the previous layer. This means that a change affecting a 10x10 area on a 1000x1000 canvas should only change a tiny fraction of the calculations through the network from the calculations for the previous frame. If the output is cached and only the affected portions are updated, the potential speed gains can be immense. This phenomena has been noted by researchers developing software packages for CNNs processing videos seeking to take advantage of the "spatio-temporal sparsity of pixel changes" (Xu et al., 2017; Cavigelli and Benini, 2019). The implementations in these papers were not easily portable to my project so I looked into my own implementation.

After building the caching system, the performance of the drawing system observationally seemed to greatly improve. To verify, I ran a series of performance tests to quantify the effect. To evaluate the system, I tested it with the three backends available for TensorFlow.js: webgl (using the gpu), wasm (using web assembly), and cpu (using the cpu). WASM allows for near native code execution speeds for the web (JavaScript is very slow compared to native programming language) and is helpful for applications dealing with large amounts of numerical calculations. For most use cases, webgl is much faster than wasm and wasm

is much faster than cpu. However, as noted in the TensorFlow documentation, wasm can be faster for small models (which is what the project uses). (*Platform and environment | TensorFlow.js* 2020)

The implementation of these different backends has implications for the caching optimization. While webgl is extremely fast, it is not fast the first time an execution is run. This means that if we apply the same network to the same size input, the webgl backend can utilize the gpu to great effect. Unfortunately, if we run custom computations based on what part of the canvas that has been updated (which is the case with the SmartCanvas) we end up running custom computations quite often.

To decide if caching was a good idea and which backend to use, I ran a series of tests. I set up a drawing system that roughly matched my average setup with and without caching for each type of backend. I used a 1000x1000 canvas, 5 pixel segment length updates, and networks with seven layers (five conv2d (4374 weights) and two maxPool2d layers).

TensorFlow Backend	webgl	wasm	cpu
No caching	645 ms	594 ms	8723 ms
Caching (SmartCanvas)	137 ms (22-31 ms for repeated ops)	**11.4 ms**	24.7 ms

Table A.1 Average time to test update by drawing system (300 updates, excluding first).

While these benchmarks do not reflect extensive testing under different conditions, they tell a compelling story. They indicate that my use of caching can be many times faster than if I did not use caching. They also show that WASM is a good choice with or without caching.

I do not have reason to believe these performance gains will translate to other types of networks. This performance test and the networks used in this project reflect a very particular situation. Initial tests using a YOLO network were not promising. YOLO is much

deeper and wider with smaller kernels. As the size of the changes increases, the overhead of inserting and slicing data from the cache will end up making it slower than running a full update. This means that YOLO benefits more from keeping its calculations on the gpu and proportionally produces large amounts of activations relative to the number of computations it makes in convolutions. Based on my tests, I believe that the caching used in this project is only helpful for projects that use small networks with big kernels applied to large canvases subject to incremental changes. Caching is probably not a good idea for most other use cases.