Abstract

Genetic Algorithms for Product Design - Working in AI's workshop

Manya Sachdev

2024

This paper examines the power of mathematical and computational processes in a subjective, product design problem scenario. In this, the paper deals with text-to-image models to generate innovative designs and then aims to optimize the designs generated with the help of genetic algorithms (GAs). GAs are popular in navigating complex problem areas, especially those where value can be expressed in computable equations and results can be visualized by automated simulations. The algorithm leverages classic evolutionary processes to search large population spaces. A reference paper has been cited that details an interesting way of representing products as binary strings and performing computations on them. Human subjectivity in design has been accounted for when designing the fitness function to measure how good each solution is. The share of choices principle has been applied, highlighting the need and importance of user feedback in design. The product chosen for this experiment is a musical instrument. For further evaluation, a prototype of the product was built and gesture control functionality was added to it. Results point to an optimistic scenario as fitness values followed an upward trend across generations.

Genetic Algorithms for Product Design - Working in AI's workshop

A Capstone Project
Presented to the Faculty of Computer Science
of
Ashoka University
in partial fulfillment of the requirements for the Degree of
Postgraduate Diploma in Advanced Computer Science

by
Manya Sachdev

Advisor: Debayan Gupta

May, 2024

Contents

List of Figures **v**

Acknowledgements **vi**

1. Introduction **1**

2. Literature Review **3**

2.1. Text-to-Image Models 3

2.2. AI and Creativity 4

2.3. Dynamic Gesture Recognition 4

2.4. Touch Sensor Technology 5

2.5. Smart Musical Instruments 6

2.6. Genetic Algorithms 6

3. Experiment **10**

3.1. The Problem 10

3.2. The Solution 10

3.3. Methodology 12

3.3.1. Design Generation 12

3.3.2. Design Optimization 12

4. Genetic Algorithm **17**

5. Evaluation **20**

5.1. Results of Optimization 20

5.2. Building the Instrument 21

5.2.1. Adding Functionality 22

6. Discussion **28**

7. Conclusion **31**

Bibliography **33**

List of Figures

3.1. The Chosen Design 13
3.2. Prompts and Images Generated 15
3.3. Final Few Iterations of the Design Generation Process on DreamStudio 16

5.1. The Chosen Design 21
5.2. Shape of Input Arrays After Data Collection and Labelling 23
5.3. Model Summary 24
5.4. TTP223 Base Circuit 25
5.5. Gesture Recognition Model Evaluation 26

6.1. Test Data Label Prediction 30

Acknowledgements

I'm extremely grateful for my advisor, Prof. Debayan Gupta, for his insightful advice and continuous patience as I navigated doing this project. His optimistic attitude was very welcome as I changed the scope of my study multiple times. I want to thank Bhavesh Neekhra for all his help in the final stages as I scrambled to define my focus area. I also want to acknowledge all the people at Makerspace. Staff and student members both, offered support and answered my endless list of question with respect to different hardware functionalities.

Chapter 1

Introduction

The capabilities of artificial intelligence (AI) are growing rapidly. The last two years have seen an increase in both the forms of AI available and the ease at which they can be accessed. This has led to a global interest in exploring its potential in various domains.

Recently, with increasing sophistication in the technology behind generative models, its use and applicability has begun to be studied in creative settings. We know that AI can produce novel products. These spans domains such as text, music, art, code, images etc. The question this project aims to examine is not if AI can design; but if AI can design *sensibly*. Emphasis is placed on coming up with an optimized solution for a design problem. As part of the study a significant chunk of the project focuses on developing a genetic algorithm that can be applied in this area. One existing paper has been identified to help structure the problem and guide the solution process.

The product chosen for this experiment is a musical instrument. The design behind a musical instrument is an amalgamation of many factors. Some examples include material used, number of distinct notes possible, range of sounds etc.

Luthiers, since ancient times, have been experimenting with available technology

to create innovative musical instruments. Over the last two decades, this has evolved into the establishment of new class of instruments known as Digital Musical Instruments or DMIs. Computational equipment such as sensors, motors and actuators are used to extend the functionality of conventional instruments. Recently, a subclass within this domain has been defined. These are Smart Musical Instruments (SMI). DMIs that use real-time signal processing, wireless communication, and embedded hardware are known as Smart Musical Instruments.

Within the larger experiment, this paper aims to combine the smartification of instruments with the creative capabilities of AI. Two state-of-the-art text-to-image models have been employed to get started with design generation. One of the images generated was chosen according to a single experimenter's human bias. This chosen image formed the basis of the initial input for the genetic algorithm. Once the GA was run, arbitrary functionalities were added to the best solution for product testing.

Chapter 2

Literature Review

2.1 Text-to-Image Models

"Generative Adversarial Text-to-Image Synthesis" by Reed et al. [1] examines several approaches to synthesize realistic looking images from text-to-image models. It recognizes the ability of deep convolutional generative adversarial networks to come up with compelling visuals of specific prompts. It uses these results to design a new, deep architecture, complete with generative adversarial network formulation to imrove upon existing text-to-image models and help further refine the results produced by aligning them more closely with a detailed text input.

"AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks" by Xu et al [2] proposes an attentional generative adversarial network, AttnGAN, for fine-grained text-to-image generation. The model incorporates attention mechanisms to highlight and map text phrases to specific regions within a generated image.

2.2 AI and Creativity

"Creative AI: On the Democratisation & Escalation of Creativity" by Roelof Pieters & Samim Winiger [3] discusses the potential impact of AI on creativity and level at which it can contribute to it. It explores the democratization and escalation of different creative processes that can be facilitated by AI technology. The paper also highlights the role that AI can play in helping individuals better engage in creative endeavors across diverse and unfamiliar domains.

2.3 Dynamic Gesture Recognition

"A Survey of Hand Gesture Recognition Techniques" by Lingchen Chen et al. [4] explores the history of hand gesture recognition and the techniques and approaches used in the process, particularly using Kinect depth data. The paper also touches on the challenges, applications, and future directions in the field of hand gesture recognition using Kinect.

As established earlier, hand gesture recognition can be carried out using multiple different techniques. An in-depth review and exploration was given in the paper "Hand Gesture Recognition Based on Computer Vision: A Review of Techniques" by Oudah Munir [5]. The paper differentiates between static hand gestures and dynamic hand movements in hand gesture recognition. Static gestures focus on the stable shape of the the hand eg. a thumbs up. On the other hand, dynamic gestures focus on the movement pattern of the hand eg. waving. For either type, the paper outlines the approaches for gesture recognition - 1. Glove based approach 2. Computer Vision Approach. Under the CV approach, it mentions multiple options for recognition techniques. A color based recognition can be done using a glove marker. This takes into account pixel values and classifies them into skin or not skin. Another technique is the skeleton based recognition. This approach adds geometric markers on top of

the gesture image to improve detection of complex features.

In "Hand Gesture Controlled Drones: An Open Source Library" by Kathiravan Natarajan [6], the author discusses gesture controlled drones and image segregation from live video feeds. Over a series of experiments, it was discovered that variability in scene settings greatly influences the accuracy of accurate image classification. Bright spaces with low background noise make for good backdrops for the camera frame. A distance of 3ft was particularly identified as improving the accuracy score.

"Hand Gesture Recognition Algorithm Using SVM and HOG Model for Control of Robotic System" by Phat Nguyen Huu. [7] proposes a method for static hand gesture recognition based on support vector machines (SVM) and histogram of oriented gradient (HOG). One of the main objectives is to use a CNN model and write an algorithm that has the ability to ignore unintentional gestures. A 70 ms/frame execution time was identified as the best outcome giving approximately a 99% accuracy rate.

2.4 Touch Sensor Technology

"Capacitive Touch Systems With Styli for Touch Sensors: A Review" by O Kyun Kwon et al. [8] discuss the latest developments in capacitive touch sensor technology, materials and use with stlyi. It explores possible furture developments in capacitive touch technology as well.

"A Review of Tactile Sensing Technologies with Applications in Biomedical Engineering" by Mohsin Tiwana et al. [9] gives a comprehensive description of tactitle sensors. It reviews its immense potential, unfortunate failure and likely revival.

2.5 Smart Musical Instruments

"Smart Musical Instruments: Vision, Design Principles, and Future Directions" by Luca Turchet [10] reviews the family of SMIs or Smart Musical Instruments. It examines how ancient luthiers utilised the technology of their time to enhance traditional instruments and highlights the application of similar processes today. It explores the level of interactions that open up by *smartification* of an instrument. Further, it discusses specific feature that make an instrument smart such as memory, embedded technology, learnibility etc.

"Sensor-based musical instruments and interactive music" by Atau Tanaka [11] focuses heavily on NIME or New Interfaces for Musical Expression. It goes one step further than enhancing existing, traditional instruments. It explores how we can exploit various computational paradigms such as digital and analog signals to introduce previously unseen forms of musical expression. It highlights specific performers who have used technology to introduce different levels of interaction in their instruments with the help of sensor technology.

"A Touch on Musical Innovation: Exploring Wearables and Their Impact on New Interfaces for Musical Expression" by David Wexler et al. discusses the potential of wearing technology in musical production, expression and learning. It experiments these factors by developing a wearing instrument with over 30 sensors that provide multi-sensory feedback. The aim is enhance the overall musical experience of the wearer.

2.6 Genetic Algorithms

Most design solutions require the designer to resort to simulation to holistically understand and evaluate the factors involved. Multiple rounds of simulation help engage in a common design practice, based on a visual feedback loop. "Design Optimization

Tool Using Genetic Algorithm" by Luisa Gama [12] discusses the idea of *goal oriented design*. Here, the computer looks for design solutions in the vast space and identifies high performance solutions with respect to specified goals. A computer has the capability of automatically generating configurations and evaluating them, presenting the user with just the optimal result. The paper notes the challenges in this approach, stating the drafting of a proper evaluation criteria as the main hurdle in using genetic algorithm-esque tools to design and optimize existing designs. It recognizes that some fields are better suited to such optimization tools. This is because there is a large number of quantifiable factors present which can be objectively computed. For instance, in environmental analysis, we can obtimize for light, temperature or energy. All three factors here are quantifiable and can be optimized for using a mathematical equation.

Victor B. discusses grouping-genetic algorithms for modular product design [13]. Modular products are those which can embody multiple functionalities. They can be paired and configures in different ways. Each *module* is designed to have the maximum possible physical and functional relations with another. The aim is to optimize for similarity between specific modular forces. Victor's paper explores non-linear programming techniques to examine and subsequently identify distinct modules, both in terms of individual design and the overall number of separable components required. Here, the design process consists of four phases: the first involves a physical interaction analysis between modules. The functions and utility values are listed a correlation matrix. The second phase deals with refining the design mandate to help evaluate each module in a consistent manner. The third phase involves coming up with a concrete definition for the objective function. The fourth and final phase consists of using a heurestic grouping genetic algorithm to follow an evolutionary process and find the optimal solution.

Shih-Wen Hsiao [14] explores multiple optimization methods in his paper. These

include Neural Networks, Support Vector Machines and Genetic Algorithms. The problem statement around which the study is centered is minimizing the costs of new product development. The consumer wants were transformed into representative linguistic variables. Additionally, a MATLAB program was constructed to simulate various solutions for an easy visual evaluation. The neural network was found to have a slow search rate. The partial minimum was obtained fairly quickly. The SVM approach was heavy to run due to a large dataset requirement. It was also comparatively complex and mandated an extensive personnel training to actually use. The genetic algorithm approach was easier to grasp. It performed better and reached a higher efficiency rate when searching the solution space. The paper highlight several reasons for ultimately choosing the GA approach. The first, parameters are calculated using code. The second, the search model is based on search of an entire population of points as opposed to a single point. This results in the program being less susceptible to being trapped in a local minima. The third reason for preferring a GA is that fitness function can be highly customised and selective. Only the required information may be included. The last benefit outlined is the random nature of the searching. Since it is non-deterministic, it results in higher efficiency values.

P. V. (Sundar) Balakrishnan wrote a paper called "Genetic Algorithms in Product Design" [15] in which he explored how evolutionary code structures can be applied in a design scenario. To do so, he took the design problem and converted it to a string structure which representated the chromosone or product profile. For instance, a bar of soap was defined using three attributes - Attribute 1 - Shape, Attribute 2 - Color and Attribute 3 - Scent. Each attribute had a certain number of *levels* to it. This represent the different values that an attribute can take. Shape had three levels - rectangle, spherical and square. Color could be red, blue, white or yellow. Scent could take the value fruity, flowery or antiseptic. In such a description, the string 01 100 01 represent a square soap, yellow in color with an antiseptic scent.

Once the product profile had been well defined in terms of the specific attributes it could possess and the values the attributes could take, an initial population of random fixed-length binary strings was generated. The evaluation procedure used here is quite unique and solves the problem of looking for a mathematical equation to facilitate evaluation. The paper determines the utility for each string (product profile) by using the concept of part-worth utilities. Customers in a group were asked to rank preferences and wants with respect to various attributes and levels. They were also asked to declare their first choice configurations. Both sets of data collected were stored in different matrices to perform computations on at a later stage. Selection of one product over another was conducted by comparing the utility of a product profile with the utility of a product profile in the first-choice matrix.

After defining the evaluation procedure, the rest of the processes described in the paper are standard to most genetic algorithms. An elitest strategy was used during reproduction to ensure preservation of attributes of the best or fittest products. Uniform crossover was applied and mutation was carried out to introduce more variation in the population space.

Chapter 3

Experiment

3.1 The Problem

Design processes are fairly subjective. Even without placing this in a computational scenario, the problem of simply coming up with a new design that appeals to mass audiences is a hard one. There is always a subset of the population that may disagree with the design and its designer. The subjectivity has been considered heavily when designing the evaluation procedure.

3.2 The Solution

Certain problems are more suited to genetic algorithms than others. GAs follow an evolutionary process when it comes to computation. Broadly, the algorithm takes as input an initial population. Each profile in the population is represented by a string describing the profile's attributes according to some predefined structure. This string is referred to as a *chromosome*. Each chromosome is passed through a *fitness_function* that mathematically computes a numerical value defining the *fitness* of the string. The fitness is essentially a score that tells the user how good a given solution is. Hence, the higher fitness value a chromosome has, the *better* it is. After fitness

evaluation, some n number of fittest strings are chosen to be parents for the next generation. This is called Reproduction and the method of choosing strings here has acquired the name *elitest strategy*. This step is undertaken to reward the best performing chromosomes and ensure that the specific characteristics that make them so are preserved for the next generation. Next, the algorithm randomly makes pairs from the n fittest strings. These are *parents* to produce *offsprings*. This is called Crossover. There are several algorithms that the larger genetic algorithms can use to perform the crossover function. Some examples include uniform crossover and multi-crossover. Next, a Mutation function is run on the chosen subset of strings. Here, a predefined number of strings is chosen at random and certain values of the string are changed. For eg. In a binary representation, 1100 may be changed to 1101 when run through the mutation function. This is done to introduce more variation in the population. The rate of mutation (% of strings to mutate) has to be chosen carefully and may differ from application to application. If the number is too high, the population becomes too close to a random population as too many results from the reproduction and crossover stage are altered. Conversely, if the number is too low, the purpose of introducing necessary variability in the population is not fulfilled. The algorithm runs loops through reproduction, crossover and mutation stages iteratively until some predefined stopping condition is met. In the paper we have considered as our guide, the stopping condition is based on moving averages. In our application however, we specify a fixed number of iterations that the algorithm must run for.

This algorithm addresses the optimization part of our larger product design problem. An equally important part of any design process is evaluation and user testing. For products like musical instruments, usage cannot be ignored. Therefore, for the second half of this experiment, we build a prototype of the best solution outputted by the genetic algorithm. Choice of different user functionalities has been accounted for in the chromosome design to a large degree. Hence, our build process is well defined.

3.3 Methodology

3.3.1 Design Generation

The experiment starts with prompt engineering. This is where we start "working in AI's workshop" as stated in the title. Our aim at this stage is to simply use some readily available text-to-image models and generate designs of innovative musical instruments or robotics equipment that can be used to play music. Multiple variations of input sequences were fed to two such models. With each iteration, the experimenter's human bias was used to judge the results and alter the input sentence to move closer to or further away from the outputs. Over 100 images were generated across 22 different prompts until an "appropriate" design was chosen.

The two models used for image generation are Stable Diffusion and DreamStudio. Stable Diffusion 2.1, by StabilityAI, uses generative adversarial networks (GANs), to synthesize realistic and diverse images based on given textual prompts. DreamStudio is a text-to-image model that was specifically designed to generate imaginative, visually appealing images from textual inputs. It employs advanced deep learning architectures and attention mechanisms to capture semantic meanings from the inputs. The model is trained on a large and diverse dataset of image-text pairs which enables it to understand nuances of natural language. The final design was chosen according to the experimenter's bias. An image of the design can be seen in Figure 3.1.

3.3.2 Design Optimization

To fit this design into a genetic algorithm, we need to represent it as a string structure. We fix eight attributes to describe the chosen image, each with three to four levels it can take. The chromosome is a binary string of length *16 bits*. The attributes and their values are:

Attribute 1: Material Selection ['Wood', 'Metal', 'Plastic']

Figure 3.1: The Chosen Design

Attribute 2: Pad Arrangement ['Circular', 'Linear', 'Grid']

Attribute 3: Touch Sensor Configuration ['Single', 'Double', 'Triple']

Attribute 4: Enclosure Color ['Red', 'Blue', 'Black', 'White']

Attribute 5: Touch Pad Size ['Small', 'Medium', 'Large']

Attribute 6: Audio Output Type ['Speaker', 'Headphone Jack', 'Bluetooth']

Attribute 7: Touch Pad Material ['Conductive Metal', 'Conductive Polymer', 'Capacitive Glass']

Attribute 8: Instrument Weight ['Light', 'Medium', 'Heavy']

These attributes limit the *creativity* of the algorithm. They define the scope within which new configurations of the instrument will be confined. This helps avoid a situation where a new, unseen feature is outputted as an optimization suggestion. Such a scenario would put the responsibility back on the user in terms of deciding how the new feature fits into the original design. The way the attributes and values are defined directly decide the freedom the algorithm has to optimize.

The size of each generation (i.e. population) was set to be *50*. The number of maximum generations that the algorithm must run for was set as *100*. From

each generation, half of the chromosomes were selected to be parents for the next generation. Mutation rate was set as *0.1*. No stopping condition was applied, the model was set to run for all 100 generations.

The structure of the genetic algorithm used here follows closely to the structure defined in P. V. Balakrishnan's paper [15].

Overview:

```
Step 1: t <- 0
        generate initial population randomly = POP(t)
        evaluate each string in POP(t)
Step 4: if stopping condition is met:
            stop
        else:
            continue
        t <- t+1
        select strings from POP(t-1) to create POP(t)
        apply genetic operators on POP(t)
        evaluate each string in POP(t)
        got to step 4
```

The paper was published in the late 90s. We have taken advantage of the updates to Python since then and made use of the PyGAD Library. Certain features remain unchanged. These include the way the product is defined, method of calculating fitness and a customer's role evaluating designs. A more in-depth look into our implementation is given in the next section.

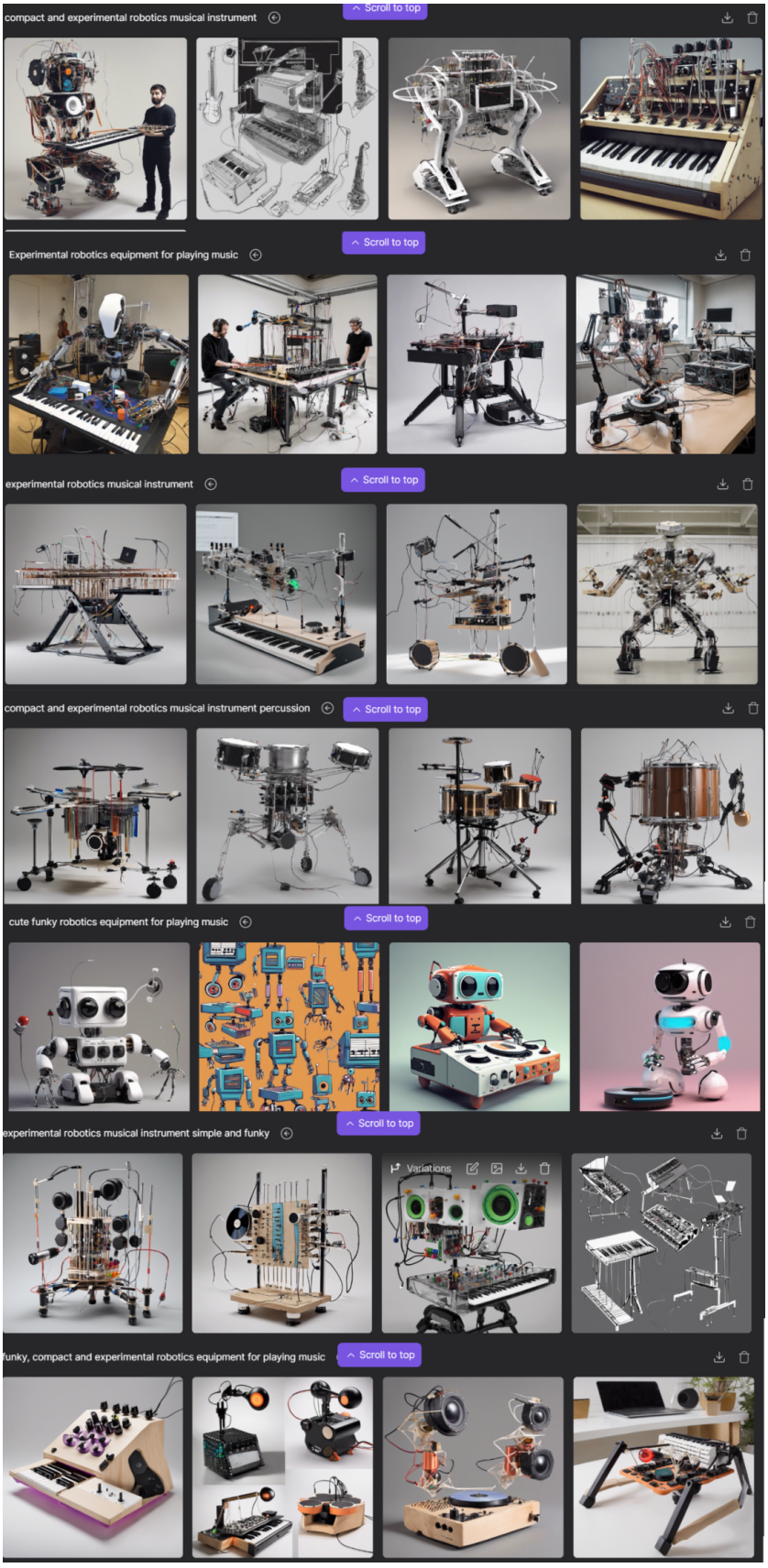

Figure 3.2: Prompts and Images Generated

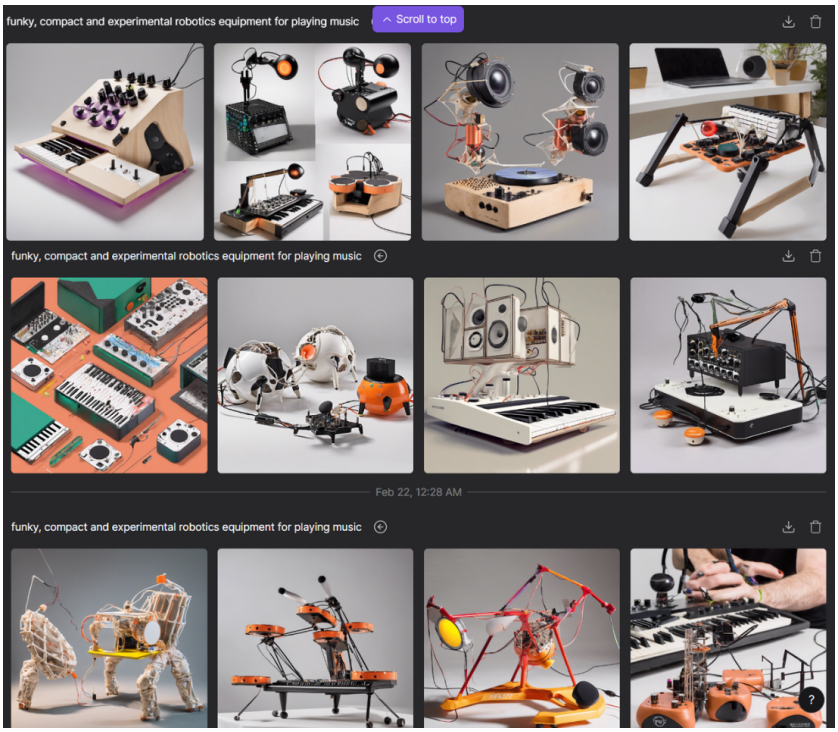

Figure 3.3: Final Few Iterations of the Design Generation Process on DreamStudio

Chapter 4

Genetic Algorithm

This section discusses the fitness function of our algorithm.

The logic behing the fitness function is the concept of *Share of choices*. This is a popular market technique for product evaluation. It is centered around measuring subjective consumer preferences for certain products. It refers to computing the percentage of a population that prefers one product over another. In our case, we introduce partworth utility to drive "preference". This method of assigning utility to a product refers to assigning a value or *utility* to each attribute of a product. This helps us perform a conjoint analysis that reflects how consumers make tradeoffs between different product attributes, i.e. measure relative importance.

To start with, we generated the partworth utility matrix by random as follows:

```
#Defining range of values for partworths utility [-1, 1]
conjoint_values = np.linspace(-1, 1, num=levels_per_attribute)
#Generating random values for each customer and attribute
BETA = np.random.choice(conjoint_values, size=(N, P))
```

Here, N is the number of customers and P is the number of strings required for binary string representation of a product profile.

The resultant *BETA* matrix:

```
[[ 0.  0. -1. ... -1. -1. -1.]
 [-1.  1.  0. ... -1.  1.  0.]
 [-1.  0. -1. ...  0.  0. -1.]
 ...
 [ 0.  1.  1. ...  0.  1. -1.]
 [-1. -1.  1. ... -1. -1.  0.]
 [ 1. -1. -1. ...  1.  0. -1.]]
```

We also defined the first-choice array of size N. Values were randomly generated. The resultant matrix:

```
array([ 7.23529244e-01, -8.08118284e-02,  1.34381586e-02, -2.87364144e-01,
   -8.39726207e-01,  3.94313729e-01, -6.26319013e-01, -5.87904219e-01,
   -4.70638691e-01,  7.54205297e-01,  9.67085013e-01,  8.18912963e-01,
    7.34670892e-01,  2.66770440e-01, -2.71661249e-01,  4.44161810e-01,
    ...
    5.48342497e-02, -1.62267510e+00, -3.73565389e-01, -2.60505890e-02,
    4.67892340e-01, -4.26847619e-01, -4.14381054e-01, -6.40516033e-01,
    5.73328836e-01,  8.77545232e-01,  1.40001113e+00, -2.78291675e-01,
   -1.21173575e+00, -1.69131722e+00,  2.81033021e-01, -3.25483580e-01])
```

The utility or *fitness* of a product was determined by multiplying the *BETA* matrix with the population matrix. This value was compared to values in the first-choice matrix. The product profile with the higher utility value was chosen to be retained as per the defined fitness function for future calculations.

This method of computing utility allows us to consider the preferences of a large population. There is no limit to the number of consumers that can be surveyed, making this method very scalable for application in a real-world scenario. The alternate to this evaluation method would be to take one experimenter's bias, similar to the

approach taken during the design generation phase and ask the generative model to make certain additions or subtractions to a design image.

The rest of the genetic operations used in the algorithm follow closely with the standard structure of most genetic algorithms. The operations include reproduction, crossover and mutation. In PyGAD, separate user-written functions for do not need to be specified for these genetic operations. We are only required to write functions to calculate fitness and decode the chromosome.

Chapter 5

Evaluation

5.1 Results of Optimization

```
Generation 1: Best solution fitness = 0.615

Generation 2: Best solution fitness = 0.6186

Generation 3: Best solution fitness = 0.6294

...

Generation 69: Best solution fitness = 0.7016

Generation 70: Best solution fitness = 0.7021999999999999

Generation 71: Best solution fitness = 0.7021999999999999

...

Generation 96: Best solution fitness = 0.7026

Generation 97: Best solution fitness = 0.7026

Generation 98: Best solution fitness = 0.7026

Generation 99: Best solution fitness = 0.7026
```

5.2 Building the Instrument

The instrument design chosen initially from the outputs generated by DreamStudio's text-to-image model can be seen in Figure 5.1. It features a sleek wooden enclosure with five to seven circular pads arranged arbitrarily on the top surface. Wires are seen connecting the pads to internal components housed within the enclosure.

Figure 5.1: The Chosen Design

The genetic algorithm gave us the string: 00 01 00 10 01 00 10 01 as the fittest individual. After decoding the string according to the predefined attributes and levels, we get a black wooden instrument with touch pads arranged in a linear fashion on the top surface. The touch sensors must be activated by a single touch. The pads must be of size medium, probably to account for individuals of all ages being able to access the instrument. The audio output must be via a speaker and the instrument as whole must not be too heavy ensuring reasonable portability.

5.2.1 Adding Functionality

In order to get suitable feedback, functionality was added to the product to make user interactions engaging.

The instrument is connected to a laptop. It uses the laptop's webcam to facilitate interaction. We wrote a dynamic gesture recognition model that is capable of recognizing five distinct action movements. These movements include:

1. Strumming the strings of a guitar
2. Moving the bow of a violin
3. Tapping on the keys of a piano
4. Hitting sticks on a drumset
5. Pressing buttons along the length of a saxophone

Visual feedback is provided to indicate the selected class, ensuring clarity to the user. The recognized action dictates what sound the instrument outputs, via the laptop's speakers. Individual musical notes can be triggered by touching the pads on the top surface of the instrument. Each pad elicits one, predefined note belonging to the chosen class of instrument. Users can create melodies and rhythms by tapping different pads separately or simultaneously.

Funcitionality Details

We use OpenCV's *VideoCapture()* function to initialize a video capture object. This object captures frames from the laptop's default webcam. We also use MediaPipe to initialize a holistic model that is capable of full-body, pose estimation.

1. A loop is initialized. It iterates over each classification category - piano, guitar, violin, drums and saxophone. For each category, it iterates over a specified number of sequences. For this project, this value has been predefined to be 30.

2. Within each sequence, it iterates over a specified number of frames. For this project, this value has been predefined to be 30. For each frame, it captures a video

frame using the webcam feed and performs pose estimation using the previously loaded MediaPipe holistic model. The model draws the detected landmarks (body pose, face, left hand, right hand) on the captured frame using a predefined function. The fully annotated frame is displayed to the user. Additionally, the current classification category for data collection and the current sequence (sample) number is displayed to tell the user what action to perform.

3. From each frame, we extract and export the detected keypoints to a .npy file and save them in the corresponding sub-directory.

4. Once the loop finishes execution, we have five classification categories. Each category has thirty "video" samples. Each same has thirty frames captured. Another small loop is run to assign labels to each sample.

```
In [ ]: np.array(sequences).shape

Out[ ]: (150, 30, 1662)

In [ ]: np.array(labels).shape

Out[ ]: (150,)
```

Figure 5.2: Shape of Input Arrays After Data Collection and Labelling

The gesture recognition model was built using functions from the Keras library. It consists of LSTM and dense layers. [16] The model was initialised to be a *Sequential()* model. Three LSTM layers were added with number of units as 64, 128 and 6 respectively. The first two LSTM layers return sequences, the third one does not. This is essential when using TensorFlow and stacking multiple LSTM layers. The activation function was set as *relu* to combat the vanishing gradients problem. The first layer also gets an additional input of *input_shape =(30,1662)*. This refers to 30 frames and 1662 keypoints. Next, two dense layers were added to process the outputs produced by the previous LSTM layers. Each dense has the same activation function (*relu*). The first one is initialised with 64 units, and the second one with 32. These layers help extract high-level features from the sample data. The last layer in the model is

the output layer. Here, the number of units is defined as the shape of the array which defines the five classification categories. The activation function used here is *softmax*. The output is a list of probabilities for each class. The class with the highest listed probability becomes the predicted class.

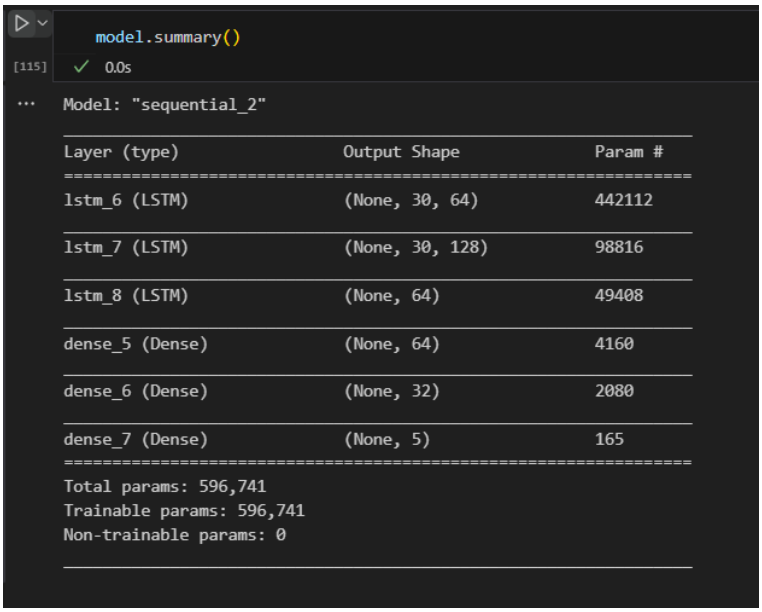

Figure 5.3: Model Summary

The code for real time recognition follows very closely to the code for data collection, hence it will not be expanded upon here.

To facilitate the production of instrument specific sounds we defined separate folders for Piano, Guitar, Violin, Drums and Saxophone. We loaded five audio files of *.wav* format into each folder. Each file is a distinct note produced by a corresponding instrument. The Simpleaudio and Pydub libraries from python are used to play the audio files.

The path for the audio files is constructed in the following way:

1. The recognized class is received from the *sentence* array.
2. The folder path is set to point to the folder for that instrument according to the directory structure defined above.
3. Each sensor is mapped to one audio file. Once a signal is received via the serial

monitor, we can determine what audio file to play according to the sensor(s) activated.

Five constant integer variables (*SENSOR_PIN_*{*i*} where i = 1,2,3,4,5) are defined in the Arduino IDE. Each variable is mapped to one TTP223 touch sensor. In the *setup()* function, the serial communication is initialized with a baud rate of 9600. Each sensor pin is configured as an input pin using the *pinMode()* function. These input pins will be used to read the *HIGH* or *LOW* value from the capacitive touch sensor. If the sensor value for one or more sensors is detected to be *HIGH*, a signal is sent to the python environment via the serial connection established earlier. This signal indicates which specific touch sensor was activated.

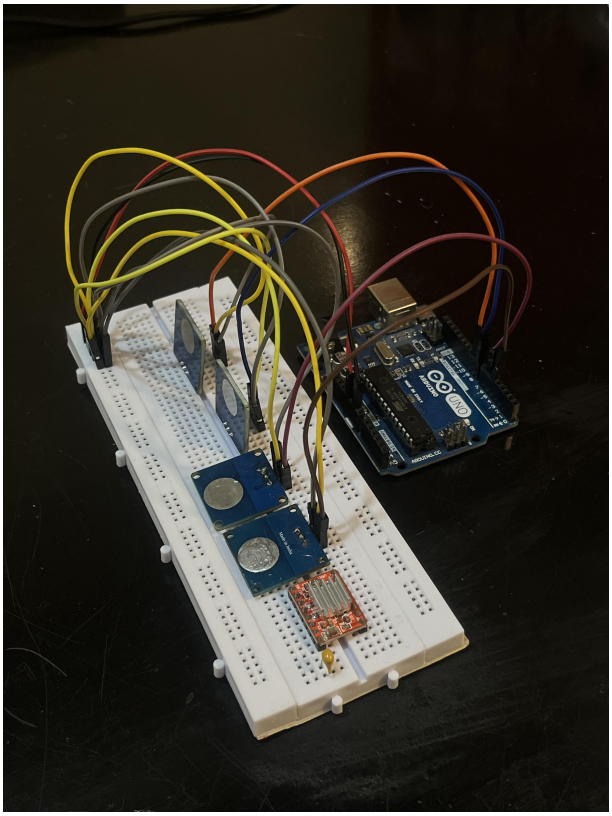

Figure 5.4: TTP223 Base Circuit

The TTP223 Capacitive Touch Sensor can detect the presence of a conductive object by measuring changes in capacitance when a conductive object touches the sensor pad. When touch is detected, an output signal is triggered and the capacitance value becomes HIGH. For the purpose of this project, capacitive sensing techniques

were employed to extend the touch-sensitive area.

The Arduino Uno microcontroller provides an open source platform on which the user can run software programs and control hardware devices. Using the Arduino IDE, one can write and upload code to the board for later execution. In this project, an Arduino board is used to interface touch sensors with a laptop to facilitate a bidirectional transfer of signals.

Results of the Gesture Recognition Model

Two evaluation metrics were used to quantify the performance of the machine learning model - Confusion Matrix and Accuracy Score. The results are given below:

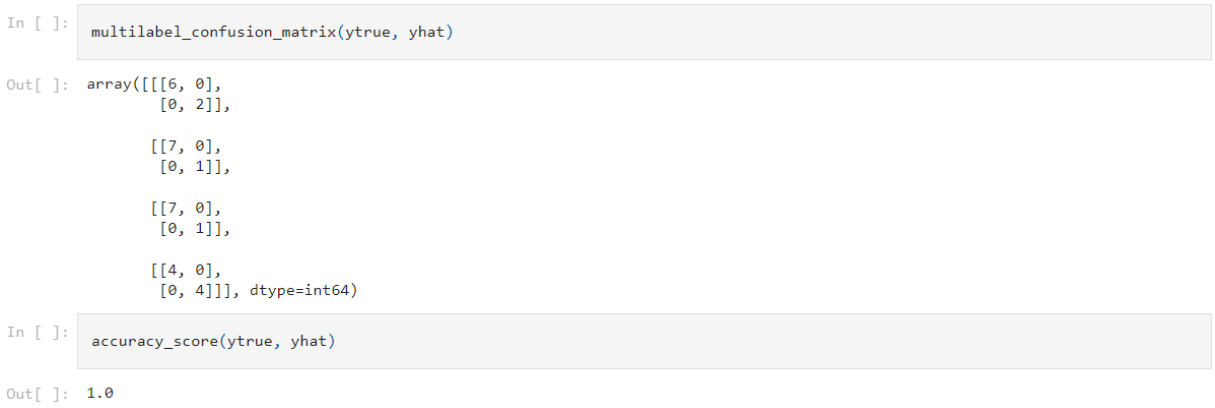

Figure 5.5: Gesture Recognition Model Evaluation

Based on the above data, we can calculate the following evaluation metrics for each class:

1. Class 0:
   (a) Precision = 6 / (6 + 0) = 1.0
   (b) Recall = 6 / (6 + 0) = 1.0
   (c) F1-score = 2 * (1.0 * 1.0) / (1.0 + 1.0) = 1.0
2. Class 1:

(a) Precision = 7 / (7 + 0) = 1.0

(b) Recall = 7 / (7 + 0) = 1.0

(c) F1-score = 2 * (1.0 * 1.0) / (1.0 + 1.0) = 1.0

3. Class 2:

(a) Precision = 7 / (7 + 0) = 1.0

(b) Recall = 7 / (7 + 0) = 1.0

(c) F1-score = 2 * (1.0 * 1.0) / (1.0 + 1.0) = 1.0

4. Class 3:

(a) Precision = 4 / (4 + 0) = 1.0

(b) Recall = 4 / (4 + 0) = 1.0

(c) F1-score = 2 * (1.0 * 1.0) / (1.0 + 1.0) = 1.0

Chapter 6

Discussion

Genetic Algorithms for Product Design

Genetic Algorithms are interesting and efficient method of result optimization. While they may be better suited to problems which can be expressed in mathematical equations or products that can be modelled and simulated automatically by some software, we have seen that we can apply the general approach to address our specific use case of designing a musical instrument.

GAs follow an evolutionary computation paradigm. They use the concept of natural selection to iteratively refine string representations. Our result shows a steady improvement in fitness scores from the first generation to the last. There were some iterations across which scores platued, but the larger theme of converging to the best solution remained intact.

The representation design for product profiles given in the reference paper proved to be effective. We were able to encode each design and variation as a binary string which helped enable a thorough exploration of the population space.

The method chosen to evaluate fitness considers human bias and input in design. By representing them in the form of attribute-wise scores, we were able to identify and prioritse certain attribute values over others.

Gesture Recognition Model

The confusion matrix of the gesture recognition model yields an interesting result. It provides precision and recall information for four classes. However, our model was trained to classify five classes. This suggests that one class may not have been represented well in the test data. Or, one class was not predicted by the model.

There could several factors behind this result. One common reason in such a case is that there is an imbalance in the number of samples in the test data for each class. If one class has a particularly low number of samples, the model may not be able to make accurate predictions. However, in our dataset, we have the same exact number of samples for each class. Another reason for such an outcome could be the presence of a bias towards certain classification categories. This could lead to the model putting predictions of one class over another, thereby resulting in ignorance of a key category. Data quality is not ensured. Since each sample was manually collected, perhaps variations in lighting, scene setting and background noise seeped into the dataset, resulting in noisy or even incomplete data. Such variations in data could lead to inaccurate predictions. This could be another reason for the exclusion of one class in the confusion matrix.

The calculations for Precision, Recall and F1-score indicate that each class depicted in the confusion matrix is perfect in these three dimensions. There were no false positives or false negatives and the model performed well, making accurate predictions in each class.

However, these scores could point to some issues in the underlying training process. Perfect scores may point to overfitting of data. This can lead to poor performance in real-time testing despite seemingly high accuracy scores. In our case, fortunately, the model seemed to perform exceedingly well even on the test data.

```
In [ ]: yhat = model.predict(X_test)

In [ ]: #Extracting predicted classes
        ytrue = np.argmax(y_test, axis=1).tolist()
        yhat = np.argmax(yhat, axis=1).tolist()

In [ ]: ytrue

Out[ ]: [3, 3, 0, 1, 3, 0, 2, 3]

In [ ]: yhat

Out[ ]: [3, 3, 0, 1, 3, 0, 2, 3]
```

Figure 6.1: Test Data Label Prediction

Chapter 7

Conclusion

The primary research objectives of this capstone project were to explore the capabilities of AI in designing a novel musical instrument, assessing its musical sensibility in terms of the physical structure of designs generated, optimizing the design and evaluating the instrument's intuitiveness by adding various functionalities.

Out initial question was "Can AI be creative?". Post experimentation, it feels fitting to expand the question to "Can computers help create?". I believe that the answer to both the questions posed is True. AI can be creative and computers can help create. Not only can computational models provide novel solutions, we can design and develop various methods of input that help mathematical and biological algorithms solve for modern, design problems. We can observe the diversity in images generated by the text-to-image models in the screenshots provided. We have examined and worked with one method of product representation. The current results, even with randomly generated product utilities, show iterative improvement as we progress along each generation.

There is a lot of interesting future work that can be carried in this domain. We could explore more expressive methods of product representation. An automated simulator could be built to help with large scale visualisation of solutions. Specific to

this project, we could administer surveys to determine product utility to a particular subset of the population. Instead of random individuals, we could target musicians specifically for actual industry insights. If solutions can be visualized autonomously with the help of a simulator, we could even come up with a completely new way of evaluating solutions, one that doesn't depend on the user inputs collected. Rather, solves a mathematical equation, as intended for most GA applications.

Bibliography

[1] Scott Reed. Generative adversarial text to image synthesis. arXiv, 2016.

[2] Tao Xu. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. arXiv, 2017.

[3] Roelof Pieters. Creative ai: On the democratisation escalation of creativity. 2016.

[4] Lingchen Chen, Feng Wang, Hui Deng, and Kaifan Ji. A survey on hand gesture recognition. In *2013 International Conference on Computer Sciences and Applications*, pages 313–316. 2013.

[5] Oudah Munir. Hand gesture recognition based on computer vision: A review of techniques. MDPI, 2020.

[6] Kathiravan Natarajan. Hand gesture controlled drones: An open source library. IEEE, 2018.

[7] Phat Nguyen Huu and Tan Phung Ngoc. Hand gesture recognition algorithm using svm and hog model for control of robotic system. In *Journal of Robotics*. 2021.

[8] O Kyon Kyon, JAE-SUNG AN, and Seong-Kwan Hong. Capacitive touch systems with styli for touch sensors: A review. IEEE, 2018.

[9] Mohsin Tiwana, Stephen Redmond, and Nigel Lovell. A review of tactile sensing technologies with applications in biomedical engineering. ResearchGate, 2012.

[10] Luca Turchet. Smart musical instruments: Vision, design principles, and future directionss. IEEE, 2019.

[11] Atau Tanaka. Sensor-based musical instruments and interactive music. 01 2012.

[12] Luisa Gama Caldas and Leslie K Norford. A design optimization tool based on a genetic algorithm. volume 11, pages 173–184. 2002. ACADIA '99.

[13] Victor B. Kreng and Tseng-Pin Lee. Modular product design with grouping genetic algorithm—a case study. volume 46, pages 443–460. 2004.

[14] Shih-Wen Hsiao, Fu-Yuan Chiu, and Shu-Hong Lu. Product-form design model based on genetic algorithms. volume 40, pages 237–246. 2010.

[15] PV (Sundar) Balakrishnan. Genetic algorithms in product design. 1996.

[16] Nicholas Renotte. Sign language detection using action recognition with python—lstm deep learning model. YouTube https://youtu.be/doDUihpj6ro?si=Kwx3Nz0nw-KGxaTZ, 2021.