Generation of new fonts using Neural Cellular Automata

Saahil B Jain
Department of Computer Science and Engineering, *PES University*
Bangalore, India
saahiljain98@gmail.com

Jeevana R Hegde
Department of Computer Science and Engineering, *PES University*
Bangalore, India
jeevanahegde@gmail.com

Srinivas K S
Department of Computer Science and Engineering, *PES University*
Bangalore, India
srinivasks@pes.edu

***Abstract*—Indian languages currently have a limited set of fonts available to choose from. Creation or generation of new fonts is at present, a manual process where designers draw letters in different styles. Further software tools to trace our these manually drawn styles. This can tend to be a tedious and time consuming process. This paper proposes a new method to automate the generation of new fonts for the Kannada scripts while simultaneously ensuring generation of "maatras" and "otaksharas". This would benefit a large audience, especially targeting the local audience for any business or enterprise.**

Keywords—Font Generation, Generative models, Kannada, Neural Cellular Automata.

I. Introduction

Currently font generation is a manual process where designers use tools to draw letters in different ways. No current work exist in automating the generation of fonts for Indian languages like Kannada. This brings the need for our project which aims to automate the generation of new fonts for the Kannada scripts while simultaneously ensuring generation of maatras.

Designers use software drawing tools to draw and create each letter of the alphabet set. Not only is this process hard because the designers have to ensure style consistency among every letter, but it is time consuming as well.

Recent research have attempted to generate fonts for the English language, however they require a large amount of pre-existing fonts to start generating new fonts. This amount of data doesn't exist for Indian languages like Kannada, which have only about 4-5 available fonts.

Hence a different approach is required to generate fonts for Indian languages so that fonts can be generated without the need for pre-existing fonts. This is where our solution comes into picture where we can generate new style consistent fonts without the need for pre-existing fonts.

II. Literature Review

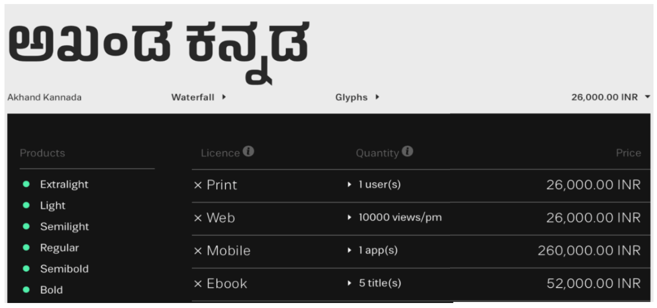
Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organisational editing before formatting. Please note sections A-D below for more information on proofreading, spelling and grammar.

A. Background Study

In this chapter, we present the current knowledge of the area and review substantial findings that help shape, inform and reform our study. Before jumping into the research papers, we looked into why this problem statement is so necessary. That's when we looked into the current time in hours taken for each of it and further the costs from a particular website.

Taking cost into considerations, Indian Type Foundry (ITF) is a company which manually creates customised fonts for various languages. The charges range from 26,000INR - 2,60,000INR for a specific typeface.

B. Architecture



ಅಖಂಡ ಕನ್ನಡ

Akhand Kannada | Waterfall ▸ | Glyphs ▸ | 26,000.00 INR ▾

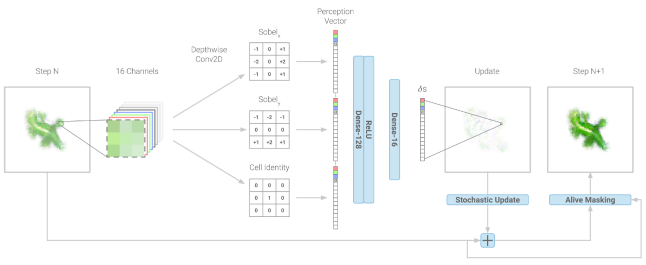
Products	Licence	Quantity	Price
● Extralight ● Light	× Print	▸ 1 user(s)	26,000.00 INR
● Semilight	× Web	▸ 10000 views/pm	26,000.00 INR
● Regular ● Semibold	× Mobile	▸ 1 app(s)	260,000.00 INR
● Bold	× Ebook	▸ 5 title(s)	52,000.00 INR

Most multicellular organisms grow from a single cell. they grow from a single cell that reliably self-assemble into a highly complex organism with precisely the same arrangement of tissues and organs. It does this every time. Morphogenesis is the process of shape development of an organism. Each cell communicates with its neighbours to decide the shape of each body part they belong to.

The Neural Cellular Automata (NCA) is an architecture used to simulate the morphogenesis process with deep neural networks. NCA can grow an entire image starting from just a single pixel. The toughest challenge with neural cellular automata is to design how the cells collective know what to build and most importantly when to stop.

[1]**Article in Online Encyclopedia: Growing, Neural Cellular Automata**, Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, Michael Levin.

This is a paper which successfully built an architecture of



A single update step of the model.

NCA to simulate the morphogenesis process with deep neural networks to generate emojis. The paper focuses on developing a cellular automata, from a single cell to a 2D image its been trained to generate.

Each cell here represents a single pixel of the image and its state is given by three values. one to signify if its alive, and three to get its RGB values. Each cell also has 12 hidden values to represent hidden states which can help in communication between the neighbours. This makes each cell a vector of 16 values

The Cellular automata runs on the 2D grid of cells, i.e. a 2D grid of 16-dimensional vectors, which would be a 3D array of shape [height, width, 16]. The operation applied to each cell must be the same and the result of this operation can only be dependent on a small neighbourhood of the cell. In their case its a 3x3 filter and hence 8 adjacent neighbours. The cell update can be split into 4 steps:

1. Perception - This decides what each cell perceives of its surroundings. This step basically tells what the states of other cells in its neighbourhood are. this is achieved using a pair sobel filter, one with x gradient and the other with y gradient. An additional filter also gives the identity of the cell.

2. Update rule - This is step decides how each cell must update itself. In this case we use a neural network on the output of the previous step to decide by how much the cell should update itself.

3. Stochastic cell update - Once the cell knows how to update itself it still needs to update along with all the other cells to keep the consistency. Hence in a cellular automata we update all cells simultaneously.

4. Alive masking - We want the growth process to starts with a single cell, and only alive cells to take part in the growth. We don't want empty cells to carry any hidden information or participate in the growth. Hence we use a alive filter to get information about the alive cells.

This paper is crucial to our project as we plan to use a similar architecture.We plan to train models using the same approach but add the generative ability to the automata by modifying the Perception step. We plan to do so by changing the sobel filters used for the Perception.

C. Previous Work

Recent research have attempted to generate fonts for the English language, however they require a large amount of pre-existing fonts to start generating new fonts. This amount of data doesn't exist for Indian languages like Kannada, which have only about 4-5 available fonts.

Hence a different approach is required to generate fonts for Indian languages so that fonts can be generated without the need for pre-existing fonts.

However to compare our outcomes and to have a benchmark lets look at 2 prominent papers we came across.

[2]**Deep-fonts**, Eric Bernhardsson.

This paper aimed at generating new fonts for English using simple neural networks. The neural network was trained on a dataset of 56,443 different fonts. This paper didn't generate very unique fonts, however it was one of the first papers to attempt this and was used as benchmark for other similar work.

The project started by cropping and converting all the images of the letters to 64 x 64 bitmaps. Then a simple neural network with 4 hidden layers was trained, with the output being 4096 (64x64) which is essentially the output image.

Though the network was able to generate the fonts, they weren't very different from the training data.

.

[3]**GlyphGAN**, Hideaki Hayashia, Kohtaro Abea, Seiichi Uchidaa.

This is a paper that successfully developed a model for style-consistent font generation for English. It is based on generative adversarial networks (GANs). GANs are now a very common framework for learning generative models using a system of two neural networks competing with each other. One neural network is used to generate images from random input vectors, and the other neural network is used to discriminate between the generated and real image. this was the generator model gets better over time and the generated images are closer to the actual image.

In GlyphGAN, the input vector of the generator neural network consists of two parts: character class vector and style vector. The character class vector is is a one-hot encoded vector that is associated with the character class of each sample image during training. The style vector is a random vector used for generation purpose.

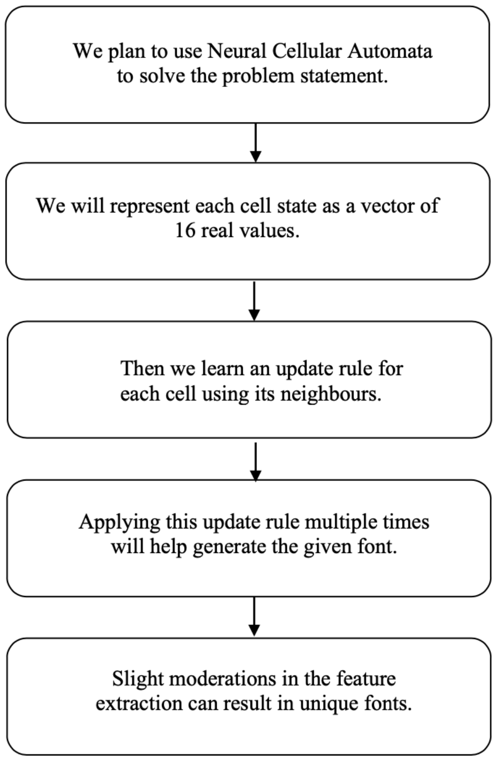
GlyphGANs was trained on a dataset of 6,561 different fonts and results have showed that fonts generated by GlyphGAN are style consistent and diverse. This project provides some of the most unique fonts and is currently the best in terms of font generation. However its data requirement is large and hence is inapplicable for generation of fonts for Indian Languages like Kannada.

COMPARISON OF THE DIFFERENT METHODS:

Method	Recognition accuracy [%] (Legibility)	Cs (Style consistency)	Cd (Diversity)
Deep-fonts	*72.51*	*0.47*	*0.33*
GlyphGAN	*83.90*	*0.46*	*0.61*

III. Methodology and Implementation

We plan in breaking down the complex problem into simpler steps to make it a seamless path.



We plan to use Neural Cellular Automata to solve the problem statement.

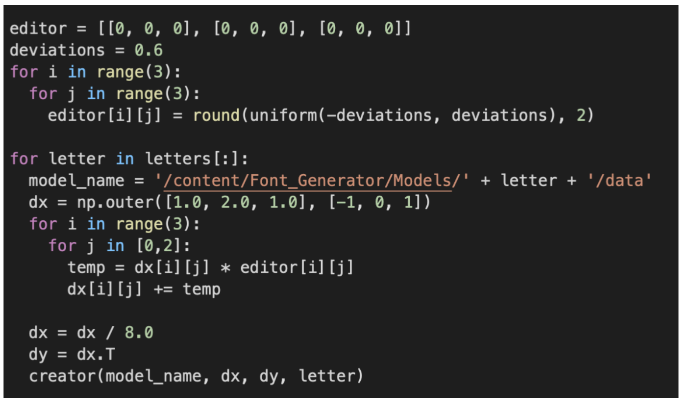
We will represent each cell state as a vector of 16 real values.

Then we learn an update rule for each cell using its neighbours.

Applying this update rule multiple times will help generate the given font.

Slight moderations in the feature extraction can result in unique fonts.

Once the models are trained for each alphabet of the language we need to update the Sobel filters used to extract the neighbourhood information for each cell. We created methods to set the filter to any filter provided as arguments. We generate random filters using the code below.



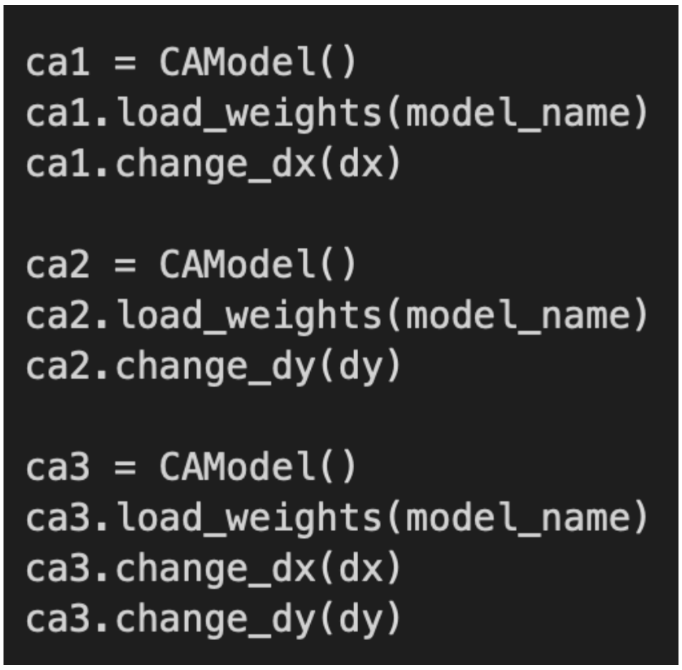
```
editor = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
deviations = 0.6
for i in range(3):
  for j in range(3):
    editor[i][j] = round(uniform(-deviations, deviations), 2)

for letter in letters[:]:
  model_name = '/content/Font_Generator/Models/' + letter + '/data'
  dx = np.outer([1.0, 2.0, 1.0], [-1, 0, 1])
  for i in range(3):
    for j in [0,2]:
      temp = dx[i][j] * editor[i][j]
      dx[i][j] += temp

  dx = dx / 8.0
  dy = dx.T
  creator(model_name, dx, dy, letter)
```

The values are not completely random because that makes the characters illegible, however each value is increased or decreased by a certain fraction of its actual value. For every random filter generated we create 3 models :

1. Where we update the x filter
2. Where we update the y filter
3. Where we update both the x filter and y



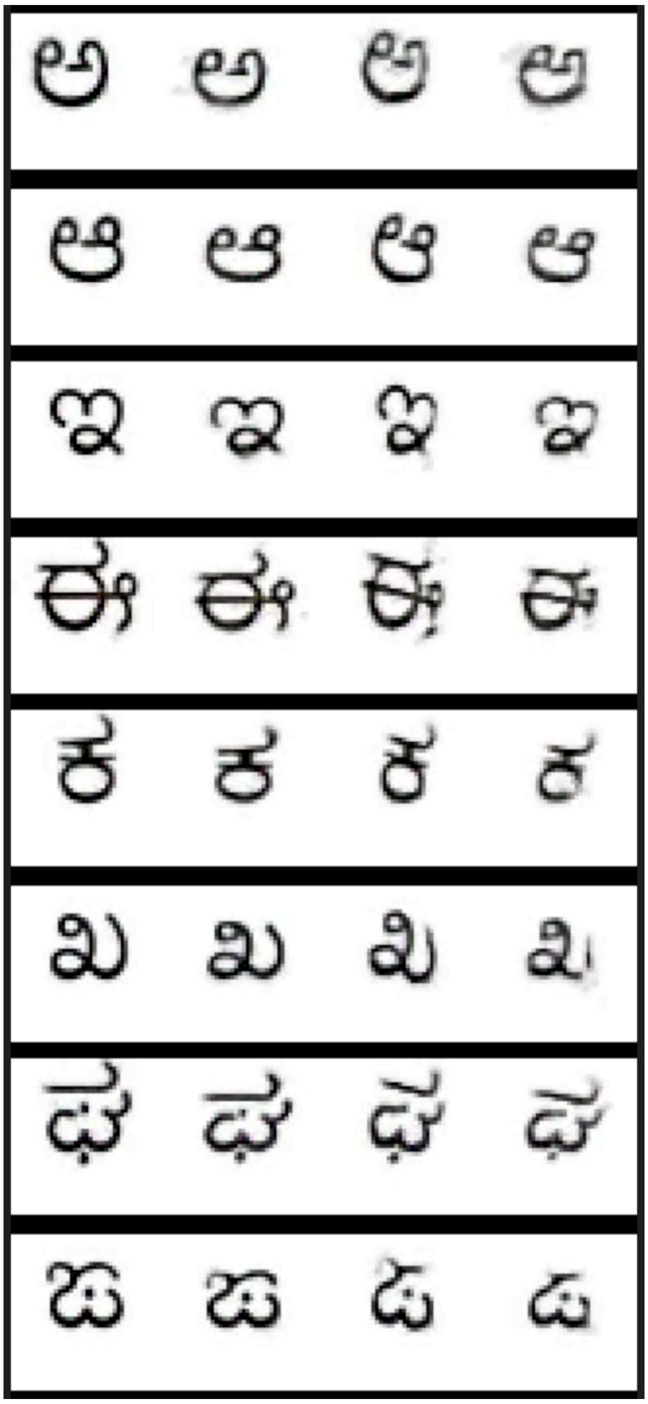
```
ca1 = CAModel()
ca1.load_weights(model_name)
ca1.change_dx(dx)

ca2 = CAModel()
ca2.load_weights(model_name)
ca2.change_dy(dy)

ca3 = CAModel()
ca3.load_weights(model_name)
ca3.change_dx(dx)
ca3.change_dy(dy)
```

IV. Results and Discussion

This approach is ideal for generation of images when data availability is low. In the case of font generation, not only was this method useful to generate unique fonts but also ensured style consistency as you can see in these images below:



V. Conclusion

We have successfully generated style-consistent fonts for Kannada script.

Started off with deep diving into the literature survey where we initially did research on how the work on this problem statement is currently being done. Moving on, we found a theoretical automated idea and read up on its implementation. Further on, we planned on implementing our Neural cellular automata to help solve the problem.

After the literature survey, we moved on to obtaining the required dataset. We coded out the downloading of all the Kannada alphabets and numbers, its maatras as well as ottaksharas. This made the process more convenient and efficient.

Then came the actual building the neural network. Done in a three step process:

1. Developing hidden channel and feature extraction. We successfully built a customised neural network
and worked on updating the filters.

2. Built the basic model of Neural Cellular automata for a single letter.

3. Built the basic model of Neural Cellular automata for all letters.

We successfully built a customised neural network and worked on updating the filters. We generate random filters. However, the values are not completely random because that makes the characters illegible, however each value is increased or decreased by a certain fraction of its actual value.

We thus achieved our goal where we learnt and generated the cellular automata for the Kannada language and changed the filters to generate new fonts. This was done with the required style–consistency..

VI. References

1. Mordvintsev, et al., "Growing Neural Cellular Automata," Distill, 2020. URL https://distill.pub/2020/growing-ca/
2. E. Bernhardsson, Analyzing 50k fonts using deep neural networks. URL https://erikbern.com/2016/01/21/analyzing-50k-fonts-using-deep-neural-networks.html
3. H. Hayashia, K. Abea, S. Uchida, “GlyphGAN: Style-Consistent Font Generation Based on Generative Adversarial Networks,” arXiv:1905.12502v2, 2019
4. K. Janssens, “Network Cellular Automata-A Development for the Future”. URL https://www.sciencedirect.com/topics/computerscience/cellularautomata
5. G Andrews, “Cellular Automata and applications”. URL https://www.whitman.edu/Documents/Academics/Mathematics/andrewgw.pdf