Style-consistent Kannada Font Generation

by Sahhi B Jain

Submission date: 17-Apr-2021 11:41AM (UTC+0530)
Submission ID: 1561669943
File name: Saahil_B_Jain.pdf (2.39M)
Word count: 7182
Character count: 36634

Dissertation on

"Style-consistent Kannada Font Generation"

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

UE17CS490B – Capstone Project Phase - 2

Submitted by:

Saahil B Jain	**PES1201700241**
Jeevana R Hegde	**PES1201700633**

Under the guidance of

Prof. K. S. Srinivas
Associate Professor
PES University

January - May 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

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 process and our project goal is mainly 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.

Taking a step ahead after generation of Kannada script is completed, we tend to inculcate inspiration from existing English fonts and create similar Kannada fonts. This is essentially Style transfer. This shall provide us with multiple fonts for the Kannada scripts just as it is in English.

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

Indian languages currently have a limited set of fonts available and Generation of new fonts 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.

Currently fonts are generated manually by designers. These 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.

Moving this to a step ahead after generation of Kannada script is completed, we tend to inculcate inspiration from existing English fonts and create similar Kannada fonts. This is essentially Style transfer. This shall provide us with multiple fonts for the Kannada scripts just as it is in English.

Our project could finally be used by a website which would offer users multiple auto- generated fonts to choose from for the Kannada script. Each option would be available separately where users can browse through all options and choose and download the ones they like most.

CHAPTER 2

PROBLEM STATEMENT

The project is to automate the generation of new fonts for the Kannada scripts while simultaneously ensuring generation of maatras.

Indian languages currently have a limited set of fonts available and Generation of new fonts 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.

Creation or generation of new fonts is at present, a manual process where designers draw letters in different styles. These designers often 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. Takes about 15 hours to 64 hours depending on the innovation and complexity that goes into it.

Moving this to a step ahead after generation of Kannada script is completed, we tend to inculcate inspiration from existing English fonts and create similar Kannada fonts. This is essentially Style transfer. This shall provide us with multiple fonts for the Kannada scripts just as it is in English.

This needs to be done in order to provide new fonts without the hassle of redrawing new styles and reduces the number of people behind the application. No new artists/ calligraphy experts are required for the process and the entire Style transfer is automatically established.

Style transfer will require a new set of English charactered datasets which need to be cleaned and labelled correctly to be obtained to create a model. This model further needs to be trained and the styles needs to be extracted and put on to the set of Kannad characters. The generation in both cases is done via Cellular Automata.

Recent researches 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.

We notice that logo fonts can make or break your logo design. Eg: Disney, Netflix, The Times of India, etc.

With continuous advancements in web-technologies and the need to deliver a great experience to non- English speaking users is creating a greater demand for high-quality, multi-script typefaces.

We particularly looked at use cases of this problem statement:

1. ***Television channels*** – All langueages require their own new channels no matter how much social media takes over. Hence a crisp, bold and clear font to depict the name of the channel with the logo on the right top corner.
2. ***Newspapers, Magazines and Books*** - A large heading on the front cover page and consecutively on every other page is mandatory to keep users engaged and ensure they are comfortable with the style and able to understand for all age groups.
3. ***Movie names*** – Movies are a huge customer for this applications as they always require a catchy unique, bold font including the tagline for the movie posters, etc.
4. ***Company/ Brand names*** – essentially any company/ brand that have local Kannada speaking/ reading audience must cater to them in their language by providing them with the name to remember of their company. This increases target audience aspect as the memory retains longer in their customers if they make sense of the unique font, name and meaning attached to it. Value of brand increases in the process. Generic Kannada brand names to attract local audience

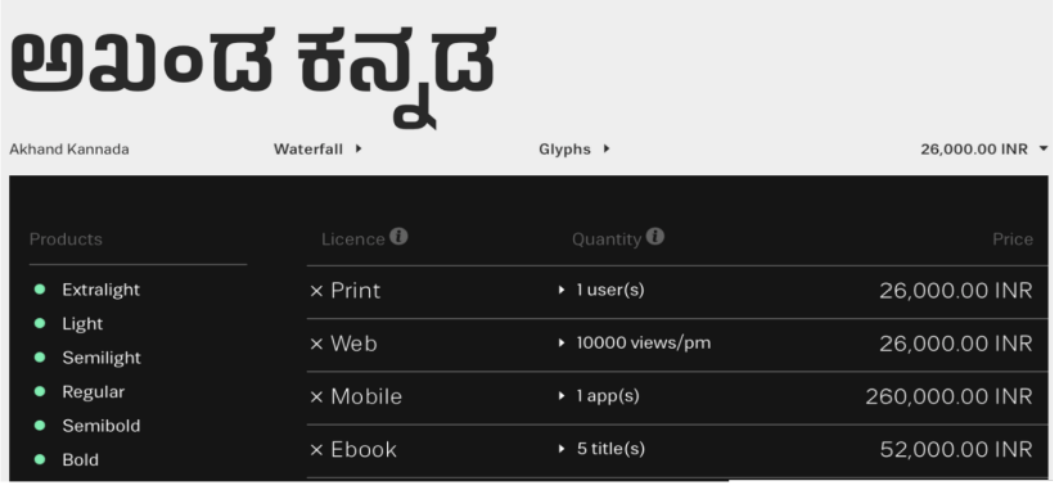
CHAPTER 3

LITERATURE SURVEY

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 customized fonts for various languages. The charges range from 26,000INR - 2,60,000INR for a specific typeface.



Having said this, 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 neighbors 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 knows what to build and most importantly when to stop.

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.

3.1 Growing Neural Cellular Automata [1]

Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, Michael Levin

This is a paper which successfully built an architecture of 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 neighbors. 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 neighborhood of the cell. In their case its a 3x3 filter and hence 8 adjecent neighbors. 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 neighborhood are. this is acheived 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.

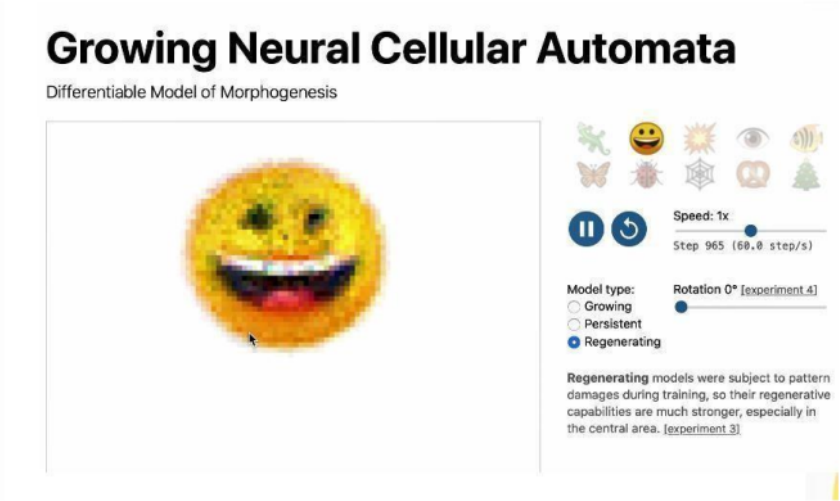
2. Update rule - This is step decides how each cell must update itself. In this case we use a neural network on the output of the previous step to decide by how much the cell should update itself.

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.

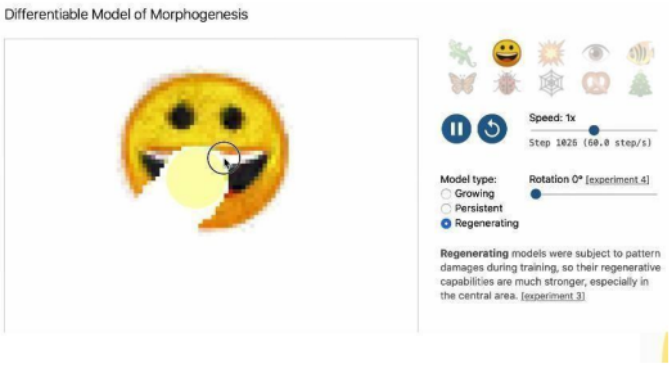
4. Alive masking - We want the growth process to starts with a single cell, and only alive cells to take part in the growth. We don't want empty cells to carry any hidden information or participate in the growth. Hence we use a alive filter to get information about the alive cells.

Simple regeneration can be seen as follows for an emoji:

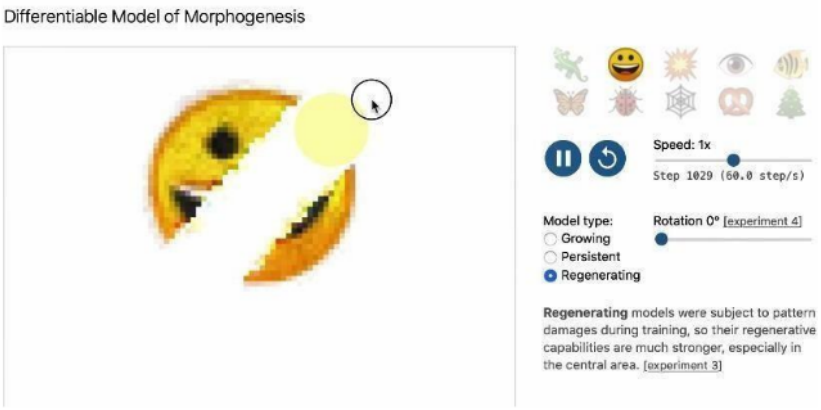
1. **Original**



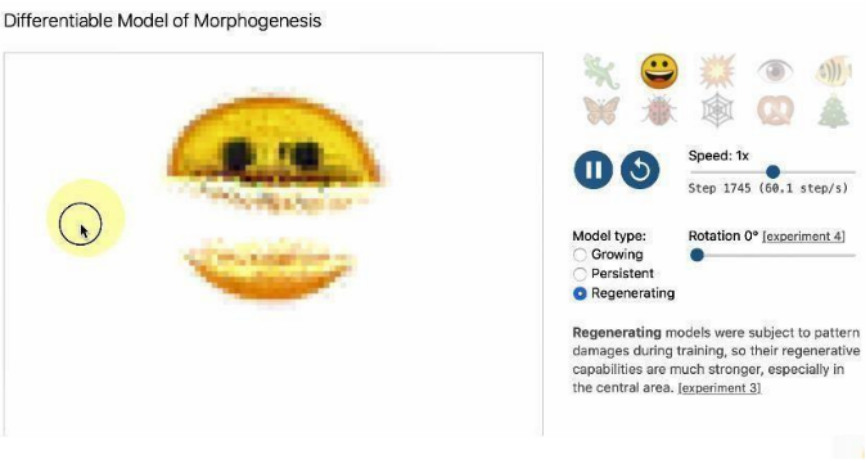
2. **Erasing some parts**



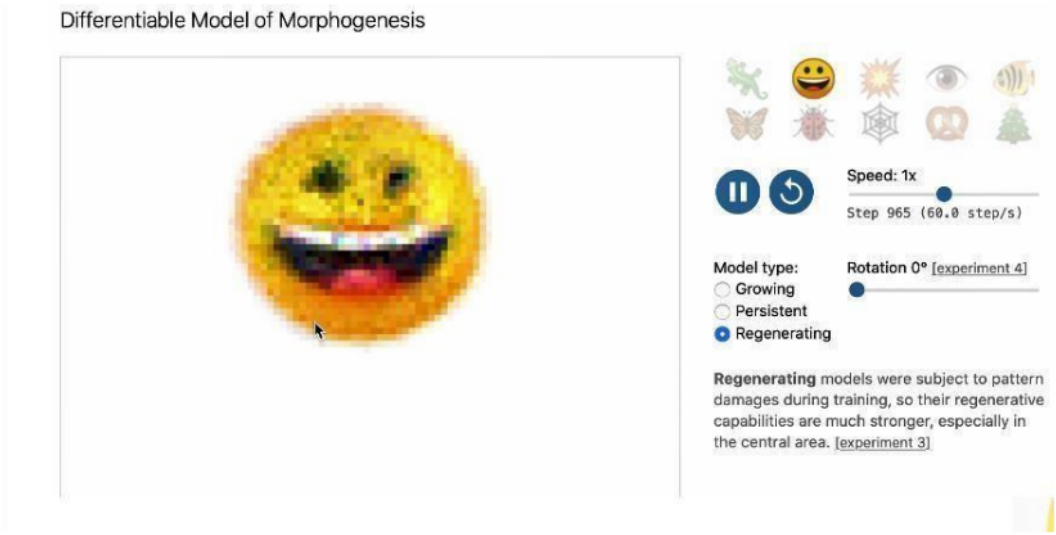
3. Erasing the middle



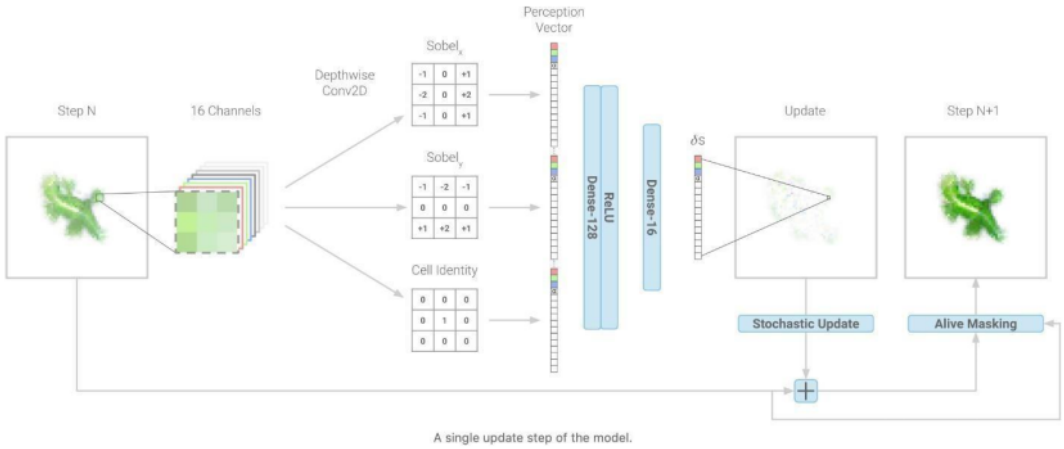
4. Erasing the middle once again



5. Regeneration of the same emoji even after being erased



The architecture is as follows:



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.

Strengths:

1. Good emoji generation from just a single point.
2. Regenerative abilities is present i.e. can complete the emoji if part of it is missing.
3. Can generate fairly complex emojis.
4. Models are quite small.

Weaknesses:

1. No new generation as such. Just builds a model to create the exact input.
2. Every emoji requires a separate model.

3.2 Deep-fonts [2]

Eric Bernhardsson

This paper planned to generate 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. For example seeing the image below, for each pair, the real character is on the

left, the model output on the right. as you can see the fonts are almost the same.

Strengths:

1. Simple architecture that can easily be replicated and used.

2. Style consistency is reasonable.

Weaknesses:

1. There is an extremely high data dependency.

2. Diversity is quite low.

3.3 GlyphGAN [3]

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 currenty 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.

In the Image below the upper rows are output and the lower rows are the closest training class.

The goal of the paper is for new font generation using neural networks.

The data required to do the same is about 56443 different fonts for training your model and further generating new fonts. Does the required job of new font generation with a low diversity.

Strengths:

1. Better legibility compared to deep-fonts.

2. Style consistency is the same.

3. More diversity is available in the fonts compared to deep-fonts.

Weaknesses:

1. High data dependency is present, which is unavailable for Indian languages like Kannada.

COMPARISON OF THE DIFFERENT MOTHODS:

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

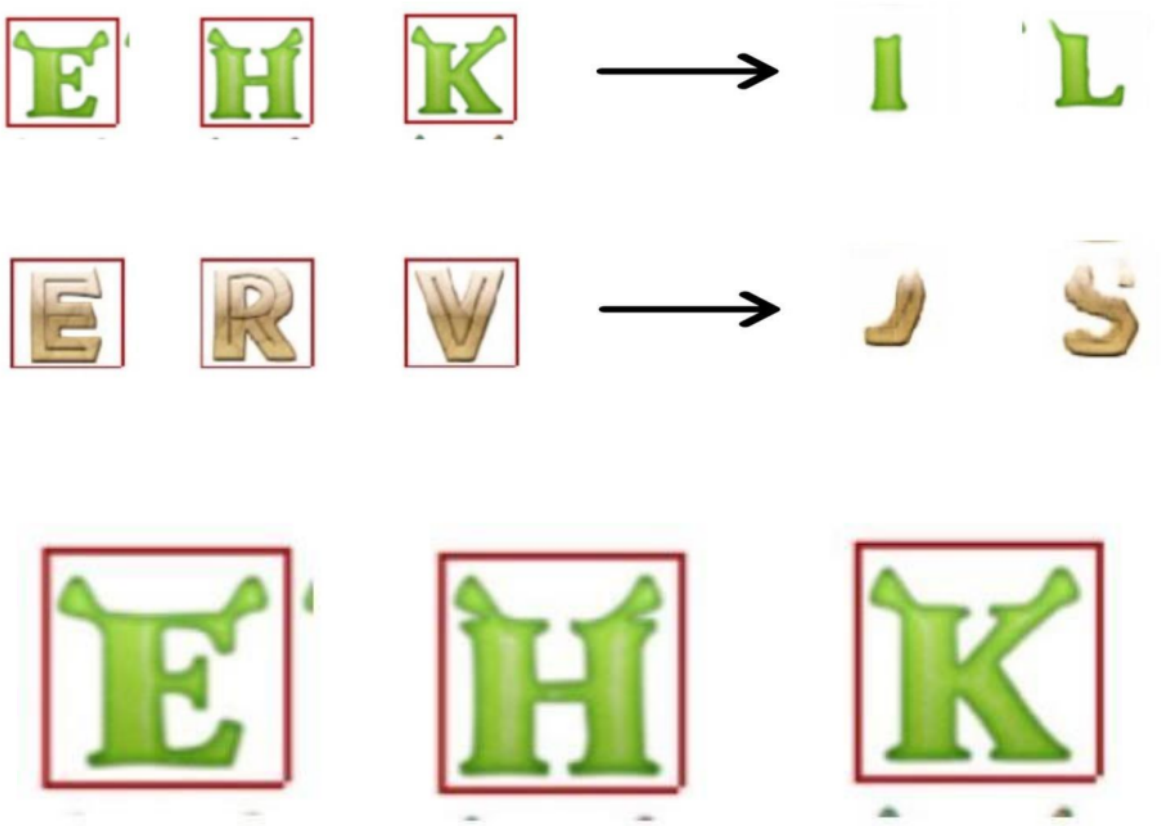
3.4 Multi-Content GAN for Few-Shot Font Style Transfer [4]

Samaneh Azadi, Matthew Fisher, Vladimir Kim, Zhaowen Wang, Eli Shechtman, Trevor Darrell

The paper proposes an end-to-end stacked conditional GAN model to acheive style transfer for fonts. The project consist of two Machine learning models, first a stacked cGAN architecture to predict the coarse glyph shapes, and next an ornamentation network to predict color and texture of the final glyphs. The two models are trained together and specialized for

each font.

For each font the models are trained on a few letters and then the style of these few alphabets is then applied to all the other letters of the alphabet. The collected dataset for this approach includes 10,000 different fonts styles of the English alphabets, which include fonts from movie titles and and gradient fonts. As per our observations the style transfer achieved using this method was the best of all the other work in this area, however the accuracy of style transfer is good only for letters very similar to the letters in the training set.



Strengths:

1.High data dependency is present, which is unavailable for Indian languages like Kannada.

Weaknesses:

1. Style transfer is good only for letters similar to the training data.
2. Works only for letters of same language.
3. Model can't be applied to languages like Kannada as the letters itself vary a lot.

3.5 Typeface Completion with Generative Adversarial Networks [5]

SYonggyu Park, Junhyun Lee, Yookyung Koh, Inyeop Lee, Jinhyuk Lee, Jaewoo Kang

The paper aims to build a model that takes the image of one character of a font as input and generates all the other characters of the alphabet in the same font of the input character.

The paper proposes Typeface Completion Network, which TCN consists of two encoders, a generator and a discriminator. the two encoders, typeface encoder and content encoder, each return a vector that combines the different kinds of information. The generator receives input and target character labels along with the two vectors from the encoders and generates the final output image. The discriminator determines if the generated image is real and this is repeated until the generated images are of desirable quality.

The concept is applied to fonts of English as well as Chinese and achieves style transfer of reasonably good quality. The English dataset consists of 907 different fonts of the uppercase alphabets. The Chinese dataset consists of 150 different fonts of the 1000 most used characters.

Strengths:

1. Can generate same fonts for all letters of the language with just one input letter.

Weaknesses:

1. Hight data dependency.
2. Style learning from input letter cannot be extrapolated to letters outside the training set.

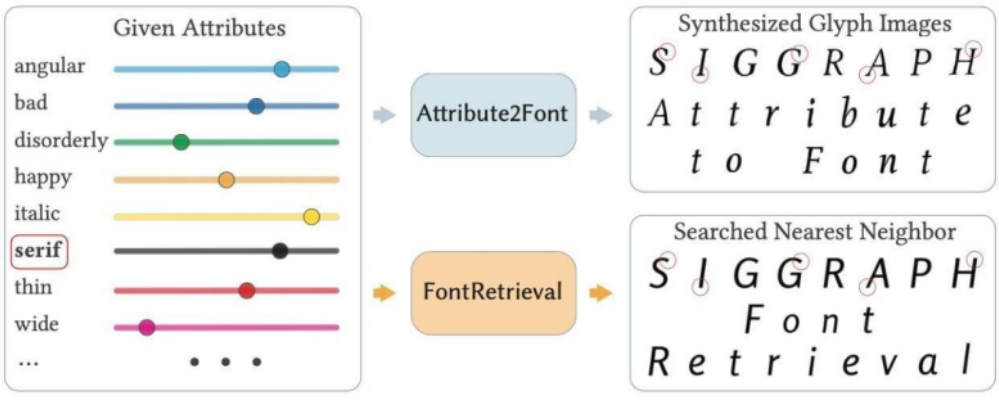
3.6 Attribute2Font: Creating Fonts You Want from Attributes [6]

YIZHI WANG, China YUE GAO, China ZHOUHUI LIAN Uchidaa

The paper aims to generate glyph images according to the user-specified font attributes and their values. There are 37 different kind font attributes some of which are - Strong, Angular, Bad, Disorderly, Happy, Italic, Serif, Thin and Wide. The value of these attributes are based on features of the glyph for example - strong defines how bold the letters are while artistic defines how handwritten the letters look.
Once trained on a large dataset the model can generate fonts for combinations of these attributes that it was never trained on.

The concept is unique by the fact that instead of boolean values for each attribute, here we use continuous values for each attribute which helps identify the magnitude of each attribute we need in the font.

The dataset consists of 148 attribute-labelled fonts and 968 unlabelled fonts.
Although the paper produces very good results, its high data dependency on fonts along with labels is a strong limitation.



Strengths:

1. Can generate style consistent fonts for an incredibly large combination of attribute values.
2. Can focus more on specific attributes and less on other attributes.
3. Solution would work for any language If the abundant data is available.

Weaknesses:

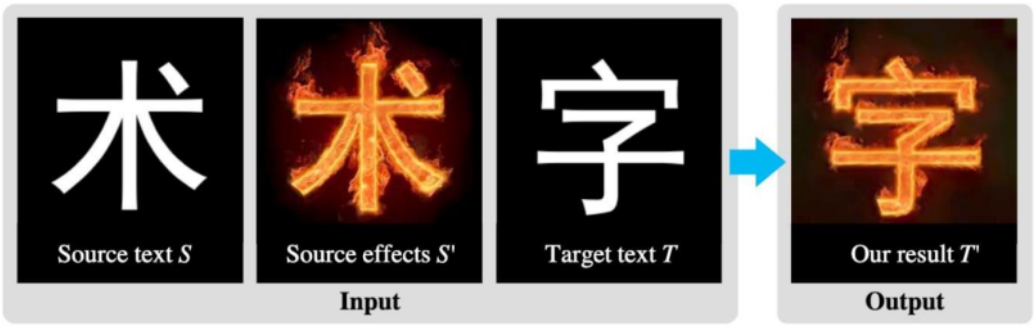
1. Large amounts of labelled data needed.
2. Can't work for Indian languages like Kannada as fonts itself are limited.

3.7 Awesome Typography: Statistics-Based Text Effects Transfer [7]

Shuai Yang, Jiaying Liu, Zhouhui Lian and Zongming Guo

This paper attempts to achieve transfer of special-effects for fonts. The special effects include texture synthesis, shadows, flames, multicolored neon gradients.
To achieve the goal the model takes 3 images as input, the source raw text image, the source styled text image and the target raw text image. the pixels are divided into 16 classes based on their distance from the skeletal input image and based on analytics the effect is applied on the target image. This analysis consists of 3 parts

Appearance Term - To preserve coarse grained texture structures and details.
Distribution Term - To ensures that the sub-effects in the target image and the source example are similarly distributed.
Psycho-Visual Term - To prevent over-repetitiveness of certain source patches.
together these three ensure a well balanced special effect transfer



(a) Target text (b) *flame* (c) *lava* (d) *rust* (e) *drop* (f) *pop* (g) *blink*

Strengths:

1. Special-effect transfer can be achieved to a very high accuracy.
2. Since machine learning is not required the genenration process is quick and new special effects can be accomodated almost instantly.

Weaknesses:

1. Cannot generate any new fonts or special effect, can just transfer from source to target.
2. doesn't work on changing the structure of the input letter, but works on generating the structure and colours of the style input

CHAPTER 4

DATA

This chapter serves to describe the data under consideration. Understanding the way all the data work is vital in the process of creating a good solution to the problem at hand.

4.1 Overview

The novelty of our project lies not only in the fact that automation of font generation is new for Indian languages like Kannada, but also the use of technologies like Neural Cellular Automata for generation problems. Neural Cellular Automata has never been used for generation problems, Our project is the first use case of Neural Cellular Automata as a generative model.

Hence this requires us to have a clean dataset in Kannada script called Glyphs and English data set further on to extend the Style transfer from them. This can be used as a proof of concept or POC for other Indian languages like Telugu.

When we talk about POC it can mean both creation or generation of a different language as well as taking inspiration from a different language. Further, a combination of languages can be used to provide inspiration. However only a single language output can be obtained towards the end.

In our case, only the English language scripts have been used to derive inspiration to our Kannada language scripts. This includes all sorts of characters as we will discuss further on in our sub sections.

4.2 Kannada dataset

Kannada language scripts were originally taken from the internet and downloaded. This was obtained from the Indian Type Foundry which is a company which provides various Inidan language scripts. This includes all sorts of characters for Hindi, Telugu, Malyalam, Marathi, etc.

The alphabets and numbers/ digits are called “glyphs” and they are present in multiple duplications, repetitions, missing data, as well as a bunch of errors. Hence is needed to be cleaned and reorganized. Though a web scraper was used, these errors were present and had to be rectified.

This was then cleaned to remove all the errors such as duplications, repetitions, missing data, etc and labelled as per their lettering chronological order and increasing set of digits and provided to a code. This code essentially put them into different folders once the images were created and further organized very effectively for mapping that would be required further on.

4.3 English dataset

Similar to the Kannada language scripts, English language scripts were originally taken from the internet and downloaded. This was obtained from the Indian Type Foundry which is a company which provides various Indian language scripts. This includes all sorts of characters for Hindi, Telugu, Malyalam, Marathi and many more.

However in this case, we had plenty of websites to choose from to obtain the dataset as English is widely used relative to Kannada or any Indian languages. This itself speaks for the level of importance given to the respective languages

The novelty of our project lies not only in the fact that automation of font generation is new for Indian languages like Kannada, but also the use of technologies like Neural Cellular Automata for generation problems. Neural Cellular Automata has never been used for generation problems, Our project is the first use case of Neural Cellular Automata as a generative model.
Neural Cellular Automata has often been used for recreational problems, but never for generation problems. Our project is the first use case of Neural Cellular Automata as a generative model. Our project is also unique because it is able to generate new fonts without any data requirements.

4.4 Combined feature of the two

To automate the generation of fonts, we had three options available. These are as follows:

- Neural Networks:
  The first step is to create a “font vector” that is a vector in latent space that “defines” a certain font. That way we embed all fonts in a space where similar fonts

  have similar vectors. Then a neural network is trained to generate the fonts from these “font vectors”. Any change in the “font vector” would then result in new fonts.
  Pros:
  - Simple and easy to build an architecture.
  - Enough data is present. Hence, slight modifications lead to the generation of

significantly large number of fonts with high legibility.
- Style consistency is reasonable.

Cons:
- There is an extremely high data dependency.
- Diversity is quite low.

- GANs:
We can train GANs which can be trained over a large dataset of fonts. Once trained the GAN could easily generate better fonts than the neural network.
Pros:
  - Better legibility compared to deep-fonts.
  - Style consistency is the same.
  - More diversity in available in the fonts compared to neural network.

Cons:
  - High data dependency is present, which is unavailable for Indian languages like Kannada.

- Neural Cellular Automata:
We first train a simple NCA model for each letter and separate model for each letter with different maatras as well. Once all models are created we only need to change the filter used to perceive the neighbouring cells. With slightly different views of the neighbours, the model generates completely different fonts.

Pros:
  - Data requirement is very low. Just a single font is required to generate significantly different fonts.
  - Style consistency is accurate.
  - Models are very small.

Cons:
  - Every letter requires a different model.
  - Different model required for every letter with each maatra.

Having read through all our options we chose the Neural Cellular Automata as this is the only method which can generate fonts without the requirement for additions data. Which is an essential requirement when we try to generate fonts for Indian languages like Kannada which have a very small amount of available fonts.

Performance constraints

To generate fonts we need to create separate models for each letter and separate model for each letter with different maatras as well. This makes generation of base models challenging as well as time consuming. Processing power required is considerably low. But increase in image size increases model size exponentially. Maintainability

Constraints

To generate fonts we need to create separate models for each letter and separate model for each letter with different maatras as well. This makes generation of base models challenging as well as time consuming.

Processing power required is considerably low. But increase in image size increases model size exponentially.

4.5 Classification Systems

Kannada alphabet

Vowels and vowel diacritics with ka

ಅ	ಆ	ಇ	ಈ	ಉ	ಊ	ಋ	ೠ
ಕ	ಕಾ	ಕಿ	ಕೀ	ಕು	ಕೂ	ಕೃ	ಕೄ
a	ā	i	ī	u	ū	ṛ	ṝ
[a]	[aː]	[i]	[iː]	[u]	[uː]	[rɨ/ru]	[rɨː/ruː]
ಎ	ಏ	ಐ	ಒ	ಓ	ಔ	ಅಂ	ಅಃ
ಕೆ	ಕೇ	ಕೈ	ಕೊ	ಕೋ	ಕೌ	ಕಂ	ಕಃ
e	ē	ai	o	ō	au	aṃ	aḥ
[e]	[eː]	[ai]	[o]	[oː]	[au]	[aŋ]	[ah]

Consonants

ಕ	ಖ	ಗ	ಘ	ಙ	ಚ	ಛ	ಜ	ಝ	ಞ
ka	kha	ga	gha	ṅa	ca	cha	ja	jha	ña
[ka]	[kʰa]	[ga]	[gʱa]	[ŋa]	[ʧa]	[ʧʰa]	[ʤa]	[ʤʱa]	[ɲa]
ಟ	ಠ	ಡ	ಢ	ಣ	ತ	ಥ	ದ	ಧ	ನ
ṭa	ṭha	ḍa	ḍha	ṇa	ta	tha	da	dha	na
[ʈa]	[ʈʰa]	[ɖa]	[ɖʱa]	[ɳa]	[ta]	[tʰa]	[da]	[dʱa]	[na]
ಪ	ಫ	ಬ	ಭ	ಮ	ಯ	ರ	ಲ	ವ	
pa	pha	ba	bha	ma	ya	ra	la	va	
[pa]	[pʰa]	[ba]	[bʱa]	[ma]	[ja]	[ra]	[la]	[va]	
ಶ	ಷ	ಸ	ಹ	ಳ	ಕ್ಷ	ಜ್ಞ			
śa	ṣa	sa	ha	ḷa	kśa	jna			
[ɕa]	[ʂa]	[sa]	[ha]	[ɭa]	[kʂa]	[ʤɲa]			

Dependencies

One set of alphabets as well as digits with all maatras.

Assumptions

Compute resource powerful enough to generate the base models in reasonable amount of time.

Interoperability

Once the fonts are generated as images and converted to fonts usable by the system, they can be downloaded on the system on which the application requires them. Once downloaded they can be used on literally any system as regular fonts are used.

Portability

Our method can easily be used to generate new fonts for other languages like Telugu or Hindi. It can also be used for many generative tasks as well.
For example: this method can also be used for data augmentation purposes for images which would be very helpful in case of low clarity images.

Legacy to modernization

Currently fonts are generated manually by designers. These 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.

4.6 Reusability

Not only can the given method be used to generate new fonts for other languages like Telugu, but can also be used for many generative tasks as well.
This method can also be used for data augmentation purposes for images which would be very helpful in case of low clarity images.

Application compatibility
Once the fonts are generated as images they could easily be converted as fonts and used along with literally any application that requires these fonts.
They would however need to be converted to fonts usable by the system and downloaded on the system on which the application requires them.

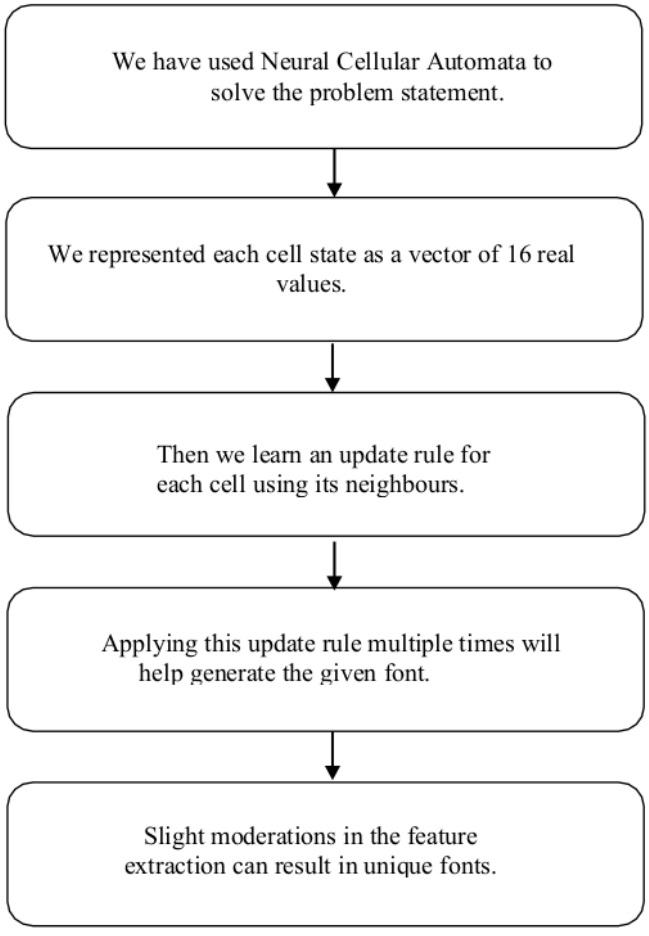
CHAPTER 5

METHODOLOGY

5.1 Overview

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

Our project started off with us obtaining the required datasets which are spoken about broadly in the previous sections. Moving on we cleaned the datasets and labelled them as per requirements to keep them more organized and easily available to train the model.



Then we worked on building a model using Neural style transfer.

5.2 Module Name

Neural style transfer

5.3 Module Architecture

5.4 Approach

- There are three inputs provided as shown. These include the style image, target character and the generated image.
- The three inputs are passed to the Neural network iteratively and the loss calculated is back propagated.
- The output of each iteration is passed as input to the next iteration.
- The process is repeated until the generated image is satisfactory.

5.5 Losses

When we try to solve any sort of supervised learning task, the most important part becomes choosing the right loss function. The initial loss function we used provided us with limited results that were not usable. This is because the style loss calculated was appropriate for images with colours where the loss functions could recognise colours and their shapes and sizes. However, for fonts that are all black and having completely different shapes, that style loss seems inappropriate.

Hence we were required to develop our own loss function. We wrote a script to help us evaluate the loss function. This script took two fonts styles as input and took all combinations of letters pairs of the first font and second font as well as all combination of letter pairs of the second font with itself.

Loss was calculated for each pair and the results were compared. Ideally the style losses for combinations of letters pairs of the first font and second font should be really high and style loss of letter pairs of the second font with itself should be minimal.

Having created our own loss function we could see the effectiveness of our new loss function clearly.
Earlier the ratio of the average style loss of the letters of same fonts to the average style loss of the letters of different fonts was about 8:10
With our new loss function, ratio of the average style loss of the letters of same fonts to the average style loss of the letters of different fonts came to be about 4:10
can be seen in the reference images below.

Content loss

$$J_{\text{content}}(C, G) = \frac{1}{2} \left\| a^{[\ell](C)} - a^{[\ell](G)} \right\|^2$$

Original/Content Generated

Style loss

$$G_{ij}^{l} = \sum_{k} g_{ik}^{l} g_{jk}^{l}$$

Gram Matrix for Style and Generated Image

$$S_{ij}^{l} = \sum_{k} s_{ik}^{l} s_{jk}^{l}$$

$$\mathcal{L}_{\text{style}}(g, s) = \sum_{i,j} \left(G_{ij}^{l} - S_{ij}^{l} \right)^2$$

Total loss

$$\mathcal{L}_{\text{total}}(G) = \alpha \mathcal{L}_{\text{content}}(C, G) + \beta \mathcal{L}_{\text{style}}(S, G)$$

Old loss function

```
Different Style :
Minimum Style Loss    :  483713.25
Maximum Style Loss    :  75419184.0
Total Style Loss      :  63178097636.0
Average Style Loss    :  16435509.270551508

Same Style :
Minimum Style Loss    :  139304.84375
Maximum Style Loss    :  55072780.0
Total Style Loss      :  49482394661.59375
Average Style Loss    :  13083658.027920082
```

New loss function

```
Different Style :
Minimum Style Loss    :  4018944.75
Maximum Style Loss    :  110331304.0
Total Style Loss      :  143954384466.5
Average Style Loss    :  37449111.463709675

Same Style :
Minimum Style Loss    :  618237.5
Maximum Style Loss    :  82597688.0
Total Style Loss      :  56669698496.5
Average Style Loss    :  14984055.657456372
```

New font generation consists of the following steps, which are repeated until desirable outputs are achieved:
1. The inputs include the style image, target character and the generated image.
2. The three inputs are passed to the Neural network iteratively and the loss calculated is back propagated.
3. The output of each iteration is passed as input to the next iteration.

The most important step in this cycle is the loss calculation. To calculate the loss at each iteration, we split it into 2 parts:
1. Content Loss - refers to the difference in structure of the output image to the input target image
2. Style Loss - refers to the difference in Style of the output image to the input style image
the total loss consists of a weighted sum of the Content Loss and Style Loss.
these are calculated as follows:

```
# Loss is 0 initially
style_loss = original_loss = 0

# iterate through all the features for the chosen layers
for gen_feature, orig_feature in zip(
    generated_features, original_img_features):
    # batch_size will just be 1
    batch_size, channel, height, width = gen_feature.shape
    original_loss += torch.mean((gen_feature - orig_feature) ** 2)

for style_features in style_features_all:
    for gen_feature, style_feature in zip(
        generated_features, style_features):
        # batch_size will just be 1
        batch_size, channel, height, width = gen_feature.shape
        # Compute Gram Matrix of generated
        G = gen_feature.view(channel, height * width).mm(
            gen_feature.view(channel, height * width).t()
        )
        # Compute Gram Matrix of Style
        A = style_feature.view(channel, height * width).mm(
            style_feature.view(channel, height * width).t()
        )
        style_loss += torch.mean((G - A) ** 2)

style_loss/=len(style_features_all)
total_loss = alpha * original_loss + beta * style_loss
```

5.6 Implementation and pseudocode

Data Collection:

Since the data requirement of our project is only one set of alphabets of whichever language we plan to generate fonts of, we found this site called Indian Type Foundry which had the glyphs of alphabets for most Indian Languages.
We use WGET which is a command line tool to download the glyph of each font.
To automate the download of data we iterate over the names of all the glyphs on the site and download each one using the we get command.
We can download the fonts for any indian language available on the site by just replacing the font ID in the URL. Here is the example code to download the glyphs for Telugu language.

```
%mkdir Telugu
%cd Telugu
for i in range(2,500):
  letter = "https://d9qdd6ey7o7ay.cloudfront.net/assets/Glyphs/Font-676/Glyph-"+str(i)+".png"
  !wget $letter
```

Updating Filters

Since our method relies on changing the 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
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.

```
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)
```

Models:

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 filter

```
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)
```

CHAPTER 6

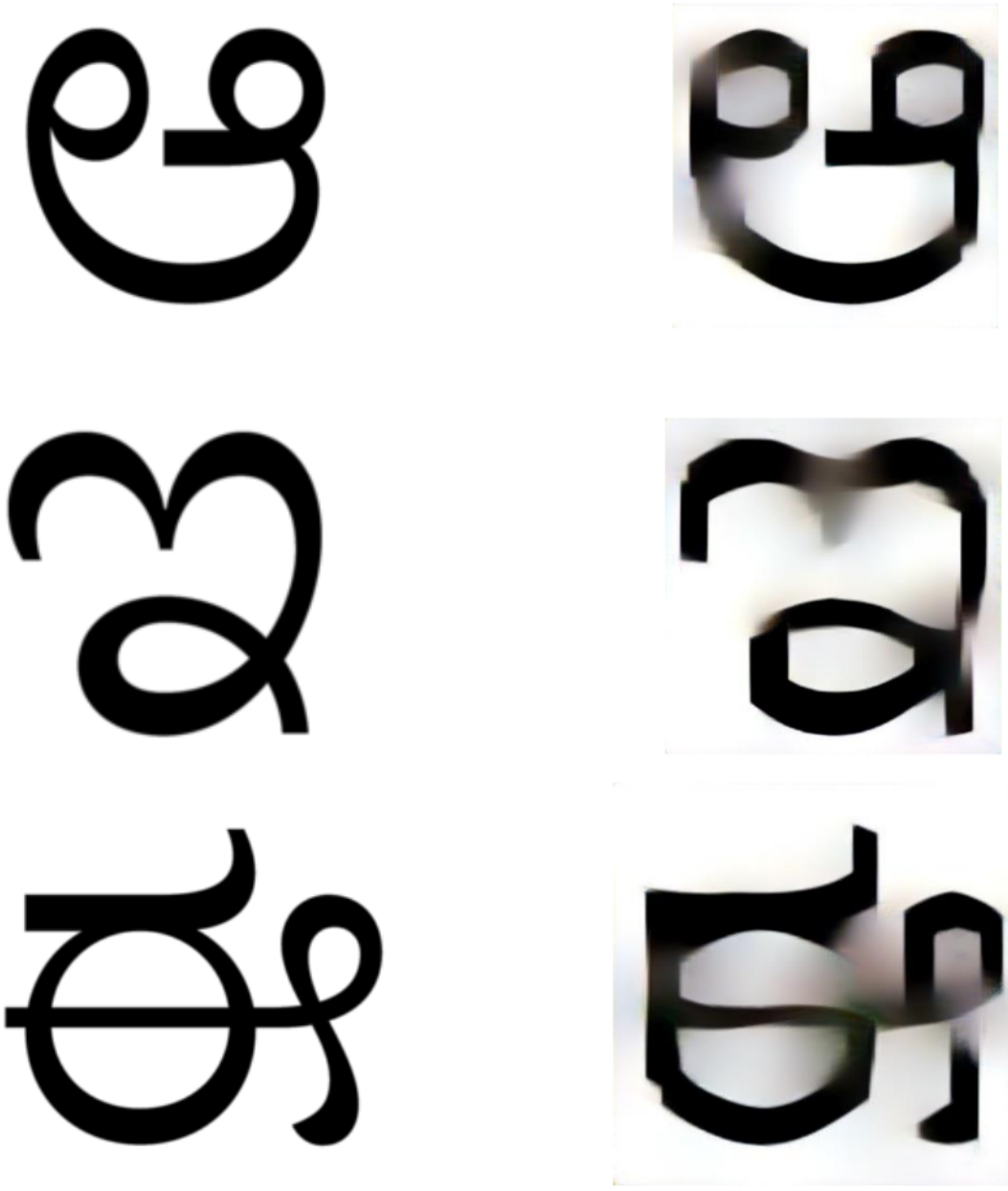
RESULT AND CONCLUSION

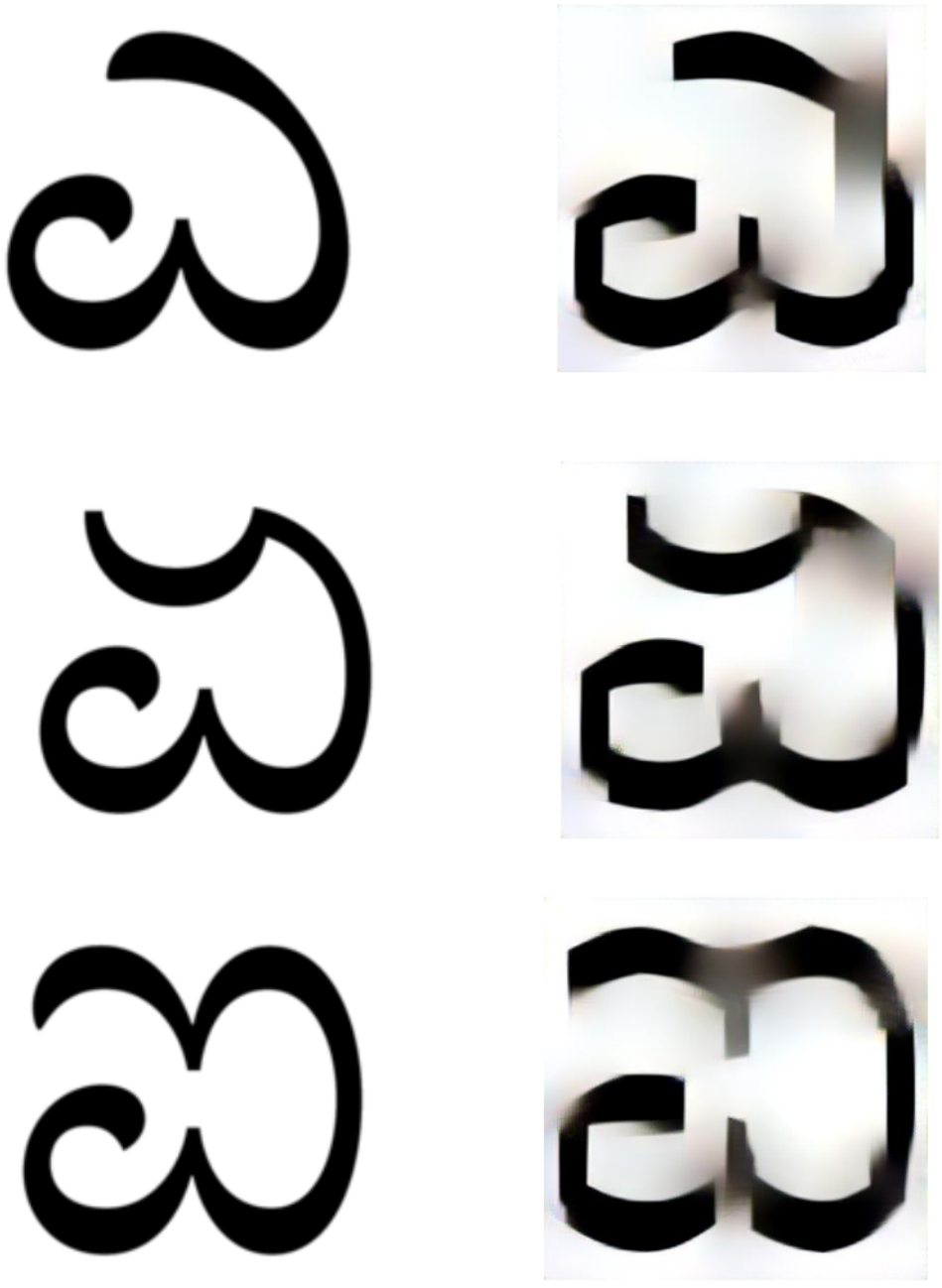
Results

STYLE INPUT

INPUT

OUTPUT





Conclusion

Phase-1
We have successfully completed the tasks allocated as Phase-1 of our project which is the Style-Consistent Kannada Font Generation.

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 customized 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 customized 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 the goals of Capstone Project Phase-1 where we learnt and generated the cellular automata of every single letter and digit of the Kannada language. This was done with the required style – consistency and we created multiple results.

Phase-2
We have successfully completed the tasks allocated as Phase-2 of our project which is the Style-Consistent Kannada Font Generation.

Started off with deep diving into the literature survey once again to gain understanding on hw to work through the problem statement. This is where we did research on the different ways in which it can be done and came up with two approaches out of the many.

This was the
1. Neural Style Transfer method as well as the
2. Autoencoders.

For both these approaches, the dataset for English was required and hence we obtained the dataset for the same. We the cleaned and successfully labelled it to help with the mapping.

We then built a model for both the approaches and trained and tested our data. The performance for

autoencoders seemed to reach a limit and not get better beyond that point and hence we stuck to Neural Style transfer and carried it forward.

This has helped us achieve our goal of Capstone Project Phase-2 which was to achieve style transfer. Overall, we have successfully completed both our phases and have created a style consistent font generation model for the Kannada scripts.

REFERENCES/ BIBLIOGRAPHY

Article in Online Encyclopedia

[1] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, Michael Levin, "Growing, Neural Cellular Automata", Feb. 11, 2020. [Online], Available: https://distill.pub/2020/growing-ca/ [Accessed Sep. 02, 2020]

Research paper

[2] Eric Bernhardsson, "Deep-fonts", Jan. 21,2016. Available: https://erikbern.com/2016/01/21/analyzing-50k-fonts-using-deep-neuralnetworks.html [Accessed Oct. 04, 2020]

Research paper

[3] Hideaki Hayashia, Kohtaro Abea, Seiichi Uchidaa, GlyphGAN, *Department of Advanced Information Technology, May 30,* 2019. Available: https://arxiv.org/pdf/1905.12502.pdf [Accessed Sep. 04, 2020]

Journal Article Abstract

[4] Koen Janssens, "Network Cellular Automata—A Development for the Future?", *Computational Materials Engineering*, 2007. [Online], Available: https://www.sciencedirect.com/topics/computerscience/cellularautomata [Accessed Sep. 03, 2020]

E-books

[5] Gavin Andrews, *Cellular Automata and applications*. [E-book] Available: https://www.whitman.edu/Documents/Academics/Mathematics/andrewgw.pdf

Research paper

[6] Samaneh Azadi, Matthew Fisher, Vladimir Kim, Zhaowen Wang, Eli Shechtman, Trevor Darrell, "Multi-Content GAN for Few-Shot Font Style Transfer", Dec. 01, 2017. [Online]

Available: https://arxiv.org/pdf/2006.06676.pdf [Accessed Jan. 16,2021]

Research paper

[7] Yonggyu Park, Junhyun Lee, Yookyung Koh, Inyeop Lee, Jinhyuk Lee, Jaewoo Kang, "Typeface Completion with Generative Adversarial Networks", Dec. 13, 2018. [Online],

Available: https://arxiv.org/pdf/1811.03762.pdf [Accessed Jan. 05, 2020]

Research paper

[8] Yizhi Wang, Peking University, "Attribute2Font", May. 16, 2020. [Online], Available: https://arxiv.org/pdf/2005.07865v1.pdf [Accessed Jan. 22, 2021]

Research paper

[9] Leon A. Gatys, Alexander S. Ecker, Matthias Bathge, "A Neural Algorithm of Artistic Style", Sep. 02, 2015. [Online], Available: https://arxiv.org/pdf/1508.06576.pdf [Accessed Feb. 01, 2021]

Research paper

[10] Shuai Yang, Jiaying Liu, Zhouhui Lian, Zongming Guo, "Awesome Typography: Statistics-Based Text Effects Transfer", *Institute of Computer Science and Technology,* Dec. 06, 2016.[Online], Available: https://arxiv.org/pdf/1611.09026.pdf

Style-consistent Kannada Font Generation

ORIGINALITY REPORT

13%	6%	4%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to PES University Student Paper	5%
2	arxiv.org Internet Source	2%
3	Hideaki Hayashi, Kohtaro Abe, Seiichi Uchida. "GlyphGAN: Style-consistent font generation based on generative adversarial networks", Knowledge-Based Systems, 2019 Publication	1%
4	Shuai Yang, Jiaying Liu, Zhouhui Lian, Zongming Guo. "Awesome Typography: Statistics-Based Text Effects Transfer", 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017 Publication	1%
5	erikbern.com Internet Source	1%
6	Yizhi Wang, Yue Gao, Zhouhui Lian. "Attribute2Font", ACM Transactions on Graphics, 2020 Publication	1%

7	export.arxiv.org Internet Source	1 %
8	Submitted to National Institute Of Technology, Tiruchirappalli Student Paper	1 %
9	scroll.in Internet Source	<1 %
10	bair.berkeley.edu Internet Source	<1 %
11	www.tuicool.com Internet Source	<1 %
12	github.com Internet Source	<1 %
13	Wenjing Wang, Jiaying Liu, Shuai Yang, Zongming Guo. "Typography With Decor: Intelligent Text Style Transfer", 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019 Publication	<1 %
14	www.mdpi.com Internet Source	<1 %
15	Submitted to Korea University Student Paper	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		