Word Count: 5010

Plagiarism Percentage 5%

Matches

1 **World Wide Web Match**
View Link

2 **World Wide Web Match**
View Link

3 **World Wide Web Match**
View Link

4 **World Wide Web Match**
View Link

5 **World Wide Web Match**
View Link

6 **World Wide Web Match**
View Link

7 **World Wide Web Match**
View Link

8 **World Wide Web Match**
View Link

9 **World Wide Web Match**
View Link

10 **World Wide Web Match**
View Link

11 **World Wide Web Match**

View Link

12 **World Wide Web Match**
View Link

13 **World Wide Web Match**
View Link

14 **World Wide Web Match**
View Link

Suspected Content

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 6

UE17CS490A – Capstone Project Phase - 1 Submitted by: Saahil B Jain Jeevana R Hegde PES1201700241 PES1201700633 Under the guidance of Prof. K S Srinivas Associate Professor PES University August - December 2020

> DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING FACULTY OF ENGINEERING 7

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

> ENGINEERING CERTIFICATE This is to certify that the dissertation entitled 7

‘Style-Consistent Kannada Font Generation’ is a bonafide work carried out by Saahil B Jain Jeevana R Hegde PES1201700241 PES1201700633 in partial fulfilment for the completion of seventh semester Capstone Project Phase - 1 (UE17CS490A) in the Program of Study -

> Bachelor of Technology in Computer Science and Engineering under rules and regulations of 13

PES University, Bengaluru during the period Aug. 2020 – Dec. 2020.

It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 7th semester academic requirements in respect of project work. 2

Signature <Name of the Guide> Designation Signature Dr. Shylaja S S Chairperson Signature Dr. B K Keshavan Dean of Faculty

External Viva Name of the Examiners Signature with Date 1. _ 2. _ DECLARATION We 11

hereby declare that the Capstone Project Phase - 1 entitled “Style-Consistent Kannada Font Generation” has been carried out by us under the guidance of Prof. K S Srinivas, Associate Professor and

submitted in partial fulfilment of the course requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering of PES University, Bengaluru during the academic 4

semester August – December 2020. The

matter embodied in this report has not been submitted to any other university or institution for the award of any degree. 5

PES1201700241 Saahil B Jain PES1201700633 Jeevana R Hegde

ACKNOWLEDGEMENT I would like to express my gratitude to Prof. 12

K S Srinivas,

Department of Computer Science and Engineering, 14

PES University, for his continuous guidance, assistance, and encouragement throughout the development of this UE17CS490A - Capstone Project Phase – 1. I am grateful to the project coordinators, Prof. Silviya Nancy J and Prof. Sunitha R, for organizing, managing, and helping with the entire process. I take this opportunity to thank Dr. Shylaja S S,

Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support 9

I have received from the department. I

would like to thank Dr. B .K. 9

Keshavan, Dean of Faculty, PES University for his help. I am deeply grateful to Dr.

M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. 10

Suryaprasad J, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement I have received from my family and friends. 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. We plan to also try and achieve style transfer from pre-existing fonts for other languages like English. Chapter No. 1. 2. 3. 4. 5. 6. 7. TABLE OF CONTENT Title INTRODUCTION PROBLEM STATEMENT LITERATURE REVIEW 3.1 Growing Neual Cellular Automata 3.2 Deep-fonts 3.3 GlyphGAN HIGH LEVEL DESIGN DOCUMENT SYSTEM DESIGN IMPLEMENTATION AND PSEUDOCODE CONCLUSION OF CAPSTONE PROJECT PHASE-1 8. PLAN OF WORK FOR CAPSTONE PROJECT PHASE-2 REFERENCE/ BIBLIOGRAPHY Page No. 7 8 11 15 17 20 23 24 27 28 29 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. 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. 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 3

the need to deliver a great experience to non- English speaking users is creating a greater demand for high-quality, multi-script typefaces. 3

We particularly loo at use cases of this problem statement: 1. Television channels - 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 each page to mark the name of newspaper. Also includes a consistent font style of the entire text or content. 3. Movies - An eye catchy font to grasp public attention with a single glance. 4. Company name font designers - 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 crutial 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. 3.1 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. 3.2 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 8

CHAPTER 4 HIGH LEVEL DESIGN DOCUMENT Design Details 1.1. Novelty 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. 1.2. Innovativeness 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. 1.3. Architecture Choices To automate the generation of fonts, we had three options available. These are as follows: A. 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 1

is trained to generate the fonts from these “font vectors”. Any change in the “font vector” would then result in new fonts. Pros: a). Simple and easy to build architecture. b). Enough data is present. Hence, slight modifications lead to the generation of significantly large number of fonts with high legibility. c). Style consistency is reasonable. Cons: a). There is an extremely high data dependency. b). Diversity is quite low. B. 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: a). Better legibility compared to deep-fonts. b). Style consistency is the same. c). More diversity in available in the fonts compared to neural network. Cons: a).High data dependency is present, which is unavailable for Indian languages like Kannada. C. 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: a). Data requirement is very low. Just a single font is required to generate significantly different fonts. b). Style consistency is accurate. c). Models are very small. Cons: a). Every letter requires a different model. b). 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. 1.4.

Performance 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 1.5. 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. 1.6. Dependencies 1 set of alphabets with all maatras. 1.7. Assumptions Compute resource powerful enough to generate the base models in reasonable amount of time. 1.8. 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. 1.9. 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. 1.10. 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 availablefonts. 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. 1.11. 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. 1.12. 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 7 SYSTEM DESIGN The system design is as follows: We plan in breaking down the complex problem into simpler steps to make it a seemless 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. CHAPTER 8 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 wget 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. 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. 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 Apart from these, we intend to implement the project with a simple UI/UX inspired from this:

CHAPTER 9 CONCLUSION OF CAPSTONE PROJECT PHASE-1 We have successfully completed the tasks allocated as PHASE-1 of you 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. CHAPTER 10 PLAN OF WORK FOR CAPSTONE PROJECT PHASE-2 Even with successful completion of Capstone Project Phase-1 where we learnt and generated the cellular automata of every single letter and digit of the Kannada language with the required style – consistency, we have a long way to go for this entire project to be applicable in real life. Here, comes the tricky part of creating multiple varying fonts with uniform moderated style consistency. This would require the right hyper parameters for the Neural Network aspect of it as well as additional features which we will be looking at. Taking it a step further, we intend to achieve a more generalized model to accept dataset of any language and provide various automated fonts. We intend to do this by starting off with a very language very similar to Kannada in terms of writing, which is Telugu. However, it turns out to be more complex due it’s significant curvature fonts with the alphabets. Once this is done, we will be looking into all the fine tuning and documentation aspects of the entire research project. This is crucial for the presentation of the entire project and to bring it all together. REFERENCE / 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 Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating

Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Generating Representative Embeddings of Drugs to Predict Functionality and Adverse Effects Dept. of CSE Jan - May 2020 Page 7 Dept. of CSE Jan - May 2020 Page 8 Dept. of CSE Jan - May 2020 Page 9 Dept. of CSE Jan - May 2020 Page 10 Dept. of CSE Jan - May 2020 Page 11 Dept. of CSE Jan - May 2020 Page 12 Dept. of CSE Jan - May 2020 Page 13 Dept. of CSE Jan - May 2020 Page 14 Dept. of CSE Jan - May 2020 Page 15 Dept. of CSE Jan - May 2020 Page 16 Dept. of CSE Jan - May 2020 Page 17 Dept. of CSE Jan - May 2020 Page 18 Dept. of CSE Jan - May 2020 Page 19 Dept. of CSE Jan - May 2020 Page 20 Dept. of CSE Jan - May 2020 Page 21 Dept. of CSE Jan - May 2020 Page 22 Dept. of CSE Jan - May 2020 Page 23 Dept. of CSE Jan - May 2020 Page 24 Dept. of CSE Jan - May 2020 Page 25 Dept. of CSE Jan - May 2020 Page 26 Dept. of CSE Jan - May 2020 Page 27 Dept. of CSE Jan - May 2020 Page 28 Dept. of CSE Jan - May 2020 Page 29