2024 4th Asian Conference on Innovation in Technology (ASIANCON)
Pune, India. Aug 23 - 25, 2024

Synthetic Image Generation using StackGAN

Sanika Nadgir
Computer Science
Cummins College of Engineering for Women
Pune, India
sanika.nadgir@cumminscollege.in

Sandhya Arora
Computer Science
Cummins College of Engineering for Women
Pune, India
sandhya.arora@cumminscollege.in

Snigdha Chitnis
Computer Science
Cummins College of Engineering for Women
Pune, India
snigdha.chitnis@cumminscollege.in

2024 4th Asian Conference on Innovation in Technology (ASIANCON) | 979-8-3503-5421-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/ASIANCON62057.2024.10838089

***Abstract*— The generation of artificially created images which resemble actual photos is known as synthetic image generation. Due to the many potential uses of synthetic image generation, it has drawn a lot of interest. Conversational chatbots that produce contextual visuals based on user input are one example of such an application. This ability is very beneficial in situations like drawing criminals from their descriptions. The field still faces difficulties in producing high-quality images that are aesthetically identical to actual photographs. In order to solve this issue, this paper offers the use of a cutting-edge Generative Adversarial Network (GAN) model, StackGAN, to produce high-quality photos. This main goal of this work is to investigate how well StackGAN can create realistic graphics from inputs that include both images and text embedding from a single dataset. It also seeks to contribute to the development of synthetic image generation techniques and offer a method for producing visually accurate artificial images by overcoming the drawbacks of prior GAN structures. The improved performance of StackGAN is demonstrated in creating high-quality photos with amazing visual fidelity through rigorous experimentation and evaluation.**

Keywords—Generative Adversarial Network, generator, discriminator, StackGAN, Loss functions

I. INTRODUCTION

Synthetic image generation refers to the process of producing computer-generated images that closely resemble real images in terms of visual realism. This technique involves the use of GANs to create these lifelike images. The architecture was first portrayed by Goodfellow et al. [1] in their study. The discriminator and generator networks are the primary parts of GANs, which are deep neural networks. The discriminator network seeks to discriminate between the generated image and a real image from the training dataset, while the generator network generates an image using random noise as input. The generator tries to fool the discriminator network to think that the image generated is in fact real, while the discriminator tries to accurately categorize the created image as fake. The above mentioned networks are trained in an adversarial manner.

GANs can be trained to produce images that are consistent with a given text description in the context of text-to-image generation. A study by Reed et al. [2] depicted the same.

The generator network can be conditioned using the text descriptions to make sure that the image produced matches the text exactly. The discriminator network may be trained to tell the difference between a genuine image that fits the written description and the generated image.

GAN aims to make use of noise as input in its basic form. After that, the generator turns this noise into a useful output. By adding noise GAN is able to generate a wide range of data and also by sampling from various points throughout the target distribution. By integrating the feedback received from the discriminator, the generator component of a GAN acquires the capacity to generate artificial data. It learns to manipulate the discriminator into classifying its generated output as authentic.

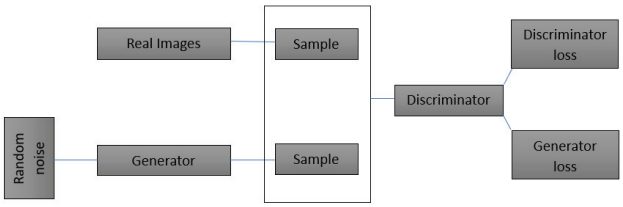

Fig. 1. Generative adversarial network architecture

The training data for the discriminator originates from two sources:

- The discriminator uses genuine data instances, such as actual photographs of people, as positive examples while being trained.
- The generator is used to generate artificial data instances, which is used by the discriminator as bad examples while being trained.

GAN uses following loss functions: -

$$\mathbf{E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))]} \quad (1)$$

G(z) refers to the output of the generator when it receives noise z as input. D(x) refers to the discriminator's estimation of the probability that the original data x is real, while D(G(z)) represents its estimation of the probability that a synthetic sample G(z) is real. E_x and E_z denote the average likelihood computed over all original and synthetic data, respectively. [1]

Discriminator loss and generator loss are further subcategories of the common GAN loss function.

- **Discriminator loss**

During the training of the discriminator, it evaluates and categorizes both real and synthetic data generated by the generator. It penalizes itself when it incorrectly identifies real as fake or artificial image that is created from a generator as a real instance. This is achieved by maximizing the following function

$$\nabla_{\phi_d} \frac{1}{m} \sum_{i=1}^{m} [\boldsymbol{log D}(\boldsymbol{x^i}) + \mathbf{log}\,(\mathbf{1} - \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z^i})))] \quad (2)$$

Here, the term "log(D(x))" represents the probability of the discriminator correctly classifying a real image. By maximizing "log(1-D(G(z)))", the discriminator aims to accurately point out the artificial image generated by the generator. [1]

- **Generator loss**

During the training, it generates output based on random noise samples. This output is then assessed by the discriminator, which classifies it as real or fake based on its ability to distinguish between the two. The generator is rewarded if it successfully deceives the discriminator and penalized otherwise. The equation below has to be lowered to train the generator:

$$\nabla_{\emptyset_g} \frac{1}{m} \sum_{i=1}^{m} [\log (1 - D(G(z^i)))] \quad (3)$$

Here, the notations are the same as described earlier. Additionally, θ_g represents a hyperparameter of an MLP that influences the behaviour of the model and corresponds to a differentiable function G (z; θ_g). This function G takes the input noise z and maps it to the data space. [1]

Overall, GAN-based image generation has advanced significantly in recent years, with a variety of methods that generate images with excellent quality and a variety of information.

There are multiple models that have different applications like Conditional GAN [11] can be used to transform sketches to images, Deep Convolutional GAN [17] can be used in medical image synthesis, StackGAN [8] can be used to generate high quality images for advertising, StyleGAN [12] for photo realistic face generation, CycleGAN [7] for graphic design or fashion.

These generated images can be utilized to add variance to ML models that will improve performance and accuracy

There are numerous possible uses for creating images from text, including in the disciplines of art, design, and computer vision. In cases where it is challenging or expensive to directly collect images, it can be used to produce images from written descriptions.

The performance of GAN can also be improved by using various techniques like filter aiding [10] or feature matching, mini-batch discrimination, historical averaging, and virtual batch normalization [9] or Spatially-Adaptive Normalization [14] or embedding techniques [16]

II. Literature Review

The GAN architecture consists of two neural networks, namely a generator and a discriminator. This architecture was first portrayed by Goodfellow et al. [1] in their study. As mentioned above, the generator creates images, and the discriminator distinguishes between the generated images and the real photos from the training dataset. Together, the two networks are trained, with the discriminator aiming to properly recognize the generated images and the generator attempting to create images that can deceive the generator.

Since then, a number of methods for using GANs in text-to-image generation have been put forth including:

GAWWN [3] an effective model for generating sharp and more accurate results in the samples using GAWWN with real or artificial key points.

AttnGAN [4] which led to Improvements in generating realistic images that closely match the given text descriptions achieved cutting edge performance in terms of diversity and image quality

Dual Attention GAN [5] this model effectively captured and emphasized relevant spatial and channel information, leading to more accurate and coherent image synthesis.

StackGAN++ [6] which achieved significant improvements in generating realistic images with fine-grained details compared to previous methods.

CycleGAN [7] which generated images that were visually convincing and demonstrated the ability to learn meaningful mappings between different visual domains.

CGAN [11] which can generate realistic handwritten digit samples conditioned on specific digit classes, the ability to perform image completion, and image transformation tasks using CGANs.

Style GAN [12] which achieves outstanding performance in terms of image quality, and disentanglement of different visual attributes.

BigGAN [13] that refers to a large-scale GAN outperforms previous models in terms of inception score, FID, and more evaluation metrics.

LAPGAN [15] which demonstrates superior performance than previous approaches in terms of visual quality and for generating images with intricate textures and details.

DCGAN [17] which presented a collection of more reliable architectural designs for training GANs and provided validation indicating that adversarial networks acquire effective image representations suitable for generative modeling purposes.

III. Description of the System or Research

The StackGAN method [8] is a recent and more developed method for generating images because it has two stages. One that generates the image and another that creates a high-resolution version of the image generated in Stage-I. In this phase, we establish the fundamental shape and colors of the object by utilizing a provided textual description, resulting in the creation of low-resolution images. Subsequently, during the second stage, the Stage-II GAN utilizes the outcomes from Stage-I and incorporates text embeddings as input, enabling the generation of high-resolution images characterized by realistic and intricate details.

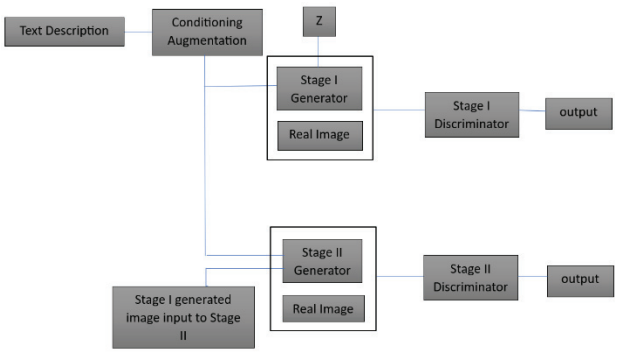

Fig. 2. Working of StackGAN to produce images via text description

In the StackGAN diagram, the process begins with the text descriptions of images being fed into the model. These text descriptions contain information about the desired image to be generated. The model then performs conditioning augmentation on the text embeddings during the training phase. Conditioning augmentation involves introducing

random noise into the text embeddings, which helps to create more diversity in the generated images.

Once the text descriptions are conditioned, the model moves into the first stage of training. In this stage, there are two key components: the generator and the discriminator.

The generator takes the conditioned text embeddings as input and aims to generate low resolution images. It gradually processes the input and upscales it to create a low-resolution image that serves as an initial approximation. The generator typically consists of convolutional and up sampling layers that learn to transform the input text embeddings into a visual representation.

The discriminator, takes both real low-resolution images and the generated low-resolution images from the generator as inputs. Its purpose is to distinguish between real and fake images. It is trained to classify the images generated as real or fake and provides feedback to the generator to improve its image generation capabilities. Through an adversarial training process, the generator and discriminator continuously learn and refine their respective abilities to generate and discriminate images.

After the first stage of training, the model moves into the second stage, which focuses on generating high-quality images. At this stage, the low-resolution images generated by the generator in stage one are used as input.

The stage II generator takes these low-resolution images and creates high resolution ones. The discriminator then takes the high-quality images and real images and tries to distinguish between them.

IV. Novelty or the Proposed Work

The StackGAN model is applied on the images file and a text description file which describes all images. The methodology to do is includes:

Stage-I of StackGAN

The generator takes the description of images as input and produces a blurry image, which is then fed into the discriminator network, which tries to differentiate between the generated image and the real image.

The necessary functions are defined for conditioning augmentation, where the mean and log standard deviation (log_sigma) are extracted from the output tensor x, the standard deviation is calculated using exponentiation, and random noise is generated using a normal distribution. There is also a function that takes as input a tensor with shape (1024,), which represents the size of the vocabulary in the text data, and passes this input through a fully connected layer with 256 units and LeakyReLU non-linearity and then applies the conditioning augmentation function to the output of this layer using a lambda layer. The resulting tensor ca is the output of the CA network, which will be used as the conditioning variable for the GAN during training. This function returns a Keras Model object that represents the CA network. It then applies a reparameterization trick to this noise by scaling it with the standard deviation and adding it to the mean, resulting in a tensor c that serves as the conditioning variable for the GAN.

Stage-I generator:

While building the stage-I generator, the two input vectors are concatenated and fed through a series of upsampling blocks and convolutional layers to produce a 64x64 image with a tanh activation function applied to it.

A conv2D function is defined that returns the final activation layer, and an embedding compressor is defined that returns a Keras model for an embedding compressor that takes a 1024-dimensional embedding vector and compresses it into a 128-dimensional vector.

Stage-I discriminator:

Then the stage-I discriminator is built, which takes two inputs, the feature from Generator and the text embedding. The concatenated features are passed through one more convolutional layer (512 filters), a 1x1 kernel size, and a LeakyReLU function. Then, the output is flattened and fed through a fully connected layer with one unit to produce a binary classification output indicating whether the input image is real or fake. Finally, the function returns a Keras model with two input layers and one output layer.

Combination of the generator and discriminator:

The adversarial model is built from a stage 1 generator model and a stage 1 discriminator model. It takes the same three input layers as the generator model and outputs two tensors: the likelihood of the image being real (probabilities) and the conditioning vector (ca) generated by the generator model. In the adversarial model, the discriminator model is frozen, so its weights are not updated during training. It is trained to maximize the probability of the discriminator classifying the generated image as real while simultaneously minimizing the difference between the generated image and the target image. Multiple functions are defined to work on the images, including loading them from our directories, normalizing or reshaping them, and saving them according to our model's requirements.

Building the stack GAN Stage I model:

The StackGAN Stage-I model is built, where it initializes various hyperparameters such as the number of epochs, size of the latent space, batch size, and learning rates for the generator and discriminator networks. Then, it builds and compiles the generator, discriminator, CA (conditional augmentation) network, and compressor using their respective functions. It also builds and compiles the adversarial model, which combines the generator and discriminator. Additionally, it sets up a checkpoint to save the models during training. A function sets up a TensorBoard instance to visualize the model's performance. Finally, the training function trains the model using the input data and various loss functions for the generator and discriminator. It iteratively trains both networks by passing the output of the generator through the discriminator to improve the generated images so that they resemble the real images. During training, the generator and discriminator loss values are printed for each batch. Additionally, the function saves sample images every 5 epochs to visualize the quality of the generated images.

The resultant images produced by this stage are saved in a specific file and can be viewed there.

Stage-II of StackGAN

In Stage-II, the generator network takes the low-resolution image and the text description as input and produces a fine-grained image that is uniform and consistent with the blurry

image and the input text. This image is then fed into the discriminator network, which distinguishes between the image that is generated and the real image from the training dataset. In this stage, functions are defined to join the conditioned text with the encoded image with dimensions, and to return a layer with computed identity mapping.

Stage-II generator:

The stage-II generator is then built, which takes two inputs: a conditioning text vector of size 1024 and a low-resolution image of size 64x64x3. It first performs conditioning augmentation on the text input using a dense layer, a Leaky ReLU function, and a lambda layer. Then, it applies a down-sampling block to the input image using 3 convolutional layers with higher number of filters, after which ReLU activation and batch normalization are conducted. Next, it concatenates the input text with the encoded image. It then applies a series of residual blocks to enhance the feature representation of the concatenated tensor. After that, it applies four up-sampling blocks to increase the resolution of the tensor. Finally, it applies a convolutional layer to generate the output image and an activation layer to normalize the pixel values. The function returns a model object that takes the conditioning text and input image as inputs and outputs the generated image and the conditioning augmentation tensor.

Stage-II discriminator:

Then the stage-II discriminator takes a 256x256x3 image as input, which is produced by the generator, and a compressed and spatially replicated embedding produced by the conditioning augmentation network. This network consists of a succession of convolutional blocks, where each block has a convolutional layer, batch normalization and a Leaky ReLU activation function. The output of the last convolutional block is integrated with the input embedding, followed by another convolutional block, batch normalization, and finally a sigmoid activation function is used to generate a binary output which depicts whether the input image is real or fake.

Final Combination:

Then the adversarial network is created, which takes three inputs: a tensor representing the textual description of the image, a tensor representing a noise vector, and a tensor representing the compressed and spatially replicated embedding. The function first creates an image from the text description input and the noise vector using the Stage-I Generator Model. The trainable property of the Stage-II discriminator and stage-I generator models is set to false to freeze their layers so that they are not updated during training. Next, the Stage-II Generator is used to generate a higher-resolution image from the conditioned embedding and the Stage-I generated image. Finally, the Stage-II discriminator predicts the likelihood of the generated image being either real or fake. The output of this function is a model object representing the entire Stage-II adversarial network with the inputs and outputs described above.

In the final phase of building the Stage-II GAN, a method initializes the object with the required parameters and also compiles and loads the weights of the Stage-I generator. It also compiles the Stage-II generator, discriminator, CA network (condition augmentation network), embedding compressor, and adversarial network, and the training method trains the Stage-II StackGAN. It loads the data, generates random noise for the latent space, and then generates the high-resolution images from the low-resolution images using the Stage-II generator. It then trains the Stage-II generator and discriminator on the real and fake images. Finally, it trains the adversarial network using the generator, discriminator, and stage-I generator. The losses of the discriminator and generator are printed at every point, and the images are saved after 5 epochs.

The images produced at this stage have been observed to have a higher resolution than the Stage-I images.

Datasets

The datasets used for the StackGAN architectures are the Caltech-UCSD Birds-200-2011 (CUB-200-2011). The dataset comprises a total of 11,788 images categorized into 200 subcategories related to birds. Out of these, 5,994 images are allocated for training, while 5,794 images are reserved for testing.

Additionally, the dataset includes a text embedding dataset that provides descriptions for each image.

Detailed annotations are available for each image, including 1 subcategory label, 15-part locations, 312 binary attributes, and 1 bounding box.

V. Results and Discussions

The previous parameters for learning rate that were experimented with were 0.1-0.8, but results were not promising even after 80 epochs.
Hence, after experimentation, with the final learning rate of the generator and discriminator of stage-I being 0.0002, the results below were obtained, which were images of 64*64 pixels according to the parameters initialized in Stage I. Stage II generates 256*256 pixel image which is a higher resolution images
The batch size used is 64, so 64 images were processed in each training iteration.
The generator was compiled with MSE (mean squared error), and binary cross entropy was used for compilation of the discriminator
The optimizer used for compilation is Adam.
These are the results obtained corresponding to the text description.
Stage I results:

After 24 epochs of Stage 1 of StackGAN:

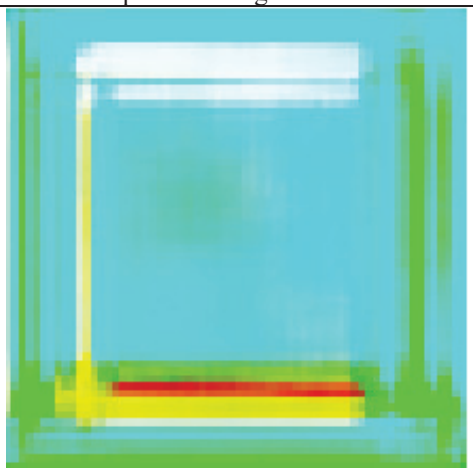

Fig. 3. Result after 24 epochs

After 85 epochs of Stage 1 of StackGAN

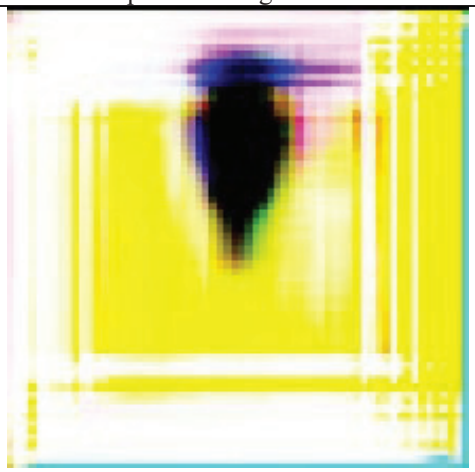

Fig. 4. Result after 85 epochs

After 170 epochs of Stage 1 of StackGAN

Fig. 5. Result after 170 epochs

After 293 epochs of Stage 1 of StackGAN

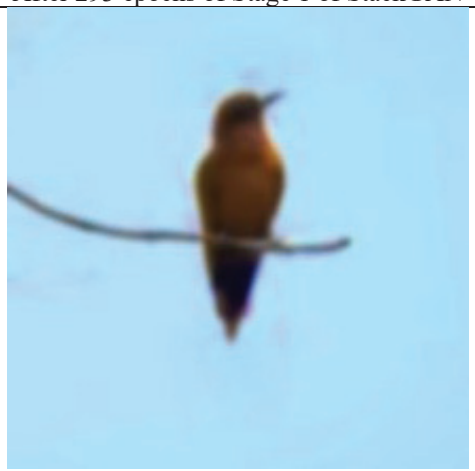

Fig. 6. Result after 293 epochs

Stage-II of StackGAN improves the resolution and quality of the images generated by Stage-I. Hence, with a learning rate of 0.0004 for generator and discriminator, this was the output after 230 epochs and a 256*256 pixel image parameter.

Stage II result:

Fig. 7. Result after Stage II of StackGAN

VI. Conclusion

The StackGAN model has been shown to generate high-quality images that match the text descriptions. In the studies conducted on StackGAN, it is seen that the hierarchical nature of the model allows it to generate images at different scales, which can be useful for generating images with complex structures and fine details. Overall, the StackGAN model is a promising approach for text-to-image generation that has shown strong performance in generating high-quality images from textual descriptions. For further research, other GAN models can also be worked on for text data and generation of images based on multiple inputs from the user can also be tested.

REFERENCES

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative Adversarial Networks", Conference on Neural Information Processing Systems (NeurIPS), 2014.

[2] Reed, Scott, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, and Honglak Lee, "Generative Adversarial Text to Image Synthesis", International Conference on Machine Learning (ICML), 2016.

[3] Reed, Scott, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee, "Learning What and Where to Draw", Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[4] Xu, Tao, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He, "AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks", Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[5] Tang, Hao, Song Bai, and Nicu Sebe, "Dual Attention GANs for Semantic Image Synthesis", Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[6] H. Zhang, T. Xu, and J. Zhu, "StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 8, pp. 1947-1962, Aug. 2020. doi: 10.1109/TPAMI.2019.2908850.

[7] Zhu, ***J.***, et al., "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", in the International Conference on Computer Vision, 2017.

[8] Zhang, H., et al., "StackGAN: Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks", In the International Journal of Computer Vision, 2017

[9] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen, "Improved techniques for training gans", arXiv:1606.03498, 2016.

[10] Jyoti S. Raghatwan and Dr. Sandhya Arora, "Improved Sketch-to-Photo Generation Using Filter Aided Generative Adversarial Network", ISSN: 2321-8169, Volume: 10, Issue: 9, 2022

[11] Mehdi Mirza and Simon Osindero, “Conditional generative adversarial nets”, arXiv:1411.1784, 2014.

[12] Tero Karras, Samuli Laine, and Timo Aila, “A style-based generator architecture for generative adversarial networks”, arXiv:arXiv:1812.04948, 2019.

[13] Andrew Brock, Jeff Donahue, and Karen Simonyan, “Large scale gan training for high fidelity natural image synthesis”, arXiv:1809.11096, 2018.

[14] Park, T., et al., “Semantic Image Synthesis with Spatially-Adaptive Normalization”, In Computer Vision and Pattern Recognition, 2019

[15] L. Denton, S. Chintala, R. Fergus, et al. “Deep generative image models using a laplacian pyramid of adversarial networks”, NIPS, 2015.

[16] Akata, S. Reed, D. Walter, H. Lee, and B. Schiele, “Evaluation of Output Embeddings for Fine-Grained Image Classification”, CVPR, 2015.

[17] Alec Radford, Luke Metz, and Soumith Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, arXiv:1511.06434, 2015.

Author’s biography:

Sanika Nadgir: She is a student in MKSSS’s Cummins college of Engineering, studying Computer Science while pursuing an Honours degree in advanced Machine Learning and Data Science. She is currently an intern at Mastercard.

Snigdha Chitnis: She is a student in MKSSS’s Cummins college of Engineering, studying Computer Science while pursuing an Honours degree in advanced Machine Learning and Data Science. She is an upcoming intern at Ecolab.

Dr. Sandhya Arora: Dr. Sandhya Arora is working as Professor in Department of Computer Science Engineering, MKSSS’s Cummins college of Engineering Pune, having 26+ years of working experience. She is doctorate in Computer science and have many research publications with 740 citations of her published papers.