

# Synthesizing and Imitating Handwriting using Deep Recurrent Neural Networks and Mixture Density Networks

<sup>1</sup>K.MANOJ KUMAR

Assistant Professor

Department of Computer Science & Engineering,  
Sri Venkateswara College of Engineering (SVCE),  
Tirupati, Andhra Pradesh 517507 India,  
kandalamanojkumar@gmail.com

<sup>2</sup>HARISH KANDALA

Software Engineer

Visa Inc.  
Bengaluru, India  
hakandal@visa.com

<sup>3</sup>Dr.N.SUDHAKAR REDDY

Principal

Sri Venkateswara College of Engineering (SVCE),  
Tirupati, Andhra Pradesh 517507 India,  
nsreddyonline@gmail.com

**Abstract** - Handwriting is a skill developed by humans from the very early stage in the order to represent his/her thoughts visually using letters and making meaningful words and sentences. Every person improves this skill by practicing and develops his/her own style of writing. Because of the distinctiveness of handwriting style, it is frequently used as a measure to identify a forgery. Even though the applications of synthesizing of handwriting is less, this problem can be generalized and can be functionally applied to other more practical problems. Synthesizing the handwriting is a quite complicated task to achieve. Deep recurrent neural networks specifically Deep LSTM cells can be used along with a Mixture Density Network to generate artificial handwriting data. But using this model we can only generate random handwriting styles which are being hallucinated by the model. Mimicking a specific handwriting style is not so efficient with this model. Mimicking or imitating a specific handwriting style can have an extensive variety of applications like generating personalized handwritten documents, editing a handwritten document by using the similar handwriting style and also it is extended to compare handwriting styles to identify a forgery. A web prototype is developed along with the model to test the results where the user can enter the text input and select handwriting style to be used. And the application will return the handwritten document containing input text mimicking the selected handwriting style. The application will also provide a way to fine-tune the handwriting styles by changing few parameters.

**Index Terms** – Hand Writing, Machine Learning, Neural Networks, Deep Learning

## I. INTRODUCTION

The problematic issue of handwriting generation has been addressed by various researchers using some complex mathematical methods but none of them achieved a decent result.

Synthesis of sequential data was impossible until Neural Networks and Deep Learning came into existence. Deep recurrent neural networks specifically Deep LSTM cells will be applied along with a Mixture Density Network (MDN) to produce artificial handwriting lines of data. This model was proposed by Google's Brain Team and it achieved good results. But using this model we can only generate random handwriting styles which are being hallucinated by the model. This model is not so efficient in retraining it to a precise style of handwriting. Therefore, mimicking any particular style, which is not in training data, can be quite difficult with this model. The next two sections describe about the motivation and previous research works on it. The fourth section elaborates the technical details of the model. Moreover, the fifth section shows the implementation details and results achieved.

## II. MOTIVATION

Humans learn writing by practicing repeatedly to study the strokes. The way we learn handwriting is almost similar to any task we learn. We learn things by repeating it until it finally becomes involuntary. Even with all these Deep Learning algorithms, computers still don't know how to learn a task. And moreover, humans deal with both historical and spatial information which is also difficult for computers to handle. Trying to solve some simple problems like handwriting may lead us to a better understanding of how humans think and can develop better algorithms for computers. The problem should be generalized as generating sequences rather than generating handwriting. This is how Alex Graves in his research paper achieved the solution. But just the generation of handwriting is not useful. The model should learn to imitate the particular style of handwriting.

### III. LITERATURE SURVEY

#### A. Survey of the Existing Models:

The number of contributions made by the researchers on handwriting synthesis is very less. Handwriting recognition is a common problem in the domain of machine learning compared to the handwriting synthesis. The main reason can be that handwriting recognition has more practical applications, rather than handwriting synthesis. A major contribution to handwriting synthesis was developed by researcher Alex Graves [1]. Before Alex Graves proposal, researchers used some classical machine learning algorithms to generate strokes and model characters. But none of those approaches generate natural human-like handwriting. All those results achieved were before advancements in Neural Networks.

Recently with the increase in computation power, Alex Graves was able to solve the problem by using Deep Recurrent Neural Networks. He generalized the problem to the generation of sequential data which even leads to the growth of Wavenet, an audio synthesis model which is widely being used by Google. His approach includes a grouping of three techniques: Deep LSTM Networks, MDN [8], and Attention Mechanism. The grouping of multiple algorithms to attain the desired result is usually stated to as Lego Effect in machine learning terms. Like already mentioned his model is not relatively feasible to imitate any particular style of handwriting.

After Alex Graves proposal, no big improvements have been made in handwriting synthesis. Generated Adversarial Networks can be assumed as major enhancements in the way we train Deep Neural Networks. These are developed very recently by Ian Goodfellow and his team. GANs are being applied to various deep learning problems to improve their accuracy or efficiency. But GANs are not pretty promising when used in conjunction with RNN. So, different approaches are developed by various other researchers. One of the popular and better one can be the C-RNN-GAN by Olof Mogren. Olof Mogren [2] applied these C-RNN-GANs to the music data and the results are observed to be better than the traditional RNN. GANs are still quite new in the field of Deep Learning and it has its individual pros and cons.

#### B. Gaps identified in the survey:

Contributions made to handwriting synthesis are very few in the field of machine learning. Classical machine learning algorithms aren't efficient enough to give good results for this application. So, Deep Learning is the only choice. Neural Networks are difficult to fine tune because they behave like a black box which doesn't give much detail about internal functionality. This can be good and bad at the identical time.

Google Brain Team makes the only biggest contribution in handwriting synthesis using Deep Learning. No proper mechanism has been derived to distinguish two different handwriting styles. Generated Adversarial Networks are quite new and achieving good results but they don't look promising in Recurrent Neural Networks. No active research on handwriting synthesis using GANs is going on.

### IV. PROPOSED WORK

#### A. Overview of proposed work:

Before describing the model, first, we will define the structure of handwriting data. Usually, handwriting data can either be Online or Offline data. Online in this framework means data is recorded as a order of pen-tip locations and Offline data means scanned images of handwriting document. In this project, all the generated and input handwriting data is online. Online data gives more information about stroke direction which can benefit the model to better understand the style. Extending the method to understand offline data too can be the future work of the project. All the training and test data is taken from IAM online handwriting database (IAM-OnDB). IAM-OnDB consists of handwritten lines of data gathered from 223 various writers using a e-smart whiteboard. Each training sample data consists a sequence of x and y coordinates along with eos parameter. The eos constraint specifies whether it is the termination of stroke or not. IAM-OnDB is separated into a training set, two validation sets, and a test set, containing respectively 5356, 1448, 1520 and 4859 handwritten lines of data are taken from 765, 172, 316 and 534 forms. In order to increase the amount of training data, we used the training set, test set and the larger of the validation groups for training and the smaller validation set for early-stopping.

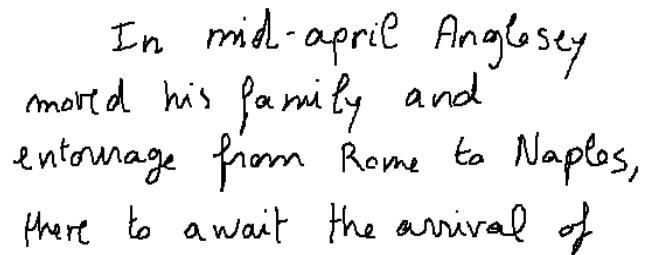


Fig.1: Training sample from IAM-OnDB

The main principal of the neural network system architecture is inspired from the Alex Graves handwriting synthesis model. As we are dealing with the temporal data, it is obvious that we need to use recurrent neural networks. But a traditional recurrent cell is not quite effective in storing long dependencies. For this purpose, Long Short-Term Memory cells are used in its place of traditional RNN cell. They maintain cell states and have longer memory by making use of various internal gates. With the help of LSTM cells, we can have long-term dependencies but one LSTM cell may not be so effective in abstracting the details of handwriting stroke. In order to have a deeper understanding of handwriting strokes to the network, multiple LSTM cells are stacked on top of each other to create a deep recurrent neural network. In our case, three LSTM layers are used. Now with this architecture, we can produce any handwritten text in some random style. Next section describes what methods were proposed to constrain the handwriting style. The rolled architecture figure shows only one timestep of the model and does not show the looped connections inside the model. In order to understand those connections, it is better to unroll the model and visualize it as multiple architectures interconnected to each other.

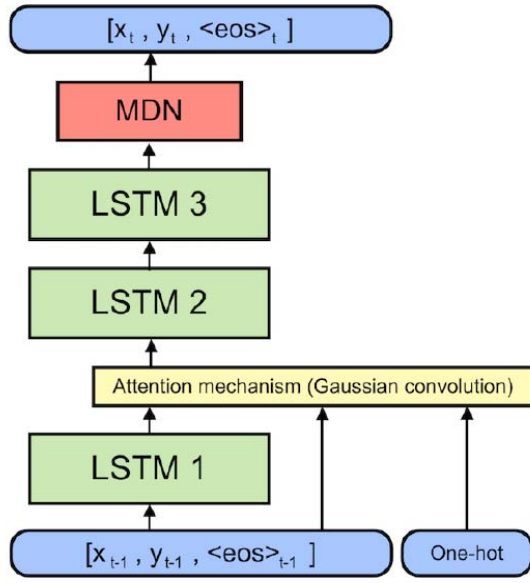


Fig.2: Complete rolled architecture of handwriting synthesis

A point to be considered is that when visualized in the unrolled architecture we observe various copies of the same model at different time instances but in reality, there won't be any various copies of the model but the same model will be looped internally at every timestep to act as a sequence of networks. So, like in unrolled architecture, we have all the LSTMs and attention mechanisms pass their hidden states to the next timestep [10]. The output of  $t-1$  step will be the input of the  $t$  step. Mixture Density networks don't have any loop as they are a regular neural network without any hidden states. Therefore, there will not be any hidden state transfer in the case of MDNs.

### B. Architecture of the Proposed System:

So far, the previous sections described how to synthesis handwriting by using Deep LSTM model with attention mechanism and Mixture Density Networks. In addition, here we introduce two methods to make the model better at imitating any handwriting style provided enough data and time to retrain.

#### Retraining the top LSTM cell:

One method is to use the same network architecture without any change and create multiple copies of the network that mimic a particular style on demand. Let's say the user provided sample handwriting style to the trained network, the approach is to make a copy of the currently trained parameters and re-train the copied network without modifying the original one. Retraining the whole network is not a good idea as it takes a substantial amount of time. So, instead of retraining the whole network we re-train only the top LSTM layer by fixing the parameters of lower layers fixed [3] [4]. After retraining the network with lower layers fixed we can use this model to generate handwriting data that mimic the particular style. While generating the results, we prime the original training sample to the network. Priming the training sample can make the network to continue the handwriting style instead of starting with null [9] [11].

With this approach, re-training is required for every new handwriting style. In addition, to make retraining a bit efficient Recurrent Batch Normalization [7] can be applied which can also affect the performance of the trained model.

#### Using GAN architecture:

Another approach is to make few changes in the network architecture by introducing a discriminative model. Generative Adversarial Nets [6] use an argumentative method by concurrently training 2 models G and D. A generative prototype G seizure the data, and a discriminative method D evaluation the possibility that a sample data came from the training data moderate than G. The training process for G is to increase the possibility of D making a error.

GANs typically resembles to a mini-max 2-player game. Generative model, in this case, will be our current network architecture and the Discriminative model will be a simple classifier using neural networks. The advantage of using this approach is discriminative model alone can be utilized to compare handwriting styles and can be used to identify a forgery. While retraining, the sample handwriting data will be provided to the Generator and Discriminator [5]. As Discriminator tries to compare with the original handwriting style both Generator and Discriminator will tend to adapt to the provided handwriting style in few training steps and the Generator can be later used to produce the handwritten text mimicking the user handwriting style. This approach may take more time to retrain but the results will be better in compared to the first approach.

### C. Proposed System Model:

In the above sections, we described the internal functionality of model descriptively. Here we will go into technical details and describe mathematical notations for each of the models. Below equations, describe the version of LSTM used in this project. H is applied by the following complex function:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (1)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (2)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (3)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \quad (4)$$

$$h_t = o_t \tanh(c_t) \quad (5)$$

Where  $\sigma$  is the logistic sigmoid function, and  $i$ ,  $f$ ,  $o$ , and  $c$  are individually the input, forget, output gates, cell & cell input initiation vectors, all of them are the similar size as the hidden vector  $h$ . The weight matrix resembles to have the clear meaning, for example,  $W_{hi}$  is the hidden input matrix,  $W_{xo}$  is the i/p-o/p gate matrix etc. Below equations describe how attention mechanism is implemented and will give an intuition for how the parameter  $\alpha$ ,  $\beta$ , and  $\kappa$  affect the window's behavior. These three parameters control the character window according to

$$(\hat{\alpha}_t, \hat{\beta}_t, \hat{\kappa}_t) = W_{h^1_p} h_t^1 + b_p \quad (6)$$

$$\alpha_t = \exp(\hat{\alpha}_t) \quad (7)$$

$$\beta_t = \exp(\hat{\beta}_t) \quad (8)$$

$$\kappa_t = \kappa_{t-1} + \exp(\hat{\kappa}_t) \quad (9)$$

Each of these parameters is output from a dense layer on top of the first LSTM which we then transform according to the above equations. From these parameters, we can construct the window as a convolution:

$$\phi(t, u) = \sum_{k=1}^K \alpha_t^k \exp\left(-\beta_t^k (\kappa_t^k - u)^2\right) \quad (10)$$

$$w_t = \sum_{u=1}^U \phi(t, u) c_u \quad (11)$$

Each input vector  $x_t$  comprises of a real-valued pair  $x_1, x_2$  that defines the pen offset from the previous input, along with a binary  $x_3$  that has value 1 if the vector ends a stroke (that is, if the pen was lifted off the board before the next vector was recorded) and value 0 otherwise.

A mixture of bivariate Gaussians was used to predict  $x_1$  and  $x_2$ , while a Bernoulli distribution was used for  $x_3$ . Each output vector  $y_t$ , therefore, comprises of the end of stroke probability  $e_t$ , along with a set of means  $\mu^j$ , standard deviations  $\sigma^j$ , correlations  $\rho^j$  and mixture weights  $\pi^j$  for the  $M$  mixture components. That is

$$x_t \in \mathbb{R} \times \mathbb{R} \times \{0, 1\} \\ y_t = \left(e_t, \{\pi_t^j, \mu_t^j, \sigma_t^j, \rho_t^j\}_{j=1}^M\right) \quad (12)$$

Note that the mean and standard deviation are two-dimensional vectors, whereas the component weight, correlation, and end-of-stroke probability are scalar. The vectors  $y_t$  are obtained from the network outputs  $y^*$ , where

$$\hat{y}_t = \left(\hat{e}_t, \{\hat{w}_t^j, \hat{\mu}_t^j, \hat{\sigma}_t^j, \hat{\rho}_t^j\}_{j=1}^M\right) = b_y + \sum_{n=1}^N W_{h^n_y} h_t^n$$

$$\begin{aligned} e_t &= \frac{1}{1 + \exp(-\hat{e}_t)} & \implies e_t \in (0, 1) \\ \pi_t^j &= \frac{\exp(\hat{\pi}_t^j)}{\sum_{j'=1}^M \exp(\hat{\pi}_t^{j'})} & \implies \pi_t^j \in (0, 1), \sum_j \pi_t^j = 1 \\ \mu_t^j &= \hat{\mu}_t^j & \implies \mu_t^j \in \mathbb{R} \\ \sigma_t^j &= \exp(\hat{\sigma}_t^j) & \implies \sigma_t^j > 0 \\ \rho_t^j &= \tanh(\hat{\rho}_t^j) & \implies \rho_t^j \in (-1, 1) \end{aligned}$$

The probability density  $\Pr(x_{t+1}|y_t)$  of the next input  $x_{t+1}$  given the output vector  $y_t$  is well-defined as follows:

$$\Pr(x_{t+1}|y_t) = \sum_{j=1}^M \pi_t^j \mathcal{N}(x_{t+1}|\mu_t^j, \sigma_t^j, \rho_t^j) \begin{cases} e_t & \text{if } (x_{t+1})_3 = 1 \\ 1 - e_t & \text{otherwise} \end{cases} \quad (13)$$

Where

$$\mathcal{N}(x|\mu, \sigma, \rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[\frac{-Z}{2(1-\rho^2)}\right] \quad (14)$$

With

$$Z = \frac{(x_1 - \mu_1)^2}{\sigma_1^2} + \frac{(x_2 - \mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1 - \mu_1)(x_2 - \mu_2)}{\sigma_1\sigma_2} \quad (15)$$

Now the loss function of the entire model can be defined as follows:

$$\mathcal{L}(\mathbf{x}) = \sum_{t=1}^T -\log\left(\sum_j \pi_t^j \mathcal{N}(x_{t+1}|\mu_t^j, \sigma_t^j, \rho_t^j)\right) - \begin{cases} \log e_t & \text{if } (x_{t+1})_3 = 1 \\ \log(1 - e_t) & \text{otherwise} \end{cases} \quad (16)$$

## V. IMPLEMENTATION AND ANALYSIS OF PROPOSED SYSTEM

The product that is developed in this research paper will be a prototype using web technologies that show a demo of how to apply the methodology in real time applications. The main model is coded separately and a REST API is developed which interacts with the trained model to get the results. Web interface uses this API to call the model and displays the results in the UI. The final prototype presented consists of a web interface, a REST API, TensorFlow based trained model. REST API relates with the model. And web interface uses REST API. In future works, the same API can be used to develop different applications. All the three modules are independent to each other which gives a better scope to scale the product.

### A. Proposed Work Features:

1. User can give an input of custom text which can be converted into handwriting data.
2. User can adjust the bias parameter of the model which can change the neatness of handwriting data.
3. The user can adjust the number of time steps parameters manually or can leave it automatic.
4. The user can select one of the handwriting styles provided in the interface.
5. Multiple samples of output are generated to let the user select the best one.
6. The number of samples of output can be provided by the user.
7. The user can switch to debug mode to visualize various parameters in graphical format.

### B. Methodology:

Previous sections deal with the architecture of the model and this section deals with how the model was implemented and the technical difficulties faced during implementation. TensorFlow is one of the famous Deep Learning frameworks in Python. Moreover, for this project, TensorFlow is used to design the model.

As described in the architecture we have 3 layers of LSTMs and one mixture density network (MDN) and an attention mechanism to restrict it to a specific text. LSTMs are already accessible in TensorFlow and those APIs were utilized in this project. The first layer of LSTM was implemented and then attention mechanism part is hard coded following the mathematical equations and it is attached to the first layer of LSTM. The outputs of attention mechanism are then passed to second layer LSTM and then to final layer LSTM. Now the output of final layer LSTM is attached to Mixture Density Network which is again hard coded in the TensorFlow as there are no built-in APIs are available for those networks. The whole model is designed in Python in such a way that any hyperparameter can be changed easily at any instance. Now the data needs to be prepared for training. Dataset cannot be used directly as there are few human errors. All those errors are removed by using various data cleaning techniques and each line is trimmed to 750 timesteps for consistency.

The cleaned dataset is stored in a file and it is configured in such a way that it provides a batch of data at every call and those batches are randomized to avoid overfitting issues. Next step is to train the model using the prepared data. It is not quite practical to train the model on normal laptops as it will take more than 3-4 days to complete the entire training process. So, to reduce the wastage of time, we bought AWS GPU instances temporarily to train the data. Entire model and data are uploaded to AWS cloud instance to prepare for training. It took an average of 1 day to completely train the model which is considerably better than training on the personal laptop.

Tuning the hyperparameters is one of the most difficult parts of the whole project. As there are multiple networks in the model, the number of hyperparameters increased which increased the difficulty a little more. At the beginning, the outcomes were very unsatisfactory because of incorrect hyperparameters. After a lot of trial and errors, results were a bit satisfactory. But the outcomes were still not as expected. Then after more than ten retrains we identified that problem is not with the hyperparameters but with the data. The amount of data in the dataset is quite low which resulted in noise in the results.

In the order to avoid that we need more data which is not available on the internet. Even if we take manual samples of handwriting data it won't create a huge impact because in the very limited time we cannot collect more than 1000 lines of data which is actually quite negligible when related to the existing dataset. Therefore, we used the same dataset and tried various tricks to make the results as good as possible.

After the model is trained, we created an API via flask framework in Python. This API provides two endpoints in which route endpoint gives the landing page information and the other endpoint is applied to submit metadata for sample generation. After developing the API, a simple user interface is designed using web technologies and integrated the UI to the API.

## VI. RESULTS AND DISCUSSION

### A. Snapshots of Proposed Work:

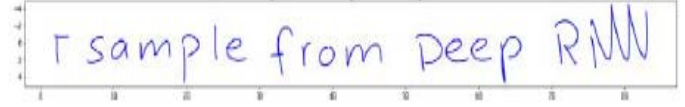


Fig.3: Generated sample of text "A sample from Deep RNN"

## Handwriting Synthesis Demo

Text to be converted to handwriting data

Number of samples to generate

Advanced Settings

Generate

Fig.4: Basic web UI for the model prototype

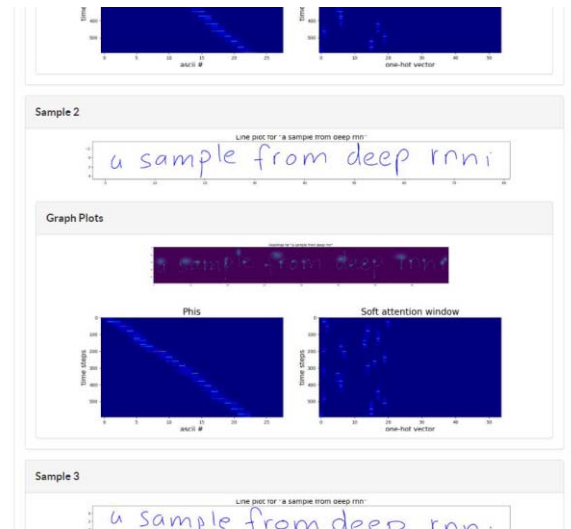


Fig.5: Results panel in web prototype

Below snapshots shows how the model generates handwriting styles from the given two sample input texts with some random handwriting style.



Fig.6: Generated sample of text "from his travels"



Fig.7: Original handwriting sample from author

The one major drawback of generative models is it is difficult to have performance metrics. The same case is there in our prototype. As of now, there is no proper mechanism to determine which generated sample is better. Identifying the best model out of generated models can alone be a separate research topic. The one and only way to evaluate the performance is to manually check the results and humans should rate the results accordingly.

## VII. CONCLUSION AND FUTURE WORK

Handwriting generation is a complicated task and it is even more difficult to mimic the particular style. With the explained model, we can achieve satisfying results. Results of the model totally depend on its hyperparameters. Tuning them properly is necessary. Few limitations observed with the model is that it cannot generate longer strings at once because of LSTMs limit. And also, the model needs more amount of data for efficient results. In future, these limitations can be avoided with the advancements in the RNNs and model can also be improved to mimic any style with less training data. This method is not limited to handwriting data. It can be functional to any sequential data with few tweaks. Moreover, in future, this designed model can be applied in a much more useful real-time application.

## REFERENCES

- [1] Alex Graves, "Generating Sequences with Recurrent Neural Networks", arXiv preprint arXiv:1308.0850v5 [cs.NE], 2014.
- [2] Olof Mogren, "C-RNN-GAN: Continuous recurrent neural networks with adversarial training", arXiv preprint arXiv:1611.09904v1 [cs.AI], 2016.
- [3] Alex Lamb, Anirudh Goyal, Ying Zhang, et al., "Professor Forcing: A New Algorithm for Training Recurrent Networks", arXiv preprint arXiv:1610.09038v1 [stat.ML], 2016.
- [4] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, et al., "Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks", arXiv preprint arXiv:1506.03099v3 [cs.LG], 2015.
- [5] Arna Ghosh, Biswarup Bhattacharya, Somnath Basu Roy Chowdhury, "Handwriting Profiling using Generative Adversarial Networks", arXiv preprint arXiv:1611.08789v1 [cs.CV], 2016.
- [6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, et al., "Generative Adversarial Nets", arXiv preprint arXiv:1406.2661v1 [stat.ML], 2014.
- [7] Tim Cooijmans, Nicolas Ballas, Caesar Laurent, et al., "Recurrent Batch Normalization", arXiv preprint arXiv:1603.09025v4 [cs.LG] 12 Apr 2016.
- [8] Christopher M. Bishop, "Mixture Density Networks", Neural Computing Research Group, Aston University, 1994.
- [9] Marcin Andrychowicz, Misha Denil, et al., "Learning to learn by gradient descent by gradient descent", arXiv preprint arXiv:1606.04474v2 [cs.NE] 30 Nov 2016.
- [10] Kandala H., Tripathy B.K., Manoj Kumar K. (2018) A Framework to Collect and Visualize User's Browser History for Better User Experience and Personalized Recommendations. In: Satapathy S., Joshi A. (eds) Information and Communication Technology for Intelligent Systems (ICTIS 2017) - Volume 1. ICTIS 2017. Smart Innovation, Systems and Technologies, vol 83. Springer, Cham.
- [11] S. A. Razvi, S. Neelima, C. Prathyusha, G. Yuvasree, C. Ganga and K. M. Kumar, "Implementation of graphical passwords in internet banking for enhanced security," 2017 International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2017, pp. 35-41.