IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 10, No. 2, June 2021, pp. 374~381
ISSN: 2252-8938, DOI: 10.11591/ijai.v10.i2.pp374-381

Implementation of generative adversarial networks in HPCC systems using GNN bundle

Ambu Karthik[1], Jyoti Shetty[2], Shobha G.[3], Roger Dev[4]
[1,2,3]Computer Science and Engineering, R. V. College of Engineering, Bangalore, Karnataka, India
[4]Sr. Architect, Machine Learning, Lexis Nexis Risk Solution, Alpharetta, Georgia, United States

Article Info

Article history:

Received Oct 24, 2020
Revised Mar 16, 2021
Accepted Mar 30, 2021

Keywords:

Artificial neural networks
Deep learning
Enterprise control language
GANs
HPCC systems

ABSTRACT

HPCC systems, an open source cluster computing platform for big data analytics consists of generalized neural network bundle with a wide variety of features which can be used for various neural network applications. To enhance the functionality of the bundle, this paper proposes the design and development of generative adversarial networks (GANs) on HPCC systems platform using ECL, a declarative language on which HPCC systems works. GANs have been developed on the HPCC platform by defining the generator and discriminator models separately, and training them by batches in the same epoch. In order to make sure that they train as adversaries, a certain weights transfer methodology was implemented. MNIST dataset which has been used to test the proposed approach has provided satisfactory results. The results obtained were unique images very similar to the MNIST dataset, as it were expected.

Corresponding Author:

Ambu Karthik
Department of Computer Science and Engineering
RV College of Engineering
Bangalore, Karnataka, India
Email: ambukarthik.cs18@rvce.edu.in

1. INTRODUCTION

The open source high performance computing cluster (HPCC) system is a cluster computing platform used for big data applications [1], [2]. For managing the data flow in HPCC while preserving the cluster-computing capabilities, a special language called enterprise control language [3]-[5], abbreviated as ECL was developed using C++. This language has many functions, data types, language embedding and various bundles as support for the user's convenience. This makes it a powerful language for big data processing and computation. HPCC Systems platform also has a tool called ECL watch [6], [7] on the cluster-computing server which enables you to manage multiple nodes, input files so they are accessible to all nodes (called spraying), output files by merging processed data from all nodes (called despraying) and to see the various outputs and logs of the process that is run [8].

Due to the cluster computing scalability of the platform, it provides for processing many petabytes of data [9], making it immensely powerful for its applications in machine learning [10]. For this, a bundle for performing machine learning operations on the ECL language was developed and is available as an open source bundle hosted on github [11]-[13]. This was further used to develop an ECL bundle capable of using neural networks which is called generalised neural networks [14].

Technology over the previous decade developed in statistical learning using various machine learning metrics and deep learning neural networks [15], through which the feature recognition capabilities for images [16], audio files [17] or even text [18] improved to amazing levels. In the last decade, there came

a marvellous idea to utilise the feature recognition capabilities of these models to be able to generate data. Utilising a minimax function with a pair of neural networks behaving as adversaries gave birth to a superior neural network framework capable of generating completely new data with similar patterns, which due to their behaviour was given the name generative adversarial networks, abbreviated as GANs.

GANs are a pair of models which behave as adversaries to each other in order to compete and learn from experience, and generate artificial data which is similar to the data given to the model while training. The generator model forms data patterns from a latent random variable, which is also simply called noise that is generated in each epoch. The discriminator model uses data fed into it in order to train, followed by using it to back-propagate to the generator that uses the feedback from the discriminator network to get better at producing artificial data of the kind input into the discriminator. The framework explained above is shown in Figure 1. This framework of neural networks was first implemented and presented by Goodfellow *et al*. [19].

As HPCC systems was a distributed computing system, there were challenges in implementing neural networks. Neural networks have been implemented in the GNN bundle described above which uses Tensorflow [20] in the back-end for the neural network computations that are needed to be performed. GANs in HPCC systems were implemented using the GNN bundle by the team working on the same. The paper focuses on how it was implemented along with the results.

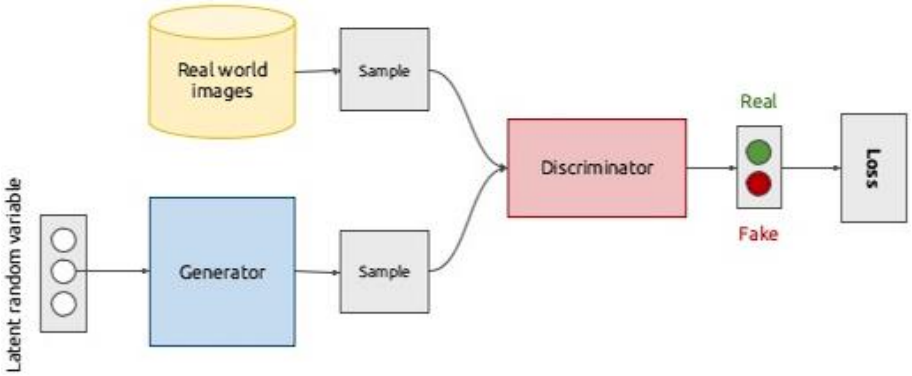

Figure 1. Architecture of GAN

2. IMPLEMENTATION OF GAN FOR HPCC SYSTEMS

2.1. Input phase

The input of data given to the GAN needs to be pre-processed so as to suit the needs of the GNN bundle [21]. All the data that is required to be given for training needs to be transformed into a dataset defined as *TensData* which has the record structure: [indexes, value]. *TensData* simulates the representation of an n-dimensional array in the numpy [22] module with the indexes and values together to represent each position in an array.

All the data required to be used need to be transformed into the structure of *TensData* using various transformation operations in ECL. For a dataset that is readily available as CSV or XLS, it is possible to easily transform these. For inputting images into the neural network, the Image module in GNN may be used. It provides straightforward functions to perform the operation described above for input.

2.2. Definition of models

Generative adversarial networks by the definition are required to have 2 neural network models, namely the generator and the discriminator acting as adversaries to each other. Here, the models are required to be given by the user. The user decides on the generator and discriminator model definitions and adds definition of each layer into a set of strings. Each model also has a compile definition which is typically a string containing the compile function. Both the models usually must have just one compile definition together. Examples for these are in section 3.

Given these definitions, generator and discriminator are defined using the GNNI function *DefineModel*(..). Along with this, a combined model which is the generator stacked over the discriminator needs to be defined. Definition of this combined model is done by appending generator layer definitions with discriminator layer definitions, with each layer in discriminator set to non-trainable. This is defined using the same compile definition as the two models.

2.3. Training phase

Each epoch of GAN training happens in the following stages:

- Generate data from random latent variables using generator.
- Fit real data with the label 1 to discriminator.
- Fit generated data with the label 0 to discriminator.
- Fit random latent variable with the label 1 to combined model. Since discriminator is non-trainable, generator is trained with random latent variables while fitting as per discriminator weights.

Here, the modified weights from discriminator need to be transferred to the combined model so that the Fitting of the data to discriminator is effective in the epoch. Along with this, the trained weights of the combined model need to be given to the generator to predict from noise. If this isn't done, the training of the previous epoch is rendered useless.

In Keras API [23] of Python 3.6, it is possible to make the combined model by adding the generator and discriminator as objects which link their layer weights. In ECL, we need to manually transfer weights to simulate this. For this, a weights transfer mechanism was implemented using *GetWeights*(..) and *SetWeights*(..) functions in the GNNI module of the GNN bundle. Figure 2 explains the weights transfer methodology for an n^{th} epoch. Here, both the generator and discriminator improve in each epoch to produce better results iteratively. This is the most important phase of the model as the results are produced mainly due to this step, and it also takes the longest time to perform.

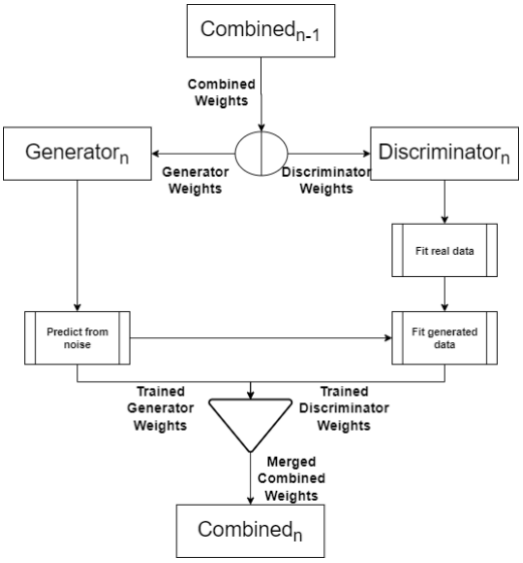

Figure 2. Weights transfer methodology

2.4. Output phase

After the epoch-wise training of the GAN model is performed, both the generator and discriminator models are returned from the module. These models can both be given their respective inputs for checking the output or for an application.

2.4.1. Generator model

Generator model takes an input of a random latent variable, and if required, multiple sets of the random variable to predict data. These predictions are obtained as a *t Tensor*, whose data can be extracted using the *GetData*(..) function in the Tensor module. The extracted data can be stored or output to understand the generator's performance. For an output of images, the Image module in GNN bundle contains functions to easily output the images as is required by the user.

2.4.2. Discriminator model

Discriminator model takes an input of data to classify them as fake or real data. This capability of the discriminator helps in applications where fraudulent data are to be detected when the number of fraudulent cases are smaller. Machine learning evaluation metrics like [24] AUC, F1 score. Can be performed on the output dataset with expected to see how the discriminator has performed

3. TESTED GAN MODELS

There were two prominent models in GAN which were sequential in nature for pure generation of data. The output of the generated dataset was checked after subsequent epochs to verify the training that was performed, which are shown in section 4. These models were obtained from a github repository called "Keras-GAN" by Noren [25]. The generator and discriminator models of these are described as follows:

3.1. Simple dense layer based generative adversarial networks

This model is described as a GAN architecture with purely dense based networks with no special transformations except for batch normalisation, dropout and activation layers. This model tries to see patterns in the image as a whole set by predicting each pixel as per the dense layer weights obtained after each training epoch. The generator and discriminator models of the simple GAN can be seen in Figure 3.

As can be seen in Figure 3(a) and Figure 3(b), the network consists of purely dense based models for data manipulation which enables the GAN to be trained pixel-wise. Due to this, the training is much slower than it's other counterparts and the generated images aren't smooth, due to the pixels generated being different. The output of the following model can be seen in section 4.1. The results in the next model, deep convolutional generative adversarial networks in section 3.2., abbreviated as DCGAN, can be seen to provide smoother images compared to Simple GAN.

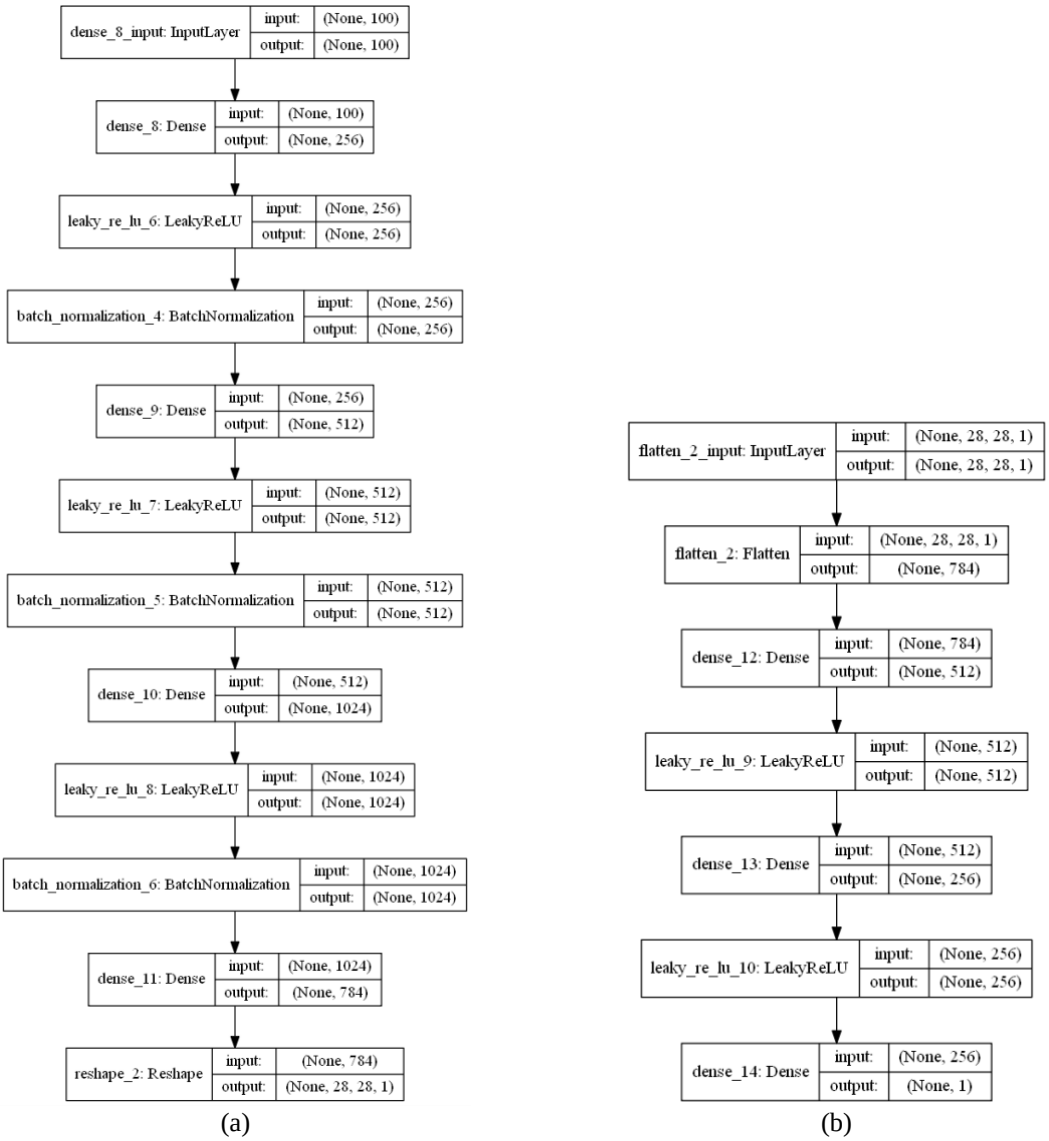

Figure 3. Simple dense layer based generative adversarial networks; (a) generator model, (b) discriminator model

3.2. Deep convolutional generative adversarial networks

This model is described as a GAN architecture with convolutional transformations in the neural network. This model consists of multiple convolutional layers which helps the models in recognizing various features in the image. The generator and discriminator models of the deep convolutional GAN can be seen in Figure 4.

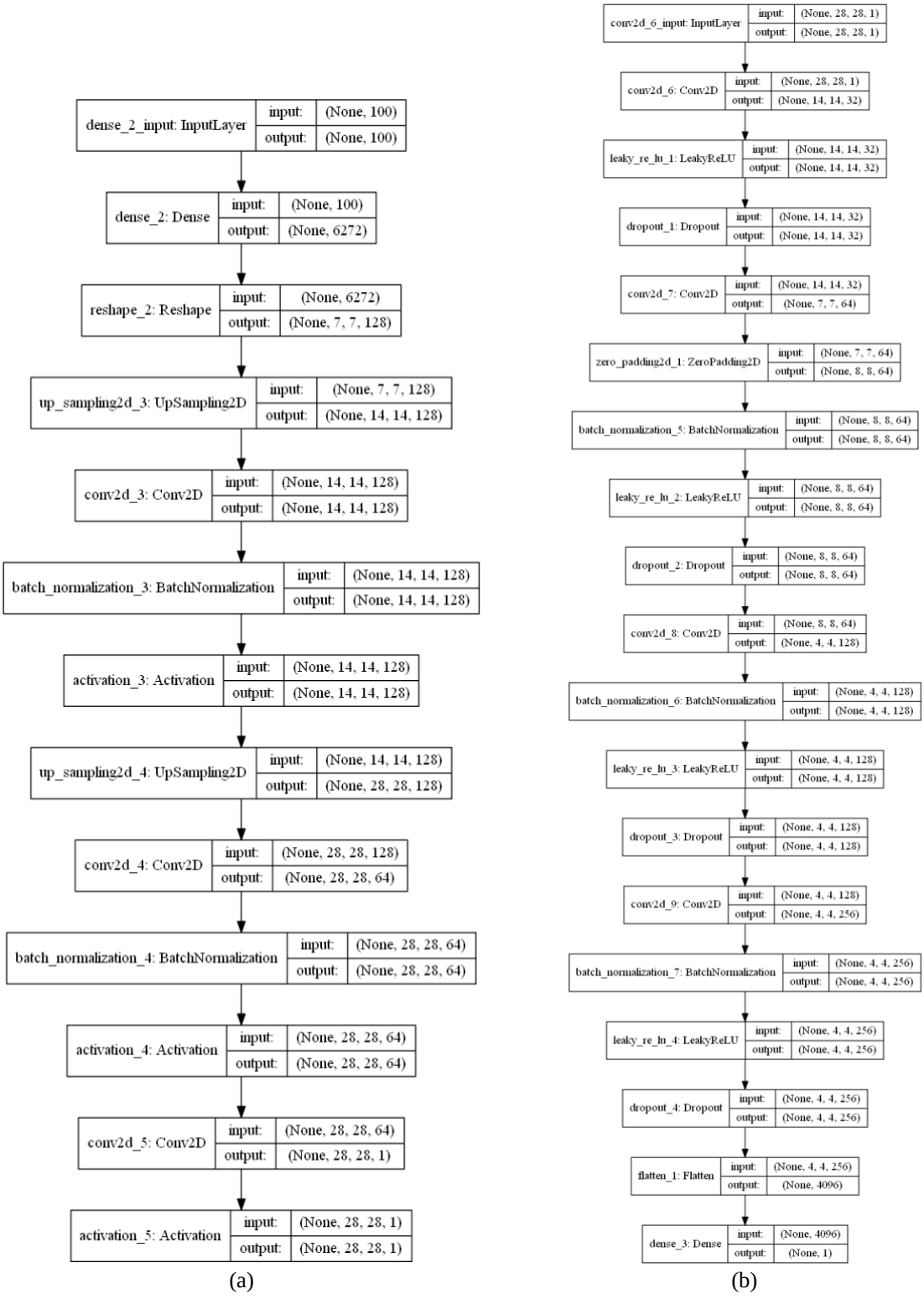

Figure 4. Deep convolutional generative adversarial networks; (a) generator model, (b) discriminator model

As can be seen in Figures 4(a) and 4(b), the network consists of convolutional layers along with dropout layers which increases the feature recognition capability of the network. Due to the improved feature recognition capability, the generated images are much smoother and more unique in nature. The outputs of the following model can be seen in section 4.2. It can be observed that DCGAN gives smoother and more unique images as compared to simple GAN.

4. RESULTS

Using the above implementation, two architectures of GANs that were described in section 3. Have been tested on HPCC Systems using the GAN implementation and executed. Snapshots of the generator predictions were taken after 2000, 3000 and 4000 epochs respectively as shown in Figure 5(a), Figure 5(b), Figure 5(c) and Figure 6(a), Figure 6(b), Figure 6(c) which shows us how the model gets better as it's trained. The outputs of the architectures are as follows:

4.1. Simple generative adversarial networks

It can be noticed in Figure 5 that simple GAN produces results very similar to the MNIST dataset [26], [27], but there are noisy pixels around them due it being a purely dense network relying on the density of each tensor. This is rectified by deep convolutional GAN which produces smoother images, with very unique images as compared to simple GAN.

(a) (b) (c)

Figure 5. Generated images from simple GAN; (a) epoch 2000, (b) epoch 3000, (c) epoch 4000

4.2. Deep convolutional generative adversarial networks

The images produced by deep convolutional GAN as can be seen in Figure 6 are much more smoother and unique in nature compared to simple GAN. The convolutional layers implemented in the model detect various features of the MNIST dataset in order to put them together to generate similar digits. The digits generated by both the GAN models get better with more number of training epochs for the GAN.

(a) (b) (c)

Figure 6. Generated images from deep convolutional GAN; (a) epoch 2000, (b) epoch 3000, (c) epoch 4000

5. CONCLUSION

Generative adversarial networks were successfully implemented on the HPCC systems platform to be able to execute any Keras sequential GAN model. The implementation has been stable for the models tested on the platform. This implementation was made into an ECL bundle to be installed on any system containing the required dependencies. The GAN bundle so created depends on *ML Core* and *GNN* bundles.

So, to use the following implementation, these dependencies need to be installed on the platform. With the implementation of the following, there is room for further research over GANs on HPCC systems as the distributed computing architecture makes it easier to handle big data. This implementation may also be expanded over to develop various flavours of GANs which assist in many interesting real world applications.

6. FUTURE ENHANCEMENTS

The implementation for GAN in HPCC systems has much room for improvement with many more features possible with the proposed architecture above. As mentioned above, this implementation of GAN works for purely sequential models which eliminates the possibility of multiple inputs or outputs as is required for a few types of GAN models. So, an architecture for GAN could be implemented using a GNN neural network model similar to the Keras functional Model API. There is quite a confusion for the user as to how the combined model looks in case the user runs into an error while using the GAN module. Visualising neural network models so as to understand the model better would be very useful. The debugging of the GAN module could be strange as the error which is logged isn't usually the cause of the error. Improving these debugging prospects of GAN to efficiently track the issue so the user can fix errors would reduce the time taken by the user in solving an error.

ACKNOWLEDGMENT

We would like to thank Lexis-Nexis and R.V. College of Engineering for their active support, encouragement and assistance in the following project on HPCC systems. The interfacing documentation and the set of clusters used to test the system were provided by Lexis-Nexis. R.V. College of Engineering provided all the necessary infrastructure support for the project.

REFERENCES

[1] HPCC Systems, "About HPCC Systems," 2020, [Online] Available: https://hpccsystems.com/about. [Accessed June 30, 2020]
[2] M. Anthony and C. Arjuna, (2011) Introduction to HPCC (High Performance Computing Cluster). [Online] Available: https://cdn.hpccsystems.com/whitepapers/wp_introduction_HPCC.pdf. [Accessed June 30, 2020]
[3] HPCC System, "ECL Language Reference", 2020, [Online] Available:https://cdn.hpccsystems.com/releases/CE-Candidate-7.12.16/docs/EN_US/ECLLanguageReference_EN_US-7.12.16-1.pdf. [Accessed June 30, 2020]
[4] HPCC Systems: Enterprise Control Language - An Overview (2011) [Online]. Available: https://cdn.hpccsystems.com/whitepapers/wp_ecl_overview.pdf. [Accessed June 30, 2020]
[5] ECL Programmer's Guide (2020) [Online]. Available: https://cdn.hpccsystems.com/releases/CE-Candidate-7.12.36/docs/EN_US/ECLProgrammersGuide_EN_US-7.12.36-1.pdf. [Accessed June 30, 2020]
[6] L. Xu, E. Muharemagic, F. Villanustre, and A. Apon, "ECL-watch: A big data application performance tuning tool in the HPCC systems platform," *2017 IEEE International Conference on Big Data (Big Data)*, Dec. 2017. doi:10.1109/bigdata.2017.8258263.
[7] Using ECL Watch. [Online]. Available: https://cdn.hpccsystems.com/releases/CE-Candidate-6.2.0/docs/The_ECL_Watch_Manual-6.2.0-1.pdf. [Accessed June 30, 2020]
[8] HPCC Systems. (2017) HPCC System Administrator's Guide. [Online]. Available: https://cdn.hpccsystems.com/releases/CE-Candidate-6.2.0/docs/HPCCSystemAdministratorsGuide-6.2.0-1.pdf. [Accessed July 1, 2020]
[9] HPCC Systems: Models for Big Data (2020) [Online]. Available: https://cdn.hpccsystems.com/whitepapers/wp_models_for_big_data.pdf. [Accessed July 1, 2020]
[10] A. Suryanarayanan, A. Chala, L. Xu, G. Shobha, J. Shetty, and R. Dev, "Design and Implementation of Machine Learning Evaluation Metrics on HPCC Systems," *2019 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS)*, Dec. 2019. doi:10.1109/csitss47250.2019.9031056.
[11] HPCC Systems, "ML Core Bundle source code," 2020, [Online] Available: https://www.github.com/hpcc-systems/ML Core. [Accessed July 1, 2020]
[12] Machine Learning Demystified (2018) [Online]. Available: https://hpccsystems.com/blog/machine-learning-demystified. [Accessed July 2, 2020]
[13] Using HPCC Systems Machine Learning (2018) [Online]. Available: https://hpccsystems.com/blog/HPCC-Sytems-Machine-Learning. [Accessed July 2,2020]
[14] HPCC Systems, "Generalised Neural Networks Bundle source code", 2020, [Online] Available: https://www.github.com/hpcc-systems/GNN. [Accessed July 2, 2020]
[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. doi:10.1109/5.726791.
[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017. doi:10.1145/3065386.

[17] Sigtia, Siddharth, Nicolas Boulanger-Lewandowski, and Simon Dixon, "Audio Chord Recognition with a Hybrid Recurrent Neural Network," ISMIR. 2015. Available: http://eecs.qmul.ac.uk/~simond/pub/2015/SigtiaBoulanger-LewandowskiDixon-ISMIR2015-Chord.pdf. [Accessed July 10, 2020]
[18] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent Convolutional Neural Networks for Text Classification", *AAAI*, vol. 29, no. 1, Feb. 2015. Available: https://ojs.aaai.org/index.php/AAAI/article/view/9513. [Accessed July 10, 2020]
[19] Goodfellow, Ian J., *et al*., "Generative adversarial networks." arXiv preprint arXiv:1406.2661 (2014). Available: https://arxiv.org/pdf/1406.2661v1.pdf. [Accessed July 10, 2020]
[20] Abadi, Martín, *et al*., "Tensorflow: A system for large-scale machine learning." 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16). 2016. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi. [Accessed July 10, 2020]
[21] HPCC Systems, "Generalised Neural Networks Bundle documentation", 2020, [Online] Available: https://d2wulyp08c6njk.cloudfront.net/pdf/ml/GNN.pdf. [Accessed July 11, 2020]
[22] Oliphant, Travis E. A guide to NumPy. Vol. 1. USA: Trelgol Publishing, 2006. Available: https://ecs.wgtn.ac.nz/foswiki/pub/Support/ManualPagesAndDocumentation/numpybook.pdf. [Accessed July 10, 2020]
[23] Gulli, Antonio, and Sujit Pal, Deep learning with Keras. Packt Publishing Ltd, 2017. Available: https://www.iaa.edu.jo/sites/default/files/webform/cvs/pdf-deep-learning-with-keras-antonio-gulli-sujit-pal-pdf-download-free-book-6083289.pdf. [Accessed July 10, 2020]
[24] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso, "Machine Learning Interpretability: A Survey on Methods and Metrics," *Electronics*, vol. 8, no. 8, p. 832, Jul. 2019. doi:10.3390/electronics8080832.
[25] Erik Linder Noren, Github, "Keras-GAN", 2020. [Online] Available: https://github.com/eriklindernoren/Keras-GAN. [Accessed July 12, 2020]
[26] Yann LeCun, "The MNIST database of handwritten digits", 2020, [Online] Available: http://yann.lecun.com/exdb/mnist/. [Accessed July 12, 2020]
[27] A. Baldominos, Y. Saez, and P. Isasi, "A Survey of Handwritten Character Recognition with MNIST and EMNIST," Applied Sciences, vol. 9, no. 15, p. 3169, Aug. 2019. doi:10.3390/app9153169.