Deep Learning-based Phase Prediction of High-Entropy Alloys

Deep Learning-based Phase Prediction of High-Entropy Alloys

Zeyad Yousif Abdoon Al-Shibaany[1, 2], Nadia Alkhafaji[*, 3, 4], Yaser Al-Obaidi[5], Alaa Abdulhasan Atiyah[6]

[1]Cardiff School of Engineering, Cardiff University, Cardiff CF24 3AA, UK.
[2]Biomedical Engineering Department, University of Technology - Iraq, Iraq.
[3]Faculty of Computing, Engineering, and Science, University of South Wales, UK.
[4]Laser and Optoelectronics Engineering Dept., University of Technology - Iraq, Iraq.
[5]Hyundai Engineering Co. Ltd. Iraq Branch, Iraq.
[6]Materials Engineering Department, University of Technology - Iraq, Baghdad, Iraq.

[*]Corresponding author: nadia.a.alkhafaji@gmail.com

Abstract. High-entropy alloys (HEAs) offer a new approach to the design of superior metallic materials, wherein alloys are based on multiple principal elements rather than just one. Deep Neural Networks (DNNs), machine learning tools that are efficiently used for prediction purposes, are transforming fields, from speech recognition to computational medicine. In this study, we extend DNN applications to the field phase prediction of high-entropy alloys. Using the built-in capabilities in TensorFlow and Keras, we train DNNs with different layers and numbers of neurons, achieving a 90% prediction accuracy. The DDN prediction model is examined in detail with different datasets to verify model robustness. Due to the high cost of HEAs and in order to save time, it is important to predict phases in order to design alloy composition. Through this study, we show trained DNNs to be a viable tool for predicting the phases of high-entropy alloys, where 90% phase prediction accuracy was achieved in this work.

Keywords: Deep learning, machine learning, artificial intelligence, prediction, high-entropy alloys.

1. Introduction

Machine learning (ML), a branch of artificial intelligence (AI), is based on the concept that computers (machines) can use data and learn to identify patterns and make predictions. These predictions and pattern-identification capabilities can be done with minimal programming instructions. The high computing power in ML comes from the use of Artificial Neural Networks (ANNs), which are basically computing systems with several interconnected processing elements called neurons that map an array of input variables (features) to one or more outputs (labels). Figure 1 shows schematic of an ANN where the circles refer to the neurons (the processing elements of ANNs) and the arrows refer to the connections between these neurons; the direction of the arrow shows the direction of the data flow through the network.

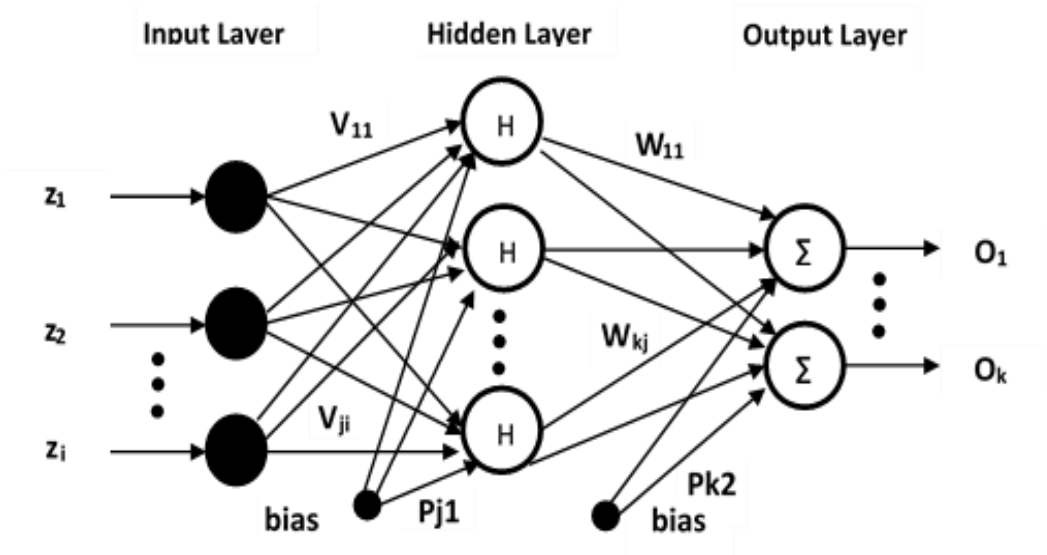

Figure 1: Schematic of Artificial Neural Networks [1]

The parameters of the ANN shown in Figure 1 are listed in Table 1 below.

Table 1. List of the ANN parameters [1].

Parameter	Description
ni	Number nodes in the input layer
nj	Number of nodes in the hidden layer
nk	Number of nodes in the output layer
$\mathbf{V_{ji}}$	Weight between (j) hidden node and (i) input node
$\mathbf{W_{kj}}$	Weight between (k) output node and (j) hidden node
$\mathbf{P_{j1}}$	Weight between (j) hidden node and bias node
$\mathbf{P_{k2}}$	Weight between (k) output node and bias node
h	Hidden layer node
$\mathbf{z_i}$	Input of (i) input node
$\mathbf{O_k}$	Output of (k) output node
H	Activation function

The following steps explain the calculation of the output of ANN:

1. The input data are applied to the input layer. In this layer, the data will not be processed.
2. The data are then passed to the hidden layer, and the output of each neuron at the hidden layer is calculated as follows:

$$hnet_j = H(h_j) \tag{1}$$

Where h_j represents the j[th] neurons in the hidden layer, and $hnet_j$ represents the j[th] neurons in the hidden layer.

$$h_j = \sum_{i=1}^{ni} z_i \times V_{ji} + bias \times P_{j1} \tag{2}$$

3. The output of the output layer is then calculated as follows:

$$O_k = \sum_{j=1}^{nj} hnet_j \times W_{kj} + bias \times P_{k2} \quad (3)$$

It is easy to determine the number of neurons at the input layer of any neural network by simply counting the number of independent variables that will be fed into the neural network. The same idea can be applied to the output layer, where the number of neurons is equal to the number of dependent variables. However, choosing the number of neurons at the hidden layer represents a significant challenge that still attracts the interest of many researchers. A binary search technique was used to estimate the number of neurons in the hidden layer where the number chosen was 1, 2, 4, 8, 16, 32, 64, and [2]. However, the binary search method in this work achieved up to 80% accuracy. A comprehensive systematic review of the techniques to count the number of neurons in the hidden layer was conducted by [3]. Based on the literature and for the purposes of this work, Table 2 lists the techniques used to count the number of neurons in the hidden layer in this work where:
Ni: number of neurons in the input layer
No: number of neurons in the output layer
Nh: number of neurons in the hidden layer
Nt: number of training pairs (size of the training dataset)

Table 2. List of techniques for counting the number of neurons in the hidden layer.

No.	Technique	Reference
1	$Nh = \frac{\sqrt{1+8Ni}-1}{2}$	[4]
2	$Nh = Ni - 1$	[5]
3	$Nh = \frac{1}{2} \cdot \frac{Nt}{Ni\ logNt}$	[6]
4	$Nh = \frac{Nt}{Ni}$	[6]
5	$Nh = \sqrt{Ni\ No}$	[7]
6	$Nh = log_2(Ni+1) - No$	[8]
7	$Nh = \frac{4\,Ni^2+3}{Ni^2-8}$	[3]
8	$Nh = \frac{\sqrt{1+8Ni}-1}{2}$	[4]

Deep neural networks are now transforming many research fields, including computational medicine, speech recognition and computer vision [9]. A continuous function can be approximated by a neural network with one hidden layer if this layer has a sufficient number of neurons. However, multi-variable and complex functions are not easy to be approximated with a single hidden layer ANN. This difficulty in approximation can be solved by adding more hidden layers to the ANN (going deeper) [10-13]. Classical ANNs (with one hidden layer of neurons) have been used in the field of computational materials science and engineering in the nineties of the last century [14, 15]; however, deep ANNs have only recently been used in this field [16-18].

High-entropy alloys are defined as alloys with five or more principal elements having equal atom percentage. Each principal element should have a concentration between 5 and 35 atom%. Significant research interest did not develop until after independent papers by Jien-wei Yeh and Brain Cantor. Yeh coined the term 'high-entropy alloy', attributing the high configurational entropy as the mechanism stabilising the solid solution phase. Yeh explored the area of multicomponent alloys independently from 1995 [19], later theorising that a high mixing entropy would play an essential factor in reducing the number of phases in this high order of mixing and resulting in valuable properties [20]. To understand these properties, it is necessary to characterise the chemical ordering and identify order-disorder transitions through efficient simulation and modelling of thermodynamics. One of the main challenges of working on high entropy alloys is the selection of relevant and effective structures of these alloys. The structure represents the basic attributes for alloy or constituent elements of the alloy system. A multicomponent equiatomic alloy will form a single-phase BCC, FCC and multi-phase or more solid solution phases or intermetallic compounds (IM) or an amorphous phase. Designing and implementing an artificial intelligence-based tool to predict those phases will help to implement high entropy alloys design. The main objective of this paper is to utilise deep ANNs that predict body-centred-cubic and face-centred-cubic phases in high-entropy alloys.

1.1. Dataset

The dataset used in this paper to predict the phases of the high-entropy alloys was taken from the work of Miracle and Senkov [21]. The dataset has 18 attributes representing different elements (components of the alloys), and the last attribute is called a phase, which represents the phase of the alloys. A total of 17 elements were chosen as inputs for modelling: Al, Co, Cr, Cu, Fe, Ni, Ti, Mn, Pd, V, Mo, Nb, Si, Ta, Hf, Zr, and W. Figure 2 shows a snapshot of the dataset showing all the variables.

	Al	Co	Cr	Cu	Fe	Ni	Ti	Mn	Pd	V	Mo	Nb	Si	Ta	Hf	Zr	W	phase
0	0.0877	0.1754	0.1754	0.1754	0.1754	0.1754	0.035	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	0.0909	0.0000	0.1818	0.1818	0.1818	0.3636	0.000	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0000	0.2500	0.2500	0.0000	0.2500	0.2500	0.000	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.0000	0.2500	0.0000	0.0000	0.2500	0.2500	0.000	0.25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4	0.0000	0.2000	0.2000	0.0000	0.2000	0.2000	0.000	0.00	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

Figure 2: Snapshot of the dataset

The values of the 'phase' are 0, 1, and 2, which stand for the FCC, Multiphase and BCC phases, respectively. The dataset was split into training and testing datasets. The training dataset (80% of the overall dataset) was used to train the DNN prediction model, and the rest of the data (20% of the overall dataset) was used as testing data to evaluate the model performance. The dataset was uploaded and examined for any missing values, and it was randomly shuffled to make sure there were no patterns or pre-set data.

1.2. Prediction Model Architecture

The deep neural networks prediction model in this work has an input layer with 17 neurons to handle all the 17 variables values and an output layer with one neuron that represents the 'phase' variable. The main function of this model was to read the values of the 17 elements and make a decision (prediction) about the type of the phase: FCC = 0, Multiphase =1 and BCC = 2. In terms of hidden layers and after trying the different techniques in Table 2, the model was constructed by using two hidden layers with 30 neurons in the first hidden layer and 20 neurons in the second hidden layer. All neurons of the hidden layers have a sigmoid activation function. Figure 3 illustrates the architecture of the DNN prediction model.

In this work, Python programming language was used to develop the artificial intelligence-based prediction models. Python is a general-purpose and high-level programming language, which was created by Guido van Rossum and first released in 1991 [22]. The philosophy of Python emphasises code readability to help programmers write clear, logical code for small and large-scale projects [23]. TensorFlow, which is a machine learning system that operates on a large scale and in heterogeneous environments, is also used in this work to develop the prediction model. TensorFlow supports a variety of applications, but it particularly targets training and inference with deep neural networks [24].

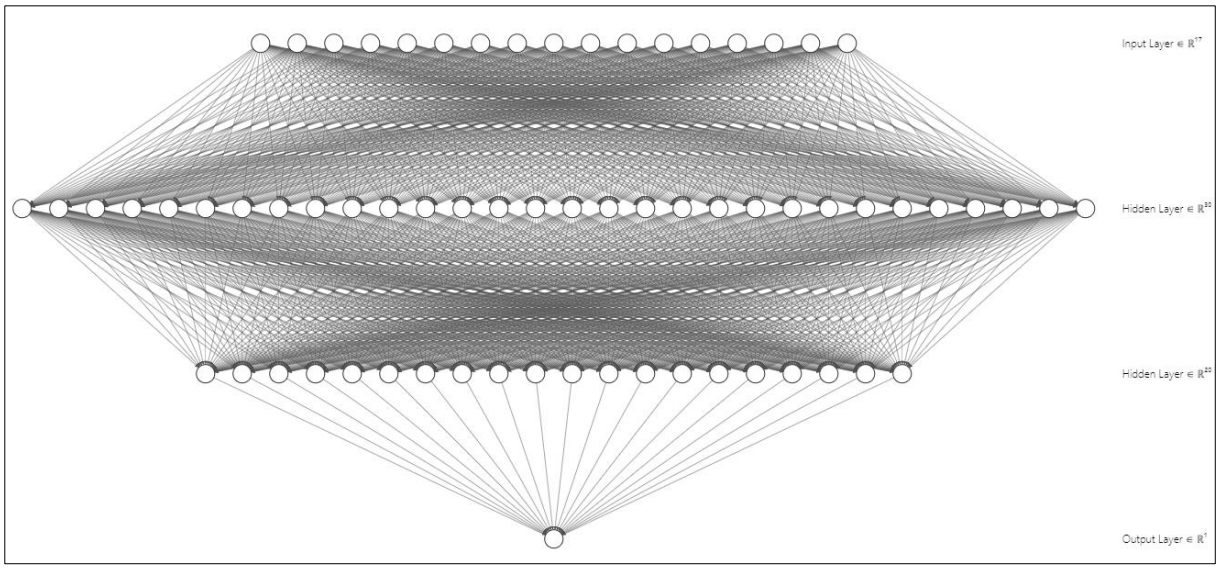

Figure 3: DNN prediction model architecture

2. Results and Discussion

The DNN prediction model was trained for 70k epochs, and the training target was to reduce the error in prediction (loss) to the minimum. Figure 4 shows the performance of the training process and how the loss function declines through the training. The loss function settled after 60k epochs, or 12.33% at the final step of the training process.

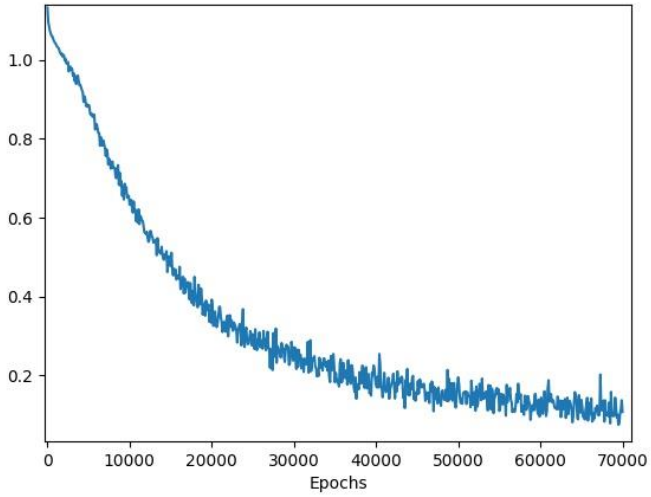

Figure 4: Loss function during the training process

The DNN prediction model was then evaluated by using different datasets (the testing dataset). In order to evaluate the model, the following parameters will be calculated:

- True Positives (TP): these are the correctly predicted positive phases, which means that the value of the actual phase (the one in the testing dataset) is the same as the value of the predicted phase.
- True Negatives (TN): these are the correctly predicted negative phases, which means that the value of the actual phase is not correct, and the value of the predicted phase is not correct as well.
- False Positives (FP): these are the values when the actual phase is not correct, and the predicted phase is correct.
- False Negative (FN): these the values when the actual phase is correct and the predicted phase in not.

Table 3 explains how to find the TP, TN, FP and FN values:

Table 3. Calculation of TP, TN, FP and FN values.

	Predicted Class		
Actual Class		**Class = YES**	**Class = NO**
	Class = YES	True Positive	False Negative
	Class = NO	False Positive	True Negative

Once the TP, TN, FP, and FN values are calculated, the following model measures performance, including accuracy, precision, recall, and F1 scores, which represents the most popular adopted metrics in classification tasks [25]. Accuracy is the most important performance measure of the DNN prediction model, and it is simply a ratio of correctly predicted observation to the total observations and is calculated as follow: $(TP+TN) / (TP+FP+FN+TN)$. Model precision represents the ratio of correctly predicted positive observations to the total predicted positive observations and is calculated as follows: $TP / (TP+FP)$. Recall, which is also called Model Sensitivity, represents the ratio of correctly predicted positive observations to all observations in actual class – yes – and is calculated by using the following formula: $TP / (TP+FN)$. Finally, the F1 score, which is a weighted average of Precision and Recall, is calculated as follows: $2 \times (Recall \times Precision) / (Recall + Precision)$.

The DNN prediction model achieved an overall accuracy of 90%. Table 4 shows the other measures of the DNN prediction model performance:

Table 4. DNN prediction Model Performance.

Phase	Precision	Recall	F1 Score
FCC	0.93	0.86	0.89
Multi-Phase	0.74	0.91	0.82
BCC	1.00	0.93	0.96

In addition to the above measures, a feature importance test was conducted by using random forest methodology. The aim of this test is to figure out the importance of each input variable. Figure 5 shows the importance level of each input (element) in the model. However, further work is ongoing to examine further methodologies for calculating variable importance.

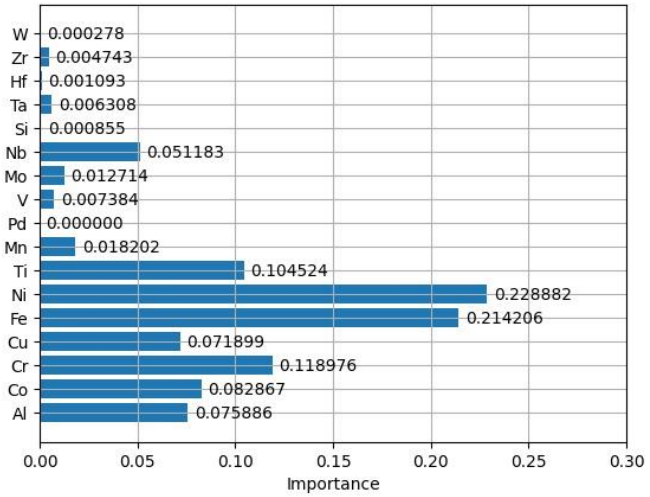

Figure 5: Variable Importance

3. Conclusion

Deep Neural Networks (DNNs) were developed in this work to predict body-centred-cubic, multi-phased and face-centred-cubic phases in high-entropy alloys. Using the built-in capabilities in TensorFlow and Keras, the authors trained DNNs with different layers and numbers of neurons, achieving a 90% prediction accuracy. The DDNs prediction model was examined in detail with different datasets to verify the model robustness. DDNs models used in this work were able to precisely predict the formation of solid solution and intermetallic phases in at least 90% of the cases on datasets. The model performs well on the prediction of the FCC & BCC dual-phase solid-solution as shown in Table 4, possibly due to phase ambiguities. Through this study, trained DNNs are shown to be a viable tool to predict the phases of high-entropy alloys. Further work is ongoing to develop the prediction model and better estimate the variable importance levels.

Authors Contribution

Conceptualisation, Alaa Abdulhasan Atiyah (A.A.A.); methodology, Zeyad Yousif Abdoon Al-Shibaany (Z.Y.A.A.); validation, A.A.A. and Z.Y.A.A.; formal analysis, Nadia Alkhafaji (N.A.), Yaser Al-Obaidi (Y.A.) and Z.Y.A.A.; investigation, Z.Y.A.A., N.A., and Y.A.; writing—original draft preparation, Z.Y.A. and N.A.; writing—review and editing, Z.Y.A. and A.A.A.. All authors have read and agreed to the published version of the manuscript.

References

[1] Z. Y. Al-Shibaany, J. Hedley, and R. Bicker, "Design of an adaptive neural kinematic controller for a National Instrument mobile robot system," presented at the 2012 IEEE International Conference on Control System, Computing and Engineering, 23-25 Nov. 2012, 2012.

[2] C. A. Doukim, J. A. Dargham, and A. Chekima, "Finding the number of hidden neurons for an MLP neural network using coarse to fine search technique," in *10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010)*, 2010, pp. 606-609: IEEE.

[3] K. G. Sheela and S. N. Deepa, "Review on methods to fix number of hidden neurons in neural networks," *Mathematical Problems in Engineering,* vol. 2013, 2013.

[4] J.-Y. Li, T. W. Chow, and Y.-L. Yu, "The estimation theory and optimisation algorithm for the number of hidden units in the higher-order feedforward neural network," in *Proceedings of ICNN'95-International Conference on Neural Networks*, 1995, vol. 3, pp. 1229-1233: IEEE.
[5] S. i. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: four layers versus three," *IEEE Transactions on Neural Networks,* vol. 8, no. 2, pp. 251-255, 1997.
[6] S. Xu and L. Chen, "A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining," in *5th International Conference on Information Technology and Applications*, 2008, 2008.
[7] K. Shibata and Y. Ikeda, "Effect of number of hidden neurons on learning in large-scale layered neural networks," in *2009 ICCAS-SICE*, 2009, pp. 5008-5013: IEEE.
[8] D. Hunter, H. Yu, M. S. Pukish III, J. Kolbusz, and B. M. Wilamowski, "Selection of proper neural network sizes and architectures—A comparative study," *IEEE Transactions on Industrial Informatics,* vol. 8, no. 2, pp. 228-240, 2012.
[9] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016.
[10] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in neural information processing systems*, 2017, pp. 6231-6239.
[11] M. Telgarsky, "Benefits of depth in neural networks," *arXiv preprint arXiv:1602.04485,* 2016.
[12] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: A tensor analysis," in *Conference on learning theory*, 2016, pp. 698-728.
[13] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," in *Conference on learning theory*, 2016, pp. 907-940.
[14] A. Skinner and J. Broughton, "Neural networks in computational materials science: Training algorithms," *Modelling and Simulation in Materials Science and Engineering,* vol. 3, no. 3, p. 371, 1995.
[15] B. HKDH, "Neural networks in materials science," *ISIJ international,* vol. 39, no. 10, pp. 966-979, 1999.
[16] J. Schmidt, M. R. Marques, S. Botti, and M. A. Marques, "Recent advances and applications of machine learning in solid-state materials science," *npj Computational Materials,* vol. 5, no. 1, pp. 1-36, 2019.
[17] A. Agrawal and A. Choudhary, "Deep materials informatics: Applications of deep learning in materials science," *MRS Communications,* vol. 9, no. 3, pp. 779-792, 2019.
[18] M. Picklum and M. Beetz, "MatCALO: Knowledge-enabled machine learning in materials science," *Computational Materials Science,* vol. 163, pp. 50-62, 2019.
[19] C.-Y. Hsu, J.-W. Yeh, S.-K. Chen, and T.-T. Shun, "Wear resistance and high-temperature compression strength of Fcc CuCoNiCrAl 0.5 Fe alloy with boron addition," *Metallurgical and Materials Transactions A,* vol. 35, no. 5, pp. 1465-1469, 2004.
[20] J. W. Yeh *et al.*, "Nanostructured high-entropy alloys with multiple principal elements: novel alloy design concepts and outcomes," *Advanced Engineering Materials,* vol. 6, no. 5, pp. 299-303, 2004.
[21] D. B. Miracle and O. N. Senkov, "A critical review of high entropy alloys and related concepts," *Acta Materialia,* vol. 122, pp. 448-511, 2017/01/01/ 2017.
[22] J. Guttag, *Introduction to computation and programming using Python: With application to understanding data*. MIT Press, 2016.
[23] D. Kuhlman, *A python book: Beginning Python, advanced Python, and python exercises*. Dave Kuhlman Lutz, 2009.
[24] M. Abadi, "TensorFlow: learning functions at scale," presented at the Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, Nara, Japan, 2016. Available: https://doi.org/10.1145/2951913.2976746

[25] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics,* vol. 21, no. 1, p. 6, 2020/01/02 2020.