A Tutorial, Literature Review, and Comparative Analysis of Restricted Boltzmann Machines and Deep Belief Networks

Ashraf Haress
Artificial Intelligence Major,
Faculty of Informatics and Computer Sciences,
The British University in Egypt,
Cairo, Egypt
ashraf196280@bue.edu.eg

***Abstract*: This paper serves as a brief tutorial for the Deep Belief Network (DBN) model by stating its history, previous work that leads up to its inception, its training mechanism, and its applications. The paper then compares different DBN models on a specific application, to identify the most suitable model for said application.**

Keywords: Deep Belief Networks, Restricted Boltzmann Machines, Markov Random Fields, Maximal Cliques, Maximum Likelihood Estimation, KL-Divergence, Gibbs Sampling, Contrastive Divergence, MNIST Dataset, Comparative Analysis

I. INTRODUCTION

A. BM vs RBM

In 1983, Hinton & Sejnowski proposed Boltzmann (BM) & Restricted Boltzmann Machine (RBM) [1]. A Boltzmann Machine is a network model where the nodes are connected in a full mesh topology [2]. It is used to learn important aspects of an unknown probability distribution based on samples from this distribution [3]. Since there are $\frac{n*(n-1)}{2}$ edges (where n is the number of nodes) which is inefficient, the Restricted Boltzmann Machine (RBM) was proposed which excludes edges on the same layer in the network. Fig. 1 demonstrates the difference between BM and RBM model.

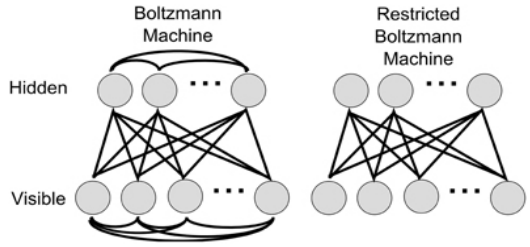

Fig. 1. BM vs RBM regarding connectivity [4]

BMs and RBMs have the same 2 layers: a visible layer, where each neuron corresponds to a component of an observation (i.e., input), and a hidden layer, where each neuron represents the dependency between the components of an observation. Hidden layers are also considered to be non-linear feature detectors [3].

B. MRFs: Structure Overview

The structure of BM and RBM is an undirected graphical model (UGM), which is also referred as Markov Network, or Markov Random Field (MRF). An MRF is a type of probabilistic graphical model (PGM) where the random variables represent nodes, and the undirected edges represent the variables which interact (correlate) in some way [5]. Therefore, one can calculate the joint probability of events of a random variable X using equation (1) [3].

$$P(X) = \frac{1}{Z} \prod_{c \in \hat{C}(G)} \phi_c(x_c) \quad (1)$$

Where P is the probability distribution of X And by assuming we have an undirected graph $G = (X, E)$ where X is the set of nodes and E is the set of undirected edges, then a clique c can be defined as a subset of X in which all the nodes of that subset are pairwise connected [6]. For example, in Fig. 2, both $\{x_1, x_2\}$ and $\{x_1, x_2, x_3\}$ are cliques of the set of cliques C, but only the latter subset is a maximal clique (which means no node can be added to the latter subset such that it will still remain a clique). Therefore, the latter subset is included in the set of maximal cliques $\hat{C}$ related to graph G (expressed as $\hat{C}(G)$).

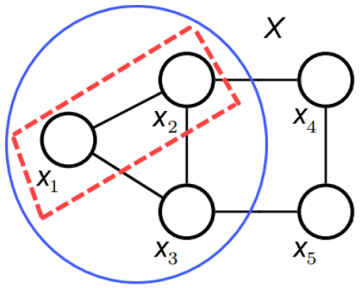

Fig. 2. Clique vs maximal clique, and calculation of $P(X)$. Adapted from [6]

Continuing with the explanation of equation (1), ϕ is the potential function applied on each maximal clique that includes events x_c such that $x_c \subset X$. This potential function does not output the conditional probabilities of x_c events included in a maximal clique c, but rather it outputs a non-negative value which represents a "compatibility score" of how much the x_c events of maximal clique c correlate. Moreover, these non-negative values do not have a specific range. Therefore, they must be normalised to be in the range [0, 1], and as such (1) can represent a valid probability distribution model. For that normalization to occur, the product of the potential function is divided by the partition function Z — also referred to as the normalising constant — which is the sum of the product of potential values of the maximal cliques of all the possible combinations of X given by finite space Ω [6]. Succinctly, this is represented by equation (2)[1].

[1] For example, if X is an n random binary variable, then there are 2^n possible states of X that are represented by Ω

$$Z = \sum_{x \in \Omega} \left(\prod_{c \in \hat{C}(G)} \phi_c(x_c) \right) \quad (2)$$

Now, recalling the ln product property in (3),

$$\ln\left(\prod_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} ln(x_i) \quad (3)$$

and the exponent raised to ln rule in (4),

$$e^{\ln(x)} = x \quad (4)$$

these properties could be utilized to change (1) to (5)

$$\begin{aligned} P(X) &= \frac{1}{Z} e^{\ln\left(\prod_{c \in \hat{C}(G)} \phi_c(x_c)\right)} \\ &= \frac{1}{Z} e^{\sum_{c \in \hat{C}(G)} \ln(\phi_c(x_c))} \\ &= \frac{1}{Z} e^{-E(x)} \end{aligned} \quad (5)$$

such that the energy function $E(x)$ is represented in (6).

$$E(x) = -\sum_{c \in \hat{C}(G)} \ln(\phi_c(x_c)) \quad (6)$$

Thus, equation (5) represents the probability distribution of MRF, which is also called Gibbs distribution [6]. In addition, we can use (3) and (4) to change (2) into (7).

$$Z = \sum_{x} e^{-E(x)} \quad (7)$$

C. MRFs: Unsupervised Learning and KL-Divergence

Now, to understand the RBM training process, Markov Random Field (MRF) unsupervised learning must be understood. Unsupervised learning means learning an unknown distribution q based on sample data. Let θ be the parameters of the energy function that we want to learn to model a distribution p on the examples of the sample data that resembles the data's distribution and let the training data with high probability $S = \{x_1, x_2, \ldots, x_m\}$ and assumed to be independent and identically distributed (i.i.d.). To estimate the parameters θ, Maximum Liklihood Estimation (MLE) is used. In the context of MRFs, MLE means finding θ that will maximize the probability of S under p; the MRF distribution [6]. If we denote the likelihood function, which is the joint probability density function of S, as L, then we have equation (8),

$$\begin{aligned} L(\theta; S) &= p(S; \theta) \\ &= p(x_1, x_2, \ldots, x_m; \theta) \\ &= \prod_{i=1}^{m} p(x_i; \theta) \end{aligned} \quad (8)$$

where the semicolon ";" indicates that the symbols that appear after it are parameters of the probability distribution[1] p.

However, following the Gibbs distribution for MRF, it is generally not possible to find θ analytically using gradients [6] for (7) and its log-likelihood[2] equivalent which is mathematically expressed using equation (9).

$$\ln\big(L(\theta; S)\big) = \ln\left(\prod_{i=1}^{m} p(x_i; \theta)\right) = \sum_{i=1}^{m} \ln\big(p(x_i; \theta)\big) \quad (9)$$

Therefore, gradient ascent is applied using Contrastive Divergence (CD), but to understand CD, we must first discuss Kullback–Leibler (KL) divergence. KL divergence — also referred to as relative entropy [7] — is a Variational Inference (VI) technique used to approximate a modeled probability distribution p to the real (target) probability distribution q [8]. In other words, if we calculate the difference between the unknown distribution of the data q and the distribution of the MRF p given a finite space Ω, then one can say that minimizing that difference is equivalent to maximizing the likelihood of S which is given by $L(\theta; S)$ in (7). The KL divergence is given by (10),

$$\begin{aligned} D_{KL}(q||p) &= \sum_{x \in \Omega} q(x) \ln\left(\frac{q(x)}{p(x)}\right) \\ &= \sum_{x \in \Omega} q(x) \ln\big(q(x)\big) - \sum_{x \in \Omega} q(x) \ln\big(p(x)\big) \\ &= \sum_{x \in \Omega} q(x) \cdot \big(\ln\big(q(x)\big) - \ln\big(p(x)\big)\big) \\ &= E\big[\ln\big(q(x)\big) - \ln\big(p(x)\big)\big] \end{aligned} \quad (10)$$

Where the last term of (10) is the expectation of the log difference between the probability of data in the original distribution q and the approximated distribution p. Therefore, approximating the expectation of p to resemble the expectation of q results in a higher log-likelihood of p. That is why maximizing the log-likelihood corresponds to minimising the KL divergence. However, one has to obtain the function of the unknown (i.e., posterior) distribution q in order to calculate the KL value, and it was shown by Andrieu et al. [9] that obtaining this function is intractable for higher dimensions of data, as they are harder to be initially approximated with a common distribution (e.g., Gaussian). Therefore, a specific method of Markov Chain Monte Carlo (MCMC) called Gibbs Sampling is used to approximate the distribution of q.

D. MCMC

A Monte Carlo method is an algorithm that estimates a probability of an event of a random variable by repeatedly observing random samples (i.e., trials). A Markov Chain is a mathematical model which represents the relationship between a sequence of events (i.e., states) which are represented by nodes while the edges between these nodes are the transition probabilities of these states. A special property of Markov

[1] The first term of (8) is asking for the likelihood of the parameters θ while the data *S* are present, but the second term is asking for the probability of observing *S* while the parameters θ are present.

[2] The usage of ln instead of log in the log likelihood function doesn't affect the results, but only change its absolute value (i.e., scale).

Chains is that the next state is dependent on the previous one only. This memoryless property is mathematically written in (11),

$$P(x^t|x^{t-1}, \dots, x^0) = P(x^t|x^{t-1}) \quad (11)$$

Where t is the time step parameter, x^t is an event of random variable X^t that we want its probability given that the previous event x^{t-1} in the Markov Chain. Fig. 3 illustrates this property.

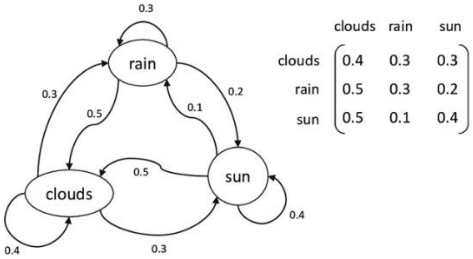

Fig. 3. Markov Chains as a state diagram and transition matrix [10]

Returning to the problem of approximating the distribution of q, we want to choose the samples which will optimally approximate q. The procedure done to achieve this approximation is to use a transition function which uses the transition matrix to output the next sample x^t given the previous sample x^{t-1}, such that this procedure is done for k iterations until the probability distribution p for the Markov Chain becomes a stationary distribution. In other words, $p(x^k) = q(x^k)$. When this is the case, we will be able to generate and choose — using the transition function — the samples $S = \{x^k, \dots, x^{k+m}\}$ which are used to optimally approximate q, noting that the discarded samples $\{x^0, \dots, x^{k-1}\}$ which allowed us to get to x^k are called "burn-in" samples [5].

Moreover, the Metropolis-Hastings (MH) algorithm — which is one of the MCMC algorithms — uses a transition function and an acceptance function to determine if we will accept the transition from event x^{t-1} to event x^t. However, this paper will discuss Gibbs sampling, which is a special case of MH algorithm that always accepts the transition (i.e., the acceptance function is equal to one) [11].

E. Gibbs Sampling

Gibbs sampling is an algorithm used to generate new values (i.e., samples) from current observations [5]. A high-level overview of the algorithm is as follows: Suppose you have two random variables X and Y:

1. Start with initial values x^0 and y^0
2. Sample a new x^1 value from $p(x^1 \mid y^0)$
3. Sample a new y^1 value from $p(y^1 \mid x^1)$

Thus, Gibbs sampling algorithm was done 1 time to generate new sample (x^1, y^1) from (x^0, y^0). Fig. 4 is a schematic representation of the aforementioned example for 5 iterations of the Gibbs sampling algorithm.

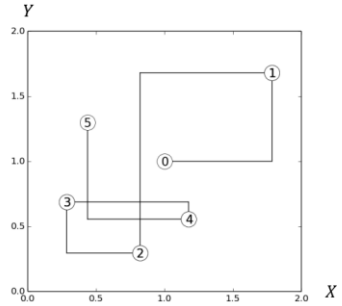

Fig. 4. Five iterations of the Gibbs sampling algorithm.

This algorithm will be re-stated when talking about RBM training.

F. Contrastive Divergence

Contrastive Divergence (CD) learning a technique used to approximate the gradient of the difference of two Kullback–Leibler (KL) divergences [12] represented by (12),

$$CD = D_{KL}(q_0 || p) - D_{KL}(q_k || p) \quad (12)$$

Where q_0 is the original data's probability distribution, and q_k is the data's distribution after Gibbs sampling has been performed k times. This variation of CD is called k-Contrastive Divergence (k-CD) [13].

G. RBM Training

This section relates the concepts mentioned in the introduction section to explain the training procedure of RBMs. Architecture-wise, an RBM consists of 2 layers (i.e., vectors): a visible vector V and a hidden vector H, such that they form a bipartite graph with weight matrix W_{nm} — where the i[th] column represents edges from all nodes of V to h_i and the i[th] row represents edges from v_i to all nodes of H — as shown in Fig. 5.

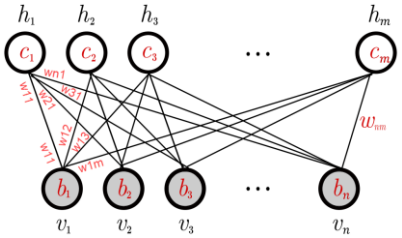

Fig. 5. Architecture of an RBM model. Adapted from [6]

Where b and c are the biases associated with each node of V and H respectively. Moreover, considering a binary RBM, we have the set of visible vectors $\mathcal{V} = \{0,1\}^n$, $|\mathcal{V}| = N$ and the set of hidden vectors $\mathcal{H} = \{0,1\}^m$, $|\mathcal{H}| = M$, where each configuration $(V, H) \in \mathcal{V} \times \mathcal{H}$ has an associated energy function defined at (16) [14]. Given the differences between the MRF and RBM architectures, (5) is now re-stated as (13),

$$P(V, H) = \frac{1}{Z} e^{-E(V,H)} \quad (13)$$

where $P(V, H)$ is the joint probability that the visible and hidden vector are true (i.e., the configuration of their events). Consequently, (7) now represents the sum of energies of all the possible configurations of V and H vectors and represented by (14).

$$Z = \sum_{i=1}^{N} \sum_{j=1}^{M} e^{-E(V_i, H_j)} \quad (14)$$

Moreover, the energy function is now modeled after the Ising model [5], instead of the free energy function presented at (6), to become (15),

$$E(V,H) = -\left(\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij}\, v_i\, h_j + \sum_{i=1}^{n} b_i\, v_i + \sum_{j=1}^{m} c_j\, h_j \right) \quad (15)$$

which represents the total energy of the system given the configuration of vectors V and H. Now, we have the following steps, which are iterated, to train an RBM model (assuming k-CD):

1. Perform Gibbs sampling for k iterations to (re)construct $\mathrm{V} = \{V^0, \dots, V^k\}$, $\mathrm{H} = \{H^1, \dots, H^{k-1}\}$, where V^0 is the original data having the probability distribution q, and V^k represents the final reconstructed visible vector (i.e., augmented data) which has the probability distribution p.

2. Measure the likelihood change from V^0 to V^k by computing the gradient.

3. Update the weights by using gradient ascent which is approximated by contrastive divergence.

1) Performing Gibbs Sampling to obtain V^k

A step of Gibbs sampling consists of two sub-steps: Forward pass and backward pass; each of them will be elaborated below.

a) Forward Pass

The first step of Gibbs sampling is to multiply the input vector V^t by the weights W and add them to the bias to obtain H_t as shown in Fig. 6:

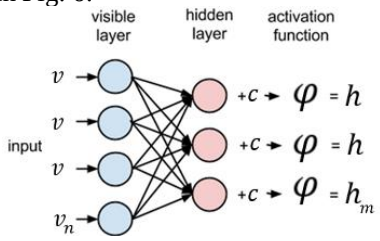

Fig. 6. Forward pass step of RBM training. Adapted from [15]

where the j[th] hidden node can be mathematically formulated as (16),

$$p\left(h_j = 1 \middle| V^t\right) = \varphi\left(c_j + \sum_{i=1}^{n} v_i\, w_{ij} \right) \quad (16)$$

noting that $\varphi(X)$ is the sigmoid function (17) [16].

$$\varphi(X) = \frac{1}{1 + e^{-X}} \quad (17)$$

Moreover, (16) can be vectorized into[1]:

$$H^t = \varphi((V^t)^T W + c) \quad (18)$$

Such that T is taking the transpose, $t \in \{1, \dots, k\}$, and as stated before, k is the number of Gibbs sampling iterations.

b) Backward Pass

The second step of Gibbs sampling is to reconstruct augmented data V^{t+1} from H^{k-1} which represents the encoded features of the previous data V^t. In other words, the backward pass is the same as the forward pass, but in the opposite direction, as illustrated in Fig. 7,

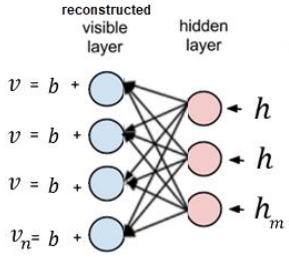

Fig. 7. Backward pass step of RBM training. Adapted from [15]

where the i[th] visible node can be mathematically formulated as (19).

$$p(v_i = 1 | H^t) = \varphi\left(b_i + \sum_{j=1}^{m} h_j\, w_{ij} \right) \quad (19)$$

Moreover, (19) can be vectorized into[2]:

$$V^{t+1} = \varphi(W H^t + c) \quad (20)$$

We can now repeat the forward and backward pass k times to obtain V^k.

2) Measure the Likelihood Change from V^0 to V^k

After obtaining V^k, we calculate the derivative of the log-likelihood function w.r.t the weights, visible, and hidden biases, which are respectively shown in (21), (22), and (23)[3],

$$\frac{\partial \ln\left(L(\theta; V^k)\right)}{\partial W} = \Delta W = p\left(h_j = 1 \middle| V^0\right) \cdot v_i^0 - p\left(h_j = 1 \middle| V^k\right) \cdot v_i^k \quad (21)$$

$$\frac{\partial \ln\left(L(\theta; V^k)\right)}{\partial b} = \Delta b = v_i^0 - v_i^k \quad (22)$$

$$\frac{\partial \ln\left(L(\theta; V^k)\right)}{\partial c} = \Delta c = p\left(h_j = 1 \middle| V^0\right) - p\left(h_j = 1 \middle| V^k\right) \quad (23)$$

[1] (18) returns a row vector, so one might take the transpose to obtain H_i as a column vector.

[2] (20) returns a column vector, so there is no need to transpose.

[3] The different definition of the L function and the derivations of how (21, 22, 23) were obtained are from [6] and are not explicitly mentioned here.

and can be used to update the weights, visible, and hidden biases respectively of the RBM model at the contrastive divergence step [6], noting that the subscripts i and j indicate that we must perform these steps at each visible and hidden node. To avoid this, the vectorized equivalent of these equations can be used[1], which are equations (24), (25), and (26).

$$\Delta W = (V^0)^T H^0 - (V^K)^T H^K \tag{24}$$

$$\Delta b = V^k - V^0 \tag{25}$$

$$\Delta c = H^0 - H^k \tag{26}$$

3) Using k-Contrastive Divergence to Increase Likelihood

Recalling section I.C., we want to maximize the probability of the visible units sampled from the probability distribution p which is achieved by maximizing the log-likelihood (9). Therefore, we can do this by performing stochastic gradient ascent, which is achieved by increasing the current model's parameters by the difference between the original distribution q and the modeled distribution p obtained after k steps of Gibbs sampling. This whole process is called K-contrastive divergence. The parameters update step is mathematically formulated as equations (27), (28), and (29).

$$W_{new} = W_{old} + \Delta W \tag{27}$$

$$b_{new} = b_{old} + \Delta b \tag{28}$$

$$c_{new} = c_{old} + \Delta c \tag{29}$$

4) RBM Training: Summary

The process of RBM training can be summarized by Fig. 8 which illustrates Gibbs sampling:

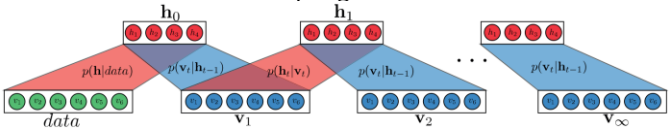

Fig. 8. Gibbs Sampling applied until model distribution equilibrium V^∞ which is intractable. Therefore K-CD is used to obtain V^k instead. Adapted from [13]

And by algorithms 1 and 2, which illustrate Gibbs sampling and K-CD algorithm respectively[2] [6]:

Algorithm 2: k-step contrastive divergence

Input: RBM $(v_1, \dots, v_m, h_1, \dots, h_n)$, training batch S

Output: gradient approximation w_{ij}, b_i and c_j for $i = 1, \dots, n$, $j = 1, \dots, m$

init $w_{ij} = b_i = c_j = 0$ for $i = 1, \dots, n$, $j = 1, \dots, m$

forall the $V \in S$ **do**

  $V^{(0)} \leftarrow V$

  for $t = 0, \dots, k-1$ **do**

    for $j = 1, \dots, m$ **do** sample $h_j^{(t)} \sim p(h_j \mid V^{(t)})$

    ;

    for $i = 1, \dots, n$ **do** sample $v_i^{(t+1)} \sim p(v_i \mid H^{(t)})$

    ;

  for $i = 1, \dots, n$, $j = 1, \dots, m$ **do**

    $w_{ij} \leftarrow w_{ij} + \{p(h_j = 1 \mid V^{(0)}) \cdot v_i^{(0)} - p(h_j = 1 \mid V^{(k)}) \cdot v_i^{(k)}\} \Delta w_{ij}$

  for $i = 1, \dots, n$ **do**

    $b_i \leftarrow b_i + \{v_i^{(0)} - v_i^{(k)}\} \Delta b_i$

  for $j = 1, \dots, m$ **do**

    $c_j \leftarrow c_j + \{p(h_j = 1 \mid V^{(0)}) - p(h_j = 1 \mid V^{(k)})\} \Delta c_j$

Algorithm 1: *Gibbs Sampling*

Algorithm 1 & 2; Gibbs Sampling & K-CD. Adapted from [6]

Noting that we can assign a learning rate hyper-parameter α for each of ΔW, Δb, and Δc.

H. Overview of DBNs

In 2006, Hinton tried to revive neural networks after it came to a halt due to the "vanishing gradient" problem, where he proposed a greedy learning method for RBM: Train every layer of the network using RBM training [5]. In other words, he treated the neural network as stacked layers of RBM, where this interpretation of the network is called the Deep Belief Network (DBN). DBN allowed good initialization of weights (using RBM training) which prevented the vanishing gradient problem.

1) DBM Architecture

In a DBM architecture, the neural network consists of ℓ layers where each layer L has L_e neurons such that $L_1 = initial\ data = X = \{x_1, \dots, x_n\}$, while , $\mathcal{W} = \{W_1, \dots, W_\ell\}$ and $\mathcal{B} = \{B_1, \dots, B_\ell\}$ represent the weight matrices and biases of each layer as depicted in Fig. 9.

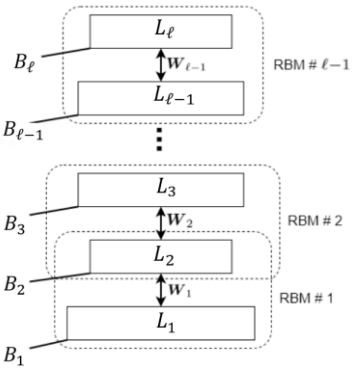

Fig. 9. DBN where each pair of layers is an RBM. Adapted from [5]

2) DBN Training

We can now consider every two successive layers as one RBM, where we obtain and train the weights and biases of each RBM using stacked RBM training, which is demonstrated in the following algorithm [5]:

1. If we're at layer 1 (i.e., $i = 1$):
   a. Initialize L_1 with X.
2. Otherwise, generate hidden values for L_i using Algorithm 1 on values of L_{i-1}. Now we consider the hidden layer L_i for the current RBM as the visible layer for the next RBM.
3. Train W_i and B_i of RBM_i using Algorithm 2.
4. Repeat steps 2 and 3 until $i = \ell - 1$.

Algorithm 3: DBN Training. Interpreted from [5]

This layer-wise training constitutes as a greedy approach [17], such that the well-initialized parameters $\mathcal{W}$ and $\mathcal{B}$ obtained

[1] Equations (24, 25, 26) are mentioned here as an alternative formulation, yet (21, 22, 23) will be used in this paper.

[2] Note that Algorithm 1 is a part of Algorithm 2, so mentioning the latter implies that the former is executed first.

from this training could be fine-tuned using back-propagation without worrying about the vanishing gradient problem.

3) Using DBN as a classifier

Classification Deep Belief Networks (CDBNs) can be used for classification problems such as image classifications, where we start with the pre-training step (i.e., DBN training), then we use labels to fine-tune the DBN's parameters by using gradient descent and connecting the last layer with an activation function like sigmoid function for binary classification problems or soft-max regression for multi-classification problems which is denoted by (30),

$$\Phi(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{30}$$

such that $Z = L_\ell = \{z_1, \dots, z_k\}$ represents the neurons at the final layer L_ℓ of the network, i is the class we want to get its probability, and $C = \{c_1, \dots, c_k\}$ represents the k classes that we want to classify [18]. This fine-tuning step is illustrated in Fig. 10.

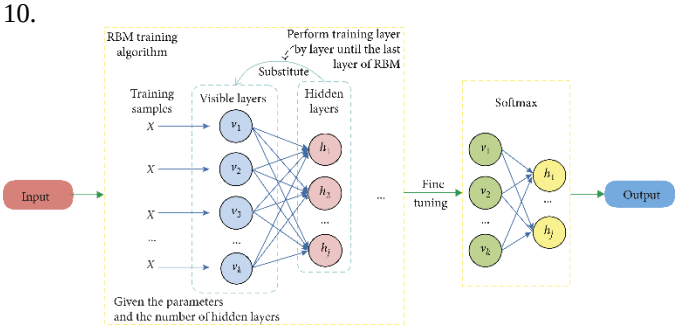

Fig. 10. CDBN pre-training & fine-tuning steps [18]

II. Literature Review

The DBN model is used in various applications including solar radiation estimation [19], time-series problems such as draught forecasting [20], speech-recognition problems such as interpreting human commands to execute program tasks [21], and image classification problems such as land cover classification [22]. This paper will focus on the image classification problem. Specifically, four CDBN[1] models trained on handwritten-digits datasets will be discussed and compared:

1. A model used by Abu Ghosh & Maghari [24], which will be denoted by DBN_1
2. Two models used by Dewi et al. [25], which will be denoted by DBN_{2B1} and DBN_{2B2}, where the latter letter "B" stands for "Best".
3. A model used by Gm et al. [26], which will be denoted by DBN_3

Regarding the discussion about the models, the following details will be covered as they are relevant when comparing the models:

1. Datasets used, as the size and quality of the input (i.e., image) is an important factor of the evaluation score.
2. Toolbox used, which is mentioned to determine default hyper-parameters which are not explicitly stated by the authors
3. Pre-processing, segmentation, and feature extraction, which are mentioned to recognize what is initially fed in the neural network (i.e., what the input data is like)
4. Architecture of the DBN, which is mentioned as the number of neurons and learning rate have an influence on the evaluation score.
5. Evaluation metrics, which are mentioned to determine the common metrics that will be later used in the comparative analysis section.

The following sub-sections discuss each of the aforementioned points, ending with a comparative analysis of the DBN models.

A. Datasets Used

For DBN_1 and DBN_2, both models were trained using different versions of the Modified National Institute of Standards and Technology (MNIST) dataset which was published by Yann and consists of 60,000 and 10,000 labeled digit images for the train and test sets respectively [27], noting that the images were normalized and centered with a fixed-size. Fig. 11 shows a sample of the images.

Fig. 11. A sample of the MNIST dataset [27]

For DBN_3 however, it was trained using the Optical Recognition of Handwritten Digits (ORHD) dataset which was published by Alpaydin & Kaynak and consists of 5620 labeled digit images [28], [29]; each one having a size of 8×8. Fig. 12 shows a sample of the images.

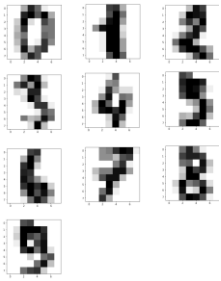

Fig. 12. A sample of the ORHD dataset [30]

B. Toolboxes Used

For DBN_1, MATLAB's "Neural Network Toolbox" was used, which has tools for feature learning and image recognition.

[1] As previously stated, CDBN refers to a DBN model used for classification tasks, so it does not refer to a convolutional DBN [23].

For DBN_2, Dewi et al. did not mention the programming language used to implement their model[1].

For DBN_3, Gm et al. adapted code from python's "scikit-learn" library and from GitHub user "alberbups" into their own GitHub repository [32].

C. Pre-processing, Segmentation, & Feature Extraction

Even though the MNIST dataset is already clean and without noise, Ghosh & Maghari applied erosion with 3×3 structuring elements to eliminate any possible one-bit errors and give a smoother edge, then they dilated the digits with 2×2 elements. Now, since they used a version of the MNIST dataset where each image is a sequence of digits, they had to perform a segmentation step which consists of decomposing the image into sub-images of individual digits, re-sizing them to 100×70 pixels and assigning appropriate labels to them. Moreover, they decided to extract from each sub-image d statistical features using MATLAB's feature extraction module which will be represented by $F = \{f_1, \dots, f_d\}$, where $d = 54$

Meanwhile, Dewi et al. did not perform any pre-processing, segmentation, or feature extraction on the version of the MNIST dataset that fixes the size of each digit image to 28×28.

Gm et al., however, have augmented the ORHD dataset to be $5620 \times 5 = 28100$ images by shifting the images by 1px to the right, left, up, and down followed by splitting the data into 22480 and 5620 images (i.e., 8:2 proportion) in the train and data sets respectively. Furthermore, they didn't perform any segmentation or feature extraction before starting the DBN training.

D. DBN Architectures

For DBN_1, the model has the following network structure:

1. An input layer consisting of 54 nodes, each representing a feature f in F.
2. Two hidden layers, each one having 100 nodes, and obtained by trial and error.
3. An output layer consisting of 10 nodes[2], where each node represents one of the digits from 0 to 9.
4. The sigmoid (i.e., logistic regression) activation function was used.

Furthermore, the authors of DBN_1 did not mention any of the other hyper-parameters that were chosen to train the model with.

For DBN_2, the authors have tested models varying in the number of hidden layers and neurons in each of these layers. Therefore, this paper will only choose and discuss DBN_{2B1} and DBN_{2B2} which have the highest accuracies[3], and each has the following network structure:

1. An input layer, where the number of nodes is $28 \times 28 = 784$ nodes.
2. DBN_{2B1} has two hidden layers, each one having 350 nodes, while DBN_{2B2} has three hidden layers, each one having 400 nodes.
3. An output layer consisting of 10 nodes.
4. Sigmoid activation function was used.

Furthermore, the learning rate was also not stated. Howerver, the number of iterations (i.e., epochs) and batch size hyper-parameters were mentioned as 1000 and 10 respectively.

For DBN_3, the model has the following network structure:

1. An input layer consisting of $8 \times 8 = 64$ nodes.
2. Two hidden layers, where the first and second hidden layers have 256 and 512 nodes respectively.
3. An output layer consisting of 10 nodes.
4. Sigmoid activation function was used.

Moreover, the learning rate, number of iterations, and batch size hyper-parameters were 0.06, 20, and 10 respectively.

E. Evaluation Metrics

For DBN_1, the authors used accuracy, performance, and execution time factors on each of the 10 digits as evaluation metrics. However, due to lack of information on the computational environments and the exclusion of performance metric of other authors' papers, only the accuracy metric will be considered in this paper and is shown in Fig. 13.

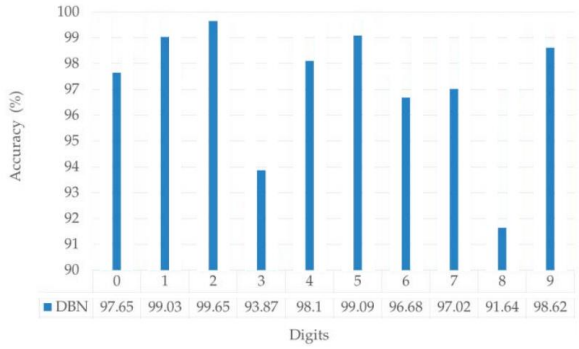

Fig. 13. Accuracy of predicting each digit by DBN_1. Adapted from [24]

[1] However, the naming convention of the functions and variables mentioned in their paper, follow the "RBM" package made by "TimoMatzen" which was written in the programming language R and includes the RBM, stacked RBM, and DBN models [31].

[2] Note that there was an erratum in [24] where the authors have written the number of output neurons as 38 even though they later mentioned that the number of output neurons is the same as the number of digits, which is 10.

[3] DBN_{2B1} has the highest accuracy out of all 2 hidden-layered models, while DBN_{2B2} has the highest accuracy out of all 3 hidden-layered models.

For DBN_{2B1} and DBN_{2B2}, the authors also used accuracy as an evaluation metric where DBN_{2B1} and DBN_{2B2} obtained an accuracy of 90.15% and 90% respectively.

For DBN_3, the authors used precision, recall, and f1-score as evaluation metrics. However, since f1-score is the harmonic mean of the former two metrics, therefore this paper will only consider the f1 evaluation metric, which is shown in Fig. 14.

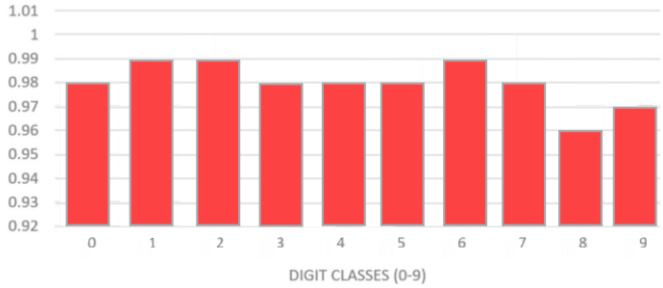

Fig. 14. F1-score of predicting each digit by DBN_3. Adapted from [26]

F. *Comparitive Analysis*

Table 1 gives a side-by-side view of the evaluation criteria (EC) discussed in the previous sub-sections and Fig. 15 is a comparison of their evaluation scores:

EC \ DBNs	DBN_1	DBN_{2B1}	DBN_{2B2}	DBN_3
Dataset	MNISTv$_1$	MNISTv2	MNISTv2	ORHD
Toolbox	MATLAB	?	?	Python
Pre-Processing?	✓	✗	✗	✗
Pre-Feature Extraction?	✓	✗	✗	✗
#Nodes in Input Layer	54	784	784	64
#Nodes in Hidden Layers	100 — 100	350 — 350	400 — 400 — 400	256 — 512
#Nodes in Output Layer	10	10	10	10
Activation Function	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Learning Rate	?	?	?	0.06
#Epochs	?	1000	1000	20
#Batches	?	10	10	10
Evaluation Metric	Accuracy	Accuracy	Accuracy	F1-Score

Table 1. Evaluation criteria of the DBN models

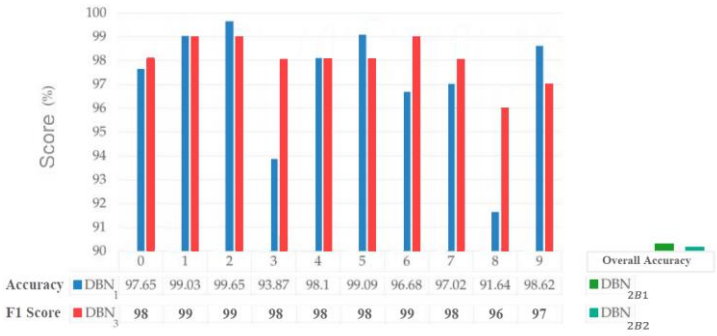

Fig. 15. Evaluation scores of the DBN models

The term "overall accuracy" refers to the accuracy of the testing set over all 10 digits, and not on each specific digit. Accuracy on each specific digit is defined by (31),

$$Accuracy_1 = \frac{Correct\ Predictions}{Total\ Cases} = \frac{TP + TN}{TP + TN + FP + FN} \quad (31)$$

noting that the symbols in (31) are described in Fig. 16.

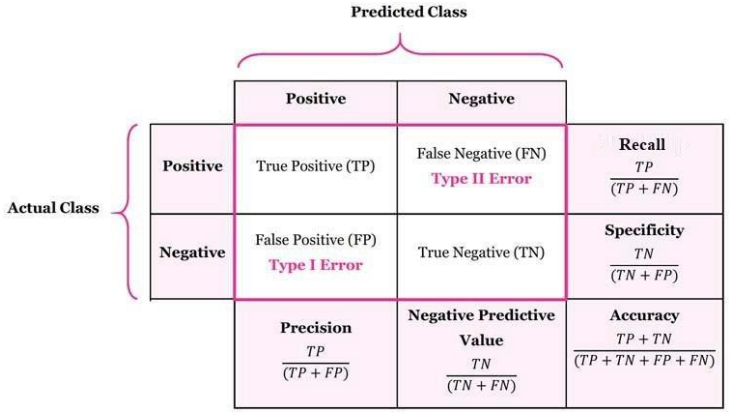

Fig. 16. Confusion matrix for binary classification. Adapted from [33][1]

Equation (31) is used to evaluate DBN_1, as each digit is considered as a binary classification problem. Moreover, equation (32) is used to evaluate the "overall accuracy" of DBN_{2B1} and DBN_{2B2}, since it is how accuracy is defined on multi-classification problems.

$$Accuracy_2 = \frac{Correct\ Predictions}{Total\ Cases} = \frac{\sum_{i=1}^{i=K} cell_{ii}}{\sum_{i=1}^{i=K}\sum_{j=1}^{j=k} cell_{ij}} \quad (32)$$

Equation (32) represents the sum of all correctly predicted classes over all predictions made by the model and is demonstrated in Fig. 17.

[1] In statistics, "recall" is referred as "sensitivity".

		PREDICTED classification				
	Classes	a	b	c	d	Total
ACTUAL classification	a	6	0	1	2	9
	b	3	9	1	1	14
	c	1	0	10	2	13
	d	1	2	1	12	16
	Total	11	11	13	17	**52**

$$Accuracy = \frac{6 + 9 + 10 + 12}{52}$$

Fig. 17. Example of accuracy used in multi-classification problems. Adapted from [34]

Now, even with the differences of accuracy measurement in DBN_1 and DBN_{2B1}, DBN_{2B2}, the relatively low accuracy of the latter two models indicate that they are worse in performance when compared with DBN_1 and DBN_3.

Comparing DBN_1 and DBN_3, however, is more difficult, as the former is evaluated using accuracy metric while the latter is evaluated using f1-score, which is the harmonic mean of recall and precision and is denoted by (33),

$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}} = 2 \times \frac{p \times r}{p + r} \tag{33}$$

where p and r are precision and recall metrics respectively.

Now, to compare between $Accuracy_1$ and F_1, we first need to understand what classification cases each of these metrics focus on when penalizing the model performance:

1. Accuracy focuses on TNs to penalize the model.
2. F1 does not focus on TNs, but rather it focuses on FPs and FNs to penalize the model, which makes it a suitable metric when training the model using imbalanced datasets.

By further observation of Fig. 15, one can notice that DBN_1 outperforms DBN_3 in the recognition of digits 2, 5, and 9 only, but DBN_3 outperforms DBN_1 in the recognition of digits 0, **3**, 6, 7, and **8** — with larger gaps of performance in recognizing 3 and 8 — while the two models perform almost identically in recognizing digits 1 and 4. Moreover, assuming that there are no detrimental consequences of misclassifying digits, then all the factors by which accuracy and f1 penalize the model should be equally important. Therefore, we can conclude that DBN_3 is the superior model.

From the comparison of the four models, one can infer possible rules-of-thumb regarding the representation of input data and hyper-parameters, which are as follows:

1. Pre-processing even if the data is relatively noise-free is still advised; one can always enhance the input data by basic image processing techniques. Moreover, data augmentation is important for training the model to recognize future unseen variations of the classes to be predicted.
2. Using statistical features of an image instead of its raw counterpart can increase the model's performance; since the data is now represented by its features, so less noise will be passed in the input layer which is now smaller (i.e., less neurons).
3. Increasing the number of nodes in the hidden layer does not guarantee higher performance, as can be shown by comparing DBN_{2B2} with DBN_1; emphasis should also be on the input layer and the size of each sample (i.e., input data).
4. Decreasing the learning rate could positively affect the model's performance, given the tolerance of longer execution times.
5. Increasing the number of epochs does not guarantee higher performance, as can be shown by comparing DBN_{2B2} with DBN_3.

III. Conclusion

This paper explained concepts related to how MRF and BM unsupervised learning was done including maximal cliques of UGM, and MLE, and KL-Divergence. It also covered the problem of intractable gradient of KL-Divergence, which led to RBM training which includes a gradient approximation technique using MCMC — specifically Gibbs sampling — and Contrastive Divergence. DBN training was then mentioned and explained. After that, applications for DBN were stated with an interest on the image classification problem of handwritten-digits, where four DBN models from three different papers were analyzed and compared to empirically infer rules-of-thumb for choosing appropriate datasets and model hyper-parameters.

References

[1] G. Hinton and T. Sejnowski, *Optimal perceptual inference*. 1983, p. 453.

[2] A. Patel and R. K. Rama, *An overview of Boltzmann Machine and its special class*. 2020. doi: 10.13140/RG.2.2.28630.88641.

[3] A. Fischer and C. Igel, *An Introduction to Restricted Boltzmann Machines*. 2012, p. 36. doi: 10.1007/978-3-642-33275-3_2.

[4] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-Time Classification and Sensor Fusion with a Spiking Deep Belief Network," *Frontiers in neuroscience*, vol. 7, p. 178, Oct. 2013, doi: 10.3389/fnins.2013.00178.

[5] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Restricted Boltzmann Machine and Deep Belief Network: Tutorial and Survey." arXiv, Aug. 05, 2022. Accessed: Oct. 20, 2022. [Online]. Available: http://arxiv.org/abs/2107.12521

[6] A. Fischer and C. Igel, "Training restricted Boltzmann machines: An introduction," *Pattern Recognition*, vol. 47, no. 1, pp. 25–39, Jan. 2014, doi: 10.1016/j.patcog.2013.05.025.

[7] J. Shlens, "Notes on Kullback-Leibler Divergence and Likelihood." arXiv, Apr. 07, 2014. Accessed: Nov. 03, 2022. [Online]. Available: http://arxiv.org/abs/1404.2000

[8] R. Ranganath, D. Tran, J. Altosaar, and D. Blei, “Operator Variational Inference,” in *Advances in Neural Information Processing Systems*, 2016, vol. 29. Accessed: Nov. 03, 2022. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/d947bf06a885db0d477d707121934ff8-Abstract.html

[9] C. Andrieu, C. Andrieu, N. De Freitas, A. Doucet, and M. Jordan, “An Introduction to MCMC for Machine Learning,” p. 39, 2003.

[10] H. Seyr and M. Muskulus, “Decision Support Models for Operations and Maintenance for Offshore Wind Farms: A Review,” *Applied Sciences*, vol. 9, Jan. 2019, doi: 10.3390/app9020278.

[11] B. Ghojogh, H. Nekoei, A. Ghojogh, F. Karray, and M. Crowley, “Sampling Algorithms, from Survey Sampling to Monte Carlo Methods: Tutorial and Literature Review.” arXiv, Nov. 02, 2020. Accessed: Nov. 03, 2022. [Online]. Available: http://arxiv.org/abs/2011.00901

[12] M. Carreira-Perpinan and G. Hinton, *On contrastive divergence learning*. 2005.

[13] F. D’Angelo and L. Böttcher, “Learning the Ising Model with Generative Neural Networks,” *Phys. Rev. Research*, vol. 2, no. 2, p. 023266, Jun. 2020, doi: 10.1103/PhysRevResearch.2.023266.

[14] M.-A. Côté and H. Larochelle, “An Infinite Restricted Boltzmann Machine.” arXiv, Mar. 18, 2016. Accessed: Nov. 03, 2022. [Online]. Available: http://arxiv.org/abs/1502.02476

[15] Z. Younas, Z. Niu, S. Sandiwarno, and R. Prince, “Deep learning techniques for rating prediction: a survey of the state-of-the-art,” *Artificial Intelligence Review*, vol. 54, Jan. 2021, doi: 10.1007/s10462-020-09892-9.

[16] A. Tammaa, *A Literature Review and Comparative Analysis for Restricted Boltzmann Machine and Deep Belief Neural Network*. 2022. doi: 10.13140/RG.2.2.19344.69127.

[17] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy Layer-Wise Training of Deep Networks,” in *Advances in Neural Information Processing Systems*, 2006, vol. 19. Accessed: Nov. 05, 2022. [Online]. Available: https://proceedings.neurips.cc/paper/2006/hash/5da713a690c067105aeb2fae32403405-Abstract.html

[18] X. Dai *et al.*, “Deep Belief Network for Feature Extraction of Urban Artificial Targets,” *Mathematical Problems in Engineering*, vol. 2020, p. e2387823, May 2020, doi: 10.1155/2020/2387823.

[19] H. Zang *et al.*, “Application of functional deep belief network for estimating daily global solar radiation: A case study in China,” *Energy*, vol. 191, p. 116502, Jan. 2020, doi: 10.1016/j.energy.2019.116502.

[20] N. A. Agana and A. Homaifar, “EMD-Based Predictive Deep Belief Network for Time Series Prediction: An Application to Drought Forecasting,” *Hydrology*, vol. 5, no. 1, Art. no. 1, Mar. 2018, doi: 10.3390/hydrology5010018.

[21] R. Sarikaya, G. E. Hinton, and A. Deoras, “Application of Deep Belief Networks for Natural Language Understanding,” *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 22, no. 4, pp. 778–784, Apr. 2014, doi: 10.1109/TASLP.2014.2303296.

[22] B. Ayhan and C. Kwan, “Application of Deep Belief Network to Land Cover Classification Using Hyperspectral Images,” in *Advances in Neural Networks - ISNN 2017*, Cham, 2017, pp. 269–276. doi: 10.1007/978-3-319-59072-1_32.

[23] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in *Proceedings of the 26th Annual International Conference on Machine Learning - ICML ’09*, Montreal, Quebec, Canada, 2009, pp. 1–8. doi: 10.1145/1553374.1553453.

[24] M. Abu Ghosh and A. Maghari, *A Comparative Study on Handwriting Digit Recognition Using Neural Networks*. 2017, p. 81. doi: 10.1109/ICPET.2017.20.

[25] C. Dewi, R.-C. Chen, Hendry, and H.-T. Hung, “Comparative Analysis of Restricted Boltzmann Machine Models for Image Classification,” in *Intelligent Information and Database Systems*, Cham, 2020, pp. 285–296. doi: 10.1007/978-3-030-42058-1_24.

[26] H. Gm, M. Gourisaria, M. Pandey, and S. Rautaray, “A comprehensive survey and analysis of generative models in machine learning,” *Computer Science Review*, vol. 38, p. 100285, Jul. 2020, doi: 10.1016/j.cosrev.2020.100285.

[27] Y. LeCun and C. Cortes, “MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges,” 2010. http://yann.lecun.com/exdb/mnist/ (accessed Nov. 07, 2022).

[28] E. Alpaydın and C. Kaynak, “Cascading Classifiers,” *Kybernetika*, vol. 34, pp. 369–374, Jul. 1997.

[29] E. Alpaydin and C. Kaynak, “UCI Machine Learning Repository: Optical Recognition of Handwritten Digits Data Set,” Jul. 01, 1998. http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits (accessed Nov. 07, 2022).

[30] P. Patidar, “Optical recognition of handwritten digits dataset.” 2019. Accessed: Nov. 07, 2022. [Online]. Available: https://kaggle.com/code/paraspatidar/optical-recognition-of-handwritten-digits-dataset

[31] “TimoMatzen/RBM: Package for fitting RBM and DBN models in R. version 0.0.0.9000 from GitHub.” https://rdrr.io/github/TimoMatzen/RBM/ (accessed Nov. 07, 2022).

[32] GM-git-dotcom, “GenerativeModels.” Jun. 27, 2020. Accessed: Nov. 07, 2022. [Online]. Available: https://github.com/GM-git-dotcom/GenerativeModels

[33] N. Shajihan, *Classification of stages of Diabetic Retinopathy using Deep Learning*. 2020. doi: 10.13140/RG.2.2.10503.62883.

[34] M. Grandini, E. Bagli, and G. Visani, “Metrics for Multi-Class Classification: an Overview.” arXiv, Aug. 13, 2020. Accessed: Nov. 08, 2022. [Online]. Available: http://arxiv.org/abs/2008.05756