ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse307942020.pdf
https://doi.org/10.30534/ijatcse/2020/307942020

Image Noise Type Identification in Microarray Images Using CNN-INC

Priya Nandihal [1], Dr. Vandana Sreenivas[2], Dr. Jagadeesh Pujari[3]
[1] Research Scholar, SDMCET, INDIA,talk2priya.nandihal@gmail.com
[2] Assistant Professor, SDMCET, INDIA, vsreenivas6@gmail.com
[3] Professor, SDMCET, INDIA, jaggudp@gmail.com

ABSTRACT
This paper presents a new method to identify the types of noise present in microarray images based on Convolution Neural Network known as CNN-INC. It does not demand any pre-processing of noisy images except resizing if needed. The size of dataset and the parameters of training options are selected to achieve 100% test accuracy (Zero ERROR).The CNN training speed is maintained substantially higher without compromising on accuracy. Experimental analysis shows that the proposed algorithm gives promising results compared to existing methods.

Key words: Microarrays, Noise types**,** Convolution Neural Network, Confusion matrix.

1. INTRODUCTION

Image noise is an undesirable, but unavoidable affliction that degrades and distorts digital images. Intrusion of noise can occur during image acquisition, processing, transmission and display. The presence of noise in a digital image diminishes its visual quality by masking the finer details. Noise in a medical image impairs its diagnostic capabilities. Other adverse effects of noise in digital images [1] are well known. Therefore, denoising is the most important pre-processing step in any image processing endeavor. Many efficient and popular denoising methods [1] are available. However the effective working of these denoising methods depends on the specific characteristics of the afflicting noise. For the purpose of understanding, analysis and assisting in denoising, image noises are classified into several types [1]-[5] based on their qualitative features. Some examples for noise types are, Gaussian, Salt & Pepper, Speckle, and Poisson and so on. Once the type of noise is known, it is much easier to eliminate it fully. There is no need for multi-stage searching and filtering to arrive at the correct noise type. Thus, to determine the type of noise is a crucial phase in the denoising process. In this work, we present a new method of noise-type detection (identification) based on deep learning. We use Convolutional Neural Network (CNN), as the deep learning tool, to find the type or class of noise present in Microarray noisy images. The proposed new method is designated as CNN-INC which stands for **C**onvolutional **N**eural **N**etwork for **I**mage **N**oise **C**lassification. In CNN-INC, the successive layers of the Convolutional Neural Net are specially designed for image noise type classification instead of general image classification [6].

2. RELATED WORKS

A few papers are available on image noise type identification. Chen and Das [7] have described noise type identification based on statistical features. In [7], noise samples are extracted from the noisy image and using them kurtosis and skewness are obtained. Based on these statistical characteristics, the noise type is determined. The problem in this approach is the need for multiple filters to separate noise samples from the noisy image. It is a sort of trial and error process. Vasuki et al. [8] have used statistical features of noise samples as the input to a Feed Forward Neural Network that acts as a pattern classifier. In this case also the extraction of noise samples is a trial and error process involving multiple filters. In [9], Wang et al. use Singular Value Decomposition, Wavelet- transformation and a Back Propagation Neural Network to classify the noisy type. This method is computationally expensive and suited for white noise and Gaussian blur. It may not work for all types of noises. Dibakar Sil et al. [10] use CNN for image noise type classification using the existing standard CNNs VGG-16 and Inception-v3.In [11], Balsiger et al explores Spatio-temporal space for reconstructing magnetic resonance fingerprinting using CNN. But the architecture of these CNNs are mainly designed for object detection/classification and not specially built for image noise type classification. Therefore the training accuracy cannot reach very high values compared to our proposed method CNN-INC. In [12], Zhang et al. have introduced FFDNet whose main objective is denoising of AWGN noise. Here, the noise type detection is restricted to Gaussian only whereas CNN-INC detects other common types of noise also. FFDnet requires fine-tuned noise level maps and down-sampled sub-images as the inputs, thus involves additional preprocessing unlike our CNN-INC which does not demand any pre-processing of noisy images except resizing.

3. RESEARCH METHOD

Microarray (MA) Images are special class of images which represent the expression levels of the corresponding genes in a compact and organized 2d format [13]. A typical MA image is shown in Figure 1 (a). The main-array has 16 sub-arrays. A single sub array in color and its gray scale equivalent are shown in Figure 1(b) and 1(c) respectively. In our CNN_INC method, for easy usage, we use gray scale MA sub-array images as the candidates for denoising. In our scheme, an MA image or simply an 'image' means a gray scale MA sub-array image.

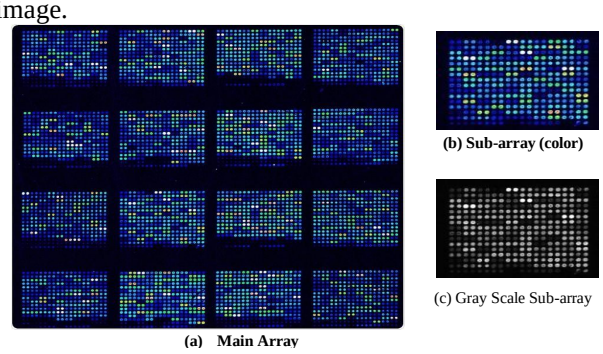

Figure 1: A typical MA image

(Credit: Andre Nantel/ Shutterstock.com)

3.1 Types of Image Noises

In CNN-INC, the following image noise types are considered for classification as shown in Table 1

Table 1: Types of Noises considered for classification

Noise Type Id or Noise Class Represented by *q*	Types of Noises Represented by *noise*(*q*)	Mean	Variance	Density
1	No noise (clean image)	----	----	----
2	Gaussian	0	0.15	-----
3	Salt & pepper	----	----	0.15
4	Poisson	----	----	----
5	Speckle	----	0.15	----
		MIXTURES		
6	Gaussian +	0	0.15	----
	Salt & pepper	----	----	0.15
7	Gaussian +	0	0.15	----
	Speckle	----	0.15	----
8	Salt & pepper+	0	----	0.15
	Speckle	----	0.15	----

The Noise type Id or Noise Class is represented by *q*.Here, *q* varies from 1 to Q where Q is the total number of classes. In the present case, Q = 8 that represents 8 different types of noises. Symbol *noise*(*q*)represents the corresponding noises. For example,

noise(1)= "No Noise" (see row 1 of Table 1)
noise(2)= "Gaussian" (see row 2 of Table 1)
noise(3)= "Salt & pepper" (see row 2 of Table 1)

---

noise(q) = Noise represented by class *q*

---

noise(Q)= *noise*(8) = "Salt & pepper and Speckle" (see row 8 of Table 1)

The objective is to determine the type (or class id) of noise, represented by *q*, which is present in the given noisy microarray (MA) image.

3.2 Clean and Noisy Microarray Images

Clean MA Images are represented by A{i}'s for i = 1 to N where N is the total number of distinct clean images. These A{i}'s can be called mother images. Each mother image A{i} generates Q noisy images (daughter images)represented by B{i, q}'s for q = 1 to Q as shown in the block diagram of Figure 2. A{i}'s and B{i, q}'s are grayscale images represented by the corresponding matrices of size HxW. The elements of these matrices belong to the data type *uint8*.

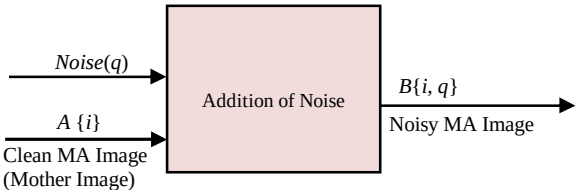

Figure 2: Addition of noise to clean MA images to get the noisy images

In Figure 1, the i[th] Clean Image is represented by *A*{i} for *i* = 1 to *N* where *N* is large, as we need a substantially large number of image samples (both clean and noisy) for the correct working of CNN-INC. For each A{i}, *noise*(*q*) is added for q = 1 to Q to get the corresponding *B*{*i, q*}'s (two subscript format).

Noisy images are obtained from the clean mother image using the **addNoise**(…) function which is based on the Matlabfunction**imnoise**(…). The details are shown in Table 2. Generation of noisy images from the clean mother images is given in Algorithm 1.

Algorithm 1: **Noisy Image Generation**

Inputs:*N* number of Distinct Clean Microarray Images (mother images), A{i}'s

Output: *K* number of Noisy Microarray Images, B(i, q}'s where *K* = *N***Q*

For *i* = 1 to *N*
For *q* = 1 to *Q*
// get ***B***{*i, q*} using the **addNoise**(**…**) function
B{i, *q*} = **addnoise**(***A***{i}, *noise*(*q*)) (1)
Endfor q
Endfor i

In Equation (1) of Algorithm 1, the addition of noise is carried out using the **addNoise**(…) function which uses the Matlab**imnoise(…)** function as shown in Table 2.

Table 2: addNoise(…) function using **imnoise(…)** for different types of noises

Q	*noise*(*q*)	Mean	Variance	Density	B{i, q} = **addNoise**(A{i}, noise(q))
1	No noise (clean image)	----	----	----	B{i,1} = A{i}
2	Gaussian	0	0.15	-----	B{i, 2} = **imnoise**(A{i}, 'gaussian', 0, 0.15)
3	Salt & pepper	----	----	0.15	B{i, 3} = **imnoise**(A{i}, 'Salt & pepper', 0.15)
4	Poisson	----	----	----	B{i, 4} = **imnoise**(A{i}, 'Poisson')
5	Speckle	----	0.15	----	B{i, 5} = **imnoise**(A{i}, 'Speckle', 0.15)
			Mixtures		
6	Gaussian and Salt & pepper	0 ----	0.15 ----	---- 0.15	B{i, 6} = **imnoise**(B{i 2}, 'Salt & pepper', 0.15)
7	Gaussian and Speckle	0 ----	0.15 0.15	---- ----	B{i, 7} = **imnoise**(B{i, 2}, 'Speckle', 0.15)
8	Salt & pepper and Speckle	0 ----	---- 0.15	0.15 ----	B{i, 8} = **imnoise**(B{i, 3}, 'Speckle', 0.15)

In the case of mixtures, **imnoise(…)** function is applied to an already noise-added image as shown in Table 2 for q = 6, 7 and 8.**imnoise(…)** function takes care of the data type of the image matrices and adds noise appropriately taking care of scaling/normalization [14] *etc*.

In Algorithm 1, for each i, the number of noisy images generated is Q and*i*varies from 1 to *N*. Therefore the total number of Noisy MA Images represented by K, are N*Q. Thus,

K = Total number of distinct noisy images = N*Q (2)

For satisfactory working of CNN-INC, the images used should be distinct. Therefore, the clean images (used to generate noisy images)are selected from a large dataset repository [MicroZip] and properly resized.

Noisy image $B\{i, q\}$is obtained by adding noise(q) to the clean image A{i}. Since the Id or class of noise(q) is q, the Noise Class Id of $B\{i, q\}$is q itself. In CNN-INC we use the symbol c(i, q) to represent the Noise Class of $B\{i, q\}$. Thus,

c(i, q) = Noise Class of $B\{i, q\}$ = q (3)

In (4), c(i, q) = q is a scalar in the range 1 to Q for i = 1 to N. For given values of i, and q, the corresponding images and Class Ids are shown in Table 3, for i = 1 to Nand for q = 1 to Q. Thus the range of A{i} is from A{1} to A{N), that of B{i, q} is from B{1, 1} to B{N, Q}. Class Id variable c{i, q) varies from 1 to Q periodically for each successive values of *i*.

Table 3: Clean Images, Noisy Images and the corresponding Class Id's for i = 1 to N and q = 1 to Q

I	**1**				**2**				---	**N**			
A{i}	A{1}				A{2}				---	A{N}			
Q	1	2	---	Q	1	2	---	Q	---	1	2	---	Q
B{i, q}	B{1, 1}	B{1, 2}	---	B{1, Q}	B{2, 1}	B{2, 2}	---	B{2, Q}	---	B{N, 1}	B{N, 2}	---	B{N, Q}
c(i, q)	1	2	---	Q	1	2	---	Q	---	1	2	---	Q

From Table 3, we see that a single unique clean image $A\{i\}$ generates a set of Q (in the present case, Q = 8) noisy images, $B\{i, 1\}$ to $B\{i, Q\}$.Therefore, noisy images [B{i, 1}, B{i, 2},…, B{i, Q}] can be treated as the noisy children of A{i}. In Table 3, subscript i increase progressively from 1 to N. Therefore, $B\{i, q\}$ can be treated as the q^{th} element of the i^{th} cycle. This fact holds for all i' s from i = 1 to N.

3.3 Special Noisy Microarray Images B(i,1)

An important feature of $B\{i, 1\}$'s for i = 1 to N, is that no noise is added to get $B\{i, 1\}$'s from A{i}'s and hence $B\{i, 1\}$= A{i} for all values of i from 1 to N. That means, for each A{i}, out of its Q noisy children B{i, 1) is a child with zero noise. Since, This special property of B{i, q}'s for q = 1 is expressed as,

B{i, 1} = A{i} = Clean Image itself, for i = 1 to N (4)

This special property is intentionally incorporated as will be explained later in the training of the CNN-INC net

Functional relation between noisy image and its clean version:-

From (1), we know that the pixel values of noisy image B{i, q} depends on its clean version A{i} and the added noise(q) which is identified by its class Id *q*. Therefore the noisy image B{i, q} can be expressed as,

$$B\{i, q\} = F(A\{i\}, q) \quad (5)$$

where,i varies from 1 to N and q from 1 to Q. Here F is a non-linear function with a high degree of non-linearity. Assuming that matrix $B\{i, q\}$is unique over i and q (This is assured when the size of B{i, q} is large), Equation (5) can be used to express q as,

$$q = F^{-1}(A\{i\}, B\{i, q\}) \quad (6)$$

Here, $F^{-1}(\ldots)$ is the functional inverse of F(…).

From (3) and (6),

$$q = c(i, q) = F^{-1}(A\{i\}, B\{i, q\}) \quad (7)$$

Substituting for A{i} in (7) from (4), we get,

$$q = c(i, q) = F^{-1}(B\{i, 1\}, B\{i, q\}) \quad (8)$$

Since B{i, 1} is constant over q = 1 to Q, Equation (8) can be reformulated as,

$$q = c(i, q) = E(B\{i, q\}) \quad (9)$$

Where,

$$E(B\{i, q\}) = F^{-1}(B\{i, 1\}, B\{i, q\}) \quad (10)$$

Here, E(…) is a complex piecewise non-linear function that relates B{i, 1} to q.
Equation (9) means, the Noise Class Id *q* can be theoretically calculated by knowing ***B***{i, q}'s. This fact is used to explain how the CNN-INC method extracts the Class Id of the noise from B{i, q}'s.

3.4 Convolution Neural Network

A representative diagram of a CNN is shown in Figure 3.A CNN is formed by inter connected neurons in multiple cascaded layers. . For details refer [15].

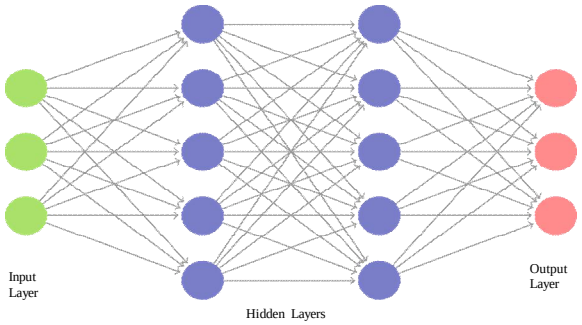

Figure 3: Representative Diagram of a CNN (Courtecy: Holzinger, et al. [16])

In CNN-INC, the CNN is used as a classifier which is designated as CNN-INCnet. The basic block diagram of the CNN-INCnet is shown in Figure 4.

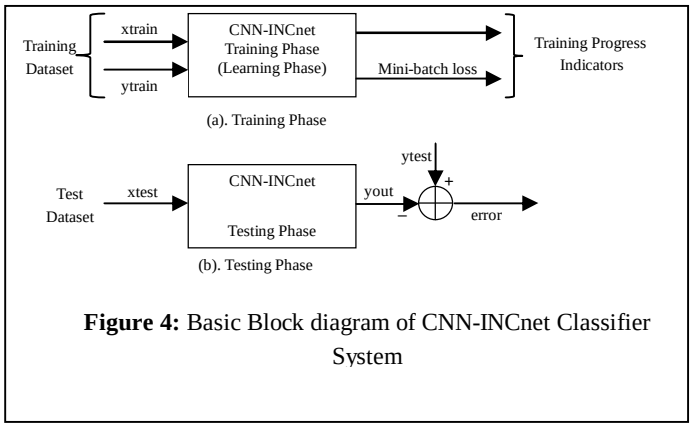

Figure 4: Basic Block diagram of CNN-INCnet Classifier System

Solving a classification problem using CNN-INC has 4 phases.

3.5 Four Phases of CNN-INC

The four phases are:

1. Creation or setup of CNN_INCnet
2. Training Phase of CNN_INCnet
3. Testing Phase of CNN_INCnet
4. Application of CNN_INCnet

3.5.1. Creation of CNN-INCnet

CNN uses layered architecture with different functionalities for different layers. In this work, CNN-INCnet has 7 convolution layers along with the input layer, the output layer (classification Layer) and other necessary layers. A short description of different layers used in CNN-INCnet is listed below. For further details about various layers see references [17]-[20].

1) ImageInputLayer([H,W, 1])
The input layer accepts the noisy input images of size H*W. The CNN-INCnet requires the input to be a 3D array. Hence, H*W*1 is used in place of just H*W. TheimageInputLayer([H,W,1]) function builds the Image Input Layer.

2) Convolution Layer
This layer constructs the first2d convolution layer which has 10 convolution filters of size 5x5. Padding by zeros is used to keep the size of the output of this layer same as that of its input.

Mini-batch accuracy

3) Batch Normalization Layer
The function of this layer is to normalize each input stream and it is inserted between the convolution2dlayer and the succeeding ReLulayer. The batch normalization layer constructs the First Batch Normalization Layer [17].

4) Relu Layer
Relu stands for Rectified Linear Unit. This layer performs nonlinear activation. The signal transfer function of Relu layer and that of an ideal single diode rectifier are same. Statement 'reluLayer' constructs this layer.

5) Maximum Pooling Layer
The max pooling layer down-samples the input by dividing it into rectangular regions and it computes the maximum of each region.

6) Average Pooling Layer
This layer is similar to the maxPooling Layer, but extracts the average of each pool instead of its maximum value.

7) Fully Connected Layer
Here, all the neurons from the previous layer are exhaustively connected as the inputs to this layer.

8) Softmax Layer
Softmax layer implements softmax(…) function [21] that calculates the matching probabilities of different classes with respect to the input class in a multi-class classification problem.

9) Classification Layer
The ultimate layer of the CNN-CC is the ClassificationLayer. This Layer receives the probability values from the previous softmax layer and assigns each value to the corresponding class out of Q distinct classes of the problem under consideration. Once created, CNN-INCnet is ready for Training, Testing and Application.

3.5.2. Training Phase of CNN-INCnet

In CNN-INC, Noisy image matrices B{i, q}'s and the corresponding integer sequence c(i, q)'s [see Table 3] for i = 1 to N, form the dataset for training/testing. To separate the training and test dataset, the range of the subscript variable i is split into two sub ranges as,

$$i = [1 \text{ to } N] = [\ [1 \text{ to } R], [R+1 \text{ to } N]\ ] \quad (11)$$

Data from the first sub range [1 to R] is used for testing and the data from the remaining range [R+1 to N] is used for testing. In (10), R is set at (about) 90% of N and the length of the remaining range, represented by S is (N – R) which is shown in Table 4. The corresponding training and test data, B{i, q}'s and c(i, q}'s are also shown in Table 4. Here, for each i, the q varies from 1 to Q. Thus a given i, we have Q number of B{i, q}'s and c(i, q)'s.

Table 4: Training and test data set ranges

Total count	N							
Sub-range Count	**R = 90% of N**				**S = N – R**			
Operation	**Training (i = 1 to R)**				**Testing (i = R+1 to N)**			
I	**1**	**2**	**---**	**R**	**R+1**	**R+2**	**---**	**N**
B{i, q}'s	**B{1, q}'s**	**B{2, q}'s**	**---**	**B{R, q}'s**	**B{R+1, q}'s**	**B{R+2, q}'s**	**---**	**B{N, q}'s**
c{i, q}'s	**c{1, q}'s**	**c{2, q}'s**	**---**	**c{R, q}'s**	**c{R+1, q}'s**	**c{R+2, q}'s**	**---**	**c{N, q}'s**

CNN-INCnet is trained by applying two inputs,*xtrain* and *ytrain*as shown in Fig.2. Inputs *xtrain* and *ytrain* are the training dataset for CNN-INCnet and it learns the causal relation between them in the form of a function [18, 27]

$$ytrain\{i, q\} = G(xtrain\{i, q\}) \quad (12)$$

In CNN-INC, xtrain{i, q}, the input to function G(…) , is B{i, q} and ytrain{i, q} is s c(i, q) for i = 1 to R . That is,

$$xtrain(i, q) = B\{i, q\} \quad \text{for } i = 1 \text{ to } R \quad (13)$$

$$ytrain\{i, q\} = c(i, q) \quad \text{for } i = 1 \text{ to } R \quad (14)$$

Then, from (4), (13), (14) and (12),

$$q = c(i, q) = G(B\{i, q\}) \text{for } i = 1 \text{ to } R \quad (15)$$

Comparing (15) and (10), it can be seen that function G(…)represents E(…) and the recovery c(i, q) from B{i, q} is logically justified. Therefore, after proper training the CNN-INCnet using B{i, q}'s for i = 1 to R as the training input (xtrain) and c{i, q}'s (also for i = 1 to R) as the response input (ytrain), it is possible to recover the class Id'sof B{i, q}'s for i> R or any other noisy image.

For the CNN-INC net, the input training data has to be 4 dimensional [22, 23]. Therefore the actual training data set has to be reshaped. In CNN-INC, the size of B{i, q} matrix is HxW. The total number of matrices used for training, corresponding to i = 1 to R and q = 1 to Q is R*Q. Hence the total size of the training data set due to B{i, q}'s is H*W*R*Q. Then, the training set is reshaped, according to the requirement of CNN as,

$$xtrainCNN = reshape(B\{i, q\}|_{i = 1 \text{ to } R \text{ and } q = 1 \text{ to } Q}, H, W, 1, R*Q) \quad (16)$$

Since c(i, q) is a scalar, the size of dataset due to c(i, q)'s for i = 1 to R and q = 1 to Q is R*Q. Therefore they train set is reshaped according to the CNN requirement as,

$$ytrainCNN = reshape(c(i, q)|_{i = 1 \text{ to } R \text{ and } q = 1 \text{ to } Q}, R*Q, 1) \quad (17)$$

Here, ytrainCNN is a column vector of length R*Q. Once the CNN-INCnet layers are selected and training data set ready, as given by (13) and (14), the training options [21] are chosen before starting the training process.

3.5.3 Testing Phase for CNN-INCnet

During testing, the input to CNN-INCnet is the data set xtest which is formed by the samples, B{i, q}'s for i = R+1, R+2, and so on up to N as already shown in Table 4. The xtest samples are enumerated as,

$$xtest\{1, q\} = B\{R+1, q\}$$
$$xtest\{2, q\} = B\{R+2, q\}$$
$$\text{----------}$$
$$xtest\{S, q\} = B\{R+S, q\}$$

Where S = N–R = total number of test samples. Introducing its own index j, for xtext values, the above relation is expressed as,

xtest{j, q} = B{R+j, q} = B{i, q}
(18)

for j = 1 to S and i = R+1 to R+S. Index j of (16) is related to index i as j = i – R for i = R+1 to R+S where R+S = N. In (18), index j varies from 1 to S and for each j, the second index q varies from 1 to Q , therefore the size of the dataset **xtest** is
size(**xtest**) = S*Q (19)

This information is required for reshaping **xtest** as needed by the `classify(net,xtest)` function.

Similarly, ytest{j, q} is given by,

ytest{j, q} = c{R+j, q} = c{i, q} = c{R+j, q} (20)

ytest{j, q}'s are already known and used to determine the error incurred by CNN-INCnet if any. In (20), index j varies from 1 to S and for each j, the second index q varies from 1 to Q, therefore the size of the dataset **ytest** is

size(**ytest**) = S*Q
(21)

The data set **ytest**can be treated as a matrix of size SxQ whose elements are ytest{j, q}'s
During the test phase, the output from CNN-INCnet is yout{j, q} which corresponds to the input xtest{j, q} as,

yout{j, q} = G(xtest{j, q}) = G(B{i, q}) (22)

The function G(…) has been learnt by CNN_INCnet during training phase. If there is no error (CNN-INCnet is trained perfectly), from (22) and (15),
yout{j, q} = c{R+j, q) = q (23)

for j = 1 to S and q = 1 to Q. (size of **yout** is S*Q). When (23) is satisfied for all j's and q's, it is assured that the CNN-INCnet is trained perfectly. Under this condition the error{j, q} (See Fig. 4) would be zero. The error term given by,

error{j, q} = ytest{j, q} – yout{j, q} (24)

for j = 1 to S and q = 1 to Q. If the training of CNN-INCnet is not sufficient, error{j, q} will be non-zero. The test ERROR is taken as the total number of non-zero terms in error{j, q}'s. That is,

ERROR = Number of non-zero terms of error{j, q}'s over j = 1 to S and q = 1 to Q
(25)

Essentially, ERROR gives the number of misclassifications out of a total of S*Q samples. Therefore the percentage error is given by,
%ERROR = 100*ERROR/(S*Q)
(26)

In CNN-INC, we ensure that the classification ERROR is zero. If not, the CNN-INCnet is retrained with suitable fine tuning.
For the test phase, the xtest input should be reshaped according to the requirement of the standard `classify(...)` function as,

xtestCNN = reshape(xtest{j, q}$_{j = 1 \text{ to } S \text{ and } q = 1 \text{ to } Q}$, H, W, 1, S*Q) (27)

During the test phase, the actual output from CNN-INCnet, designated as **youtCNN** is,

`youtCNN` =`classify(net,` **`xtestCNN`**`)` (28)

The size of **youtCNN** is S*Q and it is reshaped to get **yout**and yout{j, q}'s for calculating the error as given by (24).

2.5.4 Testing Phase for CNN-INCnet

After the training and testing (with zero error) of CNN-INCnet has been completed, it is competent to classify the noise type of the present noisy micro-sub-array image.Let us denote the present Sub-Array Noisy Image by ***ni***. Let ***ni*** be contaminated by one of the noise types as listed in Table 1. Let the exact noisy type be represented by q(***ni***). Here, q(***ni***) is unknown but it is given that q(***ni***) belongs to the range 1 to Q. Since, CNN-INCnet trained/tested using gray scale images of size HxW, an important requirement of NI is it should also be a grayscale image of size WxN.Now, ni is reshaped to match the input size of CNN-INCnet as,

niCNN = reshape(***ni***, H, W, 1, 1) (29)

Then, ***niCNN*** is applied as the input argument to the function `classify(...)` to get the noise class (type) of q(***ni***)as,

q(***ni***) = `classify(net,`***`niCNN`***`)` (30)

q(***ni***) obtained from (30) is an integer in the range 1 to Q and once q(***ni***) is found its noise type is determined from Table 1.

The basic working of CNN_INC is given in Algorithm CNN-INC

Algorithm CNN-INC

Input: Large set of noise free Micro Array (MA) images.
Output: Noise type of a given noisy micro sub-array image.

1. Select a large N which is the number of micro sub-arrays for training/testing
2. Get N number of distinct clean grayscale sub-arrays images from the given MA image set

3. Add different types of noises as given in Table 1, so that each clean image spawns Q number of noisy images. // total number of noisy images would be N*Q
4. Split N*Q noisy images into two groups as R*Q and S*Q for training and testing respectively where R = round(0.9* N) and S = N–R.
5. Construct CNN-INCnet with different layers as described in earlier section 3.5.1
6. Prepare the data set for training as explained in section 3.5.2
7. Train CNN-INCnet
8. Test CNN-INCnet to determine the error.
9. If error is zero,
   goto step 11
   else
10. Adjust the CNN_INCnet parameters appropriately towards zero error and goto step 7.
11. Training/testing completed.
    //CNN-INCnet is ready for classifying the noise type of an unknown noisy sub-array image
12. Apply the unknown noisy sub-array image as input to CNN-INCnet to get its noise type.
13. Over.

---

4. RESULTS AND DISCUSSION

Table 5 shows the parameters used in a basic CNN-INC experiment. Typical values [22] are used for those parameters of CNN-INCnet that are not given in Table 5. Image height and width means the size of the corresponding matrix in terms of number of rows and columns of that matrix.

Table 5: VALUES OF VARIOUS PARAMETERS/VARIABLES

PARAMETERS/VARIABLES	SYMBOL	VALUES
IMAGE HEIGHT	H	160
IMAGE WIDTH	W	256
NUMBER OF NOISE TYPES OR CLASSES (SEE TABLE 1)	Q	8
NUMBER OF CLEAN MOTHER IMAGES USED TO GET CNN-INC DATA SET	N	3,840
FIRST PART OF N USED FOR TRAINING	R	3,600
SECOND PART OF N USED FOR TETING	S	240
NOISE CLASS IDENTIFIER	Q	∈ {1 TO Q}
NO. OF TRAINING DATASETS	R*Q	28,800
NO.OF TEST DATASETS	S*Q	1,920
TOTAL NUMBER OF DATA SETS	N*Q	30,720
DATA DIMENSIONS FOR INPUT LAYER	H, W, 1	160, 160, 1
MAX. NO. OF EPOCHS [36]	-----	50

4.1. Experiment 1

In this experiment, the CNN-INCnet is constructed as described in section 3.5.1 using the parameters of Table 5 and it is trained as in section 3.5.2. During the, the training process, Mini-batch Accuracy and Mini-batch Loss [24] are progressively displayed for successive iterations.

Here, CNN-INC parameter Q is varied from Q = 2 to 8 with other parameters same as in Table 5. Here,CNN-INC parameter Q is varied from Q = 2 to 8 with other parameters same as in Table 5. Now, in each case, CNN-INCnet is trained to determine the minimum number of epochs and the training iterations needed to achieve 100% accuracy with zero error on testing. For Q = 2, the training progress information is shown in Table 6

Table 6: VARIATION OF TRAINING PARAMETERS FOR Q = 2

EPOCH	ITERATION	TIME ELAPSED (HH:MM:SS)	MINI-BATCH ACCURACY	MINI-BATCH LOSS	BASE LEARNING RATE
1	1	00:00:02	51.00%	0.7027	0.0020
1	50	00:00:53	100.00%	0.1424	0.0020
2	85	00:01:28	100.00%	0.0968	0.0020

CNN-INC net training/testing for zero error is continued for Q = 3, 4, 5, 6, 7, 8. As a sample, the training progress table for Q = 6 is show in Table 7. The training progress tables for other Q's are similar.

Table 7: VARIATION OF TRAINING PARAMETERS FOR Q = 6

EPOCH	ITERATION	TIME ELAPSED (HH:MM:SS)	MINI-BATCH ACCURACY	MINI-BATCH LOSS	BASE LEARNING RATE
1	1	00:00:07	10.00%	1.8751	0.0020
1	50	00:00:57	51.00%	1.3043	0.0020
1	100	00:01:47	77.00%	1.0289	0.0020
1	150	00:02:39	78.00%	0.8957	0.0020
1	200	00:03:29	99.00%	0.7294	0.0020
2	250	00:04:19	96.00%	0.6599	0.0020
2	300	00:05:10	100.00%	0.5308	0.0020
2	350	00:06:00	100.00%	0.4048	0.0020
2	400	00:06:52	100.00%	0.3196	0.0020
3	450	00:07:42	100.00%	0.2245	0.0020
3	500	00:08:34	100.00%	0.1572	0.0020
3	550	00:09:26	100.00%	0.1085	0.0020
3	600	00:10:16	100.00%	0.0843	0.0020
4	649	00:11:12	100.00%	0.0642	0.0020

The summary of the result is shown in Table 8. Training duration, which is machine dependent, is also shown in Table 8 for understanding the effect of variable Q on Training

Duration. The third column of Table 8 gives the minimum number of training iterations needed just to reach the zero error status in test phase.

Table 8: Minimum Number of Training Iterations versus Number of Classes (Q)

No. of Classes (Q)	No. of Epochs	Min No. of Iterations for Zero Error	Training Duration	No. of Images in Training set R*Q = 3600*Q	No. of Images in Testing set S*Q = 240*Q
2	2	85	1 min 28 seconds	7,200	480
3	2	207	3 min 33 seconds	10,800	720
4	3	315	5 min 23 seconds	14,400	960
5	4	543	9 min 24 seconds	18,000	1,200
6	4	649	11 min 12 seconds	21,600	1,440
7	8	1850	31 min 25 seconds	25,200	1,680
8	11	2795	50 min 53 seconds	28,800	1,920

Minimum Number of Training Iterations (MNTI) for zero error versus Q is plotted in Figure 5.In Figure 5, the plot

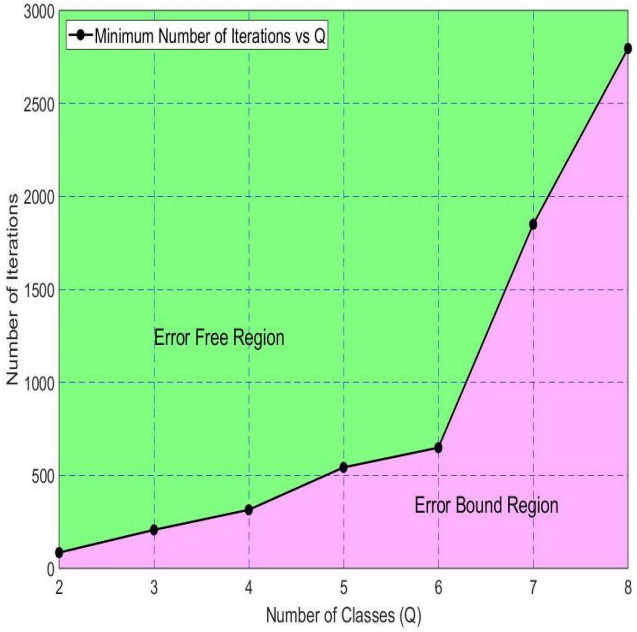

Figure 5: MNTI for zero error versus Q

divides the graph into two regions. The northwest region in green background is the error free region where the number of iterations would be greater than the minimum required. On the other hand, the southeast region in magenta background is the error bound region where the number of iterations would be lesser than the minimum required. The steep increase in MNTI for Q =7 and 8 is due to the mixed nature of noises in the noisy images as listed in Table 1.

Experiment 2

Here Q = 8 and the other parameters are same as in Table 5. The CNN-INCnet is trained for values from 480 to 1920 in steps of 480. In each case, the training process is terminated when the corresponding NTI value reaches its specified value. For successive values of NTI's the resulting classification error, denoted by ERROR in (26), is determined and is shown in Table 9. Here the total number of test samples is S*Q = 240*8 =1920.

Table 9: ERROR VERSUS NTI FOR Q = 8

NTI	480	960	1440	1920	
ERROR	82	61	12	3	
%ERROR	4.2708 %	3.1771 %	0.6250 %	0.1563 %	

From Table 9, it can be seen that the classification ERROR, during testing phase, decreases monotonically as the Number of Training Iterations increases.

Confusion Matrices

The distribution of misclassification errors over different classes, for Number of Training Iterations (NTI) = 500, 1000, 1500 and 2000are shown by the Confusion Matrices [22, 23] in Figure 6(a), 6(b), 6(c) and 6(d) respectively. For instance, In Figure 6(b), 23 samples from true class 2, are misclassified as belonging to class 7, and 38 samples from true class 7 are misclassified as belonging to class 2.

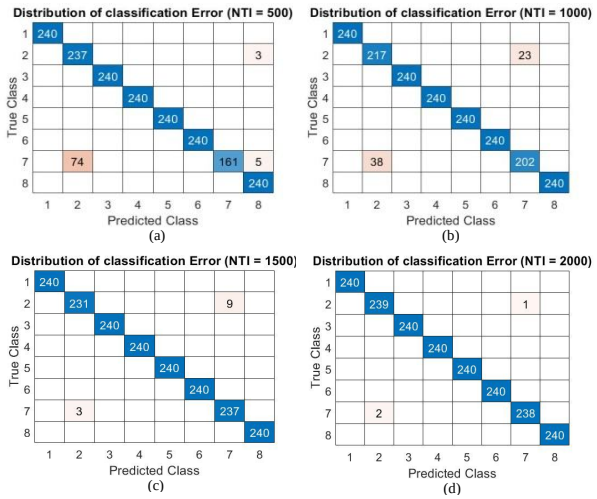

Figure 6: Confusion Matrices

Experiment 3

Our method CNN-INC is compared withVGG-16 [27] and Inveption-v3 [28] with respect to 'Training Accuracy' [29]

during training progress. Here Q is set to 7 and the maximum epoch is set to 34.

From Figure 7, it can be seen that CNN-INC reaches 100% accuracy at 30th epoch while the VGG-16 and Inception-v3 methods lag behind. The reason for the superior performance of CNN-INC is that it is specially designed for Microarray Images which have regular visible spots arranged in a grid. On the other hand VGG-16 and Inception-v3 networks are designed for general purpose image recognition and classification.

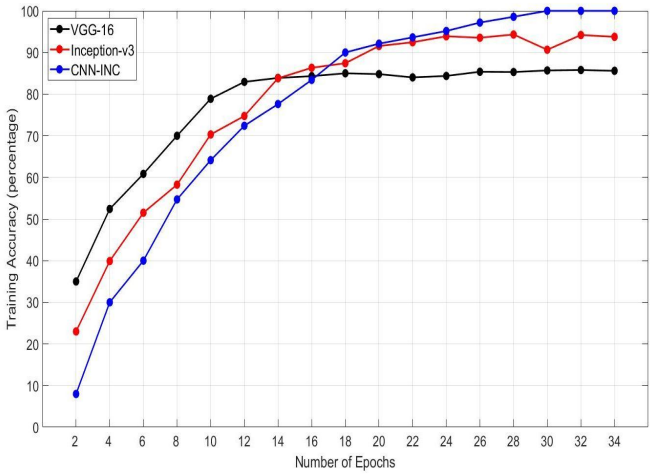

Figure 7: Percentage Training Accuracy versus Training Epochs

5. CONCLUSION

A new method, CNN-INC has been presented for the classification of different types of noises in Microarray Images. The basic noise types considered are Gaussian, Salt & Pepper, Speckle, Poisson and their mixtures. The special feature of CNN-INC is that it classifies the mixtures of noises as well as basic noise types. The CNN-INCnet is specially designed for Microarray Images which has certain uniform grid structure. At present 8 types of noises are covered for classification. Since one of the type is noise free status, CNN-INC can recognize a noise free (clean) image also. CNN-INC requires no pre-processing of images except resizing if needed. The CNN-INC method can be extended to classify additional types of noises. Compared to other methods, CNN-INC achieves 100% training accuracy and test error zero with reasonable training period.

REFERENCES

1. Rafael C. Gonzalez and Richard E. Woods. "Image Restoration and Reconstruction," Digital Image processing, 3rd Edition, Prentice Hall, August 2007. https://doi.org/10.5121/sipij.2015.6206
2. Boyat AK, Joshi B K. **A review paper: noise models in digital image processing.** *Signal & Image Processing: An International Journal(SIPI),* Vol.6,No.2,Apil 2015
3. Mandar D. Sontakke and Meghana S. Kulkarni. **Different types of Noise and Noise Removal Techniques**, *International Journal of Advanced Technology in Engineering and Science*, vol.3, No.1, pp. 102-115, January 2015.
4. Arasi, Munya & Babu, Sangita. (2019).**Survey of Machine Learning Techniques in Medical Imaging**, *International Journal of Advanced Trends in Computer Science and Engineering*, Vol 8, No. 4, pp. 2107-2116. https://doi.org/10.30534/ijatcse/2019/39852019
5. B. Dudi and V Rajesh. **Medicinal Plant Recognition based on CNN and Machine Learning**, *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 8, No. 4, 2019, pp. 999-1003. https://doi.org/10.30534/ijatcse/2019/03842019
6. Jun Liu and Feng Ping**. Image Classification Algorithm Based on Deep Learning-Kernel Function.** *Scientific Programming*, Vol. 2020,pp 1-14,Jan 2020
7. Yixin, Chen & Das, Manohar. **An automated technique for image noise identification using a simple pattern classification approach.** *50th Midwest Symposium on Circuits and Systems*, pp819-822, 2007.
8. P. Vasuki, C. Bhavana, S. Mohamed Mansoor Roomi and E. Lakshmi Deebikaa. **Automatic noise identification in images using moments and neutral networks**. *International conference on Machine Vision and Image Processing, Taipei*, pp 61-64, 2012.
9. Z. Wang, S. Wu, J. Ye and G. Yun. **Distortion types identification based on singular value decomposition and BP neural network** .*3rd International Congress on Image and Signal Processing*, Yantai, pp 2539-2542, 2010.
10. D. Sil, A. Dutta and A. Chandra. Convolutional Neural Networks for Noise Classification and denoising of Images, *2019 IEEE Region 10 Conference (TENCON)*, Kochi, pp. 447-451.
11. Balsiger, Fabian, Amaresha Shridhar Konar, Shivaprasad Chikop, Vimal Chandran, Olivier Scheidegger, Sairam Geethanath, and Mauricio Reyes. **Magnetic resonance fingerprinting reconstruction via spatiotemporal convolutional neural network,** *International Workshop on Machine Learning for Medical Image Reconstruction*, pp. 39-46. Springer, Cham, 2018. https://doi.org/10.1007/978-3-030-00129-2_5
12. Zhang, Kai &Zuo, Wangmeng& Zhang, Lei. **FFDNet: Toward a Fast and Flexible Solution for CNN based Image Denoising**, *IEEE Transactions on Image Processing*, pp. 1-15, 2018.
13. Fraser, Karl Wang, Zidong Liu. **Microarray Image Analysis: An Algorithmic Approach,** *Computer Science and Data Analysis Series*, Chapman and Hall/CRC 2017.
14. Richard E. Woods , Rafael C. Gonzalez, Eddins S.L **Digital Image Processing Using Matlab** , *Gatesmark Publishing*, Second Edition,2009
15. S. Albawi, T. A. Mohammed and S. Al-Zawi. **Understanding of a convolutional neural network**. *International Conference on Engineering and Technology (ICET)*, Antalya, pp. 1-6, 2017.

16. Holzinger, Andreas & Malle, Bernd & Kieseberg, Peter & Roth, Peter M. & Müller, Heimo & Reihs, Robert & Zatloukal, Kurt. (2017). **Machine Learning and Knowledge Extraction in Digital Pathology Needs an Integrative Approach.** *Towards integrative machine learning and knowledge extraction, CANADA,* pp 13-50,2015
17. Johan Bjorck, Carla Gomes, Bart Selman, Kilian Q. Weinberger, **Understanding Batch Normalization,** *32nd Conference on Neural Information Processing Systems, Montréal, Canada*, pp. 1-12, 2018.
18. *Albawi Saad, Tareq*. **Understanding of a Convolutional Network**. *10.1109/ICEngTechnol.2017.8308186,*2017.
19. Nielsen, M.A. (2015) ***Neural Networks and Deep Learning***. http://neuralnetworksanddeeplearning.com/
20. Jason Brownlee (2019). **A Gentle Introduction to Dropout for Regularizing Deep Neural Networks.** https://machinelearningmastery.com/dropout-for-regularizing- deep-neural-networks/.
21. Christopher Bishop. ***Pattern Recognition and Machine Learning***. Springer, New York, NY, 2006
22. Matlab help. **PreProcess images for deep learning.** http://in.mathworks.com/help/deeplearning/ug/preprocess-images-for-deep-learning/
23. Matlab help. **Train neural network for deep learning**. http://in.mathworks.com/help/deeplearning/ref/trainnetwork/
24. Jason Brownlee (2019). **A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks**. https://machine-learning-mastery.com/early-stopping-to-avoid- overtraining -neural- network-models/
25. Jason Brownlee (2019). **Make the Confusion Matrix Less Confusing**. https:// machine learning mastery. com/ confusion-matrix-machine-learning/
26. Jason Brownlee (2019). **Impact of Dataset Size on Deep Learning Model Skill And Performance Estimates.** https://machinelearningmastery.com/ impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/
27. W. Yu, K. Yang, Y. Bai, T. Xiao, H. Yao, and Y. Rui. **Visualizing and comparing AlexNet and VGG using deconvolutional layers** in Proc. International Conference on Machine Learning (ICML), New York City, NY, USA, Jun. 2016, pp. 1–7.
28. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. **Rethinking the inception architecture for computer vision** in Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, Jun. 2016, pp. 2818–2826. https://doi.org/10.1109/CVPR.2016.308
29. Matlab help. **Monitor Deep Learning Training Progress**.https://in.mathworks.com/help/deeplearning/ug/monitor-deep- learning-training-progress.html.