1. INTRODUCTION

Advancements in facial emotion recognition have surged in recent years, with such technology finding diverse applications, including emergency response and security [1, 2], video surveillance and law enforcement [3, 4], as well as access management and control [5]. However, the performance of emotion recognition systems is often challenged by variations in pose, illuminations, facial expressions, and noise interference. Comprehensive research has been conducted on the issues of expression, posture, and illumination, resulting in notable achievements. However, the presence of noise markedly decreases the accuracy of emotion recognition in most methodologies.

Noise affects facial images during stages such as compression, acquisition, transformation, and quantization. Prior to emotion recognition, various image denoising techniques can be applied [6]. However, this process may result in the loss of edge-related information, thereby hindering subsequent image recognition. Several methods, including fuzzy local binary pattern (FLBP), noise resistance LBP, and their advanced versions, have been developed to identify and eliminate noise directly from images [7, 8].

Recent studies have shown that a marginal quantity of noise added to images used to train a CNN model can, in fact, enhance performance [9]. The ability of neural networks to learn output functions independently allows for a small amount of noise at the input level without adversely affecting performance. By adding noise to the training data, networks leverage all available neural units, leading to increased stability and enhanced overall performance [10]. Noise can be added to individual units, weights, inputs, output functions, and hidden layers to improve the general performance of the neural network [11-13]. The neural network appears less sensitive to variations in weight noise when compared to variations in input noise [14]. Moreover, gradient noise addition to a 20-layer deep fully connected neural network reduces overfitting and training loss [15]. In datasets such as CIFAR-10 and MNIST, noisy autoencoders outperform denoising autoencoders [16].

In the work presented here, the proposed Facial Emotion Recognition Convolutional Neural Network (FERCNN)
model, comprising four fully connected convolution layers, two dense layers, and a flatten layer, is developed to classify facial emotion into one of seven classes. The input image dataset is transformed into a noisy image dataset, which is subsequently used to train and evaluate the model. The impact of various noise types on classification performance is experimentally examined. The accuracy and loss from the proposed FERCNN model are compared with popular CNN architectures like VGG16, VGG19, Xception, Resnet50, using the FER2013 and JAFFE datasets and the Kaggle cloud environment, which supports high GPU computational support.

The remainder of this paper is organized as follows: Section 2 discusses related work, Section 3 details the proposed model, Section 4 presents experimentation and results, and Section 5 concludes the study.

2. RELATED WORK

The conventional systems of facial emotion identification typically engage in three principal stages: face acquisition from the image, feature extraction from images, and classification of these extracted image features. The extraction of discriminative and invariant facial features is crucial to the successful identification of facial emotions. These feature extraction techniques can broadly be divided into two categories: deep features and hand-crafted features.

Gabor wavelets, which extract multi-orient and multi-scale information from an image, have been widely employed for feature extraction from faces [17]. Chen et al. [18] defined a "mother" wavelet from which 40 Gabor filters were derived, considering five scales and eight orientations. Each image was assigned to a convolution filter independently, and the outputs from all these filters were combined to form an augmented feature, termed Gabor. This feature proved robust to changes in lighting and facial emotions, yet resulted in a high-dimensional feature space. Another prevalent hand-crafted method for facial emotion identification is the Local Binary Pattern (LBP) [19, 20], which has also found use in texture classification and various other applications [21, 22].

Deep Neural Networks (DNN) [23], particularly Convolutional Neural Networks (CNN), have gained popularity for their efficacy in learning and recognizing features. Before the advent of deep learning, various approaches were employed for image classification into appropriate classes, including dance action identification [24], brain tumor identification [25-27], plant disease identification [28, 29], among other applications [30, 31]. Although CNNs prove computationally expensive and architecturally complex compared to other facial emotion recognition systems, these issues have been mitigated by recent technological advances and resource availability. Liu et al. [32] proposed a deep autoencoder network utilizing softmax regression technique in the output layer, which improved the overall performance of the network. Deep learning has been employed in recognizing facial emotions using action units and body gestures [33, 34], and also in classifying facial emotions from color image data and video data [35].

Exploring the boundaries between the well-behaved and saturated parts of the activation function can be achieved by adding noise to the saturated regions. It has been demonstrated that replacing saturated activation functions with their noisy counterparts yielded better results on various datasets, especially when training was challenging [36]. Good results were obtained when CNN versions of AlexNet were trained and tested with image datasets containing the same amount of noise [37]. Transfer learning has been employed to enhance model performance [38], with the model first trained on clean images and those with minimal noise, then subsequently on images with greater noise. When noise levels varied, some existing methods for emotion recognition did not perform as well as the dual feature fusion method [39]. HOG and CNN [40] are combinedly used in detecting facial emotions. When HOG and SVM were employed to extract the features of facial expressions and the emotions were classified using ALEXNET, the results improved [41].

In the present study, Gaussian, Salt and Pepper, Poisson, and Speckle noises were injected into the facial dataset images. The proposed FERCNN model demonstrated robust performance, even with the introduction of noise to the images.

3. PROPOSED MODEL

3.1 Methodology

It is believed that a trained model with noise injected in input facial images for classifying human facial expression using CNN is more robust than without noise. The undertaken research derived the differences between the models that functions with noise and without noise at the input level. To measure the differences, five different cases are considered which are mentioned below:

Case 1: Noise applied during training and testing the model.
Case 2: Noise not applied during training and applied during testing the model.
Case 3: Noise applied during training and not applied during testing the model.
Case 4: Noise not applied during training and testing the model.

The entire research is divided into the following stages:

Input Image Dataset→Apply Noise to the Images→Train the CNN Model→Train the CNN Model→Evaluate the Performance Accuracy

3.1.1 Used image dataset description

The renowned datasets FER2013 and JAFFE are used for experimentation in the proposed model. Fer2013 data set consists of 35887 images and JAFFE dataset which consists of approximately 3500 images having seven different facial expressions namely Happy, Angry, Sad, Surprise, Neutral, Disgust, and Fear, respectively.

3.1.2 Applied noise description

The images are applied to different types of noises and the observations after applying the images as an input to the proposed FERCNN Model is tabulated in the Experimentation and Results section. Gaussian, Salt & Pepper, Poisson, and Speckle noise methods are used to apply noise to the images and their performances under various scenarios are compared. Equations related to the noise used are mentioned below.

**Gaussian Noise**

\[ PG(Z) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \]  

where, \( PG(Z) \) represents the probability of gaussian noise
added on a random variable $Z$. $\mu$ and $\sigma$ represents the mean and standard deviation of the grey value of the given images.

**Poisson Noise**

$$y = x + \eta(x)\delta$$  \hspace{1cm} (2)

The above equation represents the image that adds generic signal dependent passion noise where the output image acquired is represented by $y$, $x$ is the given input image, $\delta$ represents gaussian noise that is independent and having mean value of zero and standard deviation value of one.

$$\eta^2(x) = \alpha x + \sigma^2$$  \hspace{1cm} (3)

The above equation represents the noise term variance which separates into noise independent and dependent components, $\sigma$ & $\alpha$ are standard deviation & passion noise parameter of gaussian noise.

**Speckle Noise**

$$P(I) = \begin{cases} \frac{I^{N-1}}{\Gamma(N)} \exp \left( -\frac{I}{I_0} \right) & I \geq 0 \\ 0 & I < 0 \end{cases}$$  \hspace{1cm} (4)

The above equation represents the probability of speckle noise added to the image $I$, $N$ represents the number of samples.

**Salt & Pepper Noise**

$$P(g) = \begin{cases} P_a & \text{for } g = a \\ P_b & \text{for } g = b \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (5)

$P(g)$ represents the probability salt & pepper noise, $a$ and $b$ represents bright and dark regions respectively, $P_a$ and $P_b$ are noise probability values in $a$ and $b$ regions.

### 3.1.3 Training and evaluation of the model

The images of the dataset after applying noise as mentioned in the above section are considered and are given for training and testing under various cases as mentioned in section 3.1. Among the described cases, Case 4 is a conventional way of classifying facial images into an appropriate class. The model trained with the applied noise during the training as well as testing phases as explained in Case 1 to Case 3. The performance metrics of above cases are compared with Case 4 and stand more robust, and the performance of the proposed FERCNN model is compared with various well-known architectures such as VGG16, VGG19, Xception and Resnet50 and found that the proposed model is better than the mentioned one. The detailed description is given in the Experimentation and Results Section.

### 3.2 Algorithm

**Algorithm 1:** Preparation of noisy dataset from the given input image dataset

Input: Image Set Consisting of Seven Different Facial Expressions

Output: NoisyImgSt Noise Apply Alg (Image Set [])

NoiseSt [] = \{\phi\}

Begin

if len(ImageSet[ ]) = 0 then/
  Begin
  for all the images in ImageSet
  resize(ImageSet[ ], 48,48)
  Normalize(ImageSet[ ])
  $l = \text{len}(\text{ImageSet}[])$
  End if
  SELECT CASE (Type of Noise)
  Begin
  Case (Gaussian Noise):
    while ($l > 0$)
    Begin loop
    imgi[ ] = randSelect(ImageSet[ ])
    gaussian = np. random. normal (mean, sigma, (48,48))
    noisy_imagei = np. zeros (imgi. shape, np. float32)
    noisy_imagei[ :, :, 0] = imgi[ :, :, 0] + gaussian
    noisy_imagei[ :, :, 1] = imgi[ :, :, 1] + gaussian
    noisy_imagei[ :, :, 2] = imgi[ :, :, 2] + gaussian
    NoisyImgSt[ ] = NoisyImgSt[ ] + noisy_imagei
    $l = l - \text{len}(\text{imgi[ ]})$
    Endloop
  Case (Salt & Pepper):
    while ($l > 0$)
    Begin loop
    imgi[ ] = randSelect(ImageSet[ ])
    noise_imagei = sp. noise (imgi, 0.005)
    NoisyImgSt[ ] = NoisyImgSt[ ] + noisy_imagei
    $l = l - \text{len}(\text{imgi[ ]})$
    Endloop
  Case (Poisson):
    while ($l > 0$)
    Begin loop
    imgi[ ] = randSelect(ImageSet[ ])
    noise_imagei = \frac{\text{poisson}(\text{imgi[ ]} + 200)}{255} * 255
    NoisyImgSt[ ] = NoisyImgSt[ ] + noisy_imagei
    $l = l - \text{len}(\text{imgi[ ]})$
    Endloop
  Case (Speckle):
    while ($l > 0$)
    Begin loop
    imgi[ ] = randSelect(ImageSet[ ])
    gaussian = np. random. normal (mean, sigma, (48,48))
    noisy_imagei = np. zeros (imgi. shape, np. float32)
    noisy_imagei[ :, :, 0] = imgi[ :, :, 0] + imgi[ :, :, 0] * gaussian
    noisy_imagei[ :, :, 1] = imgi[ :, :, 1] + imgi[ :, :, 1] * gaussian
    NoisyImgSt[ ] = NoisyImgSt[ ] + noisy_imagei
    $l = l - \text{len}(\text{imgi[ ]})$
    Endloop
  End
End
Algorithm 2: Algorithm for Classification
Input: ImgSt Consisting Images of Seven Different Facial Expressions
Output: Classification of Images Based on Expression Type
FacEmoRec(HybridImgSt)
Begin
SELECT CASE (Type of Training & Testing)
Begin
Case (Train and Test without Noise)
l = len(ImageSet)
for i in 1 to l
Begin
LabImgSt ← Label(ImageSet[i] ![imgi])
TrSt, TsSt ← Split(LabImgSt, 85, 15)
Call FERCNN Model();
End
Case (Train and Test with Noise)
l = len(NoisyImgSt)
for i in 1 to l
Begin
LabImgSt ← Label(NoisyImgSt ![imgi])
TrSt, TsSt ← Split(LabImgSt, 85, 15)
Call FERCNN Model();
End
Case (Train Without Noise and Test with Noise)
l = len(ImageSet), l1 = len(NoisyImgSt)
for i in 1 to l, l1
Begin
LabImgSt ← Label(Imageset ![imgi])
LabImgSt1 ← Label(NoisyImgSt ![imgi])
TrSt ← Select(LabImgSt, 85%)
TsSt ← Select(LabImgSt1, 15%) 
Call FERCNN Model();
End
End
End
End
FERN Model()
Begin
CNN Model ← CNN Model(TrSt)
Evaluation ← CNN Model(TsSt)
return Classified Emotion
End

4. EXPERIMENTATION AND RESULTS

4.1 Model architecture

Three fully connected layers, one flatten layer, and two dense layers make up the proposed model. The first fully connected layer has one Conv2d layer, the second fully connected layer has one Conv2d layer, one max pooling layer, and one batch normalization layer, the third fully connected layer has one Conv2d layer, and the fourth fully connected layer has one convolutional layer, one batch normalization layer, one max polling layer, and one dropout layer. The first dense layer has one dense layer and one dropout layer, while the second dense layer has only one dense layer. The kernel size for all convolutional layers was 3×3, the max polling was 2×2, and the activation functions were Relu in the hidden layers and soft max in the final dense layer. The total number of parameters used is 32,116,743, with 32,116,103 trainable and 640 non-trainable. To maintain uniformity, all images are normalized and standardized using standardization and normalization techniques before being resized to a fixed dimension of 48×48. The Detailed architecture is given in Figure 1 given below.

The training of the model is associated with high computations for which the Kaggle cloud environment that, supports CPU frequency of 2.30GHz, Two CPU core, 13GB CPU, 16GB NVIDIA GPU, Disk space of size 19.6GB, Python programming language with Jupyter notebook are used.

The number of facial images used during the training and testing phases are given below. Detailed description of various datasets used is given in Table 1.

The details of results obtained under various cases are given in Tables 2, 3 and 4 as well as the resultant graphs of the experiments are demonstrated in the figures given below.

![Figure 1. Layered architecture description](image)

<table>
<thead>
<tr>
<th>S. No</th>
<th>Data Set</th>
<th>No. of Trained Images</th>
<th>% of Trained Images</th>
<th>No. of Tested Images</th>
<th>% of Tested Images</th>
<th>No of Emotion Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FER 2013</td>
<td>28709</td>
<td>80%</td>
<td>7178</td>
<td>20%</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>JAFFE</td>
<td>2800</td>
<td>80%</td>
<td>700</td>
<td>20%</td>
<td>7</td>
</tr>
</tbody>
</table>
Figure 2(a) represents the resultant values of the confusion matrix and performance metrics when the FER2013 dataset without noise is considered and given for the proposed FERCNN model for evaluation. Whereas in Figure 2(b) represents the respective heat map, from Figure 2(a), it is observed that weighted average accuracy, precision, recall, and F1-score values are 0.61, 0.61, 0.61 & 0.61 respectively.

Figures 3 represent FERCNN model accuracy and loss comparisons under various cases such train and test with Gaussian noise, train with Gaussian noise and test without noise, train without noise and test with Gaussian noise and train and test without noise and observed that accuracy has been increased and loss has been decreased when noise is applied to the data and the details of the results are given in Table 2.

Figure 3(c) represents the resultant values of the confusion matrix and performance metrics of the proposed FERCNN model when the FER2013 dataset is injected with Gaussian noise and given for evaluation. Standard deviation and variance parameter values of 8 and 10 are as input to the Gaussian noise function to produce Gaussian noisy images. The proposed FERCNN Model resulted in weighted average accuracy of 0.92, precision of 0.92, recall of 0.92 and F1-score of 0.91 after experimentation where as Figure 3(d) represents the respective heat map of the confusion matrix.

**Figure 2.** Performance analysis of FERCNN models on FER2013 dataset without noise

**Figure 3.** Performance comparison of FERCNN model on FER2013 dataset with gaussian noise
Table 2. Performance comparison of different noises in different cases on FER2013

<table>
<thead>
<tr>
<th>S. No</th>
<th>Noise Applied</th>
<th>Mode of Training and Testing</th>
<th>Train Accuracy</th>
<th>Test Accuracy</th>
<th>Train Loss</th>
<th>Test Loss</th>
<th>Epoch Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gaussian Noise</td>
<td>Train &amp; Test with Noise</td>
<td>99.70</td>
<td>91.95</td>
<td>0.066</td>
<td>0.365</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train Without Noise &amp; Test with Noise</td>
<td>96.15</td>
<td>13.38</td>
<td>0.452</td>
<td>60.35</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train with Noise &amp; Test Without Noise</td>
<td>99.41</td>
<td>87.27</td>
<td>0.079</td>
<td>0.606</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train &amp; Test without Noise</td>
<td>62.37</td>
<td>63.61</td>
<td>1.113</td>
<td>0.636</td>
<td>16 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train &amp; Test with Noise</td>
<td>99.65</td>
<td>94.61</td>
<td>0.075</td>
<td>0.255</td>
<td>9 sec</td>
</tr>
<tr>
<td>2</td>
<td>Poisson Noise</td>
<td>Train Without Noise &amp; Test with Noise</td>
<td>99.67</td>
<td>71.54</td>
<td>0.097</td>
<td>1.393</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train with Noise &amp; Test Without Noise</td>
<td>99.64</td>
<td>88.68</td>
<td>0.085</td>
<td>0.414</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train &amp; Test without Noise</td>
<td>73.90</td>
<td>74.75</td>
<td>0.864</td>
<td>0.874</td>
<td>16 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train &amp; Test with Noise</td>
<td>99.55</td>
<td>92.45</td>
<td>0.065</td>
<td>0.327</td>
<td>9 sec</td>
</tr>
<tr>
<td>3</td>
<td>Speckle Noise</td>
<td>Train Without Noise &amp; Test with Noise</td>
<td>99.39</td>
<td>87.97</td>
<td>0.093</td>
<td>0.544</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train with Noise &amp; Test Without Noise</td>
<td>99.49</td>
<td>90.85</td>
<td>0.064</td>
<td>0.370</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train &amp; Test without Noise</td>
<td>61.06</td>
<td>64.36</td>
<td>1.128</td>
<td>1.058</td>
<td>16 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train &amp; Test with Noise</td>
<td>99.61</td>
<td>87.84</td>
<td>0.057</td>
<td>0.482</td>
<td>9 sec</td>
</tr>
<tr>
<td>4</td>
<td>Salt &amp; Pepper Noise</td>
<td>Train Without Noise &amp; Test with Noise</td>
<td>97.37</td>
<td>34.42</td>
<td>0.329</td>
<td>4.639</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train with Noise &amp; Test Without Noise</td>
<td>98.21</td>
<td>36.30</td>
<td>0.263</td>
<td>6.059</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Train &amp; Test without Noise</td>
<td>62.65</td>
<td>64.36</td>
<td>1.128</td>
<td>1.058</td>
<td>16 sec</td>
</tr>
</tbody>
</table>

Figures 4(a) and (b) represent FERCNN model accuracy and loss comparisons under various cases such as train and test with Salt & Pepper noise, train Salt & Pepper noise and test without noise, train without noise and test with Salt & Pepper noise and train and test without noises and observed that optimal accuracy and losses were obtained when the model is trained and tested with Salt & Pepper noise when compared to remaining cases. The details of the obtained results are given in Table 2.

Figure 4(c) represents the resultant values of the confusion matrix and performance metrics of the proposed FERCNN model when the FER2013 dataset is injected with Salt& Pepper noise and given for evaluation. The amount of noise added to the images is 5% and the proposed FERCNN model resulted in weighted average accuracy of 0.88, precision of 0.88, recall of 0.88 and F1-score of 0.87 after experimentation with the noisy images, Figure 4(d) represents the respective heat map of the confusion matrix.

Figures 5(a) and (b) represent FERCNN model accuracy and loss comparisons under various cases such train and test.
with Speckle noise, train with Speckle noise and test without noise, train without noise and test with Speckle noise and train and test without noise and observed that accuracy has been increased and loss has been decreased when noise is applied to the data and the details of the results are given in Table 2.
Figure 7. Comparison of FERCNN performance with various models on FER2013 dataset injected with gaussian noise

Figure 8. Comparison of FERCNN performance with various models on FER2013 dataset injected with salt & pepper noise

Figure 9. Comparison of FERCNN performance with various models on FER2013 dataset injected with speckle noise

Figure 5(c) represents the resultant values of the confusion matrix and performance metrics of the proposed FERCNN model when the FER2013 dataset is injected with Speckle noise and given for evaluation. Standard deviation, mean and variance parameter values of 20, 05 and 02 are as input to the Speckle noise function to produce Speckle noisy images. The proposed FERCNN Model resulted in weighted average accuracy of 0.92, precision of 0.93, recall of 0.92 and F1-score of 0.92 after experimentation where as Figure 5(d) represents the respective heat map of the confusion matrix.

Figures 6(a) and (b) represent FERCNN model accuracy and Loss comparisons under various cases such train and test with Poisson noise, train with Poisson noise and test without noise, train without noise and test with Poisson noise and train and test without noise and observed that accuracy has been increased and loss has been decreased when Poisson noise is applied to the data and the details of the results are given in Table 2.
Figure 10. Comparison of FERCNN performance with various models on FER2013 dataset injected with Poisson noise

(a) Accuracy comparison

(b) Loss comparison

Figure 11. Confusion matrix and performance matrix of FERCNN Models on JAFFE dataset under various noise conditions

(a) Without noise

(b) With gaussian noise

(c) With salt & Pepper noise

(d) With Poisson noise

(e) With speckle noise
Figure 6(c) represents the resultant values of the confusion matrix and performance metrics of the proposed FERCNN model when the FER2013 dataset is injected with some random Poisson noise and given for evaluation. The proposed FERCNN Model resulted in weighted average accuracy of 0.95, precision of 0.95, recall of 0.95 and F1-score of 0.9 after experimentation where as Figure 6(d) represents the respective heat map of the confusion matrix.

Figure 7(a) represents the Accuracy comparison of the proposed FERCNN model over VGG16, VGG19, Xception and Resnet 50 models in which the FER 2013 dataset is injected with Gaussian noise and given as input to the models for evaluation. Standard deviation and variance of 10 and 8 are used to produce Gaussian noise. It has been observed that the proposed FERCNN model out performed the existing models. Where as in Figure 7(b) represents the Loss comparisons of various models over Proposed FERCNN Model in the same scenario, it is observed that the proposed model loss is minimum when compared to other models. The detailed description of the results is given in Table 3.

Figures 8(a) and (b) represents the accuracy and Loss comparisons of the proposed FERCNN model over VGG16, VGG19, Xception and Resnet50 models in which the FER 2013 dataset is injected with Salt & Pepper noise and given as input to the models for evaluation, 5% of Salt & Pepper noise is injected to the images. It has been observed that the proposed FERCNN models accuracy and loss are better than existing models but not promising. The detailed description of the results are given in Table 3.

Figure 9(a) represents the Accuracy comparison of the proposed FERCNN model over VGG16, VGG19, Xception and Resnet50 models in which the FER 2013 dataset is injected with Speckle noise and given as input to the models for evaluation. Standard deviation, mean and variance of 10,5 &2 are used to produce Speckle noise. It has been observed that the proposed FERCNN model out performed the existing models. Where as in Figure 9(b) represents the Loss comparisons of various models over Proposed FERCNN Model in the same scenario, it is observed that the proposed model loss is minimum when compared to other models. The detailed description of the results are given in Table 3.

Figure 10(a) represents the Accuracy comparison of the proposed FERCNN model over VGG16, VGG19, Xception and Resnet50 models in which the FER 2013 dataset is injected with some random amount of Poisson noise and given as input to the models for evaluation. It has been observed that the proposed FERCNN model out performed the existing models. Where as in Figure 10(b) represents the Loss comparisons of various models over the Proposed FERCNN Model in the same scenario, it is observed that the proposed model loss is minimum when compared to other models. The detailed description of the results is given in Table 3.

Figure 11 shows the confusion matrix and performance metrics of Proposed FERCNN Model when different kinds of noises are injected on JAFFE dataset and given for evaluation. Figure 11(a) is the case where noise was not injected and observed weighted average accuracy, precision, recall and f1-score values of 0.95, 0.95, 0.95, 0.95 respectively. By injecting Gaussian, Poisson and Speckle noises on JAFFE data sets whose results are shown in Figure 11(b), Figure 11(d), Figure 11(e) resulted in a weighted average accuracy, precision, recall and f1-score values of 1.0, 1.0, 1.0, 1.0 which says that the classification accuracy is 100% where as by injecting Salt & Pepper noise whose results are given in Figure 11(c) resulted in a weighted average accuracy, precision, recall and f1-score values of 0.96, 0.97, 0.96, 0.96 achieving a classification accuracy of 96%.

Figure 12 represents the accuracy and Loss comparisons of the proposed FERCNN model over VGG16, VGG19, Xception and Resnet50 models in which the JAFFE dataset is injected with Gaussian noise and given as input to the models for evaluation, Standard deviation and variance of 10 and 8 are used to produce Gaussian noise. It has been observed that all the models except VGG19 produced similar results, where as VGG19 model performance is less when compared to others. The detailed description of the results is given in Table 4.

Figure 13 represents the accuracy and Loss comparisons of the proposed FERCNN model over VGG16, VGG19, Xception and Resnet50 models in which the JAFFE dataset is injected with 5% of Salt & Pepper noise and given as input to the models for evaluation. The performance of all the models is almost similar except for VGG19 whose performance is less when compared to others. The proposed FERCNN model the detailed description of the results is given in Table 4.

Table 3. Performance comparison of various models over proposed FERCNN model on FER2013 dataset

<table>
<thead>
<tr>
<th>S. No</th>
<th>Type of Noise</th>
<th>Name of The Model</th>
<th>Train Accuracy</th>
<th>Test Accuracy</th>
<th>Train Loss</th>
<th>Test Loss</th>
<th>Epoch Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gaussian Noise</td>
<td>Proposed FERCNN Model</td>
<td>99.67</td>
<td>92.17</td>
<td>0.060</td>
<td>0.354</td>
<td>10 Sec</td>
</tr>
<tr>
<td></td>
<td>VGG16</td>
<td>VGG16</td>
<td>41.11</td>
<td>41.41</td>
<td>1.449</td>
<td>1.456</td>
<td>24 sec</td>
</tr>
<tr>
<td></td>
<td>VGG19</td>
<td>VGG19</td>
<td>54.02</td>
<td>47.05</td>
<td>1.229</td>
<td>1.341</td>
<td>24 sec</td>
</tr>
<tr>
<td></td>
<td>Xception</td>
<td>Xception</td>
<td>31.38</td>
<td>28.84</td>
<td>1.880</td>
<td>1.971</td>
<td>25 sec</td>
</tr>
<tr>
<td></td>
<td>Resnet 50</td>
<td>Resnet 50</td>
<td>58.03</td>
<td>45.97</td>
<td>1.107</td>
<td>1.463</td>
<td>25 sec</td>
</tr>
<tr>
<td>2</td>
<td>Salt &amp; Pepper Noise</td>
<td>Proposed FERCNN Model</td>
<td>99.61</td>
<td>86.94</td>
<td>0.057</td>
<td>0.528</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td>VGG16</td>
<td>VGG16</td>
<td>37.53</td>
<td>17.34</td>
<td>1.558</td>
<td>2.054</td>
<td>32 sec</td>
</tr>
<tr>
<td></td>
<td>VGG19</td>
<td>VGG19</td>
<td>34.92</td>
<td>10.68</td>
<td>1.668</td>
<td>2.382</td>
<td>19 sec</td>
</tr>
<tr>
<td></td>
<td>Xception</td>
<td>Xception</td>
<td>32.10</td>
<td>10.51</td>
<td>1.682</td>
<td>2.402</td>
<td>34 sec</td>
</tr>
<tr>
<td></td>
<td>Resnet 50</td>
<td>Resnet 50</td>
<td>43.41</td>
<td>7.56</td>
<td>1.458</td>
<td>3.049</td>
<td>20 sec</td>
</tr>
<tr>
<td>3</td>
<td>Poisson Noise</td>
<td>Proposed FERCNN Model</td>
<td>95.98</td>
<td>95.07</td>
<td>0.068</td>
<td>0.227</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td>VGG16</td>
<td>VGG16</td>
<td>47.21</td>
<td>47.18</td>
<td>1.238</td>
<td>1.198</td>
<td>20 sec</td>
</tr>
<tr>
<td></td>
<td>VGG19</td>
<td>VGG19</td>
<td>55.19</td>
<td>56.93</td>
<td>1.213</td>
<td>1.132</td>
<td>22 sec</td>
</tr>
<tr>
<td></td>
<td>Xception</td>
<td>Xception</td>
<td>41.45</td>
<td>45.70</td>
<td>1.483</td>
<td>1.421</td>
<td>23 sec</td>
</tr>
<tr>
<td></td>
<td>Resnet 50</td>
<td>Resnet 50</td>
<td>61.27</td>
<td>63.97</td>
<td>1.029</td>
<td>0.639</td>
<td>24 sec</td>
</tr>
<tr>
<td>4</td>
<td>Speckle Noise</td>
<td>Proposed FERCNN Model</td>
<td>99.54</td>
<td>92.41</td>
<td>0.060</td>
<td>0.343</td>
<td>9 sec</td>
</tr>
<tr>
<td></td>
<td>VGG16</td>
<td>VGG16</td>
<td>52.91</td>
<td>51.11</td>
<td>1.201</td>
<td>1.213</td>
<td>36 sec</td>
</tr>
<tr>
<td></td>
<td>VGG19</td>
<td>VGG19</td>
<td>45.52</td>
<td>48.98</td>
<td>1.349</td>
<td>1.332</td>
<td>22 sec</td>
</tr>
<tr>
<td></td>
<td>Xception</td>
<td>Xception</td>
<td>49.18</td>
<td>48.54</td>
<td>1.319</td>
<td>1.436</td>
<td>38 sec</td>
</tr>
<tr>
<td></td>
<td>Resnet 50</td>
<td>Resnet 50</td>
<td>55.09</td>
<td>54.52</td>
<td>1.187</td>
<td>1.221</td>
<td>23 sec</td>
</tr>
</tbody>
</table>
Figure 12. Comparison of FERCNN performance with various models on JAFFE dataset injected with gaussian noise.
Figure 13. Comparison of FERCNN performance with various models on JAFFE dataset injected with salt & pepper noise

Figure 14. Comparison of FERCNN performance with various models on JAFFE dataset injected with Poisson noise
Figure 14 represents the accuracy and Loss comparison of the proposed FERCNN model over VGG16, VGG19, Xception and Resnet50 models in which the JAFFE dataset is injected with some random amount of Poisson noise and given as input to the models for evaluation. The performance of all the models are almost similar except VGG19 whose performance is not as good as other models. The details of the results are given in Table 4.

Figure 15 represents the accuracy and Loss comparisons of the proposed FERCNN model over VGG16, VGG19, Xception and Resnet50 models in which the JAFFE dataset is injected with Speckle noise and given as input to the models for evaluation. Standard deviation, mean and variance of 10, 5 & 2 are used to produce Speckle noise. It has been observed that the performance of all the models is almost similar with slight variations.

Table 4. Performance comparison of various models over proposed FERCNN model on JAFFE dataset

<table>
<thead>
<tr>
<th>S.No</th>
<th>Type of Noise</th>
<th>Name of The Model</th>
<th>Train Accuracy</th>
<th>Test Accuracy</th>
<th>Train Loss</th>
<th>Test Loss</th>
<th>Epoch Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gaussian Noise</td>
<td>Proposed FERCNN Model</td>
<td>99.80</td>
<td>100</td>
<td>0.021</td>
<td>0.004</td>
<td>1 sec</td>
</tr>
<tr>
<td></td>
<td>VGG 16</td>
<td>99.80</td>
<td>100</td>
<td>0.021</td>
<td>0.004</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VGG 19</td>
<td>86.13</td>
<td>76.37</td>
<td>0.465</td>
<td>0.664</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Xception</td>
<td>99.90</td>
<td>100</td>
<td>0.013</td>
<td>0.005</td>
<td>11 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resnet 50</td>
<td>99.78</td>
<td>100</td>
<td>0.032</td>
<td>0.004</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Proposed FERCNN Model</td>
<td>99.85</td>
<td>100</td>
<td>1.456</td>
<td>1.429</td>
<td>1 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VGG 16</td>
<td>59.65</td>
<td>70.31</td>
<td>1.048</td>
<td>0.863</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VGG 19</td>
<td>82.64</td>
<td>59.39</td>
<td>0.560</td>
<td>1.069</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Xception</td>
<td>99.70</td>
<td>99.80</td>
<td>0.021</td>
<td>0.011</td>
<td>11 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resnet 50</td>
<td>99.68</td>
<td>99.22</td>
<td>0.049</td>
<td>0.449</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Salt &amp; Pepper Noise</td>
<td>Proposed FERCNN Model</td>
<td>99.81</td>
<td>100</td>
<td>1.226</td>
<td>1.199</td>
<td>1 sec</td>
</tr>
<tr>
<td></td>
<td>VGG 16</td>
<td>99.70</td>
<td>100</td>
<td>0.018</td>
<td>0.012</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VGG 19</td>
<td>80.63</td>
<td>82.42</td>
<td>0.572</td>
<td>0.563</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Xception</td>
<td>99.92</td>
<td>100</td>
<td>0.010</td>
<td>0.003</td>
<td>11 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resnet 50</td>
<td>99.65</td>
<td>99.80</td>
<td>0.044</td>
<td>0.011</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Poisson Noise</td>
<td>Proposed FERCNN Model</td>
<td>99.89</td>
<td>99.61</td>
<td>1.387</td>
<td>1.414</td>
<td>1 sec</td>
</tr>
<tr>
<td></td>
<td>VGG 16</td>
<td>99.86</td>
<td>99.61</td>
<td>0.025</td>
<td>0.044</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VGG 19</td>
<td>86.58</td>
<td>95.12</td>
<td>0.464</td>
<td>0.356</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Xception</td>
<td>99.81</td>
<td>99.61</td>
<td>0.019</td>
<td>0.037</td>
<td>11 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resnet 50</td>
<td>99.83</td>
<td>99.61</td>
<td>0.032</td>
<td>0.027</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Speckle Noise</td>
<td>Proposed FERCNN Model</td>
<td>99.89</td>
<td>99.61</td>
<td>1.387</td>
<td>1.414</td>
<td>1 sec</td>
</tr>
<tr>
<td></td>
<td>VGG 16</td>
<td>99.86</td>
<td>99.61</td>
<td>0.025</td>
<td>0.044</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VGG 19</td>
<td>86.58</td>
<td>95.12</td>
<td>0.464</td>
<td>0.356</td>
<td>2 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Xception</td>
<td>99.81</td>
<td>99.61</td>
<td>0.019</td>
<td>0.037</td>
<td>11 sec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resnet 50</td>
<td>99.83</td>
<td>99.61</td>
<td>0.032</td>
<td>0.027</td>
<td>2 sec</td>
<td></td>
</tr>
</tbody>
</table>

Figure 15. Accuracy comparison of FERCNN with various models when JAFFE is injected with speckle noise

An observation from Figure 12 to Figure 15 and Table 4 was that the time taken for each epoch by proposed FERCNN Model is 1Sec, VGG16, VGG19 and Resnet50 models is 2Sec and Xception model is 11Sec which says that the time taken for execution is less for the proposed model when compared to other models.

5. CONCLUSION

The experiments conducted on the stated hypothesis are justifying the performance in place of noisy input. In the proposed experimental setup, the training and validation of the model with noisy images stood with high accuracy than non-noisy images. It has been noticed that training and testing accuracies with Gaussian, Poisson, Speckle, Salt & pepper
varies in between 99.55 to 99.70 and 87.84 to 94.61 whereas without injecting noise varies in between 61.06 to 73.9 and 63.61 to 74.75. The differences of classifying emotions with respect to noisy and non-noisy varies between 25.75 to 38.49 for training and 19.86 to 28.34 for testing. This proves the hypothesis that model buildup with noise is more efficient than without injecting noise. It is also witnessed that the epoch time which is needed to tune the parameters of the network is lower in case of injected input noise to non-noise images. The epoch time with noise took 9 seconds whereas in the latter case it is 16 seconds on an average, this indicates injected noise images model training is 1.77 times faster than the others. It has been noticed that the performance achieved by the FERCNN model is higher than the predefined popular CNN architectures like VGG16, VGG19, Xception, Resnet50 subject to condition if the dataset contains a large number of images and marginally equivalent if the dataset contains a small number of images. The epoch time obtained is 9 to 10 Seconds for FERCNN model whereas it is 22.25 to 29.75 Seconds for other discussed CNN models. It seems the proposed FERCNN model is not only a better choice in terms of accuracy but also model training and evaluation time. Facial emotion classification is important in a variety of applications, including education, customer satisfaction, detecting driver fatigue, and many more. In some circumstances, the input photos may not be clear enough to make necessary conclusions. Because our model was built for categorizing photos with noise, it will be effective in delivering superior results when compared to others who utilize this application. The method employed in the current study can be used to model parameters like weights, and comparison analysis can be performed, which will be a part of the future work. In the current work, noise was only introduced at the input level of the images.

REFERENCES


