To achieve our goal, we will use one of the famous machine learning algorithms out there which is used for Image Classification i.e. Convolutional Neural Network(or CNN).

For the dataset we will use the kaggle dataset of lung images.

- train dataset
- test dataset

Now after getting the data set, we need to preprocess the data a bit and provide labels to each of the image given there during training the data set. To do so we can see that name of each image of training data set is either start with "Test" or "Train" so we will use that to our advantage then we use one hot encoder for machine to understand the labels.

Libraries Required :

- **TFLearn** – Deep learning library featuring a higher-level API for TensorFlow used to create layers of our CNN
- **tqdm** – Instantly make your loops show a smart progress meter, just for simple designing sake
- **numpy** – To process the image matrices
- **open-cv** – To process the image like converting them to grayscale and etc.
- **os** – To access the file system to read the image from the train and test directory from our machines
- **random** – To shuffle the data to overcome the biasing
- **matplotlib** – To display the result of our predictive outcome.
- **tensorflow** – Just to use the tensorboard to compare the loss and adam curve our result data or obtained log.
- **Seaborn–** statistical data visualization. **Seaborn** is a Python data visualization **library** based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics

TRAIN_DIR and TEST_DIR should be set according to the user convenience and play with the basic hyperparameters like epoch, learning rate, etc to improve the accuracy.

```
os.mkdir("Images/train")
os.mkdir("Images/test")
```

Algorithm:

1. **Import Libraries**

2. **Load train and test Dataset**

3. **Identify features and response variable(s) and value(s) must be numeric numpy arrays**

4. **Make directory for test and training of data**

```
os.mkdir("Images/train")
os.mkdir("Images/test")
```

5. **Use 60% of the total dataset for Training and 40% as Test Dataset and write the data in their respective directories :**

```
if(i<round(0.6*len(files))):
        cv2.imwrite("Images/train/"+str(i)+".png",x)
      else:
         cv2.imwrite("Images/test/"+str(i)+".png",x)
```

6. **Get the image and Rescale it by 255 pixels**
7. **Reshape the image till 64*64 matrices (64,64,1)**

```
x = np.reshape(x,(64,64,1)) //np here is Numpy library
```

8. **Add noise by using noise factor**

```
img+= noise_factor*np.random.normal(mu,sigma,size=img.shape)
img = np.clip(img,0.,1.)
return img
```

$$Noise\ Factor\ (F) = \frac{\left(\frac{S_{in}}{N_{in}}\right)}{\left(\frac{S_{out}}{N_{out}}\right)}$$

9. **We are using poissons noise to train our model :**

```
def poisson_noise(self,img):

img+= numpy.random.poisson(img).astype(float)
img = np.clip(img,0.,1.)
return img
```

10. **Read in each input, perform preprocessing and get labels**

```
img = self.getImage(self.path+input_path)
img = self.rescale(img)
img = img.astype(np.float)
```

11. **Return a tuple of (input, output) to feed the network second argument is p in pa per for gaussian noise**

```
batch_x = np.array( batch_input )
batch_y = np.array( batch_output )
yield batch_x, batch_y
```

(batch x : Input variables_values_training)
(batch y: Target variables_values_training)

12. **Important) split all images in folder "Images" in 60-40 percent as 60% images in "Images/train" folder and 40% in "Images/test" folder**

```
train = DataGen("Images/train/",16,(64,64))
```

```
train_gen = train.generate() # training set generator
test = DataGen("Images/test/",10,(64,64))
test_gen = test.generate() # test set generator
```

13. **Now use epoch end to train only when the value loss is less then 0.23 (if value less is more stop the training of data)**

14. **Use keras methods and Maxpool 2d for convolutional methods to create a window clipping for testing of the data set .**
    (The window used here is of (3,3)of(64,64,1) where the denoising will be done)

15. **let it run upto 10 epochs**
```
plt.plot(hist.history["loss"],label="Training Loss")
plt.plot(hist.history["val_loss"],label="Validation Loss")
```
    (If val_loss>0.23 the epoch end will be called)

16. **Test the data gen set**
17. **Upscale again all (3,3) into (64,64,1) after Denoising .**
```
z.shape

img1 = r[0][:,:,0]
img2 = r[1][:,:,0]
img3 = r[2][:,:,0]
img4 = r[3][:,:,0]
img5 = r[4][:,:,0]

den_img1 = z[0][:,:,0]*255.
den_img2 = z[1][:,:,0]*255.
den_img3 = z[2][:,:,0]*255.
den_img4 = z[3][:,:,0]*255.
den_img5 = z[4][:,:,0]*255.
```

18. **predicted result:**
```
autoencoder.predict(r)
```

19. **Output:**
```
ssim_sum += ssim(originalSet[i], noisySet[i],data_range=originalSet[
i].max() - noisySet[i].min(), multichannel=True)

return 1.0*ssim_sum/originalSet.shape[0]

get_ssim_result(r, z)
```

20. **END**