CAP 6615 - Neural Networks
Programming Assignment-1
Single-Layer Perceptron

Venkata Vynatheya Jangal, Sohith Raja Buggaveeti, Upendar Penmetcha,
Chinmai Sai Eshwar Reddy Kasi and Venkateswarlu Tanneru

University of Florida

Friday, 28th January 2022

Contents

Single Layer Perception

1 Network Parameters

1. **Data-Set:**We created a Custom Data set which is a image data set of 16*16 pixels and the images are Number images ranging from 0 to 9.
2. **Inputs:**The are a total of 10 images which are 16*16 pixels i.e., a 10 images of each 256 pixels were taken as Input.
3. **Outputs:**The returning output is confined to 10 prediction values from 0 to 9 from the custom Data set we choose and then we generate a Image Output obtained from 256 pixel nodes with values from 0 to 1.
4. **Weights:**The total no of weights for each image is their pixel count of 256 input and 256 output pixel nodes makes the total weights around 64k.
5. **Activation Function:**We used Sigmoid function as an activation Function.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

6. **Optimizer:**Adam Optimizer is used in this model which is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.It normalises the weights along the model.
7. **Loss Function:**The Mean Squared Error (MSE) or Mean Squared Deviation (MSD) is used as a Loss Function which measures the average of error Squares i.e. Squared Difference of predicted value and true value.

$$MSD = \sum_{i=1}^{D}(x_i - y_i)^2$$

8. **Function for Noise Addition:** Guassian Noise is implemented for Noise Addition in this Model which is a Random Noise Generation Technique.Guassian Noise uses the Guassian Distribution to generate random noise over the dataset.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

2 Python Code for Single Layer Perceptron

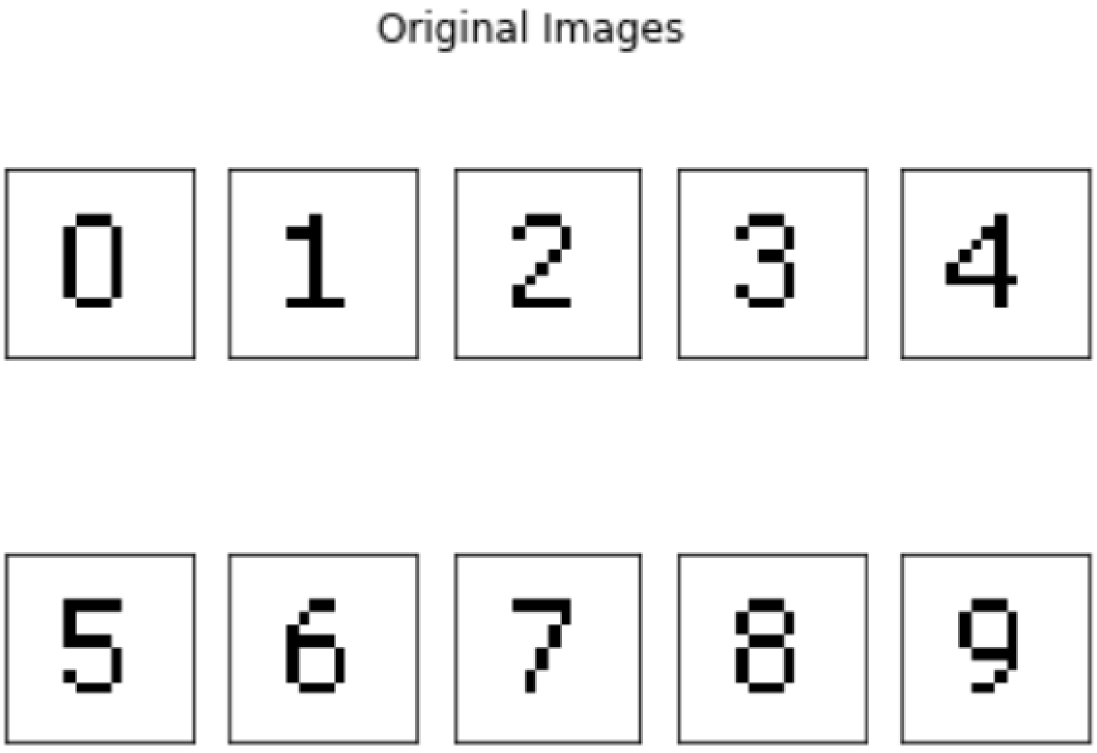
```
#------ Creating the model --------#
#------ Using Sigmoid as Activation Function --------#
#------ Configuring the model with 256 Input and Output nodes---------#
model = keras.Sequential([
keras.layers.Dense(256,activation=tf.nn.sigmoid,input_shape=(1,256)),
keras.layers.Dense(256,activation=tf.nn.sigmoid),])

#------ Using Mean Squared Error as the loss function -----------#
model.compile(optimizer=tf.keras.optimizers.Adam(),
loss='mean_squared_error')

print('Model Summary')
model.summary()
#------ Printing all the details of thge Model -----------#
print('Shapes & Bias')
for layer in model.layers:
print(layer.name)
print('Weights shape: ',layer.get_weights()[0].shape)
print('Bias Shape: ',layer.get_weights()[1].shape)

#------ Training the model --------------#
model.fit(model_train,model_train,epochs=850)
```

3 Training Set Configuration



4 SLP Output for Noiseless Data Input in Fh and Ffa Graph Representation

4.1 Noiseless data Fh and Ffa at 500 Epochs:

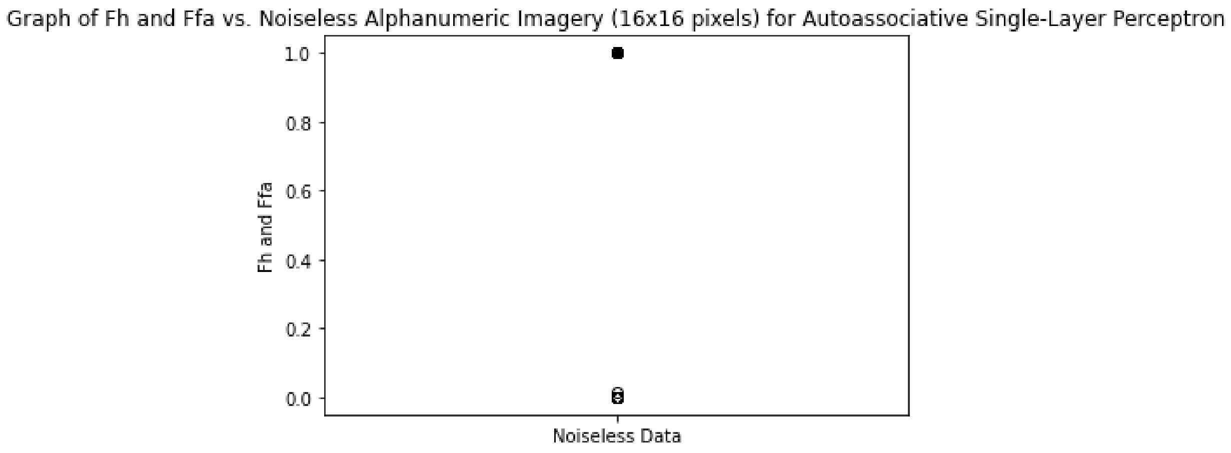


Figure 1: Fh and Ffa for 500 Epochs

4.2 Noiseless data Fh and Ffa at 1000 Epochs:

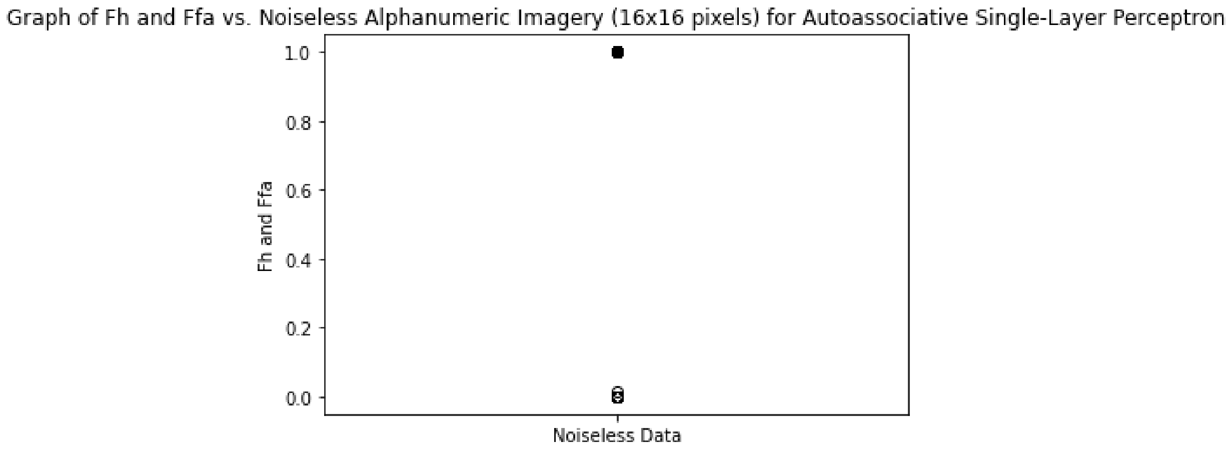


Figure 2: Fh and Ffa for 1000 Epochs

4.3 Noiseless data Fh and Ffa at 2000 Epochs:

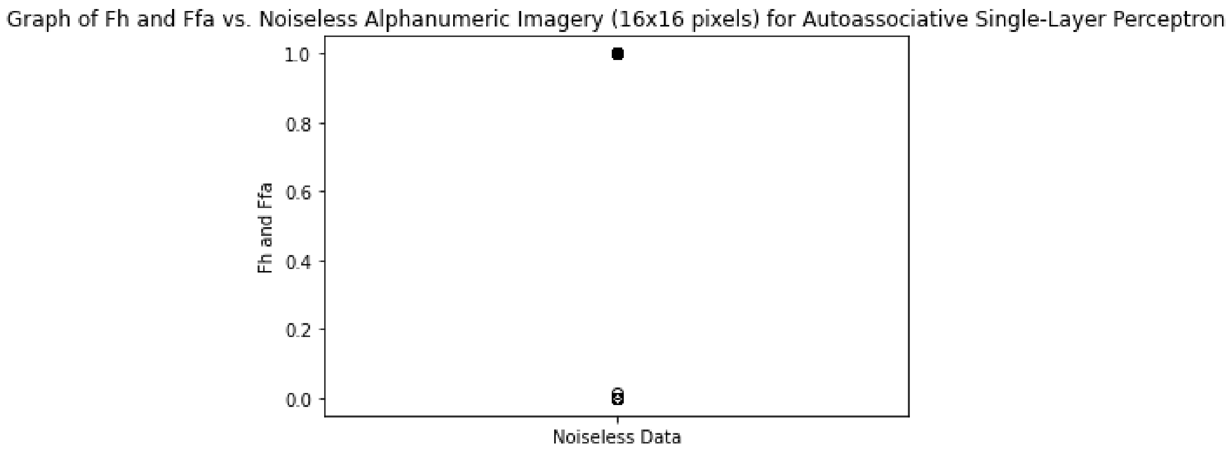


Figure 3: Fh and Ffa for 2000 Epochs

4.4 Noiseless data Fh and Ffa at 4000 Epochs:

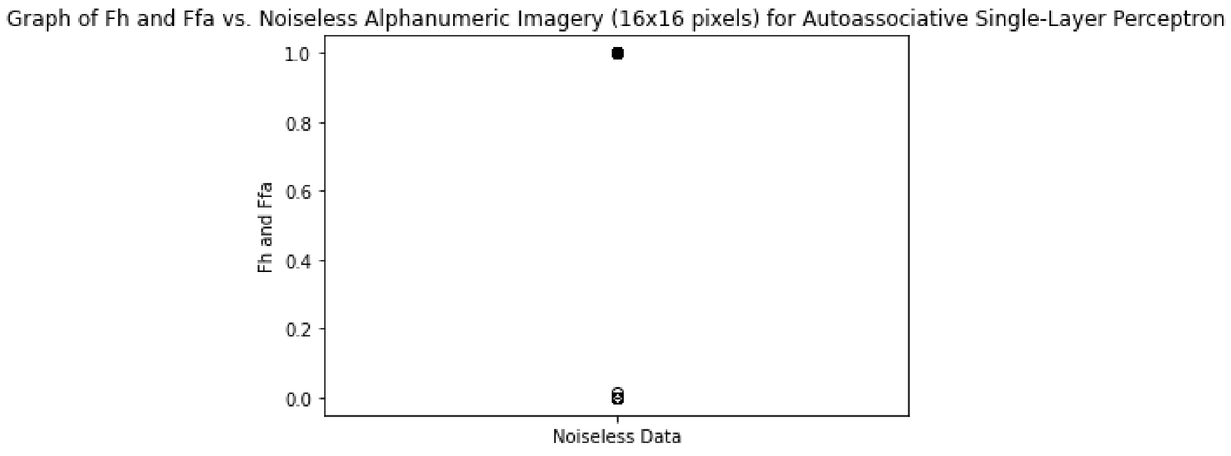


Figure 4: Fh and Ffa for 4000 Epochs

5 Fh and Ffa Calculation

5.1 Pseudo Code:

- Parse through 16*16 matrix to get the count of all correctly predicted black pixels and wrongly predicted black pixels.
- Fh can be calculated by dividing correctly predicted black pixels by original black pixels
- Ffa can be calculated by dividing wrongly predicted black pixels by original white pixels.

5.2 Python Code:

```
#------- Function for calculating FFA and FH values --------------#
def calculate_fh_ffa(original_data, predicted_data):
'''
    original_data: Original Image data
    predicted_data: Predicted Image data

'''

correctly_predicted_black_pixels_count = 0
wrongly_predicted_black_pixels_count = 0

whites_original = np.count_nonzero(original_data==1)
blacks_original = np.count_nonzero(original_data==0)

for i in range(16):
    for j in range(16):
        if original_data[i][j] == 0 and predicted_data[i][j] == 0:
            correctly_predicted_black_pixels_count += 1
        elif original_data[i][j] == 1 and predicted_data[i][j] == 0:
            wrongly_predicted_black_pixels_count += 1

fh = correctly_predicted_black_pixels_count/blacks_original
ffa = wrongly_predicted_black_pixels_count/whites_original

return fh,ffa
```

6 SLP Output Results for Noise Corrupted images

6.1 SLP Output Table for Noise Corrupted Data:

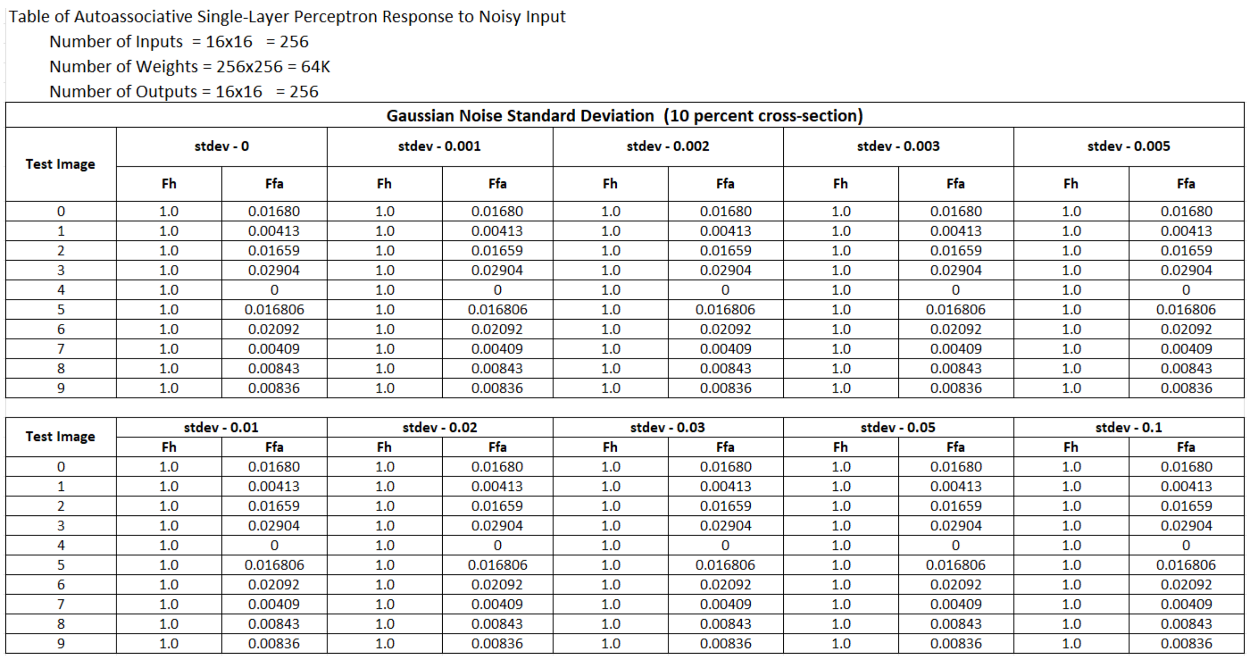


Table of Autoassociative Single-Layer Perceptron Response to Noisy Input
Number of Inputs = 16x16 = 256
Number of Weights = 256x256 = 64K
Number of Outputs = 16x16 = 256

Gaussian Noise Standard Deviation (10 percent cross-section)

Test Image	stdev - 0		stdev - 0.001		stdev - 0.002		stdev - 0.003		stdev - 0.005	
	Fh	Ffa	Fh	Ffa	Fh	Ffa	Fh	Ffa	Fh	Ffa
0	1.0	0.01680	1.0	0.01680	1.0	0.01680	1.0	0.01680	1.0	0.01680
1	1.0	0.00413	1.0	0.00413	1.0	0.00413	1.0	0.00413	1.0	0.00413
2	1.0	0.01659	1.0	0.01659	1.0	0.01659	1.0	0.01659	1.0	0.01659
3	1.0	0.02904	1.0	0.02904	1.0	0.02904	1.0	0.02904	1.0	0.02904
4	1.0	0	1.0	0	1.0	0	1.0	0	1.0	0
5	1.0	0.016806	1.0	0.016806	1.0	0.016806	1.0	0.016806	1.0	0.016806
6	1.0	0.02092	1.0	0.02092	1.0	0.02092	1.0	0.02092	1.0	0.02092
7	1.0	0.00409	1.0	0.00409	1.0	0.00409	1.0	0.00409	1.0	0.00409
8	1.0	0.00843	1.0	0.00843	1.0	0.00843	1.0	0.00843	1.0	0.00843
9	1.0	0.00836	1.0	0.00836	1.0	0.00836	1.0	0.00836	1.0	0.00836

Test Image	stdev - 0.01		stdev - 0.02		stdev - 0.03		stdev - 0.05		stdev - 0.1	
	Fh	Ffa	Fh	Ffa	Fh	Ffa	Fh	Ffa	Fh	Ffa
0	1.0	0.01680	1.0	0.01680	1.0	0.01680	1.0	0.01680	1.0	0.01680
1	1.0	0.00413	1.0	0.00413	1.0	0.00413	1.0	0.00413	1.0	0.00413
2	1.0	0.01659	1.0	0.01659	1.0	0.01659	1.0	0.01659	1.0	0.01659
3	1.0	0.02904	1.0	0.02904	1.0	0.02904	1.0	0.02904	1.0	0.02904
4	1.0	0	1.0	0	1.0	0	1.0	0	1.0	0
5	1.0	0.016806	1.0	0.016806	1.0	0.016806	1.0	0.016806	1.0	0.016806
6	1.0	0.02092	1.0	0.02092	1.0	0.02092	1.0	0.02092	1.0	0.02092
7	1.0	0.00409	1.0	0.00409	1.0	0.00409	1.0	0.00409	1.0	0.00409
8	1.0	0.00843	1.0	0.00843	1.0	0.00843	1.0	0.00843	1.0	0.00843
9	1.0	0.00836	1.0	0.00836	1.0	0.00836	1.0	0.00836	1.0	0.00836

6.2 SLP Graph Representation of Fh and Ffa for Noise Data Input :

6.2.1 Noise data Fh and Ffa at 500 Epochs:

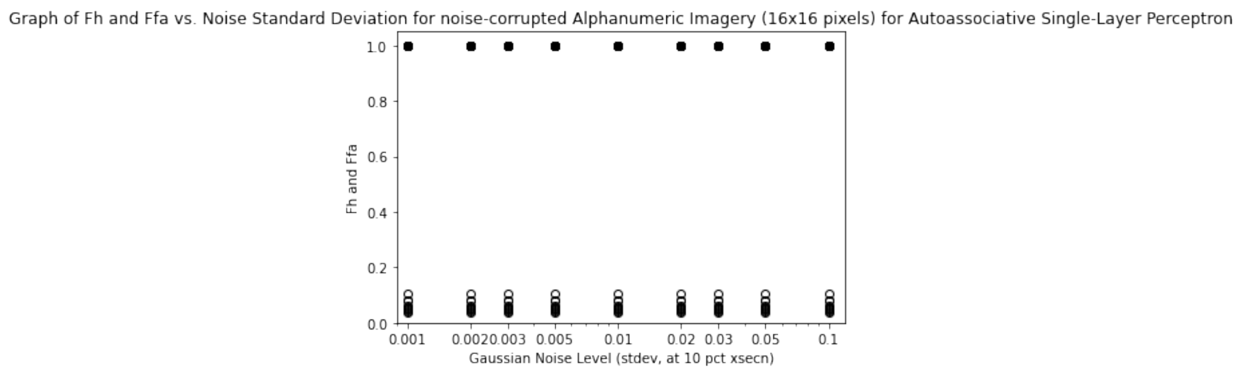


Figure 5: Fh and Ffa for 500 Epochs

6.2.2 Noise data Fh and Ffa at 1000 Epochs:

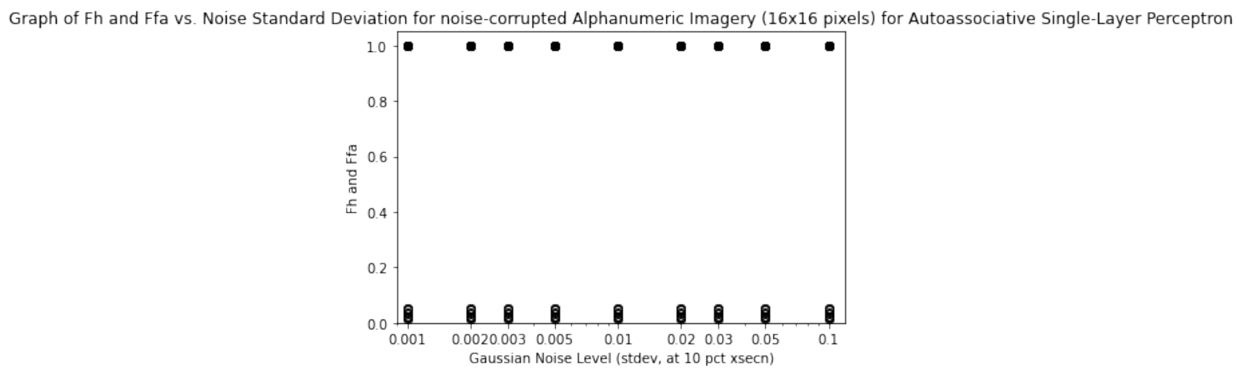


Figure 6: Fh and Ffa for 1000 Epochs

6.2.3 Noise data Fh and Ffa at 2000 Epochs:

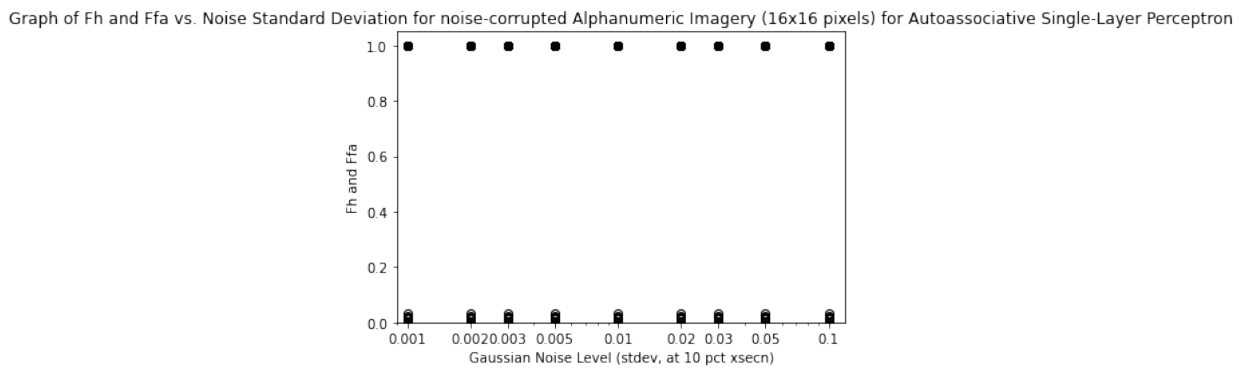


Figure 7: Fh and Ffa for 2000 Epochs

6.2.4 Noise data Fh and Ffa at 4000 Epochs:

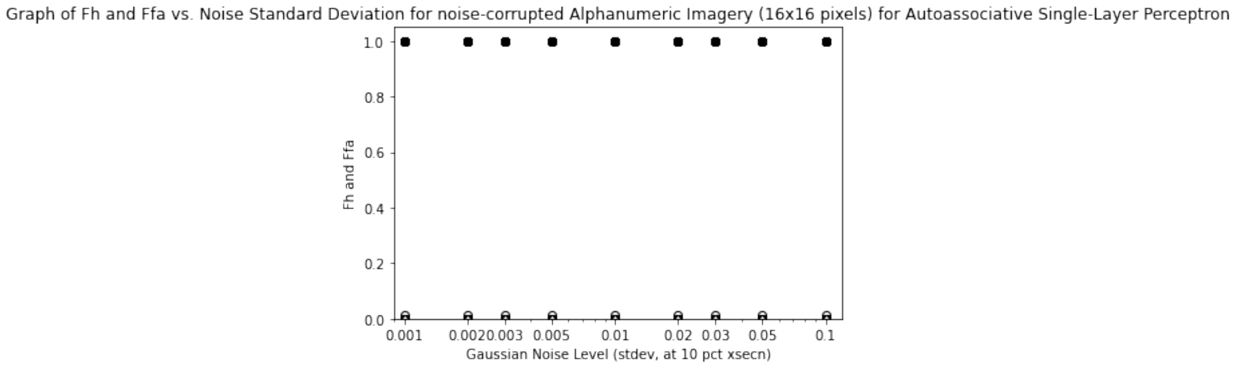


Figure 8: Fh and Ffa for 4000 Epochs

7 Single Layer Perceptron Observations

In this model, we are implementing a single layer perceptron Model. When we were implementing the model with 500 epochs, the image predictions were varying heavily. We were able to Predict the exact image at 4000 epoch.This might be due to the implementation of just a single layer.We also checked with increased back Propagation for ranging the epochs form 500 to 4000 and established the graphs of Fh(Fraction of Hits) and Ffa(Fraction of False Alarms).The Fh and Ffa clearly indicates that the images with perturbations(noise) addition can make the model even harder for prediction in a Single Layer Perceptron.

8 Appendix

8.1 Shallow Multi Layer Neural Network Parameters:

The parameters for model training were utmost same with the SLP Models **Activation Function**,**Optimizer Function**,**Loss Function** and **Gaussian Function for Noise Addition**.Even the **Inputs** and**Outputs** were same as the SLP Model but the **Weights** vary with having **64k** weights for each layer leading to **3*64K** weights totally.

8.2 Shallow Multi-layer Neural Network Python Code:

```
#----- Model Creation fo SMNN --------------#
SMM_model = keras.Sequential([
    keras.layers.Dense(256,activation=tf.nn.sigmoid,input_shape=(1,256),),
    keras.layers.Dense(256,activation=tf.nn.sigmoid),
    keras.layers.Dense(256,activation=tf.nn.sigmoid),
])
SMM_model.compile(optimizer=tf.keras.optimizers.Adam(),
    loss='mean_squared_error')
print('model summary')
SMM_model.summary()
print('Shapes & Bias')
for layer in SMM_model.layers:
    print(layer.name)
    print('Weights shape: ',layer.get_weights()[0].shape)
    print('Bias Shape: ',layer.get_weights()[1].shape)

#------- Training the Dataset --------------#
```

```
model_trains = imagetensor("C:/Users/venky/OneDrive/Desktop/NN/Nums/Train/")
model_trains=model_trains.reshape(model_trains.shape[0],1,256)
print(model_trains.shape,model_trains.dtype)

#------- Fitting the training dataset into the model --------------#
SMM_model.fit(model_trains,model_trains,epochs=950)

#------- Prediction of tested data --------------#
SMM_predict=SMM_model.predict(model_trains)

#------- Displsying Results --------------#
model_trains = np.where(model_trains<np.mean(model_trains),0,1).astype('float64')
SMM_predict = np.where(SMM_predict<np.mean(SMM_predict),0,1).astype('float64')
print(model_trains.shape,SMM_predict.shape)
display(model_trains,SMM_predict)

#------- Predicting the noisy images with the SMM model --------------#
SMM_noisy_model_prediction_data = {}
print("Predictions for Noisy data with Standard Deviation")
for std in STANDARD_DEVIATIONS:
    SMM_noisy_model = imagetensor(f"{NOISY_DATASET}{std}/")
    SMM_noisy_model = SMM_noisy_model.reshape(model_trains.shape[0],1,256)
    SMM_noisy_model = convert_array_to_0_and_1(SMM_noisy_model)

    SMM_noisy_predict = SMM_model.predict(SMM_noisy_model)
    SMM_noisy_predict = convert_array_to_0_and_1(SMM_noisy_predict)

    SMM_noisy_model_prediction_data[std] = SMM_noisy_predict
    display(SMM_noisy_model,SMM_noisy_predict,True)

#------- Calculating the FFA and FH values for all the Predicted Images
for both Trained and Noisy datasets --------------#
SMM_all_metrics = {}
SMM_metric_original = model_trains
SMM_metric_predicted = SMM_predict

SMM_noisy_model_prediction_data[0] = SMM_metric_predicted

SMM_metric_original = SMM_metric_original.reshape(SMM_metric_original.shape[0],16,16)

for k,v in SMM_noisy_model_prediction_data.items():
    SMM_metric_predicted=SMM_metric_predicted.reshape(SMM_metric_predicted.shape[0],16,16)
    SMM_all_metrics[k] = {}
    for i in range(10):
        SMM_all_metrics[k][i] = calculate_fh_ffa(SMM_metric_original[i],SMM_metric_predicted[i])

#------- Displaying all the FFA and FH for varying Deviations --------------#
print(SMM_all_metrics)

#------- Plotting all the results --------------#
plot_data(SMM_all_metrics)
```

8.3 Observations:

The Shallow Multi Layer Neural Network we implemented was given with a 3 -Layer Configuration and this SMNN Model trained was giving good results at 850 Epochs that have a good rates of accuracy and less error.The graph of this model with Fh and Ffa is plotted below.The graph clearly indicates better results at lower back-propagation.

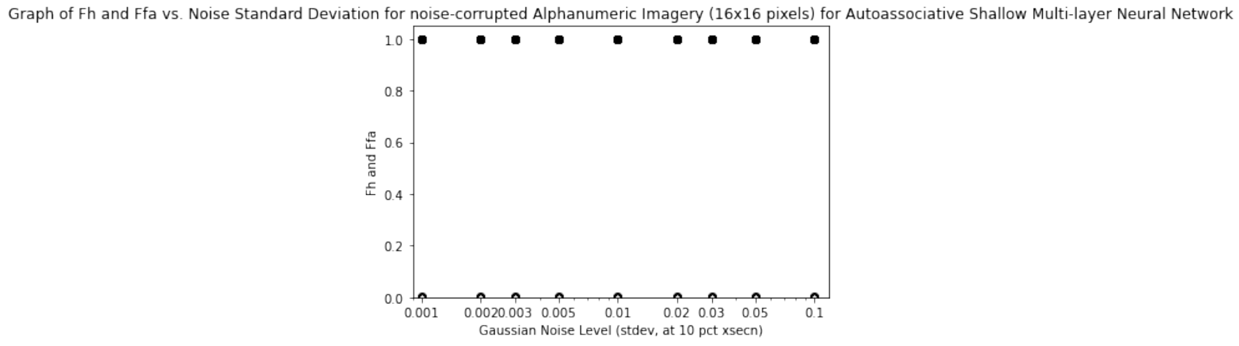


Figure 9: Fh and Ffa for SMNN