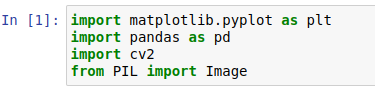
FUZZY ART MODEL:  
Fuzzy art model is a type of artificial neural network that uses the principles of fuzzy logic to perform pattern recognition and classification tasks. The basic idea behind fuzzy art model is to divide the input data into two parts: a set of crisp inputs and a set of fuzzy inputs. The crisp inputs are the well-defined variables with precise values, while the fuzzy inputs are imprecise or uncertain and have values that can be represented by membership functions. The fuzzy art model consists of two layers: the input layer and the recognition layer. The input layer receives the crisp and fuzzy inputs, and the recognition layer performs the pattern recognition and classification tasks. The recognition layer consists of a set of neurons, each of which represents a category or a class. Each neuron has a set of weights that determine its response to the input data. The fuzzy art model uses a learning algorithm called the adaptive resonance theory (ART), which allows the model to adapt to new input data and adjust the weights of the neurons accordingly. The ART algorithm ensures that the model can learn and classify new patterns without forgetting the previous ones. Overall, the fuzzy art model is a powerful tool for pattern recognition and classification tasks that involve imprecise or uncertain data. It can be used in various applications, such as image recognition, speech recognition, and data analysis.

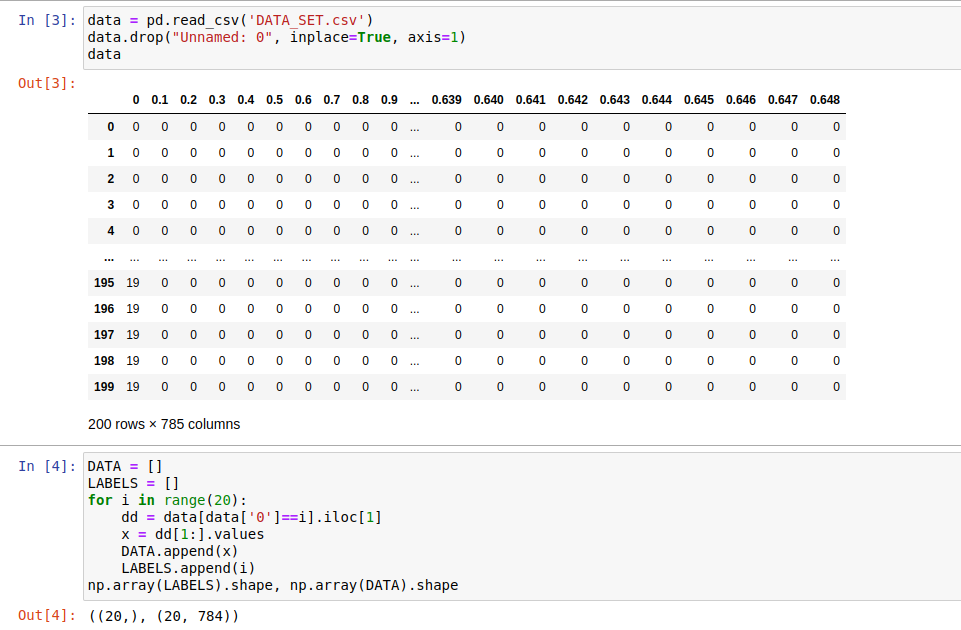
CODE EXPLAINATION:

Libraries:



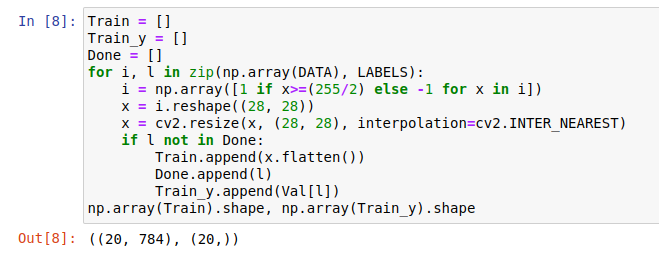
matplotlib.pyplot is used for creating data visualizations in Python. Pandas to read data and save it as data frames and series. CV2 is used for reshaping and resizing images of the DATASET for image processing, such as image filtering, object detection, and image segmentation.

Dataset:



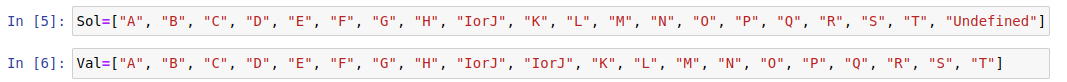
Dataset is downloaded from “Kaggle,” and it is filtered to get only one image of the alphabets from “A-T.”

Data and Lables:



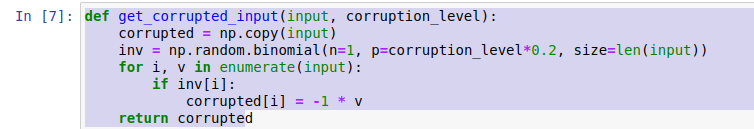
This code is preparing training data. The main loops pass over each value in DATA and LABELS. Then converts each pixel value into **1 or -1** based on a threshold of **255/2**. This is an effortless way to normalize the input values and convert them to a binary format. After that reshapes the 1D array of pixel values into a 2D array with dimensions 28x28 to make it an Image. Every single data is saved, and their Labels are also saved to evaluate the model.

VALUES:



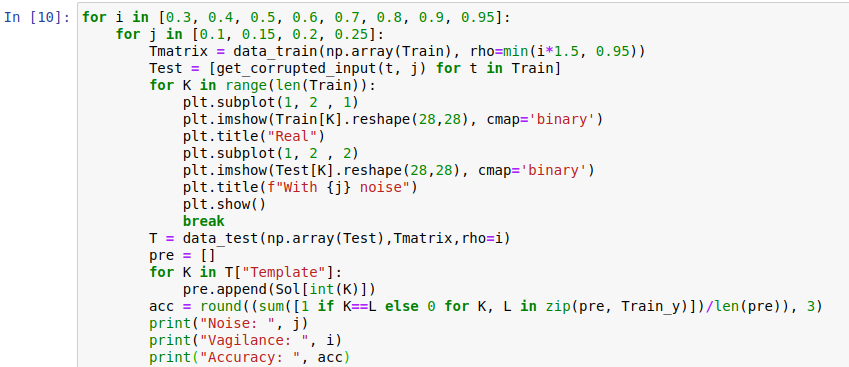
Here the values of Training and Testing are saved. The model can predict every value that was given to it but also another value “undefined” represented as “-1” so we have “undefined” as last value in the solution. The “Val” represents the values of the original Label.

Noise Function:



This is a function that takes an input array and a corruption level and returns a corrupted version of the input array. This function creates a copy of the input to ensure that the original input array is not modified and generates a binary mask of the same length as the input array. Each element in the mask has a probability of being 1, which is determined by the **corruption\_level** parameter. If the binary mask has a 1 value, then the function sets the value of the current element in the corrupted array to its negative value. This flips the sign of the element, effectively corrupting it.

MAIN FUNCTION:

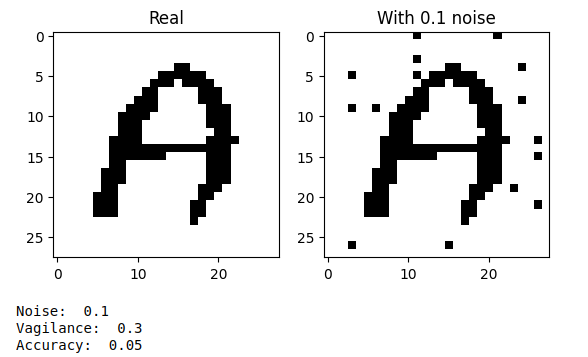


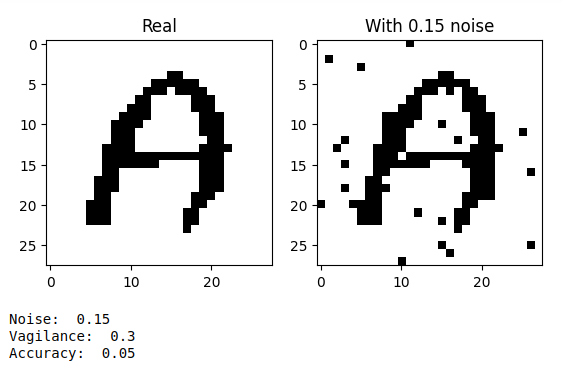
In this function we nested loops to set values of “**rho**” and **“noise”** parameter and prints the value of the accuracy and the noise level. Firstly, generate a transformation matrix by training on data with the given **rho** parameter. This matrix is used to transform the input data into a new representation that is more amenable to clustering. Then get the corrupted inputs by applying the “**get\_corrupted\_input”** function to each input in the training set with the given **noise level**. After that Display, **original** vs **corrupted** input.

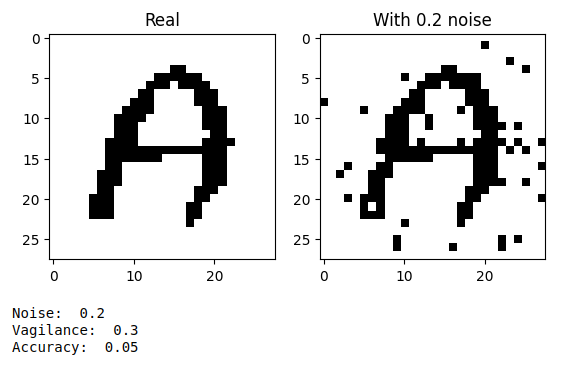
**Classify** each **noisy image then** get their value by checking the return labels by **prediction** function into **“pre” by “Sol”.**

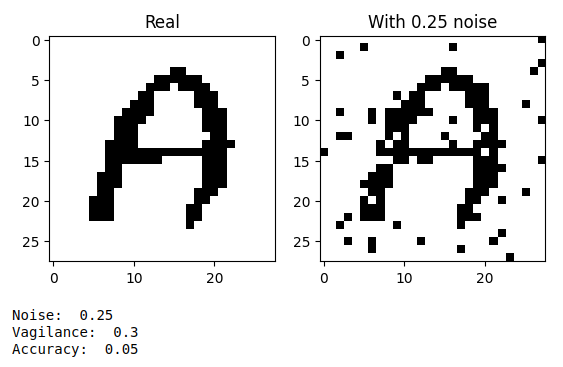
Lastly, compute the **accuracy** of the **classification** by **comparing the predicted labels to the true labels.**

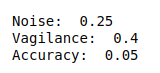
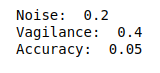
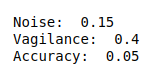
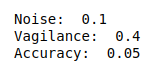
OUTPUT:

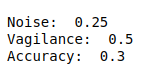
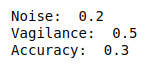
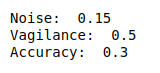
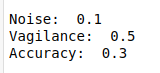


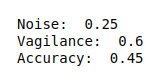
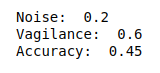
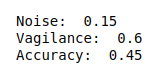
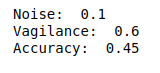


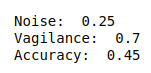
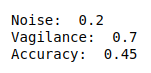
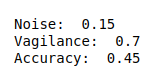
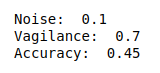


















This was the output of **main Function.**

**The Training and Testing helping code was gathered from “GitHub.”**

**References:** [**https://github.com/cbirkj/art-python.git**](https://github.com/cbirkj/art-python.git)

**This repo is like an ART network library.**