DA675 - Fuzzy Pooling

Nishchay Nilabh

***Abstract*—This paper presents an implementation of Fuzzy Pooling [1], a method that incorporates fuzziness to dynamically capture more nuanced information within neural networks. The primary objective is to evaluate the effectiveness of Fuzzy Pooling in retaining information and analyze its impact on model performance compared to traditional pooling methods, such as max pooling and average pooling. Additionally, three distinct Fuzzy Pooling models are explored, each utilizing different membership functions, to assess their comparative strengths and weaknesses. This study aims to provide insights into how the added dimension of fuzziness influences information retention and contributes to overall model efficacy.**

I. Introduction

Neural networks often use pooling layers to reduce the size of the data they are working with, which makes processing faster and easier. The most commonly used pooling methods are max-pooling and average-pooling. Max-pooling keeps only the highest value from a selected area of the data, while average-pooling calculates the average of all values in that area. Both methods condense information, but they also throw away a lot of useful details.

For example, max-pooling keeps only the largest value and ignores other significant values in the area, while average-pooling can ”flatten out” important variations by blending all values together. This can cause the network to lose important information that might help it make better predictions.

To address this, I implemented a method called fuzzy-pooling. Fuzzy-pooling takes a different approach by assigning importance or ”weights” to different parts of the data instead of treating each value as equally important. This assignment of weights is performed using membership functions. Thus, it allows the network to keep more of the critical information, giving attention to areas that hold valuable features without discarding all other details.

In fuzzy-pooling, each value in the selected area of data gets a degree of ”importance” based on how relevant it is. Rather than taking just the max value or the average, fuzzy-pooling combines values based on these assigned membership values. This keeps more useful information in the condensed data, which can lead to better overall performance of the neural network.

II. Methodology

In this work, I implemented the Fuzzy Pooling layer as described in the paper *Fuzzy Pooling* by Dimitrios E. Diamantis and Dimitris K. Iakovidis. The goal was to evaluate how effectively Fuzzy Pooling could retain valuable features compared to traditional pooling methods.

To implement a fuzzy pooling layer, the initial step is to define the fuzzy membership functions, which are typically used to evaluate the degree to which each element belongs to a particular cluster or class within the input data. Common choices for membership functions include Gaussian, triangular, or sigmoidal shapes.

These functions convert the input values into fuzzy values, which can then be manipulated using fuzzy logic operations. The choice of membership function significantly impacts the pooling process, as it defines how input values are aggregated and which regions of the input data are emphasized.

Next, aggregation methods are applied to combine the fuzzy values of multiple inputs, which summarize the information captured by the membership functions. The aggregation helps to retain essential features while reducing the dimensionality of the data. During this phase, the fuzzy pooling layer selectively highlights regions of the input, preserving important information and disregarding irrelevant details. By implementing this aggregation step, the layer achieves a pooling effect.

Finally, a defuzzification process is applied to convert the aggregated fuzzy values back into a standard numerical form, making them usable for subsequent neural network layers. Here, centre of gravity (CoG) method was used. This step transforms the fuzzy information into a concise representation, enabling effective information flow through the network. Implementing these steps provides a smooth transition from traditional pooling to fuzzy pooling, allowing for more nuanced feature extraction and data reduction.

Initially, the steps outlined in the paper were implemented using Python loops. However, this approach proved to be computationally slow, so I optimized the code by employing vectorized operations and shifting the calculations to the GPU, resulting in a significant performance improvement.

I evaluated the Fuzzy Pooling layer using a basic Convolutional Neural Network (CNN) model inspired by the LeNet architecture. This model includes a convolutional layer, a pooling layer (where I tested various pooling strategies), and a fully connected layer. The pooling layer, which was the focus of my experimentation, and I tested it with five types of pooling approaches:

- Max Pooling : (Take max value amongst all elements in the patch)
- Average Pooling : (Take average of all elements in the patch)

- Fuzzy Pooling:
  - membership functions discussed in the paper:
    - left open
    - triangle
    - right open
  - some of my experimented membership functions:
    - gaussian
    - sigmoid
    - bell-shaped
  - center-focused membership functions:
    - gaussian
    - triangle
    - bell-shaped

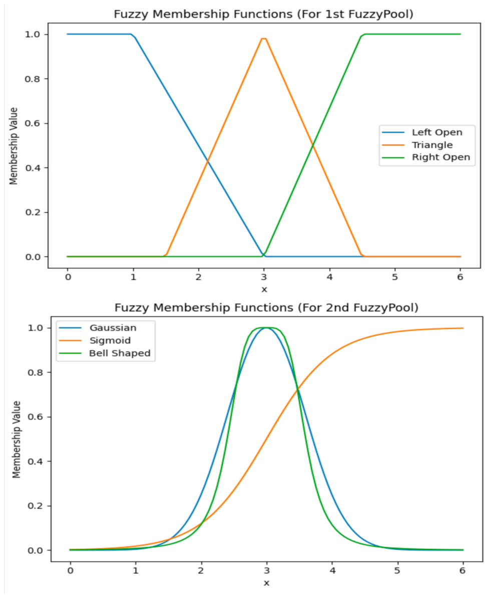

Fig. 1: Fuzzy Membership Functions Used in the Study

Initially, I experimented with zero initialization for the weights of the Fuzzy Pooling network. However, this approach resulted in extremely poor performance, yielding an accuracy of just 11%. The model consistently predicted only the value of one, indicating that it was unable to learn effectively from the training data.

Recognizing this issue, I transitioned to He initialization, which led to a significant improvement in model performance. It sets the initial weights using random numbers from a Gaussian distribution with a mean of zero and a standard deviation based on the number of input units. This helps keep the size of the activations just right as they move through the network, avoiding problems like vanishing gradients. This improvements in accuracy will be discussed in the subsequent sections of the report.

III. Results and Discussions

To compare the effectiveness of these pooling methods, I trained each configuration on the MNIST dataset, which provides a standardized benchmark for performance. The output of each experiment was then evaluated to assess the retention of essential information and the overall impact on model performance.

TABLE I: Model Performance on MNIST Dataset

Pooling Method	**Accuracy (%)**
Max Pooling	88.14
Average Pooling	87.13
Fuzzy Pooling (Left Open, Triangle, Right Open)	96.70
Fuzzy Pooling (Gaussian, Sigmoid, Bell Shaped)	97.80
Fuzzy Pooling (Gaussian, Triangle, Bell Shaped)	97.83

In my experiments on the MNIST dataset, Fuzzy Pooling showed a marked improvement in classification accuracy compared to traditional pooling methods. Max Pooling achieved 88.14% accuracy, while Average Pooling reached 87.13%. In contrast, Fuzzy Pooling models attained accuracies between 96.70% and 97.83%, indicating their effectiveness in retaining important information.

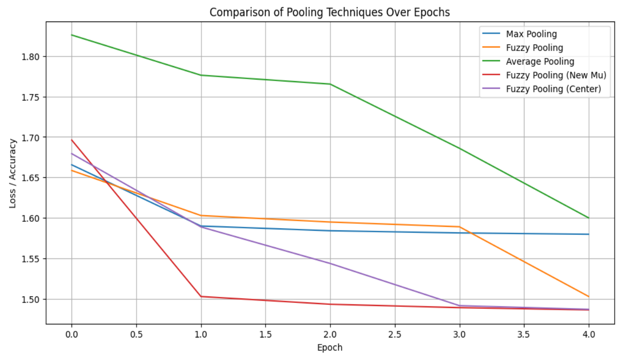

Fig. 2: Training loss plots for the five approaches (New Mu is for Gaussian, Sigmoid, and Bell-curve and Center is for Gaussian, Triangle, and Bell-curve)

Fuzzy Pooling's success comes from its ability to capture more detailed features in the data. Unlike Max Pooling, which only considers the maximum value, and Average Pooling, which can blur important details, Fuzzy Pooling keeps finer distinctions. This is crucial for recognizing handwritten digits, where small differences matter.

Additionally, I found that using membership functions that give more weight to the center of the image improves results. By focusing on central pixel values, these functions help the model learn from the most informative parts of the digits. This can be seen from Fig. 3 where the distribution of white pixels is more concentrated towards the center.

Some important details were not mentioned in the paper, such as the type of weight initialization used and the hyperparameters for the optimizer. Since this information was missing, I made my own assumptions to conduct the experiments. The initialization method can affect how well the model trains and performs, as it determines how the weights are set at the start.

Likewise, the choice of hyperparameters for the optimizer, including learning rate and momentum, can greatly influence how quickly the model learns and how effective it is. The results in this report are based on the settings I chose for these factors, which I selected from common practices in the field.

For the hyperparameters, I configured the stride to 2 and the kernel size to 3. Additionally, I applied a cap of 6 to the ReLU activation function and trained the model for a total of 5 epochs. For optimization, I used the Stochastic Gradient Descent (SGD) algorithm with a learning rate of 0.01 and a momentum of 0.9.

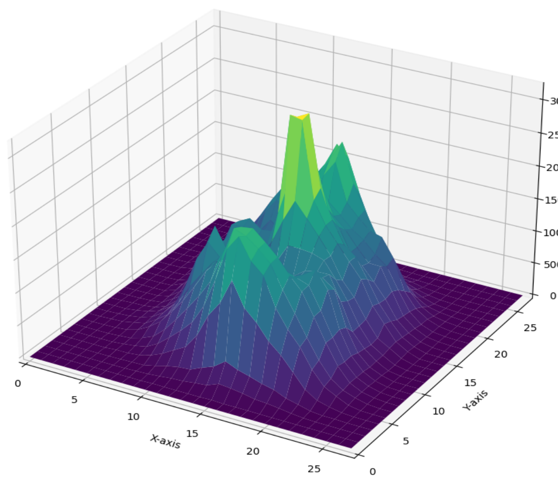

Fig. 3: Distribution of White Pixels in MNIST Dataset

IV. Conclusions and Future Scopes

In this project, I studied Fuzzy Pooling as an improvement to traditional pooling methods in convolutional neural networks. I showed that Fuzzy Pooling helps retain important features better than Max Pooling and Average Pooling. While Max Pooling achieved an accuracy of 88.14% and Average Pooling 87.13%, Fuzzy Pooling models reached up to 97.83% accuracy. Using membership functions that focus on the center of the image was key to these improvements, allowing for better extraction of features in handwritten digit recognition.

For future work, I plan to explore the following:

- **Testing on More Datasets**: I want to apply Fuzzy Pooling to different datasets, such as CIFAR-10, to see how well it works with various types of images.
- **Exploring Membership Functions**: I will investigate different membership functions to optimize performance.

References

[1] D. E. Diamantis and D. K. Iakovidis, ”Fuzzy Pooling,” in IEEE Transactions on Fuzzy Systems, vol. 29, no. 11, pp. 3481-3488, Nov. 2021, doi: 10.1109/TFUZZ.2020.3024023.

[2] L. Deng, ”The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web],” in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, Nov. 2012, doi: 10.1109/MSP.2012.2211477.