UNIVERSITI
MALAYA

WIX3001 SOFT COMPUTING

ALTERNATIVE ASSESSMENT

No.	Name	Matric Number
1	Lim Jun Yi	22004811
2	Lim Zheng Qian	U2005381
3	Lim Boon Guan	U2005273
4	Tan Teck Hou	22053703
5	Gan Zi Xiang	22004732

Table of Contents

Project Introduction

This project aims to identify the user based on the image of the handwritten character. But, identification from different characters may be less straightforward intuitively and require a lot of researches and architecture experimenting.

Our approach

We are less confident in constructing a singular neural network computer vision model that is able to identify the writing style of the user given images from different character and users. Hence, we propose a two-step model for the user identification from the input handwritten character image:

1. Identify the character using Computer Vision techniques
2. Identify the user that wrote that character

Below is the expected prediction flow diagram from input image to output classification:

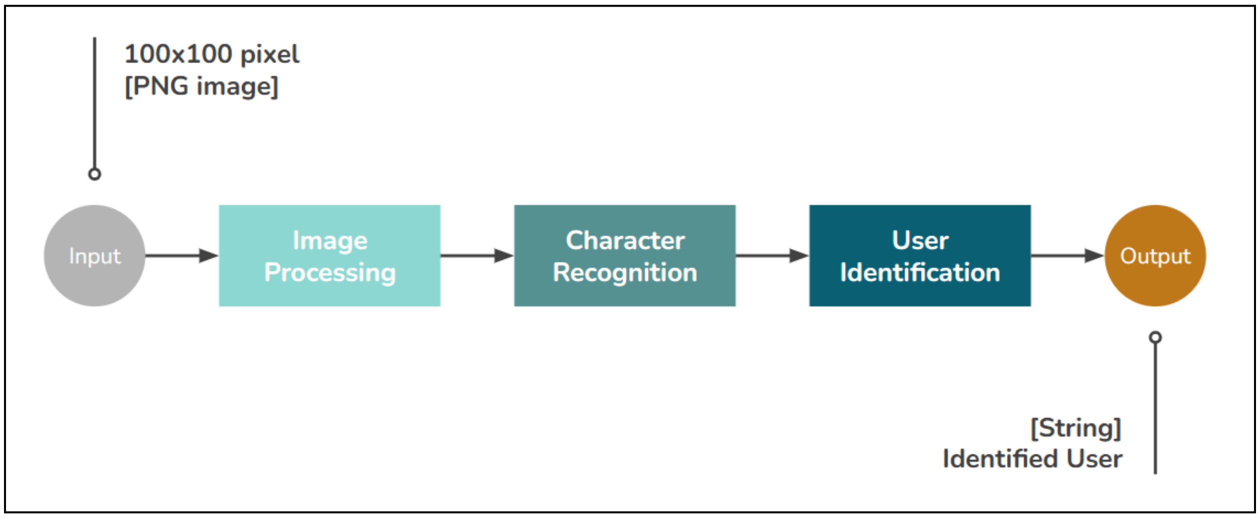


Figure 1: prediction flow diagram

In our proposed model, our input image will go through the processing stage to convert image into 2d arrays, then recognize the character of that image first, only then identify this character is written by who by comparing it with other user's image for this character.

Introduction to dataset

The dataset used for this project was collected via the han written images from the 5 team members involved in this project. A total of 720 images were collected where each member repeated 4 times for 10 digits (0 to 9) and 26 characters (A to Z). The collaborative platform used for the dataset creation was Figma which allows the ability of free-drawing to create the handwritten dataset. Each piece of data is an image with the fixed dimension for width and height of 100*100.

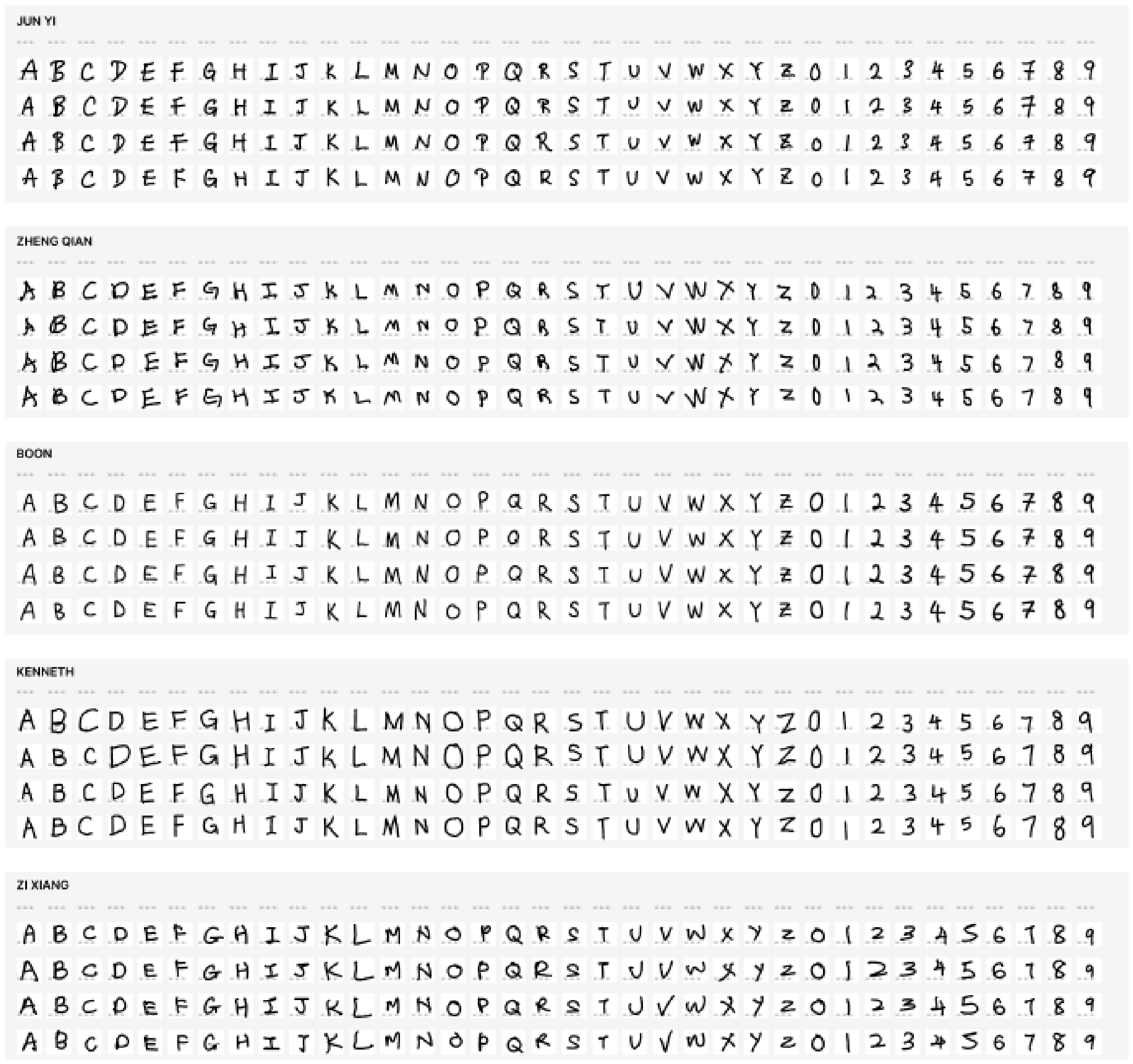


Figure 2: Handwritten Dataset from All Members

Dataset Preprocessing

Preprocessing of the dataset was conducted in the early phase of this project to ensure the usability of the data images.

```
def process_image(path):

    img = Image.open(path)

    # Resize Image
    img = img.resize(size=(28, 28), resample=Image.ANTIALIAS)

    # Convert RGBA image to 4D array
    img_pixel = np.array(img)

    # Convert to grayscale -> MNIST format
    img_pixel = 255 - (img_pixel[:, :, :3] @ [0.299, 0.587, 0.114])

    return img_pixel
```

Using the code above, the dataset images of width and height 100*100 was resized to 28*28. Then the RGBA images with 4 channels were converted into a single-channel Numpy 2D-array in order to flatten them into 1-dimension data. The end product of the preprocessing was the data with dimensions of 1*28*28 for one input image.

The following figure shows the processed image for the character ‘7’ by each author (member of the group). Note that the author label is encoded as below:

- Author 1 → Lim Jun Yi
- Author 2 → Lim Zheng Qian
- Author 3 → Tan Teck Hou / Kenneth
- Author 4 → Lim Boon Guan
- Author 5 → Gan Zi Xiang

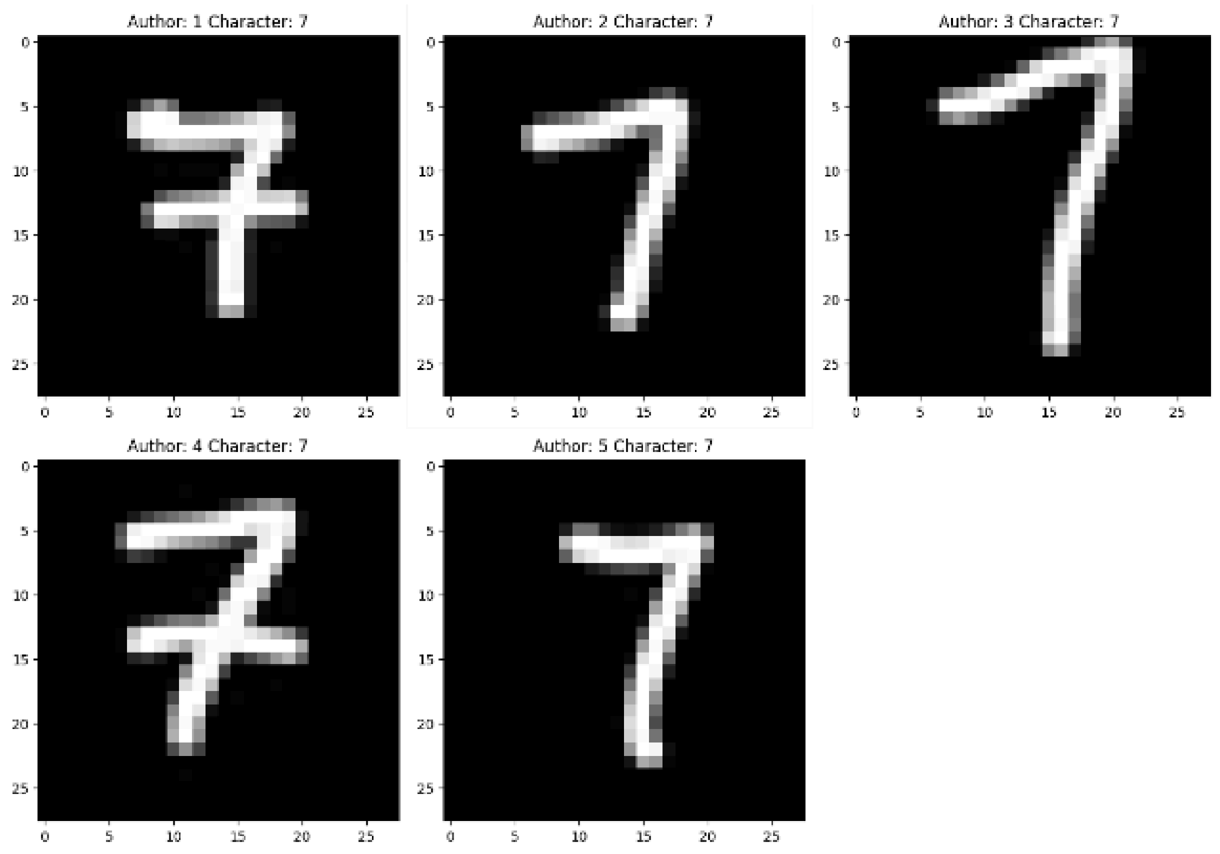


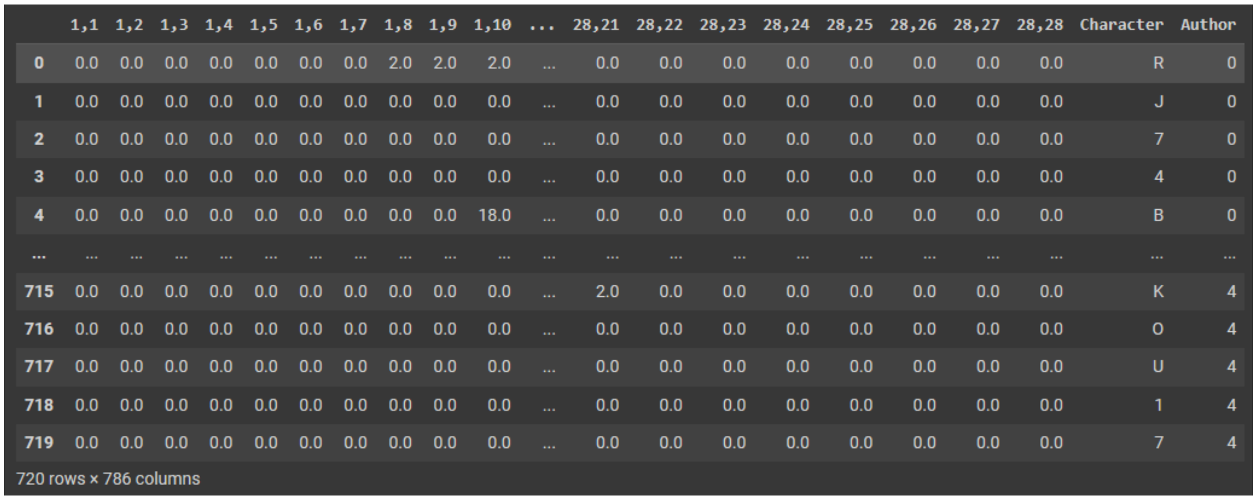
Figure 3: processed image of char ‘7’ by each author

With one image working, we expand the function to be called on all 720 images we have collected to create a dataset. The dataset that we aim to obtain contains 720 rows where each row represents one image data, consisting of the 28*28 pixels flattened and labelled according to its pixel coordinate (in terms of (row, col)), its actual character and the author.

The code snippet below displays the implementation of the concept above.

```
# Create dataset
img_path = './HandwrittingAuthorIdentifier/Dataset'

data = []

for author_id, author_path in enumerate(os.listdir(img_path)):
    image_paths = os.listdir(f'{img_path}/{author_path}')

    for image_path in image_paths:

        img_pixel = process_image(f'{img_path}/{author_path}/{image_path}')
        row = np.append(img_pixel.flatten(), image_path[0])
        row = np.append(row, author_id)
        data.append(row)

# Dataframe creation
df = pd.DataFrame(data)

# Pixel Row,Col from top left
pixel_columns = [f'{i//28+1},{i%28+1}' for i in range(784)]
df.columns = pixel_columns + ['Character', 'Author']

# Convert values to np.float32 data type
df[pixel_columns] = df[pixel_columns].astype('float32')
```

Note that after this step, the values in the pixel columns are real values within the range of [0, 255], where 0 indicates black and 255 indicates white, which is not normalized yet for visualization purposes. The image below shows the final dataframe after the processing step:



	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10	...	28,21	28,22	28,23	28,24	28,25	28,26	28,27	28,28	Character	Author
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	2.0	2.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	R	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	J	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	7	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	18.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	B	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
715	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	K	4
716	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	O	4
717	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	U	4
718	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1	4
719	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	7	4

720 rows × 786 columns

Figure 4: post-process dataframe

Soft Computing Method Used

To implement Fuzzy Logic, Neural Network, and Evolutionary Computing in our MNIST-like dataset, we separated our model into 2 steps:

1. Identify the character from the image (Character Recognition)
2. Identify the author from that character pool (User Identification)

In the first step, we used neural networks tuned using evolutionary computing methods to classify the images in the dataset into 36 classes, identifying the characters from the range of 0 - 9 and A - Z. Then, in the second step, we will be using 2 different Fuzzy Rule Bases into our Fuzzy System to identify the author for each characters.

Genetic Algorithm and Neural Network

To perform computer vision in character recognition, we will be using a Convolutional Neural Network (CNN) built from tensorflow’s Keras API.

A default basic CNN model comprises of the following layers:

- First Convolutional Block containing convolutional filters, pooling and dropout layers
- Second Convolutional Block containing convolutional filters, pooling and dropout layers
- Fully-Connected Block containing fully-connected layers, dropout layer and output layer

To make our basic CNN perform better, we will need to tune its hyperparameters, which includes how many layers the network has, how quickly it learns, and how it adjusts its internal values [1].

Hence according to the basic CNN model architecture, we will be tuning the following hyperparameters:

- First Convolutional Block’s Number of Kernels (the power of 2)
- Second Convolutional Block’s Number of Kernels (the power of 2)
- Fully-Connected Layer Number of Neurons

- First Convolutional Block’s Dropout Rate
- Second Convolutional Block’s Dropout Rate
- Fully-Connected Block’s Dropout Rate

Note that we require our convolutional block’s number of kernels to be a number that is 2^N. The reason behind this is that early neural network implementations for GPU Computing (written in CUDA, OpenCL etc) concern themselves with efficient memory management to do data parallelism, where we will need to align to the number of physical processors, which is usually a power of 2. Therefore, if the number of computations is not a power of 2, the computations can't be mapped 1:1 and have to be moved around, requiring additional memory management [4].

Before moving on, we will need to preprocess our data once again before it can be fed into the Neural Network. The following code snippet implements the preprocessing of data:

```
X = df.copy()
_ = X.pop('Author')
y = X.pop('Character')

# Reshape flatten pixels
X_reshaped = X.values.reshape(-1, 28, 28, 1)
print(X_reshaped.shape)

# Label Classes -> convert A to 10, B to 11, ...
y = y.apply(lambda x: int(x) if x.isnumeric() else ord(x) - 55)
y_cat = tf.keras.utils.to_categorical(y)
```

Fitness Evaluation Function

Firstly, we will define a CNN model inside the fitness_evaluation function. The fitness function is receiving chromosomes for each genetic algorithm generation that we will cover in the later part.

```
def fitness_evaluation(chromosome):

    conv2d_1 = chromosome[0]
    conv2d_2 = chromosome[1]
    dense_1 = chromosome[2]
    dropout_1 = chromosome[3]
    dropout_2 = chromosome[4]
    dropout_3 = chromosome[5]
```

The CNN model will be a Sequential model. Initially the pixel values of each image will be rescaled from [0,255] to [0,1].

```
model = Sequential([
    layers.Rescaling(scale=1./255),
```

In the first Convolutional Block, we applied 2 convolutional filters to the input image to extract features from the image. The number of filters is 2**conv2d_1, this will allows exponential growth of the filter size. ReLU activation methods are used in the Conv2D. MaxPool2D is applied to reduces the spatial dimensions of the image by taking the maximum value in each 3x3 pool. To prevent overfitting, we set a dropout at the end of it. The dropout value and Conv2D filters value are based on the current generation of the genetic algorithm.

```
layers.Conv2D(2**conv2d_1, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=(28, 28, 1)),
layers.Conv2D(2**conv2d_1, kernel_size=(3, 3), activation='relu', padding='same'),
layers.MaxPool2D(pool_size=(3, 3)),
layers.Dropout(rate=dropout_1),
```

In the second convolutional block, we set a similar block, but with potentially different hyperparameters based on the current generation of genetic algorithm.

```
layers.Conv2D(2**conv2d_2, kernel_size=(3, 3), activation='relu', padding='same'),
layers.Conv2D(2**conv2d_2, kernel_size=(3, 3), activation='relu', padding='same'),
layers.MaxPool2D(pool_size=(3, 3)),
layers.Dropout(rate=dropout_2),
```

A flatten layer will be applied to convert the 2D output of the convolutional layers into a 1D feature vector, which can be fed into fully connected (Dense) layers. A Dense layer with dense_1 units and ReLU activation function. This layer learns to combine the features extracted by the convolutional layers. Another dropout layer is added to prevent overfitting. The final Dense layer has 36 units (one for each class) and uses the softmax activation function to output class probabilities.

```
layers.Flatten(),
layers.Dense(dense_1, activation='relu'),
layers.Dropout(rate=dropout_3),
layers.Dense(36, activation='softmax'),
```

The model is compiled with the Adam optimizer, categorical cross-entropy loss function, and categorical accuracy as the metric, and then trained for 15 epochs. The returned fitness value is the negative of the maximum categorical accuracy achieved during training. This is done because the genetic algorithm minimizes the fitness function, so by returning the negative accuracy, it effectively maximizes accuracy.

```
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
history = model.fit(X_reshaped, y_cat, epochs=15, verbose=0)

return -max(history.history['categorical_accuracy'])
```

Evolution Computing Strategies

We perform Genetic Algorithm using the genetic algorithm [2] library obtainable from the Python Package Index (PyPI) and installable using the `pip install` command.

In each generation of the genetic algorithm, we will be fine-tuning the training hyperparameters of the Basic CNN model. The code cell below creates an array of boundaries for each gene, and the data type acceptable for that gene value.

```
# Chromosome Genes: [conv2d_1 (2^), conv2d_2 (2^), dense_1, dropout_1, dropout_2, dropout_3]
varbound = np.array([[5, 7], [5, 7], [150, 160]] + [[0.2, 0.5]] * 3)
vartype = np.array([['int']] * 3 + [['real']] * 3)
```

Then, we will initiate our genetic algorithm for best hyperparameter settings search using two strategies:

- Ranked Selection with Uniform Crossover
- Elitism Selection with Two Points Crossover

The other configurations for the evolutionary computing is as below:

- 20 Number of Generations (Each generation takes 10 - 15 minutes to evolve)
- Population Size of 10 chromosomes
- 0.2 Mutation Rate
- 0.5 Crossover Rate

We implemented the concept above in the code cells below, starting with the first strategy by setting the elit_ratio to 0 and crossover_type to uniform:

```
# Standard Selection, Uniform Crossover
strategy_1 = {
    'max_num_iteration': 20,
    'population_size': 10,
    'mutation_probability': 0.2,
    'elit_ratio': 0,
    'crossover_probability': 0.5,
    'parents_portion': 0.3,
    'crossover_type':'uniform',
    'max_iteration_without_improv': None
}

ga_1 = GeneticAlgorithm(function=fitness_evaluation, dimension=6,
                        variable_type_mixed=vartype, variable_boundaries=varbound,
                        algorithm_parameters=strategy_1,
                        function_timeout=600, convergence_curve=False)

ga_1.run()
```

To implement the second strategy, we will just change the elit_ratio to 0.01 and crossover_type to twopoint:

```
# Elitism Selection, Two Point Crossover
strategy_2 = {
    'max_num_iteration': 20,
    'population_size': 10,
    'mutation_probability': 0.2,
    'elit_ratio': 0.01,
    'crossover_probability': 0.5,
    'parents_portion': 0.3,
    'crossover_type':'two_point',
    'max_iteration_without_improv': None
}

ga_2 = GeneticAlgorithm(function=fitness_evaluation, dimension=6,
                        variable_type_mixed=vartype, variable_boundaries=varbound,
                        algorithm_parameters=strategy_2,
                        function_timeout=600, convergence_curve=False)

ga_2.run()
```

Neural Network

Basic CNN

The figure below shows the structure of our basic CNN:

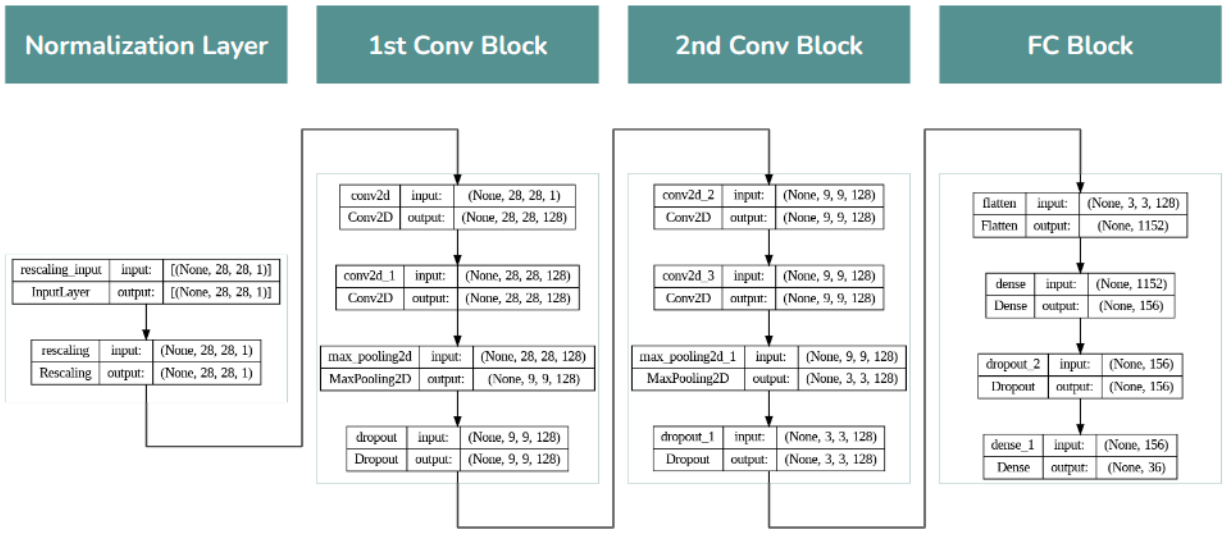

Normalization Layer
1st Conv Block
2nd Conv Block
FC Block
rescaling_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
rescaling | input: | (None, 28, 28, 1)
Rescaling | output: | (None, 28, 28, 1)
conv2d | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 28, 28, 128)
conv2d_1 | input: | (None, 28, 28, 128)
Conv2D | output: | (None, 28, 28, 128)
max_pooling2d | input: | (None, 28, 28, 128)
MaxPooling2D | output: | (None, 9, 9, 128)
dropout | input: | (None, 9, 9, 128)
Dropout | output: | (None, 9, 9, 128)
conv2d_2 | input: | (None, 9, 9, 128)
Conv2D | output: | (None, 9, 9, 128)
conv2d_3 | input: | (None, 9, 9, 128)
Conv2D | output: | (None, 9, 9, 128)
max_pooling2d_1 | input: | (None, 9, 9, 128)
MaxPooling2D | output: | (None, 3, 3, 128)
dropout_1 | input: | (None, 3, 3, 128)
Dropout | output: | (None, 3, 3, 128)
flatten | input: | (None, 3, 3, 128)
Flatten | output: | (None, 1152)
dense | input: | (None, 1152)
Dense | output: | (None, 156)
dropout_2 | input: | (None, 156)
Dropout | output: | (None, 156)
dense_1 | input: | (None, 156)
Dense | output: | (None, 36)

Figure 5: Basic CNN architecture

The intuition behind stacking two convolutional layers before pooling layers is that stacking allows more flexibility in expressing non-linear transformations without losing information. Maxpool removes information from the signal, dropout forces distributed representation, thus both effectively make it harder to propagate information [3].

By using the best hyperparameters values obtained from the best individual from the evolutionary computing, we will be able to train our tuned basic CNN with our training data to evaluate the performance of our character recognition model. Here, we will train our model in 40 epochs to obtain best accuracy. The code below implements this:

```
# Train
history1 = model1.fit(X_reshaped, y_cat, epochs=40)
```

The model performance is displayed at the Project Results section of the report.

ResNet-50

To compare our tuned basic CNN with a untuned transfer-learning ML model, we implemented a sequential model containing the ResNet-50 model. The model architecture is displayed below:

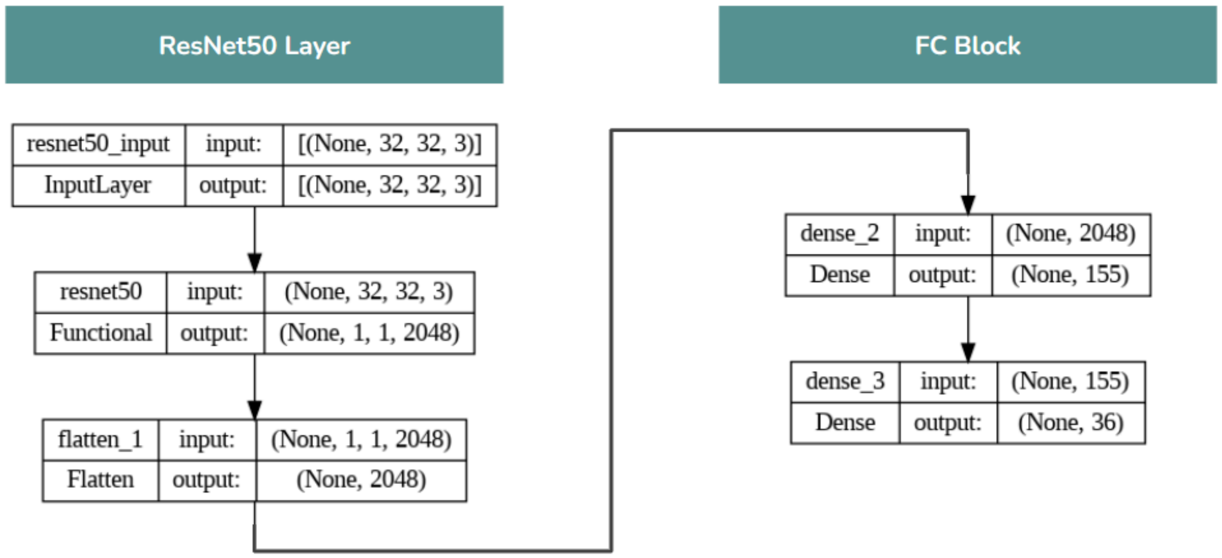

ResNet50 Layer
FC Block
resnet50_input | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
resnet50 | input: | (None, 32, 32, 3)
Functional | output: | (None, 1, 1, 2048)
flatten_1 | input: | (None, 1, 1, 2048)
Flatten | output: | (None, 2048)
dense_2 | input: | (None, 2048)
Dense | output: | (None, 155)
dense_3 | input: | (None, 155)
Dense | output: | (None, 36)

Figure 6: sequential model architecture with ResNet-50

The code implementation of this sequential model is as follows:

```
model2 = Sequential([

    # Load the pre-trained ResNet50 model without top layers
    ResNet50(weights="imagenet", include_top=False, input_shape=(32, 32, 3)),
    layers.Flatten(),
    layers.Dense(155, activation="relu"),
    layers.Dense(36, activation="softmax")
])

model2.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
```

The model used the preconstructed ResNet50 from TensorFlow library and its forward flow by adding the layers of that ResNet50 to the Sequential layers of the model.

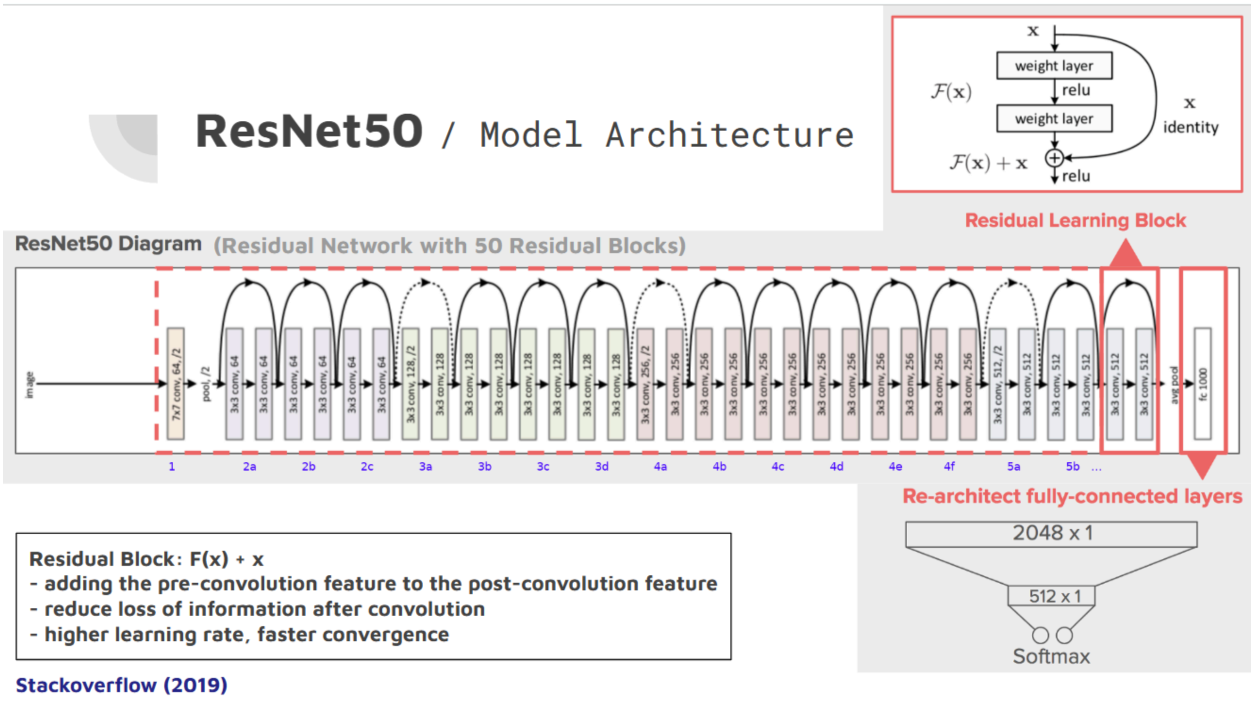

ResNet50 / Model Architecture
ResNet50 Diagram (Residual Network with 50 Residual Blocks)
Residual Learning Block
Re-architect fully-connected layers
2048 x 1
512 x 1
Softmax
Residual Block: F(x) + x
- adding the pre-convolution feature to the post-convolution feature
- reduce loss of information after convolution
- higher learning rate, faster convergence
Stackoverflow (2019)

Figure 7: ResNet50 Architecture

Setting the "weights" parameter of the ResNet50 to "imagenet" instantiated the all its layers and assigned them with the pretrained weight values. The parameter "include_top" was False to allow customisation of the dense layers of the pretrained ResNet50. The "input_shape" parameter was set to (32, 32, 3) to accept the image from dataset of 3-channel, 32-width and 32-height. The output feature of the ResNet50 was flattened to produce a 1-dimensional input for the dense layers AKA the fully connected layers.

The first dense layer formed full-connection between the N-feature of 1-dimensional input to 155-feature in 1-dimensional output using the activation function of relu. The final dense layer fully connected the 155-feature input to 36-feature which was the same as the number of classes for this particular model while using softmax for the classification. The model was compiled by using the Adam Optimizer with the initial learning rate of 0.001, while using "categorical cross entropy loss function" as the loss criterion, and "categorical accuracy" as the accuracy metrics.

Now, we can train our sequential model containing ResNet-50 using the `model.fit()` method.

```
# Train
history2 = model2.fit(
    tf.image.resize(np.repeat(X_reshaped,3,axis=-1).astype('float32')/255,[32,32]),
    y_cat,
    epochs=40
)
```

Because ResNet-50 model takes in an input shape of (32, 32, 3), we will need to resize our input image from (28, 28, 1) to (32, 32, 3) using the `tf.image.resize()` and `np.repeat()` methods. Then again, we train our model for 40 epochs.

The model performance comparison is visualized in the Project Results section. As a checkpoint, we saved both models into model files for future uses without needing to retrain the models again. The code cell below implements this step:

```
# Saving model
model1.save('best_model.h5')
model2.save('not_so_best_model.keras')
```

Fuzzy Logic

In this second step, we will get the recognized character from the first step, and use it to identify the author of this handwriting. With this, our Fuzzy System will only need to differentiate this image with the other image of the same character by different users.

To do this, we will need to identify a rule base containing rule sets for each character, i.e. a total of 36 rule sets are needed for this system to work.

Our approach to differentiating the handwriting for the same character is to calculate the skewness of the character by calculating gradients of several points. To do so, we look at an image and assign 3 points (pixel coordinates) to it, and then calculating the gradients between each points, resulting in 3 gradient values for each image. The 3 points are:

1. Top-Most Bright Pixel
2. Left-Most Bright Pixel
3. Right-Most Bright Pixel

The figure below visualize the concept above:

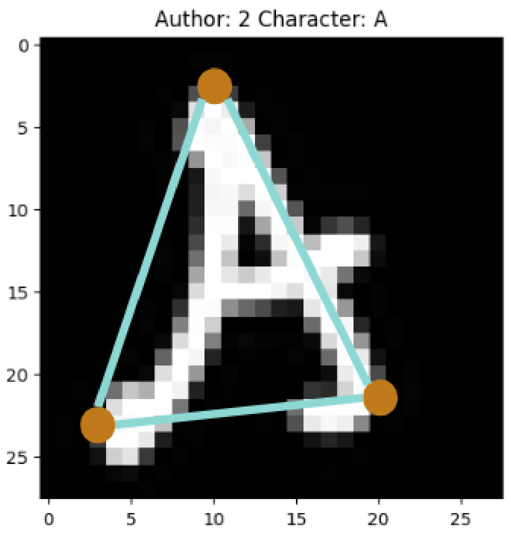
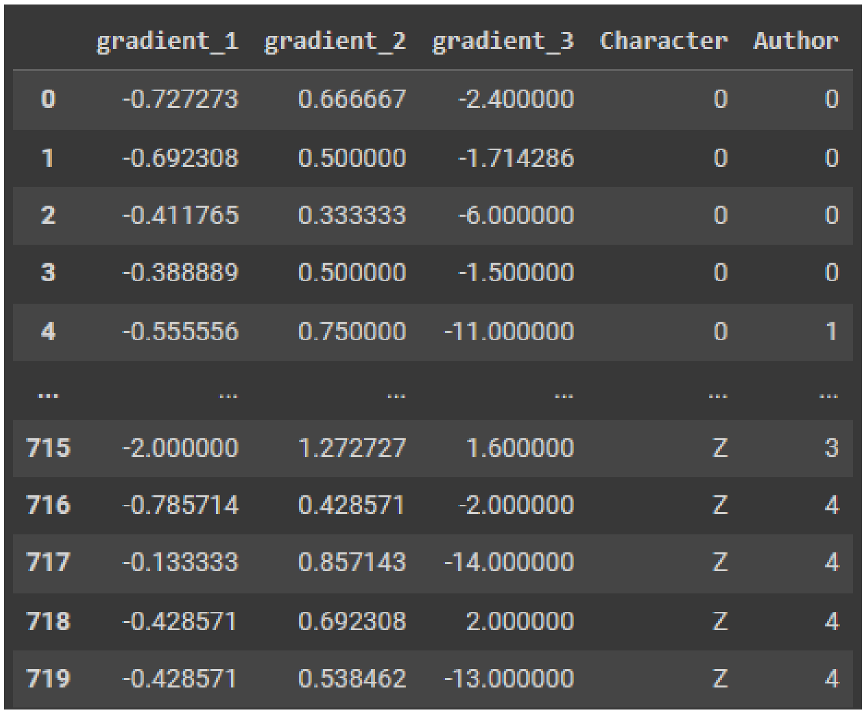


Figure 8: 3 point concept on an image

With this, we can log all of the gradient data together and perform analysis to identify which gradient values differentiate users from each other for that specific character. The code cell below implements the above concept:

```python
def calculate_gradient(x1, y1, x2, y2):

    if x1 == x2 and y1 == y2: return 0
    if x1 == x2: return y2 - y1
    return (y2 - y1) / (x2 - x1)

def get_gradient(row):

    row = row.iloc[:-2]

    # Get significant bright pixels
    bright_pixels = row[row > 50]
    bright_pixels.name = 'brightness'
    bright_pixels = bright_pixels.reset_index()
    bright_pixels['i'] = bright_pixels['index'].str.split(',').str[0].astype(int)
    bright_pixels['j'] = bright_pixels['index'].str.split(',').str[1].astype(int)

    # Get 3 points
    bright_pixels = bright_pixels.sort_values(['i','brightness'], ascending=[True,
False])
    top_i, top_j = bright_pixels.iloc[0][['i', 'j']].tolist()
    bright_pixels = bright_pixels.sort_values(['j', 'brightness'], ascending=[True,
False])
    left_i, left_j = bright_pixels.iloc[0][['i', 'j']].tolist()
    bright_pixels = bright_pixels.sort_values(['j', 'brightness'], ascending=[False,
False])
    right_i, right_j = bright_pixels.iloc[0][['i', 'j']].tolist()

    # Calculate gradients
    gradient_1 = calculate_gradient(left_i, left_j, top_i, top_j)
    gradient_2 = calculate_gradient(right_i, right_j, top_i, top_j)
    gradient_3 = calculate_gradient(left_i, left_j, right_i, right_j)

    return gradient_1, gradient_2, gradient_3
```

This results in a dataframe as shown in the figure below:



	gradient_1	gradient_2	gradient_3	Character	Author
0	-0.727273	0.666667	-2.400000	0	0
1	-0.692308	0.500000	-1.714286	0	0
2	-0.411765	0.333333	-6.000000	0	0
3	-0.388889	0.500000	-1.500000	0	0
4	-0.555556	0.750000	-11.000000	0	1
...	...	...	...	...	...
715	-2.000000	1.272727	1.600000	Z	3
716	-0.785714	0.428571	-2.000000	Z	4
717	-0.133333	0.857143	-14.000000	Z	4
718	-0.428571	0.692308	2.000000	Z	4
719	-0.428571	0.538462	-13.000000	Z	4

Figure 9: 3-gradient dataframe

With this, we can perform analysis by looking at each character one by one. But, it will be quite challenging to manually identify the gradient that is the most significant to be used in the membership function for each character, we will be using code automation to generate a set of fuzzy rule base. To test how well our automatic fuzzy rule base-generation perform, we will still design our own fuzzy rule base manually for each character.

Automated Fuzzy Rule Base Generation

To explain this part, let us visualize the gradient value distribution for character ‘0’ for each author using a distribution plot called Kernel Density Estimate (KDE) function [5].

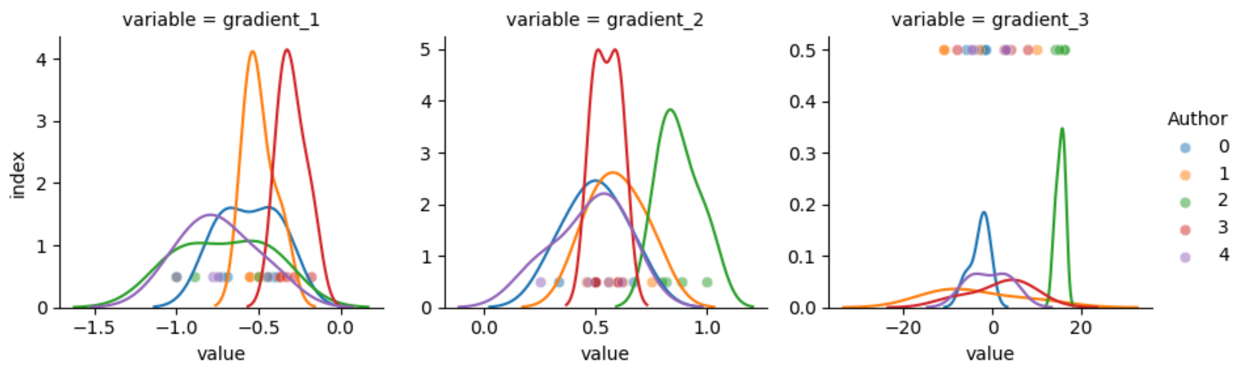


Figure 10: KDE plot for character ‘0’

As we can see, the values for each gradient are not distinctly clustered based on the author, hence we will need to identify the gradient where the peaks (high probability for that author to have this gradient value) for each author are separated as far as possible.

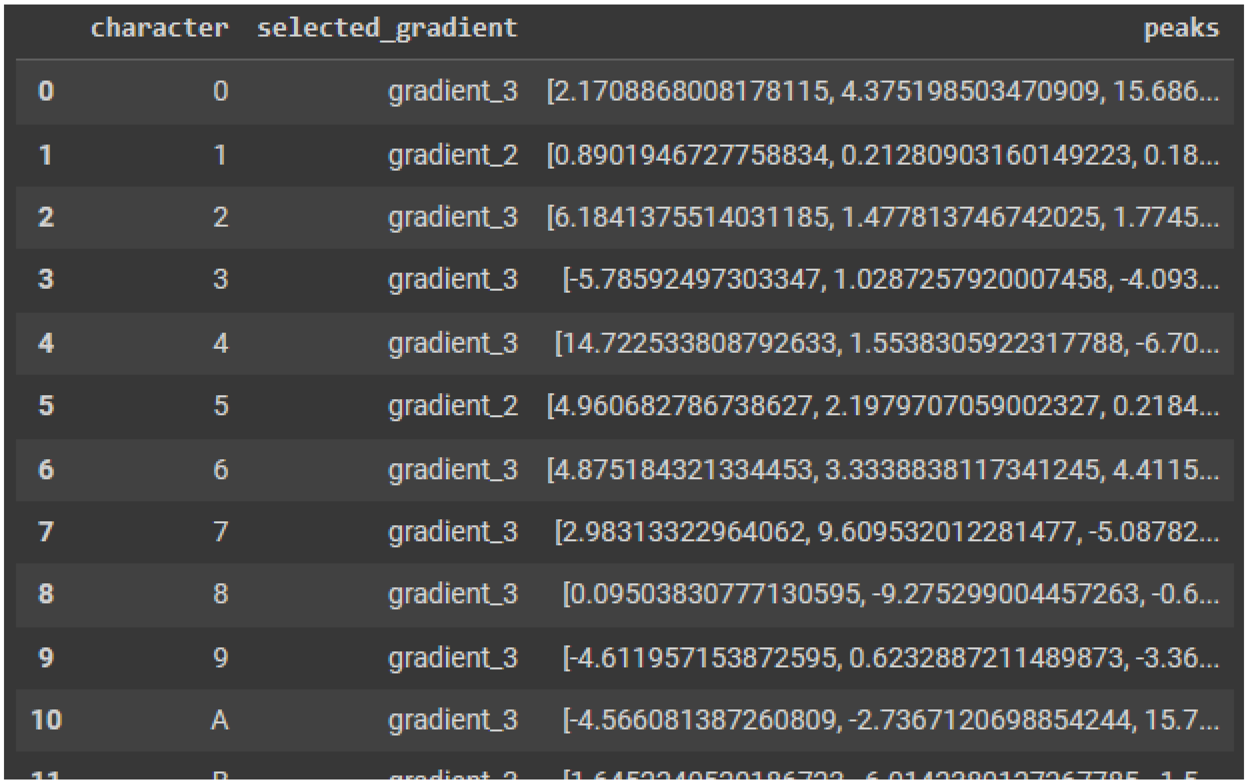
To start off, we will need to get the x-value for each peaks and identify which gradient gives the peaks that are distant to each other. The code snippet below implements this concept:

```
log['character'].append(char)

# Get gradient data
char_data = gradient_data[gradient_data['Character'] == char][['Author',
'gradient_1', 'gradient_2', 'gradient_3']]

gradient_peaks = []
gradient_range = []

for gradient in ['gradient_1', 'gradient_2', 'gradient_3']:

    # Reset plot
    plt.figure()

    ax = sns.kdeplot(data=char_data[[gradient, 'Author']], x=gradient, hue='Author')

    peaks = []

    if len(ax.lines) == 5:
        peaks.append(ax.lines[0].get_xdata()[np.argmax(ax.lines[0].get_ydata())])
        peaks.append(ax.lines[1].get_xdata()[np.argmax(ax.lines[1].get_ydata())])
        peaks.append(ax.lines[2].get_xdata()[np.argmax(ax.lines[2].get_ydata())])
        peaks.append(ax.lines[3].get_ydata()[np.argmax(ax.lines[3].get_ydata())])
        peaks.append(ax.lines[4].get_ydata()[np.argmax(ax.lines[4].get_ydata())])

    # Some points are all same for that author => no KDE line => peak set to the
point itself
    else:
        lines = deque(ax.lines)
        for author in char_data['Author'].unique():
            if len(char_data[char_data['Author'] == author][gradient].unique()) ==
1:
                peaks.append(*char_data[char_data['Author'] ==
author][gradient].unique())
            else:
                line = lines.popleft()
                peaks.append(line.get_xdata()[np.argmax(line.get_ydata())])

    gradient_peaks.append(peaks)
    ranges = max(peaks) - min(peaks)
    gradient_range.append(ranges)

log['selected_gradient'].append(f'gradient_{np.argmax(gradient_range) + 1}')
log['peaks'].append(gradient_peaks[np.argmax(gradient_range)])
```

In this code snippet, we utilize the seaborn kdeplot function and obtain the lines from each gradient's kde plot, and get the x-value when the y-value for that line is the highest using the `np.argmax()` method. However, there are certain situations where the gradient values are exactly the same (i.e. no variation for that user), so the kde curve will be unable to generate.

To handle this problem, we checked if the lines from the kdeplot is exactly 5, if it isn't, i.e. some author's gradient values are the same and no kde curves for it, we will need to manually append the gradient value itself.

After this step, we will identify which peaks are the most distantly separated. The easiest concept that we go for is to identify which set of peaks has the highest statistical range, meaning that at least the smallest peak x-value and the largest x-value are very distinct. Then, we obtain a log data for the process above displayed as below:



	character	selected_gradient	peaks
0	0	gradient_3	[2.1708868008178115, 4.375198503470909, 15.686...
1	1	gradient_2	[0.8901946727758834, 0.21280903160149223, 0.18...
2	2	gradient_3	[6.1841375514031185, 1.477813746742025, 1.7745...
3	3	gradient_3	[-5.78592497303347, 1.0287257920007458, -4.093...
4	4	gradient_3	[14.722533808792633, 1.5538305922317788, -6.70...
5	5	gradient_2	[4.960682786738627, 2.1979707059002327, 0.2184...
6	6	gradient_3	[4.875184321334453, 3.3338838117341245, 4.4115...
7	7	gradient_3	[2.98313322964062, 9.609532012281477, -5.08782...
8	8	gradient_3	[0.09503830777130595, -9.275299004457263, -0.6...
9	9	gradient_3	[-4.611957153872595, 0.6232887211489873, -3.36...
10	A	gradient_3	[-4.566081387260809, -2.7367120698854244, 15.7...

Figure 11: log dataframe

With this, we can generate the fuzzy rule membership function for this antecendent using this peaks, sorting them in ascending order and assign the membership as 0, 1, 2, 3, 4, indicating "very low", "low", "mid", "high", "very high". The consequent membership function will be simple triangular membership function for each author 0, 1, 2, 3, 4.

The following code segment implements this step:

```python
def evaluate_fuzzy_model(chosen_gradient_val, char):

    char_peaks = log_df[log_df['character'] == char]['peaks'].tolist()[0]
    a, b, c, d, e = sorted_char_peaks = sorted(char_peaks)
    rank_1, rank_2, rank_3, rank_4, rank_5 = [sorted_char_peaks.index(x) for x in
char_peaks]

    # Antecedent
    peaks = ctrl.Antecedent(np.arange(-28, 28.01, 0.01), f'peaks \'{char}\'')
    peaks['0'] = fuzz.trapmf(peaks.universe, [-28, -28, a, b])
    peaks['1'] = fuzz.trimf(peaks.universe, [a, b, c])
    peaks['2'] = fuzz.trimf(peaks.universe, [b, c, d])
    peaks['3'] = fuzz.trimf(peaks.universe, [c, d, e])
    peaks['4'] = fuzz.trapmf(peaks.universe, [d, e, 28, 28])

    # Consequent
    author = ctrl.Consequent(np.arange(0, 4.1, 0.1), 'author')
    author['0'] = fuzz.trimf(author.universe, [0, 0, 0.5])
    author['1'] = fuzz.trimf(author.universe, [0.5, 1, 1.5])
    author['2'] = fuzz.trimf(author.universe, [1.5, 2, 2.5])
    author['3'] = fuzz.trimf(author.universe, [2.5, 3, 3.5])
    author['4'] = fuzz.trimf(author.universe, [3.5, 4, 4])
```

The membership functions for antecedent ‘Y’ and consequent looks like this:

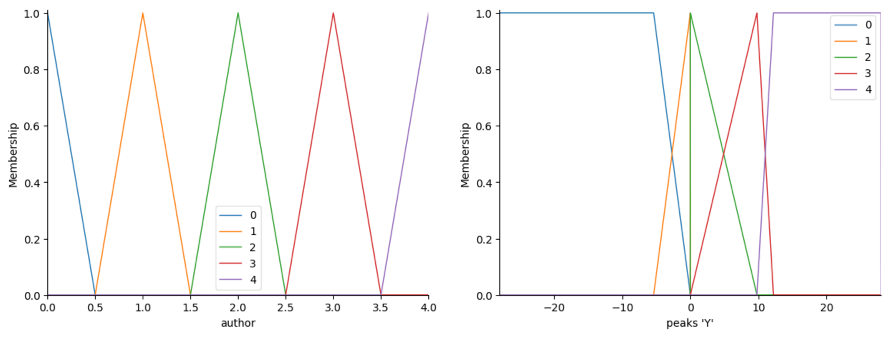


Figure 12: membership functions for antecedent and consequent

The automated fuzzy rule generation uses a simple rule generation, that is a “IF … THEN …” rule. Since we have identified which author’s handwriting gradient value for in that peak, we can just do something like “IF peak is low THEN author is 1”. The code segment below implements this concept:

```
# Rule
rules = [
    ctrl.Rule(peaks[f'{rank_1}'], author['0']),
    ctrl.Rule(peaks[f'{rank_2}'], author['1']),
    ctrl.Rule(peaks[f'{rank_3}'], author['2']),
    ctrl.Rule(peaks[f'{rank_4}'], author['3']),
    ctrl.Rule(peaks[f'{rank_5}'], author['4']),
]
```

Finally, we can input everything into the fuzzy system controller to make prediction:

```
# Control System
fuzzy_control_system = ctrl.ControlSystem(rules)

# Simulation
fuzzy_simulator = ctrl.ControlSystemSimulation(fuzzy_control_system)

# Prediction
fuzzy_simulator.input[f'peaks \'{char}\''] = chosen_gradient_val
fuzzy_simulator.compute()

return fuzzy_simulator.output['author']
```

We evaluate the fuzzy system using the code below:

```
result = {
    'prediction': [],
    'prediction2': [],
    'actual': gradient_data['Author'].astype(int).tolist()
}

for _, row in gradient_data.iterrows():

    char = row['Character']

    chosen_gradient_val = row[
        ['gradient_1', 'gradient_2', 'gradient_3'].index(
            log_df[log_df['character'] == char]['selected_gradient'].tolist()[0]
        )
    ]

    evaluated_result = evaluate_fuzzy_model(chosen_gradient_val, char)

    result['prediction'] += [evaluated_result]
    result['prediction2'] += [round(evaluated_result)]

# View Prediction Result
result_df = pd.DataFrame({
    'fuzzy_pred': result['prediction'],
    'fuzzy_pred_class': result['prediction2'],
    'actual_class': result['actual']
})
```

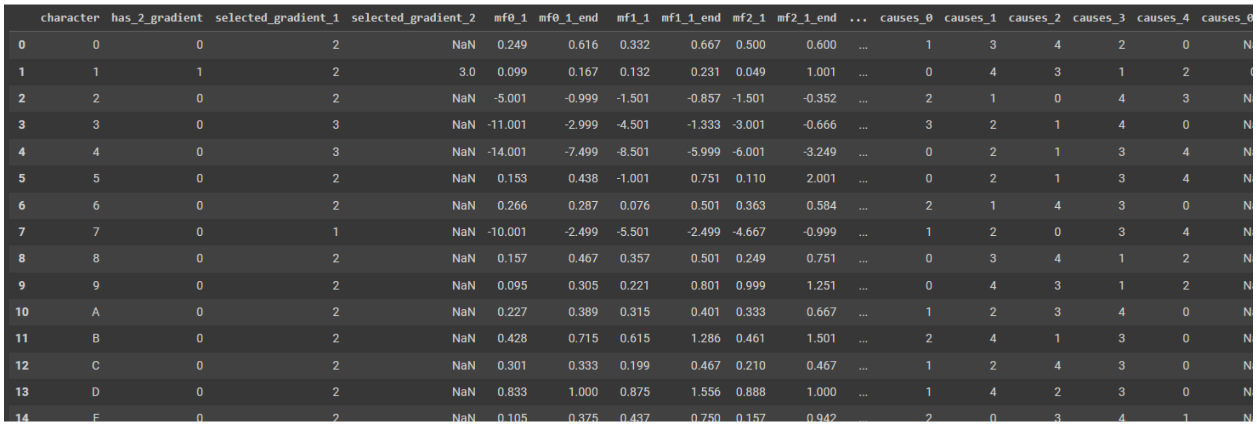
The output df will contain all predictions from this fuzzy system for further analysis on the model's performance.

With it, we can compute the RMSE and Accuracy of this automated system, where the performance is displayed in the Project Results section.

```
# Measure Prediction Metrics
rmse = np.sqrt(mean_squared_error(result['actual'], result['prediction']))
acc = accuracy_score(result['actual'], result['prediction2'])

print(f'rmse: {rmse}, accuracy: {acc}')
```

Manual Fuzzy Rule Base Generation

To verify if the automated rule generation is better, we compare it with our manually identified rule base, which we compiled all information for the 36 characters in a csv file that looks like:

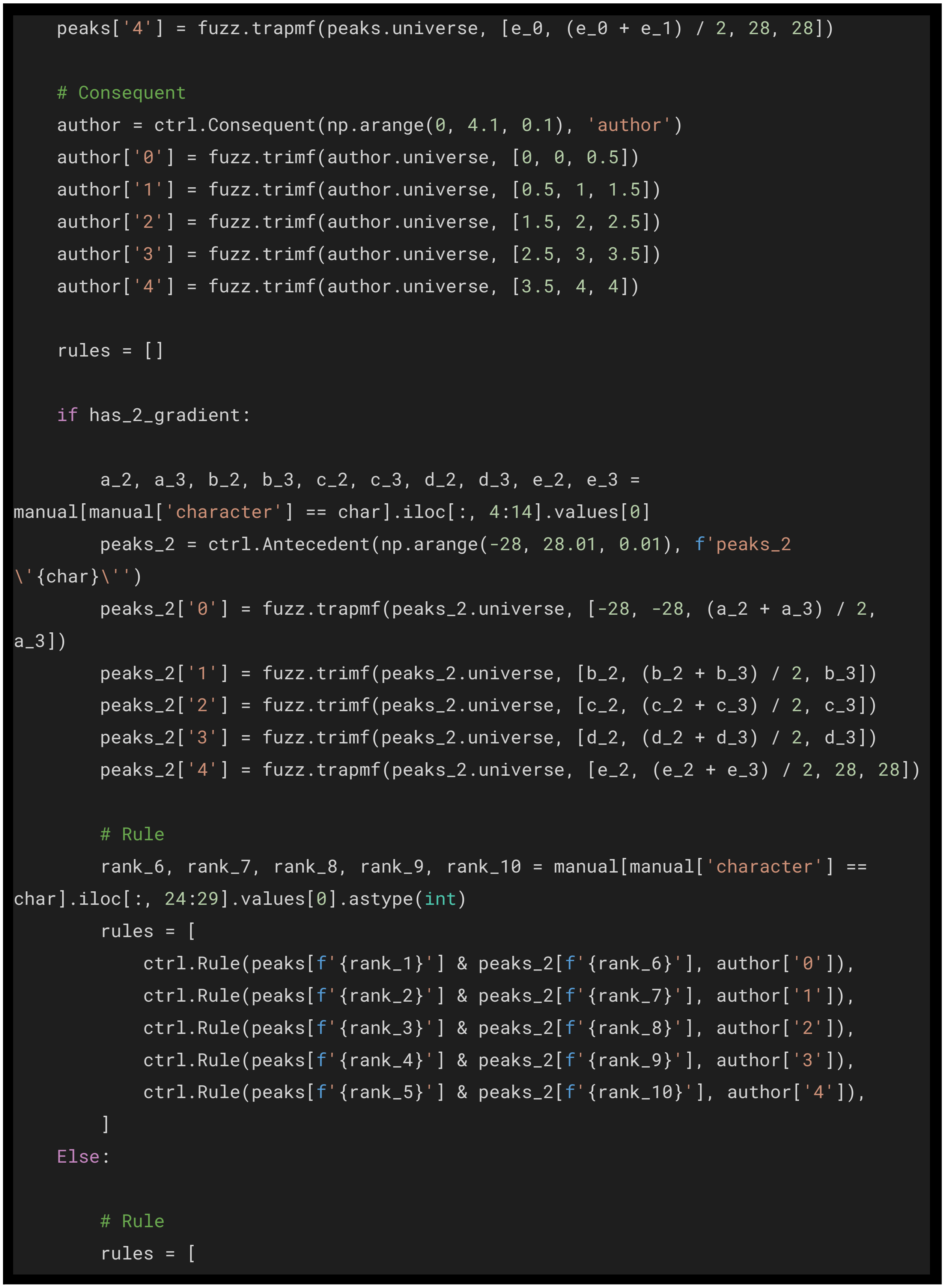


	character	has_2_gradient	selected_gradient_1	selected_gradient_2	mf0_1	mf0_1_end	mf1_1	mf1_1_end	mf2_1	mf2_1_end	...	causes_0	causes_1	causes_2	causes_3	causes_4	causes_0
0	0	0	2	NaN	0.249	0.616	0.332	0.667	0.500	0.600	...	1	3	4	2	0	N
1	1	1	2	3.0	0.099	0.167	0.132	0.231	0.049	1.001	...	0	4	3	1	2	[illegible]
2	2	0	2	NaN	-5.001	-0.999	-1.501	-0.857	-1.501	-0.352	...	2	1	0	4	3	N
3	3	0	3	NaN	-11.001	-2.999	-4.501	-1.333	-3.001	-0.666	...	3	2	1	4	0	N
4	4	0	3	NaN	-14.001	-7.499	-8.501	-5.999	-6.001	-3.249	...	0	2	1	3	4	N
5	5	0	2	NaN	0.153	0.438	-1.001	0.751	0.110	2.001	...	0	2	1	3	4	N
6	6	0	2	NaN	0.266	0.287	0.076	0.501	0.363	0.584	...	2	1	4	3	0	N
7	7	0	1	NaN	-10.001	-2.499	-5.501	-2.499	-4.667	-0.999	...	1	2	0	3	4	N
8	8	0	2	NaN	0.157	0.467	0.357	0.501	0.249	0.751	...	0	3	4	1	2	N
9	9	0	2	NaN	0.095	0.305	0.221	0.801	0.999	1.251	...	0	4	3	1	2	N
10	A	0	2	NaN	0.227	0.389	0.315	0.401	0.333	0.667	...	1	2	3	4	0	N
11	B	0	2	NaN	0.428	0.715	0.615	1.286	0.461	1.501	...	2	4	1	3	0	N
12	C	0	2	NaN	0.301	0.333	0.199	0.467	0.210	0.467	...	1	2	4	3	0	N
13	D	0	2	NaN	0.833	1.000	0.875	1.556	0.888	1.000	...	1	4	2	3	0	N
14	E	0	2	NaN	0.105	0.375	0.437	0.750	0.157	0.942	...	2	0	3	4	1	N

Figure 13: manual rule base data

In comparison to the automated ones, we changed the membership function generation to range basis instead of the peak-basis in the automated one. Then, some of the character's fuzzy rule set contains the AND operator to further improve the accuracy, so the code implementation is as follows:

```python
def evaluate_fuzzy_model_manual(chosen_gradient_val_1, chosen_gradient_val_2,
has_2_gradient, char):

    a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1, e_0, e_1 = manual[manual['character'] ==
char].iloc[:, 4:14].values[0]
    rank_1, rank_2, rank_3, rank_4, rank_5 = manual[manual['character'] ==
char].iloc[:, 24:29].values[0].astype(int)

    # Antecedent
    peaks = ctrl.Antecedent(np.arange(-28, 28.001, 0.001), f'peaks \'{char}\'')
    peaks['0'] = fuzz.trapmf(peaks.universe, [-28, -28, (a_0 + a_1) / 2, a_1])
    peaks['1'] = fuzz.trimf(peaks.universe, [b_0, (b_0 + b_1) / 2, b_1])
    peaks['2'] = fuzz.trimf(peaks.universe, [c_0, (c_0 + c_1) / 2, c_1])
    peaks['3'] = fuzz.trimf(peaks.universe, [d_0, (d_0 + d_1) / 2, d_1])
```



```
    peaks['4'] = fuzz.trapmf(peaks.universe, [e_0, (e_0 + e_1) / 2, 28, 28])

    # Consequent
    author = ctrl.Consequent(np.arange(0, 4.1, 0.1), 'author')
    author['0'] = fuzz.trimf(author.universe, [0, 0, 0.5])
    author['1'] = fuzz.trimf(author.universe, [0.5, 1, 1.5])
    author['2'] = fuzz.trimf(author.universe, [1.5, 2, 2.5])
    author['3'] = fuzz.trimf(author.universe, [2.5, 3, 3.5])
    author['4'] = fuzz.trimf(author.universe, [3.5, 4, 4])

    rules = []

    if has_2_gradient:

        a_2, a_3, b_2, b_3, c_2, c_3, d_2, d_3, e_2, e_3 =
manual[manual['character'] == char].iloc[:, 4:14].values[0]
        peaks_2 = ctrl.Antecedent(np.arange(-28, 28.01, 0.01), f'peaks_2
\'{char}\'')
        peaks_2['0'] = fuzz.trapmf(peaks_2.universe, [-28, -28, (a_2 + a_3) / 2,
a_3])
        peaks_2['1'] = fuzz.trimf(peaks_2.universe, [b_2, (b_2 + b_3) / 2, b_3])
        peaks_2['2'] = fuzz.trimf(peaks_2.universe, [c_2, (c_2 + c_3) / 2, c_3])
        peaks_2['3'] = fuzz.trimf(peaks_2.universe, [d_2, (d_2 + d_3) / 2, d_3])
        peaks_2['4'] = fuzz.trapmf(peaks_2.universe, [e_2, (e_2 + e_3) / 2, 28, 28])

        # Rule
        rank_6, rank_7, rank_8, rank_9, rank_10 = manual[manual['character'] ==
char].iloc[:, 24:29].values[0].astype(int)
        rules = [
            ctrl.Rule(peaks[f'{rank_1}'] & peaks_2[f'{rank_6}'], author['0']),
            ctrl.Rule(peaks[f'{rank_2}'] & peaks_2[f'{rank_7}'], author['1']),
            ctrl.Rule(peaks[f'{rank_3}'] & peaks_2[f'{rank_8}'], author['2']),
            ctrl.Rule(peaks[f'{rank_4}'] & peaks_2[f'{rank_9}'], author['3']),
            ctrl.Rule(peaks[f'{rank_5}'] & peaks_2[f'{rank_10}'], author['4']),
        ]
    Else:

        # Rule
        rules = [
            ctrl.Rule(peaks[f'{rank_1}'], author['0']),
            ctrl.Rule(peaks[f'{rank_2}'], author['1']),
            ctrl.Rule(peaks[f'{rank_3}'], author['2']),
            ctrl.Rule(peaks[f'{rank_4}'], author['3']),
            ctrl.Rule(peaks[f'{rank_5}'], author['4']),
        ]

    # Control System
    fuzzy_control_system = ctrl.ControlSystem(rules)

    # Simulation
    fuzzy_simulator = ctrl.ControlSystemSimulation(fuzzy_control_system)

    # Prediction
    fuzzy_simulator.input[f'peaks \'{char}\''] = chosen_gradient_val_1
    if has_2_gradient: fuzzy_simulator.input[f'peaks_2 \'{char}\''] = 
chosen_gradient_val_2

    fuzzy_simulator.compute()
```

Here, the antecedent membership function for the character ‘Y’ looks like:

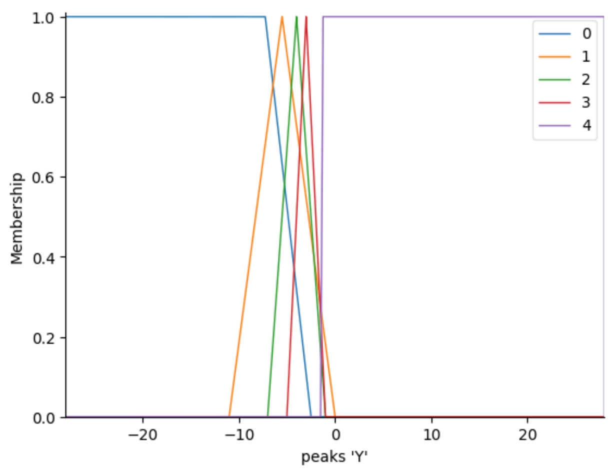


Figure 14: antecedent membership function

Again, the performance of this fuzzy system is displayed at the Project Results section.

Project Results and Discussion

All of the output results from the previous section are listed here, under each respective category.

Results: Genetic Algorithm

Just like what we had explain in the above section, we compare 2 different genetic algorithm of different configuration. We can see that the fitness value of Ranked Selection and Uniform Crossover Configuration reached the highest at 3rd generation, which is 0.9653, but starts to decline afterward. Even though there are some up and down till the end of all the generations, the fitness value still doesn't go back up and exceed its previous peek.

Rank Selection selects an individual based on its proportion of fitness value among the other individuals in that generation. When the individuals in the population have very close fitness values (which happens usually at the end of the run), each individual has an almost equal share of the pie (equal fitness proportion), and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent. This in turn leads to a loss in the selection pressure towards fitter individuals, making the genetic algorithm to make poor parent selections in such situations [6].

Meanwhile, for the Elitism Selection and Two Point Crossover Configuration, the fitness value of the genetic algorithm is actually increasing steadily across generation, the peak of it is not above the first algorithm, which is 0.9500, but we can still see a trend of rising in the second algorithm.

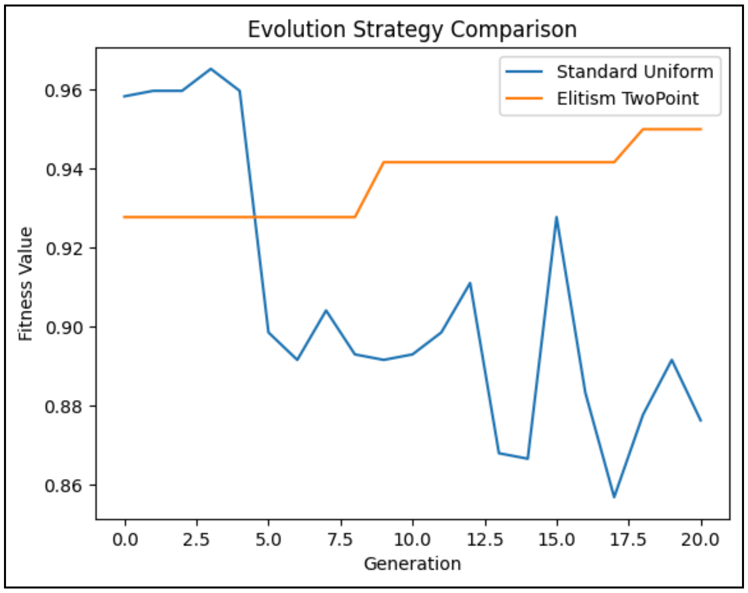
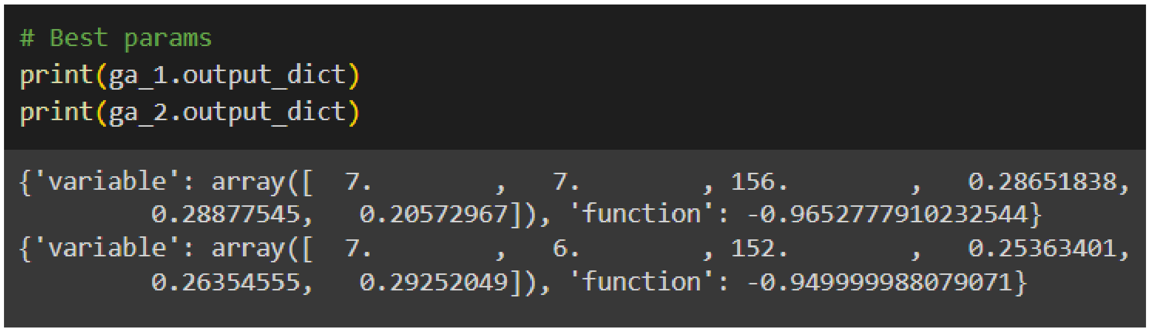


Figure 15: Graph of the fitness score vs generations for both genetic algorithm

After plotting the graph of the fitness score vs generations, we can find out the best individual for each gene amongst all the 20 generations.



```
# Best params
print(ga_1.output_dict)
print(ga_2.output_dict)
```

```
{'variable': array([  7.        ,   7.        , 156.        ,   0.28651838,
         0.28877545,   0.20572967]), 'function': -0.9652777910232544}
{'variable': array([  7.        ,   6.        , 152.        ,   0.25363401,
         0.26354555,   0.29252049]), 'function': -0.949999988079071}
```

Figure 16: Output of both genetic algorithm

Based on the above image, for the first Convolutional Block of the first genetic algorithm, we can see that the best Block 1 Kernels (conv2d_1) will be **7**, which will be 2 layers of 2^7 = 128 kernels-Conv2d. The dropout rate of Block 1 will be **0.29.** Moving on to the second block, the best Block 2 Kernels(conv2d_2_ will also be **7**, which will be 2 layers of 2^7 = 128 kernels-Conv2d. The dropout rate of Block 2 will be **0.29.** In the third block, the Block 3 FC Layer Neurons best value will be **156**, which means 1 fully-conencted layer of 156 neurons, together with a dropout rate of **0.21** as the best.

For the first Convolutional Block of the second genetic algorithm, we can see that the best Block 1 Kernels (conv2d_1) will also be **7**, which will be 2 layers of 2^7 = 128 kernels-Conv2d. The dropout rate of Block 1 will be **0.29.** Moving on to the second block, the best Block 2 Kernels(conv2d_2_ will be **6**, which will be 2 layers of 2^6 = 64 kernels-Conv2d. The dropout rate of Block 2 will be **0.26.** In the third block, the Block 3 FC Layer Neurons best value will be **152**, which means 1 fully-conencted layer of 152 neurons, together with a dropout rate of **0.29** as the best.

Since the Ranked Selection and Uniform Crossover Configuration Genetic Algorithm has the highest best fitness score, so the the gene of this algorithm is used to set as the best configuration for the basic CNN architecture.

Results: Neural Network

After using the best configuration from genetic algorithm and ran the two neural network models, one is the basic tuned CNN model, and the other one is the ResNet-50 model.

The graph below is a graph of training accuracy over epoch for both CNN model and ResNet-50 model. We can tell that the tuned CNN model is having a better accuracy most of the time except the first 5 epoch before the ResNet-50 model starts to have a slight dip in accuracy. The highest accuracy reached by the tuned CNN model is 0.993, meanwhile the highest accuracy reached by ResNet50 is 0.975.

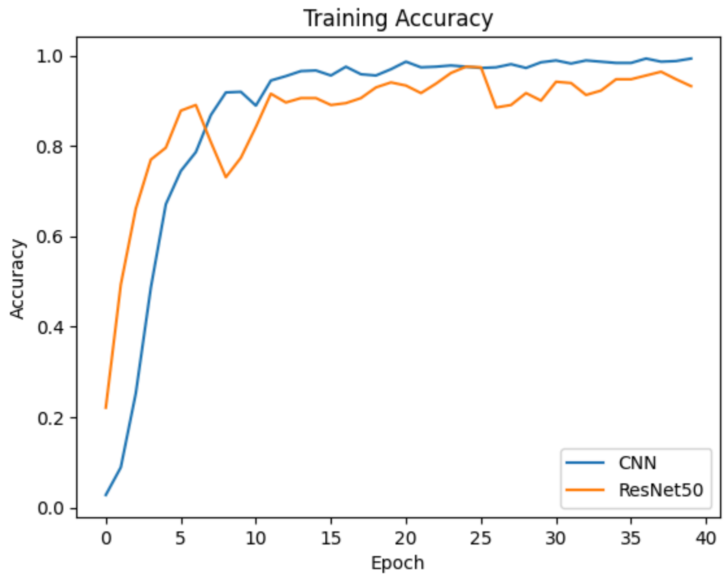


Figure 17: Graph of training accuracy vs number of epoch of both neural network models

The graph below is a graph of training loss over epoch for both CNN model and ResNet-50 model. Overall Loss: The tuned CNN model appears to have a lower overall loss compared to the ResNet50 model, especially after around 10 epochs. This suggests that the tuned CNN model is performing better on the training data than the ResNet50 model. The tuned CNN model seems to have lower training loss and smoother convergence compared to the ResNet50 model, suggesting that it is better suited for this MNIST dataset.

The main reason tuned CNN performs better is because we are tuning it using the best configuration obtained from the genetic algorithm like we mentioned earlier, for ResNet-50 its result is still good, but just lower than the fine-tuned CNN model. Further tuning of the ResNEt-50 model will further improve its accuracy.

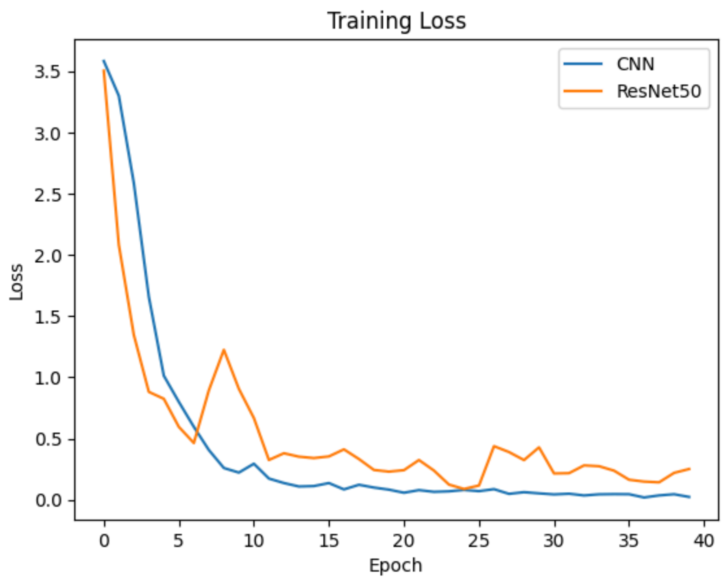


Figure 18: Graph of training loss vs number of epoch of both neural network models

Based on the loss and accuracy graphs above, we noticed that the graphs for ResNet50 were unstable with irregular fluctuations. The root cause for this issue was due to overfitting at the epoch prior to fluctuation as the ResNet50 model might be too dense for this project. The possible solutions for future implementation included:

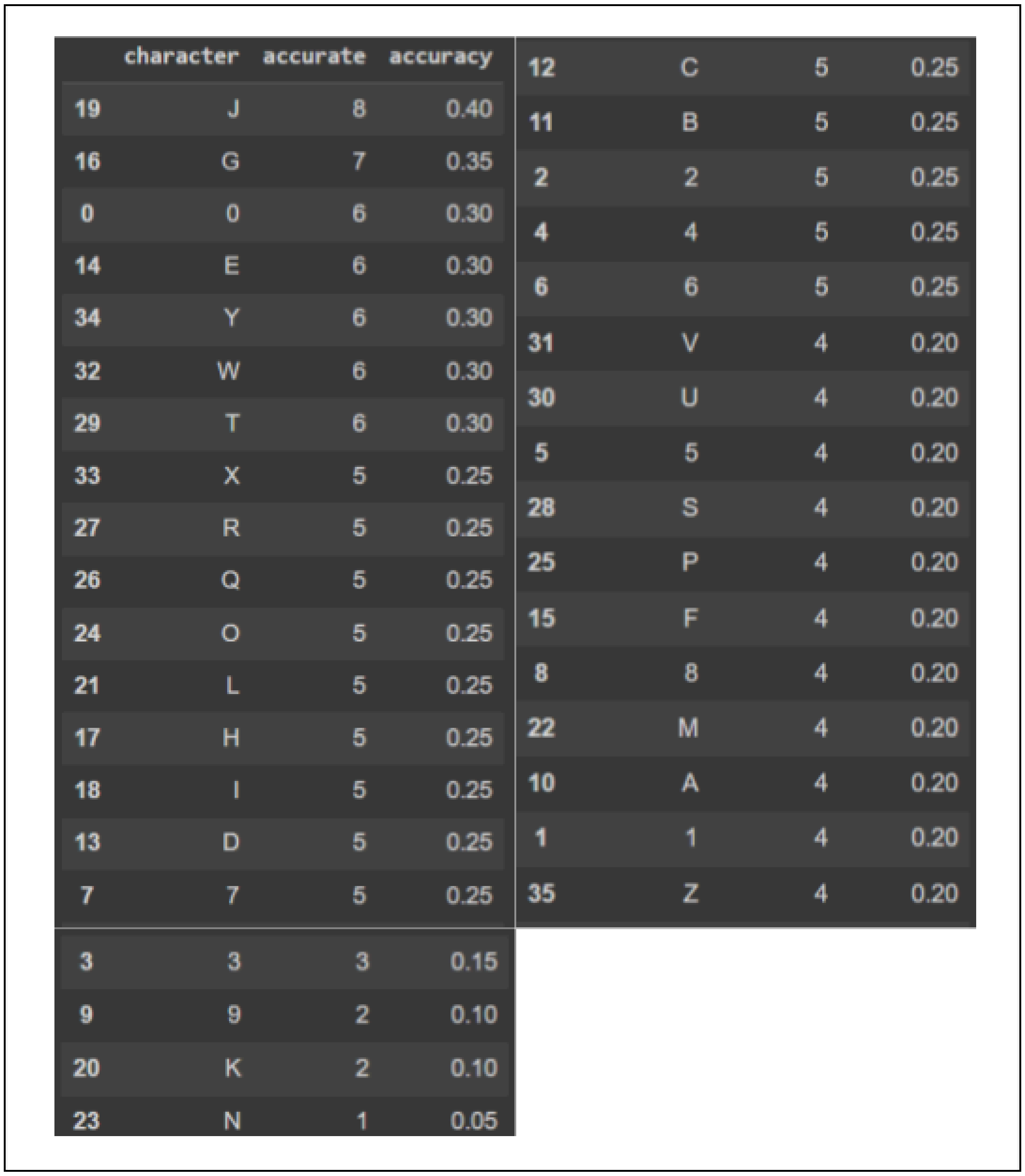
- Reduce number of Residual Blocks by using ResNet34 or ResNet18
- Increase size and variation of dataset
- Fine tune training params such as increasing batch size
- Fine tune optimizer params and scheduler params

Results: Fuzzy System

For the fuzzy system, just like what we explained in the previous section, the automated generation fuzzy system is having accuracy score of 0.232, meanwhile the manual generation fuzzy system is having accuracy score of 0.197.

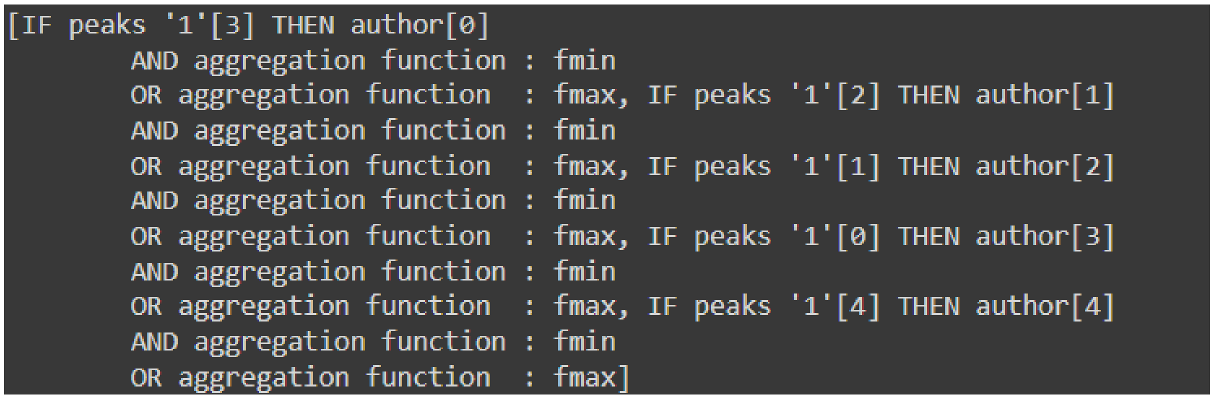
The main reason that cause both the fuzzy system to have such low accuracy is mainly due to how the handwriting for different author can either be very similar, especially for some certain characters like N, K, 9 and so on. And for each character, we have 4 more set of copies somemore, and all the copies from same author might have some differences also, which may be due to different mental state, energy level or other factors [8], making it very hard to actually set a consistent and accurate rules to define the fuzzy system. The overlapping of fuzzy rules with small difference in between all membership functions defined for the system is one of the main reason why boht of our fuzzy system registered such low accuracy in prediction which author the character belongs to.

Comparing both fuzzy systems, the main reason that the manual generation fuzzy system has a lower accuracy is because manually extraction of fuzzy rules and membership function might not be mathematically accurate and balance. There exist some bias where the person that extract the membership function and fuzzy rules might have some different opinions from the supposedly best choice. Compared to how the automated generation fuzzy system where consistency is guaranteed, it is clear why the manual generation is having a lower accuracy.



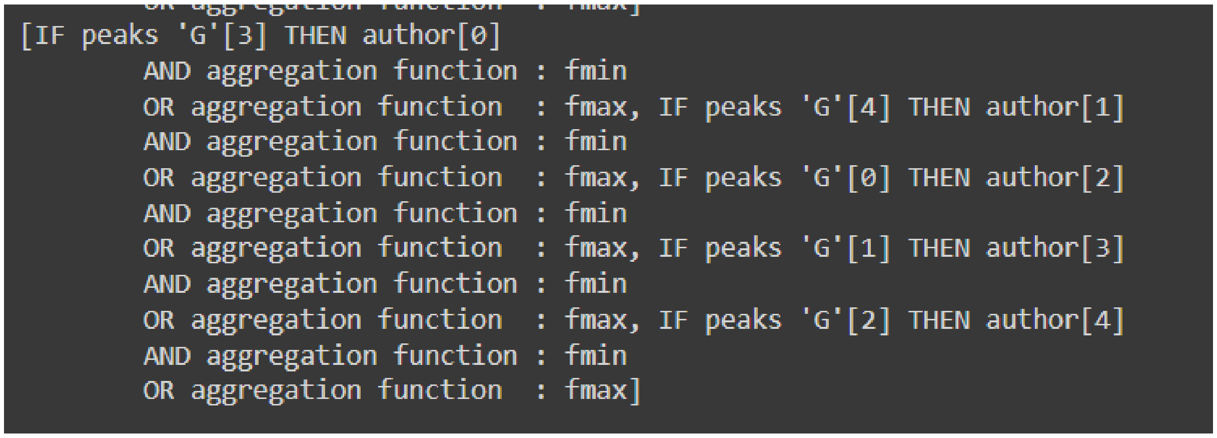
	character	accurate	accuracy
19	J	8	0.40
16	G	7	0.35
0	0	6	0.30
14	E	6	0.30
34	Y	6	0.30
32	W	6	0.30
29	T	6	0.30
33	X	5	0.25
27	R	5	0.25
26	Q	5	0.25
24	O	5	0.25
21	L	5	0.25
17	H	5	0.25
18	I	5	0.25
13	D	5	0.25
7	7	5	0.25
12	C	5	0.25
11	B	5	0.25
2	2	5	0.25
4	4	5	0.25
6	6	5	0.25
31	V	4	0.20
30	U	4	0.20
5	5	4	0.20
28	S	4	0.20
25	P	4	0.20
15	F	4	0.20
8	8	4	0.20
22	M	4	0.20
10	A	4	0.20
1	1	4	0.20
35	Z	4	0.20
3	3	3	0.15
9	9	2	0.10
20	K	2	0.10
23	N	1	0.05

Figure 19: Accuracy for all character using Automated Generation Fuzzy System



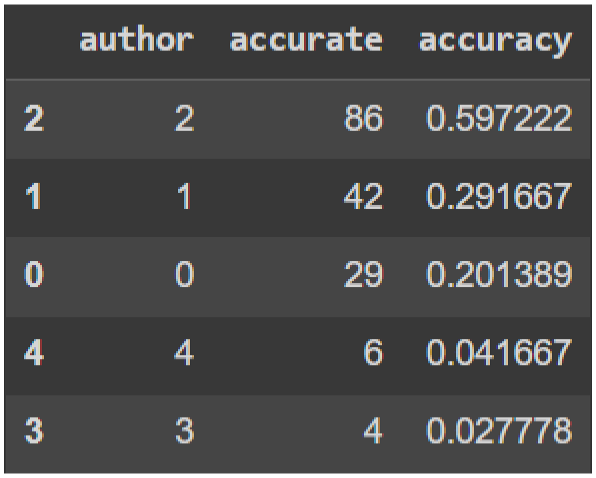
```
[IF peaks '1'[3] THEN author[0]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks '1'[2] THEN author[1]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks '1'[1] THEN author[2]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks '1'[0] THEN author[3]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks '1'[4] THEN author[4]
	AND aggregation function : fmin
	OR aggregation function  : fmax]
```

Figure 20: Sample Fuzzy Rule for Character ‘1’ in Automated Generation Fuzzy System



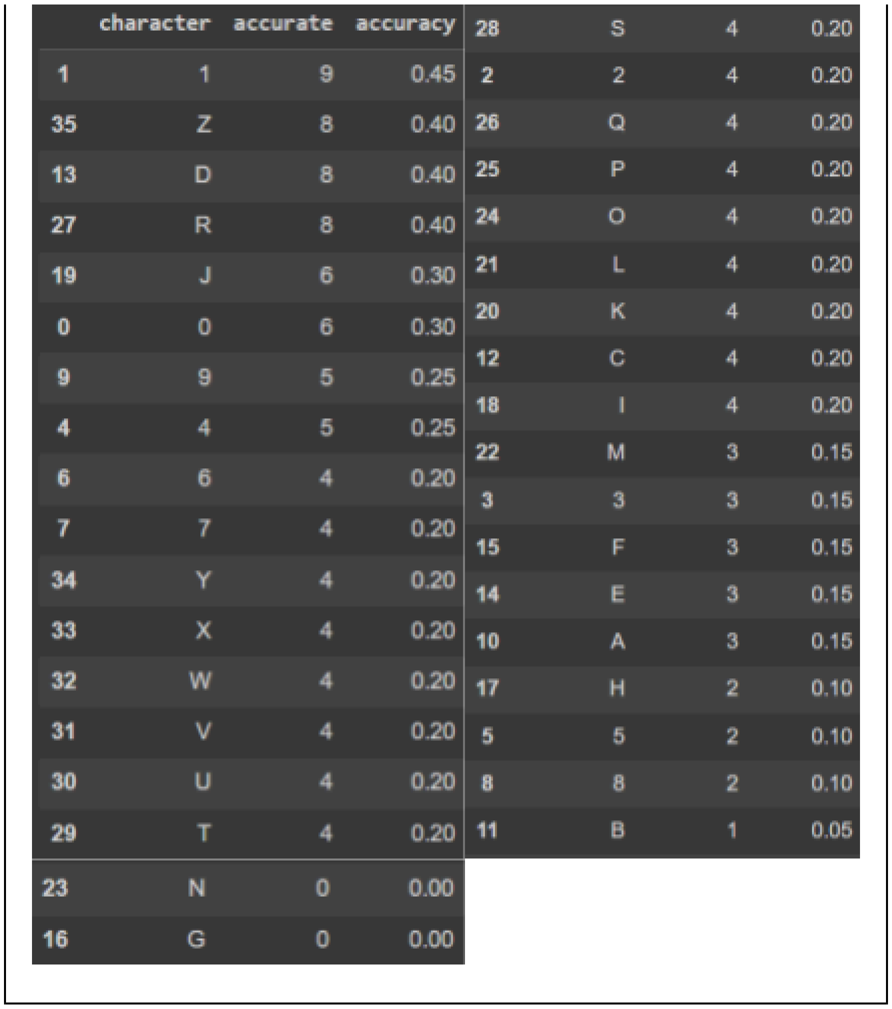
```
	OR aggregation function  : fmax]
[IF peaks 'G'[3] THEN author[0]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks 'G'[4] THEN author[1]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks 'G'[0] THEN author[2]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks 'G'[1] THEN author[3]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks 'G'[2] THEN author[4]
	AND aggregation function : fmin
	OR aggregation function  : fmax]
```

Figure 21: Sample Fuzzy Rule for Character ‘G’ in Automated Generation Fuzzy System



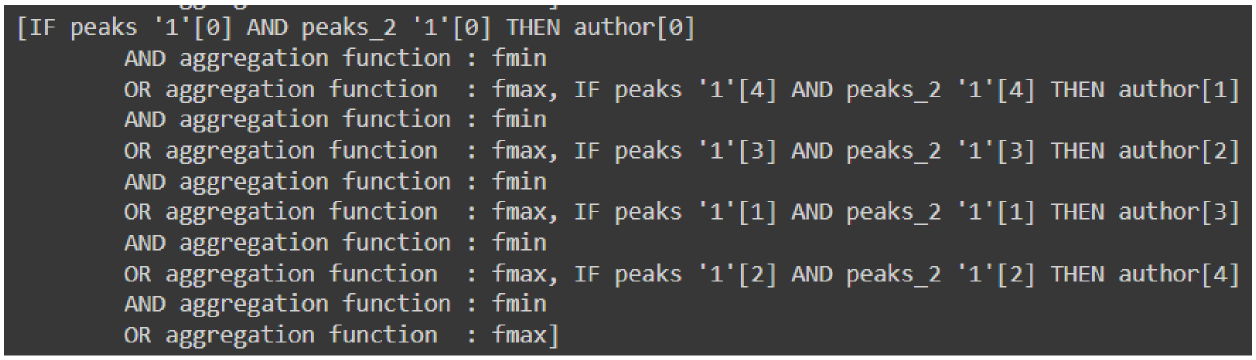
	author	accurate	accuracy
2	2	86	0.597222
1	1	42	0.291667
0	0	29	0.201389
4	4	6	0.041667
3	3	4	0.027778

Figure 22: Accuracy of Fuzzy System for each author in Automated Generation Fuzzy System



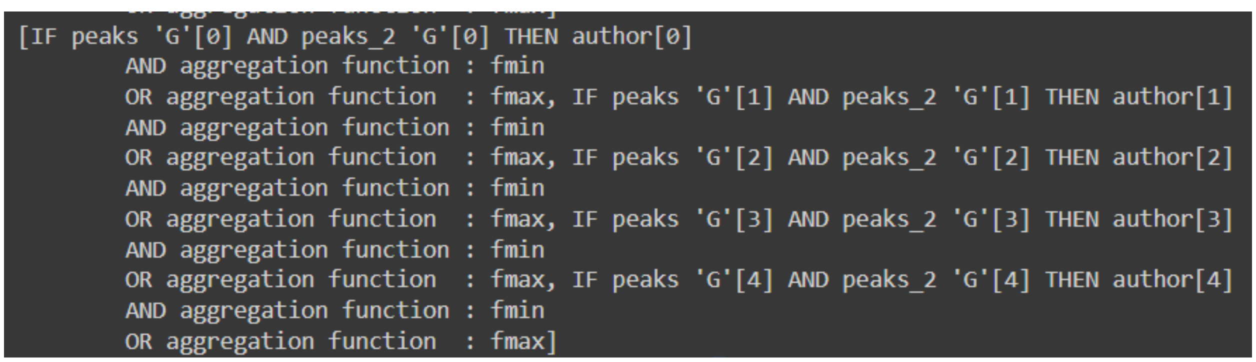
	character	accurate	accuracy
1	1	9	0.45
35	Z	8	0.40
13	D	8	0.40
27	R	8	0.40
19	J	6	0.30
0	0	6	0.30
9	9	5	0.25
4	4	5	0.25
6	6	4	0.20
7	7	4	0.20
34	Y	4	0.20
33	X	4	0.20
32	W	4	0.20
31	V	4	0.20
30	U	4	0.20
29	T	4	0.20
28	S	4	0.20
2	2	4	0.20
26	Q	4	0.20
25	P	4	0.20
24	O	4	0.20
21	L	4	0.20
20	K	4	0.20
12	C	4	0.20
18	I	4	0.20
22	M	3	0.15
3	3	3	0.15
15	F	3	0.15
14	E	3	0.15
10	A	3	0.15
17	H	2	0.10
5	5	2	0.10
8	8	2	0.10
11	B	1	0.05
23	N	0	0.00
16	G	0	0.00

Figure 23: Accuracy for all character using Manual Generation Fuzzy System



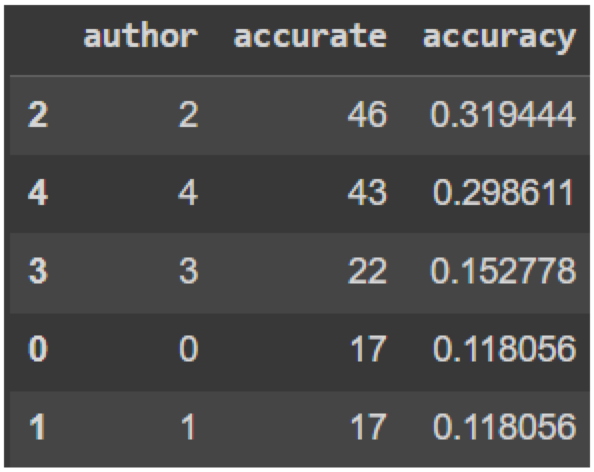
```
[IF peaks '1'[0] AND peaks_2 '1'[0] THEN author[0]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks '1'[4] AND peaks_2 '1'[4] THEN author[1]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks '1'[3] AND peaks_2 '1'[3] THEN author[2]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks '1'[1] AND peaks_2 '1'[1] THEN author[3]
	AND aggregation function : fmin
	OR aggregation function  : fmax, IF peaks '1'[2] AND peaks_2 '1'[2] THEN author[4]
	AND aggregation function : fmin
	OR aggregation function  : fmax]
```

Figure 24: Sample Fuzzy Rule for Character ‘1’ in Manual Generation Fuzzy System



```
[IF peaks 'G'[0] AND peaks_2 'G'[0] THEN author[0]
        AND aggregation function : fmin
        OR aggregation function  : fmax, IF peaks 'G'[1] AND peaks_2 'G'[1] THEN author[1]
        AND aggregation function : fmin
        OR aggregation function  : fmax, IF peaks 'G'[2] AND peaks_2 'G'[2] THEN author[2]
        AND aggregation function : fmin
        OR aggregation function  : fmax, IF peaks 'G'[3] AND peaks_2 'G'[3] THEN author[3]
        AND aggregation function : fmin
        OR aggregation function  : fmax, IF peaks 'G'[4] AND peaks_2 'G'[4] THEN author[4]
        AND aggregation function : fmin
        OR aggregation function  : fmax]
```

Figure 25: Sample Fuzzy Rule for Character 'G' in Manual Generation Fuzzy System



	author	accurate	accuracy
2	2	46	0.319444
4	4	43	0.298611
3	3	22	0.152778
0	0	17	0.118056
1	1	17	0.118056

Figure 26: Accuracy of Fuzzy System for each author in Manual Generation Fuzzy System

Final Prediction

Looking back at our prediction pipeline, our model will make a prediction on one image file by the following steps:

1. Processing the image file into numpy 2d array
2. Recognizing the character by making prediction from best CNN model
3. Predicting the user by using the respective fuzzy rule set based on the predicted character.

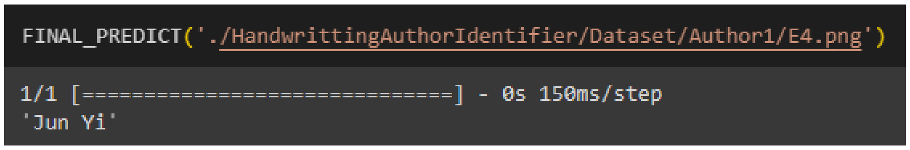
Each step is encapsulated in a function, as shown in the following code snippet:

```
# Image Processing
def process_image(path) -> numpy.ndarray:
    img = Image.open(path)
    img = img.resize(size=(28, 28), resample=Image.ANTIALIAS)
    img_pixel = np.array(img)
    img_pixel = 255 - (img_pixel[:, :, :3] @ [0.299, 0.587, 0.114])
    return img_pixel

# Character Prediction
def neural_network_prediction(image_pixel_arr, use_best_model=True) -> str:
    if use_best_model: model =
tf.keras.models.load_model('./HandwrittingAuthorIdentifier/ModelFile/best_model.h5')
    else: model = tf.keras.models.load_model('not_so_best_model.h5')
    predicted_val = model.predict(image_pixel_arr.reshape(1, 28, 28, 1)).argmax()
    return chr(predicted_val + 55) if predicted_val >= 10 else str(predicted_val)

# Author Prediction
def fuzzy_logic_prediction(image_pixel_arr, char, use_auto_fuzzy_rules=True) -> str:
    image_series = pd.Series(image_pixel_arr.flatten(), index=pixel_columns,
name='brightness')
    gradient_tuple = get_gradient(image_series)
    chosen_gradient_val = gradient_tuple[
        ['gradient_1', 'gradient_2', 'gradient_3'].index(
            log_df[log_df['character'] == char]['selected_gradient'].tolist()[0])]
    char_peaks = log_df[log_df['character'] == char]['peaks'].tolist()[0]
    a, b, c, d, e = sorted_char_peaks = sorted(char_peaks)
    rank_1, rank_2, rank_3, rank_4, rank_5 = [sorted_char_peaks.index(x) for x in
char_peaks]

    # Antecedent
    peaks = ctrl.Antecedent(np.arange(-28, 28.01, 0.01), f'peaks \'{char}\'')
    peaks['0'] = fuzz.trapmf(peaks.universe, [-28, -28, a, b])
    peaks['1'] = fuzz.trimf(peaks.universe, [a, b, c])
    peaks['2'] = fuzz.trimf(peaks.universe, [b, c, d])
    peaks['3'] = fuzz.trimf(peaks.universe, [c, d, e])
    peaks['4'] = fuzz.trapmf(peaks.universe, [d, e, 28, 28])

    # Consequent
    AUTHORS = ['Jun Yi', 'Zheng Qian', 'Kenneth', 'Boon Guan', 'Zi Xiang']
    author = ctrl.Consequent(np.arange(0, 4.1, 0.1), 'author')
    author['0'] = fuzz.trimf(author.universe, [0, 0, 0.5])
    author['1'] = fuzz.trimf(author.universe, [0.5, 1, 1.5])
    author['2'] = fuzz.trimf(author.universe, [1.5, 2, 3.5])
    author['3'] = fuzz.trimf(author.universe, [2.5, 3, 4.5])
    author['4'] = fuzz.trimf(author.universe, [3.5, 4, 4])

    # Rules
    rules = [
        ctrl.Rule(peaks[f'{rank_1}'], author['0']),
        ctrl.Rule(peaks[f'{rank_2}'], author['1']),
        ctrl.Rule(peaks[f'{rank_3}'], author['2']),
        ctrl.Rule(peaks[f'{rank_4}'], author['3']),
        ctrl.Rule(peaks[f'{rank_5}'], author['4']),
    ]
    fuzzy_control_system = ctrl.ControlSystem(rules)
    fuzzy_simulator = ctrl.ControlSystemSimulation(fuzzy_control_system)
    fuzzy_simulator.input[f'peaks \'{char}\''] = chosen_gradient_val
    fuzzy_simulator.compute()

    return AUTHORS[round(fuzzy_simulator.output['author'])]
```

Then, we can lay each function into a full pipeline as described in the prediction flow, where the code implementation is shown as below:

```
def FINAL_PREDICT(image_path, use_best_model=True, use_auto_fuzzy_rules=True,
return_char_pred=False) -> str:

    # Process Image
    image_pixel_arr = process_image(image_path)

    # Identify Character
    char_pred = neural_network_prediction(image_pixel_arr=image_pixel_arr,
use_best_model=use_best_model)

    # Identify Author
    author_pred = fuzzy_logic_prediction(image_pixel_arr=image_pixel_arr,
char=char_pred, use_auto_fuzzy_rules=use_auto_fuzzy_rules)

    if return_char_pred: return author_pred, char_pred
    return author_pred
```

An example of prediction call is shown at the figures below:

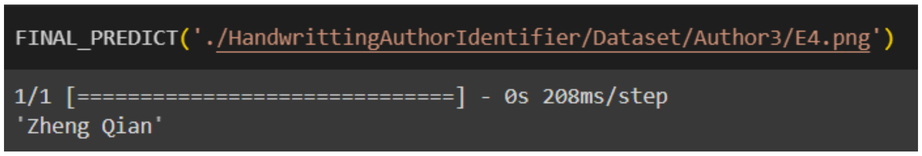


```
FINAL_PREDICT('./HandwrittingAuthorIdentifier/Dataset/Author1/E4.png')
1/1 [==============================] - 0s 150ms/step
'Jun Yi'
```

Figure 27: Prediction example

From the image above, the prediction pipeline predicts that the image is a handwriting of 'Jun Yi', which is actually Author 1 as shown in the file path. This is also an example of correct prediction by the prediction pipeline.

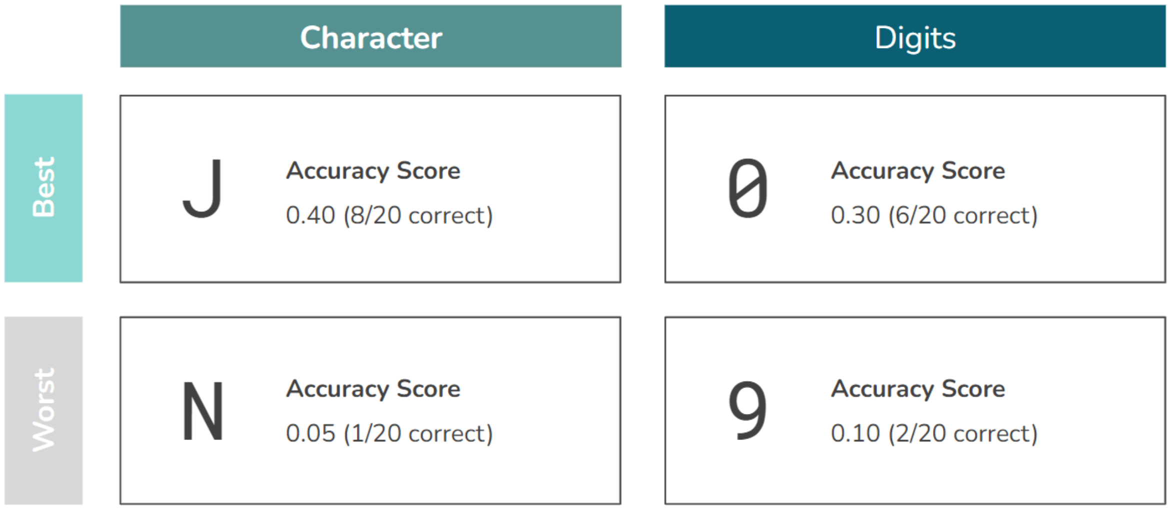
An incorrect prediction will be the one below, where it predicted the handwriting of 'Zheng Qian' but it is actually Author 3, which is actually 'Kenneth'.



```
FINAL_PREDICT('./HandwrittingAuthorIdentifier/Dataset/Author3/E4.png')
1/1 [==============================] - 0s 208ms/step
'Zheng Qian'
```

Figure 28: Incorrect Prediction example

Best/Worst Accuracy



	Character	Digits
Best	J Accuracy Score 0.40 (8/20 correct)	0 Accuracy Score 0.30 (6/20 correct)
Worst	N Accuracy Score 0.05 (1/20 correct)	9 Accuracy Score 0.10 (2/20 correct)

Digit/Alpabet with best accuracy

Digit : 0 **Accuracy** : 0.3(6/20)

Alphabet : J **Accuracy** : 0.4(8/20)

Digit/Alphabet with worst accuracy

Digit : 9 **Accuracy** : 0.1(2/20)

Alphabet : N **Accuracy** : 0.05(1/20)

Author with best accuracy

Author : Teck Hou **Accuracy** : 0.597222 (86/144 correct)

Author with worst accuracy

Author : Boon Guan **Accuracy** : 0.027778 (4/144 correct)

Metrics Used

In this assignment, overall, the metric that we used to evaluate the performance of the CNN and Fuzzy Logic models is accuracy. The table below shows the usage and benefits of using this metrics according to Purva Huilgol [7].

Accuracy Score
Easy to interpret. Since it is just the accuracy of each prediction.
Used when the True Positives and True negatives are more important
Used when the class distribution is similar
$\frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$

(Purva Huilgol, 2019)

The reason that we choose accuracy over other metrics is because:

Simplicity

Accuracy is easy to understand and interpret. It measures the proportion of correctly classified instances out of the total instances. Since our problem is a straightforward classification task, in which to classify which image belongs to which character, and which character belongs to which author, accuracy can provide a clear indication of the model's performance.

Balanced Dataset

Due to our dataset is actually fixed, with 5 author, and each author is contributing to 36 characters, ranging from 0 - 9 and A - B, and all of the author contribute 4 copy of same characters for the dataset.. In such cases, accuracy gives a fair representation of the model's ability to classify correctly across all classes.

Conclusion

In conclusion, we managed to combine the usage of fuzzy system, neural network and genetic algorithm to build a system that can classify images into character 0 - 9 and A - Z, and then further determine which author does the character handwriting style belongs to.

In our two-step model, the character recognition actually achieve a very high accuracy of 99%, meanwhile the fuzzy system that used to identify which author does the character belongs to only achieve 23% accuracy. This is actually quite unavoidable, because every author has their own style of writing, or some are having the same writing style, this might cause confusion to the system as their input to the fuzzy system might be similar. Even us as humans also cannot accurately identify the characters actually belong to which author too, just by looking at one character.

Furthermore, in forensic handwriting analysis, an analyst will use a **magnifying glass** and sometimes even a **microscope** in the comparison process. When looking at writing samples, an analyst is looking for a wide array of individual traits including letter form, line form and word spacing [9], meaning that by looking at one singular character to identify who wrote it still poses a significant challenge.

To improve the accuracy of our current system, we propose to use multiple input images from the same user before performing the identification. This approach should enhance the reliability of the author identification process.

References

[1] Rendyk. (2024, May 27). Tuning the hyperparameters and layers of neural network deep learning. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/

[2] geneticalgorithm. (2020, December 27). PyPI. https://pypi.org/project/geneticalgorithm/

[3] Intuition behind Stacking Multiple Conv2D Layers before Dropout in CNN. (n.d.). Stack Overflow. https://stackoverflow.com/questions/46515248/intuition-behind-stacking-multiple-conv2d-layers-before-dropout-in-cnn

[4] In neural networks, why conventionally set number of neurons to 2^n? (n.d.). Stack Overflow. https://stackoverflow.com/questions/63515846/in-neural-networks-why-conventionally-set-number-of-neurons-to-2n

[5] GeeksforGeeks. (2023, December 21). KDE Plot Visualization with Pandas and Seaborn. GeeksforGeeks. https://www.geeksforgeeks.org/kde-plot-visualization-with-pandas-and-seaborn/

[6] Genetic Algorithms - parent selection. (n.d.). https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm

[7] Huilgol, P. (2024, January 15). Accuracy vs. F1-Score - Analytics Vidhya - Medium. Medium. https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2

[8] Pratt, M. (2022, June 6). Is your handwriting always different? Here’s why! ArtofScribing.com. https://artofscribing.com/handwriting/is-your-handwriting-always-different-heres-why/

[9] Layton, J. (2023, October 23). How handwriting analysis works. HowStuffWorks. https://science.howstuffworks.com/handwriting-analysis.htm