

Slovak University of Technology in Bratislava

Faculty of Informatics and Information Technologies

Artificial Intelligence **Assignment 3**

Classification

Implementation of a Neural network for classifying points in 2D space into predefined classes (colours) based on their coordinates.

Table of Content

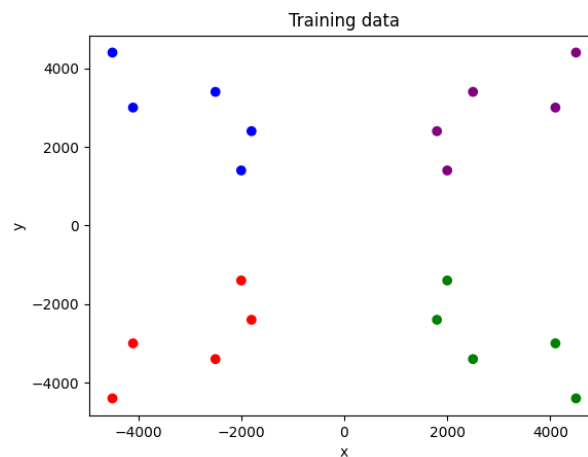
The assignment	3
Implementation	4
Description of the dataset, and its processing.....	4
Choice of the selected ratio for testing and training data	4
Experimenting with network architecture	5
Evaluation of the pros and cons of the used network architecture	7
Visualization of the results	8
User Manual.....	10

The assignment

The goal of this assignment is to design and implement a neural network capable of classifying points within a 2D space into four predefined categories (red, green, blue, and purple) based on their X and Y coordinates.

The neural network should learn the underlying patterns from an initial set of 20 points (5 for each class) and subsequently classify new points accurately.

R: [-4500, -4400], [-4100, -3000], [-1800, -2400], [-2500, -3400] a [-2000, -1400]
 G: [+4500, -4400], [+4100, -3000], [+1800, -2400], [+2500, -3400] a [+2000, -1400]
 B: [-4500, +4400], [-4100, +3000], [-1800, +2400], [-2500, +3400] a [-2000, +1400]
 P: [+4500, +4400], [+4100, +3000], [+1800, +2400], [+2500, +3400] a [+2000, +1400]



After training on this dataset, the neural network will be tested on its ability to accurately classify new points in the 2D space. The success of the neural network will be evaluated by comparing its predictions for the class of each point against the known classes of the 40,000 generated points. This evaluation will provide insights into the effectiveness of the neural network in generalizing from the training data and classifying unseen instances in the 2D space.

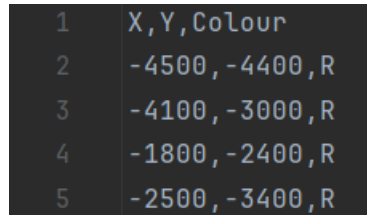
For each experiment conducted, it is essential to create visualizations of the resulting 2D surface. This visualization involves colouring the entire area based on the classifications made by the neural network. By visually representing the coloured regions, one can gain insights into how well the neural network has learned and generalized the patterns within the 2D space.

Documentation is a key aspect of this assignment, requiring a comprehensive description of the specific algorithm and data representation employed in the neural network. Documentation should provide a clear and detailed explanation of the chosen neural network architecture, the training process, and any optimizations or techniques utilized. Additionally, it should include insights into the data representation methods employed for both the initial set of points and the newly generated dataset.

Implementation

Description of the dataset, and its processing

The employed dataset is derived from the parameters outlined in the assignment. The 20 points, each characterized by its coordinates, were transcribed into a ".csv" file. Figure 1. illustrates the format of the file, comprising three columns: one for the X coordinate, another for the Y coordinate, and a third for the point's colour.



1	X,Y,Colour
2	-4500,-4400,R
3	-4100,-3000,R
4	-1800,-2400,R
5	-2500,-3400,R

Figure 1 Dataset visualization

To handle the dataset information, the pandas library is imported and utilized. Within the `load_data()` function, the subsequent line is employed to read data from the dataset into the `data_frame` variable.

```
data_frame = pd.read_csv(file)
```

To enhance data manipulation and facilitate numerical representation in mathematical models, the colours in the dataset are converted into numerical values.

```
class_mapping = {'R': 0, 'G': 1, 'B': 2, 'P': 3}
data_frame['Colour'] = data_frame['Colour'].map(class_mapping)
```

Subsequently, the data is divided into inputs and outputs. All columns, excluding the last one, serve as inputs, while the last column represents the outcome. Following the correct partitioning of the data, it is further divided into training and testing sets utilizing the `train_test_split()` function from the TensorFlow library. The `test_size` parameter can be configured in the `config.py` file.

```
X = data_frame[data_frame.columns[:-1]].values # Inputs
Y = data_frame[data_frame.columns[-1]].values # Outputs

# Train and Test
X, Y, XT, YT = train_test_split(X, Y, test_size=config.TEST_SIZE, random_state=0)
```

Choice of the selected ratio for testing and training data

Given that the assignment dataset comprises only 20 points, it is essential to exercise caution when setting the `test_size`. For instance, if `test_size` is configured to 50%, it would mean that only 10 points remain for training the model. It is noteworthy that the default value is set to 10%. Careful consideration of the `test_size` parameter is crucial to ensure a meaningful and representative training of the model.

```
test_size=0.2
1250/1250 [=====] - 2s 1ms/step - loss: 8.8378 - accuracy: 0.9276
```

```
test_size=0.3
1250/1250 [=====] - 2s 1ms/step - loss: 172.368 - accuracy: 0.7476
```

```
test_size=0.5
1250/1250 [=====] - 2s 1ms/step - loss: 41.1994 - accuracy: 0.8723
```

```
test_size=0.9
1250/1250 [=====] - 2s 1ms/step - loss: 260.896 - accuracy: 0.5855
```

These results highlight the impact of varying `test_size` on the model's performance, emphasizing the need for thoughtful parameter tuning to achieve optimal outcomes.

Experimenting with network architecture

During the exploration for the optimal neural network model, a series of experiments were conducted to assess various models. Presented below are examples along with their respective effectiveness.

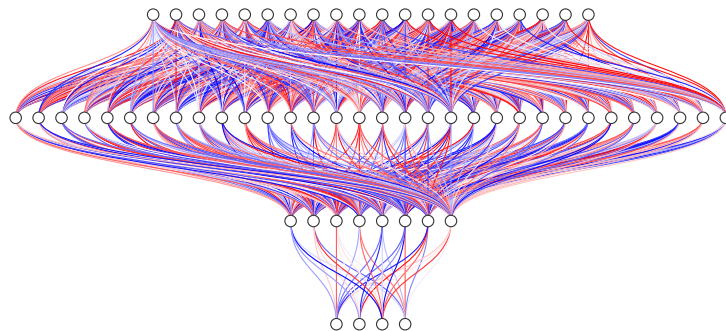


Figure 2 Neural network of Architecture 1 and Architecture 2

The comparison between Architecture 1 and Architecture 2 suggests that employing the relu activation function in hidden layers yields greater effectiveness.

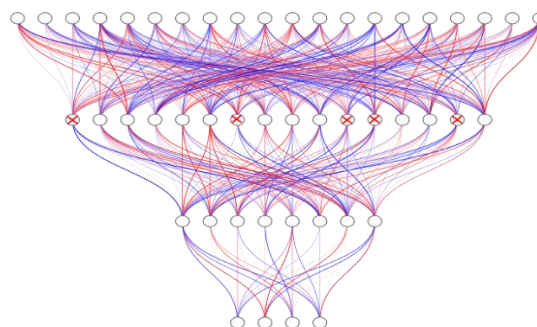
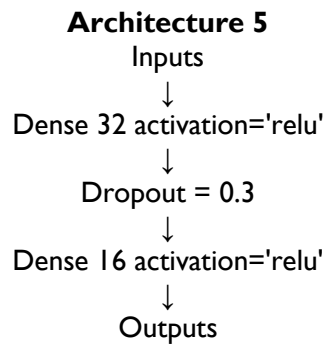
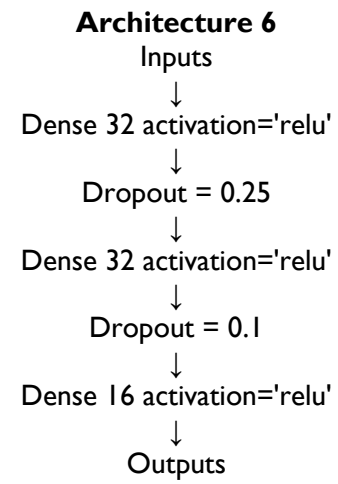


Figure 3 Neural network of Architecture 3 and Architecture 4

The inclusion of dropout in the hidden layers significantly augmented the accuracy of the neural network.



loss: 20.7130 - accuracy: 0.8938



loss: 9.2251 - accuracy: 0.9265

Architecture 6 is designated as the default neural network architecture utilized in the program. Users can modify the architecture by adjusting the configuration in the config.py file.

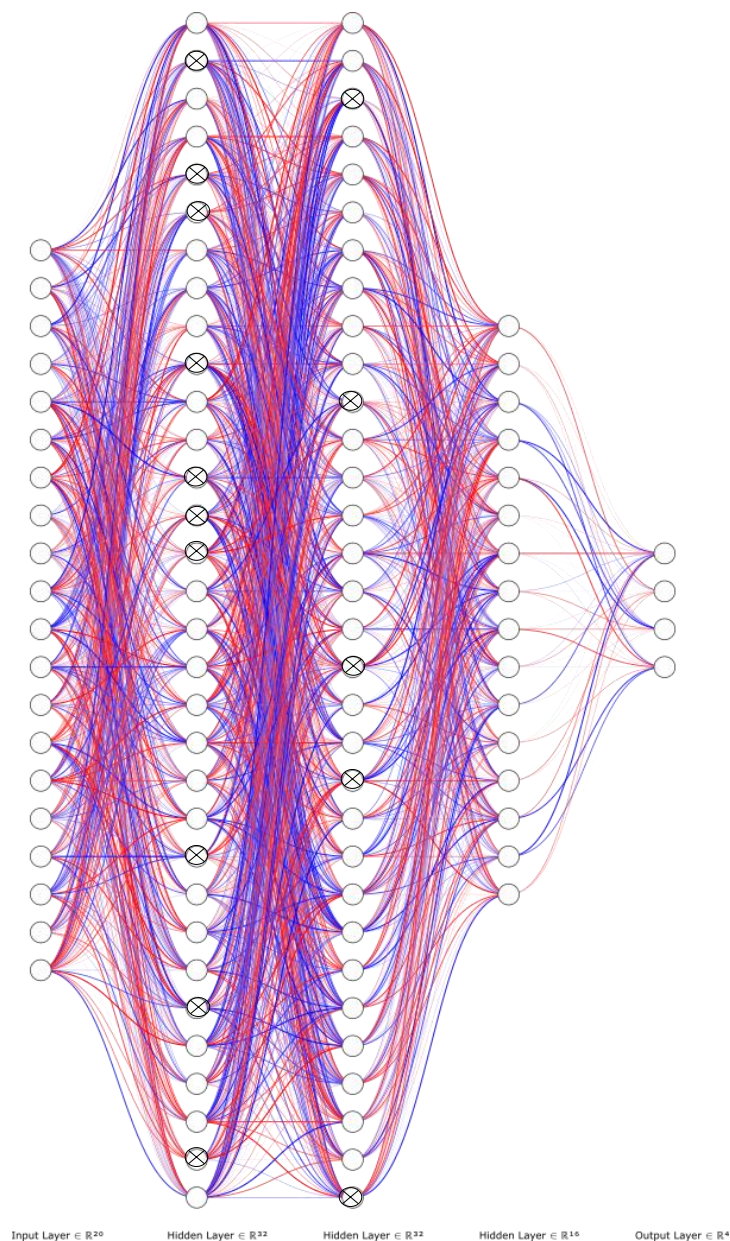


Figure 4 Architecture 6 - This architecture is used in program.

Evaluation of the pros and cons of the used network architecture

This section of the documentation will address the advantages and disadvantages of the employed neural network.

Advantages of used architecture

Sequential Architecture: Using a sequential model makes it easy to understand and implement neural networks.

Three hidden layers: The model incorporates three hidden layers, allowing it to learn hierarchical representations of the data

Activation functions: ReLU activation functions are commonly used and can facilitate faster convergence during training.

Dropout layers: Dropout layers are implemented to prevent overfitting by randomly dropping out nodes during training.

Softmax function in output layer: Softmax activation in the output layer is suitable for multiclass classification problems.

Adam optimizer: The use of the Adam optimizer is advantageous, as it adapts the learning rate during training.

Disadvantages of used architecture

Small dataset: The dataset provided is quite small. Neural networks often require larger datasets for effective training.

Imbalanced classes: The dataset has 5 points of each colour, 10% (2 points) is used to test, which mean that in training process there are no 5 points of each colour. This might lead to biased predictions.

Fixed learning rate: The learning rate is fixed (config.MODEL_LEARNING_RATE). Experimenting with different learning rates might help optimize training.

Fixed dropout rates: The dropout rates are fixed and not adaptive. Experimenting with different dropout rates or using techniques like dropout annealing might improve performance.

No early stopping: Incorporating early stopping based on validation performance could prevent overfitting and improve generalization.

Visualization of the results

To visualize the output – how program placed 40,000 generated points, the library matplotlib is used. Results of mentioned architectures above:

Architecture 1

Generated data

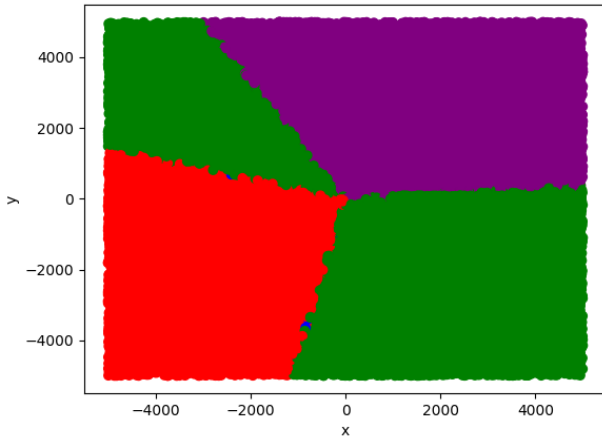


Figure 5 Classified points using Architecture 1

loss: 1.3061 - accuracy: 0.4975

Architecture 2

Generated data

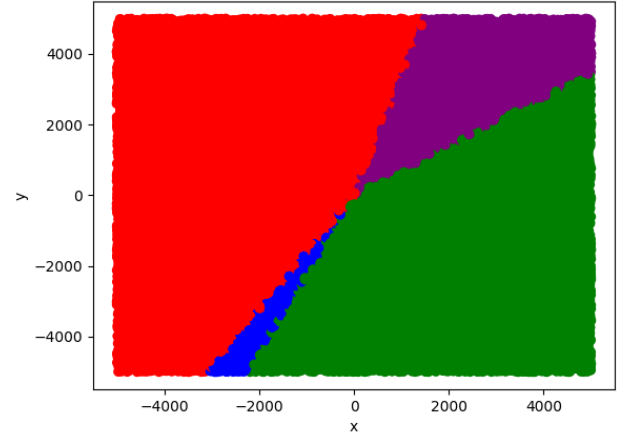


Figure 6 Classified points using Architecture 2

loss: 110.1937 - accuracy: 0.5425

Architecture 3

Generated data

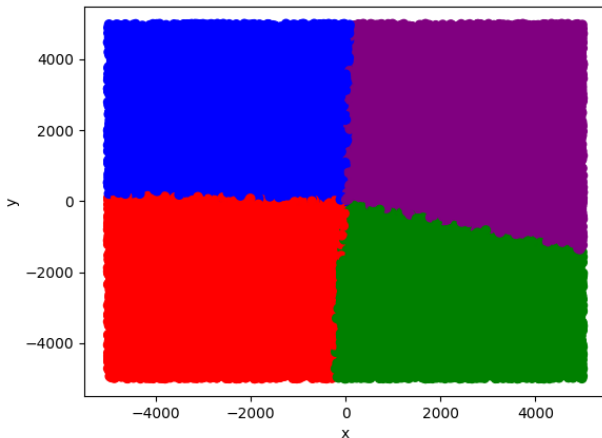


Figure 7 Classified points using Architecture 3

loss: 14.7738 - accuracy: 0.8690

Architecture 4

Generated data

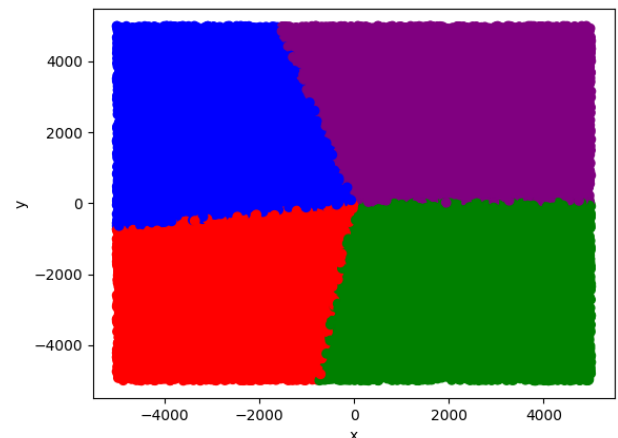


Figure 8 Classified points using Architecture 4

loss: 34.7865 - accuracy: 0.8386

Architecture 6 – Default Architecture

Generated data

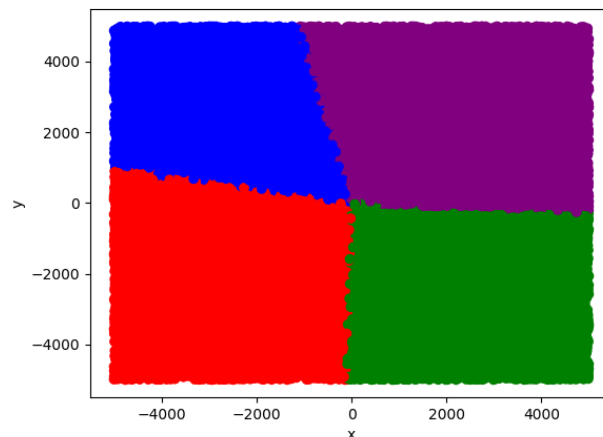


Figure 9 Classified points using Architecture 6

loss: 7.4989 - accuracy: 0.9185

Also, visualization of increasing accuracy and decreasing loss in each epoch is important to compare. There are some graphs that show how accuracy and loss changed over each epoch in training with dataset.

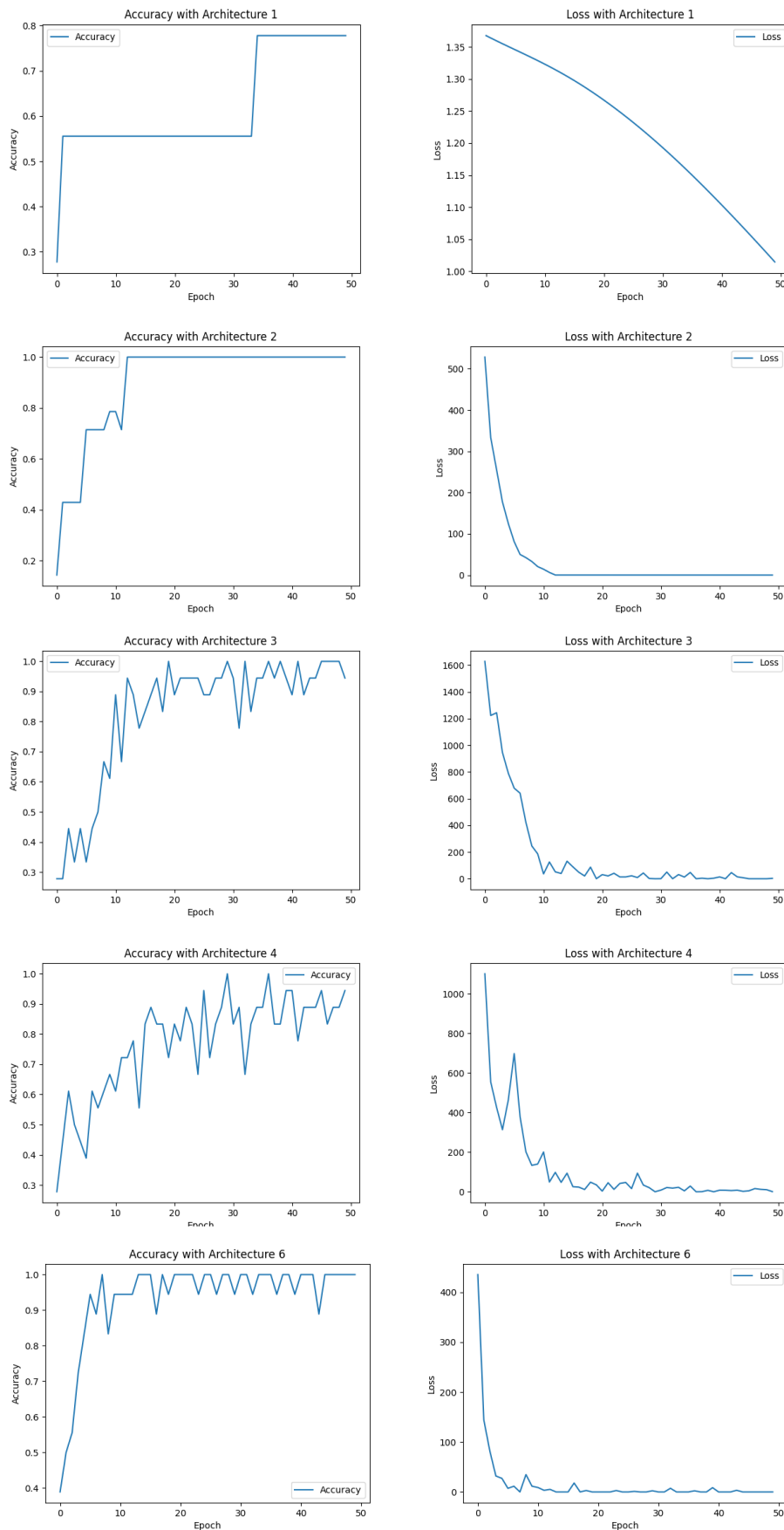


Figure 10 Loss and Accuracies with different architectures

User Manual

This section is intended to guide users in achieving their intended outcomes while using the program. Upon running the program, users are presented with two options:

- Enter 'y' to configure default settings stored in config.py.
- Enter 'n' to run the program with configurations from config.py.

```
David Truhlar - 120897 - Artificial Intelligence - Assignment 3
Neural Network for classification of points in 2D space
Do you want to configure the model? If no model uses default configuration (y/n):
```

Figure 11 Menu displayed in console

If a user wishes to modify certain parameters of the program, several options are available:

```
Which parameters do you want to change?
1. Dataset path
2. Number of points in each colour
3. Test size
4. Learning rate
5. Number of Epochs and Batch size
Enter the number of parameter you want to change: 3
Enter the test size (0.0 - 1.0): 0.1
```

Figure 12 Example - changing Test size

If the user enters 'n' and presses enter, the program utilizes parameters from config.py and commences execution.

Output:

```
Epoch 1/50
1/1 [=====] - 1s 881ms/step - loss: 542.6761 - accuracy: 0.0556 - val_loss: 294.7452 - val_accuracy: 0.0e+00
Epoch 2/50
1/1 [=====] - 0s 24ms/step - loss: 214.2739 - accuracy: 0.3333 - val_loss: 68.0284 - val_accuracy: 0.5000
Epoch 3/50
1/1 [=====] - 0s 25ms/step - loss: 155.4648 - accuracy: 0.3333 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
. . .
. . .
1/1 [=====] - 0s 96ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.00e+00 - val_accuracy: 1.0000
Epoch 49/50
1/1 [=====] - 0s 27ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.00e+00 - val_accuracy: 1.0000
Epoch 50/50
1/1 [=====] - 0s 26ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.00e+00 - val_accuracy: 1.0000
1250/1250 [=====] - 1s 862us/step - loss: 19.3608 - accuracy: 0.8889
```

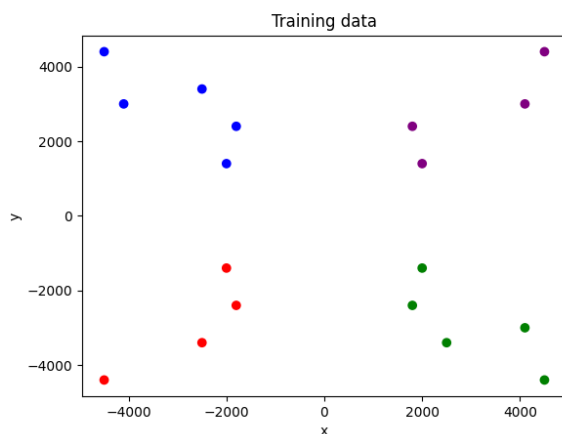


Figure 13 Picture of plotted training data

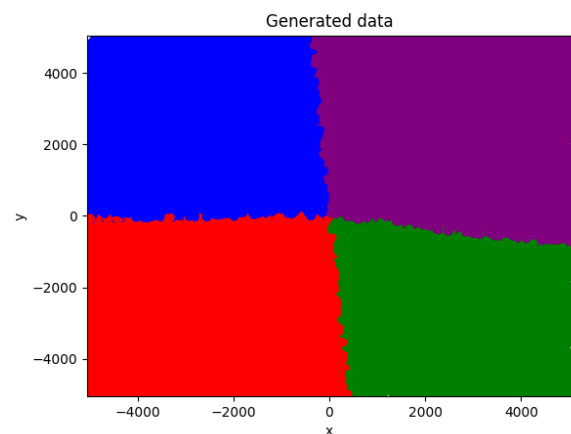


Figure 14 Picture of plotted generated data