Abstract and Reasoning Challenge: Milestone Report 1

John Bello

Introduction

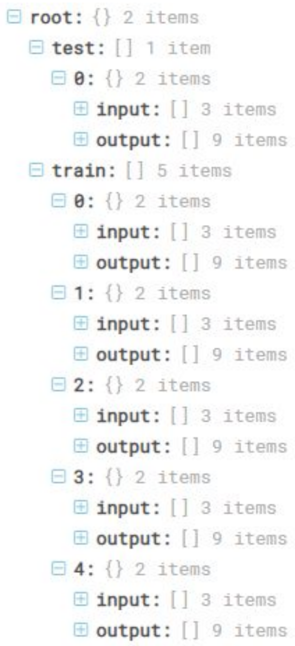
The purpose of this project is to create a model capable of solving abstract tasks as defined in the Abstract and Reasoning Challenge posted by François Chollet on Kaggle.com.

This model will be given a task consisting of a set of two to ten training samples, in the form of a grid, and asked to learn a set of abstract rules. The model will then be given a test input grid and asked to create the expected output pattern. The model will be trained on 400 tasks and evaluated on 100 new tasks.

The thought process behind this competition is that, by creating a model of learning abstract patterns given very little training data, machine learning models would be better able to mimic human thought. This ability to mimic human thought would dramatically improve the range of tasks machine learning models are capable of as well as the speed at which they learn these new tasks.

Data Structure

The data is structured within three folders, a training folder, an evaluation folder, and a test folder. For the sake of this project, the test folder will be omitted. Within the training and evaluation folders are JSON files, 400 in the training folder and 100 in the evaluation folder. The structure of the JSON files are the same in each folder, shown below in Figure 1.



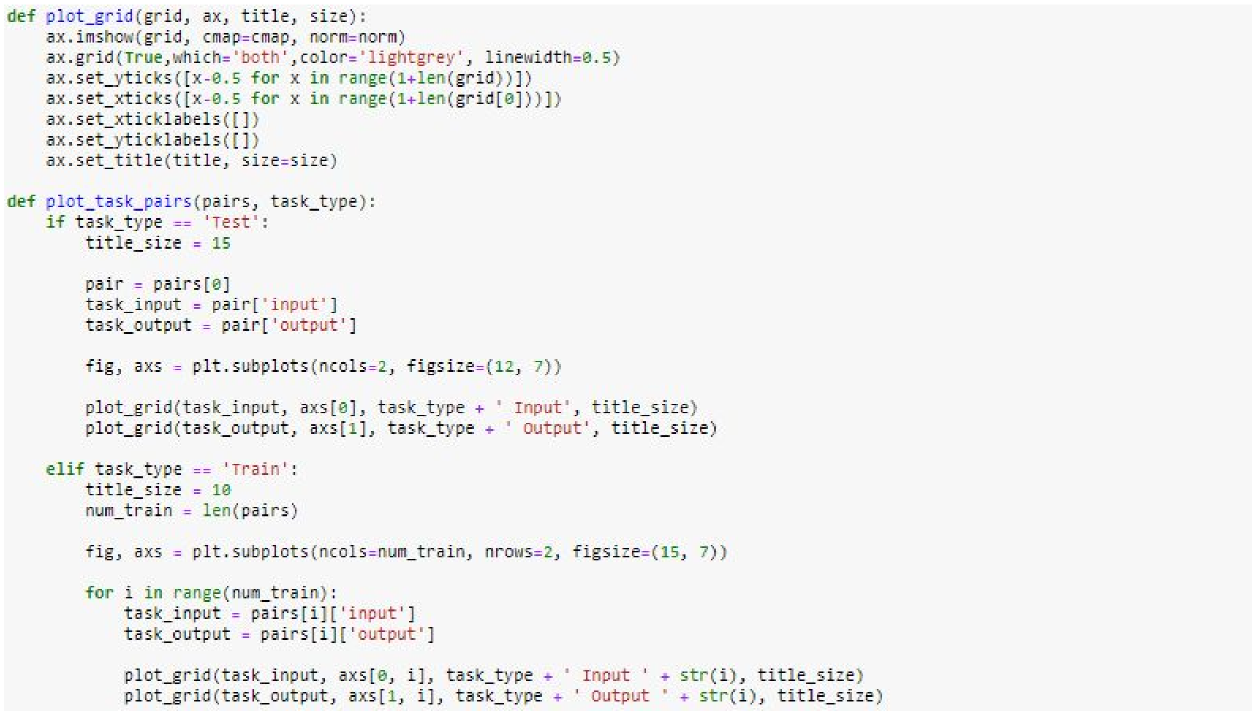
```
⊟ root: {} 2 items
  ⊟ test: [] 1 item
    ⊟ 0: {} 2 items
      ⊞ input: [] 3 items
      ⊞ output: [] 9 items
  ⊟ train: [] 5 items
    ⊟ 0: {} 2 items
      ⊞ input: [] 3 items
      ⊞ output: [] 9 items
    ⊟ 1: {} 2 items
      ⊞ input: [] 3 items
      ⊞ output: [] 9 items
    ⊟ 2: {} 2 items
      ⊞ input: [] 3 items
      ⊞ output: [] 9 items
    ⊟ 3: {} 2 items
      ⊞ input: [] 3 items
      ⊞ output: [] 9 items
    ⊟ 4: {} 2 items
      ⊞ input: [] 3 items
      ⊞ output: [] 9 items
```

Figure 1. JSON structure.

Each file is separated into a train and test set. The train set has a varying number of input and output pairs while there is always only a single training input in the test set. The dimensions of the inputs and outputs vary between each task (JSON file).

Data Wrangling

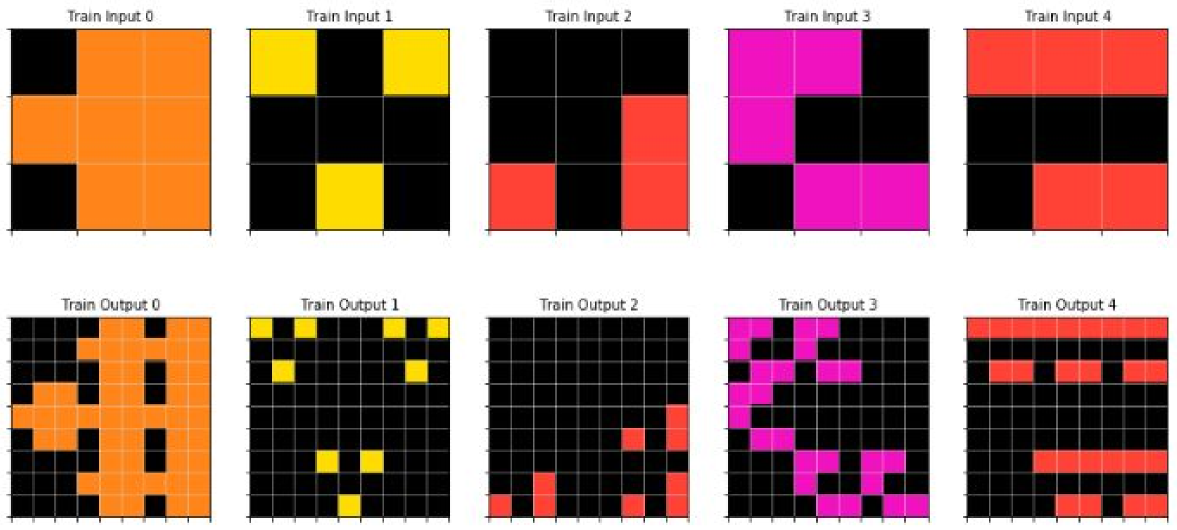
Because the data is taken from a Kaggle competition, the data is already well formatted. During this step, what was done was the data was read in, and plotted using the code below.



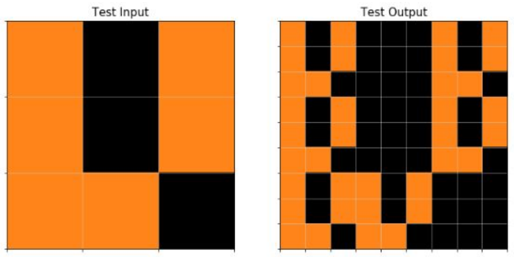
```python
def plot_grid(grid, ax, title, size):
    ax.imshow(grid, cmap=cmap, norm=norm)
    ax.grid(True,which='both',color='lightgrey', linewidth=0.5)
    ax.set_yticks([x-0.5 for x in range(1+len(grid))])
    ax.set_xticks([x-0.5 for x in range(1+len(grid[0]))])
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.set_title(title, size=size)

def plot_task_pairs(pairs, task_type):
    if task_type == 'Test':
        title_size = 15

        pair = pairs[0]
        task_input = pair['input']
        task_output = pair['output']

        fig, axs = plt.subplots(ncols=2, figsize=(12, 7))

        plot_grid(task_input, axs[0], task_type + ' Input', title_size)
        plot_grid(task_output, axs[1], task_type + ' Output', title_size)

    elif task_type == 'Train':
        title_size = 10
        num_train = len(pairs)

        fig, axs = plt.subplots(ncols=num_train, nrows=2, figsize=(15, 7))

        for i in range(num_train):
            task_input = pairs[i]['input']
            task_output = pairs[i]['output']

            plot_grid(task_input, axs[0, i], task_type + ' Input ' + str(i), title_size)
            plot_grid(task_output, axs[1, i], task_type + ' Output ' + str(i), title_size)
```

Figure 2. Plotting Functions

The `plot_task_pairs()` function takes either the training set or testing set from a single JSON file and plots the grids according to the colors shown in the app used to visualize the data. This app is located at the github page of the competition(https://github.com/fchollet/ARC). The output for the first JSON file is shown below in Figure 3.



(a)



(b)

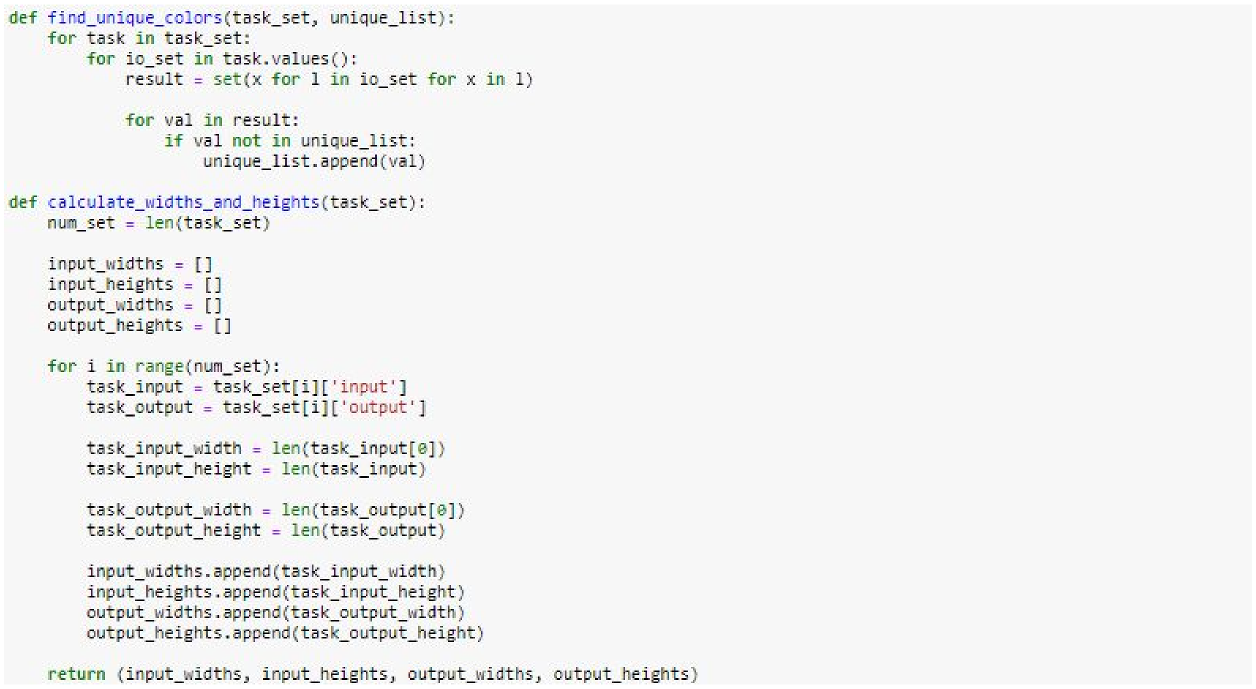
Figure 3. File: 007bbfb7.json

(a) Visualizations of training set. (b) Visualizations of testing set.

Additionally, during this step, a data frame was created containing summary statistics for each of the tasks in the training set. This data frame contains the following columns:

1. **task_label** : The JSON file identifier.
2. **num_train** : The number of input-output pairs in the training set of the specific JSON file.
3. **train_input_widths** : A list containing the widths of each input in the train set.
4. **train_input_heights** : A list containing the heights of each input in the train set.
5. **train_output_widths** : A list containing the widths of each output in the train set.
6. **train_output_heights** : A list containing the heights of each output in the train set.
7. **test_input_widths** : A list containing the widths of each input in the test set.
8. **test_input_heights** : A list containing the heights of each input in the test set.
9. **test_output_widths** : A list containing the widths of each output in the test set.
10. **test_output_heights** : A list containing the heights of each output in the test set.
11. **unique_colors** : A list containing the unique colors occuring in the train set.

The functions used to create this dataframe are shown in Figure 4 below.



```
def find_unique_colors(task_set, unique_list):
    for task in task_set:
        for io_set in task.values():
            result = set(x for l in io_set for x in l)

            for val in result:
                if val not in unique_list:
                    unique_list.append(val)

def calculate_widths_and_heights(task_set):
    num_set = len(task_set)

    input_widths = []
    input_heights = []
    output_widths = []
    output_heights = []

    for i in range(num_set):
        task_input = task_set[i]['input']
        task_output = task_set[i]['output']

        task_input_width = len(task_input[0])
        task_input_height = len(task_input)

        task_output_width = len(task_output[0])
        task_output_height = len(task_output)

        input_widths.append(task_input_width)
        input_heights.append(task_input_height)
        output_widths.append(task_output_width)
        output_heights.append(task_output_height)

    return (input_widths, input_heights, output_widths, output_heights)
```

Figure 4. functions involved in creating summary data frame

Data Storytelling

In order to gain a better understanding of the data contained in the training set, the summary data frame created in the previous section of the project was used to create visualization. First, the sizes of the training set of each JSON file was counted. This plot is shown in Figure 5.

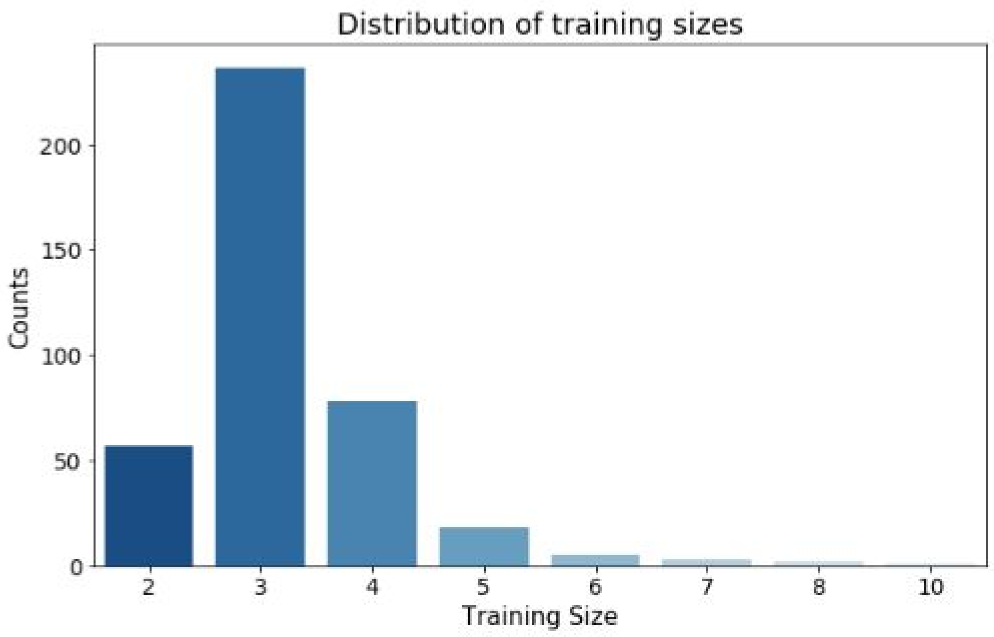


Figure 5. Distribution of training sizes.

Clearly, the majority of the tasks contain three training samples. The model must be able to accept a varying number of training samples per task. Next, the frequency of each color per training size was plotted. An example is shown below for tasks containing three training samples.

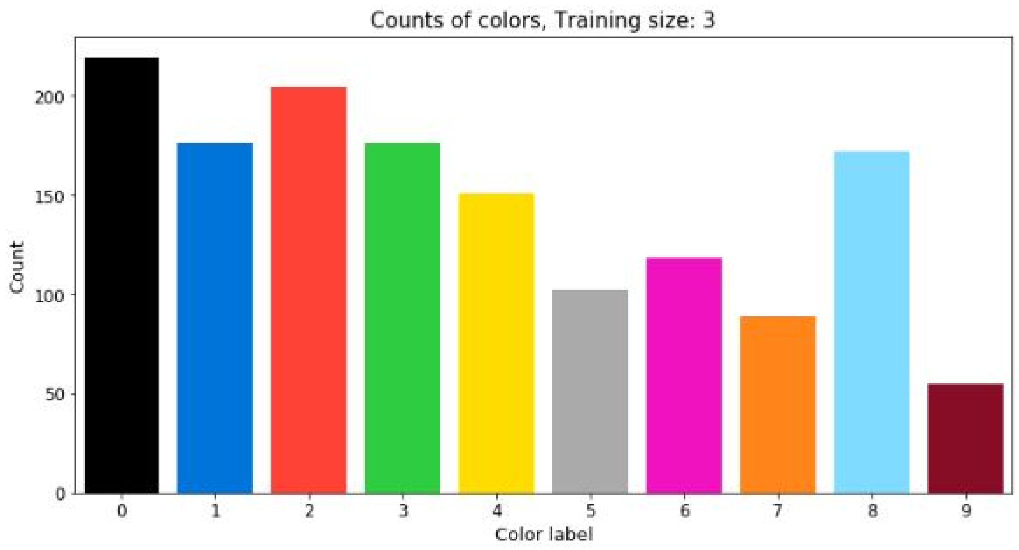


Figure 6. Plot showing how frequently each color occurs within tasks with three training samples.

From this plot we can see that colors occuring most frequently in tasks with three training samples are black, blue, red, green, and light blue. The other tasks can be analyzed similarly using their respective plots. Finally, the dimensions of the inputs and outputs were plotted to see if there were any trends between them. These are displayed below in Figure 7. Regarding the widths versus the heights of the inputs and outputs, they primarily followed a linear trend. Additionally, there were clearly many cases where the dimensions of the input matched the dimensions of the output, however, for the cases they dimensions did not match, the input dimension tended to be larger than the output dimension. This suggests that many of the transformations within the tasks were compressive in nature.

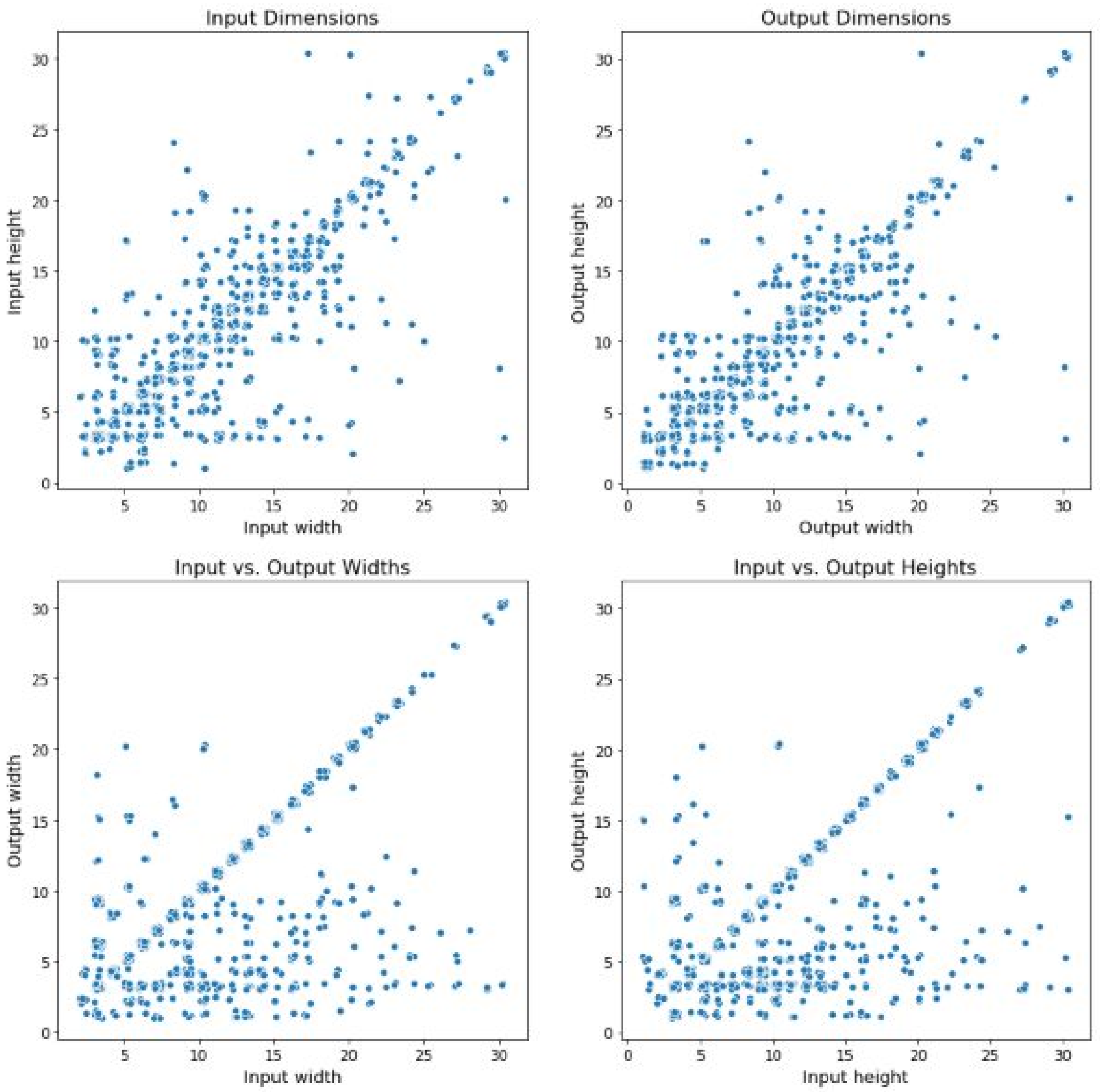


Figure 7. Dimension relationships.

Statistical Data Analysis

In order to verify many of the observations seen in the visual analysis, statistical data analysis was done. First, the correlation between colors and training sizes was plotted. This can be seen in Figure 8 below. From the correlation plot, we cannot see a clear relationship between the occurrence of the different colors and the number of samples in the training set of each file.

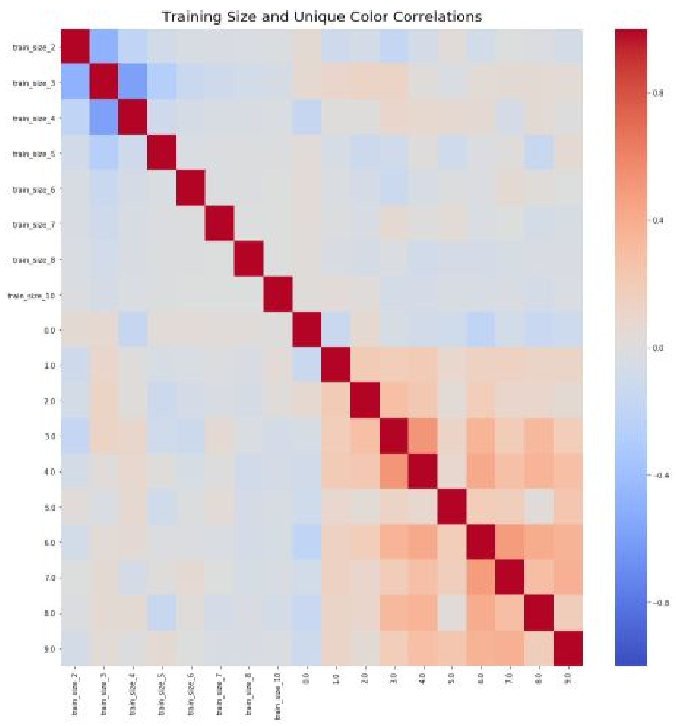


Figure 8. Correlation plot between training size and colors.

Next, the input dimensions were inspected using bootstrapping. For this section, only the largest groups were analyzed (training size = 2, 3, 4). An example of the output is shown below:

```
perform_bootstrap_analysis(col, 2, 4)
```

```
p-value: 0.0413
Observed mean difference:  1.1606
```

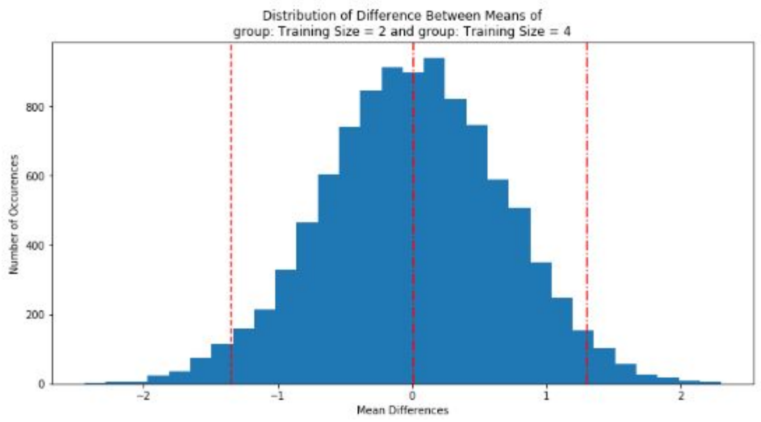


Figure 9. Bootstrapping output example.

This example analyzes the mean of the input widths of the tasks with training samples of size two and four. From the example, we can see that there is a significant difference between the means of each group's input widths, namely that tasks with two training samples tend to have larger widths than tasks with four training samples. A summary of the findings are shown in the table below:

Table I. Bootstrap findings summary table.

Group 1	Group 2	Feature	P-value	Observed mean difference	Reject null hypothesis?
2	3	train_input_width	0.8710	-0.4953	No
		train_input_height	0.9741	-0.8489	No
		train_output_width	0.0780	0.5856	No
		train_output_height	0.2092	0.3171	No
2	4	train_input_width	0.0413	1.1606	Yes
		train_input_height	0.0529	1.0353	No
		train_output_width	0.0016	1.8789	Yes
		train_output_height	0.0002	1.9909	Yes
3	4	train_input_width	0.0001	1.6559	Yes
		train_input_height	0.0000	1.8841	Yes
		train_output_width	0.0027	1.2932	Yes
		train_output_height	0.0003	1.6738	Yes