# Abstract and Reasoning Challenge: Milestone Report 2

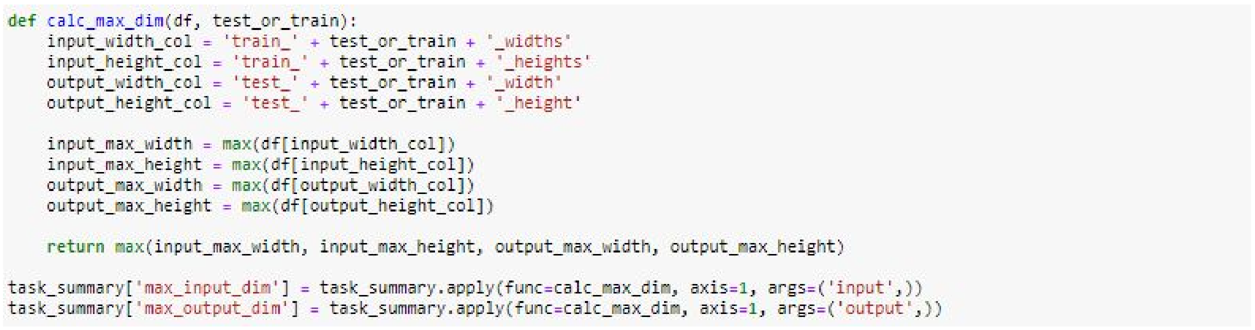
John Bello

There are three major challenges regarding this set of data:

1. The tasks have varying input and output sizes
2. There are very few training samples per task
3. It is difficult to generate new samples(to increase the size of the training set of each task) due to the abstract rules governing each set of training pair

## Preprocessing

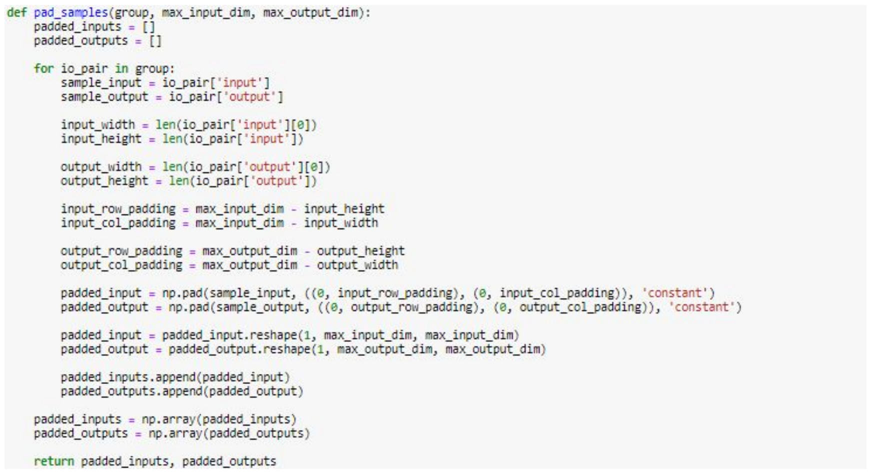
In order to tackle the first challenge, the data will be zero-padded into square grids based on the largest input and output dimensions. For example, if the largest input dimension in a task is 15x20 and the largest output dimension is 30x20, the inputs will be zero-padded to a 20x20 grid, while the outputs will be padded into a 30x30 grid. This allows the same model to tackle the varying input sizes present in that particular task. This was accomplished by first determining the largest width and heights within each task. The code used to calculate these values is shown below.



```
def calc_max_dim(df, test_or_train):
    input_width_col = 'train_' + test_or_train + '_widths'
    input_height_col = 'train_' + test_or_train + '_heights'
    output_width_col = 'test_' + test_or_train + '_width'
    output_height_col = 'test_' + test_or_train + '_height'

    input_max_width = max(df[input_width_col])
    input_max_height = max(df[input_height_col])
    output_max_width = max(df[output_width_col])
    output_max_height = max(df[output_height_col])

    return max(input_max_width, input_max_height, output_max_width, output_max_height)

task_summary['max_input_dim'] = task_summary.apply(func=calc_max_dim, axis=1, args=('input',))
task_summary['max_output_dim'] = task_summary.apply(func=calc_max_dim, axis=1, args=('output',))
```

Figure 1. Code used to calculate maximum dimensions.

Next, the training data had to be padded in according to these maximum dimensions. This was done using the function shown in Figure 2. Using numpy's `.pad` function, zeros were added to the right and bottom of the arrays.



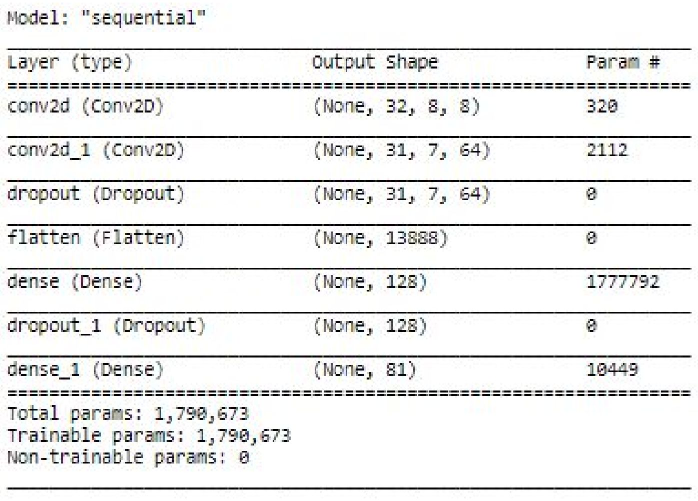
```
def pad_samples(group, max_input_dim, max_output_dim):
    padded_inputs = []
    padded_outputs = []

    for io_pair in group:
        sample_input = io_pair['input']
        sample_output = io_pair['output']

        input_width = len(io_pair['input'][0])
        input_height = len(io_pair['input'])

        output_width = len(io_pair['output'][0])
        output_height = len(io_pair['output'])

        input_row_padding = max_input_dim - input_height
        input_col_padding = max_input_dim - input_width

        output_row_padding = max_output_dim - output_height
        output_col_padding = max_output_dim - output_width

        padded_input = np.pad(sample_input, ((0, input_row_padding), (0, input_col_padding)), 'constant')
        padded_output = np.pad(sample_output, ((0, output_row_padding), (0, output_col_padding)), 'constant')

        padded_input = padded_input.reshape(1, max_input_dim, max_input_dim)
        padded_output = padded_output.reshape(1, max_output_dim, max_output_dim)

        padded_inputs.append(padded_input)
        padded_outputs.append(padded_output)

    padded_inputs = np.array(padded_inputs)
    padded_outputs = np.array(padded_outputs)

    return padded_inputs, padded_outputs
```

Figure 2. Code used to pad input and output arrays

## CNN Model Building

Regarding the second challenge, two approaches were taken. First, from the visual analysis, it was observed that the outputs are related to the inputs either by a transformation of the input data or by a set of spatial rules. Two models with the potential to capture these abstract rules are convolution neural networks(CNNs) and variational autoencoders(VAEs). Convolutional neural networks are exceptional at modeling the features of a given dataset, and due to the grid-like nature of this data, they can be thought of as images. On the other hand, VAEs are able to learn a latent variable model for its input data, or in other words, VAE's have the potential to learn the rules that created the input and output pairs.

In order to implement the convolutional neural network, Tensorflow's keras api was used to describe the architecture of the model. A example summary of the model architecture is shown below in Figure 3.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 8, 8)          320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 31, 7, 64)         2112
_________________________________________________________________
dropout (Dropout)            (None, 31, 7, 64)         0
_________________________________________________________________
flatten (Flatten)            (None, 13888)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1777792
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 81)                10449
=================================================================
Total params: 1,790,673
Trainable params: 1,790,673
Non-trainable params: 0
_________________________________________________________________
```

Figure 3. Example of CNN architecture

The shapes of the input layer, second convolutional layer, and output layer all varied due to the differing sizes of each of these tasks. Additionally, the dimension of the output was determined by the size of the flattened output array. This array was later reconstructed into the shape of the original output grid and then plotted. An example of a prediction is shown below in FIgure 4.

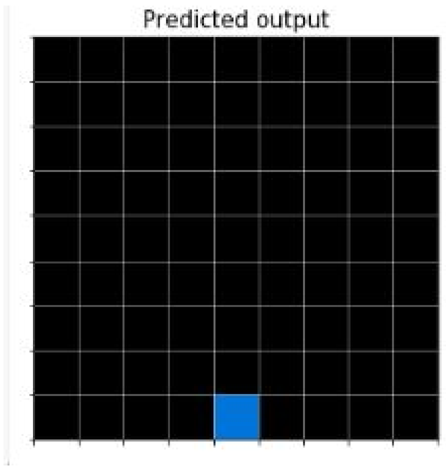


Figure 4. Example output grid.

## CNN Evaluation

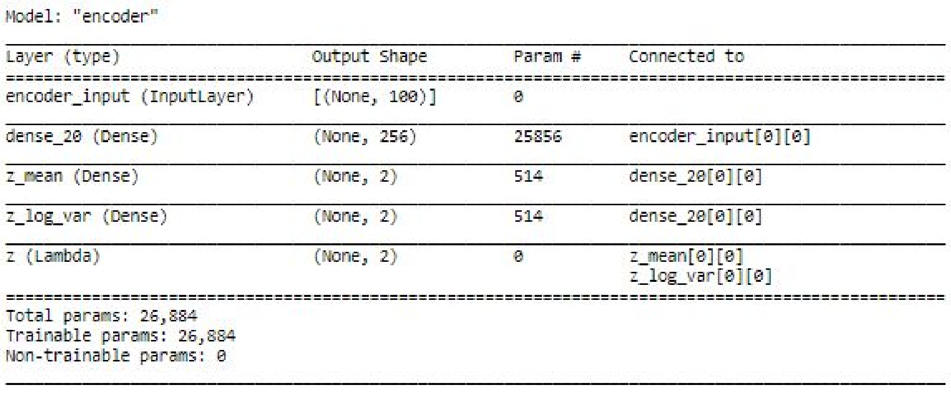
Ultimately the convolutional neural network approach was unsuccessful achieving an accuracy of 0. There are two likely reasons behind this. First, the training size of each task ranged from only two to ten samples. This is likely not enough to train a convolutional neural network. Even if it were, the network would likely severely overfit and struggle to accurately predict new cases. Additionally, the architecture of each model is not sophisticated enough to capture the unique patterns behind each task. Each task may require their own unique architecture to make accurate predictions.

A possible improvement to this implementation is to use a recurrent neural network to generate each individual cell value. By using a recurrent neural network, you are able to capture the information from all of the previous predicted cells, potentially allowing the network to understand patterns and how each cell is related to the next.

### Variational Autoencoder Model Building

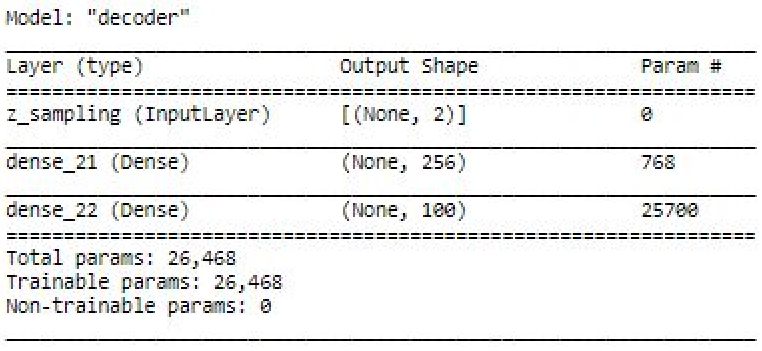
VAE's are a type of generative model that encodes the input as a distribution over a latent space. It is a generative model in that it is possible to sample that latent space to create new models. This latent space has the potential to capture rules of the patterns that created the input and output images.

The VAE is constructed by first defining its encoder. This encoder converts the input samples into two parameters in a latent space, which is then sampled. These samples are then mapped to the original input data through a decoder. A summary of the encoder and decoder architectures are shown below in Figures 5 and 6 respectively.



```
Model: "encoder"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
encoder_input (InputLayer)      [(None, 100)]        0
__________________________________________________________________________________________________
dense_20 (Dense)                (None, 256)          25856       encoder_input[0][0]
__________________________________________________________________________________________________
z_mean (Dense)                  (None, 2)            514         dense_20[0][0]
__________________________________________________________________________________________________
z_log_var (Dense)               (None, 2)            514         dense_20[0][0]
__________________________________________________________________________________________________
z (Lambda)                      (None, 2)            0           z_mean[0][0]
                                                                 z_log_var[0][0]
==================================================================================================
Total params: 26,884
Trainable params: 26,884
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 5. Encoder model summary



```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
z_sampling (InputLayer)      [(None, 2)]               0
_________________________________________________________________
dense_21 (Dense)             (None, 256)               768
_________________________________________________________________
dense_22 (Dense)             (None, 100)               25700
=================================================================
Total params: 26,468
Trainable params: 26,468
Non-trainable params: 0
_________________________________________________________________
```

FIgure 6. Decoder model summary

We were able to successfully train the model on the training data, but unable to incorporate the output data into the latent space. A possible successful implementation would have been to use the input data to train the parameters of the encoder and while using the output data to train the output of the decoder. That way, the latent space of the input data could be mapped to the latent space of the output data, and the model would successfully be able to predict the correct output.