Searching for initial parameters in cell colony spatial pattern generation

Nicolás Araya[1], Guillermo Iglesias[1] and Martín Gutiérrez[1]

[1]Escuela de Informática y Telecomunicaciones, Universidad Diego Portales, Santiago, Chile
martin.gutierrez@mail.udp.cl

Abstract

We propose a neural network based architecture to infer which parameters are fundamental, and their values, for producing specific instances of spatial patterns formed through cell colony growth. The system is trained on variations of the same pattern to recognize features that characterize it. Furthermore, selecting important parameters within our study mainly focuses on the fact that cells communicate. We use two forms of this communication as fundamental in finding the parameter values: bacterial conjugation, and environmental signals. The neural network is trained during 3000 epochs to identify the pattern class and specific parameter values needed to reproduce the desired pattern. These parameters are then inputted into a `gro` simulation to assess proximity to the original pattern. Our architecture achieved a 5% error upon pattern reproduction.

Introduction

One goal in Systems and Synthetic Biology is to be able to produce self-organization in cell populations to exhibit distinctive spatial and/or temporal features.

One form of early application of synthetic circuits is the design of synthetic circuits that exhibit spatial patterns. Examples of such circuits are: the band detector (Basu et al., 2005), the edge detector (Tabor et al., 2009) or the french flag model (Sohka et al., 2009). Resulting spatial distributions are of major importance as they set the foundations for materials and/or tissues requiring specific organization and provide a potential method for their construction (ex-vivo Tissue Engineering (Scholes and Isalan, 2017), for instance). Temporal patterns also offer an interesting path for cell colony control in a temporally organized manner. Coincidentally, the first example of synthetic circuit exhibiting a temporal pattern is the Repressilator (Elowitz and Leibler, 2000), which is one of the two founding synthetic circuits of the Synthetic Biology field.

To further understand and observe pattern formation processes, one way of classifying the behavior of biological microorganisms is by simulating spatial and/or temporal pattern formation over microbial populations. These patterns refer to recognizable and regular spatial organizations and/or temporal sequences that either are repeated or show a distinctive and predictable behavior in time and in a given spatial zone (Gutiérrez, 2017). At the metric scale of these organisms, specific spatial organizations or features are desired, and individual manipulation of these organisms is currently not possible, or would likely result in an exhausting and impractical endeavor. The aim is to be able to deliver design and manipulation precision to the level of micrometers in the creation of systems of this kind.

Due to limitations (and costs) present in natural systems and the large amount of resources needed to reproduce these patterns in biological systems, colony simulators are a crucial tool for obtaining results efficiently. The `gro` simulator (Jang et al., 2012; Gutiérrez et al., 2017) is a an agent-based simulator software that produces growing bacterial colonies simulations. The simulations treat each bacterium individually, therefore, each bacterium also harbors a specific program (in terms of gene circuits) that directs its own behavior. In light of this, it is possible to simulate spatial patterns according to given initial conditions. However, it is not possible to know how these input parameters influence the behavior of the bacterial colonies to reproduce specific patterns. Building a biological circuit and finding its corresponding input parameters is a complex and difficult process to carry out.

The use of Artificial Neural Networks (ANN) has increased in the last few years, motivated by technological advances in computational processing power and the use of units dedicated to graphics processing. This has allowed exploration of the potential of new models and architectures in a short time span. These tools imply a large impact on Computer Vision, where ANNs turn into an enabling technology for efficient acquisition, processing, generation, analysis and understanding of images. Specifically, Convolutional Neural Networks (CNN) (Lecun, 1985) are a type of ANN that use many layers of computation (Deep Neural Networks - DNN) to extract features from an image.

One of the main trademarks of a spatial pattern is the visual information it includes at a given moment in time. Within a digital image of the spatial pattern, each pixel con-

tains data relative to its context (color and location). These data could be then processed by an ANN to find additional information. Therefore, this work presents the implementation of a system based on CNN with the goal of predicting initial parameter values that led to the formation of a specific spatial pattern. In this manner, the colony evolution and spatial pattern formation processes are reproduced, accelerating and making it easier to achieve desired results.

Pattern formation in growing cell colonies

Cell colonies form patterns mostly due to interactions between individuals. Such interactions can be global (long-range) or local (short-range) in nature, and according to their interaction speed, give rise to a large array of pattern types (Wakita et al., 1998; Tokita et al., 2009; Kondo and Miura, 2010; Schaerli et al., 2014). In this work, two types of spatial patterns are the study focus: the bullseye pattern (Basu et al., 2005), and a sun pattern (branching or fluid propagation) (Wakita et al., 1998; Tokita et al., 2009). Both of the gene circuits generating these patterns were modified to include local intercellular communication among bacteria in the form of bacterial conjugation (Gutiérrez, 2017). We briefly describe both circuits, and identify important parameters governing their formation in the remainder of this section.

Bullseye Pattern

The Bullseye pattern is achieved through a multicellular synthetic circuit where the colony pattern organizes as differentiated concentric rings based on chemical gradients of an IPTG signal.

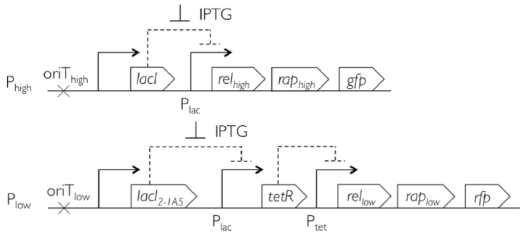

Figure 1: Design of the genetic circuit capable of generating the target pattern. The concentration of IPTG regulates the dissemination of each type of plasmid within a given zone. Specifically, one plasmid (P_{high}) reports high concentration of IPTG using *gfp*, and the other one (P_{low}) reports low concentration of IPTG by expressing *rfp*

.

For this pattern, the following parameters have a notorious effect on the specific outcome instance:

1. Conjugation rate: This parameter is related to how densely colored an area is. It determines how many conjugations are performed per life cycle. The higher the conjugation rate, the more frequently a plasmid will disseminate locally through the movement area of that plasmid. This is represented by a more uniform area in terms of color (green and red) and no color (black). This rate will be represented as Rel_{high} and Rel_{low}, corresponding to the conjugations of P_{high} and P_{low} respectively.

2. *lacI* sensitivity: This parameter has an effect on the width of the plasmid mobility areas. In practice, only the width of the zones expressing *gfp* (high concentration of IPTG, glows green) and colorless (medium concentration of IPTG) vary, due to the fact that the zone expressing *rfp* (low concentration of IPTG, glows red) only requires a threshold concentration of initial IPTG to detect and report all areas with concentrations of IPTG below that threshold. This sensitivity parameter will be represented as IP_{high}.

3. Initial amount of bacteria: The initial count of bacteria harboring plasmids that report with *gfp* and *rfp* and disseminate in their respective spatial zones. This parameter has an effect on the reporting density in all given zones. These parameters will be represented by P_{high} and P_{low} respectively.

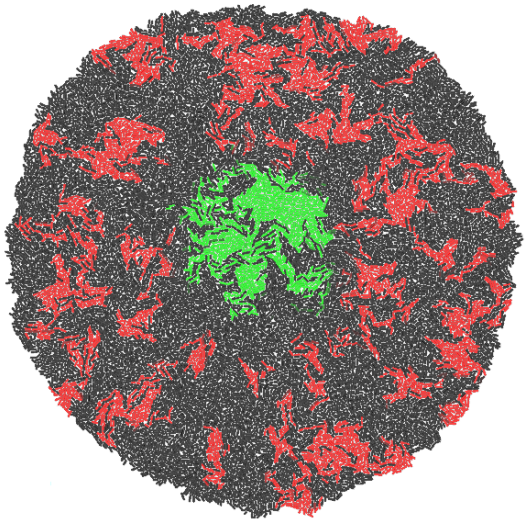

Figure 2: Pattern simulated with the help of the circuit of Figure 1 with a high conjugation rate. Many patches of color are visible in zones of the colony, as each respective plasmid is disseminated locally due to a high conjugation rate.

Sun Pattern

The Sun pattern is formed by a circular kernel located at the center of the colony and multiple ramifications of bacteria spreading from it.

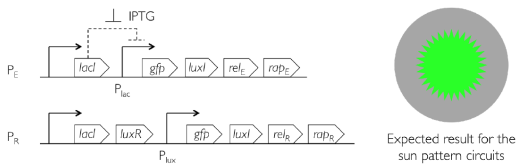

Figure 3: Design of the genetic circuit for generating the Sun pattern. Two plasmids direct the pattern formation: Emitter plasmid (P_E) and Receiver plasmid (P_R). The P_R plasmid detects sufficient concentration of IPTG and reports by using *gfp* a circular zone in the center of the colony. These bacteria also emit AHL that triggers bacterial conjugation for disseminating the P_R plasmid. This type of bacterium in turn emit AHL, creating relays resulting in the outer observable ramifications reported with *gfp*.

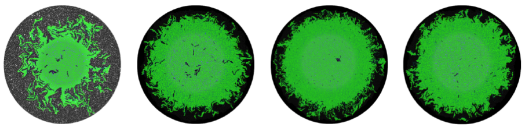

Figure 4: Pattern generated by the circuit described in Figure 3. A distinctive central zone, is colored by bacteria containing the P_E plasmid, while the ramifications are reported by bacteria with the P_R plasmid.

A single parameter needs to be considered in the tuning of this circuit:

1. Initial amount of bacteria: Determines the amount of bacteria at the start of simulation. These bacteria may contain the plasmid P_{E} (emitters) or P_{R} plasmid (receptors). These nomenclatures will be used to represent the initial count of each type of bacteria in this pattern.

Having identified the key interactions and specified parameters for each kind of studied pattern, the question of how to choose values for these parameters and organize the execution of the colony to achieve a specific instance of pattern with desired features arises as a central question.

Neural Networks for inferring parameters

Pattern formation is driven by the interactions between individuals in the colony. However, these interactions must be well characterized to ensure that the outcome fits with a given instance of a pattern. Once these interactions are selected and included in the pattern formation process, the following key aspect is to determine the parameters that govern such interactions, and assign a value to them. This can be a tricky process, since the effect of the parameter tuning is not always clear in relation to the modifications that it causes on the generated pattern.

In order to accelerate the process, analysis of generated patterns should be made automatic, and suggested parameter values proposed. Given that this is a problem exhibiting large dimensions (due mainly to the parameters linking to an analyzed image), a viable choice for serving as backbone for the system are Deep CNNs. These networks would therefore be trained to find parameters that are capable of reproducing expected instances of specific patterns.

Implementation

In this section, we describe an architecture based on a Convolutional neural network (CNN) (Lecun, 1985). We also describe how the dataset for working with the architecture was generated and then used in the network.

Dataset As a first step, a dataset composed by many images of pattern instances is created for the subsequent training of the CNN. The images are obtained from repeated simulations with the `gro` software. All simulations are carried out using the implementation of a genetic circuit, designed to form a specific spatial pattern. A set of 10 parameter configurations is selected to carry out these simulations and produce variants of pattern instances. These configurations are shown in Table 1 for the Bullseye pattern and Table 2 for the Sun pattern.

	Conjugation Rate		lacI Sensibility	Initial Bacteria	
ID	Rel_{high}	Rel_{low}	IP_{high}	P_{high}	P_{low}
1	1	1	0.1	300	100
2	1	1	0.1	200	200
3	1	1	0.1	100	300
4	1	1	1	300	100
5	1	1	1	200	200
6	1	1	1	100	300
7	0.3	3	1	200	200
8	0.3	3	0.1	200	200
9	3	0.3	1	200	200
10	3	0.3	0.1	200	200

Table 1: Parameter settings for the Bullseye pattern. These configurations produce different instances of the Bullseye pattern.

`gro` simulation Using the previously described circuits and the mentioned parameter settings, `gro` simulations are implemented. Each of the configurations is run 50 times and stops when the total bacterial count reaches 15000. Then, a picture of the colony is saved in PNG format. This procedure is repeated until the number of samples established is completed (for a total of 500 samples per type of pattern). The obtained samples are saved with the following nomenclature: `pat_p_c_n.png`, where `p` corresponds to the identifier of the pattern (0 : bullseye, 1 : sun), `c` is the identifier of the used configuration, and `n` is the sample number using the previous parameters (between 1 and 50 in each case).

	Initial Bacteria	
ID	P_{high}	P_{low}
1	200	650
2	250	600
3	300	550
4	350	500
5	400	450
6	450	400
7	500	350
8	550	300
9	600	250
10	650	200

Table 2: Parameter settings for the Sun pattern. Initial bacteria counts for generating variants of Sun pattern instances.

Furthermore, the information on the labels required for use by CNN is recorded in a $.csv$ file $(labels.csv)$ using the following format:

1. File: filename (includes extension).
2. Mode: Determine which set it belongs to. It can be training, validation or testing.
3. Pattern: Indicates which pattern the sample belongs to, expressed in identifiers from 0 to n - 1 standards $(0$: bullseye, 1 : sun$)$.
4. N_columns: N additional columns are added, depending on the number of parameters to be required. In this case, a total of 7 columns additional were required. Each one contains the value of the parameter corresponding to the sample. In case the parameter is not used in the sample, the value is replaced with a 0.

Finally, the samples are randomly divided into three sets: training (60%), validation (20%) and test (20%). The same ratio is maintained for each of the parameter settings used by each pattern.

Architecture Model Once the dataset generation is done, we constructed a CNN to classify images as Bullseye or Sun patterns and estimate the initial parameter values for producing the desired pattern instance. The CNN architecture is described in detail below:

1. Input Layer: In this layer, the image sample enters the network. The image, in PNG format, has been pre-processed using the following procedure:
   - Color transformation to gray-scale
   - Centered cropping of the image to a 600x600 pixels size.
   - Image scaling to 100x100 pixels dimension.
   - Image normalization based on mean and standard deviation of the complete dataset.
2. Convolutional Layer 1: The image is processed using filters with a 5x5 dimension. Additionally, a 2 pixel padding is added so that the filters are applied 2 pixels above the edge of the image, preserving the original dimension. This layer outputs 32 images of size 100x100. Finally, a ReLU activation function is applied to transform negative values to zero.
3. Grouping Layer 1: This layer reduces the number of parameters in the model through a downsampling process, and in turn preserves the most important information about each feature map. It provides robust feature detection for changes to the object, such as orientation, position, and scale, thus allowing for better generalization. In this layer max pooling is also used. In this process, a 2x2 pixel kernel slides through each image, keeping the maximum value within the window at each step. Also, the kernel performs a two pixel shift at each step. As a result, 32 images with half the original dimension (50x50 pixels) are obtained.
4. Convolutional Layer 2: This layer performs the same functionality as the convolutional layer 1. However, it receives 32 images of size 50x50 pixels. As a result it delivers a total of 64 50x50 pixel size images.
5. Grouping Layer 2: This layer performs the same functionality as grouping layer 1. However, the result of this layer is 64 images of size 25x25 pixels.
6. Dense Layer 1: Before entering this layer, the images from the previous layer are transformed into a (flattened) one-dimensional matrix. In this case, the 64 images of size 25x25 pixels are transformed into a matrix of size 1x40000 (64x25x25). At this point, the artificial neurons in the layer are fully connected between their inputs and their following outputs. In addition, to prevent overfitting of the network for certain features, dropout operations are carried out. This regularization mechanism deactivates some neurons randomly during the training process, thus avoiding coaptation in the detection of features. The resulting layer then applies a linear transformation to the input data. As a result, the 1000 neurons are fully connected to another 100 output ones in the next layer.
7. Dense Layer 2: This layer, like dense layer 1, is fully connected. It is made up of 100 neurons that perform a linear transformation. These neurons are fully connected to the following layers.
8. Output Layer: This last layer is made up of 2 sub-layers, where each one receives 100 inputs from the layer described above, and performs a linear transformation. Each of the sub-layers exhibits different functionalities:

- Classification: This sub-layer solves the classification of the spatial pattern. A vote is carried out using all 100 inputs. The result is divided into as many neurons as there are spatial patterns to be classified. In this case, 2 neurons represent each of the potential spatial patterns to be classified. The output will indicate which pattern the input image belongs to.
- Prediction: This sub-layer is in charge of predicting the values of each parameter used to simulate the desired pattern instance. Each neuron will be in charge of predicting the value of a single parameter. The sum of the number of parameters for all analyzed patterns will be the number of neurons present in this sub-layer. In this case, a total of 7 parameters to be predicted were placed. Since parameters are specific to each pattern, the parameter values that do not correspond to the pattern under analysis are set to a value of 0. Additionally, the parameter values used in the test set labels have previously been scaled to values ranging from 0 to 100. The final output will indicate the predicted value of each parameter and all values that do not correspond to the real pattern will be discarded.

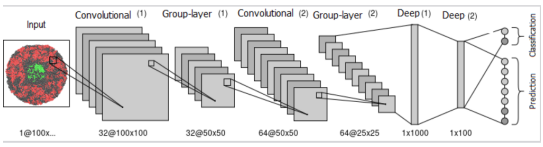

Figure 5: CNN architecture depiction. The dimensions for each layer are expressed in the format image@size.

Training This CNN is based on the Backpropagation algorithm (Rumelhart et al., 1986) for learning. It is a process that updates the weights in the CNN by using the loss value associated to the calculated estimate, and using it to readjust the weight values to decrease the error. In this case, two different loss functions are used:

- Cross-Entropy Loss:

$$loss(x, y) = -\sum xlog(y) \tag{1}$$

This loss function measures the cross-entropy between the predicted value and the ground truth value, where x is the probability of the real label and y is the probability of the predicted label. In the presented network, this function will be used with the output of the classification sub-layer.

- Smooth L1 Loss:

$$loss(x, y) = \frac{1}{n}\sum_{i}\begin{cases} 0,5(x_i - y_i)^2, \text{ if } |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5, \text{ otherwise} \end{cases} \tag{2}$$

This loss function uses a square term if the absolute error falls by below 1 and an absolute term otherwise, where n is the total number of elements, x is the ground truth value, and y is the predicted value. This function is less sensitive to outliers than other similar loss functions (for example, root mean square error loss). In some cases, it contributes in avoiding exploding gradients. In this CNN, the function will be used with the outputs of the prediction sub-layer.

The steps of the training cycle are:

1. Choose a batch of images that has not gone through the training process.
2. Input these images into the network. These will feed the network providing two estimates: pattern classification and parameter prediction.
3. Use the pattern classification results to calculate the loss function using cross-entropy loss.
4. Use the parameter prediction results to calculate the loss function using smooth L1 loss.
5. Sum the obtained loss results.
6. Using gradient descent, update the CNN weights.
7. Repeat the process with the next batch of images.

A hyperparameter for network function remains to be set: learning rate. The learning rate directly influences the loss results and prediction (Figure 6). This hyper-parameter indicates the magnitude to be used for updating the weights during the gradient descent process. Large values for this hyperparameter could decrease the time required for training, but may cause the model to fail to converge to a minimum loss value. A learning rate of 0.01 and a sample size per batch of 16 images are used as hyperparameters. This is due to a previous evaluation of several configurations over 200 iterations.

The implementation of the described CNN was done in Python, using the PyTorch library (Paszke et al., 2017). All experiments were carried out using Google Colab with a Nvidia Tesla T4 16 Gb GPU.

Results

In this section, we present the results obtained after training the CNN for 3000 epochs. These results are compiled into Figure 6 and Tables 3, and 4. It should be added that a prediction is made on a batch of 15 random images from the test set. These results are shown in Table 4.

First, 100% ranking hits are displayed. This can be explained by two possible cases: 1) the model is able to generalize the pattern features and therefore classify these unknown samples correctly, or 2) it arbitrarily classifies these

Dataset-epoch	Accuracy	Rel_{high} %	Rel_{low} %	IP1 %	P_{high} %	P_{low} %	P_E %	P_R %
test-10	51.50	96.79	101.56	96.36	97.14	101.10	101.36	100.48
test-400	100.00	44.49	53.43	80.48	19.28	17.82	3.90	3.75
test-1600	100.00	11.22	10.02	12.50	3.55	3.13	1.14	0.83
test-2000	100.00	10.33	9.95	11.94	2.78	2.90	0.79	1.20
test-2500	100.00	10.63	8.95	12.35	2.65	3.45	1.06	0.89
test-3000	100.00	10.01	7.57	12.19	2.75	2.83	1.00	0.93

Table 3: Accuracy results for all parameters involved in the formation of both pattern types.

ID	Image pattern	Rel_{high} rate	Rel_{low} rate	IP1 threshold	P_{high}	P_{low}	P_E	P_R
1	bullseye / bullseye	1.00 / 1.03	1.00 / 1.17	1.00 / 0.98	200 / 198	200 / 208	-	-
2	bullseye / bullseye	1.00 / 1.04	1.00 / 1.00	0.10 / 0.12	100 / 101	300 / 303	-	-
3	bullseye / bullseye	1.00 / 1.26	1.00 / 0.84	1.00 / 1.00	300 / 288	100 / 113	-	-
4	bullseye / bullseye	1.00 / 1.02	1.00 / 1.02	0.10 / 0.10	200 / 201	200 / 203	-	-
5	sun / sun	-	-	-	-	-	200 / 197	650 / 647
6	sun / sun	-	-	-	-	-	550 / 548	300 / 299
7	bullseye / bullseye	1.00 / 0.95	1.00 / 1.05	1.00 / 0.98	100 / 104	300 / 292	-	-
8	bullseye / bullseye	0.30 / 0.30	3.00 / 2.87	0.10 / 0.21	200 / 206	200 / 208	-	-
9	bullseye / bullseye	3.00 / 2.79	0.30 / 0.44	0.10 / 0.07	200 / 203	200 / 191	-	-
10	bullseye / bullseye	3.00 / 2.94	0.30 / 0.35	1.00 / 1.00	200 / 201	200 / 200	-	-
11	bullseye / bullseye	1.00 / 0.86	1.00 / 1.35	1.00 / 0.92	100 / 119	300 / 285	-	-
12	sun / sun	-	-	-	-	-	350 / 348	500 / 498
13	sun / sun	-	-	-	-	-	250 / 248	600 / 598
14	bullseye / bullseye	1.00 / 0.95	1.00 / 1.00	0.10 / 0.10	300 / 295	100 / 102	-	-
15	bullseye / bullseye	0.30 / 0.35	3.00 / 2.98	0.10 / 0.13	200 / 207	200 / 203	-	-

Table 4: Random samples of prediction on images from the test set. Values are presented in the format: actual / predicted.

patterns as a target, since it corresponds to the first output neuron. The first case is more likely, because when testing the model without training, random results of Classification averages are close to 50%. This indicates that without training, the model behaves in the same way as it was initialized, giving a 50% probability of achieving a successful classification.

This hypothesis is also supported by the fact that the difference between predicted parameters averages 5%. Therefore, it is likely that the network is capable of generalizing the characterization of the patterns along with the choice of very similar parameters values. It is from these observations that we can conclude that our architecture correctly classifies and predicts the associated parameter values for spatial pattern instance reproduction.

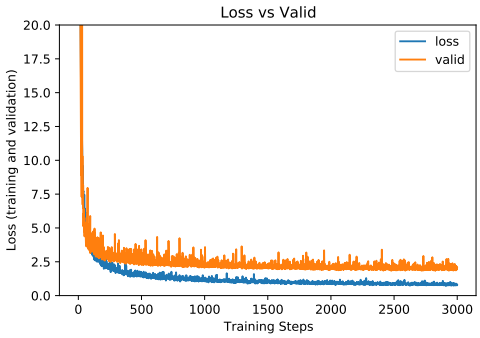

Figure 6: Loss of training results and prediction over 3000 iterations.

Discussion and future work

The CNN-based architecture presented in this work sets the foundations for characterizing numerical values for parameters driving spatial pattern formation in cell colonies. Evidently, other kinds of AI tools such as Autoencoders (AEs) or Recurrent Neural Networks (RNNs) are able to extend the approach here presented. For instance, temporal patterns could also be recognized and characterized in their driving parameters by studying time series and combining the current architecture with a RNN.

Unfortunately, it was not possible to test the architecture on microscopy images for patterns generated in an actual cell colony. Training the architecture on patterns with good definition such as in the simulations is a good way of capturing the spatial/visual features that characterize the pattern. Also, it should be stressed that all simulations did present variability at multiple levels (growth, gene expression, cell-cell communication, etc.) to produce stochasticity and randomness within the simulation, giving rise to variations of patterns (which sometimes contain spatial discontinuities or are patchy). This is why we believe that our testing

conditions for the architecture were appropriate. Even so, testing this architecture on microscopy images is something that remains to be done.

The results for executing the proposed architecture show that it is capable of recognizing spatial patterns quite accurately and estimating the parameters for reproduction of such pattern formation. However, the selection of said parameters depends on what should be varied within the experiment to produce the pattern. In this case, the non-trivial relationship between inter-cell communication and pattern formation was investigated, leading to the selection of conjugation rates, emission values and cell strain population distribution as main parameters. This should not be interpreted as a limitation, in that other parameters may be considered. In fact, the architecture is designed to be extensible in this sense, as it contains both a set of output neurons for pattern classification and a set of neurons for parameter prediction. The form of this final output layer can be adjusted to meet the classification and prediction requirements for other types of spatial patterns, but also, to include other parameters driving colony behavior for the prediction part. In this sense, another possible future application stemming from the presented architecture is to investigate how selected parameter variation affects the formation of a given spatial pattern.

Experimentation on this architecture was done through training with variations of selected spatial patterns and the parameters driving their formation. One further question that was investigated was: What happens when an image shown to the CNN is neither a bullseye nor a sun pattern colony? Our team designed a solution to this question by including an additional output neuron to serve as a "neither" label, classifying the image as none of the explicitly defined patterns. We had limited success with this approach since the behavior of the "neither" output neuron was erratic: the network was unable to assign the "neither" cases solely to its designated output neuron, but randomly selected any output neuron to activate. Therefore, next steps for this work are to achieve a global architecture involving classification of spatial and temporal patterns and prediction of parameter values for their reproduction, to establish new strategies for definition and selection of parameters to be used in the design and training of the network, define new strategies for negative control, and to test our architecture on microscopy images to assess its accuracy and generalization potential.

Acknowledgements

No funding supported this research. The authors wish to thank Javier Carrión, Demian Schkolnik and Luciano Ahumada for their input.

References

Basu, S., Gerchman, Y., Collins, C., Arnold, F. H., and Weiss, R. (2005). A synthetic multicellular system for programmed pattern formation. *Nature*, 434:1130.

Elowitz, M. B. and Leibler, S. (2000). A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335.

Gutiérrez, M. (2017). *A new agent-based platform for simulating multicellular biocircuits with conjugative plasmids*. Universidad Politécnica de Madrid, Madrid.

Gutiérrez, M., Gregorio-Godoy, P., Perez del Pulgar, G., Muñoz, L. E., Sáez, S., and Rodríguez-Patón, A. (2017). A new improved and extended version of the multicell bacterial simulator gro. *ACS synthetic biology*, 6(8):1496–1508.

Jang, S. S., Oishi, K. T., Egbert, R. G., and Klavins, E. (2012). Specification and simulation of synthetic multicelled behaviors. *ACS synthetic biology*, 1(8):365–374.

Kondo, S. and Miura, T. (2010). Reaction-diffusion model as a framework for understanding biological pattern formation. *science*, 329(5999):1616–1620.

Lecun, Y. (1985). A learning scheme for asymmetric threshold networks. *Proceedings of Cognitiva*, 85:599–604.

Paszke, A., Gross, S., Chintala, S., Ghanan, G., Yang, E., DeVito, Z., Lin, A., Desmaison, A., Antiga, L., and Lerer, A. (2017). *Automatic differentiation in Pytorch*.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Schaerli, Y., Munteanu, A., Gili, M., Cotterell, J., Sharpe, J., and Isalan, M. (2014). A unified design space of synthetic stripe-forming networks. *Nature communications*, 5:4905.

Scholes, N. S. and Isalan, M. (2017). A three-step framework for programming pattern formation. *Current opinion in chemical biology*, 40:1–7.

Sohka, T., Heins, R. A., Phelan, R. M., Greisler, J. M., Townsend, C. A., and Ostermeier, M. (2009). An externally tunable bacterial band-pass filter. *Proceedings of the National Academy of Sciences*, 106(25):10135–10140.

Tabor, J. J., Salis, H. M., Simpson, Z. B., Chevalier, A. A., Levskaya, A., Marcotte, E. M., Voigt, C. A., and Ellington, A. D. (2009). A synthetic genetic edge detection program. *Cell*, 137(7):1272–1281.

Tokita, R., Katoh, T., Maeda, Y., Wakita, J.-i., Sano, M., Matsuyama, T., and Matsushita, M. (2009). Pattern formation of bacterial colonies by escherichia coli. *Journal of the Physical Society of Japan*, 78(7):074005–074005.

Wakita, J.-i., Rafols, I., Itoh, H., Matsuyama, T., and Matsushita, M. (1998). Experimental investigation on the formation of dense branching morphology-like colonies in bacteria. *Journal of the Physical Society of Japan*, 67(10):3630–3636.