Classifying the scaffold routing of computed DNA origami designs using CNN

Tito Babatunde
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
bbabatun@andrew.cmu.edu

Weitao Wang
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
weitaowa@andrew.cmu.edu

Abstract

DNA nanotechnology can form nanostructures with user-defined geometries based on base-pairing mechanism. However, current bottom-up design tools for such structures requires expertise on DNA. With simulated annealing we can stochastically encapsulate a given structure with a scaffold single-stranded DNA (ssDNA), thus providing a novel top-bottom method for the design of complicated DNA nanostructures. To evaluate the quality of scaffold encapsulation, the existence of gaps is an important metric that differentiates fully encapsulated structures from not fully encapsulated ones. Here we propose to apply convolutional neural networks (CNN) to classify the output images of simulated annealing algorithm. We found that our CNN was able to improve its accuracy to near 100% after the dataset was reclassified to three classes: fully encapsulated, not fully encapsulated with small gaps and not fully encapsulated with big gaps. The application of deep neural networks enables us to efficiently select structures that are fully encapsulated.

1. Introduction

DNA nanotechnology, an exponentially progressing field, focuses on the design and self-assembly of predictable and programmable DNA-based nanostructures due to the distinct Watson-Crick base-pairing where adenine (A) and cytosine (C) form hydrogen bonds with thymine (T) and guanine (G) [1]. Due to DNA's programmable bases, the field offers an unmatched ability to control the formation and size of arbitrarily shaped and biocompatible DNA nanostructures [2]. The evolution of DNA nanotechnology leads to the creation of DNA origami, which entails the folding of a long single-stranded DNA (ssDNA) (or scaffold) directed by hundreds of short oligonucleotides (or staples) to form intriguing polyhedral nanostructures (Figure 1) [3,4]. Recently, DNA origami has served to create nanodevices with a plethora of functional applications such as multifluorophore beacon sensing platforms [6], cargo-sorting robots [7], and drug delivery vehicles [8].

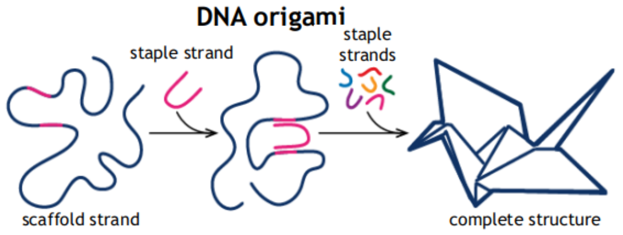


Figure 1: Formation of DNA origami nanostructure. Scaffold strand forms hydrogen bonds with complementary staple strand (pink), which folds scaffold into desired configuration [5].

2. Related Work

Currently, a majority of DNA origami nanodevices are manually designed with caDNAno, a bottom-up software that starts at the base scale to create a DNA origami design [9]. CaDNAno, which uses a square or honeycomb lattice architecture, creates 2D drawings of DNA nanostructures that fold into 3D, as shown in Figure 2(A). [9]. However, caDNAno heavily relies on the user's expertise which slows the design process and constrains the design space to simple geometries. Automated tools can help accelerate the design process and expand the design space by utilizing a top-down approach that starts with an outline of the desired configuration and works backwards to define the DNA base sequence to form the nanostructure. However, current automated tools are limited to wireframe structures [10, 11] or fully conceptualized structures of constant cross-sections [12], as shown in Figure 2 (B) and (c). Currently, there is no top-down automated tool capable of performing parameterized design to conceptualize complex multilayer origami nanodevices [13].

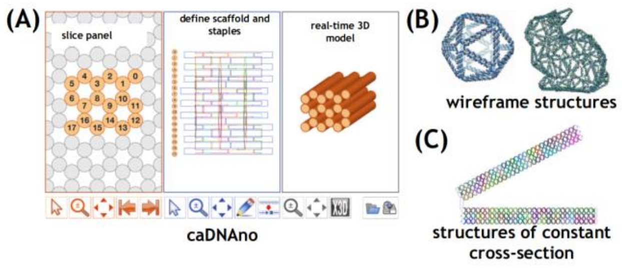


Figure 2: Current bottom-up and top-down DNA origami design

software. (A) An illustration of a simple DNA origami design in honeycomb CaDNAno [9]. (B) Examples of wireframe structures designed on the top-down DAEDELUS and vHelix [10, 11]. (C) A simple DNA origami design in MagicDNA, a top-down design tool [12].

Here we investigated a flexible top-down framework that was capable of addressing current software limitations by computing optimal scaffold routing only with a set of constraints and desired features [13]. This framework is achieved through simulated annealing, a robust and general random technique that statistically approaches global optimum among numerous local optima by accepting worse solutions early on [14]. Developed by Kirkpatrick et al., simulated annealing can stochastically optimize parameters for an arbitrary model and ensures a good solution within sufficient time [14]. The algorithm randomly samples a feasible solution *u*, and the energy at that solution, E_u, is calculated. Another random feasible solution, *v*, is sampled, and the energy, E_v, is calculated. In the case of objective minimization, *v* replaces *u* if $E_v < E_u$, of if $E_v \geq E_u$, with a probability as a function of temperature, *T*:

$$P_{accept} = \exp - \left(\frac{E_v - E_u}{T}\right)$$

If a generated random number between 0 and 1 is less than P_{accept}, then *v* is accepted; if not, *u* is not replaced. The higher the temperature, the higher the probability of accepting a worse solution. Alternatively, at set temperatures, the higher the difference between the energy states, the lower the probability of accepting worse solutions. The algorithm runs for several iterations (or mutations) at a set temperature value until convergence or equilibrium is reached, or the temperature reaches 0. The algorithm is analogous to the annealing process of metals, where the energy can be substituted for an objective function. An appropriate cooling schedule must be chosen to reduce the temperature. Although the most appropriate cooling schedule which ensures convergence to the global optimum would be one that follows a logarithmic trend, it would exponentially increase the search time for large problems. Here we use a geometric cooling schedule that follows an exponential trend $T := \alpha T$, where α is between 0 and 1. Although this schedule does not ensure convergence to the global optimum, it searches for a good solution in sufficient time. After the temperature has reduced, the algorithm runs for several iterations till T=0 is reached.

This top-down framework creates unique routing patterns of the scaffold that are constrained to the caDNAno honeycomb lattice architecture. With a cube as input from the user, the framework's capability was demonstrated with the coating application where the goal is to encapsulate the surface a user-defined geometry [13]. Since the goal is to encapsulate a given geometry, the metric of success is based on the presence of gaps. It is currently difficult to create a method for evaluating the presence of gaps. As a simple demonstration of the framework's ability to encapsulate user-defined geometries, the framework was tested with a simple cube, shown in Figure 3.

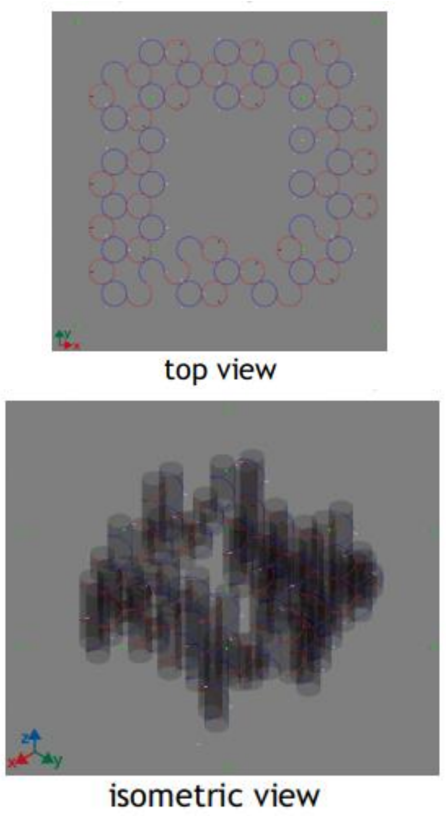


Figure 3: Scaffold DNA origami cube design generated with the coating application, where the green points are the inner and outer constraints. Input cube has side length of 6 nm.

3. Data

Since the simulated annealing algorithm can generate an infinite number of unique scaffold routing patterns due to its stochastic nature, we ran the algorithm, using tailored parameters, 1183 times to generate encapsulated DNA origami nanostructures given a cube of 6 nm in sidewall length as input. Images of the top-view of these structures were manually generated, classified and resized to 200×200 as preprocessing, and were served as the data-set to evaluate the presence of gaps.

4. Methods

The generated images were initially labeled into the two classes, class 0 as not fully encapsulated and class 1 as fully encapsulated, with 400 images in each class (see Figure 4). The images that were not fully encapsulated had a minimum of three side-walls to images missing only a single cylinder.

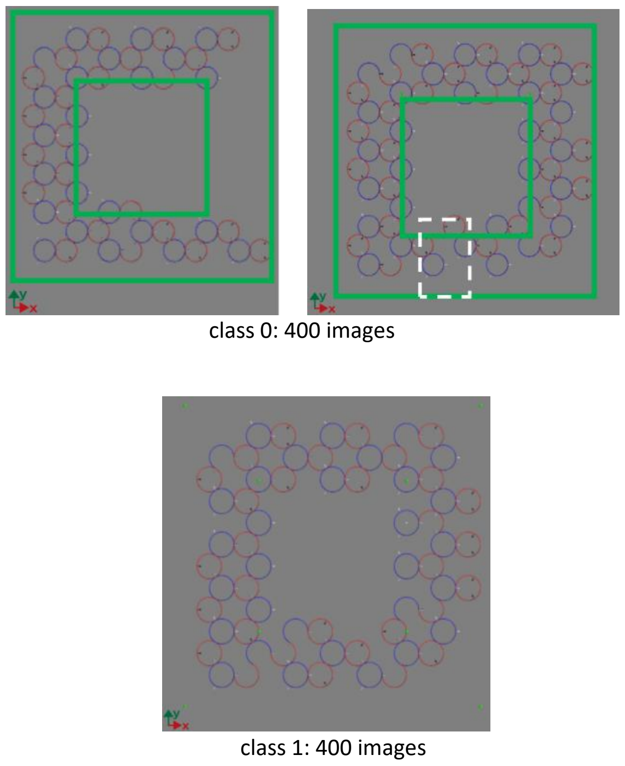
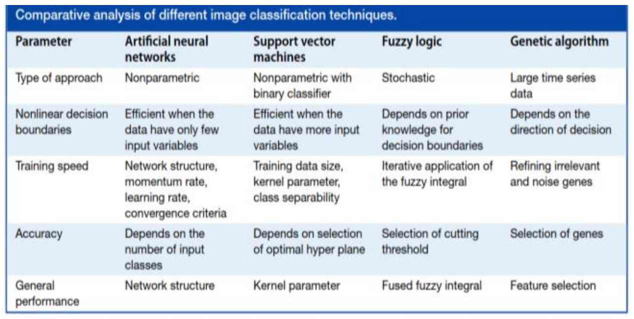


Figure 4: Top-view of image classes. Example of class 0 with a minimum of three side walls where green squares represent inner and outer constraints and white dashed square represents location of smallest gap (top). Example of class 1 with fully encapsulated cube (bottom).

Table 1 and Figure 5 show the performance comparison on cutting edge image classification and consider the accuracy of the fuzzy measure, decision tree, support vector machine, and neural network methods based on results from a literature survey [15]. Based on the comparative analysis in Figure 5, it is evident that CNNs provide a better accuracy than existing image classification techniques due to its ability to process images by extracting hidden features, allow parallel processing, and real time operation [16]. For such reasons, we created a custom CNN for image classification.



Comparative analysis of different image classification techniques.

Parameter	Artificial neural networks	Support vector machines	Fuzzy logic	Genetic algorithm
Type of approach	Nonparametric	Nonparametric with binary classifier	Stochastic	Large time series data
Nonlinear decision boundaries	Efficient when the data have only few input variables	Efficient when the data have more input variables	Depends on prior knowledge for decision boundaries	Depends on the direction of decision
Training speed	Network structure, momentum rate, learning rate, convergence criteria	Training data size, kernel parameter, class separability	Iterative application of the fuzzy integral	Refining irrelevant and noise genes
Accuracy	Depends on the number of input classes	Depends on selection of optimal hyper plane	Selection of cutting threshold	Selection of genes
General performance	Network structure	Kernel parameter	Fused fuzzy integral	Feature selection

Table 1: A comparative analysis of different image classification techniques [15].

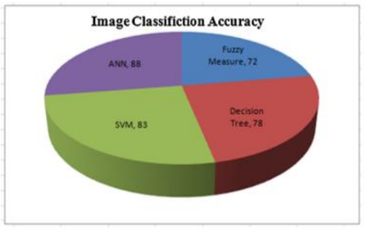


Figure 5: A comparison of the accuracy of different image classification methods [15].

Figure 6 displays the custom CNN with the following descriptions where CONV(*a*, *b*) means a convolution layer with *a* filters and a window size of *b* × *b*, POOL(*c*) means a pooling layer with a factor of *c*, FC(*a*) means a fully-connected layer with *a* units, DROPOUT(*d*) means drop *d*/100 neurons, and NORM means batch normalization [17].

CONV(16, 3), RELU, NORM, CONV(32, 3), RELU, NORM, POOL(2), CONV(64, 3), NORM, CONV(128, 3), NORM, POOL(2), CONV(256, 3), NORM, CONV(512, 3), NORM, POOL(2), CONV(512, 3), NORM, POOL(2), CONV(512, 3), NORM, POOL(2), DROPOUT(0.40), FC(1024), RELU, DROPOUT(0.30), FC(1024), RELU, DROPOUT(0.20), FC(2), SOFTMAX.

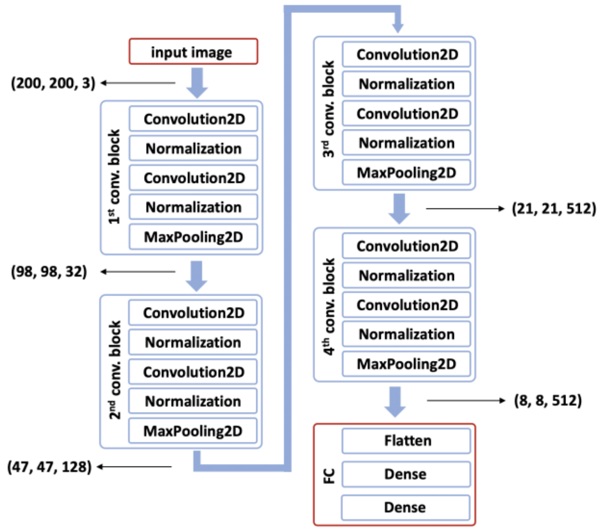


Figure 6. An overview of the custom CNN frame.

The custom CNN architecture takes images of size 200×200 as input where the convolutional layer runs a sliding window through the images and convolves the sub-image with *a* filters at each step to generate a volume with larger depth [17]. The pooling layer down samples the generated volume along the spatial dimension by *c* factor [17]. The batch normalization layer normalizes the output of the previous layer and thus allows each layer to learn more independently [18]. It is also used here as regularization to prevent overfitting. The dropout layer randomly turns *d*/100 of neurons off in order to boost the learning of the model [18]. It is also used as a regularization technique for overfitting prevention.

The following arbitrary parameters were selected for the CNN: batch size of 20, adam optimizer with a learning rate of 0.0001 and a decay of 1e-6, binary crossentropy loss, epoch length of 100, and a validation split of 30%. The following batch size was selected to reduce oscillations in the accuracy calculation. Adam was selected because it is a robust and general optimizer, where the learning rate was reduced with a time-based decay as the training progressed. Binary crossentropy loss was selected since we designed a two-class problem.

In addition, we also implemented the popular pretrained models of VGGNet and ResNet50 for comparison to the custom CNN architecture [Fig. 7].

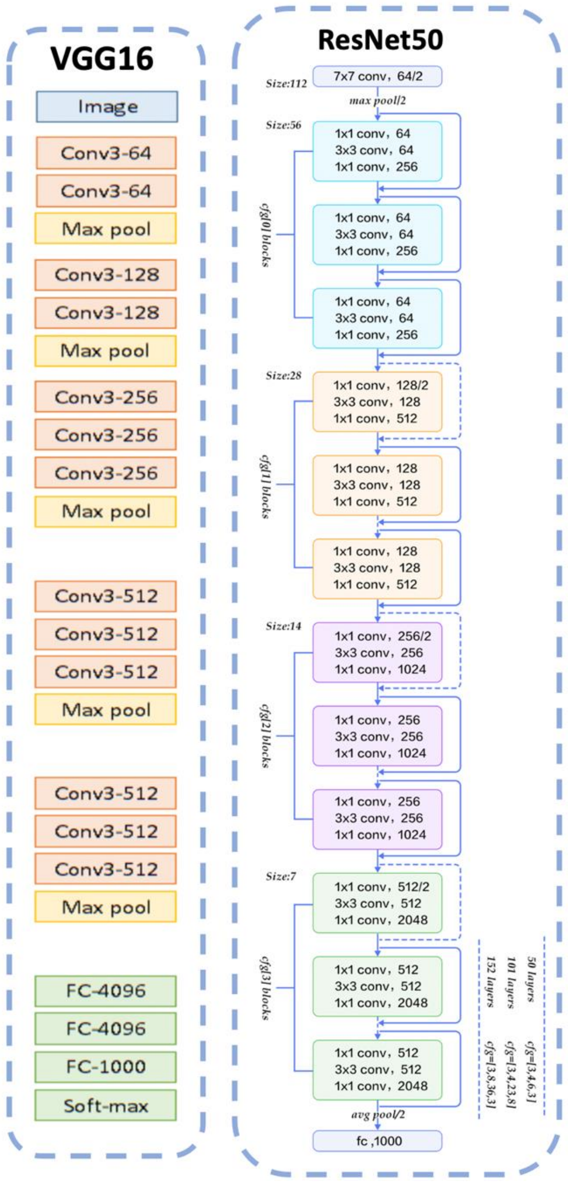


Figure 7. Popular architecture models. VGG (Visual Geometry Group) architecture (left). ResNet (Residual Network) architecture (right) [19].

5. Experiments

Experimental results, including accuracy and loss plots of the custom CNN, VGG16 and ResNet50 are shown here [Fig. 8-10]. For our custom CNN, the train set and test set accuracy reach 100% and 73% respectively after 60 epochs. The loss of train set and test set decrease to 0 and 2.6. Both the accuracy and loss trend are reasonable. But the accuracy is not as high as expected. Considering the images are relatively simple, we expected the results would be better. We then tested VGG16 and ResNet50 to see if it's because our custom CNN caused the low accuracy. The test accuracy of VGG16 reaches 86% after 50 epochs, which is better than custom CNN. However, the loss of validation set increases slowly with increasing epochs. We could not explain the exact reason, but since the loss difference is very small on the same scale, we think the loss is minimized for the validation set. ResNet50 gives a small improvement of accuracy to 78%, and a low loss for the test set.

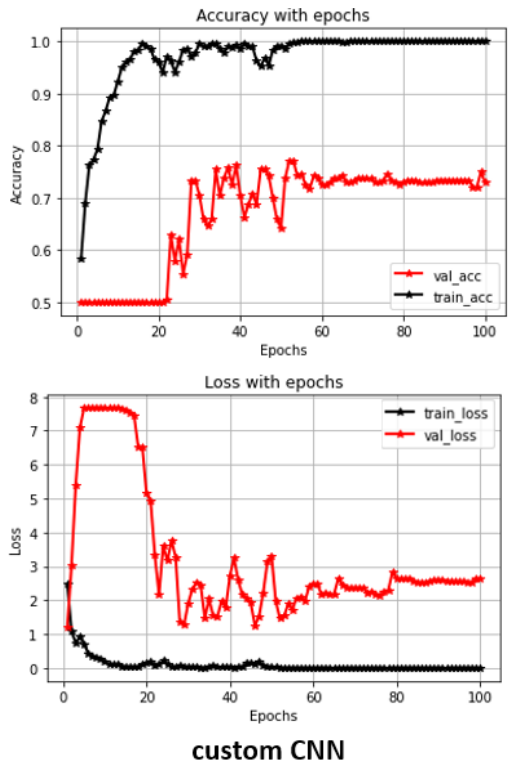


Figure 8. Accuracy and loss vs. number of epochs for custom CNN.

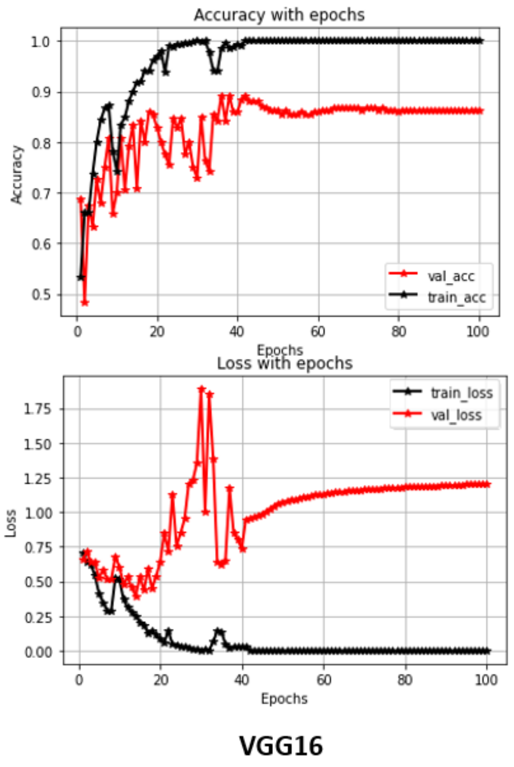


Figure 9. Accuracy and loss vs. number of epoch lengths for pretrained VGG16

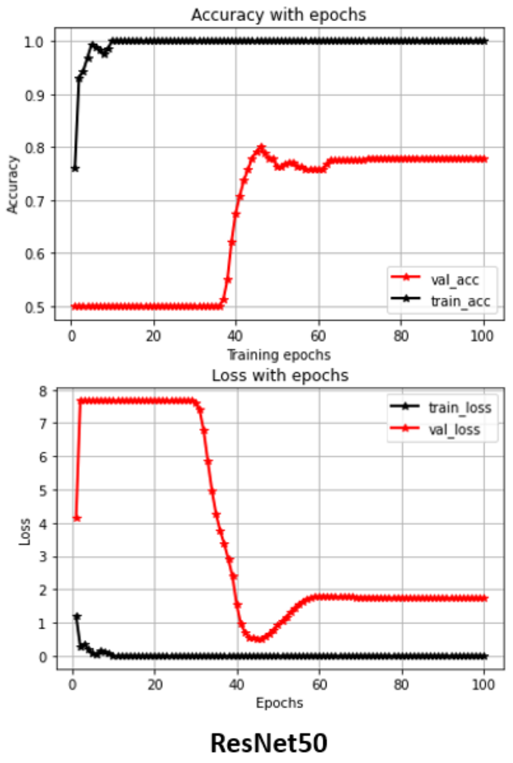
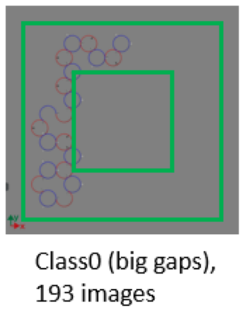
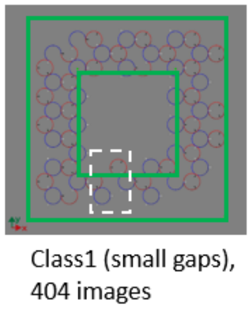


Figure 10. Accuracy and loss vs. number of epoch lengths for pretrained ResNet50.

Although we saw improvements from VGG16 and ResNet50, the accuracies did not fit our expectations. Considering both VGG16 and ResNet50 are excellent deep neural networks, we reason that the low accuracy problem is caused by our dataset, the images. As mentioned in methods dataset section, we manually classified our images to two different classes, class 0 as not fully encapsulated and class1 as fully encapsulated. However, in class 0, there are a group of images (~100) that are not fully encapsulated, but with very small gaps, as shown in Figure 4 with the white dashed block. These kinds of images are hard to differentiate from fully encapsulated ones. We think these images cause the deep neural networks to misclassify them to the fully encapsulated class (1). To prove our hypothesis, we discriminated the big gaps and small gaps in class 0, and reclassified our dataset into three classes, class 0 as not fully encapsulated with big gaps, class1 as not fully encapsulated with small gaps and class 2 as fully encapsulated (Figure 11).





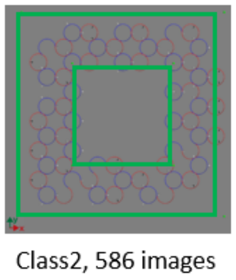


Figure 11. Dataset reclassification: class 0 as not fully encapsulated with big gaps, class 1 as not fully encapsulated with small gaps and class 2 as fully encapsulated.

We tested the dataset with new classification using the custom CNN. Results are shown in Figure 12. The accuracy for both the train set and test set reach 100% quickly. The losses are close to 0 after 30 epochs. Th results show that the custom CNN can classify our dataset correctly after reclassification.

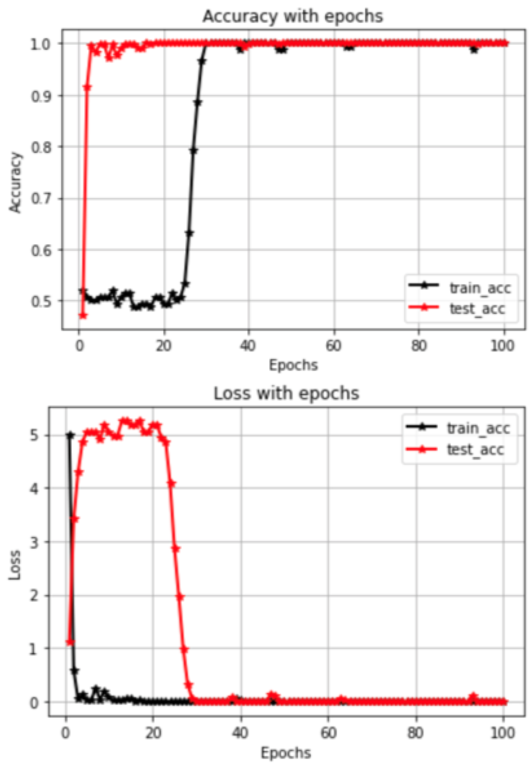


Figure 12. Accuracy and loss plots of custom CNN using three-class dataset.

6. Conclusions

To sum up, all three deep neural networks, the custom CNN, VGG16 and ResNet50, were able to classify the top-view images of scaffold encapsulation of a given geometry that is generated from simulated annealing with 70% - 90% accuracy using initial classification, where only fully encapsulated and not fully encapsulated were differentiated. After a finer dataset classification that separated not fully encapsulated structures with small gaps and big gaps, the custom CNN was able to reach near 100% accuracy of classification for validation set. Our results show that the classification of dataset is critically important for the performance of the deep neural networks. Our propose that aims to classify the quality of encapsulations from simulated annealing algorithm is successful. In future, reinforcement learning can be applied for searching, optimizing the parameters in simulated annealing, and finding the optimal encapsulation of a given geometry. By combining simulated annealing and deep learning, top-down design and manufacturing of DNA nanostructures with custom geometries will more opportunities to the community.

7. Contributions

Babatunde generated the dataset. Both Babatunde and Wang contributed to coding the custom CNN, adjusting its parameters and writing the report.

References

[1] Liu, Y., Kumar, S., and Taylor, R. E., *WIREs Nanomedicine and Nanobiotechnology*, 10(6):e1518, 2018.
[2] Tørring, T. and Gothelf, K. V., *F1000Prime Reports*, 5(14):1-4, 2013.
[3] Rothemund, P. W., *Nature*, 400(7082):297-302, 2006.
[4] Castro, C. E. et al., *Nature Methods*, 8(3):221-229, 2011.
[5] https://openwetware.org/wiki/Example_page_from_VCCRI_BIOMOD_wiki
[6] Selnihhin, D. et al., *ACS Nano*, 12(6):5699-5708, 2018.
[7] Thubagere, A. J. et al., *Science*, 357(6356):eaan6558, 2017.
[8] Douglas, S. M., Bachelet, I., and Church, G. M., *Science*, 335(6070):831-834, 2012.
[9] Douglas, S. M. et al., *Nucleic Acids Research*, 37(15):5001-5006, 2009.
[10] Benson, E. et al., *Nature*, 523(7561):441-444, 2015.
[11] Veneziano, R. et al., *Science*, 352(6293):1534-1534, 2016.
[12] Huang, C.-M. et al., https://doi.org/10.1101/2020.05.28.119701, 2020.
[13] Babatunde, B., Cagan, J., and Taylor, R. E., *Applied Sciences*, 2021, **in preparation**.
[14] Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P., *Science*, 220(4598):671-680, 1983.
[15] Gavali, P. and Banu, S., *Academic Press*, 99-122, 2019, https://doi.org/10.1016/C2018-0-00906-8.
[16] Wu, H. and Prasad, S., *IEEE Transactions on Image Processing*, 27(3):1259-1270, 2018.
[17] Padmanabhan, S., *Stanford University*, 1-8, 2016. https://web.stanford.edu/class/cs231a/prev_projects_2016/example_paper.pdf
[18] Dwivedi, R., *Analytics India Magazine*, 1, 2020, https://analyticsindiamag.com/everything-you-should-know-about-dropouts-and-batchnormalization-in-cnn/.
[19] Basaveswara, S., K., *Towards Data Science*, 1, 2019, https://towardsdatascience.com/cnn-architectures-a-deep-dive-a99441d18049