The Measure of Intelligence: Abstract and Reasoning

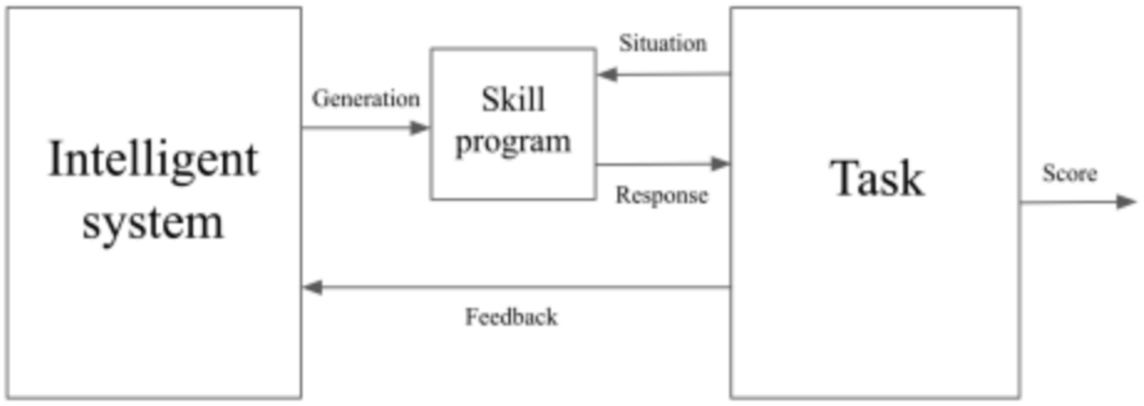
Kshitiz Malhotra

Introduction

François Chollet's paper "On the Measure of Intelligence" [1] weighs in on the crucial need for a concrete definition of Intelligence and an explicit goal to measure it, not just anecdotes. According to François, Intelligence is defined as a skill-acquisition efficiency over a scope of tasks with respect to:

- **Priors** - the information stored in the neural network structure, data augmentation applied to the given data, Hyperparameters like learning rate of the model, no. of neural network layers and any other human bias.
- **Experience** - How much data the system is learning from
- **Generalization difficulty -** The level of change a system can withstand(or still perform the same way) after the domain on which it was trained on is changed(from somewhat changed to radically changed).

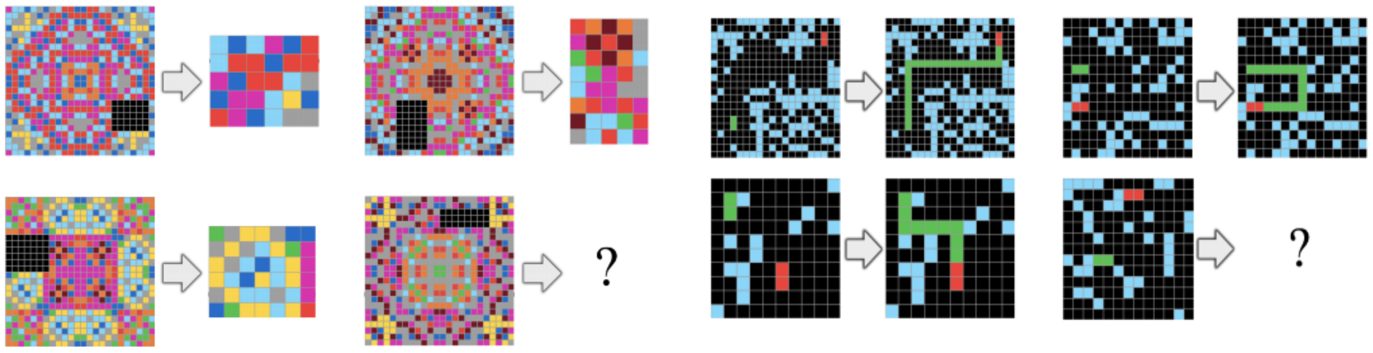
The ideal way for a system to acquire a skill or to be intelligent is shown in the relationship below, which is a constant response and feedback loop with a particular task until the skills acquired are enough to make generalizations. The skill program is the product of the intelligent system, which is the algorithm for this ability. The skills interact with the task in a framework that gives the state, makes a response and gets a reward(sort of like reinforcement learning, MDP). After getting a response score, the task sends a feedback back to the intelligent system.



Chollet describes several ways to benchmark an intelligent system. He points out that intelligence evaluating tests and their benchmarks such as the Turing Test[2], have implicit definitions of intelligence that are loaded with biases and thus can be distracting from truly understanding and generalization, which is the essence of intelligence.
To accurately measure the skill-acquisition ability over the aforementioned three scopes, François proposed ARC (Abstraction and Reasoning Corpus) as a challenge.

Abstraction and Reasoning Corpus

The dataset contains unique transformation and rule based problems, like the ones we usually see in a standardized IQ test. It Contains 1000 manually labelled sets of unique problems, where there is some prior knowledge in 3-4 related rule based transformations which is enough to generalize to a similar task. Some of the related examples are shown below



If an AI is shown an example of how to rightly complete a jigsaw-puzzle by finding the underlying pattern in the demonstration's prior knowledge, it could adopt a reasoning as to how to complete the given puzzle correctly. This dataset also imposes a big challenge in the AI community with the constraint that only a handful of demonstrations are shown to learn the rules of transformation of unique tasks. Whereas the state-of-the art machine learning models known today are known to even outperform humans but the reason is big data.

This provides a glimpse of a future where AI could quickly learn to solve new problems based on prior knowledge extracted from very few examples. Therefore, it is regarded AS one of the very few benchmarks of AGI

Method Approaches

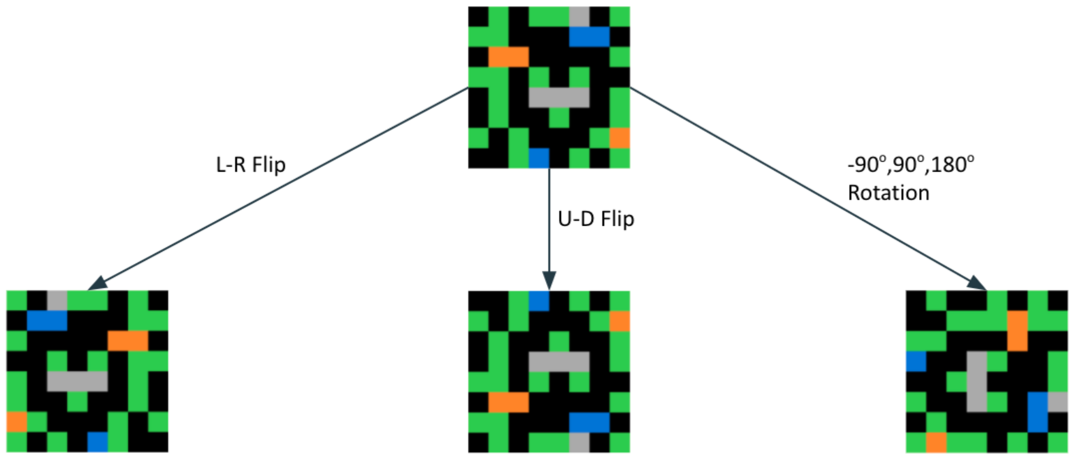
I started to approach this problem through two main learning based approaches.

Boosted Decision Trees. A Decision tree is a powerful tool to discover interaction among independent features, which here are the pixels in the grid of a problem task. Pixels that appear together in a grid are interacting with one another in the transformations. For example, certain black pixels are connected to certain green pixels and that's true even after various transformations. Certain modifications on Decisions trees result in better predictive performance when there is feature interaction. We will cover feature interaction in the upcoming sections. Decision Trees are also known for lessy noisy and more generalizable predictions on a smaller dataset with less classes, which is exactly the case with ARC. Decision trees are also sparsity aware algorithms that can learn to fill the missing values by finding sparsity patterns. Our task features will have a lot of missing values as a result of a special feature engineering. I tried different variants of boosted decision trees algorithms for better performance.

Convolutional Neural Networks. This was my first attempt on solving ARC with a learning based method. I converted the raw grid-cell data into RGB images and inputted them to a very small convolutional neural network with just two convolutional layers and a maxpool flattening layer. There were few trainable parameters but there were far lesser examples per task, even after some data augmentation. So although this method was relatively easy to set, it did not perform well on ARC due to instant overfitting.

Data Pre-Processing

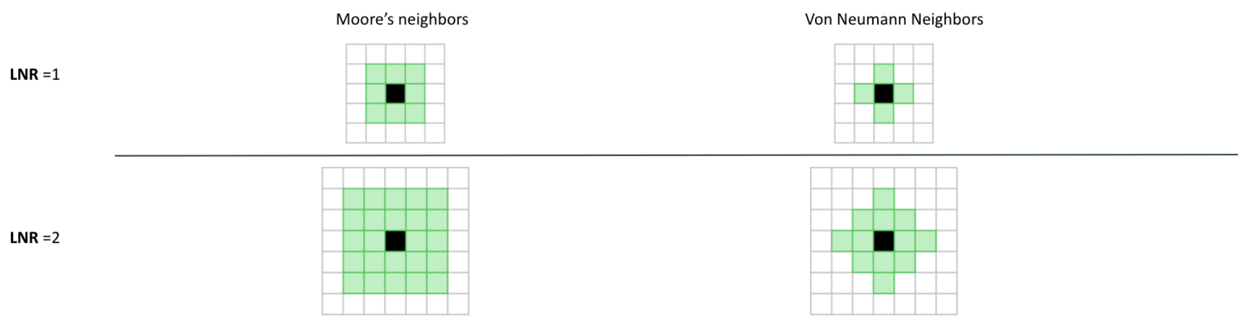
Since the CNN method failed, I had to proceed with the boosted decision trees method. In order to make the data processesable, I first converted the raw JSON files into 2-D grid using plotting functions. I also numerically encoded the colors so that the pixel values in a 2D matrix could be represented numerically. Then I flattened the 2-D matrix to a 1D vector to deal with variable size of grids and to create more training samples because now each pixel in a flattened vector is a training data point.
Now Perhaps the most important preprocessing step was data augmentation.



Creating synthetic data through data augmentation was essential since the tasks already had very few examples to learn from. The only type of data augmentation I could do was geometric transformation that is flipping the image left to right, up to down and rotating the image with an angle of 90^{o}. Since the colors were very essential to the tasks, I could not do any color based data augmentation like changing the saturation, hue etc. After all of the geometric transformations, the pixel's relation to its neighbor did not change and this is something I exploit in my feature engineering.

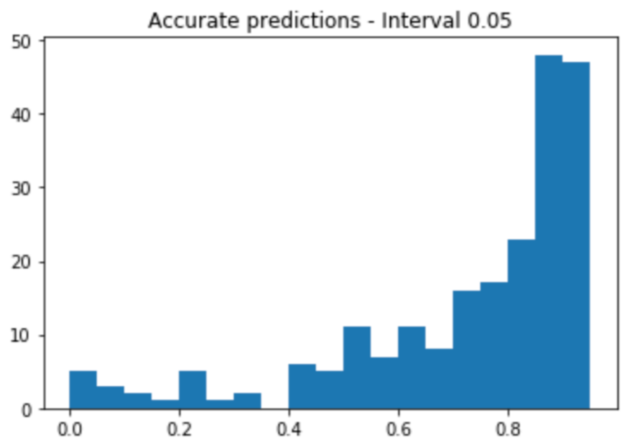
Feature Engineering

This was the most crucial part of the entire learning process as it helps build up a local awareness per pixel that is very useful for decision trees. So the neighboring pixels of a training pixel on a 2-D array are the features. As we have discussed before, decision trees take advantage of the neighboring feature interactions, so any change or no change in the local neighborhood of the pixels will be learned efficiently. The neighbor finding algorithms were inspired from Moore's neighborhood[3] and Von Neumann Neighborhood[4] image processing algorithms. Morse neighbors are all the surrounding neighbors of the current training point pixel and Von Nueman neighbors are only the above, down left and right neighbors.
Now for the training pixel on the edge, there will be a lot of missing values because we have a fixed length feature vector but there aren't many features on the corners or the edges. So we had to pad the missing values. This is where the decision tree algorithm shines because it is a sparsity awareness algorithm.
We also defined a parameter called Local neighbor range(LNR) which specifies how deep we want our neighbors to be. Do we just want to consider our neighbors or also the neighbors's neighbors and so on.



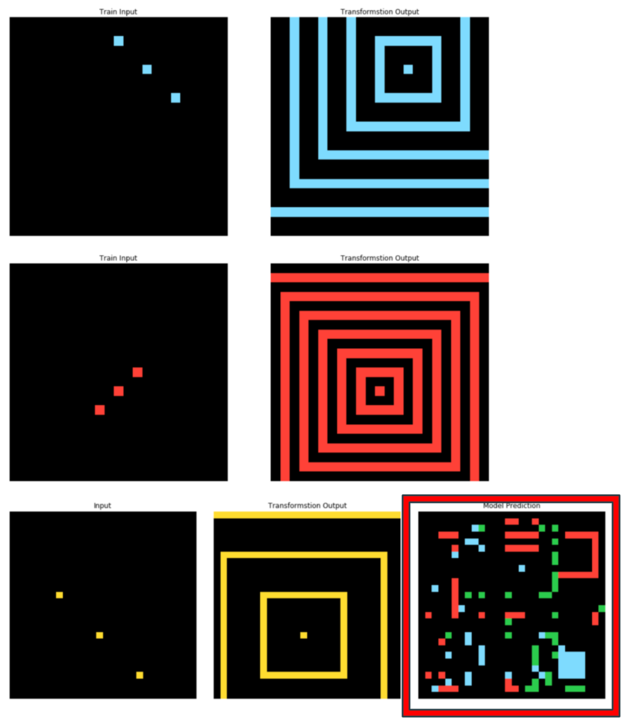
Experiments and Evaluation

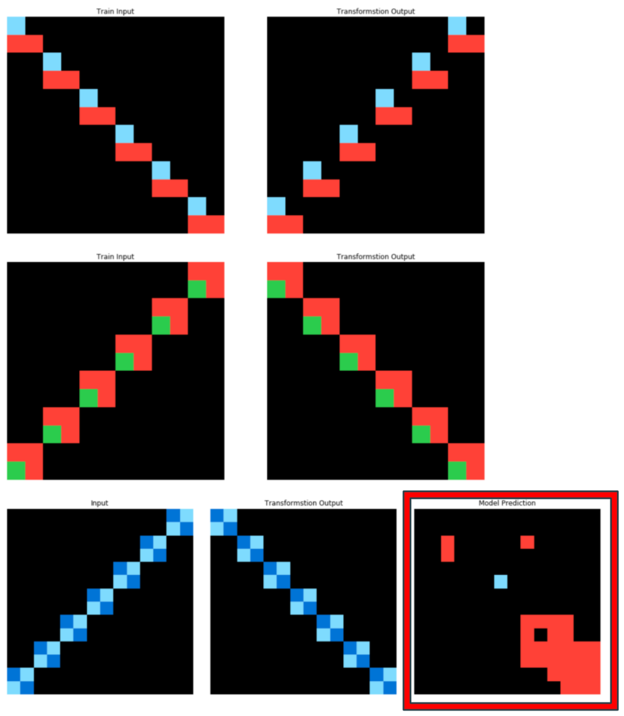
I trained my model using a technique called stacking, which combines the strength of each individual estimator by using their output as input of a final estimator to get a slight prediction boost. I used several boosting methods like XGBoost, Extra-Trees and Random Forest Algorithms. For hyperparameter tuning, I only included a few parameters like the No. of estimators in each algorithm, Tree-Depth, Loss(for gradient-boosting) and LNR(only 1 and 2).
For evaluation, accuracy was directly pixel-to-pixel comparison in the grid ground truth and the grid predicted output. Pixel values are the color types ranging from 0-9. If all the pixels in the predicted output match the pixels in the ground truth, it was a correct prediction. If 85% - 99% pixels matched, it was almost correct and less than 85% were Incorrect prediction. The histogram below shows the distribution of accuracies of all these predictions within an interval of 5%.



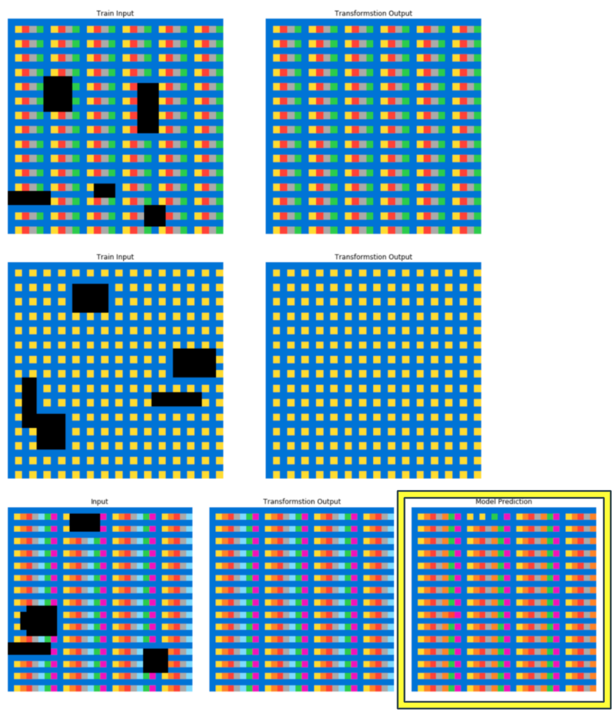
The result above is from the best model obtained from hyper parameter tuning. There were about 33/300 right evaluations with 100% accuracy. This is not a bad result. For a reference, the best result on the ARC dataset was from a DSL technique[5] with 87/300 right predictions. Below are some visual results

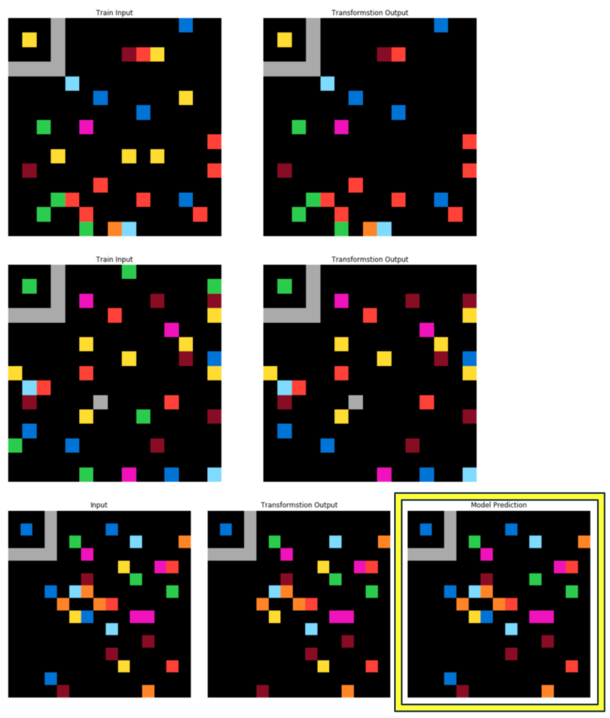
Incorrect Predicitons



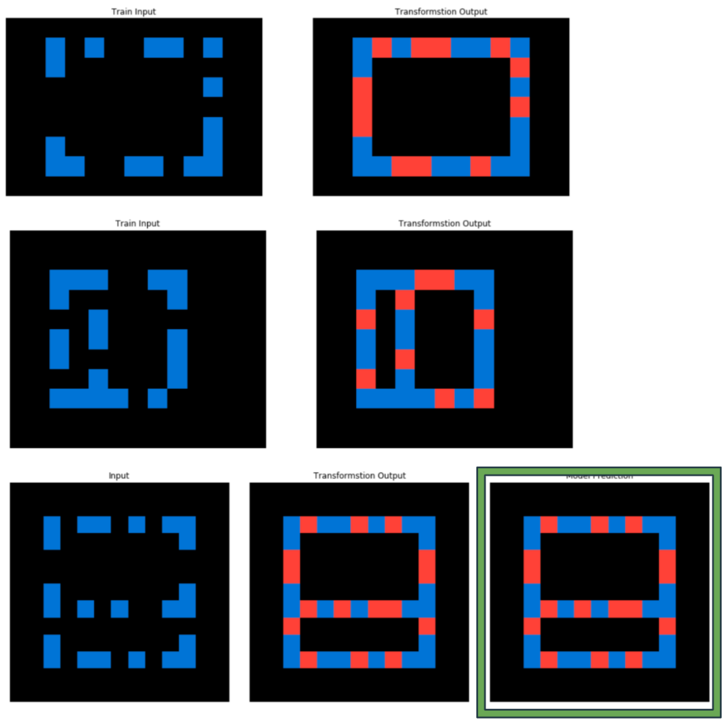


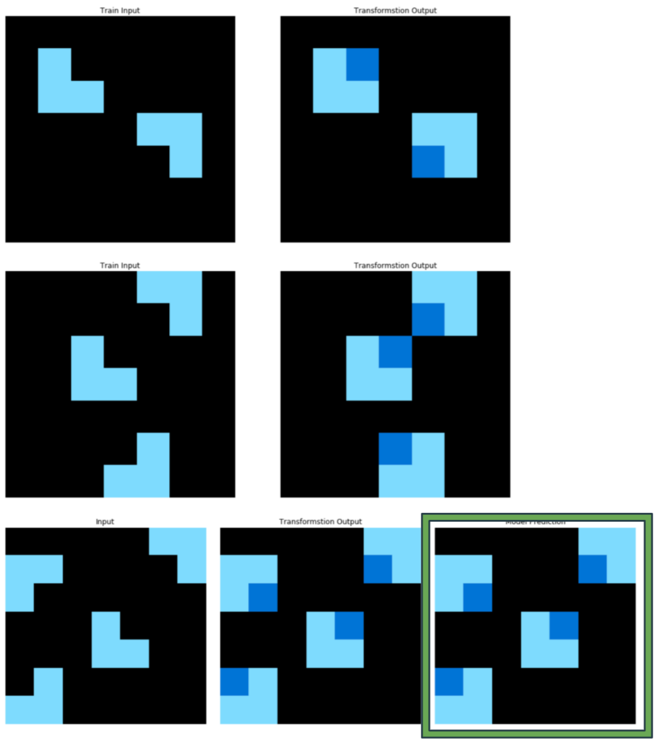
Almost Correct Predictions

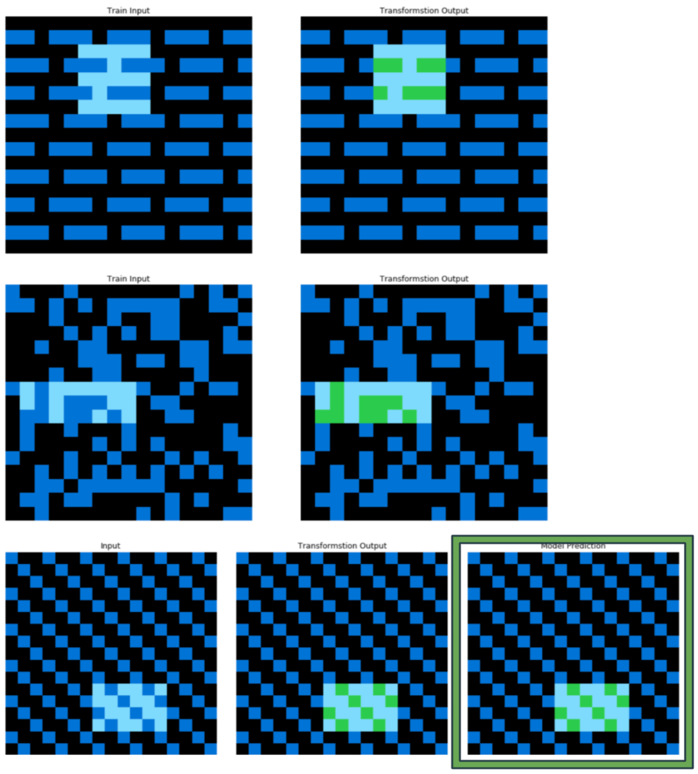


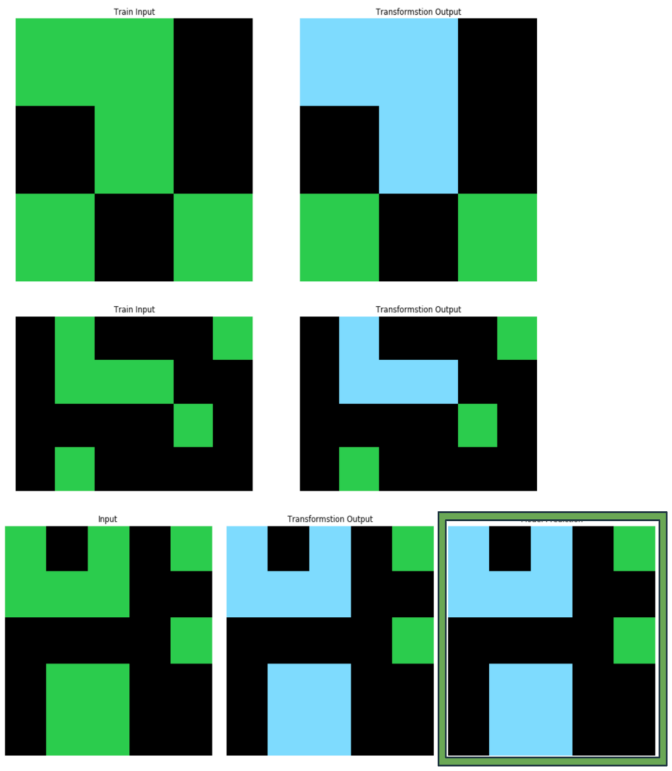


Correct Predictions









Observation and Discussion

There were many key observations from these experiments. We observe that Boosted decision trees can find underlying patterns on task but only locally, since it only considers the training pixel's local neighboring pixels as features. Therefore they're not so capable of solving tasks with complex transformations that require some sort of global information. This was pretty evident in the visual results where the tasks solved by Decision trees were relatively easier than the ones they could not solve. With the Conv-net experiment failure, we got to know that A good statistical model fits at its best to a specific domain of tasks, having similar underlying patterns. But AGI needs the opposite. And perhaps the most important observation, the output of the best solution is not really learned but engineered. The best solution with DSL[5] has all the different mutations of grid transformations manually coded. Then the genetic algorithm finds the best sequence. Same goes for Decision trees with the neighbor selection algorithm where we basically tell the algorithm how to get a local awareness.

References

[1] Chollet, François. "On the measure of intelligence." *arXiv preprint arXiv:1911.01547* (2019).
[2] A. M. TURING, I.—COMPUTING MACHINERY AND INTELLIGENCE, *Mind*, Volume LIX, Issue 236, October 1950, Pages 433–460, https://doi.org/10.1093/mind/LIX.236.433
[3] Liu, T., Moore, A. W., Gray, A. G., & Yang, K. (2004, December). An investigation of practical approximate nearest neighbor algorithms. In *NIPS* (Vol. 12, p. 2004).
[4] Tanimoto, Naonori, and Katsunobu Imai. "A Characterization of von Neumann Neighbor Number-Conserving Cellular Automata." *Journal of Cellular Automata* 4.1 (2009).
[5] Visser, Eelco. "WebDSL: A case study in domain-specific language engineering." *International summer school on generative and transformational techniques in software engineering*. Springer, Berlin, Heidelberg, 2007.