Abstraction and Reasoning Challenge

Nikhil Gupta Divyansh Kumar Roy Amal Syriac
Georgia Institute of Technology
{ngupta332, droy41, asyriac3}@gatech.edu

Abstract

Abstraction and Reasoning Challenge[1] comprises of a set of tasks which aim to develop an AI system that develops general intelligence as opposed to task-specific intelligence. In our work we examine the effectiveness and demerits of using various conventional Neural network (NN) architectures in solving these tasks. We analyzed situations in which NN can work better and solved approximately 30% of tasks in our test dataset.

1. Introduction, Background and Motivation

1. Our expectations from Artificial Intelligence (AI) has increased over time with the pursuit to create systems that can truly work independently with minimal supervision from humans. Even though AI are able to perform a large number of tasks which are hard even for humans, they lack a notion of general intelligence. Through the course of this project, we want to explore the ability of neural networks to solve unseen and unique logical reasoning tasks.

2. Many state of the art models are exceptionally good at doing a very specific task but they seem to lack an Artificial General Intelligence which can adapt to different unseen scenarios.

3. This challenge presents a neat way to train systems for developing skills necessary for general intelligence. Any advancements in this area would have wide ranging implications on the development of machines which possess a set of cognitive abilities and use advanced logic and reasoning which is common in humans.

4. The author of this challenge states that current Machine and Deep learning techniques would be unsuccessful in building a single model that could solve all these tasks because of its variety. Since, the challenge is fairly recent, there is no published work present on this. Our aim is to test if we can come up with architectures that can tackle variety of tasks in the dataset.

2. Data

1. The dataset provided in the Kaggle challenge contains 400 tasks for training and 400 tasks for evaluation. Within each task, there are typically 3 training pairs containing input and output image and one test pair for which output is not given. The task is to learn the logical reasoning used in training pairs and then to apply it to test pair to get the output pair.

2. Each example is a json file containing an input and output pair. Each of these are a grid of values ranging from 0 to 9, with each value representing a color.

3. Given the difficulty of the task, we decided to modify the dataset to reduce its difficulty by focusing on task in which output grid size remains equal to input grid size. Each task will require understanding of the logic used to determine which cell needs to be given which color. We also decided to remove the color aspect from the tasks as 50% of task's reasoning do not depend on color.

4. In our new modified dataset, we have 59 tasks in the training dataset and 43 tasks in our output dataset. Sample train and test pair are shown in figure 1 and 2 respectively. The red and black cells are referred as ON and OFF cells respectively.

3. Approach

We noticed that the examples provided could be roughly divided into 3 different main categories:

1. **Substitution:** These tasks involve the whole or a part of the input pattern being substituted by another in the output.

2. **Replication:** In these type of tasks, the output involves replicating a portion of the input pattern multiple times in the output.

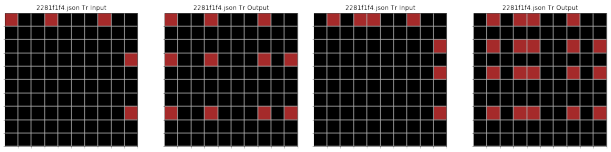

Figure 1. Training pairs

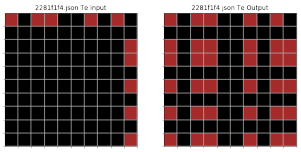

Figure 2. Test Pair

3. **Translation:** Here, a part of the input pattern is moved along specific directions to obtain the output pattern.

There are examples that did not fit into these categories, but we decided to focus on solving the above mentioned categories first as they were dominant in the dataset. These categories can be identified using simple heuristics which is mentioned in the Complete Pipeline section. After category detection, we need to find approaches to solve that task. The approaches we have explored are listed below:

1. **Neural Network based approach**: For each task category (substitution, replication or translation), we trained the pairs on various neural network architectures whose details are mentioned in the Experiments section, and evaluated the performance.

   One issue that we faced was that while our output pattern had the same grid size as the input for one pair, the grid sizes could vary across the pairs for the same task. To account for varying sizes of the inputs, we padded the inputs with zeros such that all input images have the size of the largest input. As we needed the output grid to have the same size as the input, we appropriately adjusted the filter size and the padding in our architecture.

   The various data augmentation techniques and loss functions used are mentioned below.

   (a) **Data Augmentation**
   The limited number of examples for each task within a category was another issue. To overcome this, we used data augmentation techniques[2] so as to increase the number of training examples. Some of the data augmentation techniques that we used include the following:

   i. Since the tasks (substitution/replication/translation) are not affected by rotation, we obtained more examples for a task by flipping an image vertically and horizontally. This gave us 4 different sets of pairs from a given pair.

   ii. Since the ON pixels were encoded as 0 and the OFF pixels as 9, we also tried changing these values to other values so as to obtain more examples as logic is not supposed to depend on exact values.

   (b) **Loss Functions**
   We tried various loss functions for our neural network architectures. These include:

   i. Number of wrong cells:
   Count the number of cells that are incorrectly predicted by the model. There are many issues with this loss function such as it is discontinuous which was not good for training NN. It does not focus on which cells are wrongly predicted.

   ii. Mean Squared Loss:
   This was calculated by taking the sum of squared difference between the pixel values of the predicted image and the actual output image.

   iii. Cross entropy loss:
   Here, we calculated the binary cross entropy loss between the predicted and actual output images. A variant to this is to give different weights to the loss from the cells that are correctly and incorrectly predicted.

2. **Decision Tree based approach:**[3]
   Many tasks can be solved using simple if and else condition which require multiple layer NN to solve because of their linearly inseparable nature. We use the standard sklearn library for this.

4. Experiments

1. **Neural Network Based Approach**
   Every task in the dataset is unique which makes the challenge tough and exciting. The architecture of the Neural Network significantly depends upon the unique nature of the task. Please find below the different type of architectures that we have tried and other relevant information such as why we choose such an architecture, pros and cons of that architecture etc.

   (a) **M Convolution filters and L layer Architecture:**

   i. This architecture has M convolution filters and there are L such layers of them.

   ii. Sigmoid is used as an activation function between the layers for all architectures described here until explicitly mentioned otherwise.

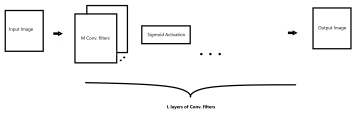

Figure 3. M Conv. filters L layer Architecture

iii. We used sigmoid function because we are interested in the probability of whether a cell is ON or OFF.

iv. Since the size of the input image may vary across the pairs, we use a combination of kernel size, padding and stride such that the size of resulting image is same as the input image.

v. This type of Architecture can be used for several substitution and translation tasks. For instance, task in figure 4 are provided as input to the model, in which M=1, L=1 and filter size=3, and figure 5 is the output predicted correctly by the model. Similar is the situation for another task as given in figure 6 and 7.

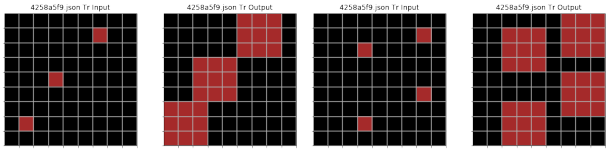

Figure 4. Training pairs

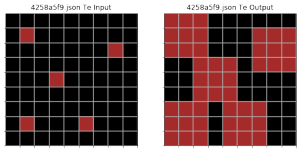

Figure 5. M Conv. L Layer Model Successful Prediction

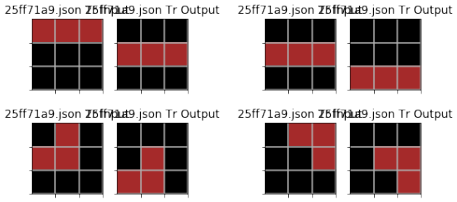

Figure 6. Training pairs

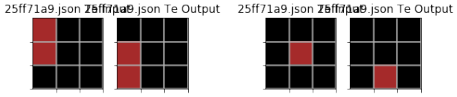

Figure 7. M Conv. L Layer Model Successful Predictions

vi. **Pros:** This kind of architecture is really powerful for cases where a consistent logic is used and that logic depends locally on the neighbourhood. For instance, the task in figure 9 can be very tricky if we want to code the logic for that directly. However, this architecture solved it very elegantly.

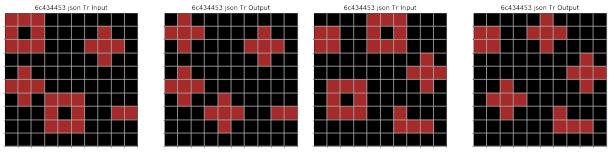

Figure 8. Training pairs

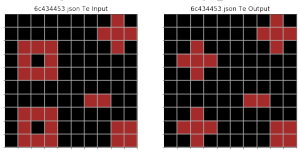

Figure 9. Model Successful Prediction (difficult task)

vii. **Cons:** This kind of architecture is not helpful when logic for solving the task is not confined to the neighbourhood or the logic for solving a specific task depend upon the input pair and is not consistent across pairs.

(b) **Customizable Recurrent Neural Networks:**[4]

i. Some logical tasks can be successfully modeled as language translation problem and can be successfully solved using Recurrent Neural Network.

ii. The neural network can have L layers inside it and each layer could have M convolution filters or fully connected layers or any other architecture.

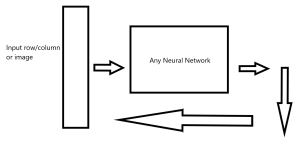

Figure 10. Customizable Recurrent Neural Network

iii. In such tasks, each row or column of the image can be treated as a word vector and to generate the corresponding output row or column, all you need is history and current input. For pairs having different grid dimension, maximum dimension is used as the size of input vector.

iv. As shown in figure 11, each column of input can be treated as vector and the output column depends on what it has seen so far. The model successfully solves the task as seen in

figure 12. M Conv. L Layer Architecture will not work in such scenario.

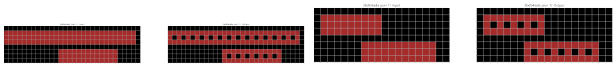

Figure 11. Training pairs

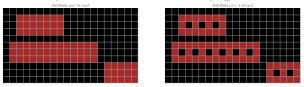

Figure 12. RNN Model Successful Prediction

v. The RNN has two parameters which decide the weights it needs to give to the current input and to the history, thus in the sense are cutomizable.

vi. In figure 13, we can observe that the output column can be uniquely determined from the corresponding input column and do not depend on the history. The RNN model was successfully able to solve this task.

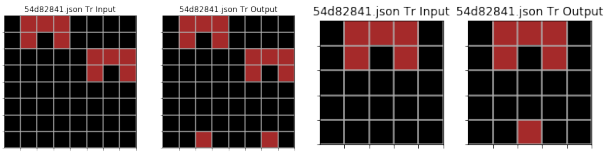

Figure 13. Training pairs

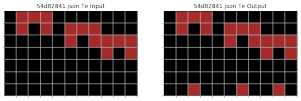

Figure 14. RNN Model Successful Prediction (history weight = 0)

A. In this specific task, L=1 will not be sufficient because output needs to be OFF when their are either 2 or 0 ON cells in that column which makes it linearly inseparable. However, using L=2 solves the task.

B. These kind of tasks also advice that sometimes additional information such as number of ON cells in that column, position of column can be proved useful. For instance, in this task, if number of ON cells are provided, this can be solved with L=1 as well.

vii. **Pros:** Such architectures can be helpful when the logic is not confined in the neighbourhood of the cell but present somewhere else in the image.

viii. **Cons:** Such architecture usually take more time to train because it has to process entire image column/row wise.

(c) **XOR/XNOR Architecture (inspired by LSTM networks[5]):**

i. In some of the tasks, it has been observed that partial input remain as it is in the output and other parts are modified according to some logic.

ii. This can be solved by architecture that can learn which part of the input to keep as it is and which parts need to be modified. We can choose to apply any of the above 2 architectures to learn from the modified parts of the inputs.

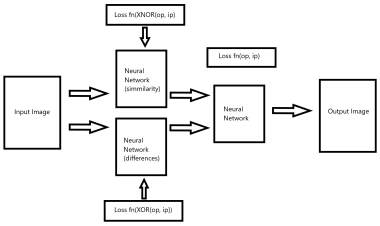

Figure 15. XOR/XNOR Architecture

iii. We are using 3 types of losses in this architecture. Total loss is the sum of all these losses.

A. XNOR Loss: This tries to learn what part of the input should remain as it is. We can get the ground truth information by taking XNOR of input and output image.

B. XOR Loss: This tries to learn what part of the input should change. We can get the ground truth information by taking XOR of input and output image.

C. Remaining Loss: This takes as input the images generated by XOR and XNOR network and tries to predict actual output which is compared with the expected output.

iv. On the task in figure 16 and 17, simple RNN or M Conv. L Layer did not work. But this architecture successfully solved this task most probably because it is easy to learn the difference and similarity individually and then combine both as opposed to learning both under single convolution filter.

(d) **Feed Forward Network:**

i. It has been observed that in some of the tasks, all the ON cells in the input are ON in the output as well. Some additional cells turns ON in the output image.

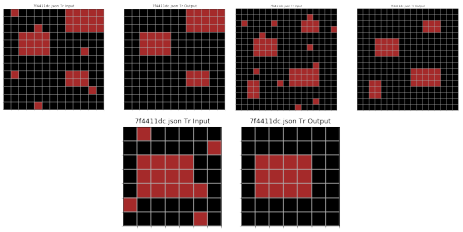

Figure 16. Training pairs

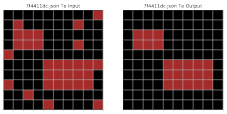

Figure 17. XOR/XNOR Model Successful prediction

ii. The idea was to construct the output as input plus difference where difference will be learned.

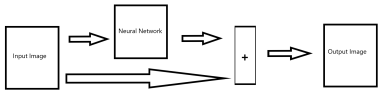

Figure 18. Feed Forward Architecture

iii. On the task in figure 24, feed forward is predicting correctly as shown in figure 20.

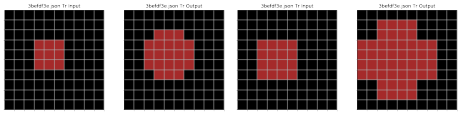

Figure 19. Training pairs

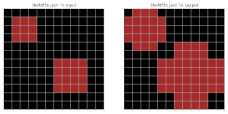

Figure 20. Feed forward Model Successful prediction

(e) **Sparse Fully Connected Neural Networks:** Sparse Neural networks[6] are simple neural networks that use L0 norm and perform well in solving replication tasks such as the one shown in figure 22 and 23.

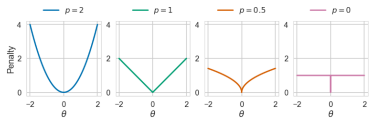

Figure 21. L-p norm penalties for different values of p

In this task the solution involves replicating the contents of the top-left cell in all the other cells of the grid. Since, the object to be replicated changes across pairs, it won't be possible to use convolution filter since there's no common pattern.

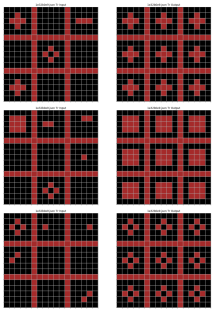

Figure 22. Training pairs

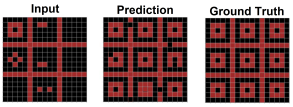

Figure 23. Sparse NN almost successful prediction

Dense neural networks can be useful in this task but they tend to learn weights which are mostly non-zero and every feature in the previous layer contributes some amount to the next layer's neurons.

A possible solution is the use of a dense network which would force a large number of weights to be zero and learn a few but strong relations between 2 layers. Using Sparse Dense networks we saw a huge increase in the training accuracy when compared to a Fully-connected dense layer. We achieved near perfect validation accuracy of 97.23% by pre-processing the input to contain just the top-left cell in the grid.

(f) **Convolutional Neural Networks:** In task shown in figure 24 and 25, using just vanilla CNNs can be enough to solve the task. Data augmentation also helped in this case.

2. **Decision Tree Based Approach:** The task given in figure 26 and 27 requires multi-layer architecture of neural network because output is not linearly separable. However, it can be easily solved by decision tree (DT). The decision tree is passed each cell as a tuple of 4 values - rowId, columnId, gridHeight, gridWidth.

5. Complete Pipeline to solve tasks

Given an input task, we will perform the following steps to solve the tasks.

1. **Category Detection**

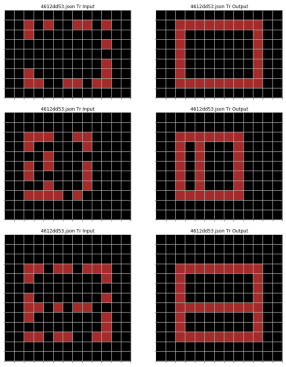

Figure 24. Training pairs

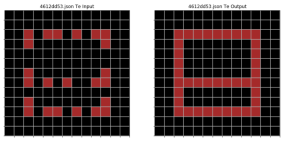

Figure 25. Testing pairs

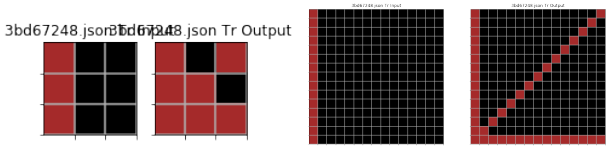

Figure 26. Training pairs

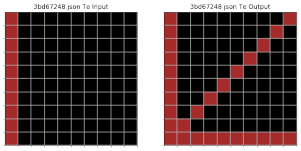

Figure 27. DT model Successful prediction

(a) We are using the heuristics to create an order in which various architectures are tried.

(b) We define all ON cells adjacent to each other as an object. The task is assumed to be substitution if number of objects in the input image are same but individual object differs.

(c) If output image consists more number of instances of an object than input image, it can be assumed as an Replication task.

(d) The task is assumed to be translation if number of instances of most objects are same in both input and output image.

(e) For replication task, Sparse Fully Connected Neural Network are preferred. For translation and substitution, all other architectures discussed such as M Conv. L Layer, RNN, Feed Forward, XOR/XNOR could work. If majority of ON cells remain ON in output, then XOR/XNOR structure may be preferred. Otherwise, M Conv. L Layer and RNN are first choices. Similarly, for an architecture, we may want to try different values of kernel size etc.

2. **Task Solver**

(a) Once we have an idea of the category of the task, we will go through all architecture in a particular order governed by category and some heuristics as mentioned above.

(b) We have developed some more heuristics such that if all training examples are successfully solved but model fails on test example, one possibility may be that the required pattern is not visible during training. Then we might want to augment the data and retry.

(c) We assume that if a cell has a probability of 0.5 or more, it would be interpreted as ON. It has also been observed that sometimes neural network tries to reduce the loss by pushing the probability close to 1 or 0. This might not help as a cell having probability of greater than 0.5 will be assumed to be ON. So, focusing on increasing it probability to close to 1 can shift focus from cells which are predicted incorrectly. To tackle this issue, we change the loss function such that all cells which are correctly predicted will not contribute towards loss.

6. Results

(a) We refer to accuracy as the number of correct pixels in the prediction when compared to the ground truth output. We also use an additional performance metric called success which denotes if the model achieves 100% accuracy on the test pair or not.

(b) **Training Dataset:** The following table shows how many tasks under a category are successfully solved by some of the architecture.

Category	Solved	Total	%Solved Successfully
Substitution	14	20	70%
Replication	4	21	20%
Translation	2	10	20%
Others	1	8	12%
Total	21	59	36%

The following table shows how many tasks are successfully solved by a specific architecture.

Architecture	Solved	%Solved Successfully
M Conv. L Layer	10	17%
XOR/XNOR	3	5%
RNN	2	3%
Sparse FC NN	2	3%
CNN	2	3%
Decision Tree	1	1%

(c) **Testing Dataset:** The following table shows how many tasks under a category are successfully solved by some of the architecture.

Category	Solved	Total	%Solved Successfully
Substitution	8	15	53%
Replication	2	16	20%
Translation	1	4	25%
Others	1	8	12%
Total	12	43	28%

The following table shows how many tasks are successfully solved by a specific architecture.

Architecture	Solved	%Solved Successfully
M Conv. L Layer	6	14%
RNN	2	5%
Sparse FC NN	2	5%
CNN	2	3%
XOR/XNOR	1	2%
Decision Tree	1	2%

7. Result Analysis

1. As expected and evident from the results, the substitution tasks performed well when the logic required to solved the tasks is confined to the neighbourhood. If logic is not confined spatially, then RNN, XOR/XNOR architectures can be used to solve the tasks.

2. Replication tasks and Translation tasks are among the most difficult task for the neural networks. The primary reason for this is attention. There was too few pairs to train an attention mechanism. Without an attention mechanism, it is supposed to difficult for Neural Network to find the right object that needs to be replicated or to find the logic of how much to translate.

3. Some of the tasks can be easily solved by humans by writing simple if and else statements. That's why we chose to use Decision Tree because they can learn decisions which are not linearly separable.

4. Please find some of the tasks solved by our Neural Networks in figure 28 & 29, 30 & 31, 32 & 33 and 34 & 35.

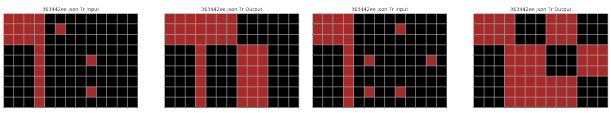

Figure 28. Training pairs

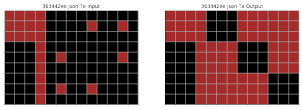

Figure 29. M Conv. L Layer Model Successful Prediction

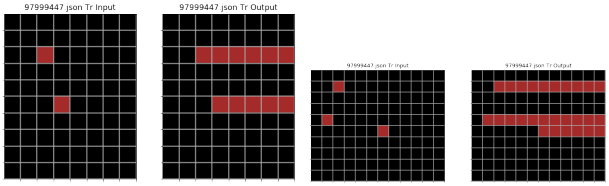

Figure 30. Training pairs

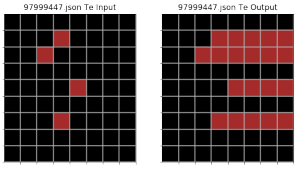

Figure 31. Recurrent CNN Model Successful Prediction

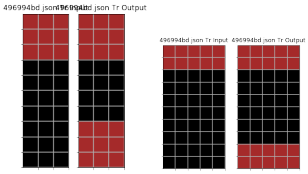

Figure 32. Training pairs

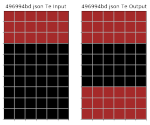

Figure 33. Sparse FC Model Successful Prediction

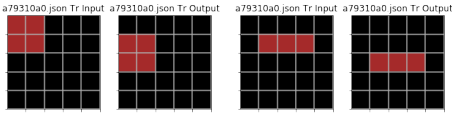

Figure 34. Training pairs

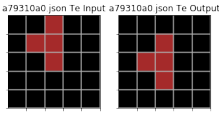

Figure 35. M Conv. L Layer Successful Prediction

5. The table summarises the hyper-parameters used to solve the task. LR stands for learning rate and Arch. for Architecture used.

Task	Arch.	Epochs	LR	Optimizer
28 & 29	M Conv. L	500	0.01	SGD
30 & 31	RNN	1000	0.001	SGD
32 & 33	Sparse FC	1000	0.001	Adam
34 & 35	M Conv. L	500	0.01	SGD

8. Challenges encountered in Neural Network And Decision Tree based approach

We found following challenges while solving the logical tasks using Neural Network or Decision Tree based approach.

1. In these tasks, we want 100% accuracy i.e. we want each cell of the output grid to be exact match.

2. Some pattern required to solve the task are not present in the training data. This can be mitigated with data augmentation but cannot be eliminated completely.

3. Picking right size of convolution filter can be tricky. If its too large, it can simply memorize the input and output pairs. If two small, it will not be able to solve. This can be mitigated by trying out a range of sizes that seem to be working in input set.

4. If input pairs has too many random patterns it will trouble NN even if the logic to convert those random figure into regular shape is consistent.

5. Sometimes trying to learn the output image directly in one time step is very difficult. Thus, RNN can be used for such cases but these scenarios can be difficult to detect.

6. Also sometimes its better to learn some skills and then to try to learn the output image as opposed to directly learning it. But it is very difficult to come up with the loss function that cause network to learn those skill even if the architecture is designed to facilitate that. The simple reason for that, network can go towards any local minimum which does not represent learning those skills but reduces loss function.

7. The biggest challenge is that the possible hyper parameters and architecture to try increase exponentially. Thus, it is critical to design heuristics to pick right set of hyper parameter and architecture depening upon a task.

9. Other potential Approaches

1. **Genetic Neural Network based Approach**
   One of the biggest limitation of neural network is they cannot create new neural architectures. For these type of tasks, it is essential to develop new functions or architecture so that unseen task can be solved. A potential step towards this could be to combine Genetic Algorithms[7, 8] that uses natural selection along with neural networks.

10. Conclusion

Through the course of this project, we try to use various architecture that can solve logical reasoning task which human can solve easily. We presented various architecture and the rationale for their working, pros and cons.

We found that some tasks can be easily solved by Neural networks which can be tricky if we start to code them. One such example was in figure 9. We achieved the success rate of about 30% on our test dataset.

However, due to limited data (2-3 training pairs) and a wide variety logical reasoning tasks, it is difficult to come up with a finite set of architectures that can solve majority of tasks. Thus, there is need for a way to be able to generate new functions and architecture along the way.

11. Work Division

The following table summarizes the contribution of each team member.

Student Name	Contribution Details
Nikhil Gupta	50% Starter Code for training NN, Applying and analyzing M Conv. L Layer, RNN, XOR/XNOR, Feed Forward and decision tree models to solve substitution tasks. Contributed equally towards report.
Divyansh Kumar Roy	50% Starter Code for training NN , applying and analyzing Sparse Fully Connected NN, CNN Feed forward models to solve replication tasks. Contributed equally towards report.
Amal Syriac	Data Scrapping and providing visualization code, Preparation of new dataset (test and train), applying and analyzing M Conv. L Layer, RNN, CNN to solve Translation tasks. Contributed equally towards report

Overall all members of the team have contributed equally towards the project.

References

[1] François Chollet. *On the Measure of Intelligence*. 2019. arXiv: 1911.01547 [cs.AI].

[2] Xiaodong Cui, Vaibhava Goel, and Brian Kingsbury. “Data augmentation for deep neural network acoustic modeling”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23.9 (2015), pp. 1469–1477.

[3] Guolin Ke et al. “Lightgbm: A highly efficient gradient boosting decision tree”. In: *Advances in neural information processing systems*. 2. 2017, pp. 3146–3154.

[4] Tomáš Mikolov et al. “Recurrent neural network based language model”. In: *Eleventh annual conference of the international speech communication association*. 2010.

[5] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[6] Christos Louizos et.al. *Learning Sparse Neural Networks through L0 Regularization*. 2019. arXiv: 1712.01312 [cs.LG].

[7] Emilio Parisotto et al. “Neuro-Symbolic Program Synthesis”. In: *CoRR* abs/1611.01855 (2016). arXiv: 1611.01855. URL: http://arxiv.org/abs/1611.01855.

[8] Neel Kant. “Recent Advances in Neural Program Synthesis”. In: *CoRR* abs/1802.02353 (2018). arXiv: 1802.02353. URL: http://arxiv.org/abs/1802.02353.