Advanced Machine Learning
FINAL PROJECT
ABSTRACT REASONING CHALLENGE

Giorgio Zannini Quirini, Giulia Scikibu Maravalli, Egon Ferri

MAY 2020

Abstract

What is intelligence? Trying to synthesize the common sense, a suitable definitions could be: "Intelligence is the ability of an agent to achieve goals (being highly skilled) in a wide range of environments (being able to generalize)." If we assume this definition, seems clear that what is commonly known as an Artificial Intelligence, is today only capable to match the first part of this definition, not being able to generalize. This is the reason that lead Francois Chollet to set the Abstract Reasoning Challenge, giving a data set and a benchmark to measure the power of a model to generalize knowledge from many simple tasks. But what are the limits of the state of the art deep-learning in handling a task of this type? The aim of our work will be to try to answer this question in a manner as clear and complete as possible.

Contents

1 Introduction

1.1 The measure of intelligence

What is intelligence? An apparently simple question hides a whole universe of answers, ideas, and perspectives. In the paper that has given birth to the competition that we will tackle in our project (1), François Chollet tries to investigate this problem, to inspect what intelligence is, and what repercussions can this have over what we think today of AIs.

A good summary into a single statement is given by Legg and Hutter 2007 (2): "Intelligence measures an agent's ability to achieve goals in a wide range of environments." In this definition we can find the two key points of the common belif on intelligence: emphasis on task-specific skill ("achieving goals"), and focus on generality and adaptation ("in a wide range of environments").

Given this general definition, could we really say that modern artificial intelligence is, indeed, intelligence? Renè Descartes said in 1637: "Even though such machines might do some things as well as we do them, or perhaps even better, they would inevitably fail in others, which would reveal they were acting not through understanding, but only from the disposition of their organs."

Today it's obvious that an artificial static chess player based on mini-max and tree search is not informative about the complex human mind nor capable to challenge the humans in any other field, but it was not only 50 years ago, when it was believed that chess-playing captured and required the whole complexity of the brain. Maybe is still not so trivial that in 2020 the methods to solve complex video games still follows the same patterns.

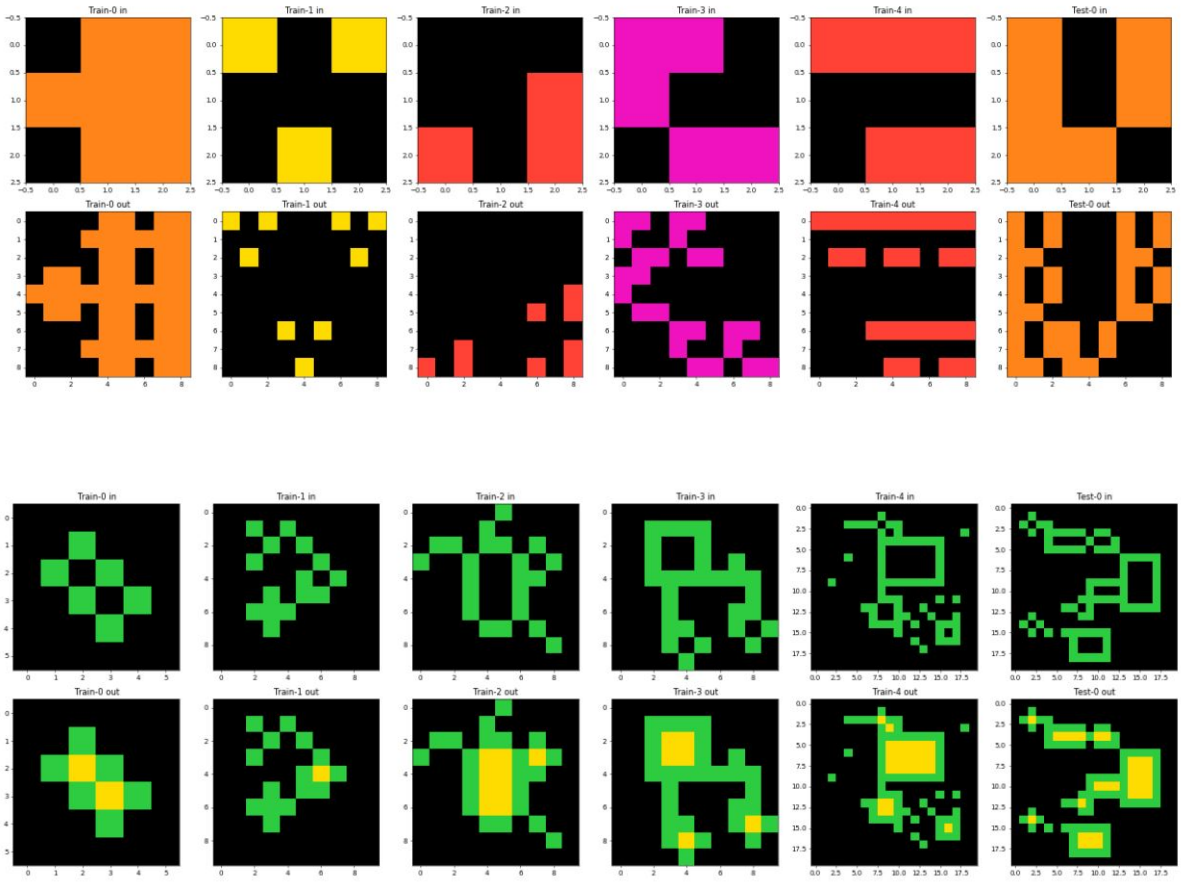
The more modern AIs are extremely powerful when it comes to the first part of what is intelligence to us, bringing "skill" in specific tasks to incredible peaks, but they are still very weak in generalization: "they have still stark limitations, being brittle, data-hungry, unable to make sense of situations that deviate slightly from their training data or the assumptions of their creators, and unable to re-purpose themselves to deal with novel tasks without significant involvement from human researchers."(1)

1.2 Arc: Create an AI capable of solving reasoning tasks it has never seen before

To give a direction to studies on general intelligence, we need a dataset to use as benchmark, so the author provides the Abstraction and Reasoning Corpus. "ARC can be seen as a general artificial intelligence benchmark, as a program synthesis benchmark, or as a psychometric intelligence test. It is targeted at both humans and artificially intelligent systems that aim at emulating a human-like form of general fluid intelligence"(1).

The dataset is composed by a training set and an evaluation set. The training set has 400 tasks, while the evaluation set features 400 tasks. Each task consists of a small number of demonstration examples (3.3 on average), and a small number of test examples (generally 1, although it may be 2 or 3 in rare cases). Each example consists of an "input grid" and an "output grid". Each "grid" is a literal grid of symbols (each symbol is typically visualized via a unique color). There are 10 unique symbols (or colors). A grid can be any height or width between 1x1 and 30x30.

Some examples:



A typical human is able to solve all of these tasks without specific training. The tasks of this dataset are built on the assumption of some knowledge priors(1):

- Objectness priors:
  Ability to parse pieces of the grid into "object" according some continuity criteria like color or spatial contiguity.

- Goal-directedness prior:
  even if ARC does not make use of the concept of time, many of the input/output grids can be effectively as being the starting and end states of an intentional process.

- Numbers and counting priors:
  Many ARC tasks involve counting or sorting objects, comparing numbers or sizes.
- Basic geometry and topology priors:
  ARC tasks feature a range of elementary geometry and topology concepts, like lines, rectangular shapes, symmetries, rotations, translations, scalings, containing, being contained, drawing lines, connecting points...

1.3 The focus of our work

The author of the paper states that this problem is not suited for the deep learning, at least in the sense in which is commonly known today, 2020. But is this really true? And if it is, why, and how? The scope of our project will be to try to use models of the state-of-the-art to get some insights: what will happen when why use some basic networks on all the task without sharing learning? What changes when we build infinite trainset for a task? After answering this questions and more, we hope to have a clear and well documented idea of the limits of deep learning in this challenge.

2 Related work

Since this challenge aims to explore a completely new field, it is hard to find papers and works that can fit our task in a large sense. The only similar experiment is been made by David G.T. Barrett et al. (3), but it's slightly different because they aim to predict the third table of a sequence. So, apart from the main article written by Chollet (1), we used model-specific sources for the model that we used; to describe them is not the purpose of this experiment, and a complete list will be provided in the sources.

3 Proposed method explained

3.1 Main difficulties

As said, the task are extremely different from each other. They not only differ in "scope" and size, but also from the relation of the size between the input and the output. To understand this better we divided them in macro-groups based on this relation:

```
------------ train  shapes ------------
t1 all inputs are equal:  210 of which:
t1_1 also output equal:  134
t1_2 output smaller but fixed:  43
t1_3 output bigger but fixed:  19
t1_4 output size depends on input 14
t2 input different:  190 of which:
t2_1 output equal to input:  128
t2_2 output smaller :  50
t2_3 output bigger :  12
```

```
------------ eval  shapes ------------
t1 all inputs are equal:  157 of which:
t1_1 also output equal:  92
t1_2 output smaller but fixed:  32
t1_3 output bigger but fixed:  20
t1_4 output size depends on input 13
t2 input different:  243 of which:
t2_1 output equal to input:  178
t2_2 output smaller :  51
t2_3 output bigger :  14
```

```
------------ test  shapes ------------
t1 all inputs are equal:  41 of which:
t1_1 also output equal:  23
t1_2 output smaller but fixed:  8
t1_3 output bigger but fixed:  7
t1_4 output size depends on input 3
t2 input different:  59 of which:
t2_1 output equal to input:  42
t2_2 output smaller :  15
t2_3 output bigger :  2
```

We can see that in each subset, in the majority of task the input has the same shape of the output, but in some cases ($T1.1$) the shape is the same across all the inputs, in other cases the shape changes between each input-output pair ($T2.2$).

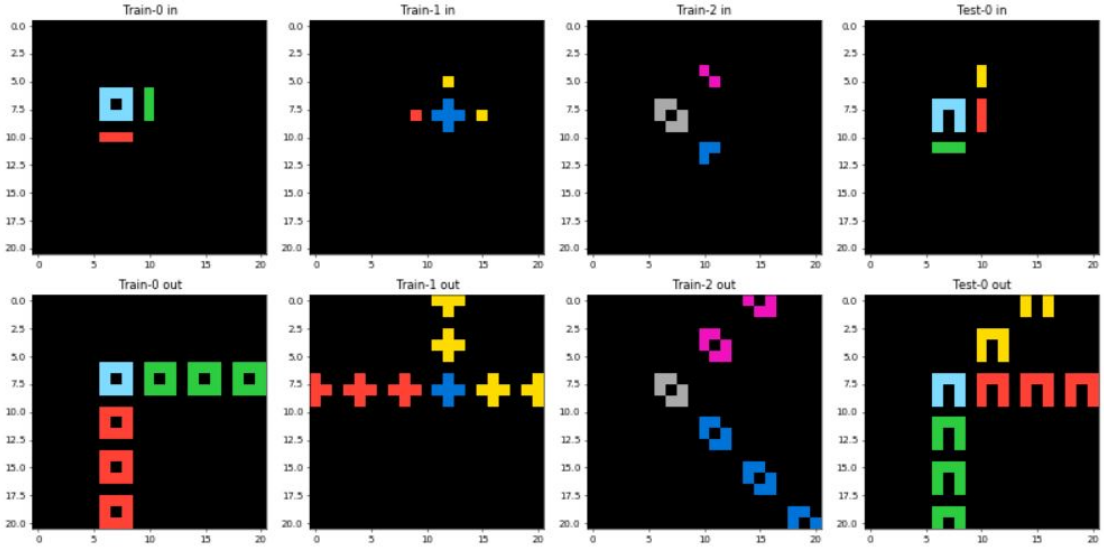


Figure 1: T1.1 example

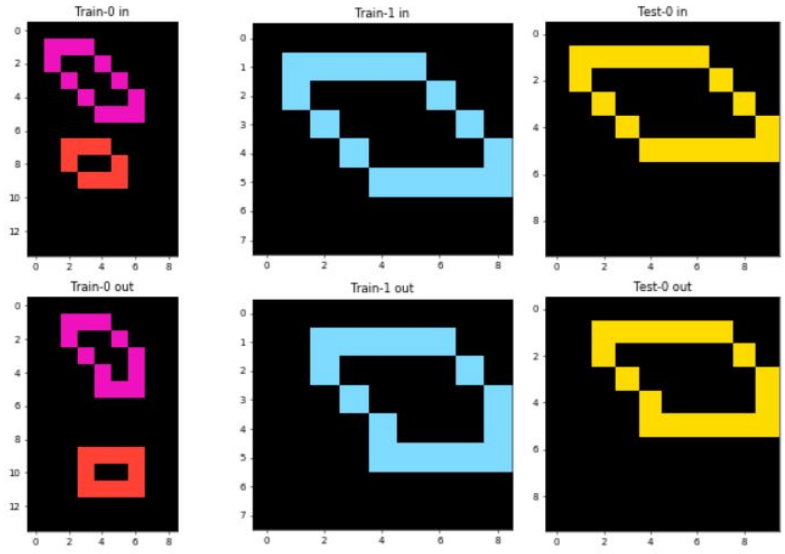


Figure 2: T2.1 example

In other cases the output size is given in absolute sense ($T1.2, T1.3$).

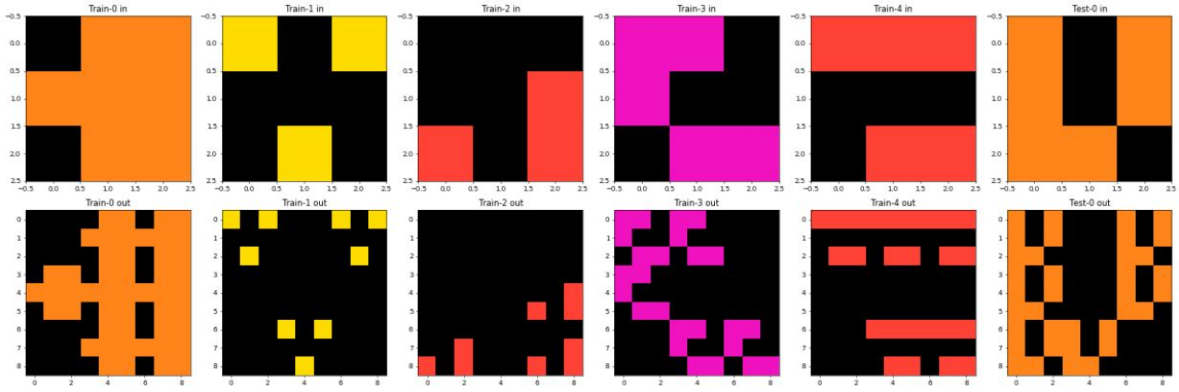


Figure 3: T1.3 example

Or given the input size ($T2.3$), having a fixed ratio between input and output size.

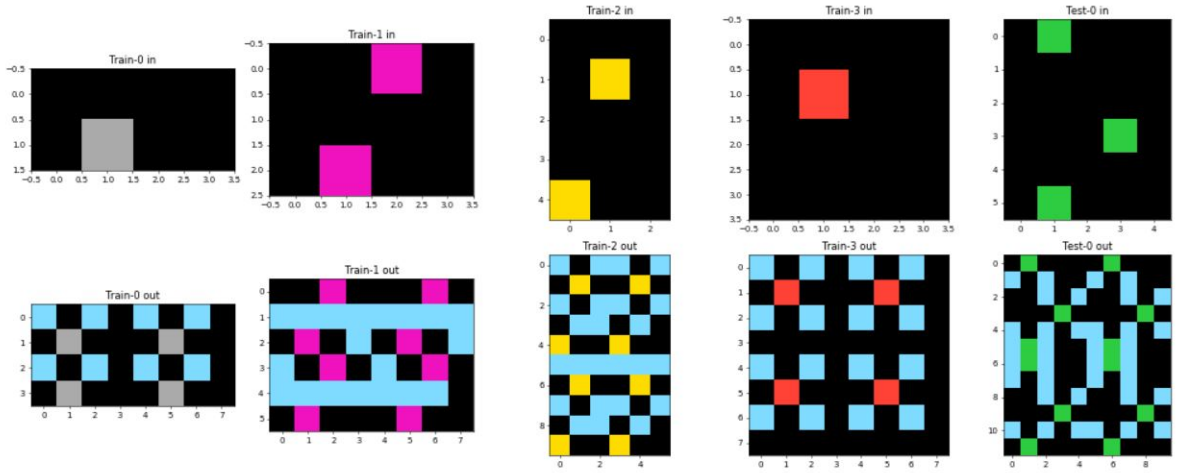


Figure 4: T2.3 example

At last, we also have task in which the output size has to be somehow "learned" from the input, like cropping tasks ($T2.2, T1.4$).

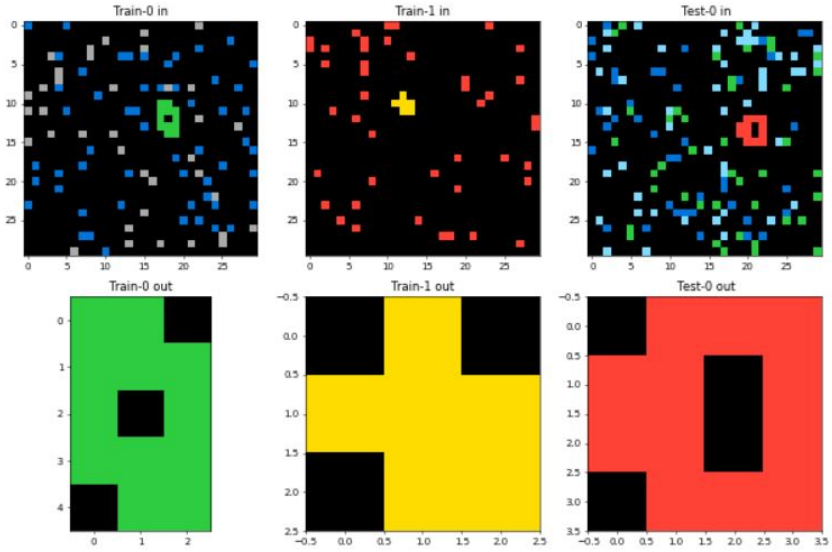


Figure 5: T1.4 example

3.2 How to use a deep-learning approach

Since this problem is build to be intrinsically not adapt to be crunched with deep learning, to explore it's possibility we have to try to tackle it from different sides. We'll try at first to treat each task as a different problem to see what models works for at least some tasks and what are the difficulties encountered. The first thing to do is transform our grids into object easily understandable for a machine, so we will transform our hXw grids with numbers (that

represent class or colors) into hXwX10, a one-hot encoding representation with 10 channels. Then we will use basic models like CNN, FCNN, LSTM and (maybe) more advanced why to insert attention. Then we will try to "propagate" the learning between different task using meta-learning.

3.3 Ad-hoc models exploration

In this preliminary and exploratory phase, we will start with some simple tasks to familiarize with this data set, and we'll try to see if some basic models with some ad-hoc modification (like feeding automatically also an up-scaled or flipped version of the input) are actually able to solve a task, and how they perform if tried on a larger subset of tasks, to see if they are capable to solve some of the tasks apart from the one used to adapt the model.

3.4 Expanding data sets

Here we will write some generators that can expand the data set for a single task and see what changes if we can work with tending to infinite generated data, trying simple models to monitor if at least in this guided framework are able to solve the problem.

3.5 Few-shot learning

While people need only a few samples to recognize, classify an object or to understand the logic behind a simple task, deep learning algorithms usually require a very big amount of data in order to work correctly. To lower the gap between human learning and a computer driven approach it's indeed needed to teach the machine to handle with only a few examples the above mentioned goals. Few-shot learning aims to do just this with algorithms that specifically address the problem of having only few samples per task. The algorithms and models that most typically are referred to when talking about the few-shot learning are matching networks, prototypical networks and model agnostic meta-learning. While the first two we figured out wouldn't work in our case, the latter has a broader field of application.

3.6 Attention mechanism

Attention has become an increasingly relevant concept in deep learning. Basically, it can be interpreted as a mechanism that assigns weights according to how strongly an element attends to other elements. It was developed for Natural Language Processing (NLP) field, but it has brought many breakthroughs also in other areas, as for instance in image processing. In Figure 6 it was drawn a timeline of the development of attention. All starts with *Seq2Seq* model (4), i.e. map sequence inputs into sequence outputs for translation tasks. In this

architecture some information are not caught by fixed length vector, therefore Bahdanau et al. (5) in order to keep information from all hidden states, it is suggested to align the source and target inputs in a vector such that the model attends (i.e. pay attention) to specific part of the input understanding better the relationship between input and output. Then an attention mechanism was proposed by Xu et at. (6) for processing images (actually input an image and output its description in words), the images are analyzed with convolutional layers to extract features and then these features are aligned thanks to an RNN with attention. In other models (7), attention is used on various levels. More recently, in *Attention is all you need* (8), as the title suggests, attention has been used on its own i.e. without convolution and recursive operations. In this implementation (*transformer* architecture), the representation of the sequence is computed through a multi-head self-attention layers that align words in a sequence with other words in the sequence. The multi-head self-attention consists of several attention layers running in parallel. This model has had a great impact on deep learning, since it is more efficient in computation and effective in representation. The idea behind transformers lead to development of many other interesting models as Bidirectional Encoder Representations from Transformers (BERT) (9).

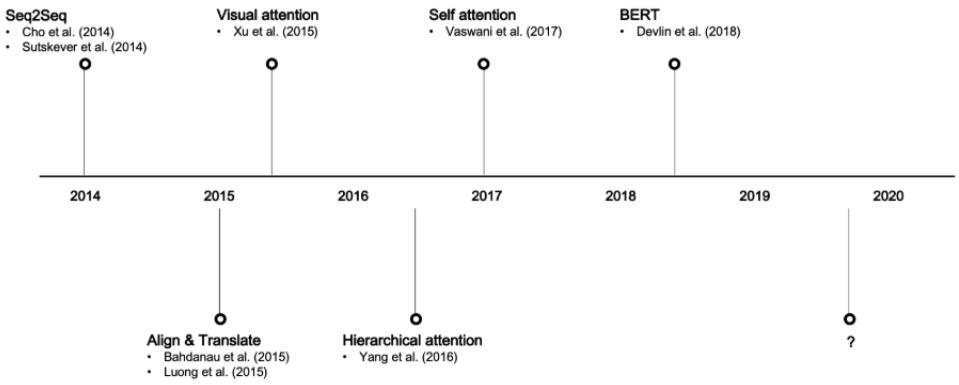


Figure 6: Figure shows several popular models, with corresponding papers, which implement attention mechanisms through time.

Now, coming back to our problem of generated solved tasks, we thought that could be an idea to try to improve our model adding an attention mechanism. Indeed, attention has revealed to be valid in improving performance for sequential data and for extracting information from images. As consequence, we developed a basic attention block. Basically, it was placed after the convolutional layers for our basic CNN model and after the LSTM for the LSTM model. The attention block is capable of taking inputs of different dimensions (2D or 3D). It produces weights by applying a linear layer and an activation (i.e. ReLU) to the input and then the result is passed to a softmax. Finally we take the sum of the attention input values weighted by the attention vector as the approximation of the target. When we try the CNN and the LSTM models equipped with attention on on a set of many different tasks, the results are poor as the ones resulting from basic CNN and LSTM models. However when we tried these models with attention for an infinite train of singular tasks, they sometimes bring

an improvement in term of accuracy compared to their basic versions. This is for example the case of Task 2 or the CNN for Task 3, as it is possible to see in Section 4.2.

4 Experimental results

4.1 Ad-hoc models

To start familiarizing with the data, we decided to take a task and try to, in some sense, "overfit" tuning a simple model until supervising it by hand until it was enough to understand easily the task. The first task that we chose is this:

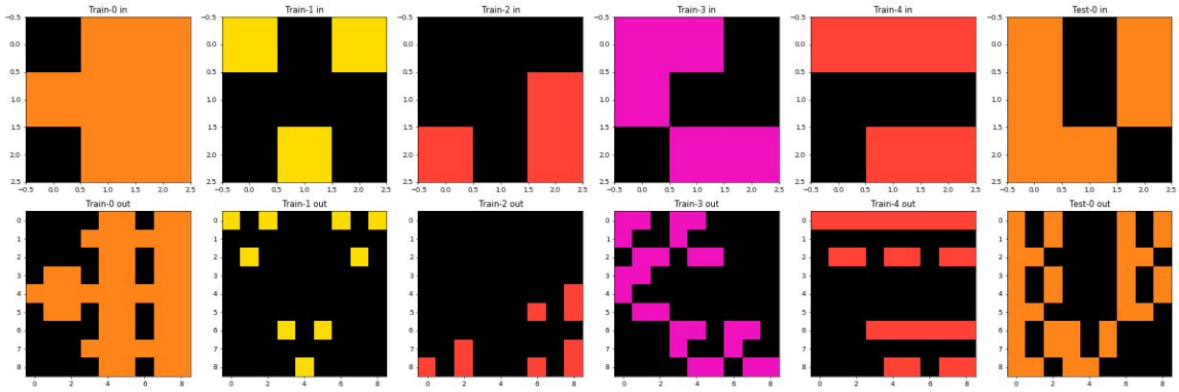
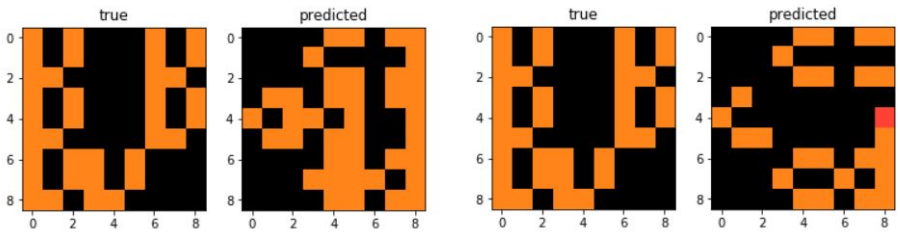


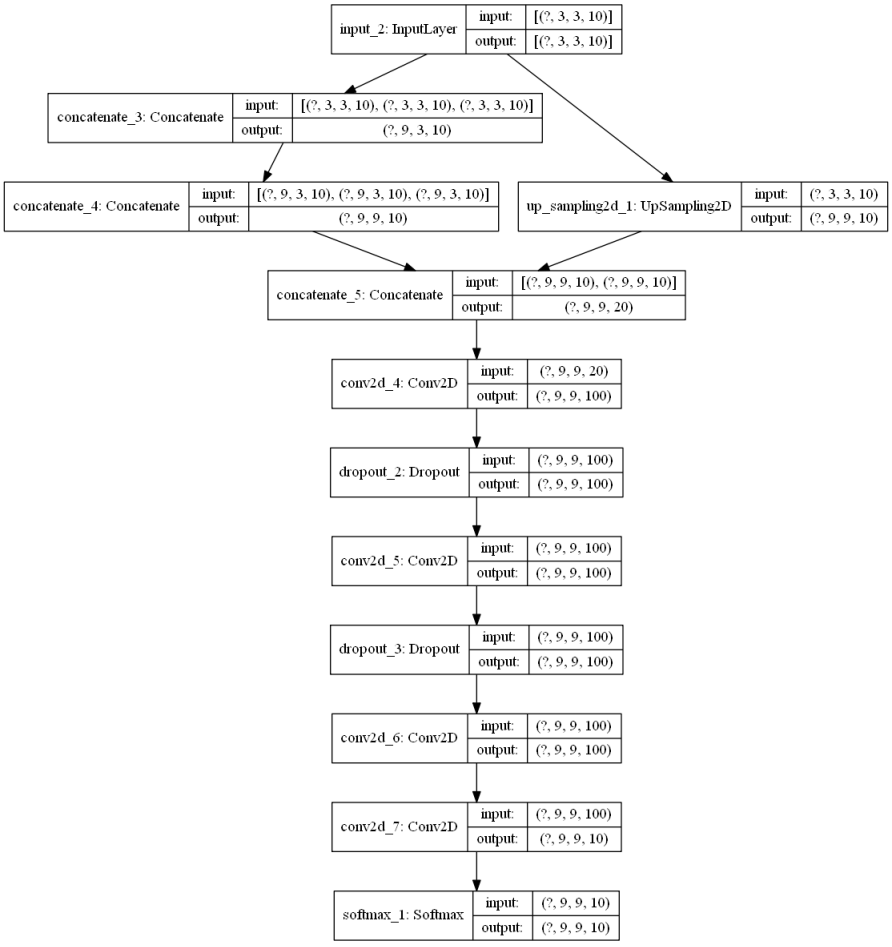
Figure 7: T1.3 example

It comes from the group of the task where the input size is fixed and the output size is in direct relationship with the input size (specifically $h_{out}, w_{out} = 3h_{in}, 3w_{in}$). It's quite simple (although not immediate) for a human to understand that the output is a combination of an upsampling and a 3x3 concatenation, but it is not as easy for simple networks to manage it.

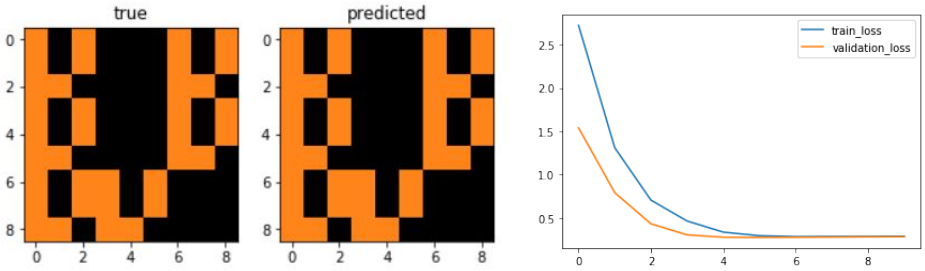
With simple networks with combinations of convolutional and dense layers, even with all the possible regularization (L1 or L2 weight regularization, dropout...) and hyperparameters tuning (descent method, activation functions...) it does not seem possible to solve this simple puzzle. These are some of the results obtained:



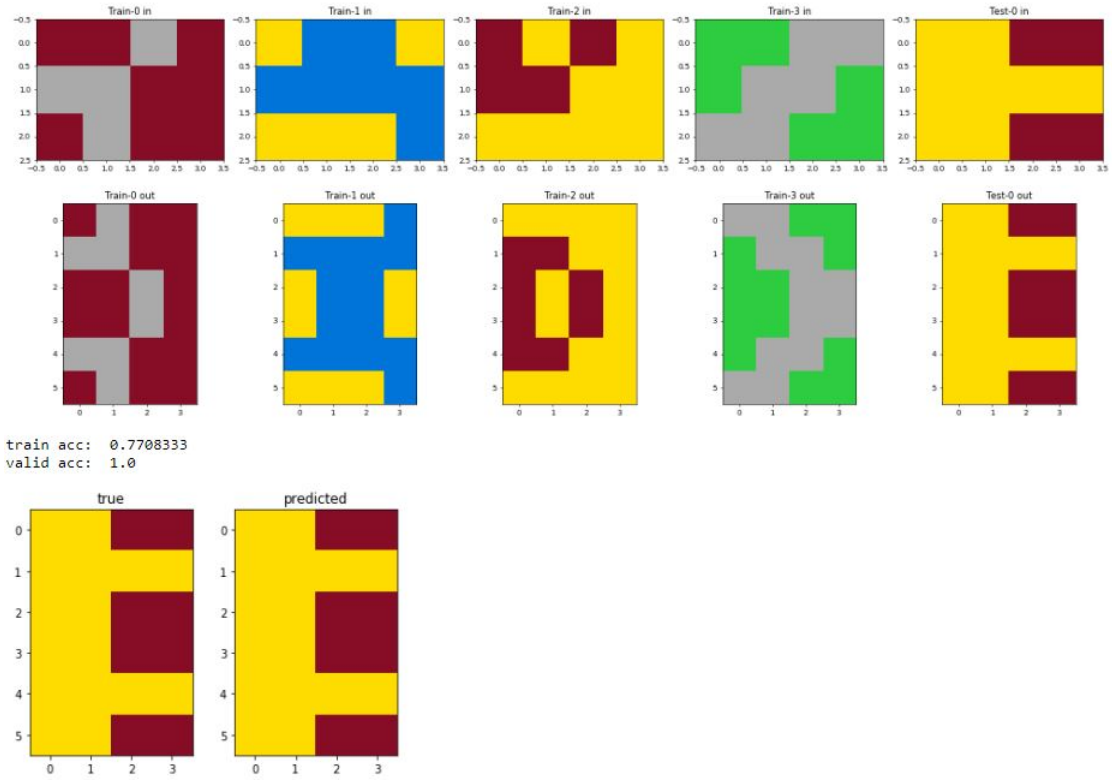
When we have at least a dense layer that allows connection between all the pixel, as we could expect, perfect train accuracy is always reached over-fitting just in a few epoch. But with just a bunch of data, and without the ability to "search for the simple solution" the prediction is almost random. To achieve the result, we tried to tailor the model giving it a little nudge toward the correct solution.

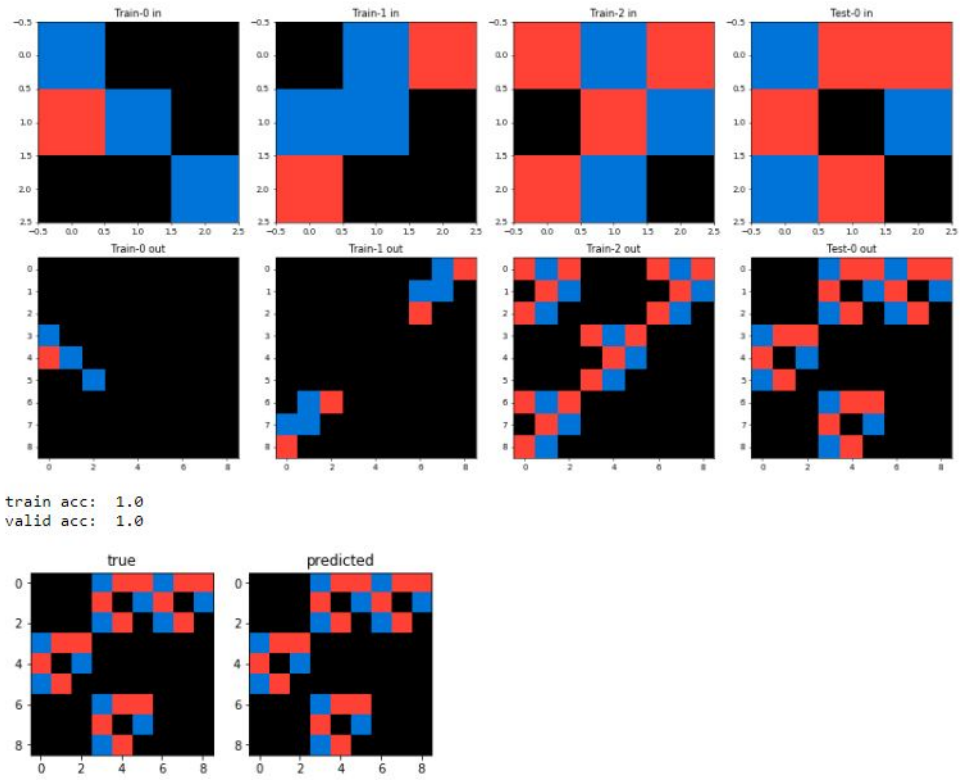


With this model, in just a bunch of epoch we are able to reach perfect accuracy both over the train and over the validation set. We have immediately created the concatenation and the upsampling, then it's easy for it to combine this information with a series of 1x1 Convolution filters (that are in some sense just per-pixel dense layer).

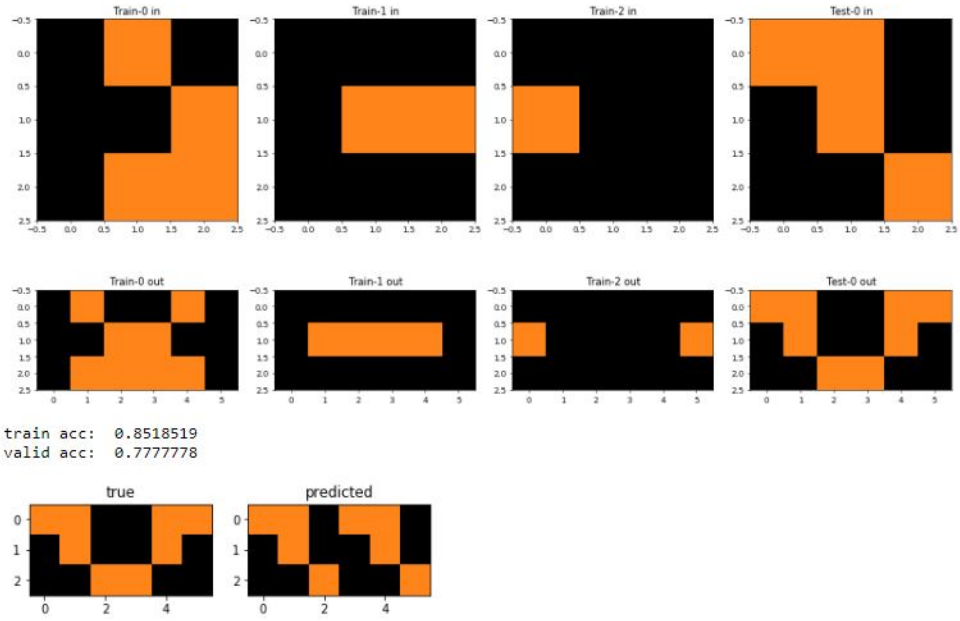


We have seen that, at least for some task, a proper neural network can solve the problem, but we now have to understand if and how this can help us with more than just this task. The simplest generalization that can be done is to use the same identical model for all the task of this group. It actually succeeded in solving some of them, for example:

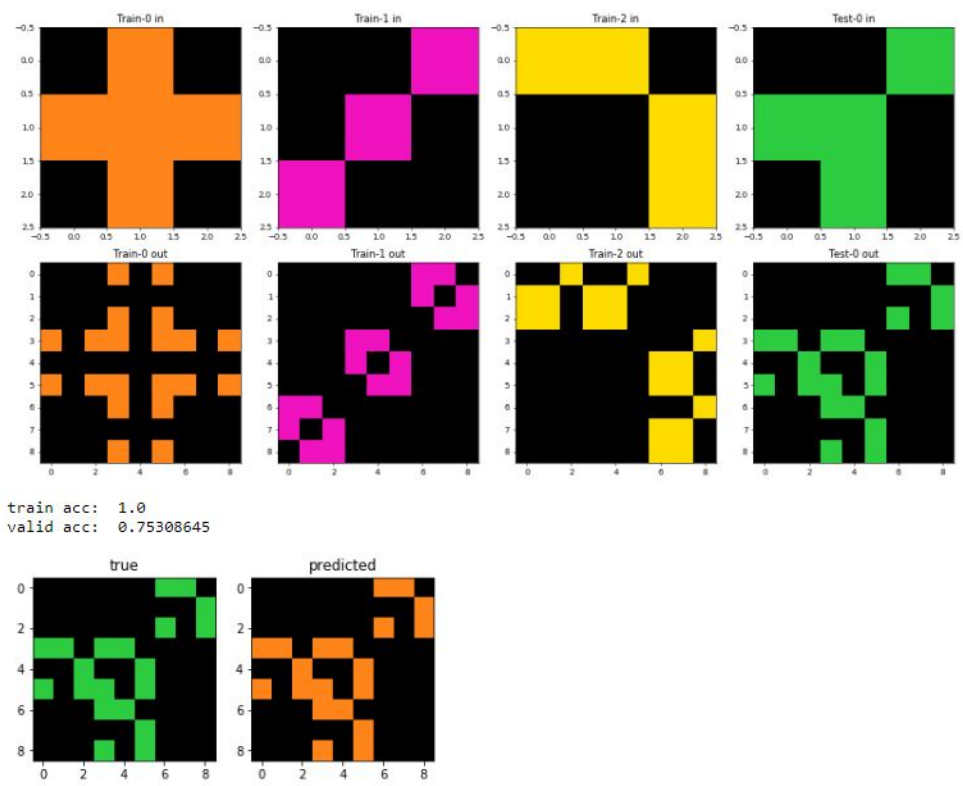




In general this model solves all the task where a simple concatenation or a clean combination of combination and upscaling is enough to get the things done. Unfortunately they are not a lot (4/20 for the train, 2/20 for the validation set). It's interesting to analyze some of the "close" answers to see what's happening. We have to type of problems that seem to be easily solved:

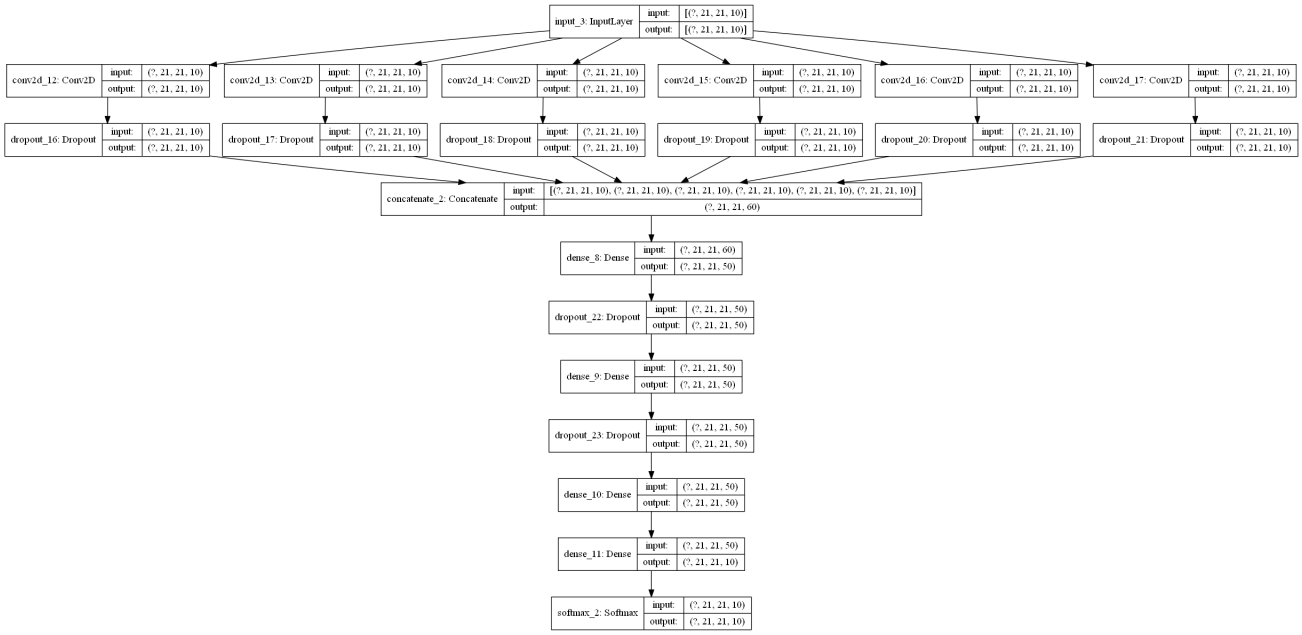


We have a lot of flips (both up and down) that maybe could be incorporated in some models. But the fact that sometimes It's a mirroring and so the solution is a concatenation of the input and the input flip, makes the problem a little bit harder that what could seem at first glance.



We also have the "I've never seen this color" problem, and this could be easily solved augmenting the data preprocessing them with a random change of color. This is what we did, solving this task and one more (increasing the count to 5/20 and 3/20).

We did a similar, but more rough, experiment over the biggest and group of tasks (the one where input output. Since some people on kaggle claim convolutional neural networks as a reasonable road to follow, we tried different kernel size. Since each kernel size was able to solve some of the tasks (typically, 3x3 kernel were able to solve tasks with substitution of pattern of this size etc) we, similarly as before, created a method that concatenated kernels of different sizes, solving 11/134 training tasks and 3/92 validation tasks.



With these experiments we learned that, without a proper smart "lead" it's really difficult to solve a task in this setting. It seems like, without the powerful "Occam razor principle" it's really hard for the machine to find the proper path to the solution, and in general to parse the problem. In the next paragraph we will inspect what happens if we increase the data set size.

4.2 Expanded data sets

In this section, we aim to analyze what simple model are able to retrieve correctly information from different type of tasks, somehow emulating the work of Bai et al (10). We will try this collection of models:

- basic FC network without hidden layers
- basic FC network with one hidden layer
- CNN with 100 filters of size 5
- CNN with 100 filters of size 5 and attention
- CNN with 100 filters of size 3 and attention
- LSTM with attention
- LSTM without attention
- GRU with attention

All the models are trained for 5 epochs...

4.2.1 Task 1

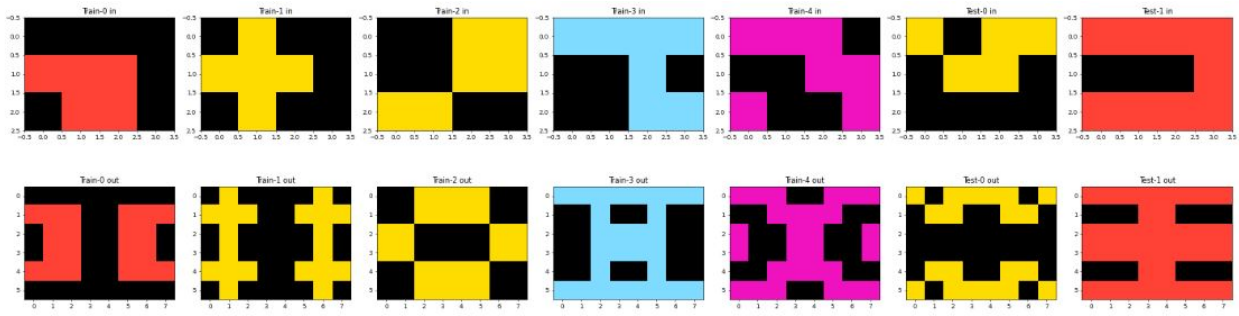


Figure 8: Seven pair samples of the 1st task. Each pair represents the input (top) and the output (bottom). The task consists in flipping the original image on the right and then flipping again the resulting image to the bottom, obtaining in this way a symmetric output. As consequence, the output image has four time the area of input image.

Table 1: The table reports for each of the eight model used the resulting train and validation accuracy on the entire image (i.e. predicted image is identical to the target image) and according to number of correct pixels. Best results for this task are obtained with *FullyCon* model.

	Train acc.	Train acc. pixel	Val. acc.	Val. acc. pixel	100% acc. epoch
CNN3	0.11	87.870	1.0	88.222	/
CNN_no	0.09	83.312	0.0	83.458	/
CNN	0.09	83.618	0.0	84.000	/
FullyConDeep	98.13	99.959	100.0	100.000	4
FullyCon	100.00	100.000	100.0	100.000	4
GRU	6.27	95.794	6.0	95.750	/
LSTM_no	26.70	97.471	26.0	97.562	/
LSTM	40.15	98.357	34.0	98.146	/

This task is perfectly solved by the Dense neural network, thanks to the fact that each pixel of the output is perfectly related to the pixel in a given position of the input (for example, all the corners of the output are just the same pixel of the up-left corner of the input, and so on and so forth). In this case, the CNN is not able to understand the task, due to the long-dependency between pixels. GRU and LSTM come really close to the solution, but usually misclassify one or two pixels:

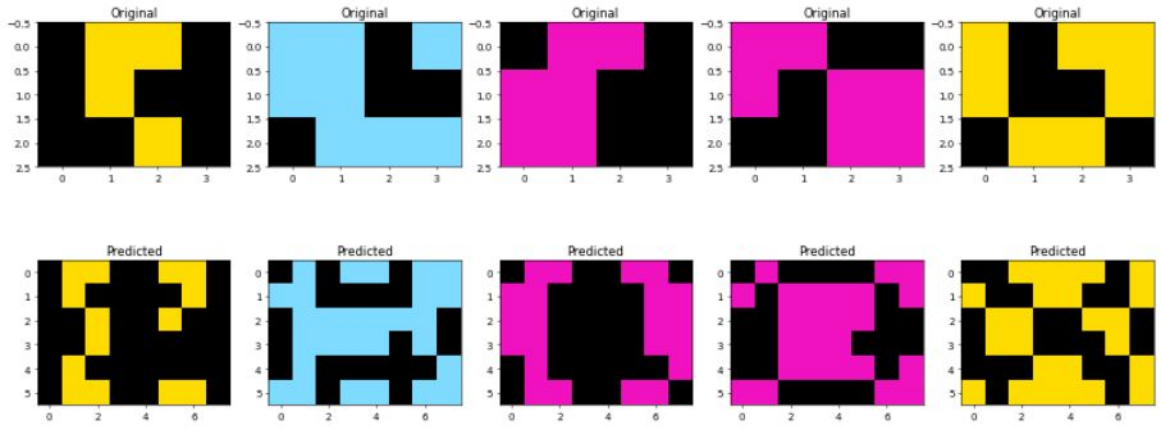


Figure 9: Some GRU predictions

4.2.2 Task 2

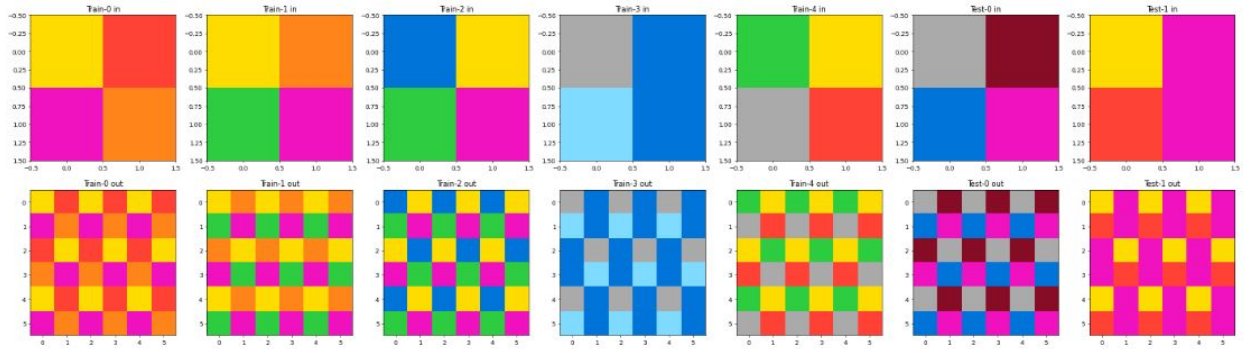


Figure 10: Seven pair samples of the 1st task. Each pair represent the input (top) and the output (bottom). The task consists in repeating the original image three times for three lines, note that in the central line the image is flipped. As consequence, the output image has nine time the area of input image.

Table 2: The table reports for each of the eight models used the resulting train and validation accuracy on the entire image (i.e. predicted image is identical to the target image) and according to number of correct pixels. Best results for this task are obtained with *CNN* and *FullyCon* models.

	Train acc.	Train acc. pixel	Val. acc.	Val. acc. pixel	100% acc. epoch
CNN3	0.02	89.373	1.0	89.694	/
CNN_no	99.87	99.996	100.0	100.000	/
CNN	99.56	99.987	100.0	100.000	4
FullyConDeep	99.66	99.991	99.0	99.972	/
FullyCon	100.00	100.000	100.0	100.000	4
GRU	0.00	96.904	0.0	96.750	/
LSTM_no	67.33	99.092	78.0	99.389	/
LSTM	0.00	80.144	0.0	80.278	/

This task is solved by both dense networks, for the same reason of the former, and for CNN, given that the kernel size is big enough. In fact, with a kernel size of just 3, the net has problems understanding properly borders:

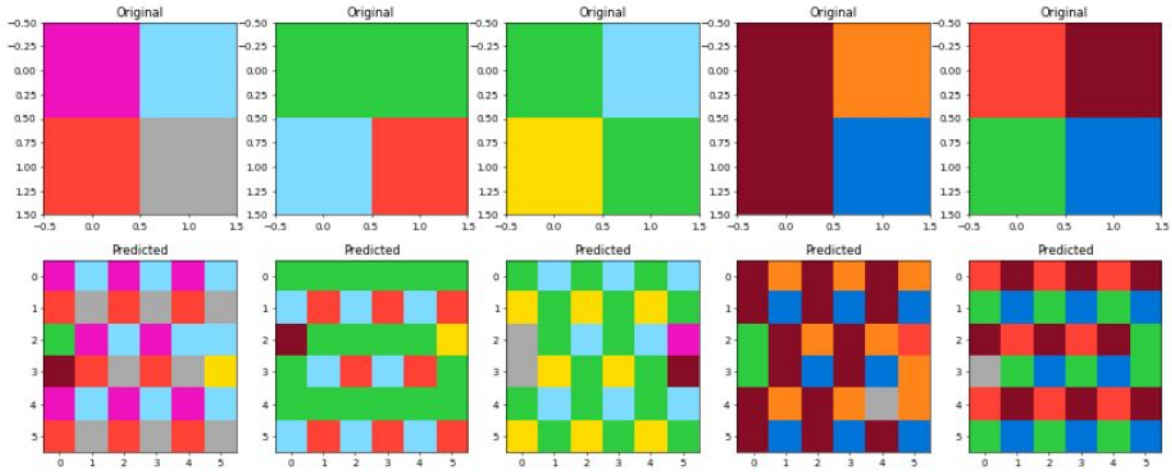


Figure 11: CNN kernel 3 predictions

GRU and LSTM present the same problems of the previous task.

4.2.3 Task 3

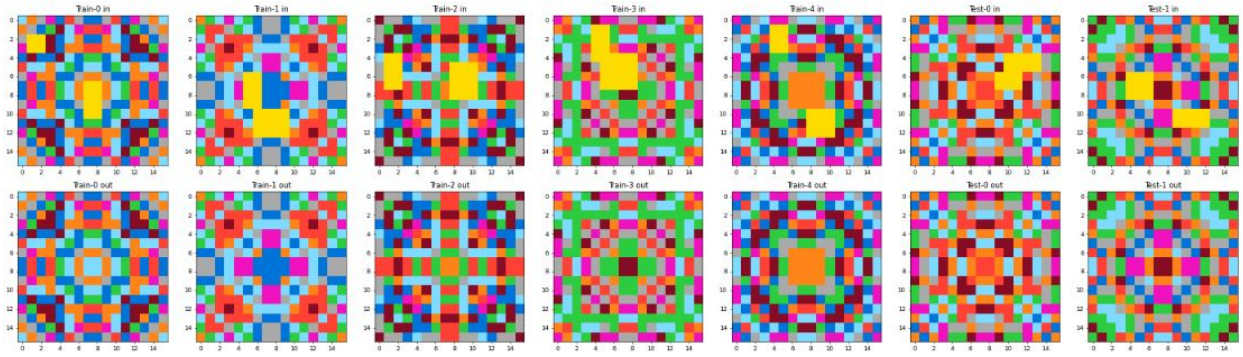


Figure 12: Seven pair samples of the 1st task. Each pair represents the input (top) and the output (bottom). The task consists in retrieving pixels that in the original image are covered by yellow patches. Note that the images are always horizontally, vertically and diagonally symmetrical. Further, input and output dimensions do not change.

Table 3: The table reports for each of the eight models used the resulting train and validation accuracy on the entire image (i.e. predicted image is identical to the target image) and according to number of correct pixels. Best results for this task are obtained with *FullyCon* model.

	Train acc.	Train acc. pixel	Val. acc.	Val. acc. pixel	100% acc. epoch
CNN3	0.00	94.117	0.0	93.844	/
CNN_no	0.00	94.212	0.0	94.113	/
CNN	0.00	94.497	0.0	94.324	/
FullyConDeep	0.00	77.621	0.0	73.352	/
FullyCon	80.16	99.881	58.0	99.613	/
LSTM_no	0.00	15.864	0.0	15.750	/
LSTM	0.00	21.357	0.0	21.480	/

The only model that performs reasonably well is the shallow Dense network. This is because it is the only model that is able to retrieve the long-distance dependencies given by symmetry, obviously not because understand the concept of symmetry but because, as in the former task, encodes all the positional relations from input and output. Also the a-little-bit-deeper dense network fails in this task, given the fact that with also the hidden layer we have an enormous number of possible relation and it over-fits too much. As we have said, the result of the shallow FCNN is quite satisfying:

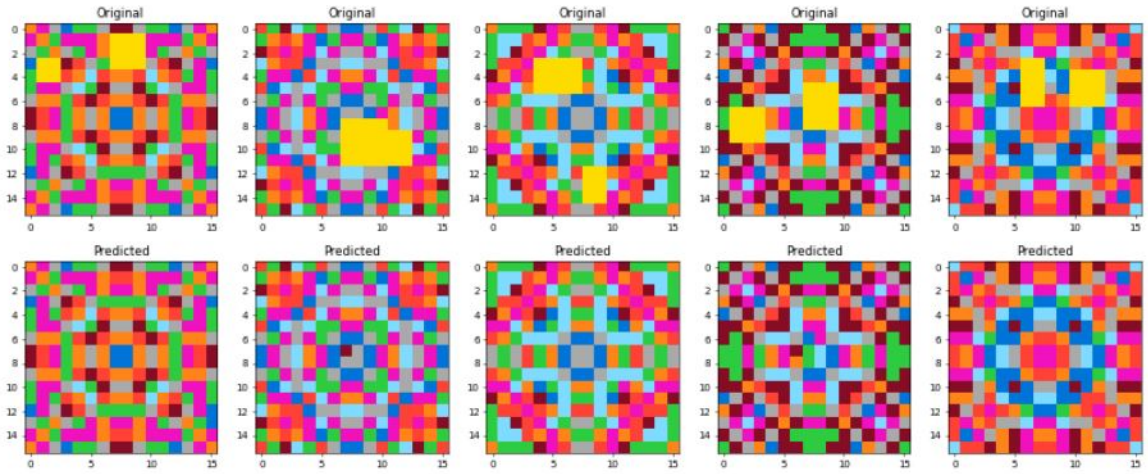


Figure 13: FCNN wrong predictions

Even when it goes wrong, it's usually because the solution is impossible (when the patch hides the center) or because it misclassifies only one pixel, usually not even one of the hidden ones.

4.2.4 Task 4

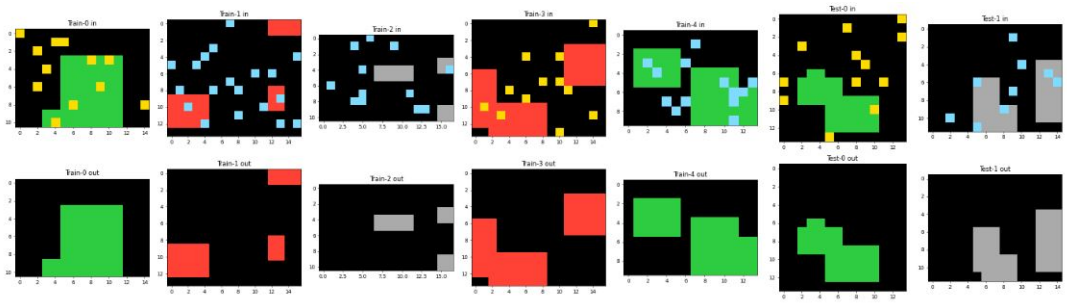


Figure 14: Seven pair samples of the 1st task. Each pair represents the input (top) and the output (bottom). The task consists in de-noising the original image from the colored single pixels placed randomly in it. Each output shape is equal to the corresponding input shape, but the inputs can have different shape between each other.

Table 4: The table reports for each of the eight models used the resulting train and validation accuracy on the entire image (i.e. predicted image is identical to the target image) and according to number of correct pixels. Best results for this task are obtained with *CNNs* models.

	Train acc.	Train acc. pixel	Val. acc.	Val. acc. pixel	100% acc. epoch
CNN3	76.78	99.838	76.0	99.854	/
CNN_no	69.05	99.773	70.0	99.803	/
CNN	75.99	99.833	76.0	99.848	/
FullyConDeep	0.00	88.817	0.0	88.310	/
FullyCon	46.16	99.550	26.0	99.064	/
GRU	0.00	73.403	0.0	72.741	/
LSTM_no	0.00	56.269	0.0	55.932	/
LSTM	0.00	57.067	0.0	56.219	/

This task is solved quite good with attention Cnn models. It's totally reasonable that the one with the 3x3 kernel performs like the 5x5 one, since all the needed information is contained in the 3x3 shell.

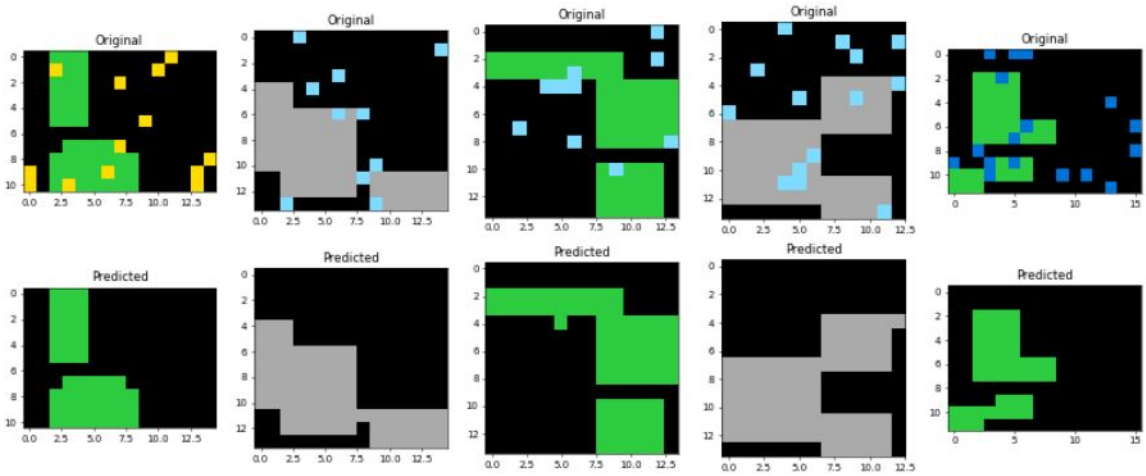


Figure 15: CNN wrong predictions

The errors are usually given from the fact that to decide the color under the noise it usually considers only the colors at the sides of the pixel.Not being able to consider the fact that we have big rectangular shapes, this can lead to small errors at the edge of the rectangles.

4.2.5 Task 5

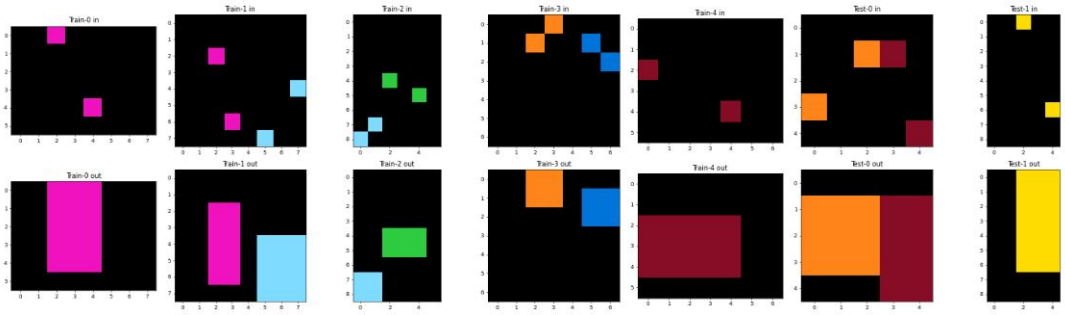


Figure 16: Seven pair samples of the 1st task. Each pair represents the input (top) and the output (bottom). The task consists in filling the space between two pixel with the same color. Note that the clean image is composed by object of rectangular shapes (even if they can "merge") Each output shape is equal to the corresponding input shape, but the inputs can have different shape between each other.

Table 5: The table reports for each of the eight model used the resulting train and validation accuracy on the entire image (i.e. predicted image is identical to the target image) and according to number of correct pixels. Best results for this task are obtained with *CNN* models.

	Train acc.	Train acc. pixel	Val. acc.	Val. acc. pixel	100% acc. epoch
CNN3	0.25	65.822	0.0	67.420	/
CNN_no	6.81	81.852	10.0	81.999	/
CNN	7.19	81.964	10.0	82.307	/
FullyConDeep	3.03	84.704	2.0	81.656	/
FullyCon	3.59	83.765	2.0	80.921	/
GRU	0.00	65.066	0.0	65.743	/
LSTM_no	0.00	51.446	0.0	52.225	/
LSTM	0.00	54.930	0.0	56.383	/

None of these models seem to be able to solve this task. The fact that the input can present two different couples of vertices of the rectangles, thus greatly increasing the possible images patterns, makes the problem very complicated for the computer (even if it's so simple for our mind). CNN somehow understand some simple ones:

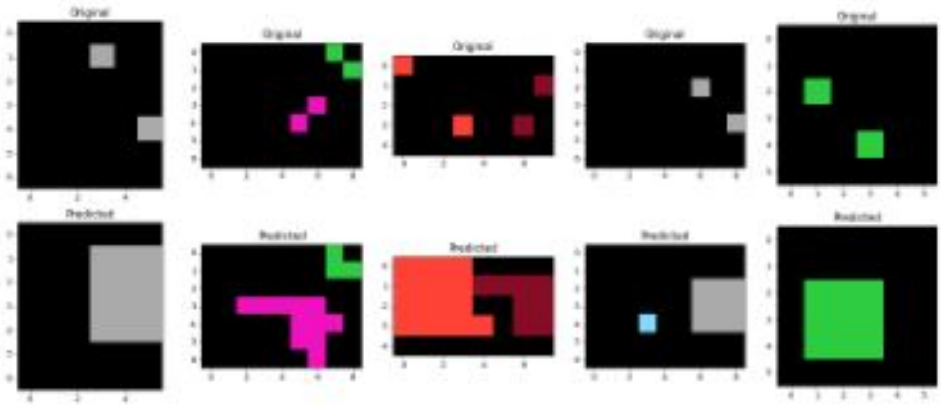


Figure 17: FCNN wrong predictions

4.2.6 Task 6

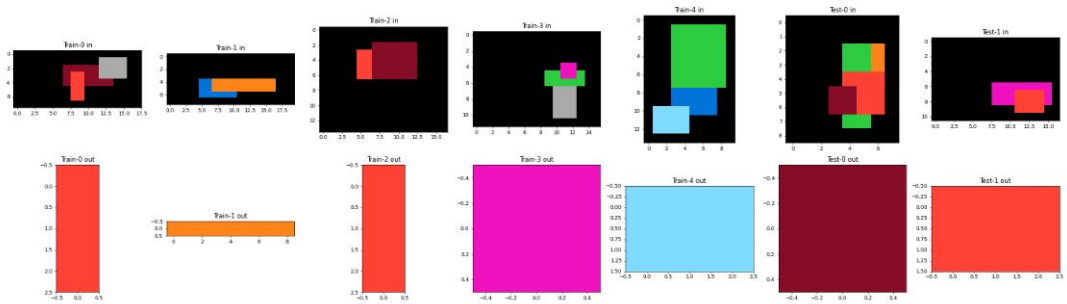


Figure 18: Seven pair samples of the 1st task. Each pair represents the input (top) and the output (bottom). The task consists in identifying the smallest square in the original image and output it, therefore outputs have variable dimensions.

We decided to report this task to show an example of a task that, without a specific tailored model ad-hoc, seem very hard to solve. Since the output shape is part of the solution, all the simple models that we used for the other tasks are useless. We could argue that some method based on bounding boxes could be good (since they seem naturally more suited than, for example, methods that build the image as a sequence with special tokens ¡start¿, ¡stop¿, and maybe a parameter that regulates the length of the different rows, that could be too "free" to model this problem), but it would be nonetheless a guided adaptation to this single task that would bring back us to the problem of the first chapters.

4.2.7 Task 7

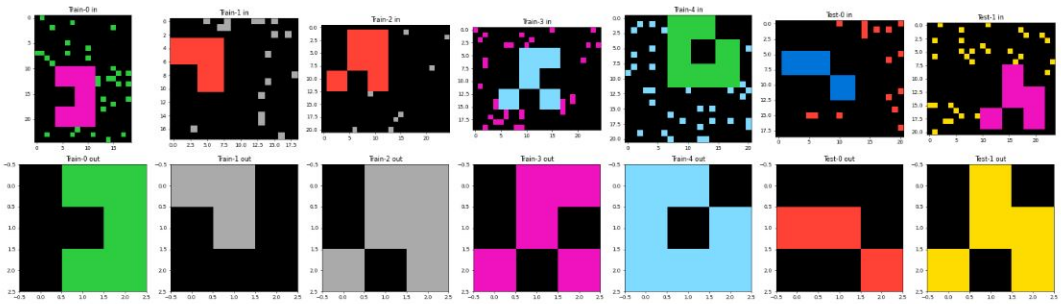


Figure 19: Seven pair samples of the 1st task. Each pair represents the input (top) and the output (bottom). The task consists in: de-noise from smallest single pixel placed at random, and output a square containing the main figure of the original image and color it of the same color of the dropped single pixels. Inputs have variable shape, but all the outputs have the same shape

Table 6: The table reports for each of the eight models used the resulting train and validation accuracy on the entire image (i.e. predicted image is identical to the target image) and according to number of correct pixels. Best results for this task are obtained with *FullyCon* models.

	Train acc.	Train acc. pixel	Val. acc.	Val. acc. pixel	100% acc. epoch
CNN3	0.00	24.442	0.0	27.000	/
CNN_no	0.04	31.981	0.0	33.333	/
CNN	0.01	17.137	0.0	17.778	/
FullyConDeep	0.32	49.377	1.0	41.333	/
FullyCon	1.52	57.450	0.0	42.778	/
GRU	0.18	50.290	0.0	50.111	/
LSTM_no	0.22	49.510	0.0	49.222	/
LSTM	0.22	49.947	2.0	51.556	/

This task is extremely difficult and no models is able to come neither close to the solution. The combination of things to do (detection, downsampling, color swap) seems really too much to solve with a non-tailored model.

4.2.8 Summary

Table 7: The table report the validation accuracy values for the entire image (i.e. predicted image is identical to the target image) realtive to all the eight model used for each of the seven tasks.

	CNN3	CNN_no	CNN	FullyConDeep	FullyCon	GRU	LSTM_no	LSTM
Task 1	1.0	0.0	0.0	100.0	100.0	6.0	26.0	34.0
Task 2	1.0	100.0	100.0	99.0	100.0	0.0	78.0	0.0
Task 3	0.0	0.0	0.0	0.0	58.0	0.0	0.0	0.0
Task 4	76.0	70.0	76.0	0.0	26.0	0.0	0.0	0.0
Task 5	0.0	10.0	10.0	2.0	2.0	0.0	0.0	0.0
Task 6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Task 7	0.0	0.0	0.0	1.0	0.0	0.0	0.0	2.0

Here we report a summary of all the results achieved. There is not a lot to add, as we saw in the previous paragraphs, the convolutional neural networks are good to solve the task where we have features that are strictly local, while deep neural network where able to model long-distance pixel to pixel relation. Recurrent neural networks, on the other hand, while being able to reach sometimes good pixel accuracy, were never able to solve the tasks for all of the pixel (even if often it were wrong just for one or two pixels).

4.3 Few shot learning

Since the examples per task we have are very limited, few shot learning seemed like the appropriate way to tackle our problem.
We first started with prototypical networks: after some initial doubts on whether the algorithm were suited for our problem, we quickly realised that the performances were very poor compared to a more simple dense network. Prototypical networks assume there exists an embedding where the samples from the classes make a cluster around a prototypical representation of them, which is their mean. However our application is not a "simple" classification task but more precisely an identification of the pixels value (which can be seen as classes ranging from from 0 to 9) based on some logic. Thus, the mean of the embeddings of each class is not representative of the color the pixel would assume in another sample. Based on this assumption, we dropped the usage of prototypical networks.
Model agnostic meta-learning is an algorithm which generalises to any differentiable model to learn an initialisation of the network to train the model on. The algorithm works by creating at each training epoch a new model (called "fast") which is trained on a meta batch of samples. Then, the fast weights are updated according to a simple stochastic gradient descent. In the next phase a few more examples from the same task are drawn and the model

is trained by taking the gradient with respect to the sum of the losses of the model trained on the meta batch samples. The second update can be seen as taking a second order derivative update, thus requiring many computations to go through the whole process. Hence it was proposed to only make a first order approximation for reducing the time complexity but it leads, in general, to poorer, although not too different, performances. We decided then to resort to the second order approximation but longer training time. We proceeded to implement MAML with a simple linear model and a modified version of the ResNet on three different tasks taken together: the results for the linear model are good on training set ($\sim$ 80%) but poor on the validation set ($\sim$ 20%). The ResNet, instead, performed very poorly on both the training and validation set ($\sim$ 15%).

5 Conclusion and future work

5.1 What we learnt

Developing this project was very challenging to the point that was almost frustrating.

The project started with an enthusiastic "let's make our methods work" but diving deeper into the paper by Chollet (1) we understood that this challenge was tailored specifically to enlighten the weaknesses of the state-of-the-art artificial intelligence. As explained in the introduction, the lack of ability in generalization, and the hunger for complete data sets of the modern AIs are not just features that we can solve easily but deeply inherent of how it's (at least today) formulated and developed. Maybe we were to naive to think that we could be genial enough to propose something new (something that, however, nobody is been able to propose at the moment as we will report in the next paragraph) that could shake the study field. The main problem is been that, at some point, it was clear that deep-learning models that we studied in this course were just not the right tool for this problem. It felt like trying to make orange juice with anvil and hammer. These tasks are based on the assumption that we have some natural priors (concept of symmetry, privilege for regular geometrical shapes etc...), that help us to solve them with extreme speed and precision, so every model that is not able to encode this information naturally seems very disadvantaged.

Since for the scope of this project we were tangled to use deep-learning methods, we had to adjust our direction and to refine our search, that became "let's show how and why you can't use the hammer and the anvil to make orange juice" that, since we are not really squeezing oranges, it's not as trivial, and could help us to get interesting insights both of the task and the models themselves, a part from being a useful exercise of programming and modeling.

In the end, we are not very satisfied about the outcomes of this project, but for sure we reflected about deep-learning from a new perspective that, even if in this particular case didn't help us to solve this particular problem, will be always part of our cultural background and could be exploited in the future.

5.2 What others tried

As we mentioned early, the Kaggle Challenge has ended and nobody has solved the challenge with novel methods. A lot of people tried to start some work using deep-learning (especially convolutional neural network like here), but nobody really went deep enough with it (and that's why we were persuaded that it was worth a shot). The only relatively good results was achieved by the user IceCube (here his description of his work), that using a Domain Specific Language is been able to solve 20% of the evaluation test tasks. Is solution, however, consisted in, even if a very fashion and highly optimized way, a pseudo hard coded solution, that, for his own admission, doesn't bring us any nearer to a more broad intelligent IA. He, briefly, defined a small set of possible functions (for example a mirroring of the image, the draw of the borders...), and explored all the possible combinations of this functions for each of the tasks, choosing as answers the combination that fitted better the train images, privileging shortest combination. More over, he defined the function by looking specifically at the tasks of the evaluation set (isn't it a little bit cheating?).

Another interesting path that could be followed is the transformer based approach proposed by yuequi and released after the end of the challenge (so we had not the time to try and to explore expansions of his solution). His transformer encodes and reasons about the relations between objects among the input-output pairs. At first pretrains the model on the training set tasks to build representations for core knowledge priors, then at test time, first train on the demonstration examples of the unseen tasks, then predict the test output. Unfortunately, despite building a quite complex structure that includes multiple branches and encodings, he didn't program a way to tackle input-output different shapes, and even if for some task the results are pretty interesting, at the moment his method only solved 1% of test task (less than just trowing convolution over everything).

5.3 Possible future directions

One future direction that maybe could be tried, following the solution given by IceCube, could involve reinforcement learning. After the definition of the possible functions, this function could maybe be interpreted as moves that the IA can make to move toward the solution. Then could be defined so a game with the final goal of transforming the input image in the output image with the smaller number of moves (using the Occam Razor principle that the simplest solution is often the best, ceteris paribus). At this point, with this goal, the AI could be trained, in a way similar to the alpha go zero (11). In this way maybe, exploiting Monte Carlo tree search, we could speed up the computation, avoiding to follow paths that do not approach the solution.

References

[1] C. Francois, "On the measure of intelligence," *arXiv:1911.01547 [cs.AI]*, 2019.

[2] L. Shane and H. Marcus, "A collection of definitions of intelligence," 2007.

[3] B. David and al., "Measuring abstract reasoning in neural networks," *arXiv:1807.04225v1*, 2018.

[4] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[5] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[6] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[7] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, 2016, pp. 1480–1489.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[10] V. K. Shaojie Bai, J. Zico Kolter, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv:1803.01271v2 [cs.LG]*, 2018.

[11] D. Silver, T. Hubert, J. Schrittwieser, and al., "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play."