Learning to solve geometric construction problems from images*

Jaroslav Macke[1,2][0000−0001−6938−4776], Jiri Sedlar[2(✉)][0000−0002−4704−3388], Miroslav Olsak[3][0000−0002−9361−1921], Josef Urban[2][0000−0002−1384−1613], and Josef Sivic[2][0000−0002−2554−5301]

[1] Charles University in Prague, Czech Republic
[2] Czech Technical University in Prague, Czech Republic
jiri.sedlar@cvut.cz
[3] University of Innsbruck, Austria

Abstract. We describe a purely image-based method for finding geometric constructions with a ruler and compass in the Euclidea geometric game. The method is based on adapting the Mask R-CNN state-of-the-art visual recognition neural architecture and adding a tree-based search procedure to it. In a supervised setting, the method learns to solve all 68 kinds of geometric construction problems from the first six level packs of Euclidea with an average 92% accuracy. When evaluated on new kinds of problems, the method can solve 31 of the 68 kinds of Euclidea problems. We believe that this is the first time that purely image-based learning has been trained to solve geometric construction problems of this difficulty.

Keywords: computer vision · visual recognition · automatic geometric reasoning · solving geometric construction problems

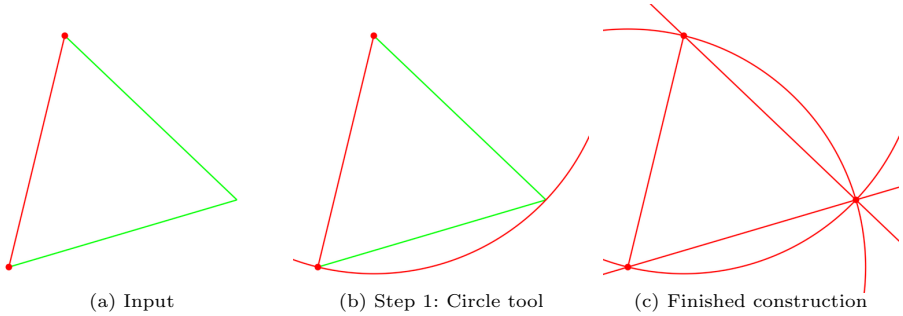
1 Introduction

In this work, we aim to create a purely image-based method for solving geometric construction problems with a ruler, compass, and related tools, such as a perpendicular bisector. Our main objective is to develop suitable machine learning models based on convolutional neural architectures to predict the next steps in the geometric constructions represented as images.

In more detail, the input to our neural model is an image of the scene consisting of the parts that are already constructed (red) and the goal parts that remain to be drawn (green). The output of the neural model is the next step of the construction. An example of the problem setup is shown in Fig. 1.

Our first objective is to solve as many geometric construction problems as possible when the method is used in a *supervised setting*. This means solving construction problems that may look very different from images in the training

* This work was partly supported by the European Regional Development Fund under the projects IMPACT and AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000468 and CZ.02.1.01/0.0/0.0/15_003/0000466) and the ERC Consolidator grant *SMART* no. 714034.



(a) Input (b) Step 1: Circle tool (c) Finished construction

Fig. 1: Example solution of Euclidea level *Alpha-05* (construct an equilateral triangle with the given side). In all examples, the red channel contains the current state of the construction and the green channel the remaining goal. (a) Initial state: one side is given and the goal is to construct the remaining two sides. (b) State after the first step: construction of a circle. (c) State after the last step: finished construction.

problems, but are solved by the same abstract sequences of construction steps. This setting is still hard because the neural model needs to decide where and how to draw the next construction step in a new image. Our second objective is to solve problems unseen during the training. This means finding sequences of construction steps that were never seen in the training examples. We evaluate our method in both these settings.

Tool	Arguments
Point	(coordinates)
Line	(point*, point*)
Circle	(point, point)
Perpendicular Bisector	(point*, point*)
Angle Bisector	(point*, point, point*)
Perpendicular	(line, point)
Parallel	(line, point)
Compass	(point*, point*, point)

Table 1: Euclidea tools and their argument types. The asterisk denotes interchangeable arguments. For a detailed description of the tools and their arguments see the supplementary material available at the project webpage [1].

We train and test our models on instances of geometric problems from the Euclidea [2] game. Euclidea is an online construction game where each level represents one kind of geometric problem. Table 1 lists the construction tools available in Euclidea. Each level specifies which of the tools can be used. Euclidea construction problems vary across a wide spectrum of difficulty. While lower levels are relatively simple or designed specifically to introduce a new tool, more advanced levels quickly grow in difficulty. These advanced problems are not trivial even for reasonably trained mathematicians, including participants of the International Mathematics Olympiad (IMO). Our high-level research objective

is to explore the question whether computers can learn to solve geometric problems similarly to humans, who may come up with solutions without knowing any algebraic and analytic methods. Solving formally stated IMO problems has already been considered as a grand reasoning challenge[4].

Solving construction problems from input images poses several challenges. First, the same geometric problem can have an infinite amount of different variants with a different scale, rotation or different relative position of individual geometric primitives. The visual solver has to deal with this variability. Second, the search space of all possible geometric constructions is very large. For example, a construction with ten steps and (for simplicity) ten different possible construction choices at each step would require searching 10^{10} possible constructions. To address these challenges we adapt a state-of-the-art convolutional neural network visual recognizer that can deal with the large variability of the visual input and combine it with a tree-search procedure to search the space of possible constructions. We namely build on the Mask R-CNN object detector [3] that has demonstrated excellent performance in localizing objects (e.g. cars, pedestrians or chairs) in images and adapt it to predict next steps in geometric constructions, for example, to draw a circle passing through a point in the construction, as shown in Fig. 1b. Despite the success on real images, the off-the-shelf "Vanilla" Mask R-CNN approach can solve only the very basic level packs of the Euclidea game and adapting Mask R-CNN to our task is non-trivial. In this work we investigate: (i) how to train the network from synthetically generated data, (ii) how to convert the network outputs into meaningful construction steps, (iii) how to incorporate the construction history, (iv) how to deal with degenerate constructions and (v) how to incorporate the Mask R-CNN outputs in a tree-based search strategy.

Contributions. In summary, the contributions of this work are three-fold. First, we describe an approach to solving geometric construction problems directly from images by learning from example constructions. This is achieved by adapting a state-of-the-art Mask R-CNN visual recognizer and combining it with a tree search procedure to explore the space of construction hypotheses. Second, we demonstrate that our approach can solve the first 68 levels (which cover all available construction tools) of the geometric construction game Euclidea with 92% accuracy. Finally, we show that our approach can also solve new problems, unseen at training. The system as well as our modified Euclidia environment are available online.[5]

The rest of the paper is structured as follows. Section 2 gives a brief overview of related work. Section 3 presents our Euclidea environment. Section 4 describes the methods we developed to solve problems in the supervised setting. This includes a description of the neural image recognition methods and their modifications for our tasks. Section 5 describes our methods for solving new problems, unseen during the training. This includes generating sets of proposed steps and

[4] https://imo-grand-challenge.github.io/

[5] github.com/mackej/Learning-to-solve-geometric-construction-problems-from-images, github.com/mirefek/py_euclidea/

searching the tree of possible constructions. Section 6 evaluates the methods on levels seen and unseen during the training.

2 Related Work

Visual recognition techniques can be used for interpreting a geometrical question given by a diagram. Such a geometry solver was proposed in [4,5]. The input problem is specified by a diagram and a short text. This input is first decoded into a formal specification describing the input entities and their relations using visual recognition and natural language processing tools. The formal specification is then passed to an optimizer based on basin hopping. In contrast, we do not attempt to convert the input problem into a formal specification but instead use a visual recognizer to directly guide the solution steps with only images as input.

The most studied geometry problems are those where the objective is to find a proof [6]. This contrasts with our work, where we tackle construction problems. An algebraic approach to a specific type of construction problem is used by Argotrics [7]. This is a Prolog-based method for finding constructions that satisfy given axiomatically proven propositions.

Automated provers for geometry problems are mostly of two categories. They are either synthetic [8,9], i.e., they mimic the classical human geometrical reasoning, and prove the problems by applying predefined sets of rules / axioms (similar triangles, inscribed angle theorem, etc). The other type of solvers are based on algebraic methods such as Wu's method [10] or the Gröbner basis method [11]. These have better performance than the synthetic ones but do not provide a human readable solution. There are also methods combining the two approaches. They may for example use some algebra but keep the computations simple (the Full angle method or the Area method [12]). We cannot compare directly with such provers because of the constructional nature of the problems we study. However, our approach is complementary to the above methods and can be used, for example, to suggest possible next construction steps based on the visual configuration of the current scene.

Automated theorem provers (ATPs) such as Otter [13] and Prover9 [14] have been used for solving geometric problems, e.g., in Tarskian geometry [15,16,17]. Proof checking in interactive theorem provers (ITPs) such as HOL Light and Coq has been used to verify geometric proofs formally [18]. Both ATPs and ITPs have been in recent years improved by using machine learning and neural guidance [19,20]. ATPs and ITPs however assume that a formalization of the problem is available, which typically includes advanced mathematical education and nontrivial cognitive effort. The formal representations are also closer to text, which informs the choice of neural architectures successfully used in ATPs and ITPs (GNNs, TNNs, Transformers). In contrast, we try to skip the formalization step and learn solving geometric construction problems directly from images. This also means that our methods could be used on arbitrary informal images, such as human geometry drawings, creating their own internal representations.

3 Our Euclidea geometric construction environment

Euclidea is an online geometric construction game in 2-dimensional Euclidean space. The main goal is to find a sequence of construction steps leading from an initial configuration of objects to a given goal configuration. The construction steps utilize a set of straightedge and compass-based tools (see Table 1). Every tool takes up to 3 arguments with values specified by the coordinates of clicks on the image of the scene, e.g., circle(A, B), where A, B are points in the image. Euclidea is divided into 15 level packs (Alpha, Beta, Gamma, ..., Omicron) with increasing difficulty; each level pack contains around 10 levels with a similar focus. In Euclidea, each level has its analytical model, which is projected onto an image and the player does not have access to this model, only to the image. Each construction is validated with the analytical model to prevent cheating by drawing lines or circles only similar to the desired goal.

In addition, our Euclidea environment can also generate new instances of the levels. A new instance is generated by randomly choosing initial parameters of the level inside Euclidea. However, some of these instances can be "degenerate", i.e., unsolvable based on the image data. To prevent such degenerate configurations, we enforce multiple constraints, e.g., that different points cannot be too close to each other in the image or that a circle radius cannot be too small. We use this process of generating new problem instances for collecting examples to train our model.

4 Supervised visual learning approach

This section describes our method for learning to solve geometric problems. We build on Mask R-CNN [3], a convolutional neural network for the detection and segmentation of objects in images and videos. Given an image, Mask R-CNN outputs bounding boxes, segmentation masks, class labels and confidence scores of objects detected in the input image.

Mask R-CNN is a convolutional neural network architecture composed of two modules. The first module is a region proposal network that proposes candidate regions in the image that may contain the target object (e.g. a "car"). The second module then, given a proposed candidate region, outputs its class (e.g. "car", "pedestrian" or "background"), bounding box, segmentation mask and confidence score.

We adapt the Mask R-CNN model for the task of solving geometric construction problems. Fig. 2 shows the diagram of our approach. The main idea is to train Mask R-CNN to predict the tool that should be used at a given step, including its arguments. For example, as shown in Fig. 2, the input is the image depicting the current state of the construction in the red channel of the image (three points in red) and the goal in the green channel (the three sides of the triangle). Mask R-CNN predicts here to execute the Line tool. The predicted bounding box of the line is shown in magenta. For this purpose, Mask R-CNN network has to recognize the two points in the input image and predict their location, represented by the rectangular masks. The output masks are then

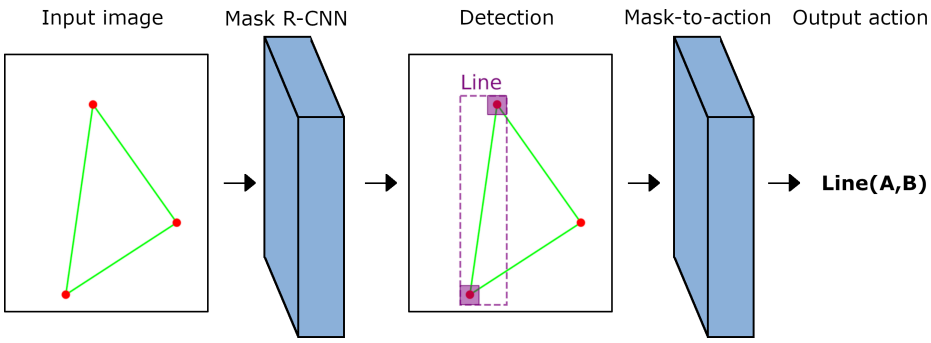


Fig. 2: Diagram of our approach. The goal is to construct a triangle given three points. The input is an RGB image with the current state of the construction in the red channel (the three points) and the goal is given in the green channel (the three sides of the triangle). The Mask R-CNN predicts a line between two of the three points. The dashed rectangle denotes the bounding box of the line and the small square magenta masks denote the two points on the line. This Mask R-CNN line detection is then converted into a Euclidea tool action, $Line(A, B)$, represented by the tool name and its arguments (the locations of the two points). The process that converts the Mask R-CNN output masks into actions in the Euclidea environment is described in Section 4.2.

transformed to coordinates of the two points that need to be "clicked" to execute the Line tool in the Euclidea environment.

To train Mask R-CNN to solve geometric construction problems, we have to create training data that represent applications of the Euclidea tools and adjust the network outputs to work with our Euclidea environment. To generate training data for a given Euclidea level, we follow a predefined construction of the level and transform it to match the specific generated level instances (see Section 3). Each use of a Euclidea tool corresponds to one example in the training data. We call each application of a tool in our environment an *action*, represented by the tool name and the corresponding click coordinates. For example, the Line tool needs two action clicks, representing two points on the constructed line.

The following sections show the generation of training data for Mask R-CNN (Section 4.1), describe how we derive Euclidea actions from the detected masks at test time (Section 4.2), present our algorithm for solving construction problems (Section 4.3), and introduce additional components that improve the performance of our method (Section 4.4).

4.1 Action to Mask: generating training data

Here we explain how we generate the training data for training Mask R-CNN to predict the next construction step. In contrast to detecting objects in images where object detections typically do not have any pre-defined ordering, some of the geometric tools have non-interchangeable arguments and we will have to modify the output of Mask R-CNN to handle such tools.

We represent the Mask R-CNN input as a 256×256 RGB image of the scene with the current state in the red channel and the remaining goal in the green channel; the blue channel contains zeros. Note that for visualization purposes, we

render the black background as white. Each training example consists of an input image capturing the current state of the construction together with the action specifying the application of a particular tool. The action is specified by a mask and a class, where the class identifies the tool (or its arguments, see below) and the mask encodes the location of each point click needed for application of the tool in the image, represented as a small square around each click location. The Perpendicular tool and Parallel tool have a line as their argument (see Table 1), passed as a mask of either the whole line or one click on the line.

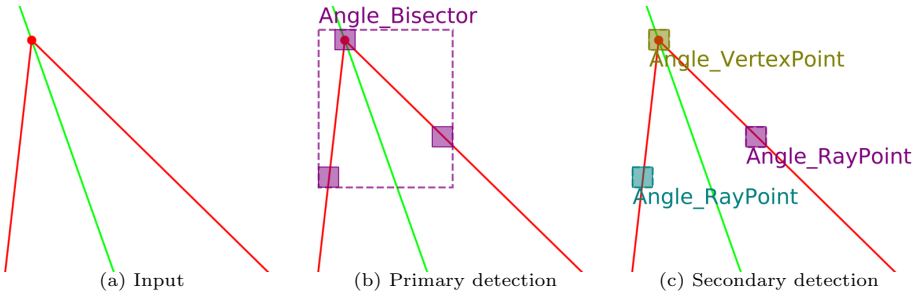


Fig. 3: An example from training data for Euclidea level *Beta-02* (construct the line that bisects the given angle). The current state is in red and the remaining goal in green. (a) Input for the Mask R-CNN model. (b) Primary detection of the Mask R-CNN model identifying the tool type: Angle Bisector tool (purple). (c) Three secondary detections identifying the arguments of the tool: one angle vertex point (yellow) and two angle ray points (purple and turquoise).

Primary and secondary detections. Encoding clicks as in the previous paragraph is not sufficient for tools with non-interchangeable parameters. For example, the Circle tool has two non-interchangeable parameters: 1) the center and 2) a point on the circle, defining the radius. To distinguish such points, we add a secondary output of the Mask R-CNN model. For example, for the Circle tool, we detect not just the circle itself but also the circle center and the radius point. We denote the detection of the tool (Line tool, Circle tool, ...) as the *primary detection*, and the detection of its parameters as the *secondary detections*. Fig. 3 shows an example of primary and secondary detections for the Angle Bisector tool, including the corresponding classes: Angle_Bisector (primary detection), Angle_VertexPoint, and Angle_RayPoint (secondary detections).

4.2 Mask to Action: converting output masks to Euclidea actions

To solve Euclidea levels, we have to transform the Mask R-CNN output to fit the input of the Euclidea environment. We refer to this step as "mask-to-action" as it converts the output of Mask R-CNN, which is in the form of image masks specifying the primary and secondary detections (see Section 4.1), into tool actions in the Euclidea environment. The mask-to-action conversion consists of two stages. The first stage obtains locations of individual "point clicks" from

the primary detections for the predicted tool and the second stage determines the order of parameters using the secondary detections.

First stage: To localize individual points, we use the heat map produced by the final Mask R-CNN layer. The heat map assigns each pixel the probability of being a part of the mask and can be transformed into a binary mask by thresholding. Instead, we use the heat map directly to localize the detected points more accurately. We select points with the highest probability in the masks using a greedy non-maximum-suppression method [21].

Second stage: We will explain this stage on the example of the Angle Bisector tool (see Fig. 3). A detection of this tool has 4 detection outputs from Mask R-CNN, namely, 1 primary and 3 secondary detections. The primary detection corresponds to the whole tool and the secondary detections to the individual points, i.e., one angle vertex point and two angle ray points (see Fig. 3). To use the Angle Bisector tool, we have to determine the correspondence between the primary and secondary detections. We obtain 3 point coordinates from the primary detection in the first stage, as described above. We can also get 3 points from the 3 secondary detections, one point per detection. Each point in the primary detection should correspond to one point in the secondary detection, but these points may not exactly overlap. The point correspondence is determined by finding a matching between the primary and secondary points that minimizes the sum of distances between the primary and secondary points such that each point is used exactly once.

4.3 Solving construction problems by sequences of actions

Next, we can create an agent capable of solving Euclidea construction problems. In the previous section, we have described how to get a Euclidea action from Mask R-CNN outputs. However, Mask R-CNN can predict multiple candidate detections (that correspond to different actions) for one input image. Mask R-CNN returns for each detection also its score, representing the confidence of the prediction. To select the next action from the set of candidate actions (derived from Mask R-CNN detections) in each step, the agent follows Algorithm 1, which chooses the action with the highest confidence score at each state.

Result: Test level inference: True if level completed, False otherwise.
Initialize a level;
while *level not complete* **do**
  $s \leftarrow$ current state of the level;
  $p \leftarrow model.predict(s)$;
  if *predictions* p *are empty* **then**
    return False;
  $a \leftarrow$ action from p with highest score;
  execute a
return True;

Algorithm 1: Solving construction problems by choosing the action with the highest score.

4.4 Additional components of the approach

Here we introduce several additional extensions to the approach described above and later demonstrate their importance in Section 6.

Automatic point detection. Our Euclidea environment requires that each point important for the solution is identified using the Point tool. For example, when we have to find the third vertex of the triangle in Fig. 1, we have to use the Point tool to localize the intersections of the circles. The Automatic point detection modification automatically adds points to the intersections of objects.

History channel. To better recognize which construction steps have already been done and which still need to be constructed, we add a third, history channel (blue) to the input of Mask R-CNN, containing the construction state from the previous step.

4+ Stage training. Mask R-CNN is typically trained in 2 stages: first, only the head layers are trained, followed by training of the whole network, including the 5-block convolutional backbone. The 4+ Stage training modification splits the training into 3 stages: first, the head layers are trained, then also the fourth and fifth backbone blocks, and finally, the whole network.

Intersection degeneration rules. To decide whether a generated level can be solved using only the image information, we apply the following rules to identify degenerate configurations: a) the radius of a circle cannot be too small, b) the distance between points, lines, or their combinations cannot be too small. In this modification, we add a third rule: c) any intersection of geometric primitives cannot be too close to points that are necessary for the construction. This prevents possible alternative solutions from being too close to each other and the auxiliary intersections created during the construction from being too close to points from the initial state and the goal.

On-the-fly data generation. Generating training data on-the-fly allows us to (potentially infinitely) expand the training set and thus train better models.

5 Solving unseen geometric problems via hypothesis tree search

In the previous section, we have shown how to train a visual recognition model to predict the next step of a given construction from a large number of examples of the same construction with different geometric configurations of the primitives. In this section, we investigate how to solve new problems, which we have not seen at training time. This is achieved by (i) using models trained for different construction problems (see Section 4) to generate *a set of hypotheses* for each construction step of the new problem and then (ii) searching the tree of possible constructions. These two parts are described next.

5.1 Generating action hypotheses

Each primary detection from the Mask R-CNN model (see Section 4.1) can be transformed into an action. We denote each action, its arguments, and results as

a *hypothesis*. The result of an action contains a geometric primitive, constructed during the action execution, and a reward, indicating whether the output primitive is a part of the remaining goal or not. If an action constructs a part of the goal, the reward is $1/n$, where n is the total number of primitives in the initial goal, otherwise it is equal to zero. Fig. 4b shows a hypothesis that successfully constructs one of the four lines in the goal and its reward thus equals 0.25.

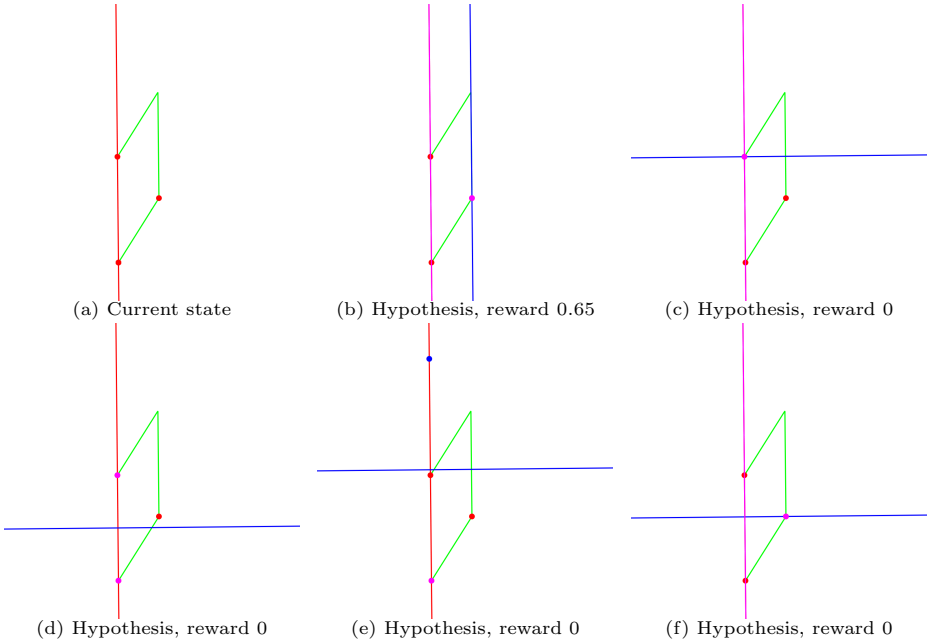
We can extract multiple actions from the Mask R-CNN model output by considering multiple output candidate detections, transform them into multiple hypotheses, and explore their construction space. We can also utilize hypotheses from models trained for different tasks. However, the Mask R-CNN scores are not comparable across hypotheses from different models, so in a setup with multiple models we have to search even hypotheses with lower scores.

5.2 Tree search for exploring construction hypotheses

We use tree search to explore the hypothesis space given by the predictions from one or more Mask R-CNN models. The tree search has to render the input image and obtain predictions from all considered Mask R-CNN models in each node of the tree, which increases the time spent in one node. However, as a result, we search only within the space of hypotheses output by the Mask R-CNN models, which is much smaller than the space of all possible constructions. We use iterative deepening, which is an iterated depth-limited search over increasing depth (see Algorithm 2).

```
Result: Solve level with Iterative Deepening.
Function IterativeDeepening-DFS(InitialState):
    SolutionMaxLength ⟵ 0;
    Solution ⟵ null;
    while (Solution = null) and (SolutionMaxLength < MaxIterationDepth)
     do
        SolutionMaxLength ⟵ SolutionMaxLength + 1;
        Solution ⟵ FindSolution(InitialConfig, SolutionMaxLength);
    return Solution
Function FindSolution(CurrentState, Depth):
    if CurrentState.IsTheGoal then
        return success // collect solution on the backtracking
    if Depth = 0 then
        return null // e.g., solution not found.
    Hypotheses ⟵ Models.GetAllHypotheses(CurrentState);
    // Hypotheses sorted in the order: Reward, Confidence score
    foreach h ∈ Hypotheses do
        NewState ⟵ Apply(h, CurrentState);
        Solution = FindSolution(NewState, Depth − 1);
        if Solution ≠ null then
            return Solution
    return null
```

Algorithm 2: Tree search for exploring construction hypotheses using Iterative Deepening Depth-First Search algorithm.



(a) Current state (b) Hypothesis, reward 0.65 (c) Hypothesis, reward 0

(d) Hypothesis, reward 0 (e) Hypothesis, reward 0 (f) Hypothesis, reward 0

Fig. 4: Multiple hypotheses for the next step in Euclidea level *Epsilon-03* (construct a parallelogram whose three of four vertices are given). (a) Current state. (b) Hypothesis with reward 0.25 that constructs one of the four lines in the goal (green). (c-f) Hypotheses with reward 0. Each image contains the current state (red), remaining goal (green), hypothesis produced by Mask R-CNN (blue), and parameters of the tool (purple). For example, hypothesis (b) selects the Parallel tool to construct the blue line as parallel to the purple line and intersecting the purple point. A positive reward indicates contribution to the goal, so we select the hypothesis (b), which has the highest reward.

Hypotheses produced by different Mask R-CNN models increase the branching factor and thus also the search time. To speed up the tree search, we group all mutually similar hypotheses and explore only one of them. For this purpose, we consider two hypotheses as similar if their output geometric primitive is the same; note that such hypotheses can have different arguments, including different tools.

6 Experiments

This section shows the performance of our method for both the *supervised setting*, where we see examples of the specific level in both training and test time, and the *unseen setting*, where we are testing on new levels, not seen during training. We will compare the benefits of the different components of our approach and show an example solution produced by our method.

6.1 Benefits of different components of our approach

As the base method we use the off-the-shelf Mask R-CNN approach ("Vanilla Mask R-CNN"), which has a low accuracy even on simple Alpha levels. To improve over this baseline, we have introduced several additional components, namely, "Automatic point detection", "History channel", "4+ Stage training", "Intersection degeneration rules", and "On-the-fly data generation" (see Section 4.4). Table 2 compares their cumulative benefits. These components are crucial for solving levels in advanced level packs, e.g., the Gamma level pack, which could not be solved without the Intersection degeneration rules.

Component / Level pack	Alpha	Beta	Gamma	Delta	Epsilon	Zeta
Vanilla Mask R-CNN [3]	71.0	-	-	-	-	-
+ Automatic point detection	95.1	-	-	-	-	-
+ History channel	98.1	69.3	-	-	-	-
+ 4+ Stage training	91.7	82.3	-	-	-	-
+ Intersection degeneration rules	91.7	82.3	79.9	73.5	-	-
+ On-the-fly data generation	**98.7**	**96.2**	**97.8**	**99.1**	**92.8**	**95.7**

Table 2: Performance of the base method (Vanilla Mask R-CNN) together with the additional components of our approach. The Performance is evaluated on Euclidea level packs with increasing difficulty (from Alpha to Zeta). We trained a separate model for every level in each level pack and evaluated the model on 500 new instances of that level. The table presents the accuracy averaged across all levels in each level pack. The components are applied to the base method (Vanilla Mask R-CNN) in an additive manner. For example, the 4+ Stage training includes all previous components, i.e., Automatic point detection and History channel. A description of the different components is given in Section 4.4.

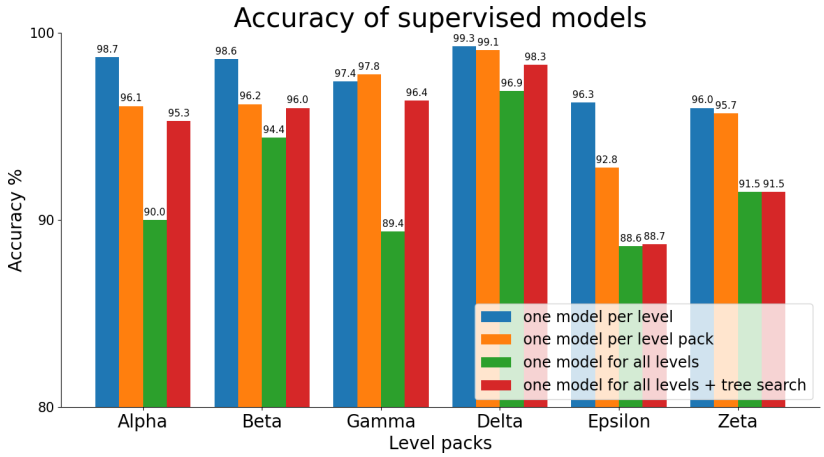


Fig. 5: Accuracy of our approach on Euclidea level packs Alpha to Zeta. The model uses all components from Table 2. We compare four approaches: one model per level (blue), one model per level pack (orange), one model for all levels (green), and one model for all levels with hypothesis tree search (red). All were evaluated on 500 instances of each level and the accuracy was averaged across all levels in each level pack.

6.2 Evaluation of the supervised learning approach

We evaluated our method on the first six level packs of Euclidea with various training setups. The results (see Fig. 5) show that models specialized to individual level packs ("one model per level pack") or even levels ("one model per level") have better accuracy than a single, more general model for multiple levels/packs ("one model for all levels"). We also investigate the benefits of using our tree search procedure (see Section 5.2) instead of using only the most confident hypothesis, as in the supervised setting. We find the tree search improves the accuracy, especially on Alpha and Gamma level packs, by searching the space of possible candidate solutions for each step of the construction.

6.3 Evaluation on unseen problems

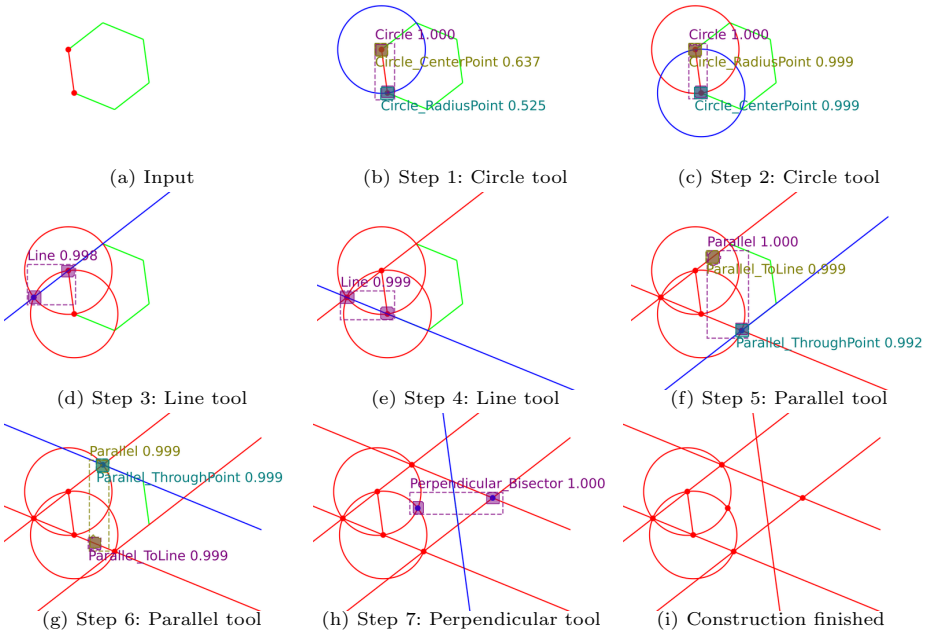
To evaluate performance on unseen levels, we use the leave-one-out (LOO) evaluation. In our setup, *LOO levels* evaluates, e.g., level *Alpha-01* using models trained on each of the other Alpha levels, whereas *LOO packs* evaluates, e.g., each Alpha level using models trained on level packs Beta, Gamma, Delta, Epsilon, and Zeta. Table 3 compares the LOO evaluation and the supervised approach, both with the hypothesis tree search, for level pack Alpha. We ran a similar evaluation for all 6 levels packs and were able to solve 30 out of 68 levels using *LOO levels* and 31 out of 68 levels using *LOO packs*. The results show that our method can solve many levels unseen during the training, although some levels remain difficult to solve.

Alpha levels	LOO levels	LOO packs	supervised
01 T1 Line	40.0	10.0	85.0
02 T2 Circle	5.0	45.0	100.0
03 T3 Point	100.0	90.0	100.0
04 TIntersect	100.0	100.0	99.0
05 TEquilateral	50.0	70.0	100.0
06 Angle60	55.0	100.0	94.0
07 PerpBisector	35.0	100.0	99.0
08 TPerpBisector	0.00	100.0	75.0
09 MidPoint	5.0	60.0	100.0
10 CircleInSquare	0.00	100.0	87.0
11 RhombusInRect	25.0	40.0	99.0
12 CircleCenter	10.0	0.00	100.0
13 SquareInCircle	0.00	10.0	100.0
Average	42.5	63.4	95.3

Table 3: Leave-one-out evaluation on the Alpha levels. Completion accuracy of the leave-one-out evaluation across levels within a level pack (LOO levels), across level packs (LOO packs), and, for comparison, our best supervised model trained to solve the first six Euclidea level packs (Alpha-Zeta); the tree search was used in all three cases. The leave-one-out evaluation was performed on 20 instances of each level, while the supervised model was evaluated on 500 instances of each level. Using models from all level packs except Alpha (LOO packs) works better than using models trained only on other levels of the Alpha level pack (LOO levels). This is likely because models in the "LOO packs" set-up have seen a larger set of different constructions.

6.4 Qualitative example

Fig. 6 shows a step-by-step walk-through construction of an advanced Euclidea level.



(a) Input (b) Step 1: Circle tool (c) Step 2: Circle tool

(d) Step 3: Line tool (e) Step 4: Line tool (f) Step 5: Parallel tool

(g) Step 6: Parallel tool (h) Step 7: Perpendicular tool (i) Construction finished

Fig. 6: Example construction of Euclidea level *Epsilon-12* (construct a regular hexagon by the side). (a) Initial configuration of the problem. (b-h) Seven construction steps, including Mask R-CNN detections, and an object proposed to construct in each step. (i) Final construction step, level solved. Red denotes the current state of the construction, green the remaining goal, blue the geometric primitive proposed by the detection, and other colors the prediction masks, bounding boxes, class names, and scores for the next predicted action. More examples can be found in the supplementary material available at the project webpage [1].

Connections between levels. From the leave-one-out evaluation, we can also observe which levels are similar. We denote that level X is *connected* to level Y if a model trained for Y contributes with a hypothesis to a successful construction during the inference for level X. Note that this relation is not symmetric, e.g., when X is connected to Y, then Y is not necessarily connected to X. We run the hypothesis tree search during the leave-one-out evaluation and obtain connections in the following way: If the search is successful, we collect all models that contributed to the solution in the final backtracking of the search. The connections for all levels in the level pack Alpha are shown in Fig. 7.

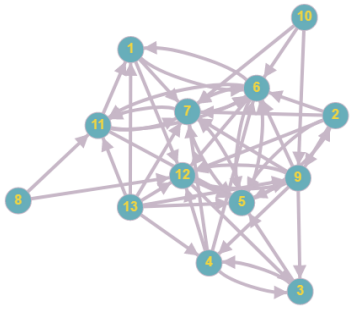


Fig. 7: Connection graph created during the leave-one-out evaluation of the Alpha level pack using models for individual levels. Models trained on more difficult levels (indicated by higher numbers) often help construct easier levels (lower numbers). For example, level 13 could not be solved (has no incoming connections), but its predictions were used for solution of levels 1, 4, 5, 7, 9, 11, 12. Our method can often construct simpler tasks based on more complex constructions. However, combining multiple simple tasks into a complex construction remains a challenge.

7 Conclusion

We have developed an image-based method for solving geometric construction problems. The method builds on the Mask R-CNN visual recognizer, which is adapted to predict the next step of a geometric construction given the current state of the construction, and further combined with a tree search mechanism to explore the space of possible constructions. To train and test the method, we have used Euclidea, a construction game with geometric problems with an increasing difficulty. To train the model, we have created a data generator that generates new configurations of the Euclidea constructions.

In a supervised setting, the method learns to solve all 68 kinds of geometric construction problems from the first six level packs of Euclidea with an average 92% accuracy. When evaluated on new kinds of problems unseen at training, which is a significantly more challenging set-up, our method solves 31 of the 68 kinds of Euclidea problems. To solve the unseen problems our model currently relies on having seen a similar (or more complex) problem at training time. Solving unseen problems that are more complex than those seen at training remains an open challenge. Addressing this challenge is likely going to require developing new techniques to efficiently explore the space of constructions as well as mechanisms to learn from successfully completed constructions, a set-up similar to reinforcement learning.

Although in this paper we focus on synthetically generated data, our results open up the possibility to solve even hand-drawn geometric problems if corresponding training data is provided. In real-world settings, descriptions and discussions of problems in math, physics and other sciences often contain an image or a drawing component. Diagrams and illustrations are also an essential part of real-world technical documentation (e.g. patents). The ability to automatically

identify and combine visually defined geometrical primitives into more complex patterns, as done in our work, is a step towards automatic systems that can identify compositions of geometric patterns in technical documentation. Think, for example, of an "automatic patent lawyer assistant".

References

1. Project webpage. https://data.ciirc.cvut.cz/public/projects/2021GeometryReasoning.
2. Euclidea. https://www.euclidea.xyz.
3. He K., Gkioxari G., Dollár P., and Girshick R. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.
4. Seo M. J., Hajishirzi H., Farhadi A., and Etzioni O. Diagram understanding in geometry questions. *American Association for Artificial Intelligence*, 2014.
5. Seo M., Hajishirzi H., Farhadi A., Etzioni O., and Malcolm C. Solving geometry problems: Combining text and diagram interpretation. *Association for Computational Linguistics*, 2015.
6. Quaresma P. Thousands of geometric problems for geometric theorem provers (TGTP). Springer Berlin Heidelberg, 2011.
7. Vesna M. ArgoTriCS – Automated Triangle Construction Solver. *Journal of Experimental & Theoretical Artificial Intelligence*, 29(2), 2017.
8. Balbiani P. and Cerro L. F. Affine geometry of collinearity and conditional term rewriting. Springer Berlin Heidelberg, 1995.
9. Chou S. C., Gao X. S., and Zhang J. Z. A deductive database approach to automated geometry theorem proving and discovering. *Journal of Automated Reasoning*, 2000.
10. Gao X. Transcendental functions and mechanical theorem proving in elementary geometries. *Journal of Automated Reasoning*, 1990.
11. Deepak K. Using Gröbner bases to reason about geometry problems. *Journal of Symbolic Computation*, 2(4), 1986.
12. Chou S. C., Gao X. S., and Zhang J. Machine proofs in geometry: Automated production of readable proofs for geometry theorems. 1994.
13. McCune W. and Wos L. Otter: The CADE-13 Competition Incarnations. *Journal of Automated Reasoning*, (2), 1997.
14. McCune W. Prover9 and Mace4. http://www.cs.unm.edu/~mccune/prover9/.
15. Beeson M. and Wos L. Finding proofs in Tarskian geometry. *Journal of Automated Reasoning*, 58(1), 2017.
16. Stojanovic Durdevic S., Narboux J., and Janicic P. Automated generation of machine verifiable and readable proofs: A case study of Tarski's geometry. *Annals of Mathematics and Artificial Intelligence*, 74(3-4), 2015.
17. Quaife A. *Automated Development of Fundamental Mathematical Theories*. Kluwer Academic Publishers, 1992.
18. Beeson M., Narboux J., and Wiedijk F. Proof-checking Euclid. *Annals of Mathematics and Artificial Intelligence*, 85(2-4), 2019.
19. Jakubuv J., Chvalovský K., Olsák M., Piotrowski B., Suda M., and Urban J. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In *IJCAR*, volume 12167 of *LNCS*, pages 448–463, 2020.
20. Gauthier T., Kaliszyk C., Urban J., Kumar R., and Norrish M. TacticToe: Learning to prove with tactics. *Journal of Automated Reasoning*, 65(2):257–286, 2021.
21. Hosang J., Benenson R., and Schiele B. Learning non-maximum suppression. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.