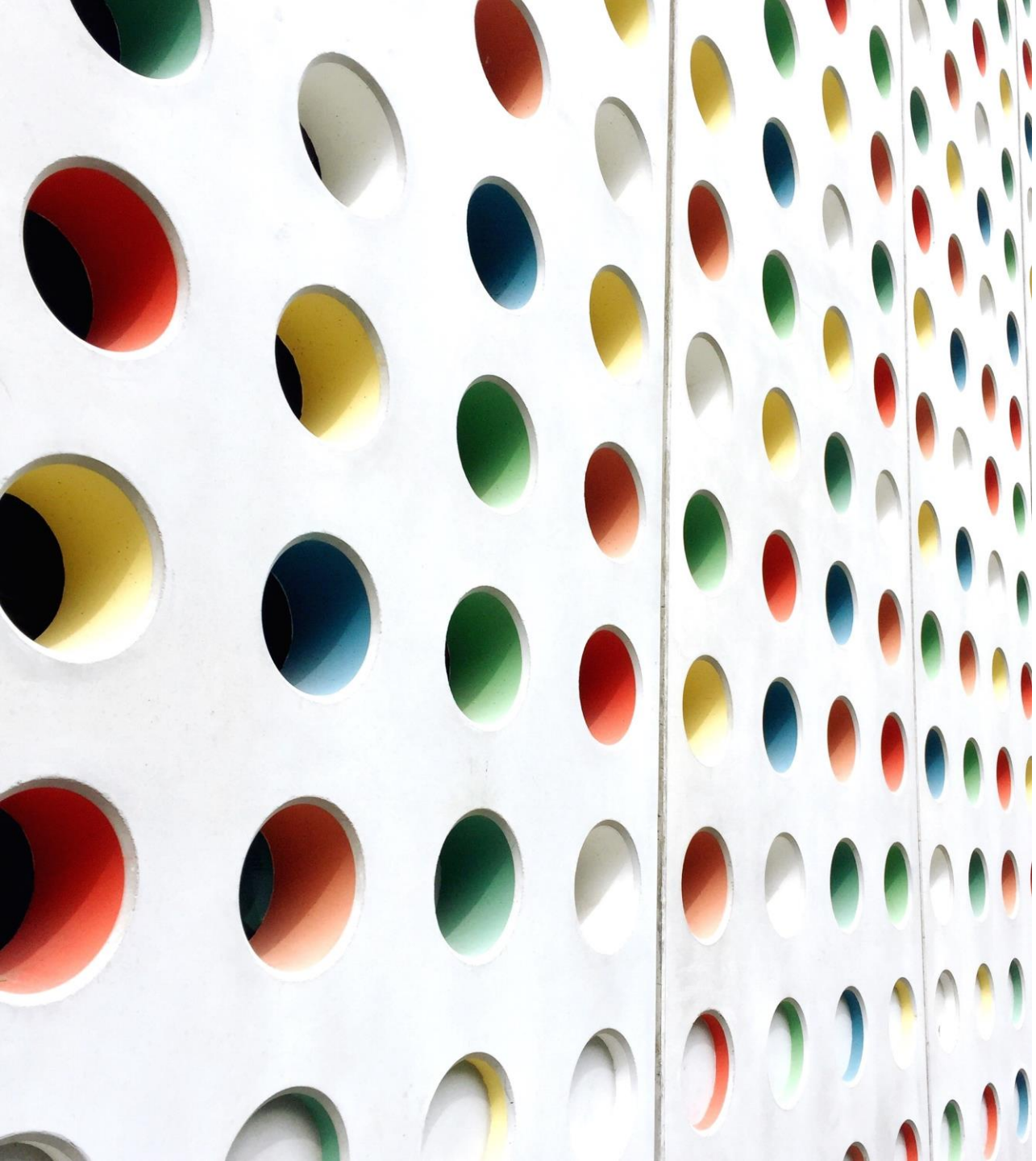


TransPath: Comparison with Neural A*

LIDIA FARGUETA PELUFO

Index

- Some background
- Goals to achieve (and achieved) with TransPath
- Some characteristics in TransPath
- The method:
  - Heuristics learned
  - Learning supervision
- Neural Network Architecture
- Training and evaluation
- Differences with Neural A*

Some background…

„As grids can be viewed as the binary images, it is appealing to utilize the recent advances in CNNs to extract the informative features from the image representations of the pathfinding problems and embed these features to the heuristic search algorithm“ → In **Neural A* paper**

A matrix-based A* was proposed and used for learning. Thus, the deep neural network model was trained end-to-end. In that work not the conventional cost-to-go heuristic was predicted but rather the additional cost that was assigned to each grid cell with the intuition that unpromising nodes will be assigned a high cost by the neural network.. Consequently at the planning phase the search will avoid the cells with the high costs.

Goals to achieve (and achieved) with TransPath

1) Reduce the computational effort of the A* up to a factor of 4x.

2) Outperform the competitors

„The performance of heuristic search algorithms is heavily dependent on the input heuristic that comes in the form of a function that estimates the cost of the path to the goal for each node of tha graph.“ (cost-to-go heuristics)

Some characteristics in TransPath…

We follow the paradigm of Neural A* and extended it. Some charactistics from this model:

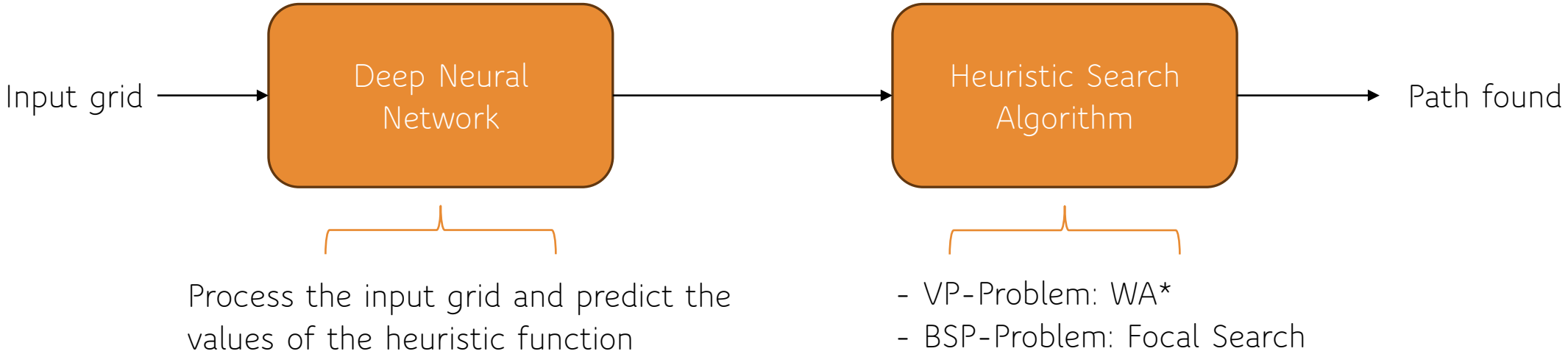
- 8-connected grids with non-uniform costs.
- We don't learn to predict the values of the perfect cost-to-go heuristic, we suggest learning **the correction factor** and the **path probability map**.
  - Correction factor: Ratio between instance-independent heuristic (Manhattan distance) and the perfect heuristic (learned)
  - Path probability map: Assigns to each grid cell the probability of belonging to the shortest path.
- To learn we use supervised deep learning:
  - CNN + Transformers: To find local features and relations between them.
- Dataset: extends the one used at Neural A* paper.

The method

Two types of problems:

1) Find a valid path on a grid without specifying any constraints on the cost of the path → **VP-Problem**

2) A suboptimality bound w>=1 is specified and the task is to find a path whose cost does not exceed the cost of the optimal path by more than a factor of „w“ → **BSP- Problem**



The method: heuristics learned

- Correction factor: $cf(n) = \frac{h(n)}{h^*(n)}$

  $\frac{h(n)}{h^*(n)} \rightarrow$ Instance-independent heuristic

  $\frac{h(n)}{h^*(n)} \rightarrow$ Perfect heuristic

For the WA* algorithm:

$$f(n) = g(n) + \frac{h(n)}{cf(n)} = g(n) + \frac{h(n)}{w(n)}$$

- The range of the correction factor is [0, 1], no auxiliary transformations needed.
- More heuristic information (combination of instance-dependent and instance-independent heuristics).

The method: heuristics learned

- Path probability: How likely it is that this cell is lying on the shortest path between start and goal.

$$pp - values(n) \in [0, 1]$$

Used as h_focal values in the FS (Focal Search).

The method: learning supervision

Two types:

1) Create a rich dataset of pathfinding instances with the annotated ground-truth cf-values and pp-values and learn the neural network to minimize the error between its prediction and ground-truth values. No search/planning is happening at the training phase. (our type)

2) Include the search algorithm in the learning pipeline and to backpropagate the search error through it. (when it is hard or impossible to create ground-truth samples).→ Neural A* paper

- Not a problem to create ground-truth values for cf and pp.
- Learning without differentiable planner is much faster (4x faster)
- Learning the suggested model with the supervision from the ground-truth values leads to a consistently better performance

Learning supervision: creating ground-truths

1) For cf-values: uninformed search that starts in the goal location on a map and computes true distances to the goal from any cell.

2) For pp-values: We run Theta*, an any-angle search algorithm that can be though of as A* with online path smoothing. We assign the values of 1 for such paths in the resultant PPM. For all other cells, we computed the value that tells us how close the cost of the path through cell n to the one of Theta* is:

$$cost(\pi(s,g))/cost(\pi(s,n)) + cost(\pi(n,g))$$

If the value was greater than or equal to 0.95, we used it as the pp-value; if not, the pp-value was set to 0.

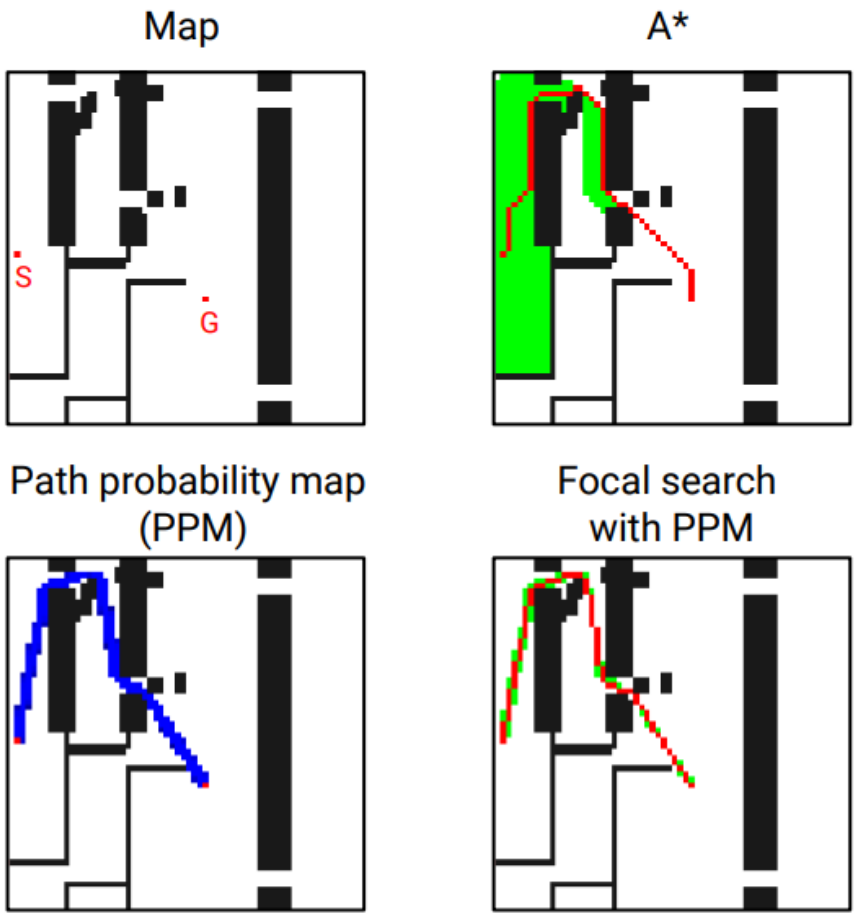


Figure 1: The difference between A* and our approach. Expanded nodes are shown in green, while path in red. Blue regions are predicted by the neural network to contain path cells with high probability.

Neural network architecture

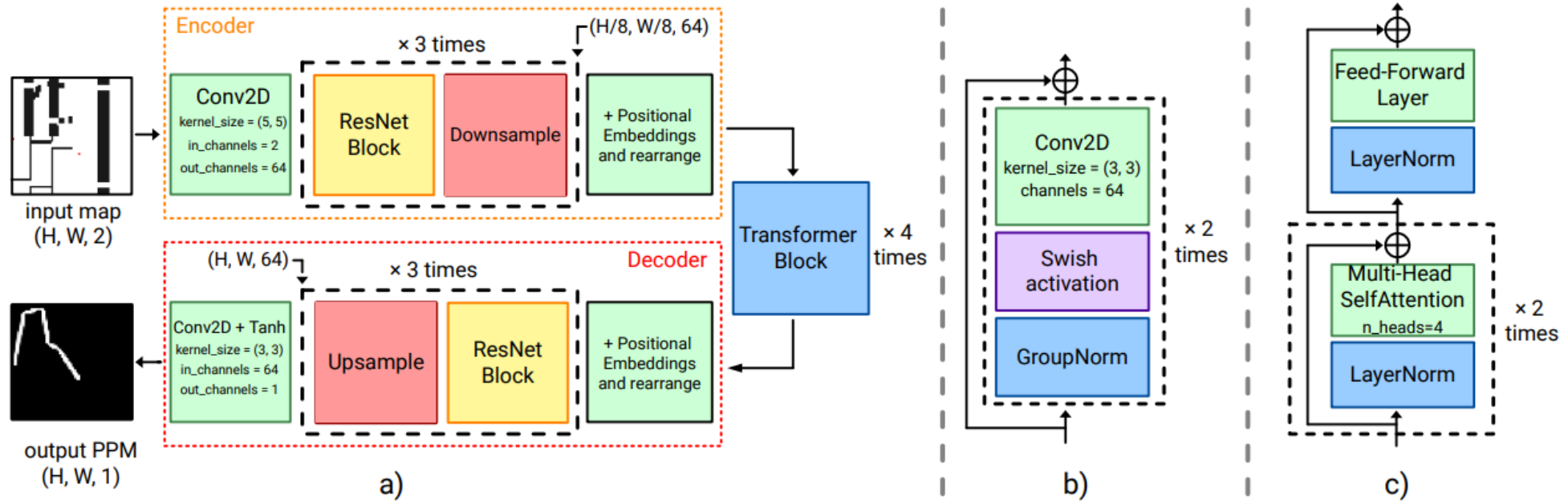


Figure 2: Overview of the neural network architecture. a) Design of the whole model. CNN-encoder is used to produce local features which are further fed into the transformer blocks to catch the long-range dependencies between the features. The resulting representation is passed through the CNN-decoder to produce output values. b) Architecture of the ResNet block. c) Architecture of the Transformer block.

Neural network architecture

The input is slightly different for learning cf-values or pp-values:

- **For pp-values** the input contains the grid, represented as a binary iage and the start-goal matrix, which has the same size as the grid and contains the values of 1 only for start and goal, while all other pixels are zeroes.

- **For cf-values** this matrix contains only one non-zero element – the goal one.

Training and Evaluation

Dataset: Tiled Motion Planning (TMP), the one that was used in the Neural A* paper. We increased the size of the dataset (from 4000 to 64000) via augmentation by mirroring and rotating each of the four parts of the TMP maps. For each map we generated 10 problem instances. Overall 640000 instances were generated. (8:1:1, train, validation and test subsets).

Planners: WA*+CF (Weighted A* with the correction factor), FS+PPM (Focal Search with Path Probability Map) and Greedy Best First Search with PP (GBPS+PPM). They used the official code of Neural A* and modified it to handle non-uniform move costs. They employed the designed neural network model in Neural A* to provide a fairer comparison.

Training setup: Adam optimizer, 35 epochs, batch size 512, OneCycleLR learning rate scheduler at a maximum learning rate of 4x10^-4. 3.5h to train on NVDIA A100 80GB GPU. Neural A* with our training data took 16 hours to complete the training.

Differences with Neural A*

- Neural A* is fully implemented in Python, while WA*+CF, FS+PPM and GBFS+PPM feature both Python for neural networks' machinery and C++ for the search itself.

- Now we have non-uniform costs instead of unit cost domains.

- The search algorithm is not included in the learning pipeline, as it was in the Neural A* algorithm.

- We learn the correction factor and the path probability instead of directly the perfect heuristic value.

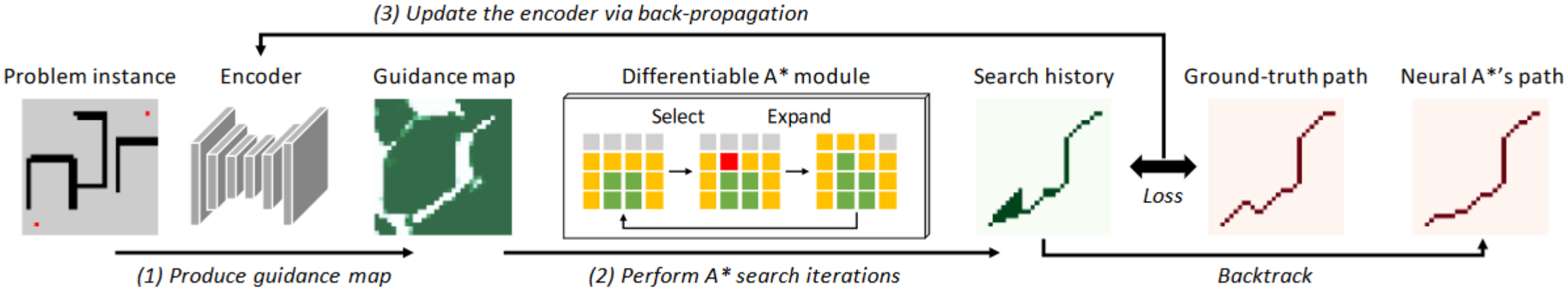
- Now the architecture is formed by the convolutional encoder block with ResNet, the ViT Transformer and the convolutional decoder. Before, it was a U-Net with the VGG-16 backbone.

- The mentioned planners with this new architecture is 4 times faster than the Neural A* for the respective architecture.

Differences: training characteristics

Planners	Size of dataset	Optimizer	Mini-batch size	Epochs	Learning rate
Neural A*	4000	RMSProp	100	400	0.001
WA*+CF, FS+PPM, GBPS+PPM	640000	Adam	512	35	0.0004

Neural A*



TransPath: