Learning-Based Branch-and-Bound for Template Matching

Asaf Solomiak & Karin Sifri
Guided by Dr. Simon Korman
Department of Computer Science, University of Haifa

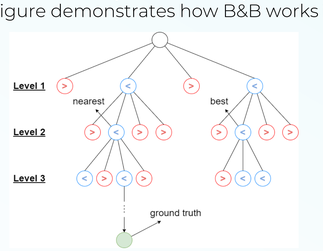
Introduction

FAsT-Match (Fast Affine Template Matching) is an algorithm designed by Simon Korman, Daniel Reichman, Gilad Tsur and Shai Avidan to search a fixed template inside an image, using the Branch and Bound technique.
The template which the algorithm is trying to find, can be found under an arbitrary affine transformation, therefore the number of transformations to consider is huge.

On each level of the algorithm, we check many transformations (also called configurations) which are spread along the search space. Each configuration is characterized by the photometric distance between the template and the place on the image where this configuration transforms the template into, on sampled points. At the end of each level, we are left with an array of such distances, and we need to decide around which configurations to expand. Therefore, we use a threshold (the higher bound) on those distances' values, computed as

$$threshold = minimum\ (called\ also\ best)\ distance + f(\delta)$$

The following figure demonstrates how B&B works in FAsT-Match:



The configurations in red were not close enough to expand in the next level ($distance > threshold$), while the configurations in blue, passed the threshold and therefore were chosen to be expanded in the next level of the algorithm ($distance < threshold$).

The $f(\delta)$ part of the bound is a linear combination of the parameter δ of the algorithm, which specifies the grid resolution of the search space, and is lowered by the same factor each level. This linear combination has been computed manually through trial and error and it uses only one parameter – δ (delta). Because of those reasons, it is not always optimal.

To tackle this issue, we have improved the decision process of the abovementioned higher bound by designing a Neural-Network model, resulting in time and accuracy improvement of the FAsT-Match algorithm.

In detail, we modified the threshold to be

$$threshold = minimum\ distance + f(set\ of\ features)$$

While the $f(set\ of\ features)$ part is now computed by the Neural-Network model, and it isn't only using a single parameter but a set of features (which includes delta).

The value which our model returns, ideally should be higher than the distance of the nearest configuration in the level, the one which on expansion will eventually get us to the real configuration of the template – the ground truth (green circle in the left figure).

Samples Collection

In order to build a deep learning model for the threshold, we have created a data-set by running the algorithm over 100 random templates on 100 random images on each level of the algorithm we have collected a total of 37 features as well as the ideal higher bound which we'll use as the target output of the model.

We collected 73.5 KB of 68854 samples, with a total computation time of 121.86 hours.

Deep-Learning Model

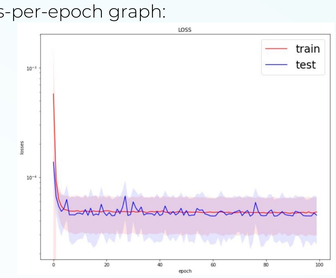
The next step was to implement a Deep-Learning model that will choose the optimal bound and improve the algorithm run-time and accuracy. We chose Multi-Layer Perceptron for the Regression model, which is a simple Neural-Network consisting of some fully-connected linear layers.

Additionally, we decided to use only 10 out of the 37 features which we collected on our samples, because after the collection process, we noticed that many features were highly correlated or not necessary for our model.

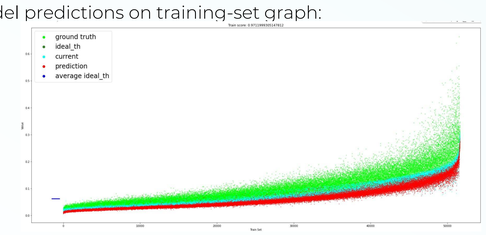
The process of choosing the model's parameters was conducted mainly by trial and error, and estimating the model performance by the following values:

- Loss – we used MSE (squared distance between the target and the model's prediction) as our model Loss-Function.
- Score – the ratio between our loss to the loss we would have got if we were predicting the average target value every time.
- Accuracy – percentage of times we succeeded on passing the ground-truth configuration to the next level in the algorithm ($prediction > ground\_truth$).

Example loss-per-epoch graph:

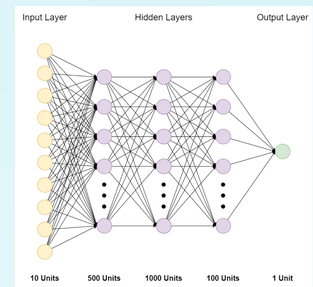


Model predictions on training-set graph:



Deep-Learning Model Cont.

After many trials, we came at our final model configuration:



With Tanh activation function and the following hyper-parameters:

$$number\_epoch = 100$$
$$size\_minibatch = 100$$
$$learning\_rate = 0.00001$$
$$weight\_decay = 0.000001, (for\ ADAM\ optimizer)$$

Testing & Results

The last step of the project was to check whether integrating this model in the FAsT-Match algorithm can improve its speed and accuracy results.

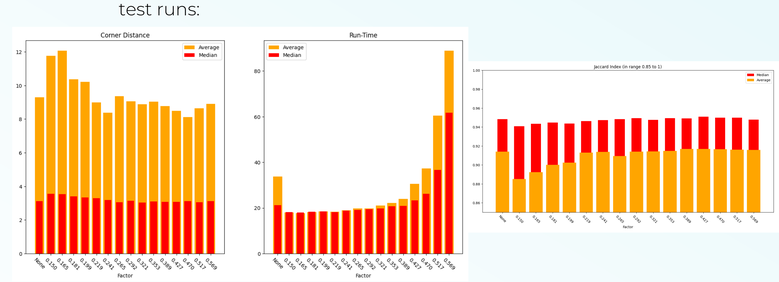
We ran on various target values to compare the results of each of them to the version of the algorithm without any such model.

The target value that was used is:

$$\min\_distance + factor * (ground\_truth - \min\_distance)$$

Every time, we checked on a different set of factor values and continued to search around factor values which gave us good results. We compared the speed results by the average time per run, and the accuracy results by the Jaccard Index (union over bound) as well as the average distance between the corners of the found template to the ground truth one.

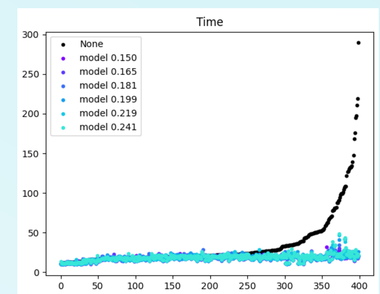
On the final round, we measured the results of 15 models with factor in the range 0.15-0.6 and got the following results on 400 test runs:



Testing & Results Cont.

We can clearly notice the trade-off between accuracy and run-time. For instance, models with factor in range 1.5 to 2.1 got corner distance values higher and Jaccard index values lower than the rest of the models, while models with higher factor values got good accuracy results but it took them more seconds to complete a run.

We can also notice by comparing to the first bar which represents the previous method (without model), that our models can barely improve the accuracy, but they can improve the speed of the algorithm, mainly the worst cases as indicated by the average value of the bar in the Time graph. This worst-case improvement is demonstrated in the following graph which shows the time results on 400 runs (on each model), sorted by the time values got on the runs without any model (in black):



We could use this trade-off to integrate the desired model according to the purpose of our usage of the algorithm, but if we would have needed to choose one model which works good generally, we would integrate the model with the factor 0.241 for its good speed results and slight accuracy improvement.

This improvement shows that the use of such model in the FAsT-Match algorithm increases its stability in terms of speed, while keeping the same quality of results.