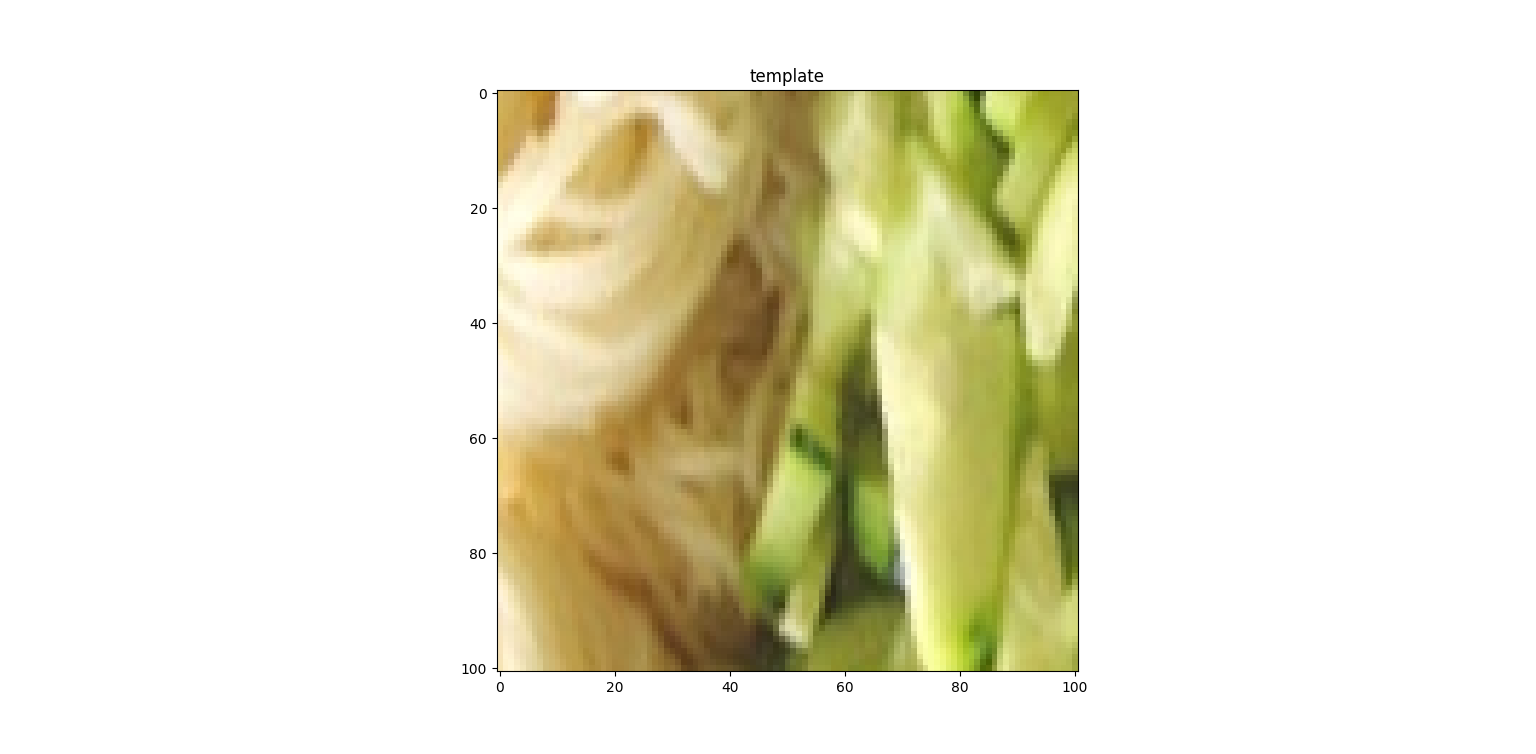
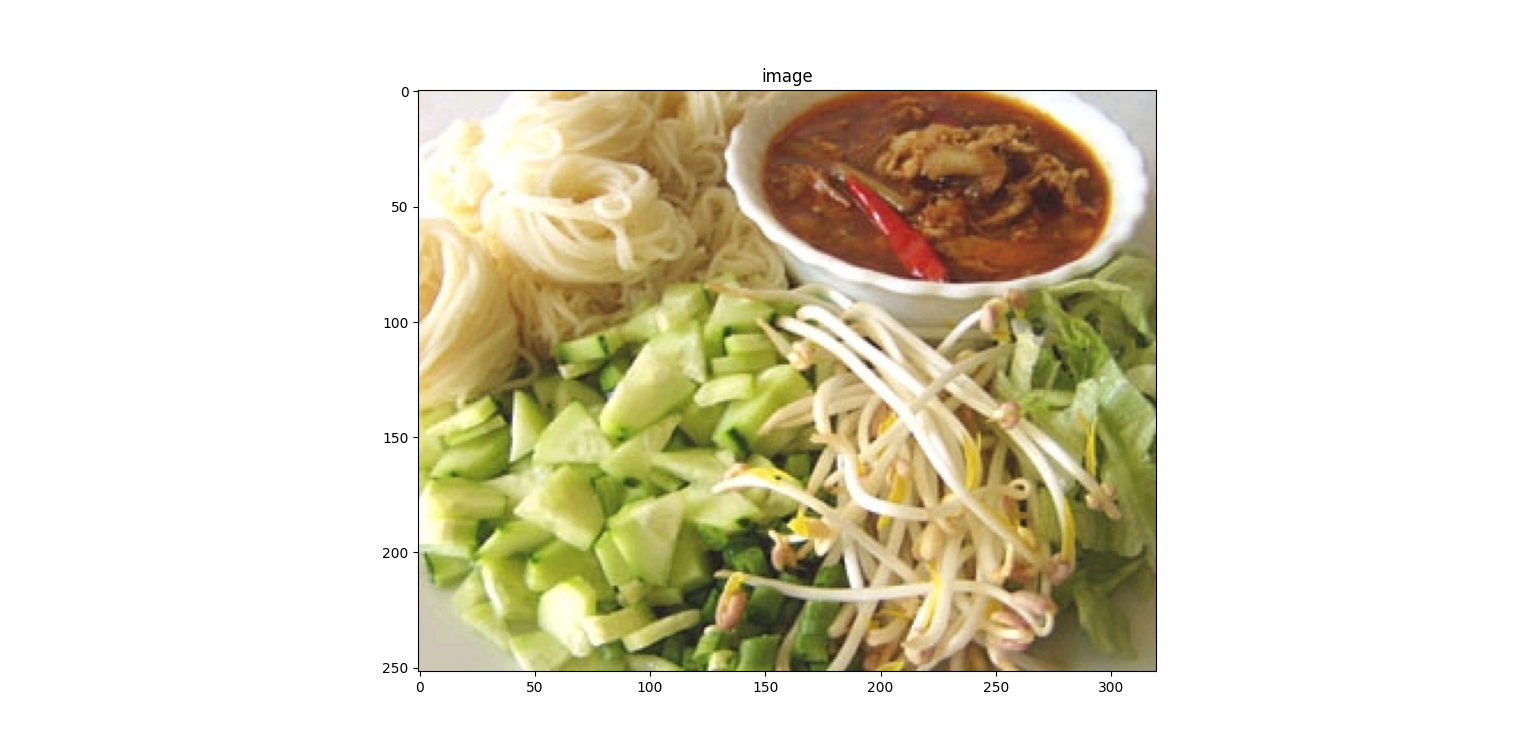
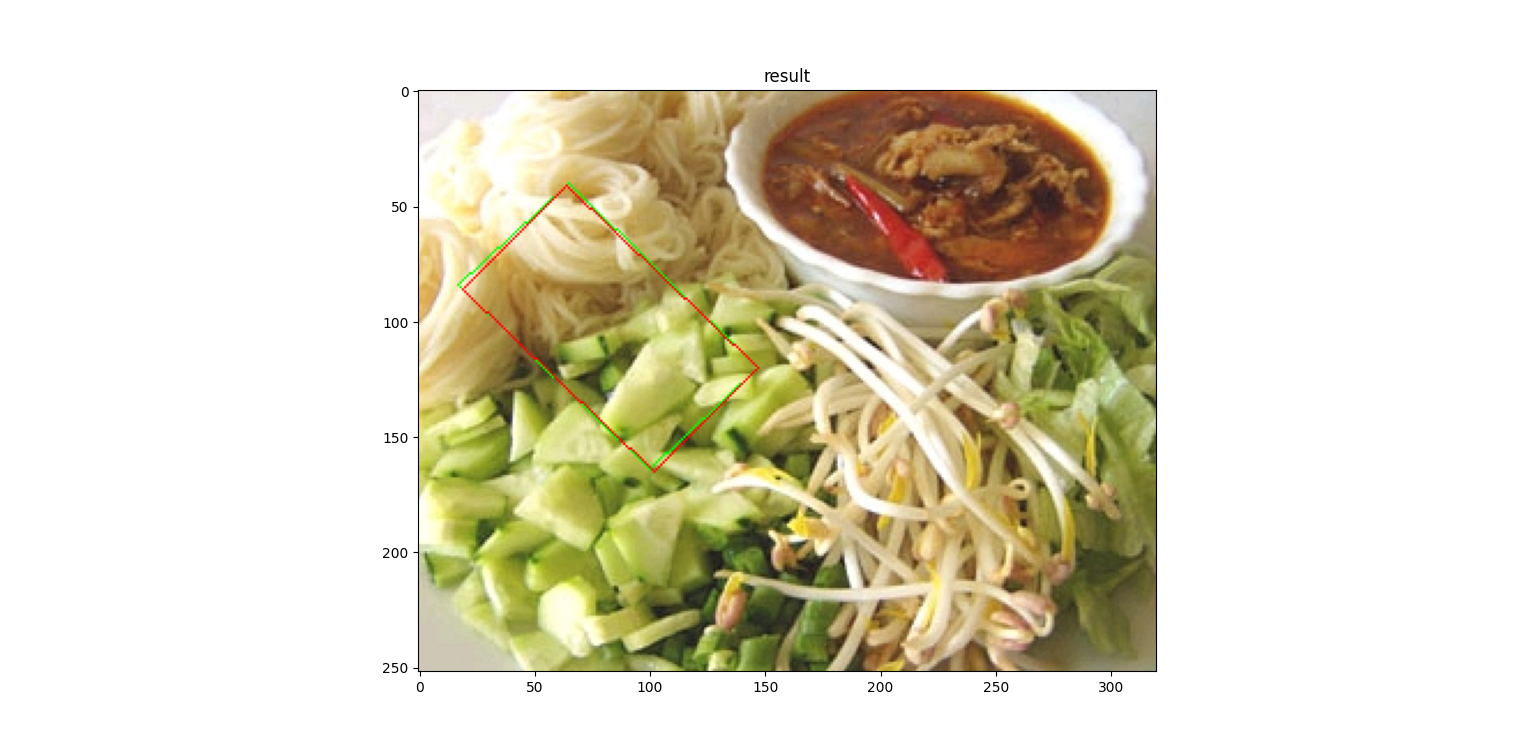
****Learning-based Branch-and-Bound for template matching

****

**University of Haifa,**

**Department of Computer Science**

**Etgar 14**

**August 2022**

Guided by: Dr. Simon Korman

Asaf Solomiak, 212585863, [asafucho@gmail.com](mailto:asafucho@gmail.com)

Karin Sifri, 213883903, [Karin.sifri@gmail.com](mailto:Karin.sifri@gmail.com)

Table of Contents

[Problem Statement 3](#_Toc111201902)

[הגדרת הבעיה 3](#_Toc111201903)

[הגדרת הפתרון לבעיה 3](#_Toc111201904)

[פירוט המימוש 4](#_Toc111201905)

[הרצת האלגוריתם 9](#_Toc111201906)

[פירוט מאפיינים 9](#_Toc111201907)

[איך ניתן להמשיך את הפרויקט 9](#_Toc111201908)

[מה למדנו במהלך הפרויקט 9](#_Toc111201909)

[צוואר הבקבוק 9](#_Toc111201910)

# Problem Statement

Branch-and-Bound (B&B) is a general technique for accelerating brute-force in large domains. It is used when a globally optimal is desirable, while more efficient optimization methods (like gradient descent) are irrelevant (e.g. due to the minimized function being nonconvex). A particular example of interest, very common in computer vision, is that of template matching (or image alignment), where one image (the template) is searched in another. This is useful, for example, in applications like stitching of panoramas, object localization and tracking.

FAsT-Match (Fast Affine Template Matching) is an algorithm designed by Simon Korman, Daniel Reichman, Gilad Tsur and Shai Avidan ([source](https://www.cs.haifa.ac.il/~skorman/FastMatch/index.html)) to search a fixed template inside an image, using the B&B technique.

The template which the algorithm is trying to find, can be found scaled up/down, rotated and affined, therefore the number of transformations to take into account is huge, and the lower bound on each sub-space of the domain is varied with respect to the current parameters and the loss function. This lower bound has been computed manually through trial and error using only one parameter, and because of that, it is not always optimal.

# Project Definition

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

# 

# Implementation

First, we implemented the algorithm in the Python programming language, by translating the original code from its MATLAB version.

Afterwards, we created a data-base which we could use later for the Neural-Network creation process. We collected 100 random images from the Internet and on each image, we ran the algorithm on 100 random templates inside the image. Each level of the algorithm provided us with the following data which we collected:

* The image dimensions
* The template dimensions
* The ratio between the average pixel value in the template to the average pixel value in the image
* The template smoothness (average template derivatives)
* The template pixels’ standard deviation
* Delta – the one value which the lower bound in the original implementation of the FAsT-Match algorithm is computed by
* Data about the distance array of the current level’s subspace: minimum distance, average distance, distances standard deviation, percentiles (5, 10, 15, …, 95), distances range, distances amount (total number of configurations in the current sub-space).
* Affine matrix of the configuration with the minimum distance.

Total number of features collected: 37, as well as the ideal lower bound which we’ll use as the target output of the model.

In addition, on a different file we stored for each level the minimum and maximum distance values and the histogram values of the distance array divided into 100 bars. This data was collected to estimate the model’s accuracy – the percentage of the sub-space which is higher than the lower bound.

In order to ease the process of samples collection, we divided the images into 20 folders and ran the algorithm on them concurrently. We collected 73.5 kb of 68854 samples, in total computation time of 121.86 hours.

The next step was to implement a machine learning/deep learning model which on integration in the algorithm, will help us choose the optimal bound and improve the algorithm time and accuracy.

At first, we were required to do some research in those areas and learn how to implement such thing and what is the best model for our problem. Lastly, we chose to use Multi-Layer Perceptron for Regression model, which is a simple Neural-Network consisting of some fully-connected linear layers.

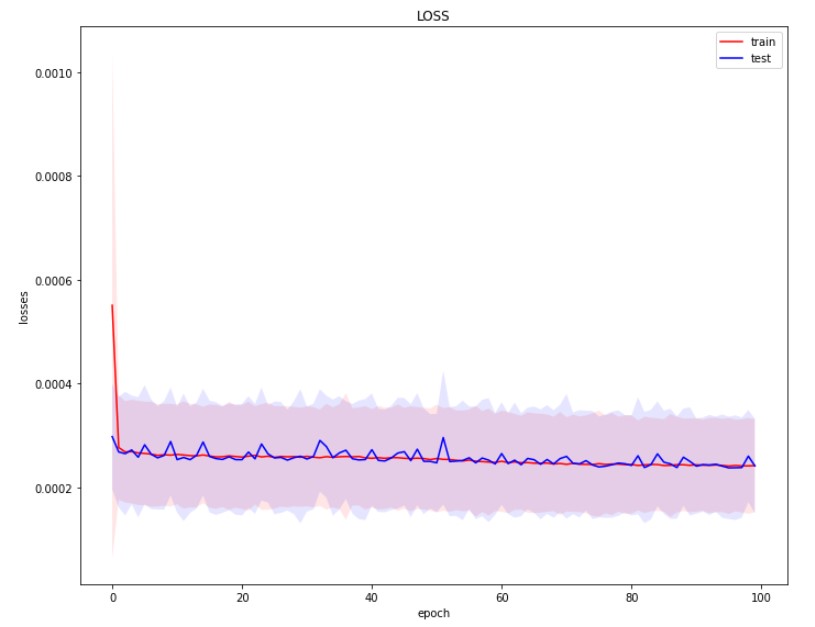
Additionally, we have decided to use only 10 out of the 37 features which we collected on our samples, because after the collection process, we noticed that many features were highly correlated or not necessary for our model. The 10 features used:

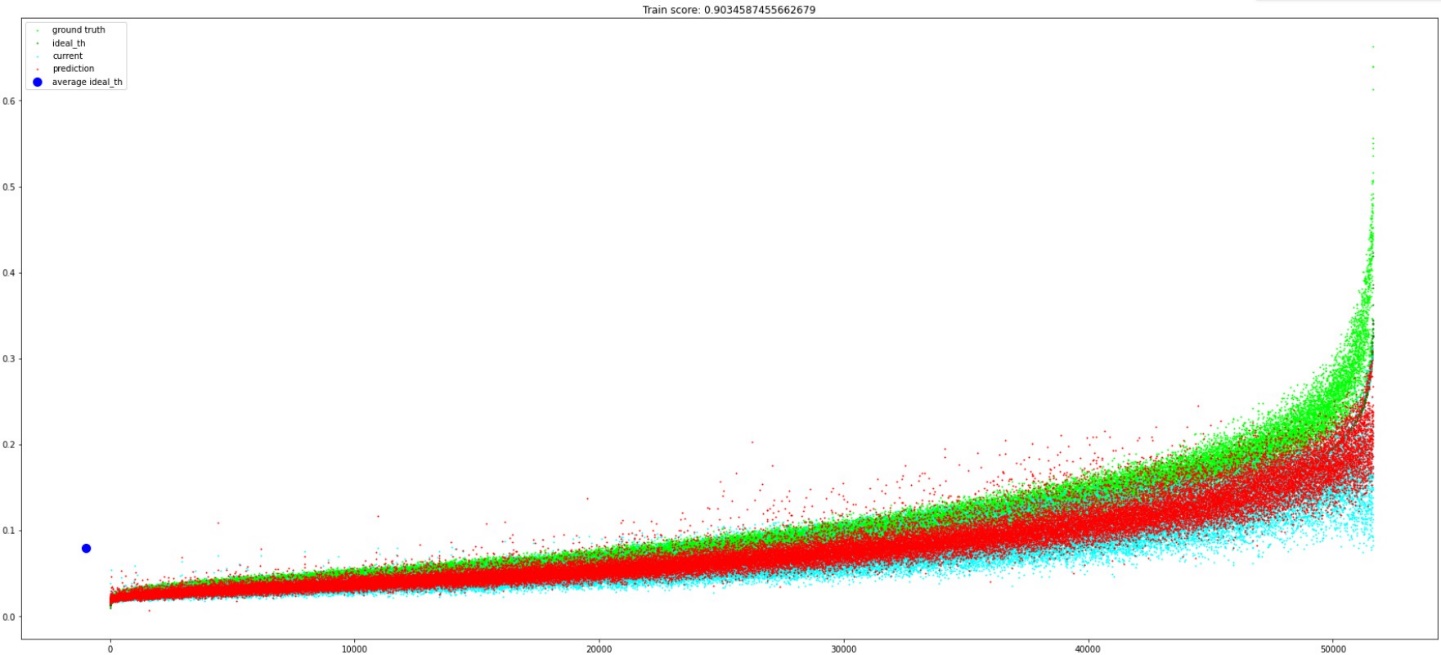
* The template dimensions
* The ratio between the average pixel value in the template to the average pixel value in the image
* The template smoothness (average template derivatives)
* The template pixels’ standard deviation
* Delta – the one value which the lower bound in the original implementation of the FAsT-Match algorithm is computed by
* Data about the distance array of the current level’s subspace: minimum distance, average distance, distances standard deviation, distances range, distances amount (total number of configurations in the current sub-space).

The process of choosing the right 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 ().

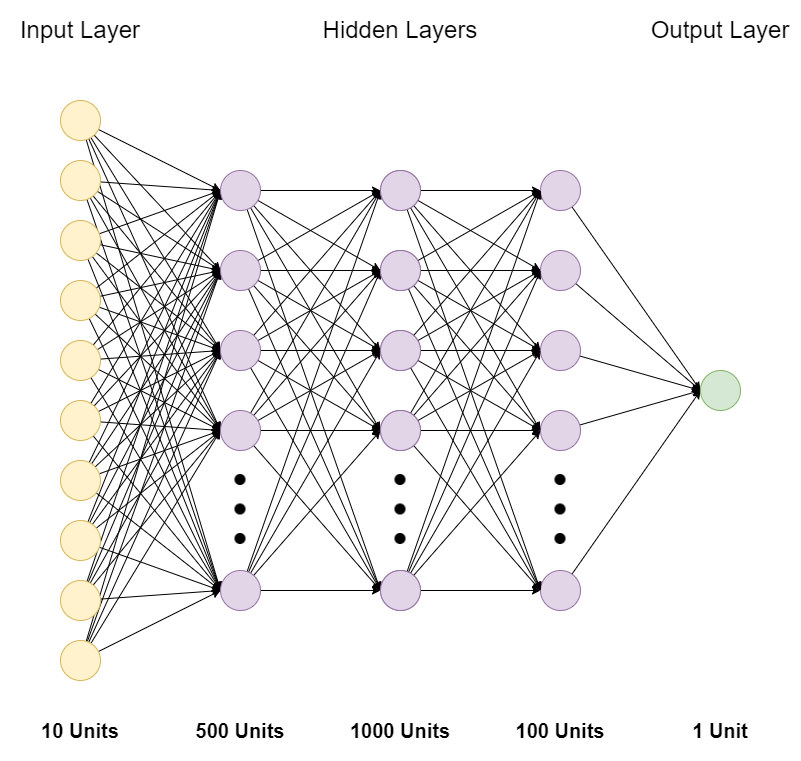
Example:

Loss per epoch graph:

Model predictions on training-set graph:

* Green – target
* Red – prediction
* Lime – ground-truth
* Cyan – the current lower bound (without a model)

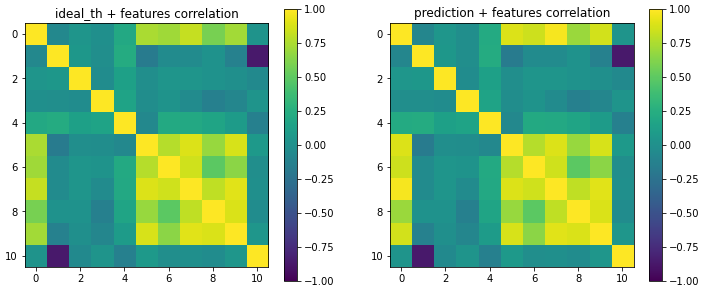
Metrics on the test-set:

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

And the following hyper-parameters:

Feature correlation image, plus the target value (in the 0th row and column)

*תמונת ה-feature correlation פלוס ערך ה-target (בעמודה והשורה ה-0):*

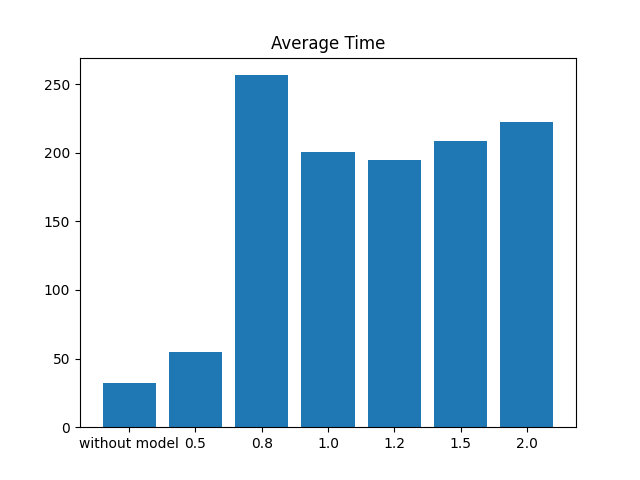
**

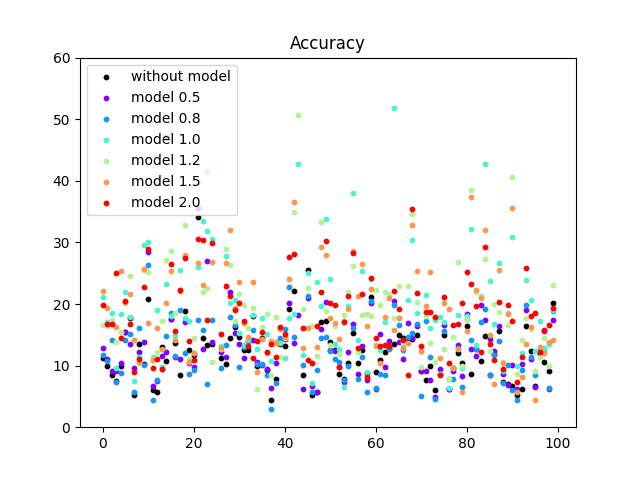
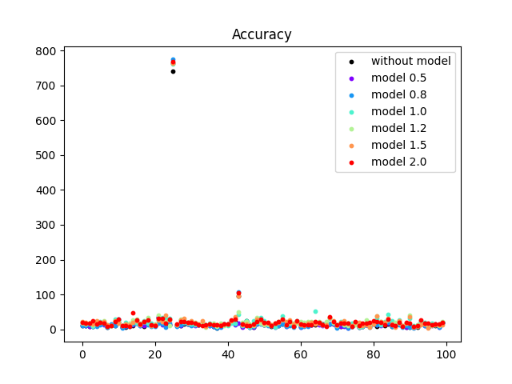
*בעזרת המודל הנ"ל אימנו 6 מודלים שאותם שילבנו ובדקנו באלגוריתם עצמו.*

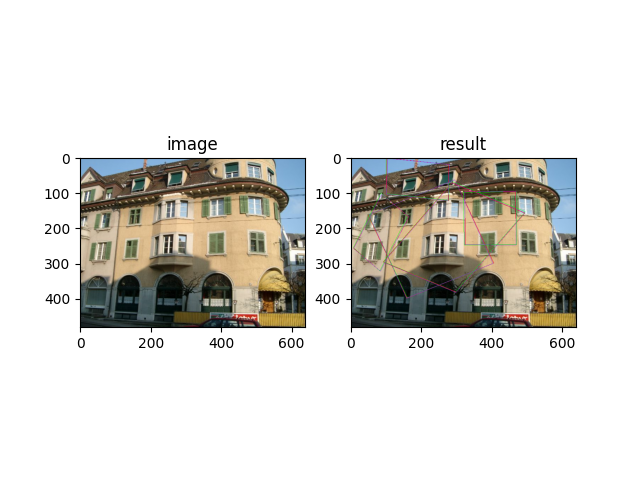
*ההבדל במודלים הוא בערך ה-target שלהם:*

*כאשר ערך ה-factor משתנה בין המודלים מבין הערכים:*

*השלב האחרון בפרויקט היה לבדוק איך האלגוריתם FAsT-Match מתפקד בשילוב המודלים, והאם הצלחנו לשפר את הדיוק שלו.*

*הרצנו 100 פעמים על כל מודל (10 תמונות ו-10 תבניות לכל תמונה) ועוד פעם אחת נוספת ללא מודל, ומדדנו את הזמן הממוצע עבור ריצה יחידה לכל מודל וכן את הדיוק של המודל בעזרת המרחק של הפינות שנמצאו מהפינות האמיתיות של התבנית. קיבלנו את הנתונים הבאים:*



ריצות לדוגמה עם מודל:

# הרצת האלגוריתם

ניתן להריץ את הקוד מתוך PyCharm, על ידי הרצת ה-main. על מנת להריץ את הקוד יש להוריד את ה-packages הבאים:

* Numpy
* Matplotlib
* Cv2
* Sklearn
* Pandas
* Torch

# פירוט מאפיינים

בהרצת האלגוריתם ניתן לבחור בכמה פרמטרים שונים המגדירים את מרחב החיפוש של האלגוריתם ואת כמות נקודות הדגימה שהאלגוריתם דוגם בתבנית על מנת להעריך את דיוק הקונפיגורציות. בנוסף, ניתן להגדיר שהאלגוריתם ינרמל את רמת הבהירות בתמונה ובתבנית (במידה והתבנית לא נלקחה באופן מלאכותי מהתמונה).

כמו כן, ניתן לבחור מבין המודלים שאימנו את המודל בו האלגוריתם ישתמש, או לבחור שהוא לא ישתמש באחד המודלים.

לבסוף, בהפעלת הריצה עצמה יש לבחור תמונה עליה האלגוריתם ירוץ, ותבנית העשויה להיות מוגדרת מראש או שייבחר תבנית אקראית.

# איך ניתן להמשיך את הפרויקט

ישנם כמה כיוונים שאיתם חשבנו שיש עם מה להתקדם.

ניתן לחשוב על כיוון אחר מבחינת המודל, או לנסות להשיג תוצאות טובות יותר על ידי שינוי ה-target ו/או הפרמטרים. כמו כן, אפשר להשתמש במודל למידה אחר להכרעת פרמטרים נוספים באלגוריתם.

# מה למדנו במהלך הפרויקט

למדנו לקרוא ולהבין את שפת התכנות MATLAB אשר ממנה תרגמנו את האלגוריתם. כמו כן, מכיוון שהפרויקט כולו נכתב בשפת התכנות Python, הצלחנו לשפר את היכולות שלנו בו ולהבין איך לייעל תהליכים בשפה זו.

בנוסף, למדנו רבות על תחום ה-Deep Learning וכיצד לבחור ולאמן רשת נוירונים.

# צוואר הבקבוק

לפי דעתנו, הקושי הגדול ביותר בפרויקט היה לממש את האלגוריתם ב-Python. בהתחלה אף ניסינו לתרגם את האלגוריתם מ-C++ אך לא הצלחנו, ואז עברנו לנסות לתרגם מ-MATLAB ואפילו כשבסוף הצלחנו לממש, נתקלנו בבעיה שהמימוש שלנו לקח זמן רב באופן משמעותי יותר מהמימוש ב-MATLAB.

בשביל לתקן זאת היינו צריכים לחפש פתרונות לאיך לייעל את הריצה בכמה דרכים.