Real-time Deep-Learning Image Matching on Edge

Mykhailo Buleshnyi
Faculty of Applied Sciences
Ukrainian Catholic University
L'viv, Ukraine

Maksym Buleshnyi
Faculty of Applied Sciences
Ukrainian Catholic University
L'viv, Ukraine

Artur Pelcharskyi
Faculty of Applied Sciences
Ukrainian Catholic University
L'viv, Ukraine

Davyd Ilnytskyi
Faculty of Applied Sciences
Ukrainian Catholic University
L'viv, Ukraine

Roman Milishchuk
King's College London,
United Kingdom

Vasyl Borsuk
Faculty of Applied Sciences
Ukrainian Catholic University
L'viv, Ukraine

Mykola Morhunenko
Faculty of Applied Sciences
Ukrainian Catholic University
Ukraine

Abstract—**In recent years, machine learning has experienced an immense surge in popularity. As the accuracy of models has significantly improved, there has been a corresponding increase in the resources required to execute them. In certain scenarios, it becomes crucial to employ complex deep-learning models on embedded devices in real-time. The objective of this work is to enhance the ALIKE model's speed while minimizing any potential loss in accuracy, targeting the embedded platforms, such as a BeagleBone AI-64.**

I. Introduction

Image matching is one of the main tasks of 3D Computer Vision and is necessary for solving tasks such as Visual localization, Pose Estimation, and 3D Reconstruction. It also allows significantly improve the quality of video from the camera through multi-frame super-resolution and video stabilization. It is widely used in autonomous robotics platforms like cars, drones, etc. The goal of this work is to efficiently run a deep learning keypoint extractor named "Accurate and Lightweight Keypoint Detection and Descriptor Extraction" (ALIKE) on a single-board computer BeagleBone AI-64 to make it possible to use it for real-time tasks.

II. Experiental setup

For our project, we use BeagleBone AI-64, a high-performance single-board computer with a specialized processor for particular machine learning and neural networks.

Important tech specs:

- Processor: Texas Instruments (TI) TDA4VM
- GPU: PowerVR$^{®}$ Series8XE GE8430
- SDRAM: LPDDR4 3.2 GHz Q3222PM1WDGTK-U
- Built-in flash memory: eMMC Kingston EMMC16G-TB29-PZ90 (16GB)

III. EdgeAI TIDL and optimization

TIDL is a comprehensive software product for the acceleration of Deep Neural Networks (DNNs) on TI's embedded devices.

It optimizes overall model performance by converting NN operations to operations that are more efficient to execute on the particular TI's embedded devices.

Another part of optimization is quantization. Quantization is a common technique used to reduce the model size. This is achieved by converting model parameters (weights) from floating-point precision (32-bit or 64-bit) to lower precision integers (e.g., 8-bit or even binary) where possible.

Fig. 1 illustrates the DNN development and deployment workflow on TI devices.

IV. Problems with model optimization

Initially, our decision was to employ ALIKED [Zha+23], the latest iteration of this algorithm, which has deformable convolutions as a vital component of the model. However, we encountered challenges during the process of model conversion and quantization. Unfortunately, the Edge AI TIDL framework does not offer support for deformable convolutions, which leaves a big part of our model without optimization.

However, ALIKE [Zha+22], a little older model with ordinary convolutions, is much lighter and not much worse in terms of accuracy. It also includes some parts that can't be optimized fully with the TIDL - Differentiable Keypoint Detection (DKD) block, but overall, there is much more space for further optimization. So, we took this model for our work.

V. ALIKE

As we took ALIKE as our model, it is important to know some details about the model structure. ALIKE applies a differentiable keypoint detection module to detect accurate sub-pixel keypoints. The model is designed to run at 95 frames per second for 640 x 480 images on a NVIDIA Titan X (Pascal) GPU.

A. Overall model structure

The model is designed to be as lightweight as possible. Logically, it can be divided into 4 steps:

- **(a) The image feature encoder** encodes image using basic CNN block. It contains four blocks. The first block is a two-layer 3×3 convolution with "ReLU" activation and the last three blocks contain a max-pooling layer and a 3×3 basic ResNet block.

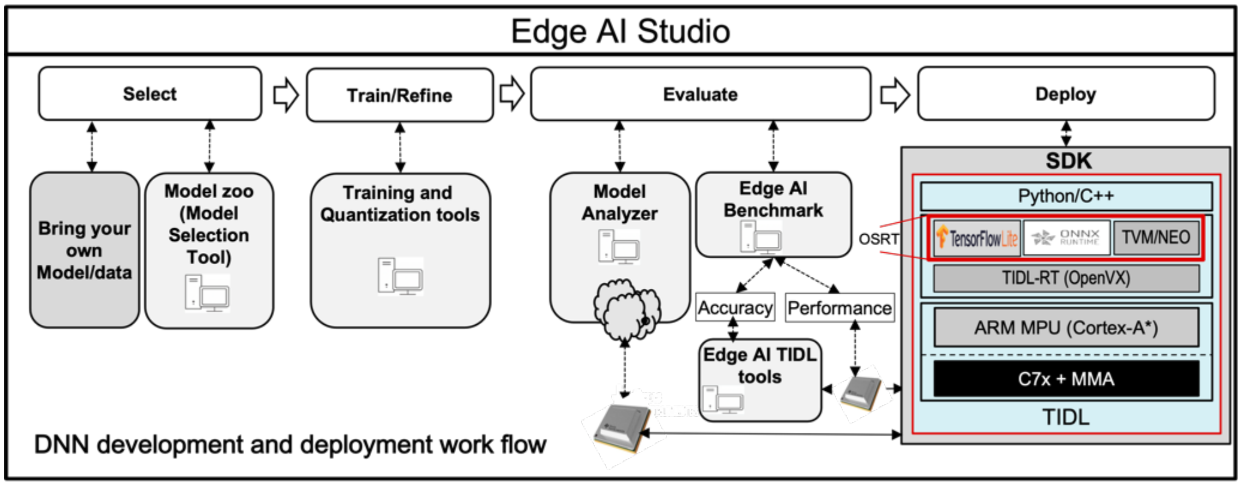

Fig. 1. EdgeAI TiDL pipeline [Ins24].

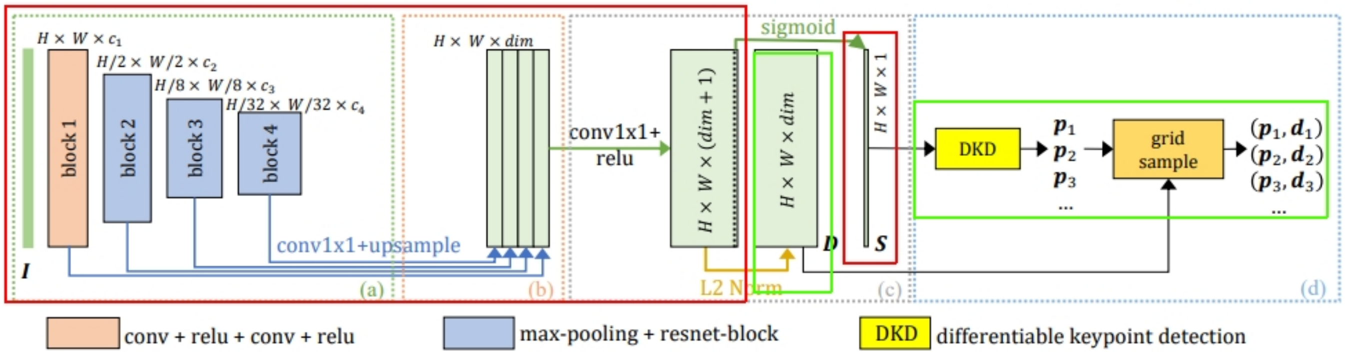

Fig. 2. Model structure, red denotes first stage, that can be optimized fully with TIDL, green – can not be optimized fully that way. [Zha+22]

- **(b) The feature aggregation module** aggregates multi-level features from the encoder.
- **(c) The feature extraction head** outputs an feature map.
- **(d) The differentiable keypoint detection and descriptor sampling** is the last stage, where keypoints are extracted and descriptors are sampled.

B. *Differentiable keypoint detection*

To detect key points in score map S, a widely used method is the Non-Maximum Suppression (NMS). It finds the pixels with the maximum score within local windows. In ALIKE, this was improved by using differentiable parts to utilize Deep Learning. For this goal, the DKD block uses the softmax function.

First, the block is gaining maximum values via NMS for each NxN window. After it subtracts the maximum value from the pixels in the window, normalizing the window. Softmax maps each value to the probability that this value is a keypoint. The last step is multiplying the probability for each pixel by its real values. At the end, the output subpixel key point is given as:

$$\boldsymbol{p} = [u, v]^T_{\text{soft}} = [u, v]^T_{NMS} + [\hat{i}, \hat{j}]^T_{\text{soft}} .$$

C. *Learning discriminative descriptor*

Descriptors describe how keypoints are transformed on different images. Thus, descriptors of the same keypoints (in different images) should be similar, whereas descriptors of different keypoints should be distinct.

- **Reprojection loss**. To address this problem, this block is warping keypoints from image A to image B to get displacement.

$$\boldsymbol{p}_{AB} = \text{warp}_{AB}\left(\boldsymbol{p}_A\right)$$

Getting the distance from one point to another via subtracting the keypoint map of image B from A.

$$\text{dist}_{AB} = \|\boldsymbol{p}_{AB} - \boldsymbol{p}_B\|_p$$

To normalize this distance, the block is taking the average between the distance from A to B and from B to A

$$\mathcal{L}_{rp} = \frac{1}{2}\left(\text{dist}_{AB} + \text{dist}_{BA}\right)$$

- **Dispersity peak loss.** The minimization of reprojection loss optimizes the scores in the local window through the soft term $[\hat{i},\hat{j}]^T_{\text{soft}}$. However, the gradient step to improve $[\hat{i},\hat{j}]^T_{\text{soft}}$ might affect the $[u,v]^T_{NMS}$. To align their optimization directions, this block regularizes the scores in the local window to be "peaky": that is, it should have a high score at the keypoint and low scores around it in the local window. In such a case, even if the local window centered on $[u,v]^T_{NMS}$ is slightly shifted, it still contains the keypoint, and $[\hat{i},\hat{j}]^T_{\text{soft}}$ will be regulated to a new soft offset w.r.t. the new local window center so that the keypoint position remains stable. To force the score patch to be "peaky" exactly at the keypoint, we propose the score dispersity peak loss. It takes into account the spatial distribution of scores, resulting in higher scores at the keypoint and lower scores further away.
Considering a $N \times N$ score patch, the distance of each pixel $[i,j]^T$ in the patch to the soft detected keypoint $[\hat{i},\hat{j}]^T_{\text{soft}}$ is

$$d(i,j) = \left\{ \left\| [i,j] - [\hat{i},\hat{j}]_{\text{soft}} \right\|_p \mid 0 \leq i,j < N \right\}.$$

The dispersity peak loss of this patch is then defined as

$$\mathcal{L}_{pk} = \frac{1}{N^2} \sum_{0 \leq i,j < N} d(i,j) s'(i,j),$$

where s' is the softmax score.

VI. Metrics

A. Model evaluation metrics

Mean Matching Accuracy (MMA) and **The Mean Homography Accuracy (MHA)** are proposed in [Zha+22].

B. Dataset

For this task, we used a sample of the Hpatches dataset [Bal+17] – the benchmark of handcrafted and learned local descriptors.

VII. ALIKE optimization on TI's device

A. Different versions of ALIKE models

TABLE I
EXECUTION TIME MEASURED ONLY FOR THE FIRST SEGMENT OF THE MODEL, AS OUTLINED IN THE 2, ON THE DEVICE.

Model	MMA	MHA	Ex. time	Num. of keyp.
Alike-t	12.73%	40%	1.09	1017
Alike-n	13.70%	40%	2.48	726
Alike-l	15.38%	50%	4.5	403

The ALIKE model possesses different configurations: alike-t, alike-n, and alike-l. They are ordered in ascending order with respect to their accuracy and execution time (Table I). As our prior goal was to optimize the algorithm to make it possible to use it for the real-time tasks, we decided to use the configuration alike-t. Also important to note is that even though the number of keypoints decreases, the time taken for DKD block also decreases. The model was trained on 640 × 480 images, so for consistency, we fixed that shape for our model.

B. Where to start

TABLE II
TIME FOR INFERENCE OF ALIKE (ALIKE-T) ON THE DEVICE BEFORE ANY OPTIMIZATIONS.

Total time 1.72	Step 1 1.09	Step 2: 0.63
		Normalization: 0.29
		DKD: 0.26 (NMS: 0.25)
		Other: 0.08

C. Logical parts

Generally, we can divide our model into 2 high-level steps. The first one combines the CNN encoder, Feature Aggregation module, and part of the feature extraction head, particularly split on descriptor and score map + sigmoid operation. The second step combines L2 Normalization from the (c) block and the whole (d) block, (see Fig. 2). The reason for this split is described in more detail in the following sections.

D. First stage optimization

The main goal of this research is to use the Edge AI TIDL library to make the model faster on embedded devices. However, not all operations can be converted with the library, or even if it is possible, there is a chance that such conversion would not be efficient. So, before optimization, it is important to prepare a model for conversion. That is the main reason why we chose this split is that the first part has operations that are supported by the TIDL. However, the normalization operation is not supported, and that is why we put it in the second stage.

Preparation of the model:

1) Move the Transpose and Usqueeze operations outside the converted part, as TIDL does not support Transpose and Usqueeze operations.
2) Convert Gather and Slice operations into Split operations, as Gather, Slice is not supported.
3) Convert all Maxpool operations to Maxpool 2x2 operations for better optimization.
4) Convert all Unsample as a combination of Unsample 2x2 and Unsample 4x4 for better optimization.

Now, our model satisfies the condition with supported layers by TIDL. But to make it work correctly, the model has to be converted to ONNX.

E. ONNX model conversion

TIDL interface for conversion expects the model to be either in tflite or ONNX format. For our case, we need to convert our model to ONNX format. ONNX defines a common format for representing deep learning models. This format consists of a computational graph, where nodes represent operations, and edges represent the flow of data.

After model conversion to ONNX, it is necessary to infer shapes. The goal is to determine the shape of each tensor at every point in the graph. The TIDL library will later use this information.

F. TIDL conversion and test results

After all the preparation steps, we can successfully convert the model with the TIDL. We compared 8-bit and 16-bit quantization. Step 1, on average, takes 0.16 seconds for 8-bit conversion and 0.19 for 16-bit conversion. As there was no significant reduction in accuracy, we decided to use 8-bit quantization. Compared to 1.09 seconds, it is **6.8 times** faster!

VIII. Second stage optimization

A. DKD (Differentiable Keypoint Detection block)

We have significantly accelerated the first part of our model. The second part of the model can also be optimized using a different approach. A big part of the DKD block is Non-maximum suppression (NMS). Almost all the time of which is taken by Maxpool block. One of the possible optimizations is to convert Maxpool operation to ONNX format. After doing all the same operations with the converted Maxpool operation, NMS time came from 0.25 to 0.05 seconds.

Further, we noticed that the algorithm used 2 iterations of Maxpool to find more keypoints. We tested it with only one iteration and found that the number of keypoints, on average, increased by 4 keypoints. Testing this setup on accuracy, we have not noticed any reductions compared to the 2-iteration version. Time taken for NMS reduced from 0.05 seconds to 0.03.

B. Additional optimization

In addition, we cleaned the model from unnecessary copying and declarations and reduced the overall model output time by another 0.06 seconds.

IX. Results

TABLE III
Time for inference of ALIKE (alike-t) on the Device after the optimizations.

Total time 0.51	Step 1 0.16	Step 2: 0.35
		Normalization: 0.29
		DKD: 0.04 (NMS: 0.03)
		Other: 0.02

Also, the normalization block takes a significant amount of time for part 2 of the block. But when working with real-time video, there are special TIDL plugins that will make normalization. This library goes out of the scope of this project, but we left it for general statistics.

After all manipulations, the performance of part 1 was increased by 6.8 times, and the performance of part 2 was increased by 1.67 times. The general time of execution changed from 1.72s to 0.51s (see Table II, III). The frame rate increased from 0.58 to 1.97 frames per second, including normalization. As was mentioned, the normalization part for real-time video can also be optimized, so the frame rate will be even higher.

X. Conclusion

In this project, we got an overall boost of **3.3 times**! Certainly, there are some other optimizations that can be done, but they will not change the time of execution significantly. In this project, the main optimization was using the Edge AI TIDL library. Also, some memory and code manipulation was done. Overall, a boost in performance enables us to use this approach for real-time problems. The code of the project is available at https://github.com/UCU-Pokemons/Image-Matching-on-Edge.

References

[Bal+17] Vassileios Balntas et al. “HPatches: A benchmark and evaluation of handcrafted and learned local descriptors”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5173–5182.

[Zha+22] Xiaoming Zhao et al. “Alike: Accurate and lightweight keypoint detection and descriptor extraction”. In: *IEEE Transactions on Multimedia* (2022).

[Zha+23] Xiaoming Zhao et al. “Aliked: A lighter keypoint and descriptor extraction network via deformable transformation”. In: *IEEE Transactions on Instrumentation and Measurement* (2023).

[Ins24] Texas Instruments. “TiDL EdgeAI repository ReadME: https://github.com/TexasInstruments/edgeai-tidl-tools”. In: (2024).