

NVIDIA TensorRT

How to deploy high-performance deep learning inference on NVIDIA GPUs

Deep Learning

Simone Murari
VR502597
University of Verona

Contents

Motivation	3
Objectives	3
Methodology	3
The libraries	3
The model	3
The dataset	4
The approach	4
Results	4
Conclusion	6
Bibliography	6

Motivation

The motivation behind this project was to see if it was possible to have an high accuracy anomaly detection algorithm deployed on NVIDIA GPUs to achieve the highest possible inference speeds with an acceptable memory usage. This scenario could lead to the implementation of real-time anomaly detection systems that could be employed in industrial settings, for example in an assembly line to separate defective items from defect-free ones.

Objectives

This Deep Learning project aims to show how to deploy an anomaly detection model on an NVIDIA GPU using **TensorRT**¹, an ecosystem of APIs for high-performance deep learning inference on NVIDIA GPUs, and compare its inference time and memory usage among other frameworks, in particular: **PyTorch**, **ONNX** and **OpenVINO**.

Methodology

The libraries

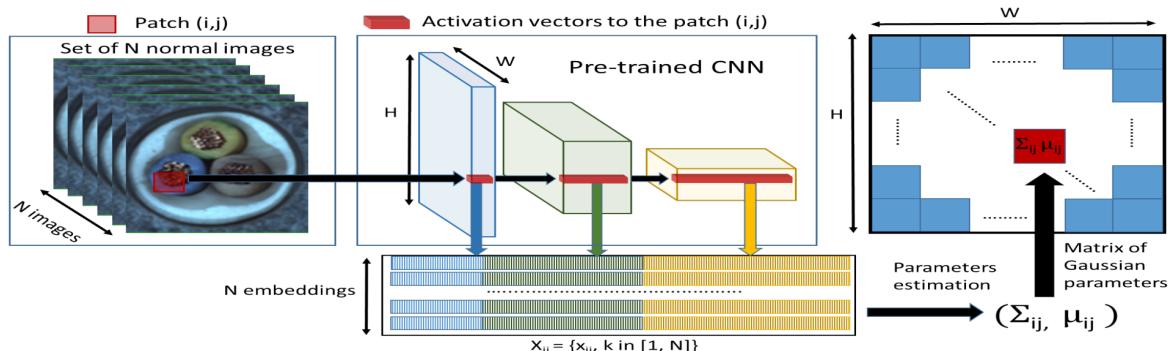
The main library employed in this project is **Anomalib** [1], an open source deep learning library for anomaly detection that provides state-of-the-art models and that can be used off-the-shelf. This library was used in all the different parts of the project: model initialization, training, testing and evaluation. Another fundamental library is **anomalib-tensorrt-python** that helps in adding TensorRT inference to the Anomalib library. This library worked only with TensorRT 8.5 so some changes were made to make it work with TensorRT 10.1.0. Finally, other very important libraries used worth mentioning, some of which are part of Anomalib, are **PyTorch Lightning**, **CUDA**, **NumPy**, **matplotlib**, **Pillow**...

The model

The model used here is **PaDiM**, a Patch Distribution Modeling Framework for Anomaly Detection and Localization that *makes use of a pretrained convolutional neural network (CNN) for patch embedding, and of multivariate Gaussian distributions to get a probabilistic representation of the normal class. It also exploits correlations between the different semantic levels of CNN to better localize anomalies.* [2]

Here you can see the architecture of the model:

Fig. 2. For each image patch corresponding to position (i, j) in the largest CNN feature map, PaDiM learns the Gaussian parameters (μ_{ij}, Σ_{ij}) from the set of N training embedding vectors $X_{ij} = \{x_{ij}^k, k \in [1, N]\}$, computed from N different training images and three different pretrained CNN layers.



The particular pre-trained network that was used in this project is a **Wide ResNet-50-2** [3].

¹<https://docs.nvidia.com/deeplearning/tensorrt/index.html>

The dataset

The dataset is the **MVTec²** dataset, it is useful for benchmarking anomaly detection methods with a focus on industrial inspection. It contains over 5000 images divided into 15 different categories and each category comprises defect-free training images and test images with various kind of defects, these are some samples from the dataset:



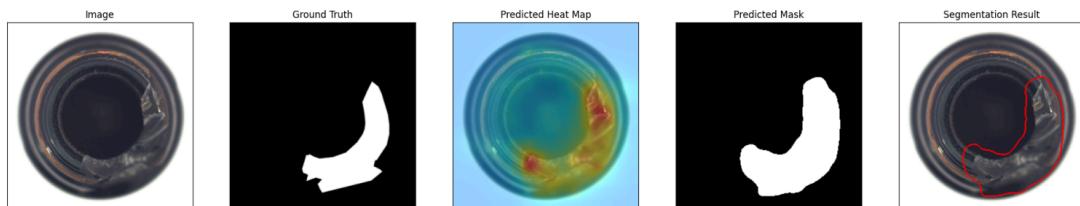
Figure 1: samples from MVTec dataset

The approach

The first step of our project was to load the PaDiM model into PyTorch with FP32 precision, then the model was trained on the training set and evaluated on the testing set using an NVIDIA RTX 4070 Laptop with 8GB of VRAM. Then the weights were halved to get the FP16 model that was also evaluated using the same GPU. After that, the FP32 model was exported into ONNX, a crucial and mandatory step to be able to get to the TensorRT model. The ONNX model was evaluated on the CPU, a Ryzen 7840HS, with 32GB of RAM, because it would not fit in the GPU. To have a comparison between different CPU frameworks the original FP32 model was also exported in OpenVINO, converted in FP16 and evaluated. The final step was converting the ONNX model into TensorRT and evaluating it on the GPU, and then comparing the results among all the different frameworks.

Results

The results were evaluated in a qualitative way and in a quantitative way. At first, for each prediction of each framework, a 5 image panel showing: the input image, the ground truth mask, the predicted heat map, the predicted mask and the segmentation result was plotted, here you can see an example for the FP32 PyTorch model:



Then the average pixel-level and image-level F1 and AUROC scores were calculated over all the predictions made with a single model, for example the FP32 PyTorch model had scores of:

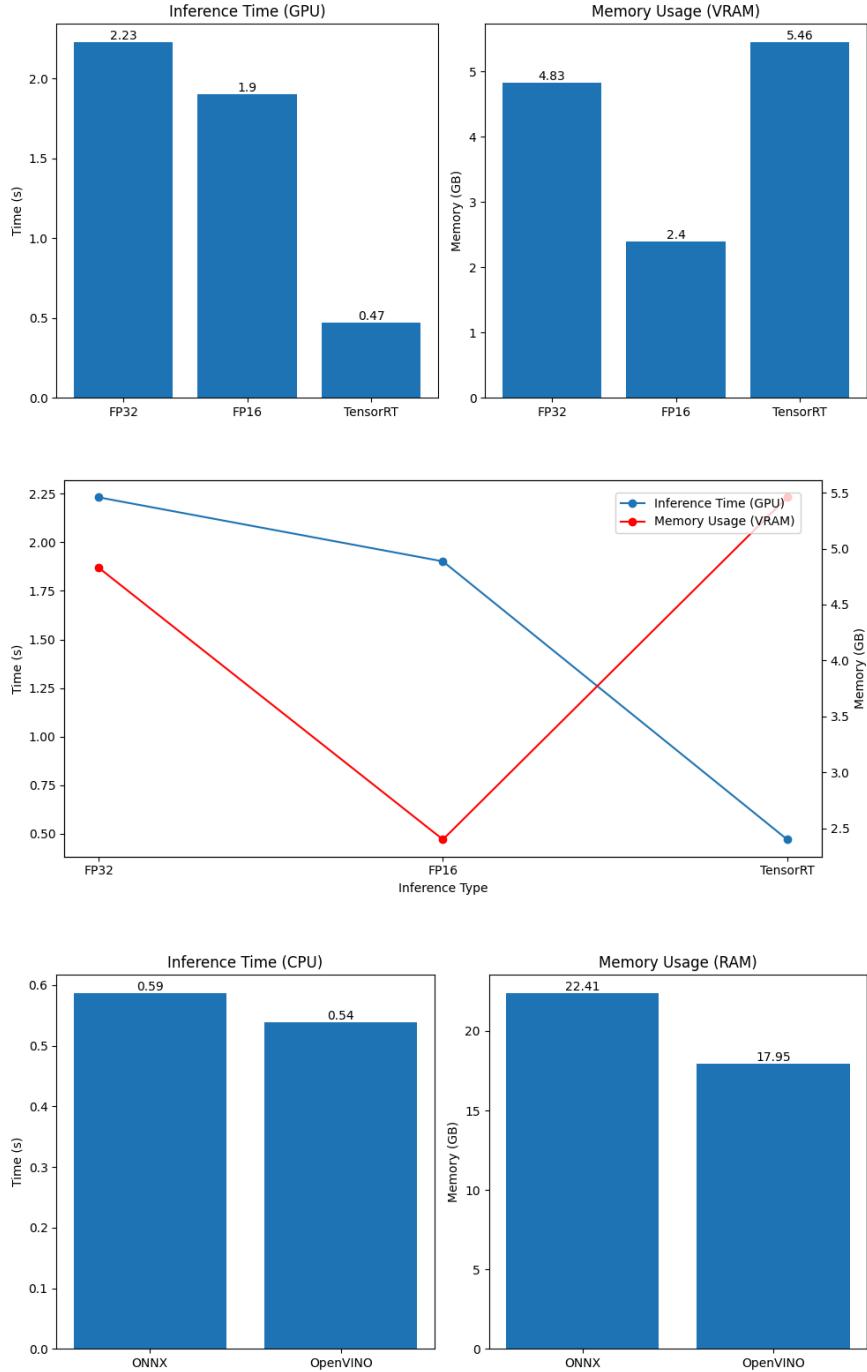
- Average pixel-level F1 score: 0.748151957988739
- Average image-level F1 score: 0.9572950601577759
- Average pixel-level AUROC: 0.9305204749107361
- Average image-level AUROC: 0.9679543375968933

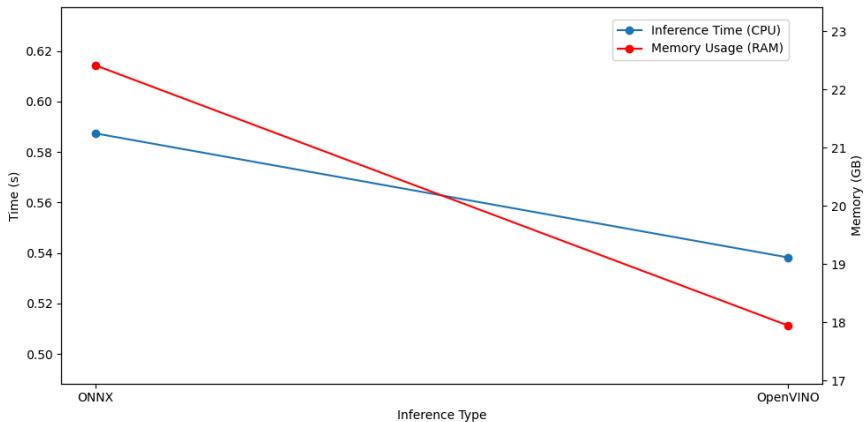
²<https://www.mvtec.com/company/research/datasets/mvtec-ad>

After that the time taken for inference and the memory usage (RAM on CPU or VRAM on GPU) across all the predictions made by a model were printed, e.g. for the FP32 PyTorch model:

- Time taken for inference (FP32): 2.23 seconds
- Memory used (VRAM): 4.83 GB

These statistics were then later compared between each model running on the same accelerator (CPU or GPU) and these were the results obtained:





As you can see from these results, the TensorRT model is the fastest model running on GPU, with an inference time improvement of 78.89%, and it does not use a lot more memory than the original FP32 PyTorch model, using 13.02% more VRAM. On CPU OpenVINO and ONNX perform roughly the same with respect to inference time while the OpenVINO model uses 20% less RAM.

The final analysis was made looking at the segmentation results of each model and comparing them using the AUROC and Dice metrics keeping the prediction made with the FP32 PyTorch model as the ground truth. These are the results looking at the first test image:



In the Jupyter Notebook you can have a more in-depth look at all the results of the different models.

Conclusion

In conclusion it can be said that the TensorRT model achieves much lower GPU inference time than the FP32 or FP16 PyTorch model with minimal memory usage increase and it still has high Dice and AUROC scores with respect to the FP32 model. This could lead to the use of the TensorRT model in an industrial setting to tackle real-time anomaly detection tasks.

Bibliography

- [1] S. Akcay, D. Ameln, A. Vaidya, B. Lakshmanan, N. Ahuja, and U. Genc, “Anomalib: A Deep Learning Library for Anomaly Detection.” [Online]. Available: <https://arxiv.org/abs/2202.08341>
- [2] T. Defard, A. Setkov, A. Loesch, and R. Audigier, “PaDiM: a Patch Distribution Modeling Framework for Anomaly Detection and Localization.” [Online]. Available: <https://arxiv.org/abs/2011.08785>
- [3] S. Zagoruyko and N. Komodakis, “Wide Residual Networks,” *CoRR*, 2016, [Online]. Available: <http://arxiv.org/abs/1605.07146>