NVIDIA TensorRT

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

Deep Learning

Simone Murari
VR502597
University of Verona

Contents

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

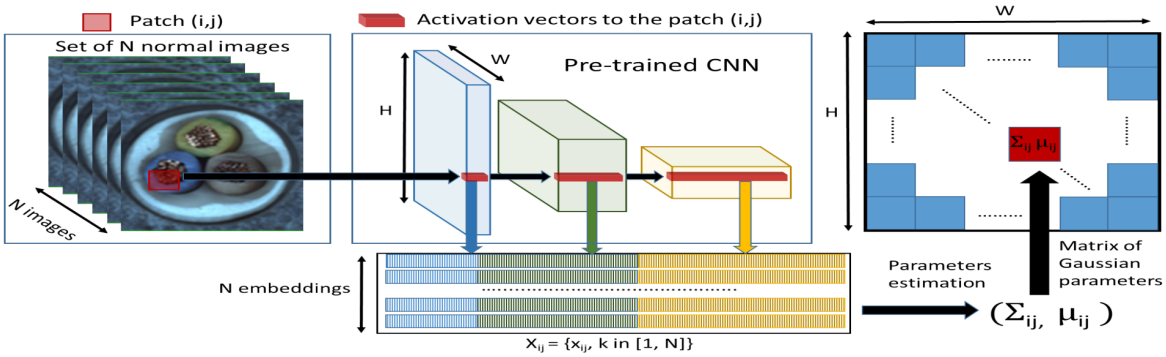
The libraries

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 **MVTec**[2] 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:

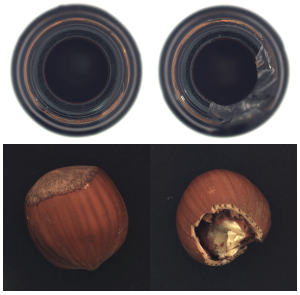


Figure 1: samples from MVTec dataset

The approach

The first step of the project was loading 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.

Experiments

The evaluation of the performance of each model was done with a quantitative and a qualitative analysis.

The quantitative metrics used include:

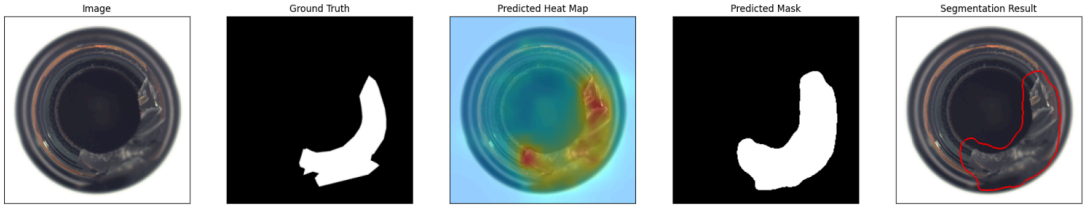
- Image-level F1 score: it measures the model's performance in correctly identifying the presence or absence of a particular class or object in an entire image
- Pixel-level F1 score: it measures the model's performance in correctly classifying individual pixels in an image
- Image-level AUROC: it measures the model's ability to distinguish between different classes at the whole-image level
- Pixel-level AUROC: it measures the model's ability to correctly classify individual pixels in an image across dynamically determined threshold values
- Dice coefficient: a statistical measure used to gauge the similarity between two samples
- Inference time: the time taken for the CPU or GPU to make inferences
- Memory used: the RAM or VRAM used by the model

For the qualitative analysis, for each framework a 5 image panel was plotted showing: the input image, the ground truth mask, the predicted heat map, the predicted mask and the segmentation result. This made it possible to have an immediate look at how a specific model performs. To compare the models between each other, a panel with the segmentation results of each model was plotted so the differences and similarities in the detection of anomalies were immediately evident.

[2] https://www.mvtec.com/company/research/datasets/mvtec-ad

Results

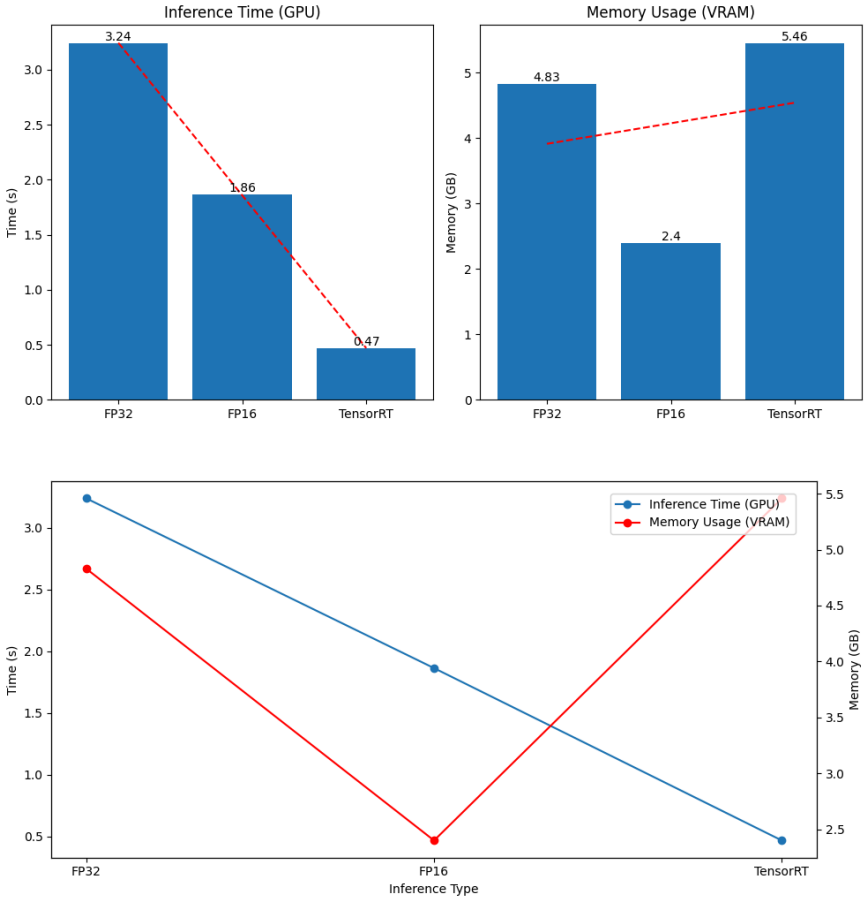
This is an example of qualitative analysis for the PyTorch FP32 model:



These are the quantitative results for the same model:

- Average pixel-level F1 score: 0.737
- Average image-level F1 score: 0.950
- Average pixel-level AUROC: 0.931
- Average image-level AUROC: 0.962
- Time taken for inference (FP32): 3.24 seconds
- Memory used (VRAM): 4.83 GB

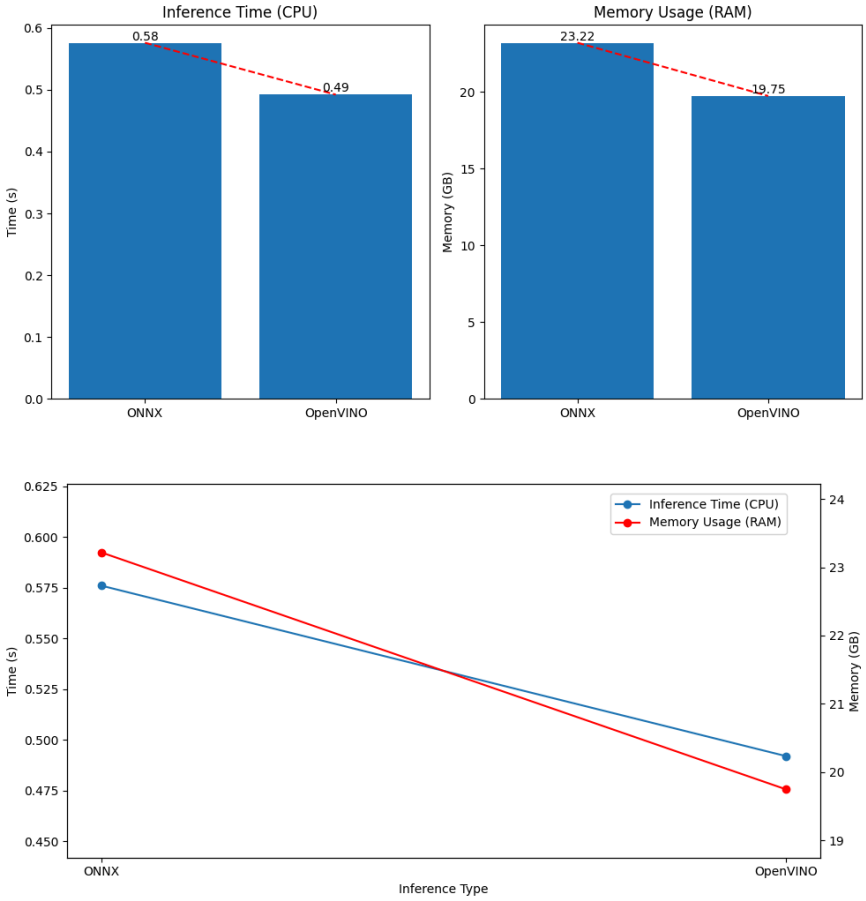
These graphs show the performance of the PyTorch and TensorRT models on GPU:



You can see that there is a clear downward trend in inference time on the GPU (represented by the red dashed line): the TensorRT model takes much less GPU time to make predictions than the PyTorch models, in fact there is an inference time improvement of 85.62%, while there is an upward trend in memory usage going from a PyTorch FP32 model to a TensorRT model with the same weights: the TensorRT model uses 13.02% more VRAM. This means that there is a tradeoff between inference time

and memory usage but the TensorRT model is a clear winner because it takes much less time using only just a bit more memory than the other models.

These graphs show the performance of the ONNX and OpenVINO models on CPU:



The OpenVINO model has an inference time improvement of 15% and uses 15% less RAM than the ONNX model. This could make the OpenVINO model preferable for hardware with RAM constraints because of the lower RAM usage.

The next image shows the segmentation comparison between all the models. It is highlighted with a blue circle the mistake that the ONNX, OpenVINO and TensorRT models made w.r.t. to the FP32 PyTorch model. This is also visible in their Dice and AUROC scores with the FP32 PyTorch prediction taken as ground truth: there is a slight decrease in the Dice coefficient meaning that they are less similar to the ground truth and a decrease in AUROC value.

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