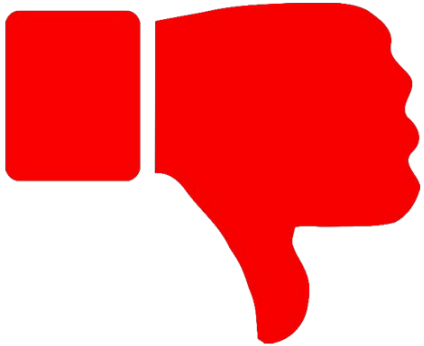
Image classifier using Tensorflow

An improvement inference speed model using dataloaders, Flask API and Docker containers

Issues and old performance

Issues:

- The model was downloaded from a public registry.
- Uncomment code.
- No logs management.
- No track experiment manager.
- Impossible to deploy in production environment.
- Bad quality code.
- No docker container provided.
- Model loaded each iteration.
- No documented code



Performance:

Poor time inference. The time spent in labeling 5 images is between 8~9 seconds on CPU

```
Time spent: 8.671179056167603
```

New functionalities

- Logs management: the code has a local and remote log manager. Accessible from any computer
- Deployed on both, one single script or in API. The user can get results instantly.
- “Private” model register.
- Docker for app deployment.
- Inference on device of your choice CPU or GPU.
- Model can read the whole folder for unit tests with local images and model data

Performance:

Excellent time inference. The time spent in labeling 5 images is between 0.3~0.4 seconds on CPU.

~20x faster inference!!!!

How we did it?

- The crucial tool is Weights and Biases: this online tool allow to register the model and also track all the experiments. **W&B ACCOUNT REQUIRED**.
- Tensoflow was the framework used for inference.
- Flask allowed to create an API for deployment
- Docker will allow to export the application to any server.

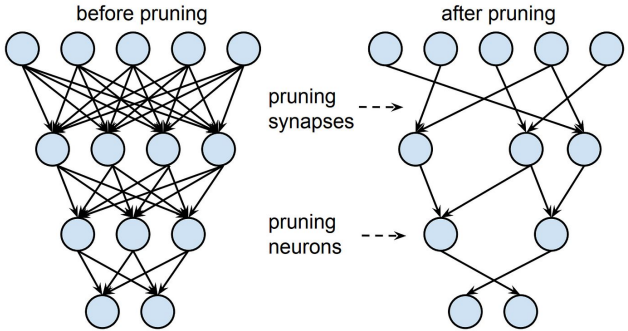
All models and code applied can be seen in these links:

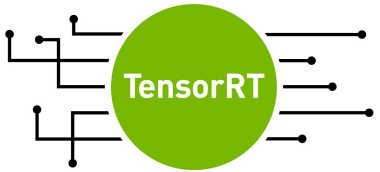
- https://github.com/hdnh2006/tensorflow-classifier
- https://hub.docker.com/r/hdnh2006/tensorflow-classifier
- https://wandb.ai/hdnh2006/bird_classifier

How to improve the model (next steps)

Multiple tasks can be carried out in order to improve model performance:

- Pruning and sparsity: some neurons could not affect the results and they just reduce the time inference.
- Calibration or half precision: half precision would highly reduce the time inference or int8 could reduce even more the speed inference.
- Export to OpenVino in case of CPU deployment.
- Export to TensorRT in case of GPU deployment.
- Add Grad-CAM to search explainability of the model.



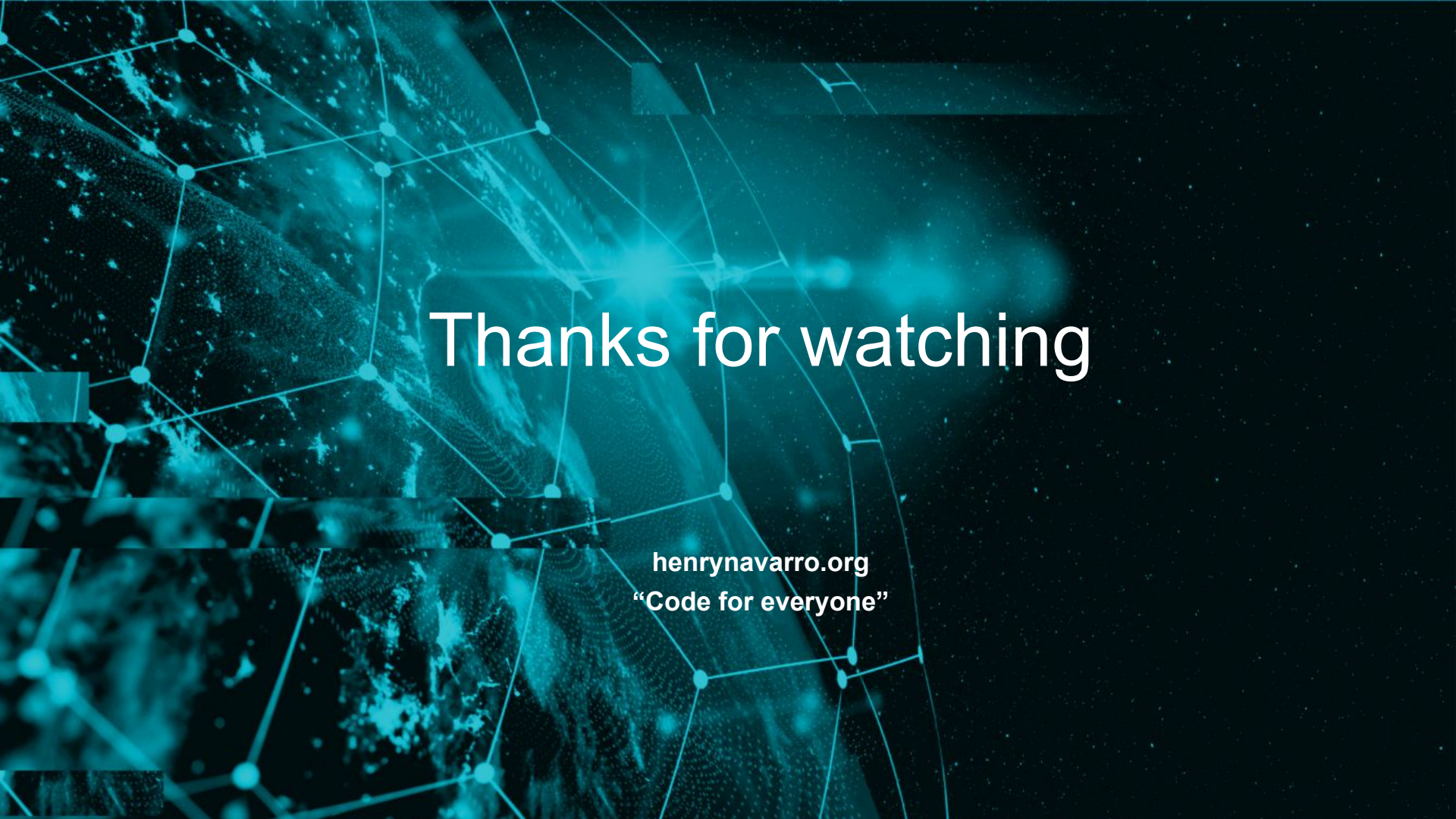


OpenVINO™

Architecture proposed

We propose a classic architecture for general purpose:

- Data storage: store new data on Amazon S3
- Container Orchestration: Use Amazon ECS to manage the deployment of the Dockerized image
- Load Balancing: The container orchestration platform will provides built-in load balancing across several instances.
- Auto Scaling: Utilize the auto scaling features to dynamically adjust the number of instances based on resource usage, incoming traffic, or custom metrics, etc.
- Health Checks: If a container becomes unhealthy, it is automatically replaced.
- Horizontal Scaling: add or remove instances of the Dockerized application based on demand.
- Monitoring and Metrics: Leverage monitoring and metrics provided by the container orchestration and Weights and Biases to track resource utilization, response times, and other relevant metrics.


Thanks for watching
henrynavarro.org
“Code for everyone”