Project Logbook

AMD Pervasive AI Contest: No More Service Dogs

Tables of content:

Initial Project Presentation:

This project is supposed to solve a main problem encountered by visually impaired individuals: their lack of independency. Indeed, without any external help (including a service dog or any human help), it is very hard for these individuals to navigate the world.

To solve this problem, I had the idea to use an image classification model finetuned on the Daily Object Dataset (https://www.kaggle.com/datasets/humansintheloop/dollar-street-dataset). I first thought about resnet-50 which has proven its capabilities many times. I then thought about implementing text to speech to make the user aware of their surroundings.

Here, the real advantage of using AMD AI technology is the fact that it makes this technology portable and usable daily for a visually impaired individual thanks to the fast execution times provided by the hardware. This once again contributes to make visually impaired people more independent because no internet access is required since everything is quickly executed locally.

Project Adjustments/Upgrades:

Something that I thought about was the fact that at the current state, the project could tell the user the general area that he was in but not what objects were precisely in these areas.

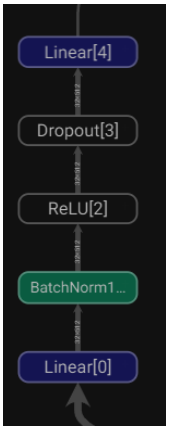
That is the reason why I decided to implement a fastercnn-resnet50 (pretrained on the COCOV1 dataset) to be able to recognize individual objects within the scene. After exporting it to ONNX, quantizing it and running it on the NPU, I managed to get very satisfying execution times on it.

Project Execution:

Model Finetuning:

At first, the goal was to finetune the chose image classification model (resnet50) for the purposes of the project.

To finetune the model, I had to change and train the classifier layer. As I was only going to update the parameters on the classifier, I decided to change the single linear layer to two linear layers with batch normalization and dropout in between to prevent overfitting (c.f classifier graph on the right).



I chose to finetune it on a part of the Daily Places Dataset. Indeed, a part of the dataset was reserved for Object Detection but I needed the first model to do image classification to detect what was the general area where the user was in. I then choose to keep only the "Places" part of the Daily Places Dataset (setting the "Objects" and "Abstract" parts apart).

Moving on to data preprocessing, the steps to extract the images from the dataset are visible in the "environnementModel\dataset.py" file.

For the training sequence I decided to go with some data augmentation to help the model generalize better.

After a trying different hyperparameters, I found that a learning rate of about 1e-5 alongside SGD with momentum and weight decay was the best combination to train the model.

After some training, the model reached an accuracy of 80 to 90% on the Daily Places Dataset.

Model Quantization:

- Resnet-50 for Image Classification

I first wanted to go with Quantization Aware Training for this model since I had to train it at some point. However, I noticed that this process took a long time for results that were not far for the results obtained by Post Training Quantization with ONNX.

I then proceeded to quantize the model using Vistis AI ONNX Quantizer's static quantization. I used the recommended CNN configuration to deploy on the IPU with maximum IPU compatibility. (The PTQ is visible in the environnementModel\ONNXVistisAIQuantization.py file).

- Resnet-50-fasterrcnn for Object Detection

I had a lot more trouble quantizing this mode, getting a lot of errors during preprocessing, and quantization. I indeed had to skip the symbolic shape during preprocessing and move away from the recommended CNN configuration.

This quantized model therefore does not run entirely on the IPU but still gets very satisfying execution times and accuracy. (The PTQ is visible in the objectsModel\VistisAIQuantization.py file).

Deployment:

After making a simple UI with PyQt5, I moved on to deploy the model alongside this UI. I first load the two ONNX inference sessions on the IPU. Then at, at the push of a button on the UI, a frame is captured by the camera and given to the models for prediction.

The current overall place where the user is in is written on the UI whereas the user is informed of the objects detected with Text-to-Speech if the option is enabled by the user.

Final results:

Execution Times:

Tests made on the final application: Finetuned resnet50 + fastercnn-resnet50

The tests are only taking the execution speeds of the two ONNX sessions on a single inference within the final application.

PTQ+IPU	FP32+CPU
0.63s	1.17s

Power Consumption:

Tests made on the final application: Finetuned resnet50 + fastercnn-resnet50

The power consumption is recorded by a software called HWMonitor (HWMonitor's website). It includes the overall power consumption of the monitored components with nothing other than the python app running.

I will be using the package power usage from the CPU as data here.

Model Running on IPU	Model Running on CPU
33.52W	42.16W

Utilization:

Installing the requirements:

You can install the library requirements by first activating the AMD Ryzen Software v1.1 conda environement and running pip install –r "requirements.txt" (file found on the root folder of the project).

Taking a look at the scripts:

The training script can be found under environnementModel\train.py.

! The dataset has been removed from the delivered project to save space

The quantization script for the finetuned resnet50 can be found under environnementModel\ONNXVistisAIQuantization.py.

To infere on the finetuned resnet50 on the IPU with any image, you can use the environnementModel\inference.py script.

The unused pytorch qat script can be found in environnementModel\unused.

For the fasterrcnn-50, the quantization script can be found under objectsModel\VistisAIQuantization.py.

To infere through the quantized fasterrcnn-resnet50 on the IPU, you can use objectsModel\ONNX_IPU_Inference.py.

Using the final app:

To use the final application, you can use the “FinalApp.py” file and the UI will pop up.

! You have to have a camera plugged in to use this file. If you want to infere on saved images, use the environnementModel\inference.py and objectsModel\ONNX_IPU_Inference.py files.