Deep Learning at the Edge – Activity 4

Task 4: Fruit Image Classification on the standalone Jetson Nano

Introduction

In this lab, we will go through the steps of creating an image classification project using a RESNET-18 Convolutional Neural Network (CNN) that is trained on the Jetson Nano developer kit.

Required Hardware

The Jetson Nano is an embedded Linux development kit featuring a GPU accelerated processor (NVIDIA Tegra) targeted at edge AI applications. For this Lab, you will be using a USB camera which is connected as device /dev/video0.

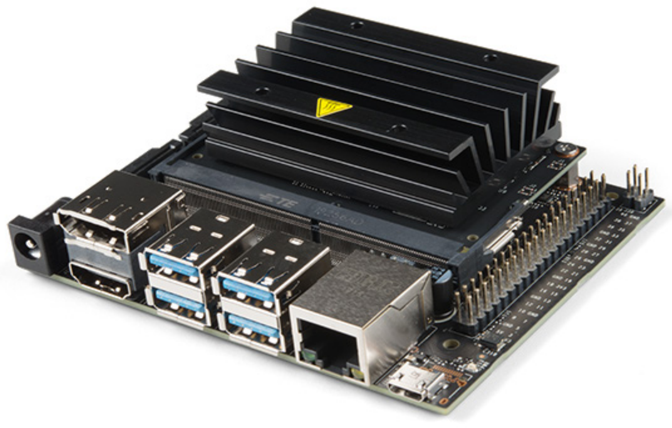


Fig. 1: Nvidia Jetson Nano

The Jetson Nano dev kit is already set up for you and can be accessed remotely using MobaXTerm software available on the host machine. The IP address for connecting remotely to the nano board is 192.168.55.1. Open a terminal connection using SSH and when prompted, enter the password that is ‘nvidia**X**’ for your Jetson device. (Note that ‘**X**’ is the specific board number). Run the following command from the ~/home/nvidiaX directory. We will be using a docker container that includes all the necessary scripts and correct environment settings for performing image classification on the Jetson Nano. You may be asked to enter the same password again to give you super user access.

```
cd jetson-inference

docker/run.sh
```

1. Collect Data

We will start with a small 'fruits' image dataset. The dataset contains a separate set of train, test and valid images for 8 categories of fruit.

- apple – train: ~40 images, val: 5 images, test:2 images
- banana – train: ~40 images, val: 5 images, test:2 images
- grape – train: ~40 images, val: 5 images, test:2 images
- pear – train: ~40 images, val: 5 images, test:2 images
- pineapple – train: ~40 images, val: 5 images, test:2 images
- strawberry – train: ~40 images, val: 5 images, test:2 images
- watermelon – train: ~40 images, val: 5 images, test:2 images
- banana – train: ~40 images, val: 5 images, test:2 images

Unzip the fruits.zip file into the /*data* folder using the below commands:

```
cd python/training/classification/data

unzip fruits.zip
```

Notice that a *labels.txt* file along with a *train*, *test* and *val* folder containing images in *data/fruits/* folder is now created. The Jetson Nano expects the labels to be in alphabetical order.

This dataset will be used to train a RESNET-18 CNN model on the Jetson Nano.

Importing this dataset into Edge Impulse (optional)

If you like, you could also import this dataset to an Edge Impulse project by:

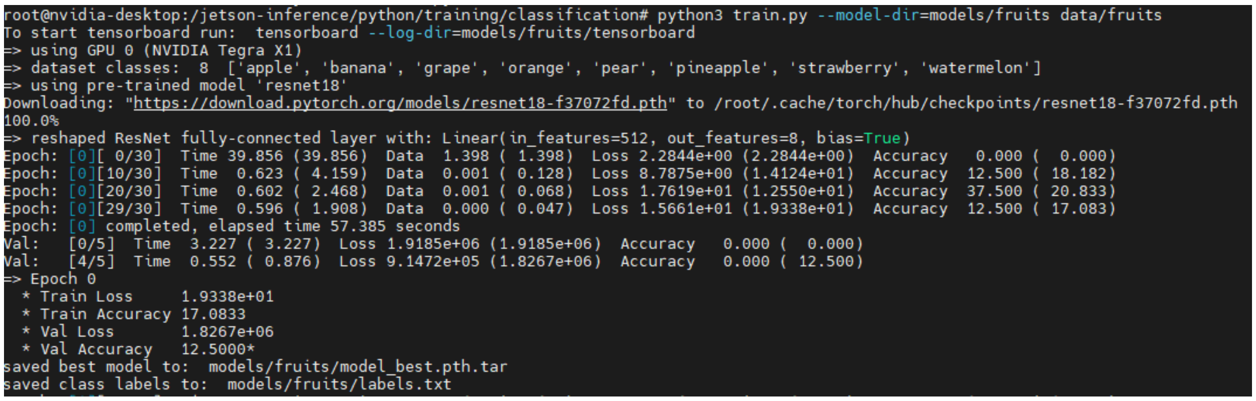
1. Downloading the "**fruits.zip**" dataset from https://github.com/brenmul/deep_learning_lab/tree/main/datasets
2. Unzip the file in a location of your choice.
3. In your Edge Impulse project, go to **Data acquisition** and click on the 'Upload icon'. Follow the instructions on the screen.

2. Train the Model on the Jetson Nano

Head back to the classification folder in your Jetson Nano console and run the following script which uses the PyTorch framework to train the model.

```
cd /jetson-inference/python/training/classification

python3 train.py --model-dir=models/fruits data/fruits
```

By default, the CNN model that is trained is a ResNet-18 model - as training begins, you should see text appear from the console like the following:



```
root@nvidia-desktop:/jetson-inference/python/training/classification# python3 train.py --model-dir=models/fruits data/fruits
To start tensorboard run:  tensorboard --log-dir=models/fruits/tensorboard
=> using GPU 0 (NVIDIA Tegra X1)
=> dataset classes:  8  ['apple', 'banana', 'grape', 'orange', 'pear', 'pineapple', 'strawberry', 'watermelon']
=> using pre-trained model 'resnet18'
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100.0%
=> reshaped ResNet fully-connected layer with: Linear(in_features=512, out_features=8, bias=True)
Epoch: [0][ 0/30]  Time 39.856 (39.856)  Data  1.398 ( 1.398)  Loss 2.2844e+00 (2.2844e+00)  Accuracy   0.000 (  0.000)
Epoch: [0][10/30]  Time  0.623 ( 4.159)  Data  0.001 ( 0.128)  Loss 8.7875e+00 (1.4124e+01)  Accuracy  12.500 ( 18.182)
Epoch: [0][20/30]  Time  0.602 ( 2.468)  Data  0.001 ( 0.068)  Loss 1.7619e+01 (1.2550e+01)  Accuracy  37.500 ( 20.833)
Epoch: [0][29/30]  Time  0.596 ( 1.908)  Data  0.000 ( 0.047)  Loss 1.5661e+01 (1.9338e+01)  Accuracy  12.500 ( 17.083)
Epoch: [0] completed, elapsed time 57.385 seconds
Val:   [0/5]  Time  3.227 ( 3.227)  Loss 1.9185e+06 (1.9185e+06)  Accuracy   0.000 (  0.000)
Val:   [4/5]  Time  0.552 ( 0.876)  Loss 9.1472e+05 (1.8267e+06)  Accuracy   0.000 ( 12.500)
=> Epoch 0
  * Train Loss      1.9338e+01
  * Train Accuracy 17.0833
  * Val Loss        1.8267e+06
  * Val Accuracy    12.5000*
saved best model to:  models/fruits/model_best.pth.tar
saved class labels to:  models/fruits/labels.txt
```

- Epoch: an epoch is one complete training pass over the dataset
  - Epoch: [N] means you are currently on epoch 0, 1, 2, ect.
  - The default is to run for 35 epochs (you can change this with the --epochs=N flag)
- [N/30] is the current image batch from the epoch that you are on.
  - Training images are processed in mini-batches to improve performance
  - The default batch size is 8 images, which can be set with the --batch=N flag
  - Multiply the numbers in brackets by the batch size (e.g. batch [10/30] => image [80/240])
- Time: processing time of the current image batch (in seconds)
- Data: disk loading time of the current image batch (in seconds)
- Loss: the accumulated errors that the model made (expected vs. predicted)
- Train Accuracy/Loss:
  - This gives the training set accuracy and loss performance.
- Valid Accuracy/Loss
  - This gives the valid set accuracy and loss performance.

You can keep an eye on these statistics during training to gauge how well the model is trained, and if you want to keep going or stop and test, you can restart training again later if you desire. It looks like this model needs more epochs or data to do better!

3. Convert the model to ONXX format

To run our re-trained ResNet-18 model with TensorRT for testing and real time inference, first we need to convert the PyTorch model into ONNX format so that TensorRT can load it. ONNX is an open model format that supports many of the popular ML frameworks, including PyTorch, TensorFlow, TensorRT, and others, so it simplifies transferring models between tools.

PyTorch comes with built-in support for exporting PyTorch models to ONNX, so run the following command to convert our generated *fruits* model with the provided onnx_export.py script:

```
cd /jetson-inference/python/training/classification

python3 onnx_export.py --model-dir=models/fruits
```

This will create a model *resnet18.onnx* in the following folder

/jetson-inference/python/training/classification/models/fruits/

4. Testing - Processing Images with TensorRT

To classify some static test images, we'll use the extended command-line parameters to imagenet to load our customized ResNet-18 model that we re-trained above. To run these commands, the working directory of your terminal should still be located in: jetson-inference/python/training/classification/

First make a *test_output/apple* folder in the *data/fruits* directory and then run *imagenet.py* script as shown below.

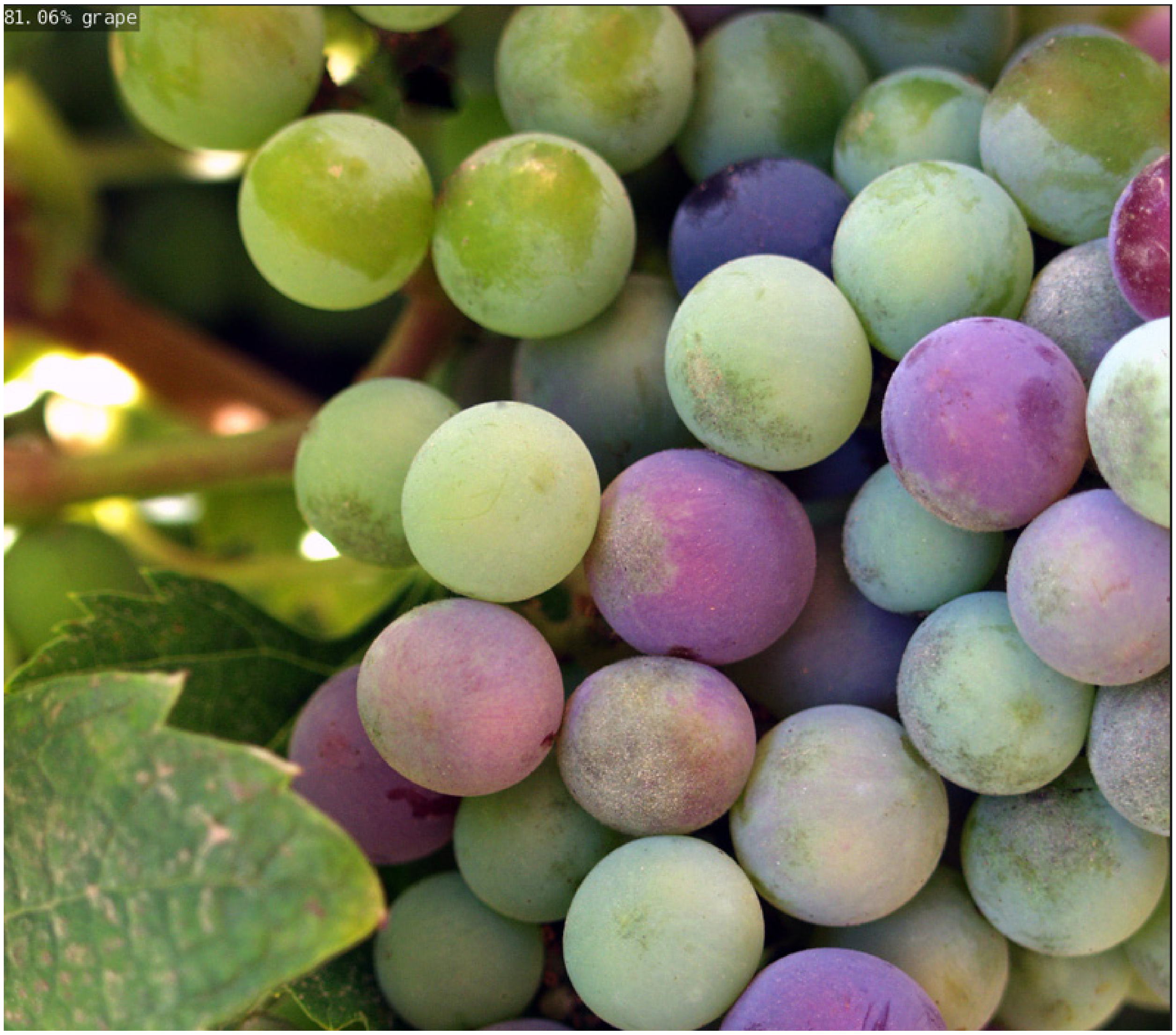
```
cd /jetson-inference/python/training/classification

mkdir data/fruits/test_output/apple

imagenet.py --model=models/fruits/resnet18.onnx --input_blob=input_0 --
output_blob=output_0 --labels=data/fruits/labels.txt data/fruits/test/apple
data/fruits/test_output/apple/
```

This script runs the onnx model classification by passing images in the *data/fruits/test/apple* folder and sending the results to the *data/fruits/test_output/apple/* location.

The above code can be modified to test all the fruit categories.

You can then view the generated images with the classification results overlayed in the image (top right hand side) by viewing the images in the *test_output* folder.

Here we can see that the grape image is correctly classified with 81% confidence.



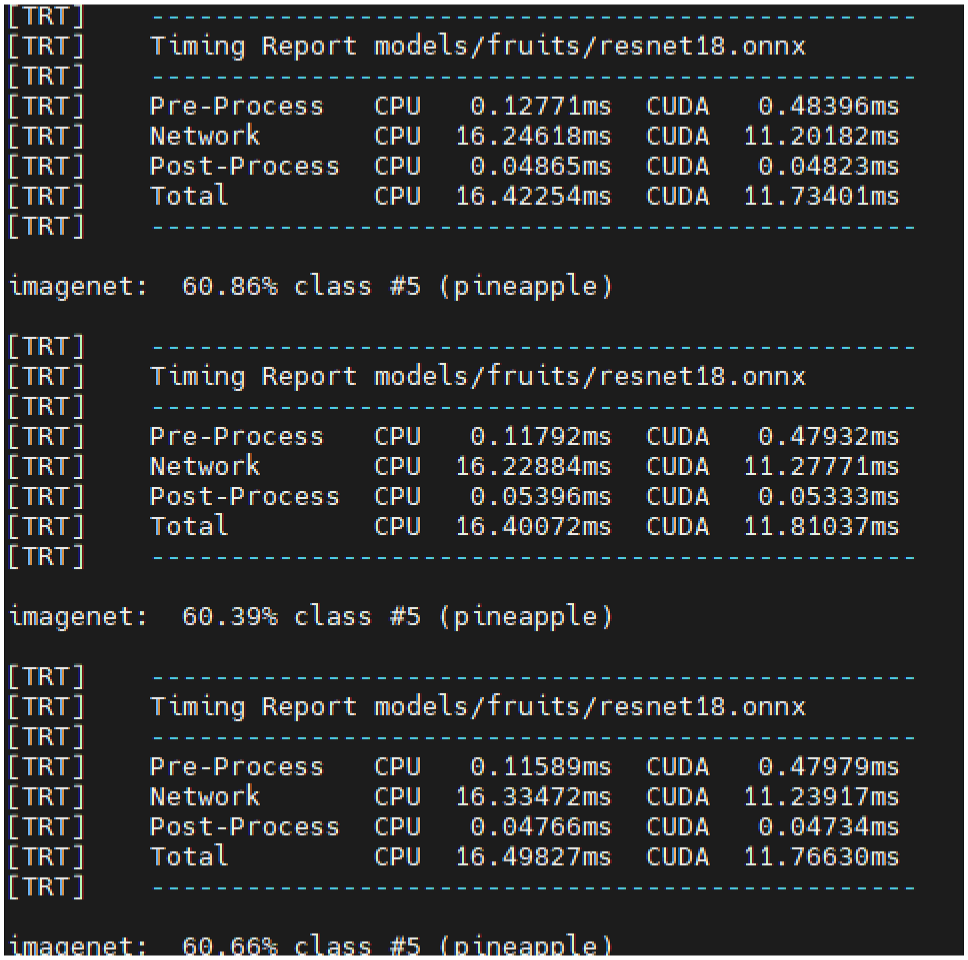
5. Testing – Running the live camera

If you have some spare fruit lying around, you can run the camera program and see how it works!

Like the previous step, imagenet supports extended command-line parameters for loading customized models: Use the following command to set up the USB camera as input and to stream the video data remotely.

```
cd /jetson-inference/python/training/classification

imagenet.py --model=models/fruits/resnet18.onnx --input_blob=input_0 --
output_blob=output_0 --labels=data/fruits/labels.txt /dev/video0 webrtc://@:8554/output
```

You should see similar text as below in the console window.

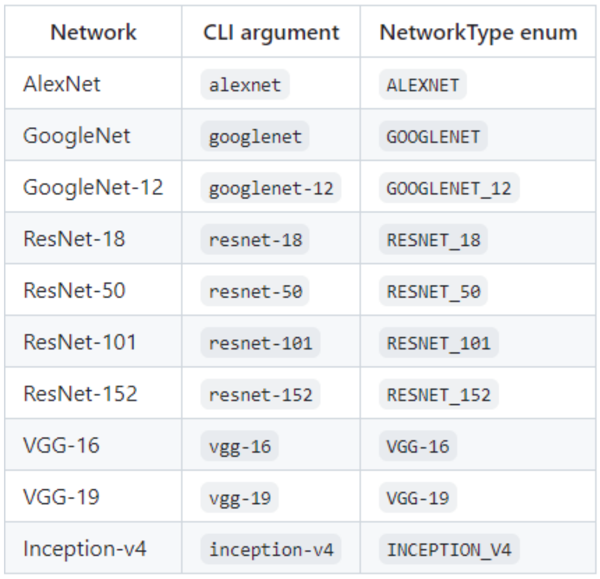


```
[TRT]    ---------------------------------------------------
[TRT]    Timing Report models/fruits/resnet18.onnx
[TRT]    ---------------------------------------------------
[TRT]    Pre-Process   CPU   0.12771ms  CUDA   0.48396ms
[TRT]    Network       CPU  16.24618ms  CUDA  11.20182ms
[TRT]    Post-Process  CPU   0.04865ms  CUDA   0.04823ms
[TRT]    Total         CPU  16.42254ms  CUDA  11.73401ms
[TRT]    ---------------------------------------------------

imagenet:  60.86% class #5 (pineapple)

[TRT]    ---------------------------------------------------
[TRT]    Timing Report models/fruits/resnet18.onnx
[TRT]    ---------------------------------------------------
[TRT]    Pre-Process   CPU   0.11792ms  CUDA   0.47932ms
[TRT]    Network       CPU  16.22884ms  CUDA  11.27771ms
[TRT]    Post-Process  CPU   0.05396ms  CUDA   0.05333ms
[TRT]    Total         CPU  16.40072ms  CUDA  11.81037ms
[TRT]    ---------------------------------------------------

imagenet:  60.39% class #5 (pineapple)

[TRT]    ---------------------------------------------------
[TRT]    Timing Report models/fruits/resnet18.onnx
[TRT]    ---------------------------------------------------
[TRT]    Pre-Process   CPU   0.11589ms  CUDA   0.47979ms
[TRT]    Network       CPU  16.33472ms  CUDA  11.23917ms
[TRT]    Post-Process  CPU   0.04766ms  CUDA   0.04734ms
[TRT]    Total         CPU  16.49827ms  CUDA  11.76630ms
[TRT]    ---------------------------------------------------

imagenet:  60.66% class #5 (pineapple)
```

In your web browser go to http://192.168.55.1:8554/ to view the camera view remotely.

Question: What results did you get? Try pointing the camera at some Google images – how does this model perform and what can be done to improve the performance?

Note that you can specify which model to load by setting the --arch flag on the command line for the *train.py* script to one of the corresponding CLI arguments from the table below.



Network	CLI argument	NetworkType enum
AlexNet	`alexnet`	`ALEXNET`
GoogleNet	`googlenet`	`GOOGLENET`
GoogleNet-12	`googlenet-12`	`GOOGLENET_12`
ResNet-18	`resnet-18`	`RESNET_18`
ResNet-50	`resnet-50`	`RESNET_50`
ResNet-101	`resnet-101`	`RESNET_101`
ResNet-152	`resnet-152`	`RESNET_152`
VGG-16	`vgg-16`	`VGG-16`
VGG-19	`vgg-19`	`VGG-19`
Inception-v4	`inception-v4`	`INCEPTION_V4`

Example: To retrain the fruits dataset using a larger resnet-50 model, use the following command.

```
cd /jetson-inference/python/training/classification

python3 train.py --arch=resnet-50 --model-dir=models/fruits data/fruits
```

6. Conclusion

Congratulations, I hope this lab has helped you get familiar with training an image classification exercise on the Jetson Nano development kit and then testing its performance.

7. Acknowledgement

This lab material was guided by the contents of the Nvidia Jetson - Hello AI world. For more information, you can access https://github.com/dusty-nv/jetson-inference

The *fruits* dataset was obtained from https://www.vicos.si/resources/fids30 citing the authors work *Automatic fruit recognition using computer vision* by Škrjanec Marko.