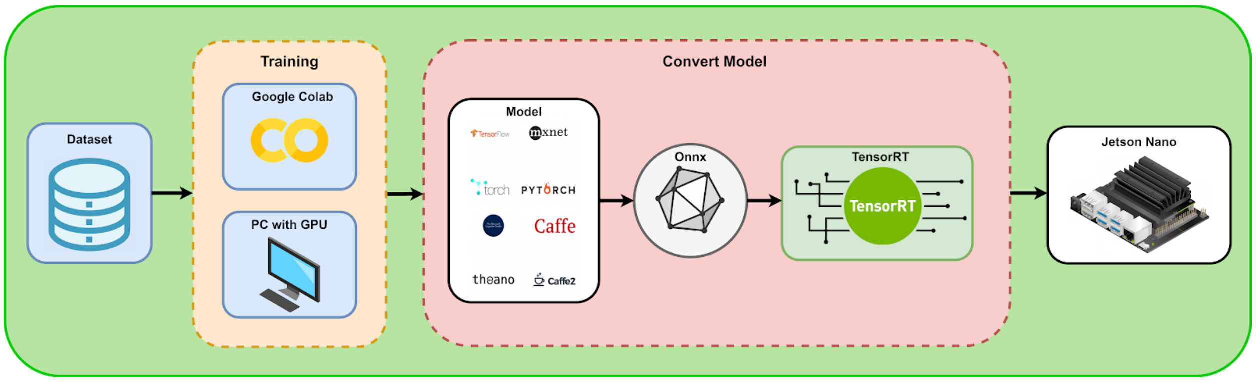
Transfer Learning for Edge Devices

Introduction

Creating deep learning models from scratch can be time consuming and not the best use of your time. We are fortunate to have many pre-trained models available that we can use as a starting point for our own applications. We can use transfer learning to re-train only a small portion that we need in order to develop our feature or solve our problem.

For this project we will be re-training a base Model SSD-mobilenet-v1 pre-trained on the PASCAL VOC dataset for object detection. We will download data from Google Open Images to retrain the model to detect a custom object class (vehicle license plate). We will then convert and deploy model onto an Nvidia Jetson Nano.



Download dataset for training

Our base model can detect 20 objects out of the box but we want to re-train it to detect license plates. We will be using Google Open Images to download our dataset. We have a choice of 600 different classes but we will be using 'Vehicle registration plate' for our model. Using the website's provided script, we end up downloading 6800 images conveniently split up into test/train/validation. This process can be done with any number of classes but we will be using one class for our example.

```
python3 image_downloader.py --class-names 'Vehicle registration plate' --data=data/plate
```

```
.-03-12 20:08:48 - Requested 1 classes, found 1 classes
.-03-12 20:08:48 - Read annotation file data/plate/train-annotations-bbox.csv
.-03-12 20:09:09 - Available train images:  5365
.-03-12 20:09:09 - Available train boxes:   7849

.-03-12 20:09:09 - Read annotation file data/plate/validation-annotations-bbox.csv
.-03-12 20:09:10 - Available validation images:  385
.-03-12 20:09:10 - Available validation boxes:   511

.-03-12 20:09:10 - Read annotation file data/plate/test-annotations-bbox.csv
.-03-12 20:09:12 - Available test images:  1109
.-03-12 20:09:12 - Available test boxes:   1566

.-03-12 20:09:12 - Total available images: 6859
.-03-12 20:09:12 - Total available boxes:  9926
```

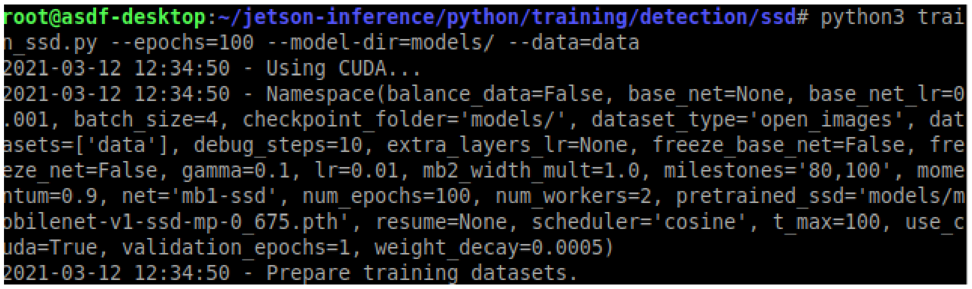
The images themselves vary quite a bit in terms of viewing depth/angles and license plate types. We will be trying to detect USA license plates but our dataset includes plates from all over the world.

Dataset Name: Google Open Images dataset v6 (600 classes)
Dataset Link : [LINK]
Dataset Size: [3.18] GB
Dimensions: Width [1024, 480], Height [1024,575]
of images: 6859   [.JPEG]
of classes: 1. ('Vehicle registration plate')
of images per class: 6859
[Train images #= 5365
Test images #= 1109
Validation images #= 385]
Images file size: [.035, 1.9] MB

A big limitation of this dataset is its high variance in viewing angles, distances, cars and plate types. Unfortunately, many of the highly curated datasets related to this particular object class are proprietary and hard to find.

Re-training existing model

For this project we first re-trained on the Jetson Nano 2gb using the provided Jetson Utitlities script with 100 epochs.



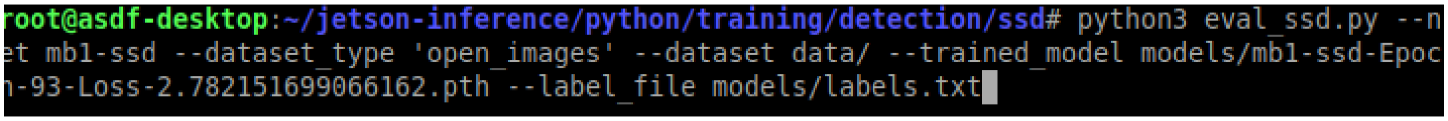
```
root@asdf-desktop:~/jetson-inference/python/training/detection/ssd# python3 trai
n_ssd.py --epochs=100 --model-dir=models/ --data=data
2021-03-12 12:34:50 - Using CUDA...
2021-03-12 12:34:50 - Namespace(balance_data=False, base_net=None, base_net_lr=0
.001, batch_size=4, checkpoint_folder='models/', dataset_type='open_images', dat
asets=['data'], debug_steps=10, extra_layers_lr=None, freeze_base_net=False, fre
eze_net=False, gamma=0.1, lr=0.01, mb2_width_mult=1.0, milestones='80,100', mome
ntum=0.9, net='mb1-ssd', num_epochs=100, num_workers=2, pretrained_ssd='models/m
obilenet-v1-ssd-mp-0_675.pth', resume=None, scheduler='cosine', t_max=100, use_c
uda=True, validation_epochs=1, weight_decay=0.0005)
2021-03-12 12:34:50 - Prepare training datasets.
```

The 2Gb Nano is not very powerful for training and as a result takes almost 5 days to train. A faster way to do this is by using Google Colab or your own higher powered GPU. We trained 100 epochs in 6.7 Hours using Google Colab on an Nvidia T4. The average time per epoch on the Nano was 86.6 mins while Google Colab took 4 mins per epoch.

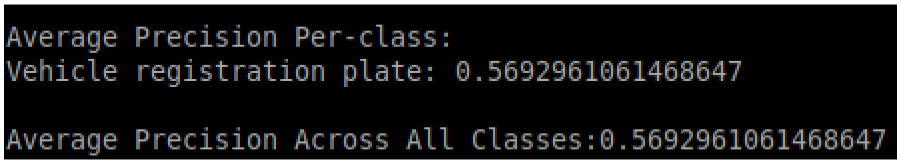
At the end of our training we achieved a loss of 2.733.

Model Evaluation

The next step is to evaluate our model against some test data. Fortunately, during our initial download of our Open Images dataset we created a test folder we can use to evaluate. The test folder has 1109 Images. Running our evaluation script, we achieve a mAP of .569.



```
root@asdf-desktop:~/jetson-inference/python/training/detection/ssd# python3 eval_ssd.py --n
et mb1-ssd --dataset_type 'open_images' --dataset data/ --trained_model models/mb1-ssd-Epoc
h-93-Loss-2.782151699066162.pth --label_file models/labels.txt
```



```
Average Precision Per-class:
Vehicle registration plate: 0.5692961061468647

Average Precision Across All Classes:0.5692961061468647
```

Getting more than half recognition is ok performance but again the test set pictures are quite variable.

Some pre-trained license plate detection models can have mAP of over 90%.

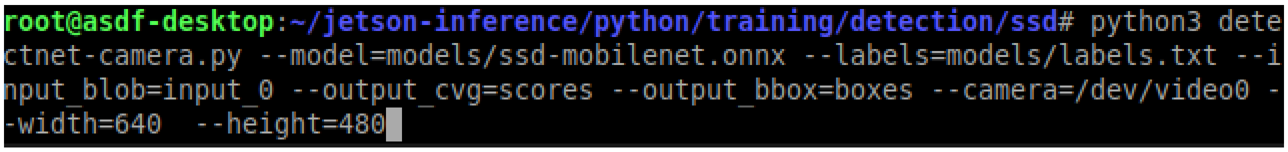
The next step is to convert this model and deploy it to our Jetson Nano.

Converting and Deploying

After re-training our model we need to convert it to an .onnx file to load onto the Jetson Nano.

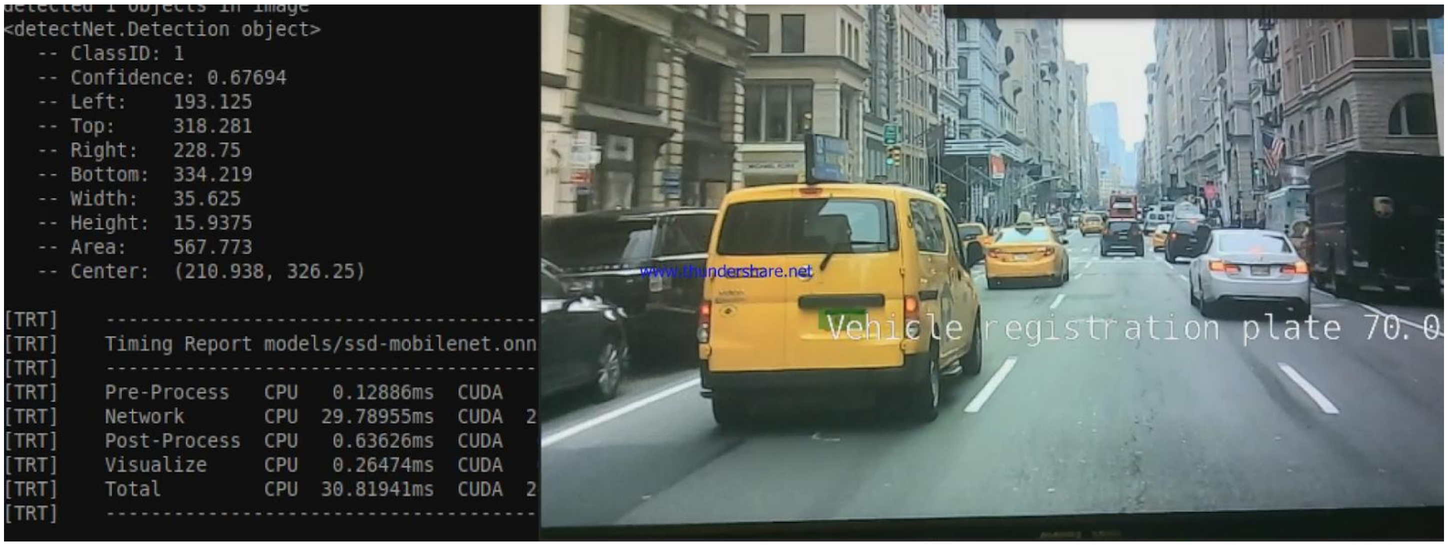
ONNX is an open model format that supports many of the popular ML frameworks, including PyTorch, TensorFlow, TensorRT, and others, so it simplifies transferring models between tools.

We can use the onnx_export.py script provided by jetson utilities. Then with TensorRT (pre-loaded onto the nano) we can optimize the model for deployment on the Jetson Nano.



```
root@asdf-desktop:~/jetson-inference/python/training/detection/ssd# python3 dete
ctnet-camera.py --model=models/ssd-mobilenet.onnx --labels=models/labels.txt --i
nput_blob=input_0 --output_cvg=scores --output_bbox=boxes --camera=/dev/video0 -
-width=640  --height=480
```

Running in real-time on the Jetson Nano we get a consistent 30fps.



```
detected 1 objects in image
<detectNet.Detection object>
   -- ClassID: 1
   -- Confidence: 0.67694
   -- Left:    193.125
   -- Top:     318.281
   -- Right:   228.75
   -- Bottom:  334.219
   -- Width:   35.625
   -- Height:  15.9375
   -- Area:    567.773
   -- Center:  (210.938, 326.25)

[TRT]    ------------------------------------
[TRT]    Timing Report models/ssd-mobilenet.onn
[TRT]    ------------------------------------
[TRT]    Pre-Process   CPU   0.12886ms  CUDA
[TRT]    Network       CPU  29.78955ms  CUDA  2
[TRT]    Post-Process  CPU   0.63626ms  CUDA
[TRT]    Visualize     CPU   0.26474ms  CUDA
[TRT]    Total         CPU  30.81941ms  CUDA  2
[TRT]    ------------------------------------
```

In this image, we ran a youtube video and pointed a web-camera at the screen for real-time detection. Overall the device captured most vehicles that were on screen, the caveat being the viewing angles were very tight. Viewing angles beyond 30* don't seem to get detected.

Potential Applications

We are only detecting one object in this example to save training time and to deploy on a low end edge device. Even with a Jetson Nano (entry level compute product) we can get good real-time detection. We could potentially expand this to detect several objects of our choosing like pose estimation or facial recognition. Perhaps we can make a smart doorbell or a room monitor.

Pros and Cons of Transfer Learning Method

Some potential applications include medical imaging and sensing, smarter NLP and chatbots, computer vision problems, forecasting specific events.

The Pros of Transfer Learning

- Increase performance vs. starting from scratch
- Saves time
- Doesn’t necessarily need massive amounts of data

The Cons of Transfer Learning

- Need model that’s somewhat related to your task, sometimes they don’t exist
- Risk of negative transfer where model ends up performing worse

Conclusion

The new paradigm of transfer learning allows for smaller organizations and individuals to create their specific features for their specific application. This is more akin to learning from previous generations and building upon that collective knowledge. Using pre-trained models saves us time and energy to try and develop our own unique features and uses.

Transfer learning is popular because of its wide range of applications but not all problems are suited for this type of method. Identifying the problem beforehand will greatly aid you in figuring out the best method to deploy. Overall, transfer learning techniques are here to stay and will continue to be an important part of AI and Machine Learning applications in the future.