San Francisco State University
School of Engineering

Hello AI World
With NVIDIA Jetson Nano 2GB

SFSU Bioelectronics Laboratory
By: Tigran Kurkchiyants
June 7, 2021

Contents

1 Introduction

The purpose of this guide is to provide instructions on using the principles of computer vision and machine learning to build and train deep neural networks using NVIDIA Jetson Nano 2GB Developer Kit. Jetson Nano is a smal and powerful edge device that processes neural networks with its 128-core graphics processing unit (GPU). Like many other microprocessors, Jetson Nano runs the Ubuntu operating system, which is provided by NVIDIA in its JetPack (software development kit for AI applications).

This guide mainly focuses on two types of image recognition: classification and detection. The former one, as the name suggests, is used to classify an object and the latter type is used to detect an object and draw a bounding box around it. In our case, we will be running real-time inference using a camera module. In other words, we will connect a camera, and it will classify and detect whatever objects we train our models on. There are other ways of running inference such as having a set of pictures or videos from the internet but we will use a camera to make it a little bit more interactive.

While training our own models, transfer learning will be used on deep neural networks (DNN). This means that we will take already pre-trained models and add our custom data to it making it more specific for our implementation. Transfer learning makes training much easier and faster than training a model from scratch, which is a very computationally- and time-expensive process. Additional information on types of image recognition can be found here.

2 Setting Up Jetson Nano

2.1 JetPack Overview

To set up Jetson Nano, we have to download NVIDIA JetPack, which is a comprehensive Software Development Kit (SDK) for developing and deploying AI applications. JetPack is a one-stop package, which simplifies installation of the operating system and all the necessary drivers. It contains the following:

- **Operating System.** NVIDIA L4T provides the bootloader, Linux kernel 4.9, necessary firmwares, NVIDIA drivers, sample filesystem based on Ubuntu 18.04, and more.
- **TensorRT.** TensorRT is a high performance deep learning inference runtime for image classification, segmentation, and object detection neural networks. TensorRT is built on CUDA, NVIDIA's parallel programming model, and enables you to optimize inference for all deep learning frameworks.
- **cuDNN.** CUDA Deep Neural Network library provides high-performance primitives for deep learning frameworks. It provides highly tuned implementations for standard

routines such as forward and backward convolution, pooling, normalization, and activation layers.

- **CUDA.** CUDA Toolkit provides a comprehensive development environment for C and C++ developers building GPU-accelerated applications. The toolkit includes a compiler for NVIDIA GPUs, math libraries, and tools for debugging and optimizing the performance of your applications.
- **Multimedia API.** The Jetson Multimedia API package provides low level APIs for flexible application development.
- **Computer Vision.** Computer vision library, visual computing applications.
- **Developer Tools.** CUDA Toolkit provides a comprehensive development environment for C and C++ developers building high-performance GPU-accelerated applications with CUDA libraries.

Download NVIDIA SDK here.

2.2 Flashing the OS

In order to install the operating system on Jetson Nano, we will have to flash a microSD card with the SDK file we have downloaded in Section 2.1. To prepare the microSD card, you will need a computer with the internet connection and the ability to read and write SD cards, such as a built-in SD card reader or an adapter. Steps for writing the image of the SDK are slightly different depending on what operating system you are using. Steps for Windows, MacOS, and Linux can be found on the official NVIDIA page here. Once you open the link, navigate to the "Write Image to the microSD Card" tab on the left-hand side, where you will find instructions for the three operating systems. Follow the instructions and once the flashing process is done, proceed to the next section.

2.3 First Boot

1. Insert the microSD card (with system image already written to it) into the slot underneath the Jetson Nano module.
2. Set the device on a desk or the paper box it came in.
3. Turn on a monitor and connect it to the Nano. Also connect a USB keyboard, mouse, Wi-Fi adapter, and a Raspberry Pi or USB camera.
4. Connect your DC power supply to the Jetson Nano. The device will power on and boot automatically. A green LED next to the Micro-USB connector will light as soon as the developer kit powers on. When you boot the first time, the developer kit will take you through some initial setup, including:

- Review and accept NVIDIA Jetson software EULA
- Select system language, keyboard layout, and time zone
- Create username, password, and computer name
- Optionally configure wireless networking
- Select APP partition size. It is recommended to use the max size suggested
- Create a swap file. It is recommended to create a swap file

After logging in, you will see the home screen. Connect to a Wi-Fi network and at this point, we are ready to clone files from Github.

3 Downloading Files

Now, that the operating system is installed, we can download necessary libraries and support packages from Github. Open up the terminal and run the following commands:

```
$ git clone --recursive https://github.com/dusty-nv/jetson-inference
$ cd jetson-inference
$ docker/run.sh
```

As you can see from the last command, we will be using a Docker container because it automatically pulls the correct container tag from DockerHub based on your currently-installed version of JetPack-L4T. This way, the Docker container will automatically download all additional packages and libraries we need for deep learning. After running the last command, a window will pop up, which will ask you to select which pre-trained models to download. We will leave everything as default for now and hit Enter to continue. In the future, you can download more models and experiment with them. At this time, let us see if the camera is recognized and works properly. Run the following commands:

```
$ ls /dev/video*
$ video-viewer csi://0
```

If your camera is recognized [1], the output of the first command should be something like

```
/dev/video0
```

and the output of the second command will activate your camera. If everything works properly, all setup steps have been done, and we can start running inference and training our own models. If, for some reason, your camera is not detected, check your connection and reboot Jetson Nano.

[1] If you are using a USB camera instead of MIPI Raspberry Pi camera module, replace csi://0 with dev/video0 for all commands found in this guide.

4 Object Classification

4.1 Inference With Pre-Trained Models

To get your feet wet, let us begin by running a real-time inference on already pre-trained models. We can explore what objects are recognized and play around with them. Full list of objects that the model can recognize can be found here. To classify objects using a default model (GoogleNet in this case), perform the following steps:

1. Open the terminal
2. Run the following commands:

```
$ cd jetson-inference/
$ docker/run.sh
$ imagenet csi://0
```

4.2 Inference With Custom Models

Now, let us train our own model for image classification. As mentioned in **Introduction** (Section 1), we will use transfer learning to train our model, which is essentially loading the pre-trained model and adding our custom dataset to it. There are many ways of collecting custom datasets. For example, we can download images from the internet or take our own using the Raspberry Pi camera module. Since we have a camera, let us proceed with the latter option. To collect our own images, the Github repo provides "camera-capture" tool for capturing and labeling images on Jetson Nano from live video. To start collecting images, open the terminal and run the following commands:

```
$ cd jetson-inference
$ docker/run.sh
$ cd python/training/classification
$ camera-capture csi://0
```

Feel free to select whatever objects you have at hand. In this case, I will use several tools I have found around the house. Before we start taking pictures of them, we have to create a folder with a .txt file in it. Do the following:

1. Open up file explorer and navigate to

   jetson-inference/python/training/classification/data

2. Create a new folder named **tools**
3. Inside the **tools** folder, right-click and choose Create New → Empty File. Name the new file **labels.txt**

4. Double click on **labels.txt** and write all classes we intend to train on in alphabetical order. It is paramount that the labels are alphabetized, otherwise, objects will not be correctly classified. For example, my **labels.txt** looks like the following:

   background

   drill

   hammer

   screwdriver

Now, in the camera capture window, leave Dataset Type as Classification but set Dataset Path to the **tools** folder. For Class Labels, choose **labels.txt** that you have created. At this point, we are ready to start taking pictures. For best results, it is recommended to take several pictures of an object in the same position. I will take 100 pictures for each class for train, 20 and 20 for val and test respectively (total of 560 pictures). For best results, it is suggested to follow the 80–10–10 rule: 80% of pictures should be in **train**, 10% in **val**, and another 10% in **test**.
The tool will create datasets with the following directory structure on the disk:

- train/
  - class-A/
  - class-B/
  - ...
- val/
  - class-A/
  - class-B/
  - ...
- test/
  - class-A/
  - class-B/
  - ...

where class-A and class-B are subdirectories containing the data for each object class that you have defined in a class label file.

Once all data is collected, within the Docker container in the termial window, run the following command:

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

This will take you to the classification folder, in which you can start training your custom model. To do so, run the following:

```
$ python3 train.py --model-dir=models/tools --batch-size=4 --workers=1 data/tools
```

Depending on the size of the dataset, training can take a while. Once the model is trained, the next step is to convert it into ONNX formate so that it can be used by TensorRT for inference. To export the model into ONNX, run the following:

```
$ python3 onnx_export.py --model-dir=models/tools
```

Once this step is done, we can run inference in real time using the following command:

```
$ imagenet --model=models/tools/resnet18.onnx --labels=data/tools/labels.txt \
--input_blob=input_0 --output_blob=output_0 csi://0
```

This command is pretty long so pay attention to spelling and spaces. The \ sign indicates a new line but the command can be written in one line.

5 Object Detection

5.1 Inference With Pre-Trained Models

Just like with object classification, we have downloaded pre-trained models for object detection. Specifically, the network that is used in this case is SSD-Mobilenet. SSD-Mobilenet is a popular network architecture for real-time object detection on mobile and embedded devices. To perform object detection with a pre-trained model, run the following commands:

```
$ cd jetson-inference
$ docker/run.sh
$ detectnet.py csi://0
```

Feel free to experiment with whatever objects you have at hand and see what objects the model can recognize. Full list of classes that are recognized can be found here.

5.2 Inference With Custom Models

Now, that we have played around with the pre-trained model, let us create our own. Like with object classification, we will train our own model with a custom data set and test it using the Raspberry Pi camera module. To start collecting data, do the following:

1. Open up the terminal and navigate yourself to jetson-inference/
2. Run the following commands:

   ```
   $ cd jetson-inference
   $ docker/run.sh
   $ camera-capture csi://0
   ```

3. Open the file explorer, and go to jetson-inference → python → training → detection → ssd → data.

4. Create a new folder. I will name mine **tools**. Within the **tools** folder, create **labels.txt** like you did in the classification training. Populate **labels.txt** with classes in alphabetical order. For example, my **labels.txt** looks like the following:

   drill

   hammer

   screwdriver

5. In the camera capture window, choose Detection in the Dataset Type, select Dataset path to data/tools, and select the labels.txt file for Class Labels.

At this point we are ready start collecting data. Place an object in front of the camera. For data collection, make sure "Save on Unfreeze" is checked, "Clear on Unfreeze" is unchecked, and "Merge Sets" is checked. Whenever ready, click the "Freeze/Edit" button. The live camera will become frozen, and you will be able to draw a bounding box around the object. You can select the appropriate object class for each bounding box in the grid table in the control view. When you are done labeling the image, click the depressed "Freeze/Edit" button again to save data and unfreeze the camera view for the next image. It is important to collect images from varying orientations, camera viewpoints, and lighting conditions to make the model as robust as possible. If you find that your model is not performing as well as you would like, try adding more training data and playing around with conditions. I will collect 100 images for each class, so 300 images total. Once all images are taken, we can close the camera capture window.

To train the model, run the following commands:

```
$ python3 train_ssd.py --dataset-type=voc --data=data/tools \
--model-dir=models/tools --batch-size=2 --workers=1
```

Training the model will take some time. Once it is complete, we will export the model into ONNX format by running the following command:

```
$ python3 onnx_export.py --model-dir=models/tools
```

Once the model is converted into ONNX, run the following command to perform real-time inference:

```
$ detectnet --model=models/tools/ssd-mobilenet.onnx \
--labels=models/tools/labels.txt --input_blob=input_0 --output-cvg=scores \
--output-bbox=boxes csi://0
```

6 Appendix

Below are some useful commands and additional code:

1. imagenet.py for object classification can be found here.
2. detectnet.py for object detection can be found here.
3. To check for all video devices connected to the Jetson Nano, run the following command:

```
$ ls /dev/video*
```

7 Bill Of Materials

Listed below are all the necessary items for this project:

1. NVIDIA Jetson Nano 2GB Developer Kit (purchase link)
2. Raspberry Pi v.2 Camera Module (purchase link)
3. CanaKit Raspberry Pi USB-C Power Supply (purchase link)
4. Samsung 32GB microSD (purchase link)
5. (Optional) 5 V Noctua DC fan (purchase link)

In case you decide to purchase a DC fan for extra cooling, instructions for enabling it can be found on the Github page here.

8 References

1. *Hello AI World*, NVIDIA Corporation, access link.

2. *Image segmentation, Object detection and Optimizing efficiency with Custom datasets*, Dhruva Krishna, access link.