Capstone Project

Edge computing device programming for AI projects

Lecture 3

Dr. Wilton Fok &

Carol Chen

Recap: Training classification model

	Operation	Results
Dataset preparation	Preparing the catdog classification dataset	Get a prepared dataset
Train model	*$ cd ~/jetson-inference/python/training/classification* *$ python3 train.py data/cat_dog --model-dir=models/cat_dog --batch-size=1 --workers=1 --epochs=1*	Obtaining a .pth model
Convert model	*$ python3 onnx_export.py --model-dir=models/cat_dog*	Obtaining a .onnx model
Run model	*/home/nvidia/jetson-inference/build/aarch64/bin/imagenet --model= models/resnet18.onnx --input_blob=input_0 --output_blob=output_0 --labels data/cat_dog/labels.txt data/cat_dog/test/cat*	Visualize the output of the model

Training with your own dataset

A group of dataset

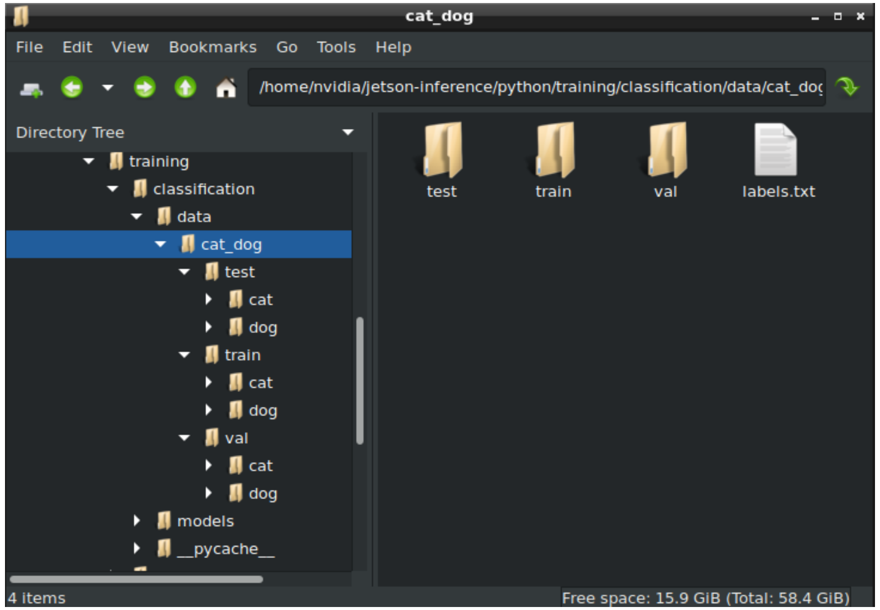
Two important parameter should be defined

- Where to store your data (***YourDataPath***)
- Where to store your model (***YourModelPath***)

Following the similar steps to finish model training, conversion and inference.

Step1: Dataset preparation

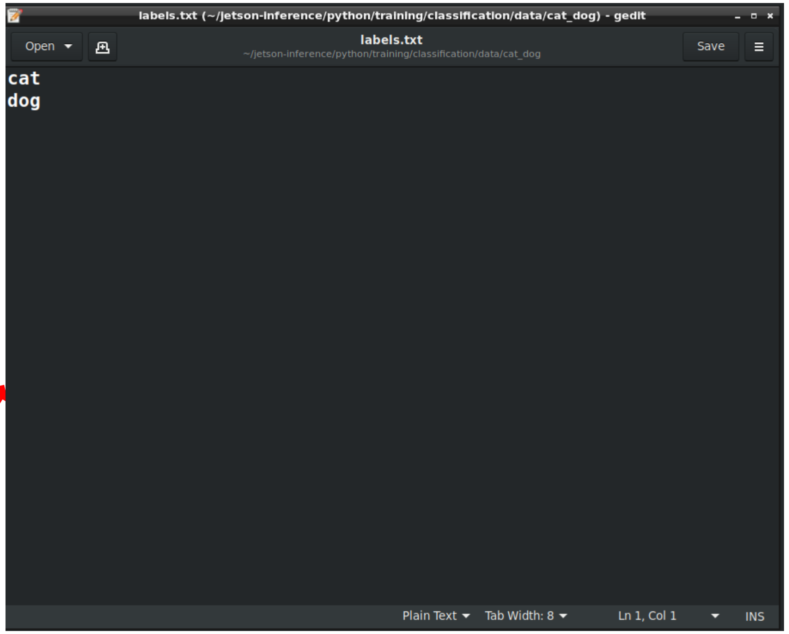
- Now you are familiar with how to train classification model.
- Before training, we should collect the dataset, and organize them similar to the catdog dataset.



/home/nvidia/jetson-inference/python/training/classification/data/cat_do(

```
training
  classification
    data
      cat_dog
        test
          cat
          dog
        train
          cat
          dog
        val
          cat
          dog
    models
    __pycache__
```

```
--data/catdog
------train
-----------cat
-----------dog
------test
-----------cat
-----------dog
------validation
-----------cat
-----------dog
------labels.txt
```



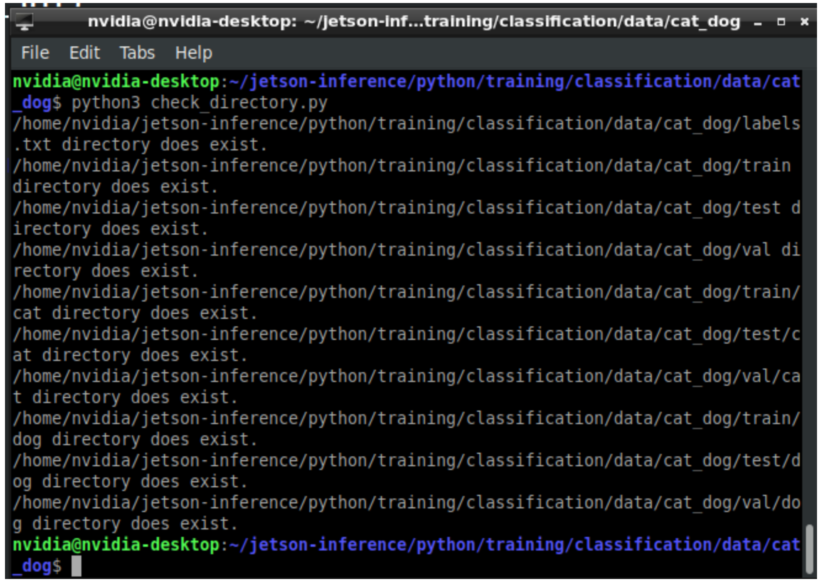
labels.txt (~/jetson-inference/python/training/classification/data/cat_dog) - gedit

```
cat
dog
```

Open
labels.txt
~/jetson-inference/python/training/classification/data/cat_dog
Save
Plain Text
Tab Width: 8
Ln 1, Col 1
INS

Step1: Dataset preparation

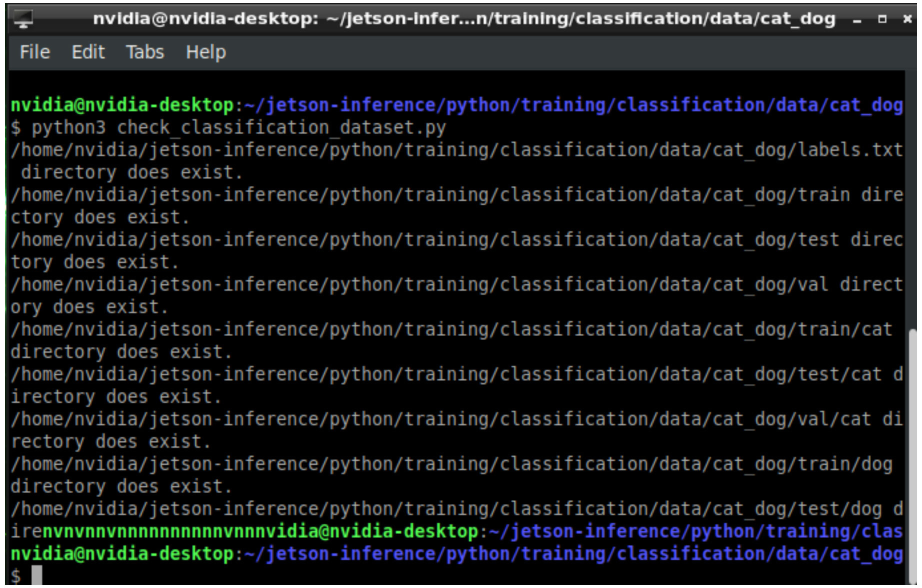
- Now, you can download the images you want to classify from google.
- Remember the image among train, validation and test is around 7:2:1
- After finishing the dataset, run “check_dataset.py” to verify whether the dataset structure is valid.



nvidia@nvidia-desktop: ~/jetson-inf...training/classification/data/cat_dog
File Edit Tabs Help

```
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification/data/cat
_dog$ python3 check_directory.py
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/labels
.txt directory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/train
directory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/test d
irectory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/val di
rectory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/train/
cat directory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/test/c
at directory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/val/ca
t directory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/train/
dog directory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/test/d
og directory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/val/do
g directory does exist.
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification/data/cat
_dog$ 
```

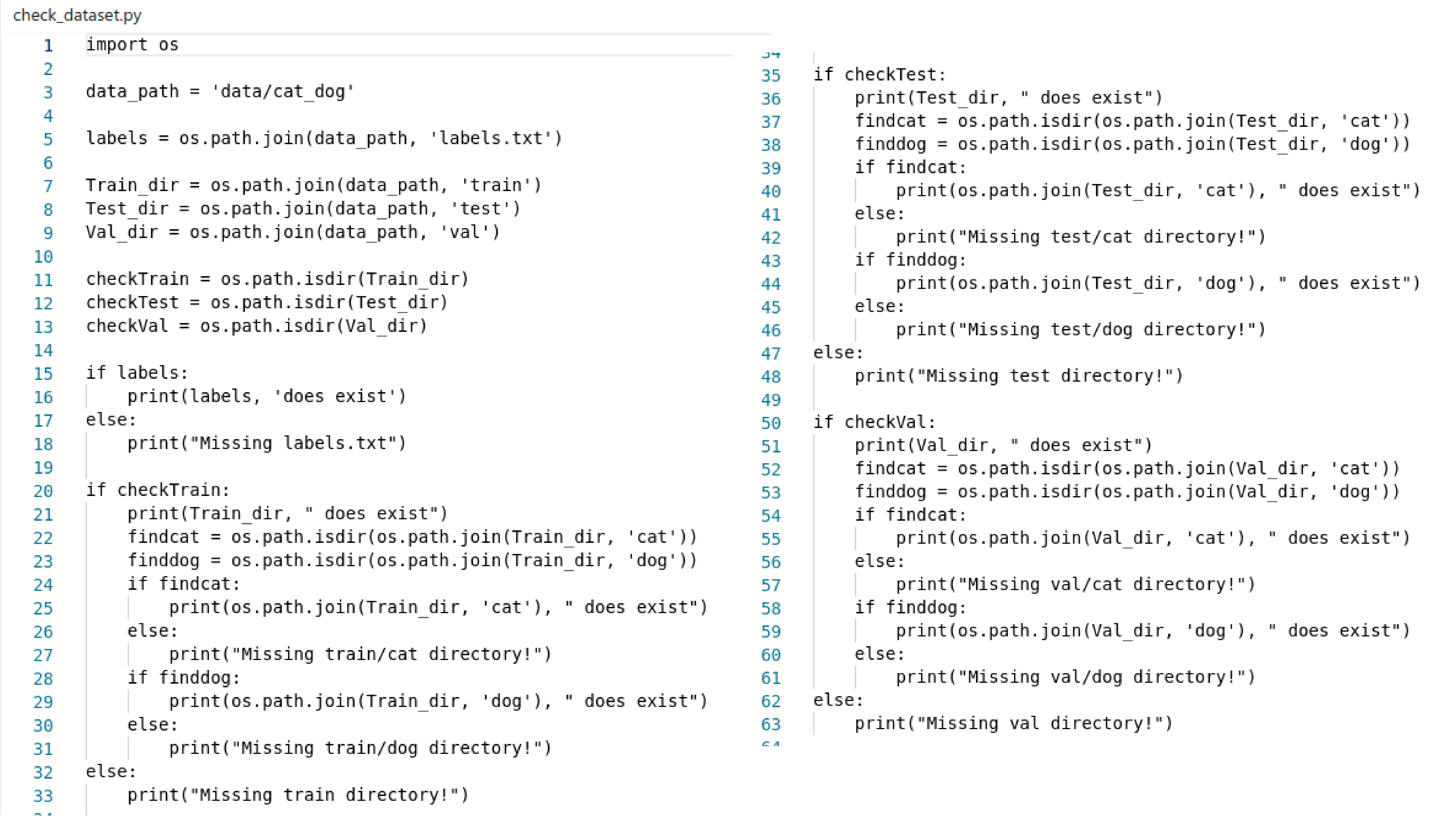
Correct directory



nvidia@nvidia-desktop: ~/jetson-infer...n/training/classification/data/cat_dog
File Edit Tabs Help

```
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification/data/cat_dog
$ python3 check_classification_dataset.py
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/labels.txt
 directory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/train dire
ctory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/test direc
tory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/val direct
ory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/train/cat
directory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/test/cat d
irectory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/val/cat di
rectory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/train/dog
directory does exist.
/home/nvidia/jetson-inference/python/training/classification/data/cat_dog/test/dog d
irenvnvnnvnnnnnnnnnnvnnnvidia@nvidia-desktop:~/jetson-inference/python/training/clas
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification/data/cat_dog
$ 
```

Missing "val/cat"



check_dataset.py

```python
import os

data_path = 'data/cat_dog'

labels = os.path.join(data_path, 'labels.txt')

Train_dir = os.path.join(data_path, 'train')
Test_dir = os.path.join(data_path, 'test')
Val_dir = os.path.join(data_path, 'val')

checkTrain = os.path.isdir(Train_dir)
checkTest = os.path.isdir(Test_dir)
checkVal = os.path.isdir(Val_dir)

if labels:
    print(labels, 'does exist')
else:
    print("Missing labels.txt")

if checkTrain:
    print(Train_dir, " does exist")
    findcat = os.path.isdir(os.path.join(Train_dir, 'cat'))
    finddog = os.path.isdir(os.path.join(Train_dir, 'dog'))
    if findcat:
        print(os.path.join(Train_dir, 'cat'), " does exist")
    else:
        print("Missing train/cat directory!")
    if finddog:
        print(os.path.join(Train_dir, 'dog'), " does exist")
    else:
        print("Missing train/dog directory!")
else:
    print("Missing train directory!")

if checkTest:
    print(Test_dir, " does exist")
    findcat = os.path.isdir(os.path.join(Test_dir, 'cat'))
    finddog = os.path.isdir(os.path.join(Test_dir, 'dog'))
    if findcat:
        print(os.path.join(Test_dir, 'cat'), " does exist")
    else:
        print("Missing test/cat directory!")
    if finddog:
        print(os.path.join(Test_dir, 'dog'), " does exist")
    else:
        print("Missing test/dog directory!")
else:
    print("Missing test directory!")

if checkVal:
    print(Val_dir, " does exist")
    findcat = os.path.isdir(os.path.join(Val_dir, 'cat'))
    finddog = os.path.isdir(os.path.join(Val_dir, 'dog'))
    if findcat:
        print(os.path.join(Val_dir, 'cat'), " does exist")
    else:
        print("Missing val/cat directory!")
    if finddog:
        print(os.path.join(Val_dir, 'dog'), " does exist")
    else:
        print("Missing val/dog directory!")
else:
    print("Missing val directory!")
```

Step 2: Training

- After getting the dataset, similarly, we need to open a terminal, and:

```
$ cd ~/jetson-inference/python/training/classification
$  python3 train.py YourDataPath --model-dir=YourModelPath --batch-size=1 --workers=1 --epochs=10
```

Then the model will begin to train.

*Also, Please keep the path of **YourModelPath** and **YourDataPath** in mind! They are necessary in the following*

Some concepts of commands and AI

During this time, I will introduce what’s the meaning of the commands entered.

Remember: Terminal is a controller communicating with your computer!

Commands translation

(1)

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

Translation: Entering the base location, because our AI files are written and we will begin here.

(2)

```
python3 train.py data/cat_dog --model-dir=models/cat_dog  --batch-size=1 --workers=1 --epochs=1
```

Translation: Train a model. We use the data stored in data/cat_dog to train and the model will be stored in models/cat_dog. The batch size, workers and epochs are set to be 1.

(3)

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

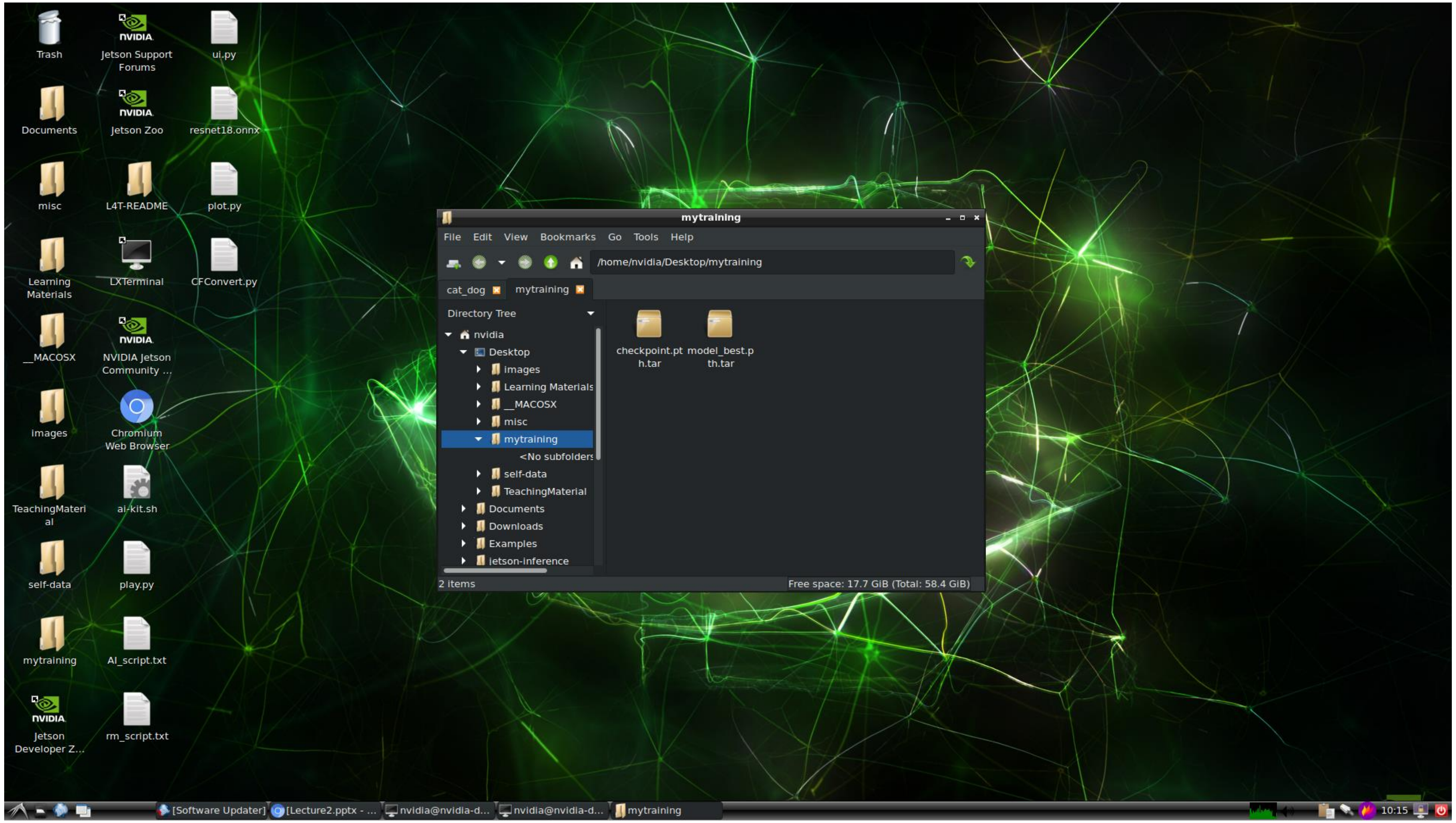
Translation: We convert the model in models/cat_dog to onnx.

(4)

```
/home/nvidia/jetson-inference/build/aarch64/bin/imagenet --model=models/resnet18.onnx --input_blob=input_0 --output_blob=output_0 --labels data/cat_dog/labels.txt data/cat_dog/test/cat
```

Translation: We use the "…/imagenet" to execute running images, with loading the model

- Training with your own data

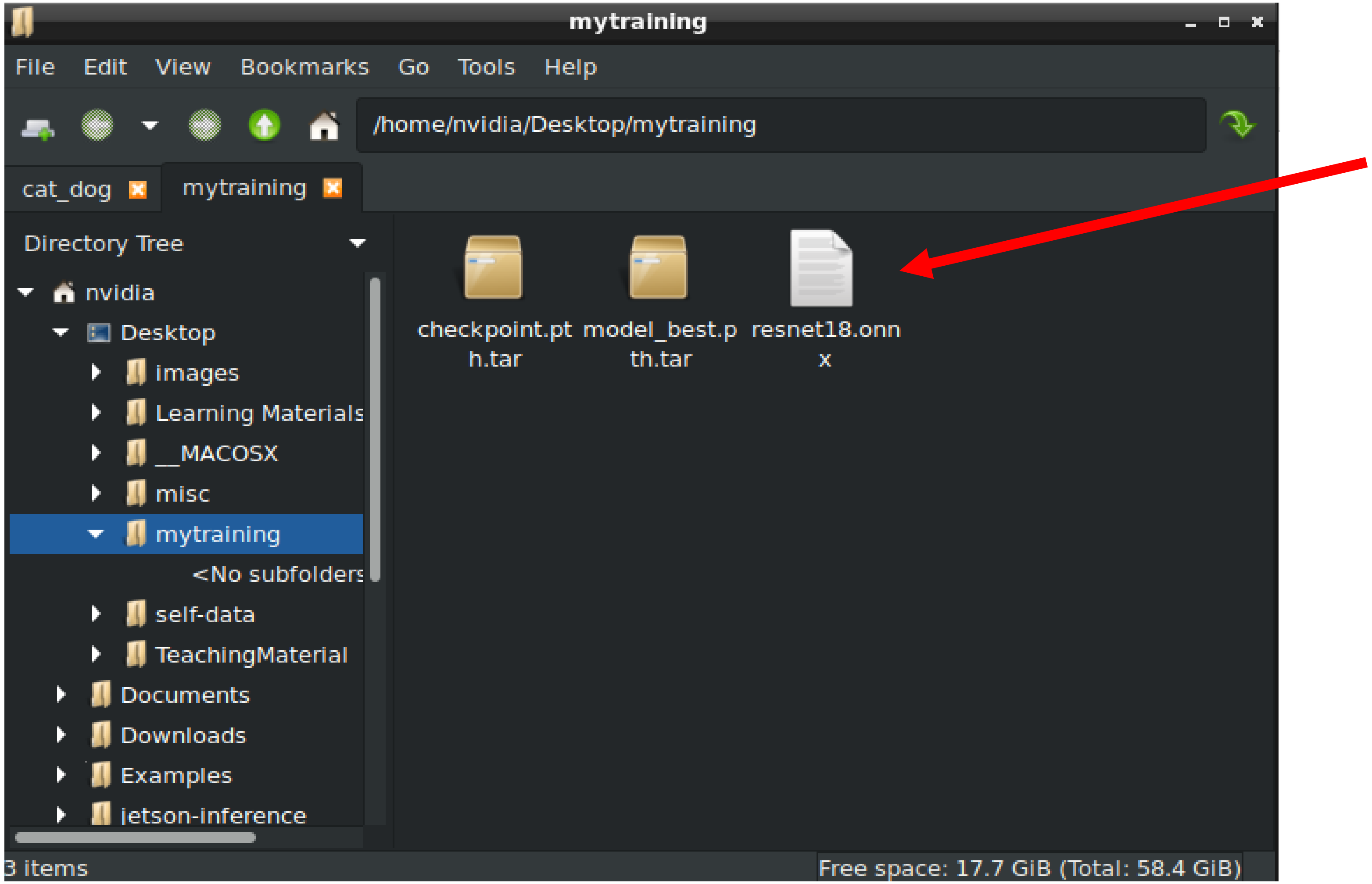


After finishing your training, you would find this two files in "mytraining"

- Converting the Model to ONNX

This will create a model called resnet18.onnx under:

YourModelPath



mytraining

/home/nvidia/Desktop/mytraining

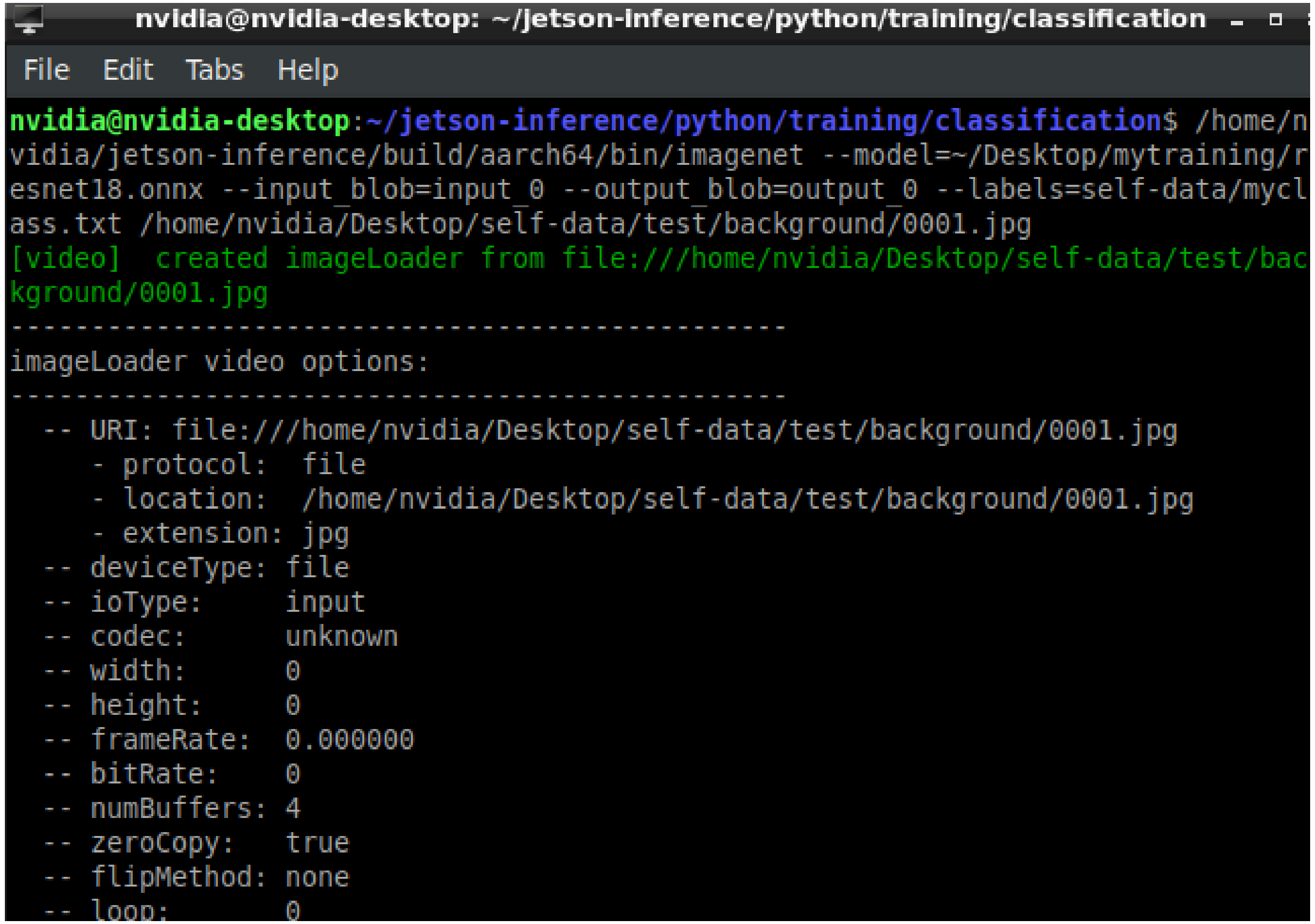
checkpoint.pth.tar
model_best.pth.tar
resnet18.onnx

3 items

Free space: 17.7 GiB (Total: 58.4 GiB)

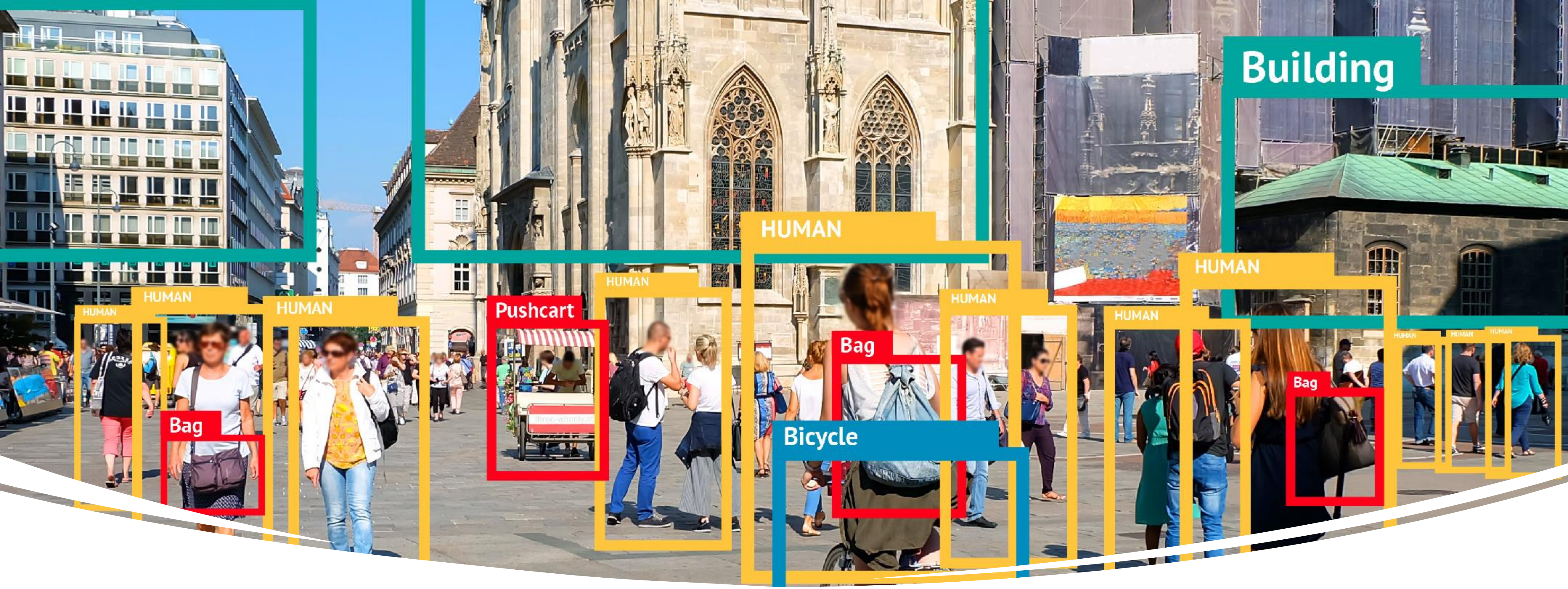
- Processing Images with TensorRT

  *$ /home/nvidia/jetson-inference/build/aarch64/bin/imagenet --model=**YourModelPath**/resnet18.onnx --input_blob=input_0 --output_blob=output_0 --labels=self-data/class.txt **yourOwenPicturePath***


nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification
File
Edit
Tabs
Help

```
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ /home/n
vidia/jetson-inference/build/aarch64/bin/imagenet --model=~/Desktop/mytraining/r
esnet18.onnx --input_blob=input_0 --output_blob=output_0 --labels=self-data/mycl
ass.txt /home/nvidia/Desktop/self-data/test/background/0001.jpg
[video]  created imageLoader from file:///home/nvidia/Desktop/self-data/test/bac
kground/0001.jpg
------------------------------------------------
imageLoader video options:
------------------------------------------------
  -- URI: file:///home/nvidia/Desktop/self-data/test/background/0001.jpg
     - protocol:  file
     - location:  /home/nvidia/Desktop/self-data/test/background/0001.jpg
     - extension: jpg
  -- deviceType: file
  -- ioType:     input
  -- codec:      unknown
  -- width:      0
  -- height:     0
  -- frameRate:  0.000000
  -- bitRate:    0
  -- numBuffers: 4
  -- zeroCopy:   true
  -- flipMethod: none
  -- loop:       0
```

Task2: Object detection training

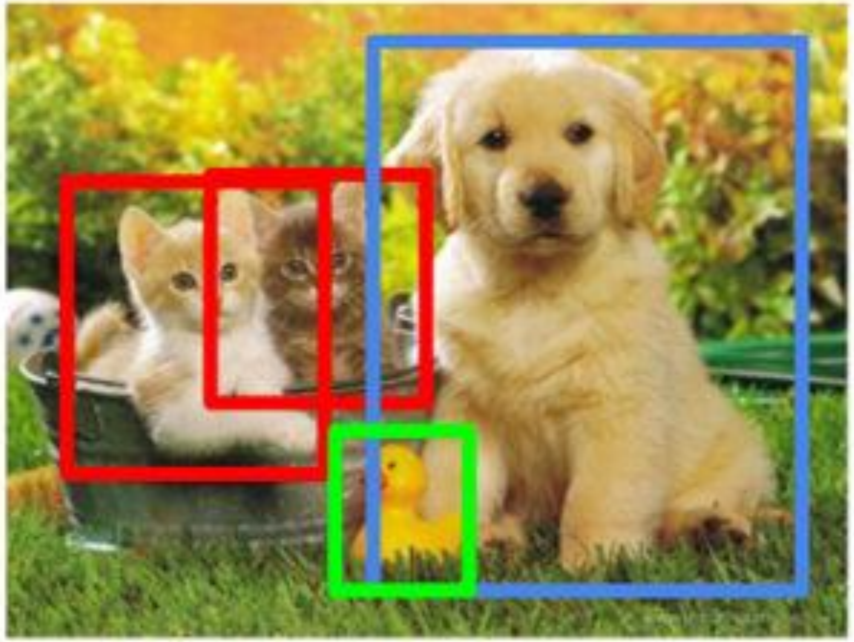


Object Detection is a computer vision task in which the goal is to detect and locate objects of interest in an image or video. The task involves identifying the position and boundaries of objects in an image, and classifying the objects into different categories.

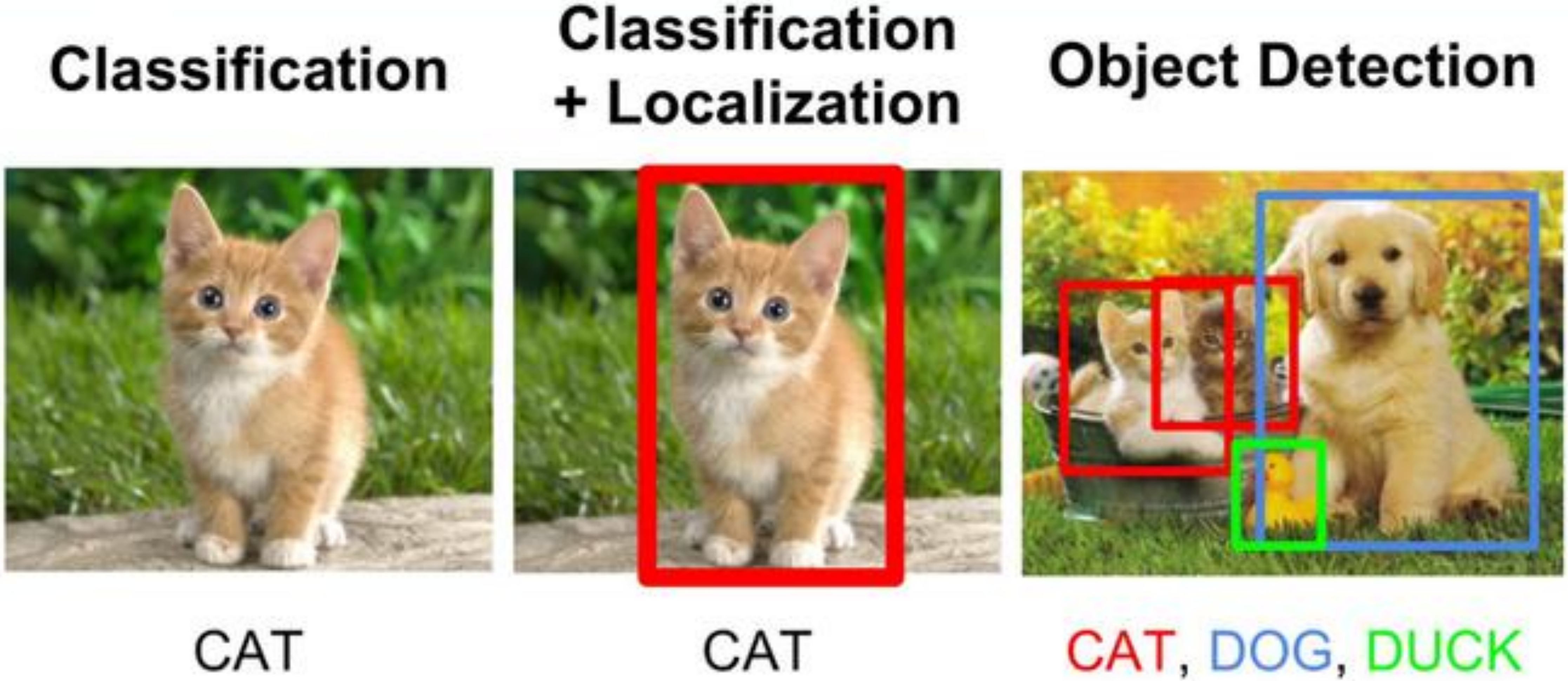
Classification

CAT

Object Detection

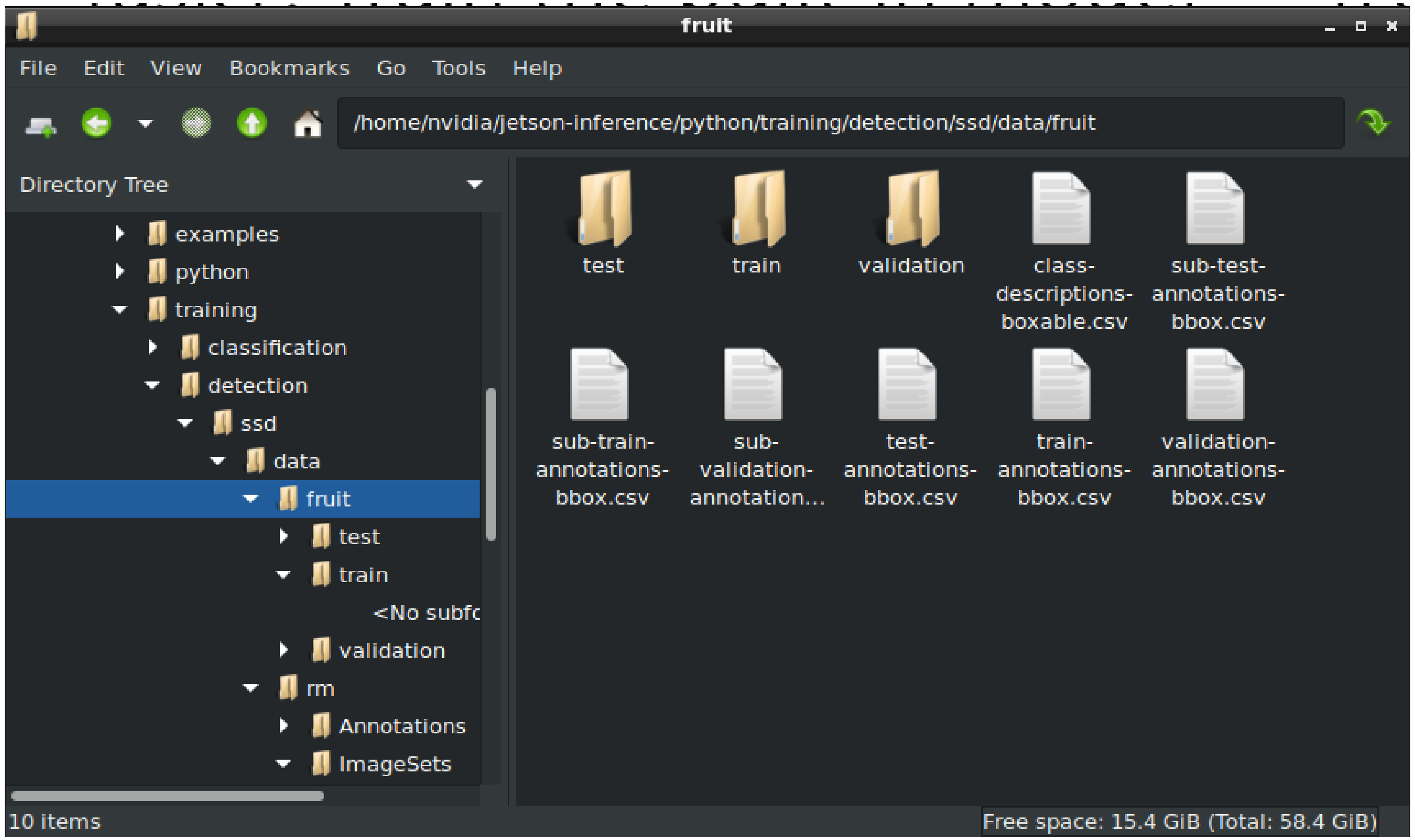


CAT, DOG, DUCK


Classification
Classification + Localization
Object Detection
CAT
CAT
CAT, DOG, DUCK

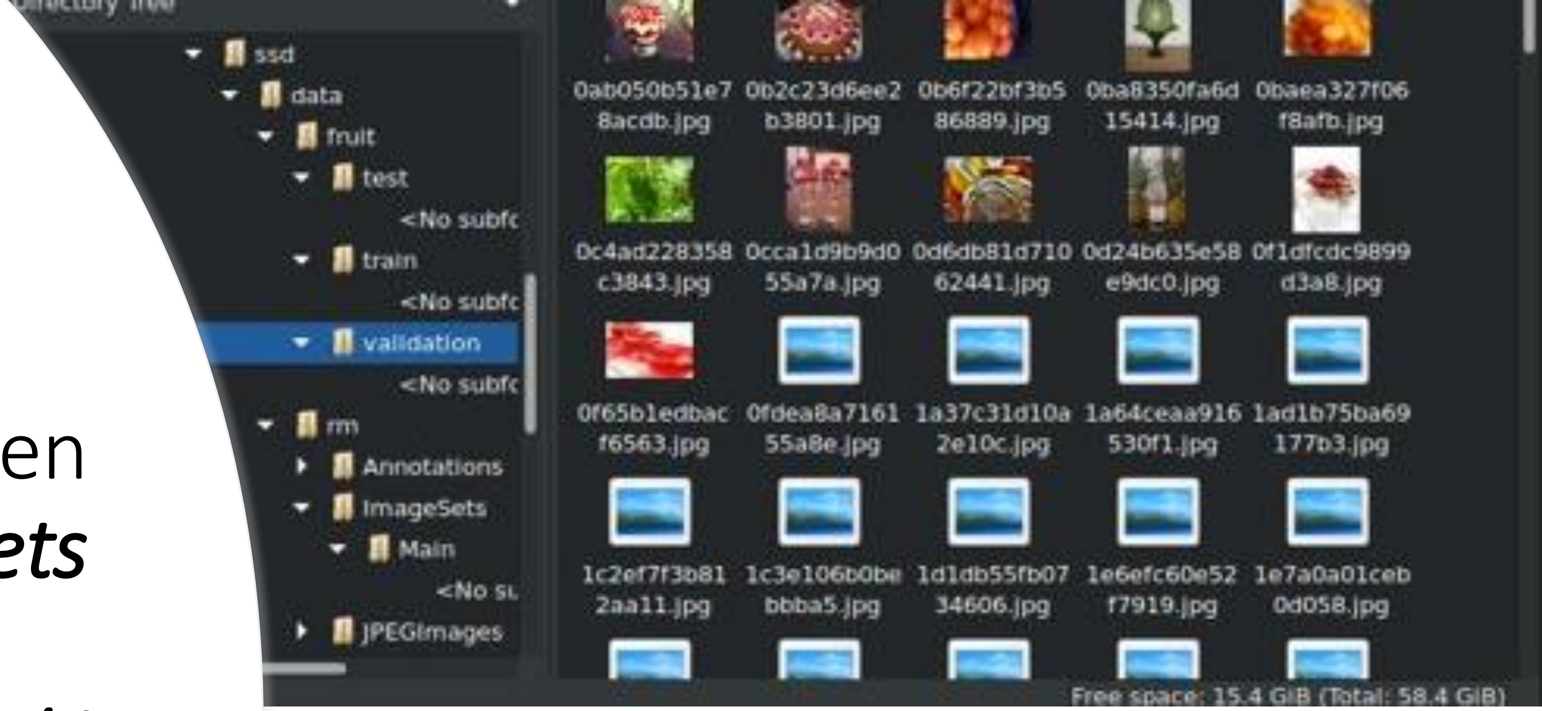
Train the built-in model --- fruit

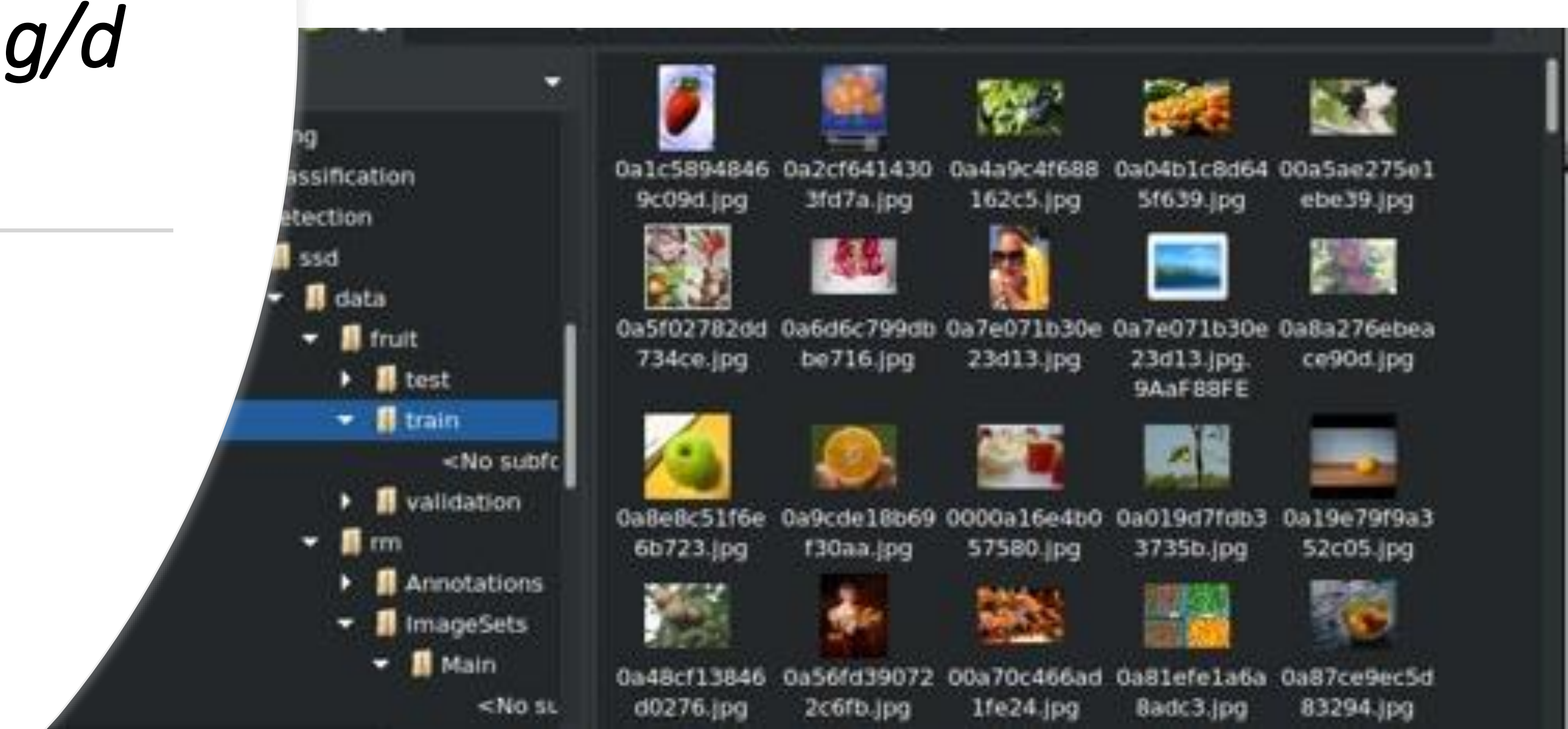
- Mission: Train a fruit recognition model using these images, and use it to detection different kinds of fruit.



fruit
File Edit View Bookmarks Go Tools Help
/home/nvidia/jetson-inference/python/training/detection/ssd/data/fruit
Directory Tree
examples
python
training
classification
detection
ssd
data
fruit
test
train
<No subfc
validation
rm
Annotations
ImageSets
test
train
validation
class-descriptions-boxable.csv
sub-test-annotations-bbox.csv
sub-train-annotations-bbox.csv
sub-validation-annotation...
test-annotations-bbox.csv
train-annotations-bbox.csv
validation-annotations-bbox.csv
10 items
Free space: 15.4 GiB (Total: 58.4 GiB)

The fruit dataset have been stored in */home/nvidia/jetson-inference/python/training/detection/ssd/data* path.





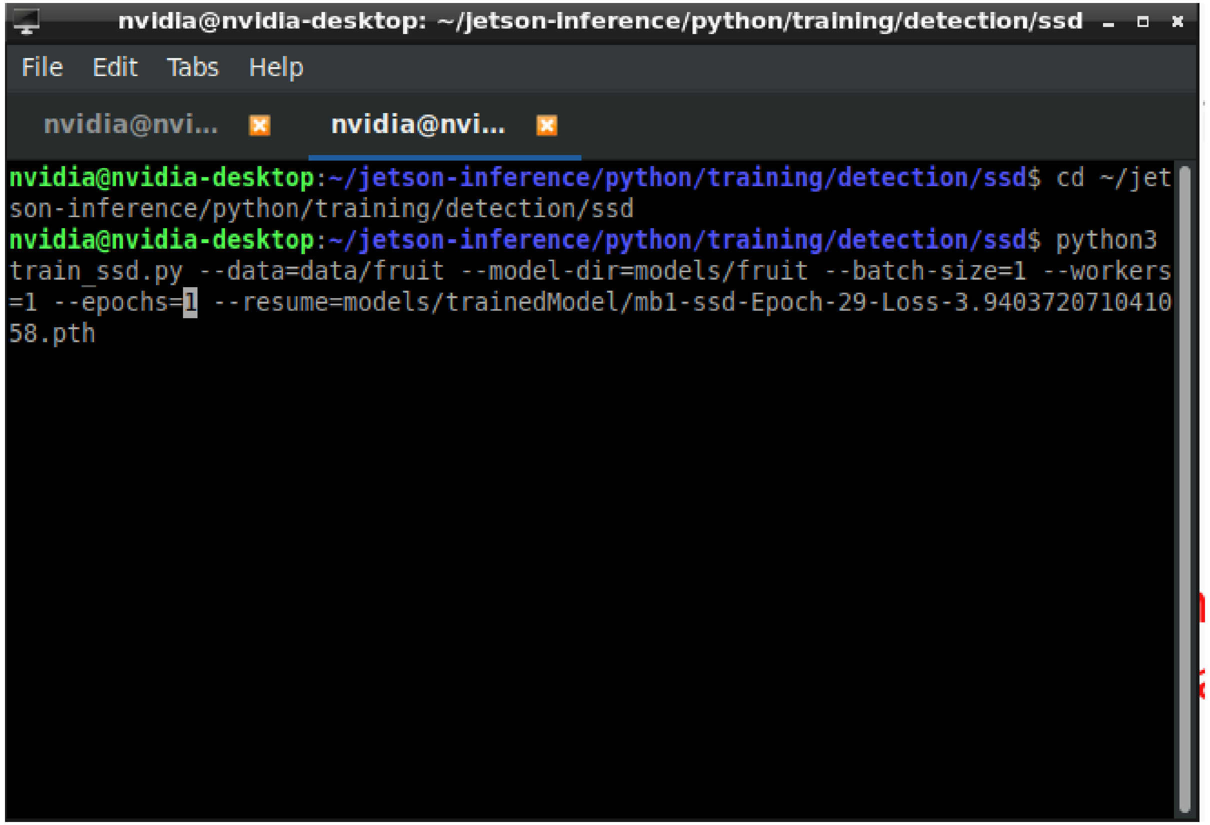
Step 1: Train model

- To train the model, we need firstly download the pretrained model [mb1-ssd-Epoch-29-Loss-3.940372071041058.pth]

  and put it in models/trainedModel
- Download the dataset [fruit] and put it in data/fruitt

- then open a terminal:

```
$ cd ~/jetson-inference/python/training/detection/ssd
$ python3 train_ssd.py --data=data/fruit --model-dir=models/fruit --batch-size=1 --workers=1 --epochs=1 --resume=models/trainedModel/mb1-ssd-Epoch29-Loss-3.940372071041058.pth
```



nvidia@nvidia-desktop: ~/jetson-inference/python/training/detection/ssd
File
Edit
Tabs
Help
nvidia@nvi...
nvidia@nvi...

```
nvidia@nvidia-desktop:~/jetson-inference/python/training/detection/ssd$ cd ~/jet
son-inference/python/training/detection/ssd
nvidia@nvidia-desktop:~/jetson-inference/python/training/detection/ssd$ python3
train_ssd.py --data=data/fruit --model-dir=models/fruit --batch-size=1 --workers
=1 --epochs=1 --resume=models/trainedModel/mb1-ssd-Epoch-29-Loss-3.9403720710410
58.pth
```

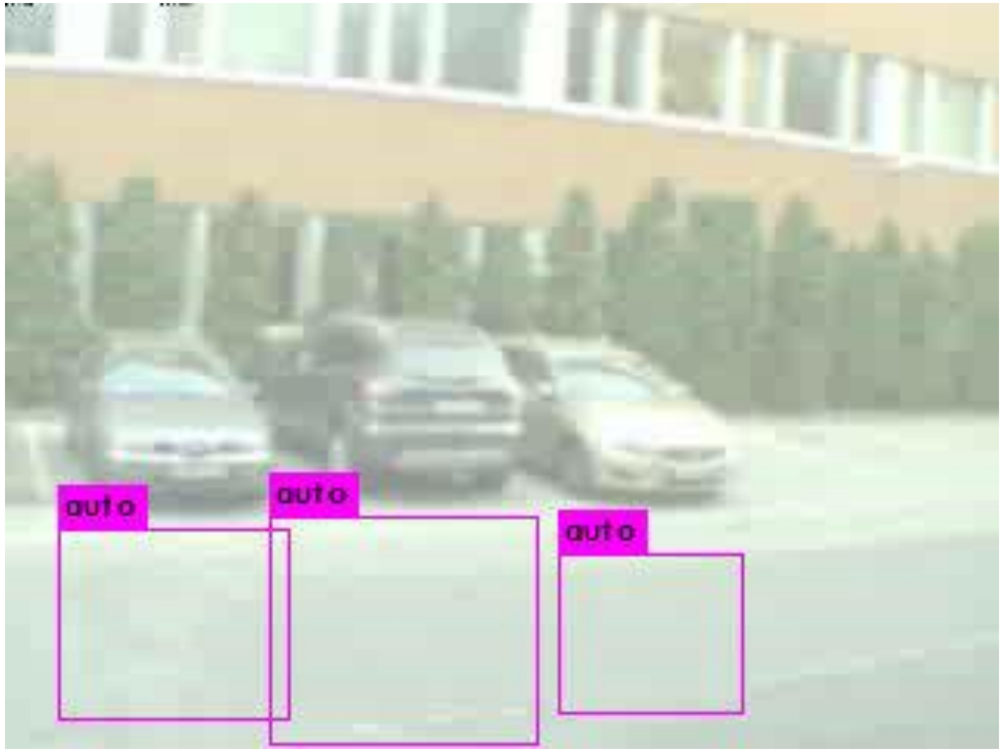
Step 1: Train model

- During training, the terminal will show lots of information.

```
2020-07-10 13:14:12 - Epoch: 0, Step: 10/1287, Avg Loss: 12.4240, Avg Regression Loss 3.5747, Avg Classification Loss: 8.8493
2020-07-10 13:14:12 - Epoch: 0, Step: 20/1287, Avg Loss: 9.6947, Avg Regression Loss 4.1911, Avg Classification Loss: 5.5036
2020-07-10 13:14:13 - Epoch: 0, Step: 30/1287, Avg Loss: 8.7409, Avg Regression Loss 3.4078, Avg Classification Loss: 5.3332
2020-07-10 13:14:13 - Epoch: 0, Step: 40/1287, Avg Loss: 7.3736, Avg Regression Loss 2.5356, Avg Classification Loss: 4.8379
2020-07-10 13:14:14 - Epoch: 0, Step: 50/1287, Avg Loss: 6.3461, Avg Regression Loss 2.2286, Avg Classification Loss: 4.1175
…
2020-07-10 13:19:26 - Epoch: 0, Validation Loss: 5.6730, Validation Regression Loss 1.7096, Validation Classification Loss: 3.9634
2020-07-10 13:19:26 - Saved model models/fruit/mb1-ssd-Epoch-0-Loss-5.672993580500285.pth
```

Avg Regression Loss:

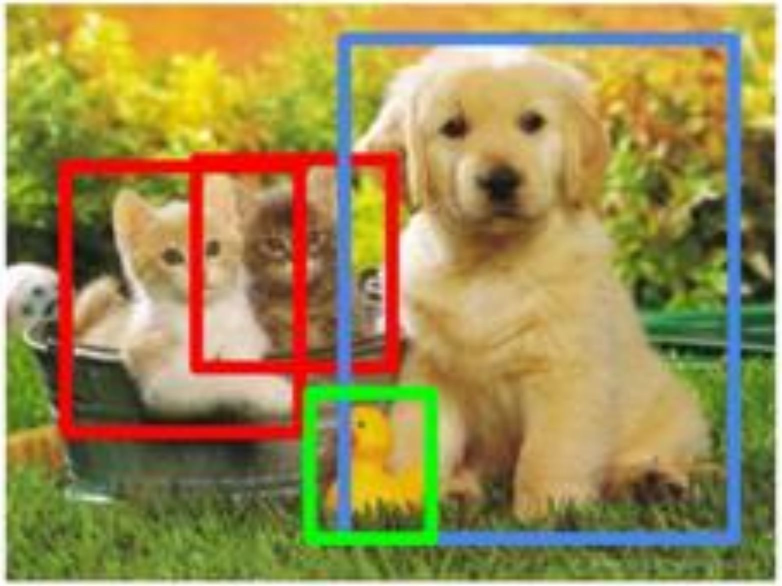
- Avg Regression Loss is the average error of the bounding box locations.



Avg Classification Loss

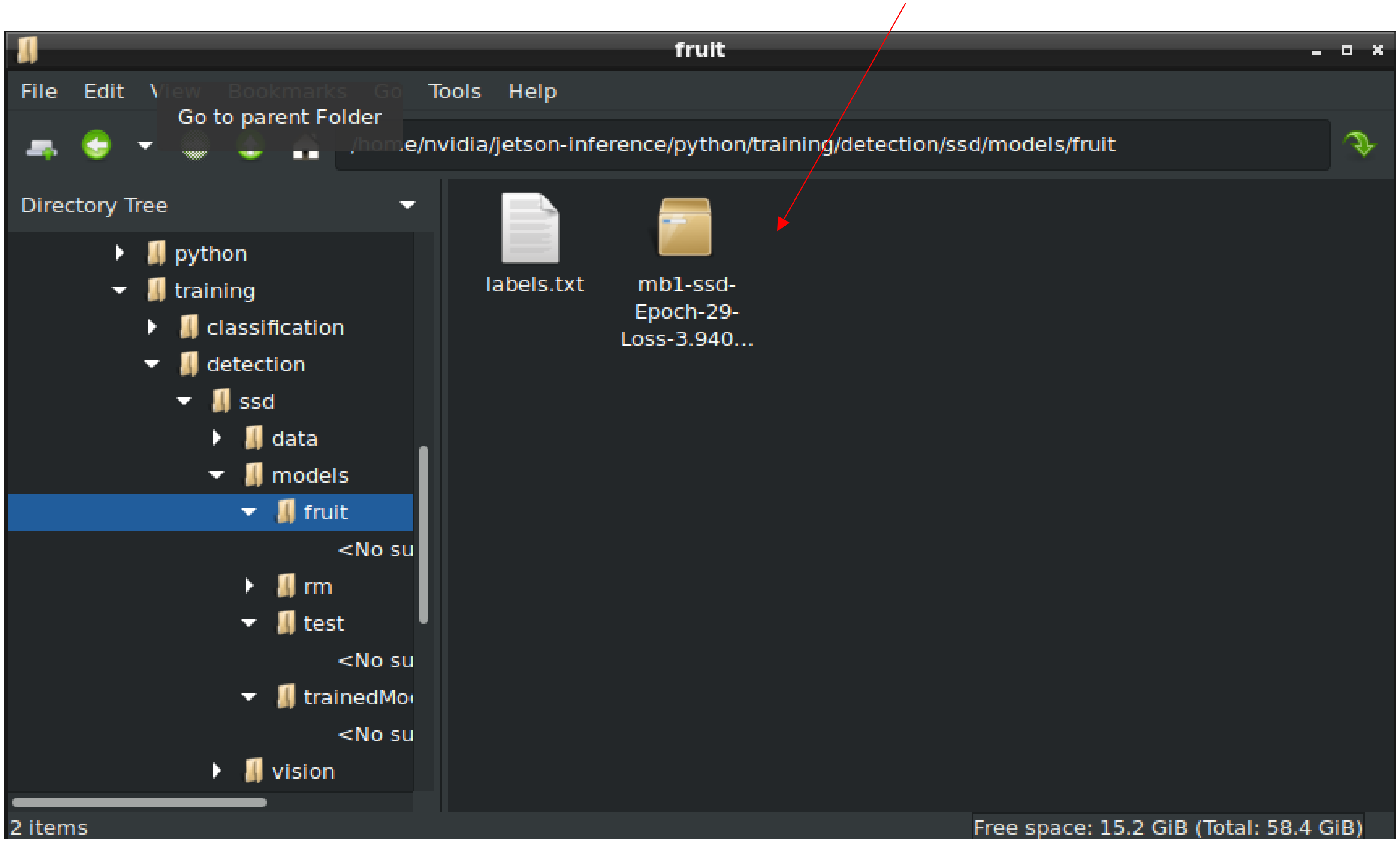
- **Avg Classification Loss: Avg Classification Loss is the average error of the object classifications.**

- **E.g. Dog Cat Duck (0% loss)**



Step 2: Convert model

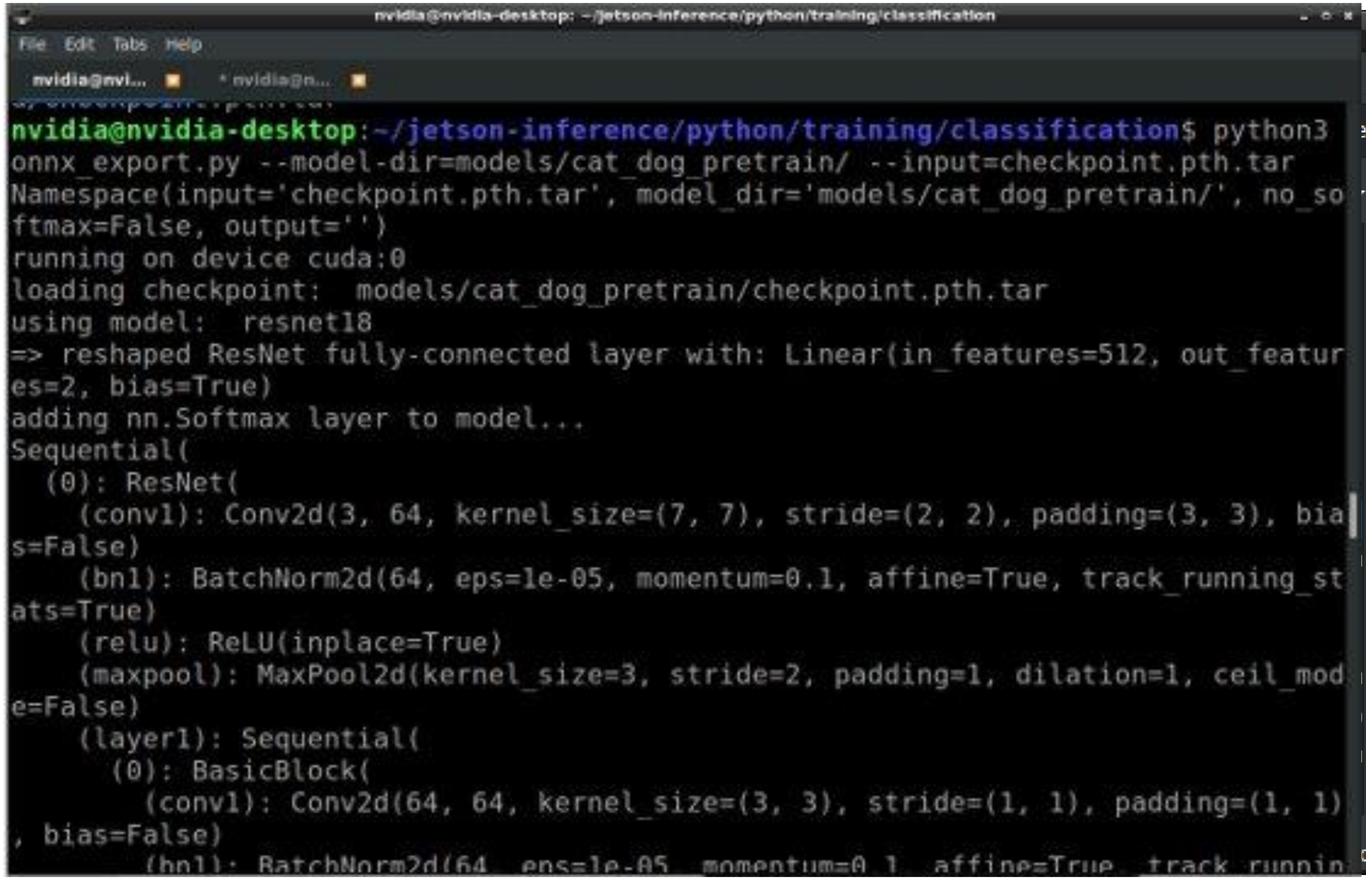
After training, the model folder will be:



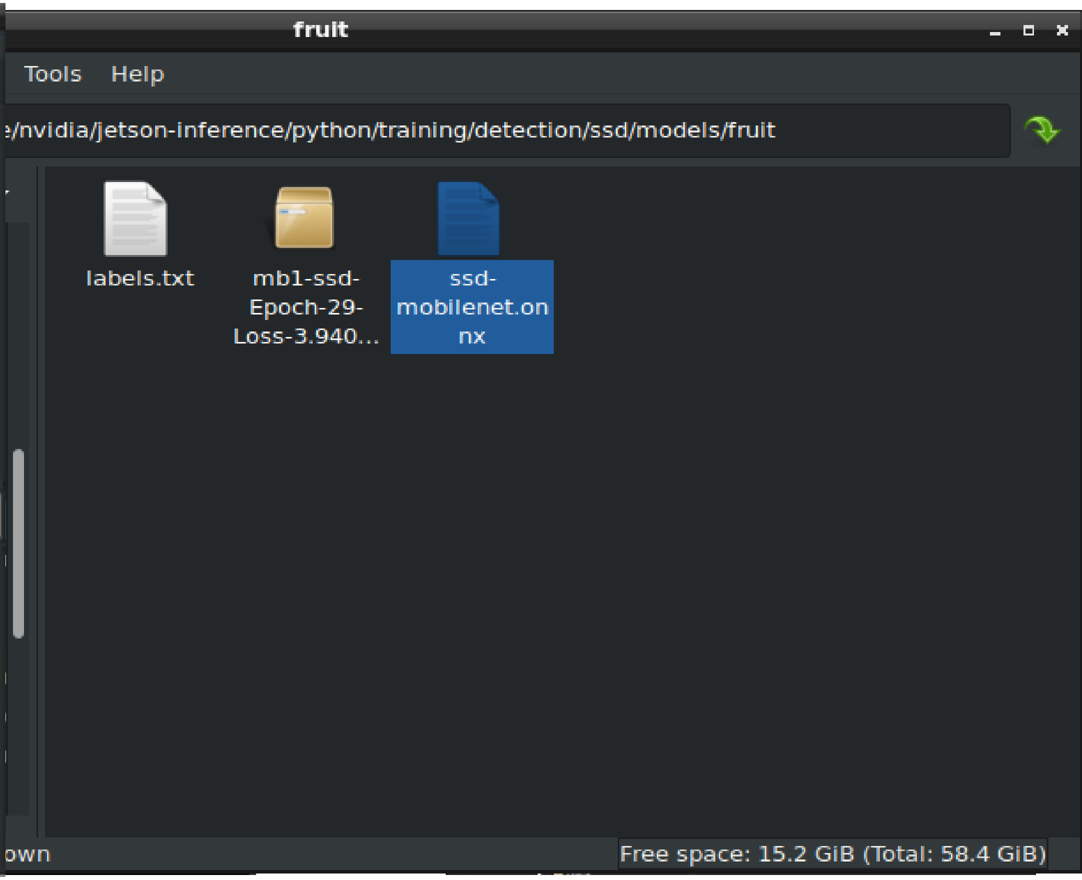
Then, we need to convert the model using:

$ python3 onnx_export.py --input=models/fruit/mb1-ssd-Epoch-29-Loss-3.94037207104 1058.pth --model-dir=models/fruit/

When you find a file with .onnx in the folder, it succeeds!


nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification
File Edit Tabs Help
nvidia@nvi...
* nvidia@n...

```
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ python3
onnx_export.py --model-dir=models/cat_dog_pretrain/ --input=checkpoint.pth.tar
Namespace(input='checkpoint.pth.tar', model_dir='models/cat_dog_pretrain/', no_so
ftmax=False, output='')
running on device cuda:0
loading checkpoint:  models/cat_dog_pretrain/checkpoint.pth.tar
using model:  resnet18
=> reshaped ResNet fully-connected layer with: Linear(in_features=512, out_featur
es=2, bias=True)
adding nn.Softmax layer to model...
Sequential(
  (0): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bia
s=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
e=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
, bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runnin
```



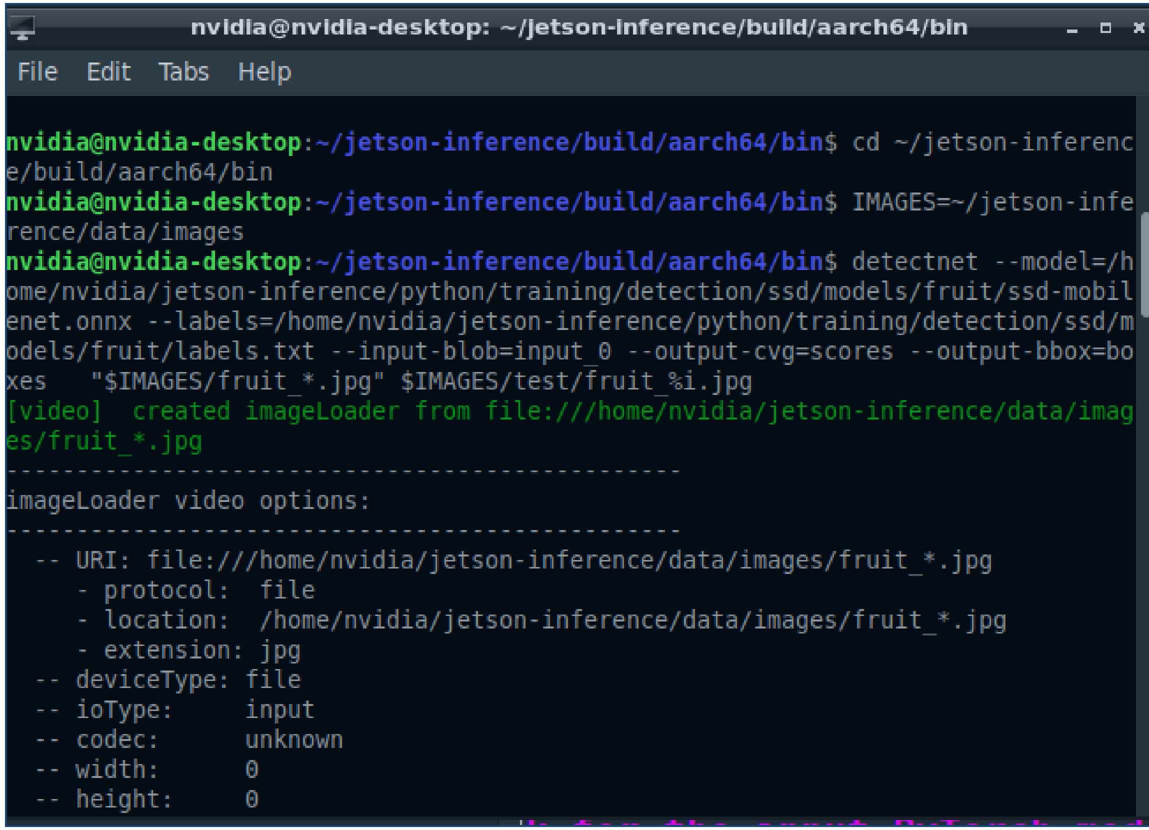
e/nvidia/jetson-inference/python/training/detection/ssd/models/fruit

Free space: 15.2 GiB (Total: 58.4 GiB)

Step 3: Inference model with image

- Now you get a model runnable! In the next step, we need to use it to make prediction on different images.
- Take fruit for example:

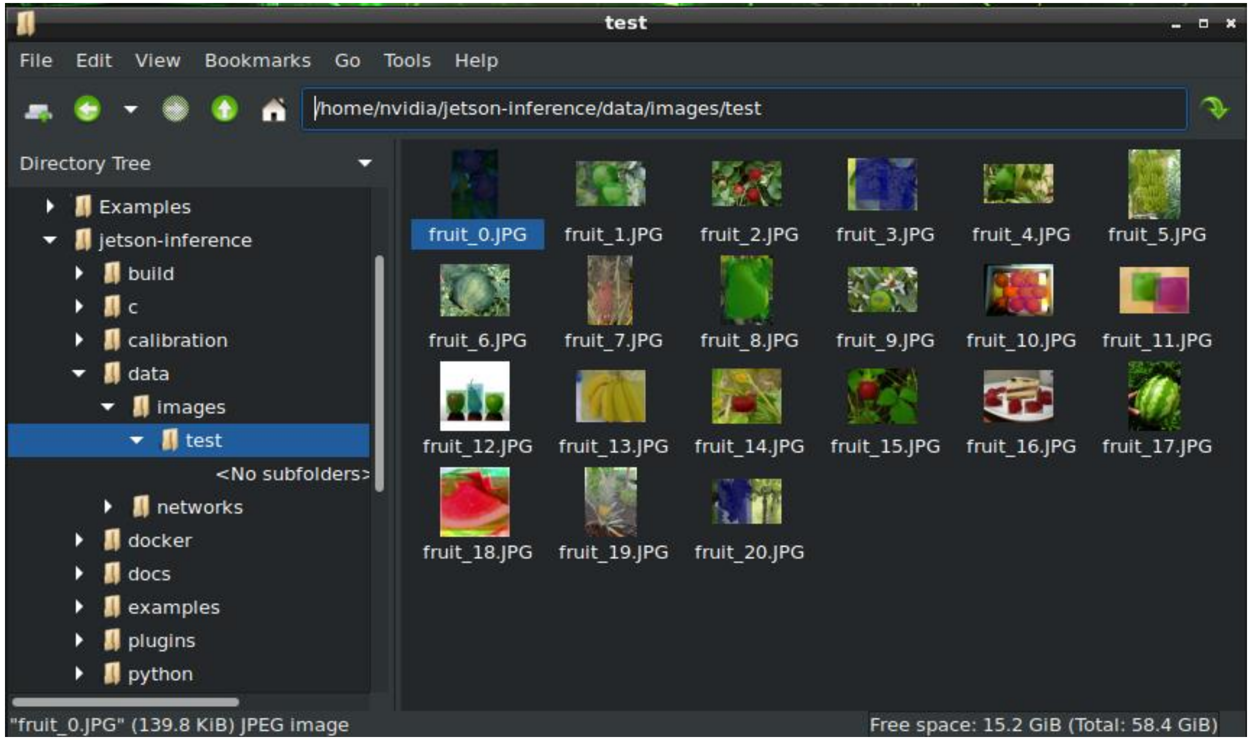
```
$ cd ~/jetson-inference/build/aarch64/bin

$ IMAGES=/home/nvidia/jetson-inference/data/images

$ detectnet --model=/home/nvidia/jetson-inference/python/training/detection/ssd/models/fruit/ssd-mobilenet.onnx --labels=/home/nvidia/jetson-inference/python/training/detection/ssd/models/fruit/labels.txt --input-blob=input_0 --output-cvg=scores --output-bbox=boxes   "$IMAGES/fruit_*.jpg" $IMAGES/test/fruit_%i.jpg
```



```
nvidia@nvidia-desktop:~/jetson-inference/build/aarch64/bin$ cd ~/jetson-inferenc
e/build/aarch64/bin
nvidia@nvidia-desktop:~/jetson-inference/build/aarch64/bin$ IMAGES=~/jetson-infe
rence/data/images
nvidia@nvidia-desktop:~/jetson-inference/build/aarch64/bin$ detectnet --model=/h
ome/nvidia/jetson-inference/python/training/detection/ssd/models/fruit/ssd-mobil
enet.onnx --labels=/home/nvidia/jetson-inference/python/training/detection/ssd/m
odels/fruit/labels.txt --input-blob=input_0 --output-cvg=scores --output-bbox=bo
xes   "$IMAGES/fruit_*.jpg" $IMAGES/test/fruit_%i.jpg
[video]  created imageLoader from file:///home/nvidia/jetson-inference/data/imag
es/fruit_*.jpg
------------------------------------------------
imageLoader video options:
------------------------------------------------
  -- URI: file:///home/nvidia/jetson-inference/data/images/fruit_*.jpg
     - protocol:  file
     - location:  /home/nvidia/jetson-inference/data/images/fruit_*.jpg
     - extension: jpg
  -- deviceType: file
  -- ioType:     input
  -- codec:      unknown
  -- width:      0
  -- height:     0
```

Step 3: Inference model with image

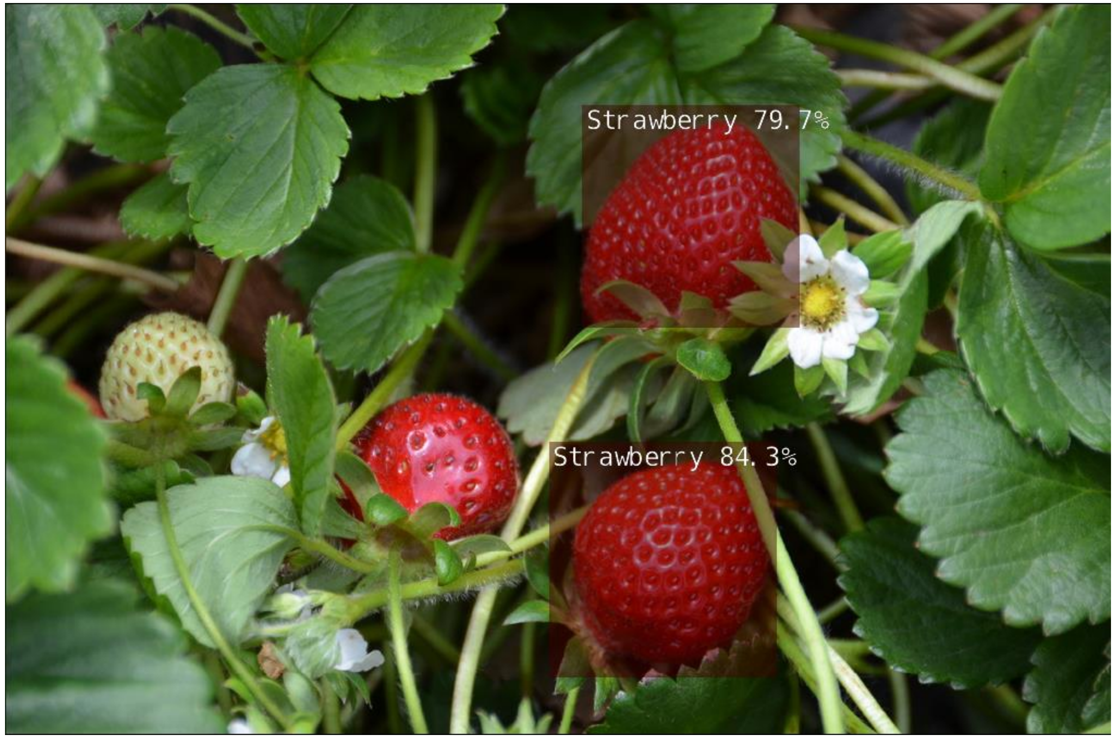
- You can find the result at */home/nvidia/jetson-inference/data/images/test*



/home/nvidia/jetson-inference/data/images/test

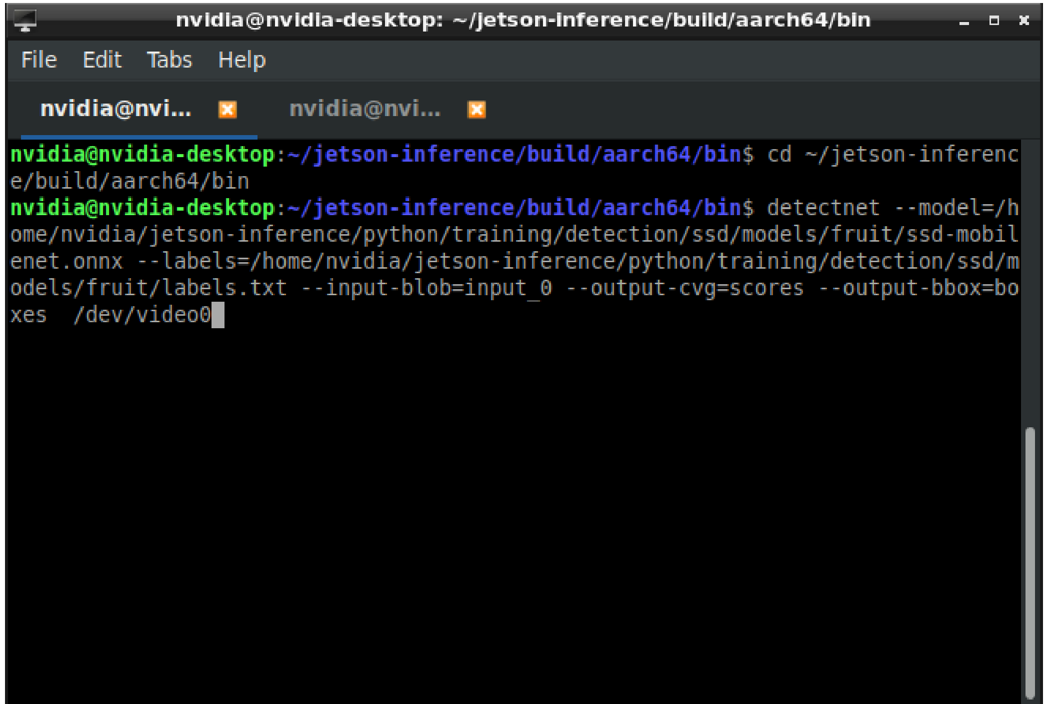
"fruit_0.JPG" (139.8 KiB) JPEG image

Free space: 15.2 GiB (Total: 58.4 GiB)

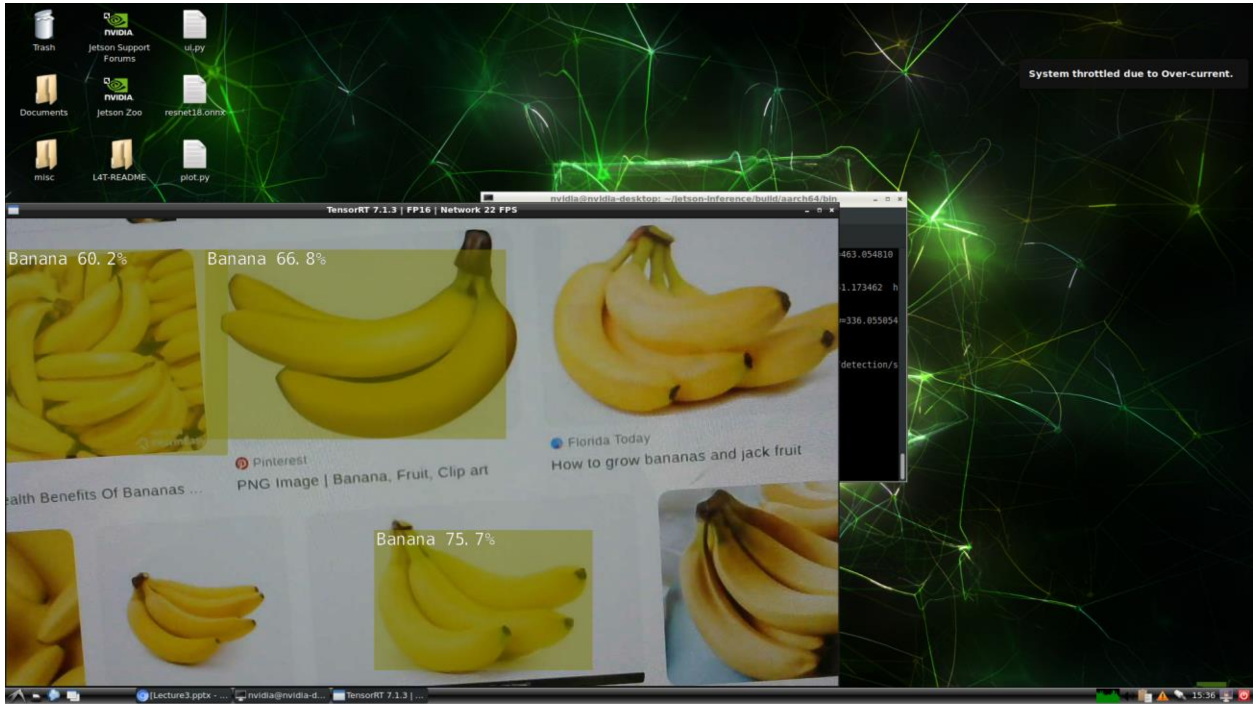


Fruit detection with live camera

```
$ cd ~/jetson-inference/build/aarch64/bin
$ detectnet --model=/home/nvidia/jetson-inference/python/training/detection/ssd/models/fruit/ssd-mobilenet.onnx --labels=/home/nvidia/jetson-inference/python/training/detection/ssd/models/fruit/labels.txt --input-blob=input_0 --output-cvg=scores --output-bbox=boxes  /dev/video0
```



```
nvidia@nvidia-desktop:~/jetson-inference/build/aarch64/bin$ cd ~/jetson-inferenc
e/build/aarch64/bin
nvidia@nvidia-desktop:~/jetson-inference/build/aarch64/bin$ detectnet --model=/h
ome/nvidia/jetson-inference/python/training/detection/ssd/models/fruit/ssd-mobil
enet.onnx --labels=/home/nvidia/jetson-inference/python/training/detection/ssd/m
odels/fruit/labels.txt --input-blob=input_0 --output-cvg=scores --output-bbox=bo
xes  /dev/video0
```


Trash
Jetson Support Forums
ui.py
Documents
Jetson Zoo
resnet18.onnx
misc
L4T-README
plot.py
System throttled due to Over-current.
nvidia@nvidia-desktop: ~/jetson-inference/build/aarch64/bin
TensorRT 7.1.3 | FP16 | Network 22 FPS
Banana 60. 2%
Banana 66. 8%
Banana 75. 7%
Pinterest
PNG Image | Banana, Fruit, Clip art
Florida Today
How to grow bananas and jack fruit
ealth Benefits Of Bananas ...
[Lecture3.pptx - ...
nvidia@nvidia-d...
TensorRT 7.1.3 | ...
15:36

Coding Your Own Object Detection Program

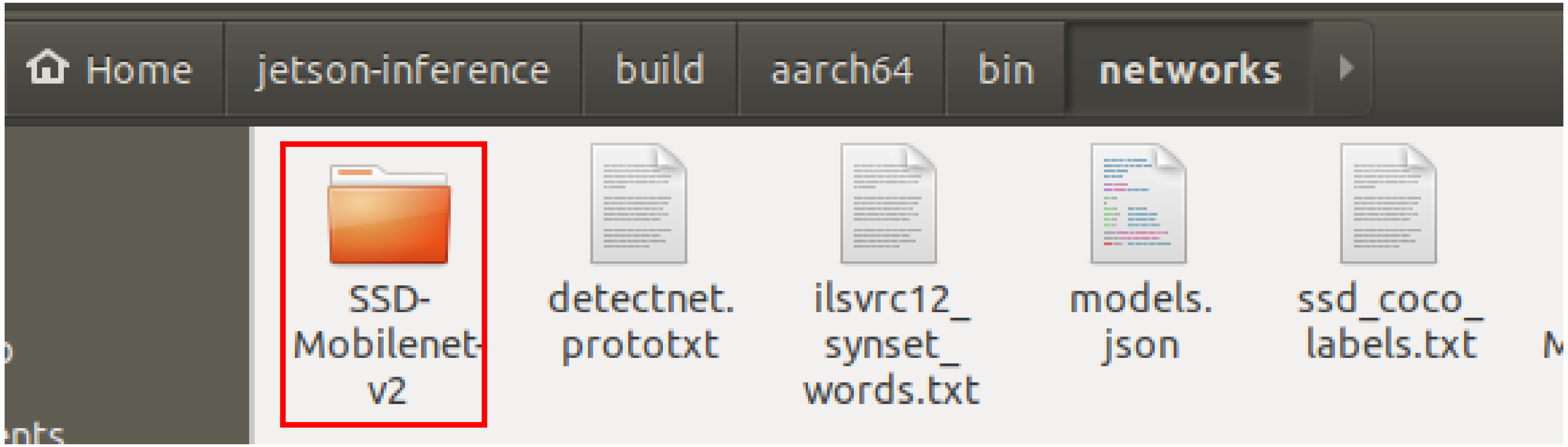
First, open up your text editor of choice and create a new file named **my-detection.py**. Below we'll assume that you'll save it on your host device under **home/jetson-inference/examples/**,

At the top of the source file, we'll import the Python modules that we're going to use in the script. Add import statements to load the jetson_inference and jetson_utils modules used for object detection and camera capture.

```
=================================================================================
#from jetson_inference import detectNet
#from jetson_utils import videoSource, videoOutput
import jetson.inference
import jetson.utils
```

Loading the Detection Model

Next download the SSD-Mobilenet-v2 model and put the folder in the path:**/home/nvidia/jetson-inference/build/aarch64/bin/networks**



Then use the following line to create a detectNet object instance that loads the 91-class SSD-Mobilenet-v2 model:

==

```
net = detectNet("ssd-mobilenet-v2", threshold=0.5)
```

Opening the Camera Stream

To connect to the camera device for streaming, we'll create an instance of the videoSource object:

```
========================================================================
camera = videoSource("/dev/video0")      # '/dev/video0' for V4L2
```

Display Loop

Next, we'll create a video output interface with the videoOutput object and create a main loop that will run until the user exits:

```
===================================================================
display = videoOutput("display://0") # 'my_video.mp4' for file

while display.IsStreaming(): # main loop will go here
```

Camera Capture

The first thing that happens in the main loop is to capture the next video frame from the camera. camera.Capture() will wait until the next frame has been sent from the camera and loaded into GPU memory.

```
============================================================================
display = videoOutput("display://0") # 'my_video.mp4' for file


while display.IsStreaming(): # main loop will go here
        img = camera.Capture()

        if img is None: # capture timeout
                continue
```

Next the detection network processes the image with the net.Detect() function. It takes in the image from camera.Capture() and returns a list of detections:

==

```
#from jetson_inference import detectNet

#from jetson_utils import videoSource, videoOutput

import jetson.inference

import jetson.utils


net = detectNet("ssd-mobilenet-v2", threshold=0.5)

camera = videoSource("/dev/video0")      # '/dev/video0' for V4L2

display = videoOutput("display://0") # 'my_video.mp4' for file

while display.IsStreaming():

    img = camera.Capture()

    if img is None: # capture timeout

        continue

    detections = net.Detect(img)
```

You can add a **print(detections)** statement here, and the coordinates, confidence, and class info will be printed out to the terminal for each detection result.

Rendering

Finally we'll visualize the results with OpenGL and update the title of the window to display the current peformance:

===

```
while display.IsStreaming():
    img = camera.Capture()

    if img is None: # capture timeout
        continue

    detections = net.Detect(img)

    display.Render(img)
    display.SetStatus("Object Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
```

Running the Program

```
$ python3 my-detection.py
```

example

-- ClassID: 5

-- Confidence: 0.927246

-- Left: 2.44141

-- Top: 60.791

-- Right: 499

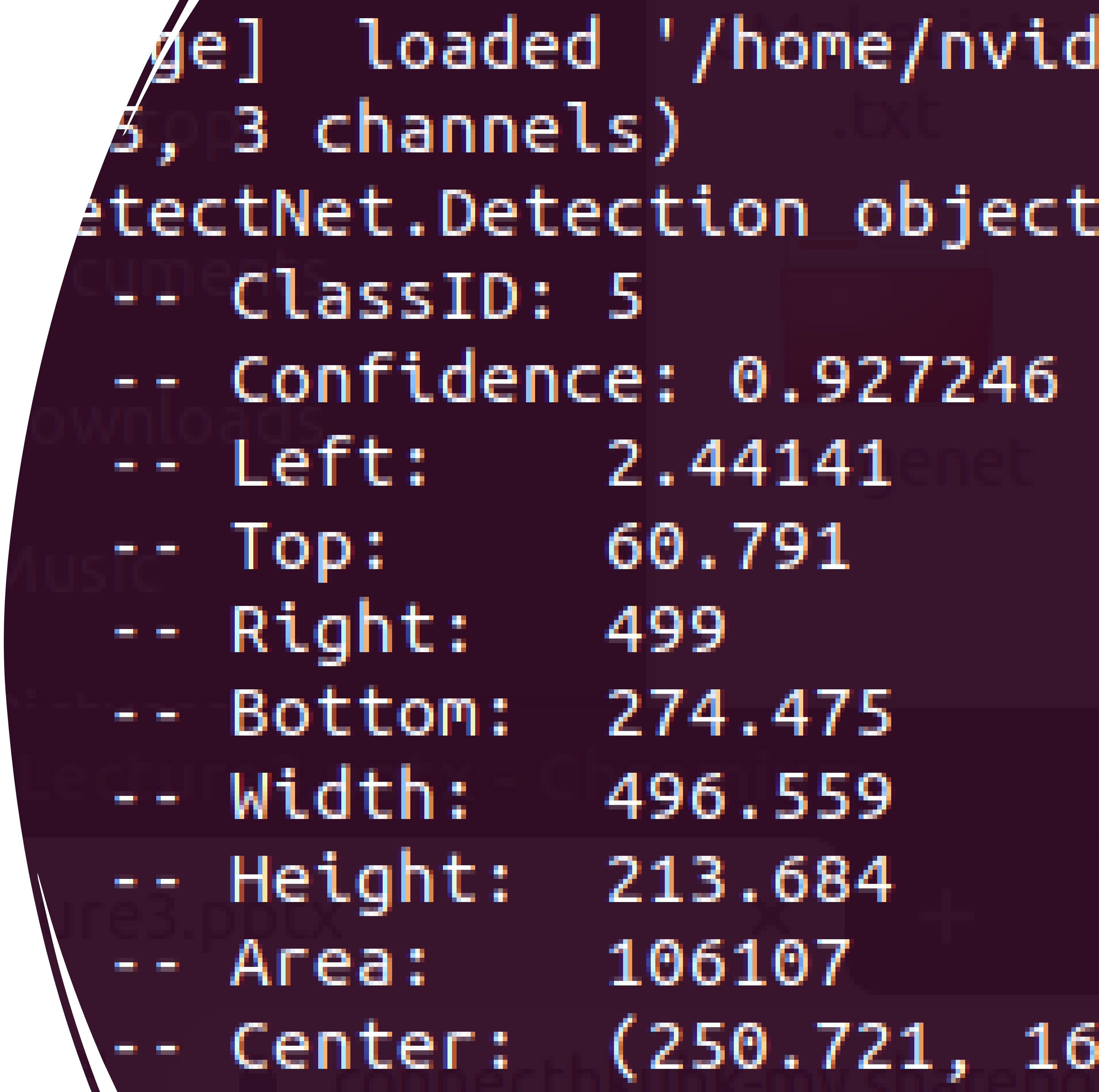
-- Bottom: 274.475

-- Width: 496.559

-- Height: 213.684

-- Area: 106107

-- Center: (250.721, 167.633)



```
ge]  loaded '/home/nvid
5, 3 channels)
etectNet.Detection object
 -- ClassID: 5
 -- Confidence: 0.927246
 -- Left:      2.44141
 -- Top:       60.791
 -- Right:     499
 -- Bottom:    274.475
 -- Width:     496.559
 -- Height:    213.684
 -- Area:      106107
 -- Center:    (250.721, 16
```

Some concepts

-- **ClassID**: The ClassID represents the category or class of the detected object.

-- **Confidence**: The Confidence score indicates how certain the model is about its prediction for the detected object.

-- These values define the coordinates of the bounding box around the detected object.

Left & Top: Left and Top specify the coordinates of the top-left corner of the bounding box.

Right & Bottom: Right and Bottom specify the coordinates of the bottom-right corner. These coordinates are essential for locating the object in the image.

-- **Width**: Width and Height are derived from the bounding box coordinates and represent the dimensions of the bounding box. They show how wide and tall the detected object is in the image.

-- **Height**: Width and Height are derived from the bounding box coordinates and represent the dimensions of the bounding box. They show how wide and tall the detected object is in the image.

-- **Area**: The Area is the total size of the bounding box, calculated as Width × Height.

-- **Center**: The Center represents the midpoint of the bounding box, calculated using the coordinates.

Assignment 3

Find out the coordinates, confidence, and class info of two image **detection** results including **ClassID, Confidence, Left, Top, Right, Bottom, Width, Height, Area, Center**.

Please upload the code (your-detection.py) to github, a **doc or pdf to the moodle** (including github link, the detected image and the detection results)

Deadline: 03/03/2025 **11:59pm**

Tips

- Change the video input to one of the image input
- Choose one of the class output
- Learn to use debug function in Visual Studio Code