



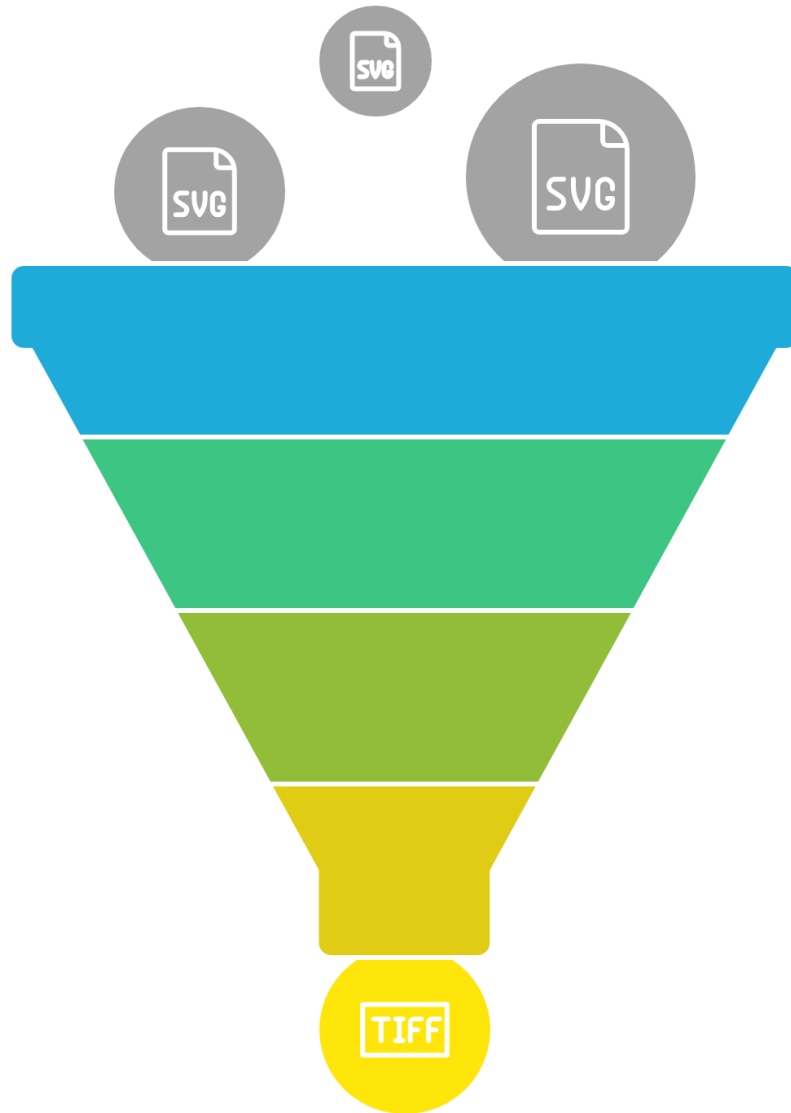
Capstone Project

Edge computing device programming
for AI projects

Lecture 2: Image classification

Dr. Wilton Fok & Carol Chen

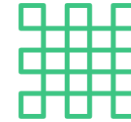
Image classification



1 -

Input Processing

Convert images to numerical data



Feature Extraction

Identify patterns in images



Feature Selection

Reduce dimensionality and summarize features

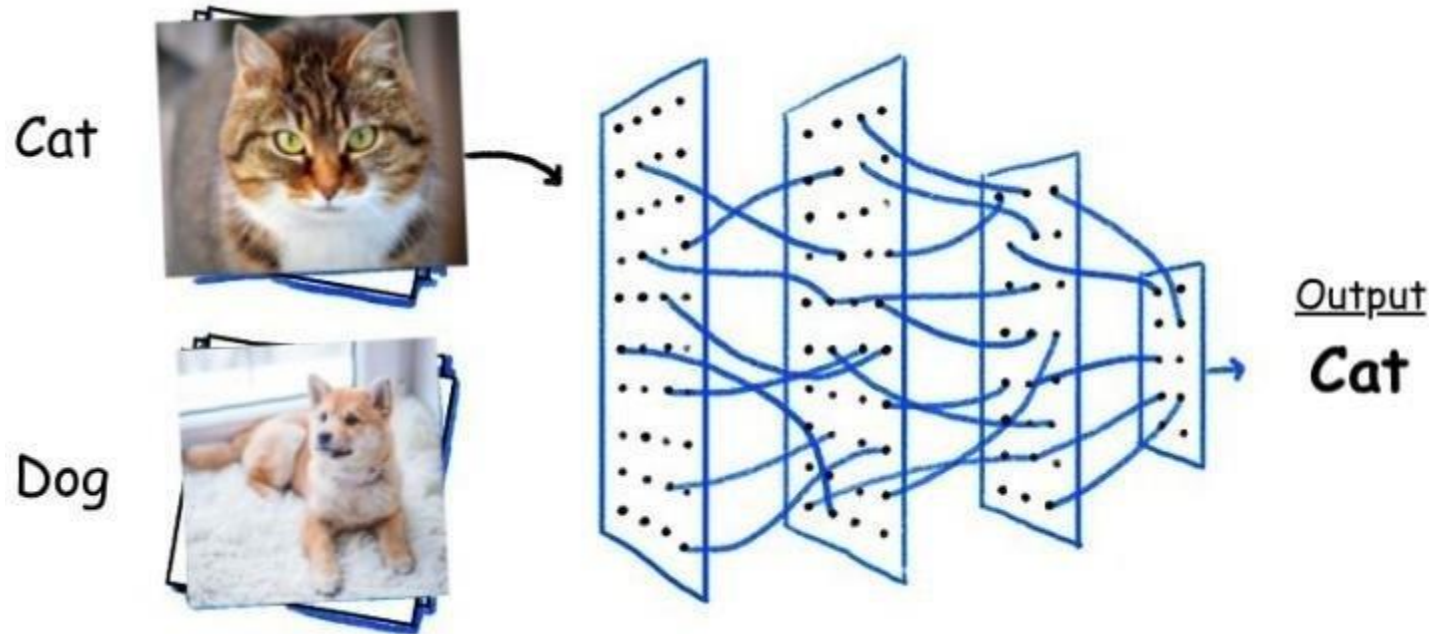


Classification

Determine image class probabilities

Mission

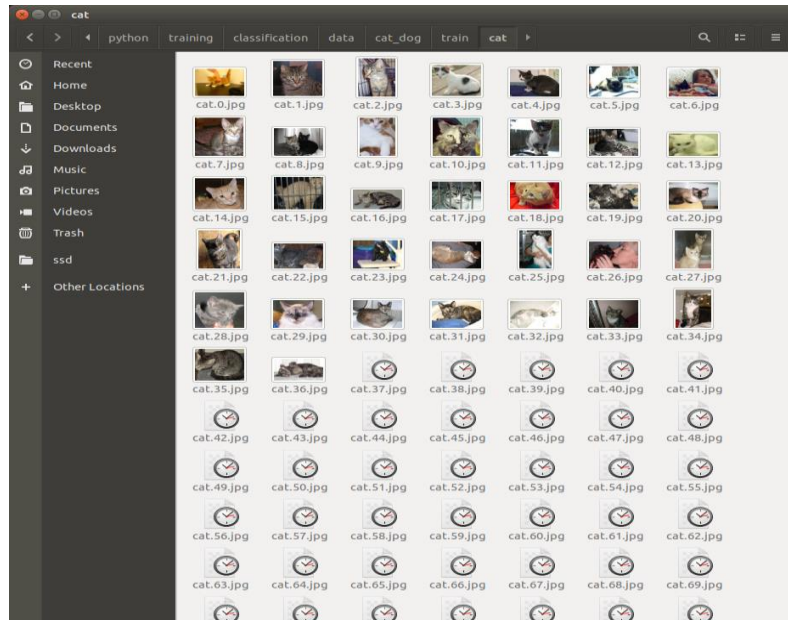
- Train a catdog recognition model using these images, and use it to classify cat and dog.



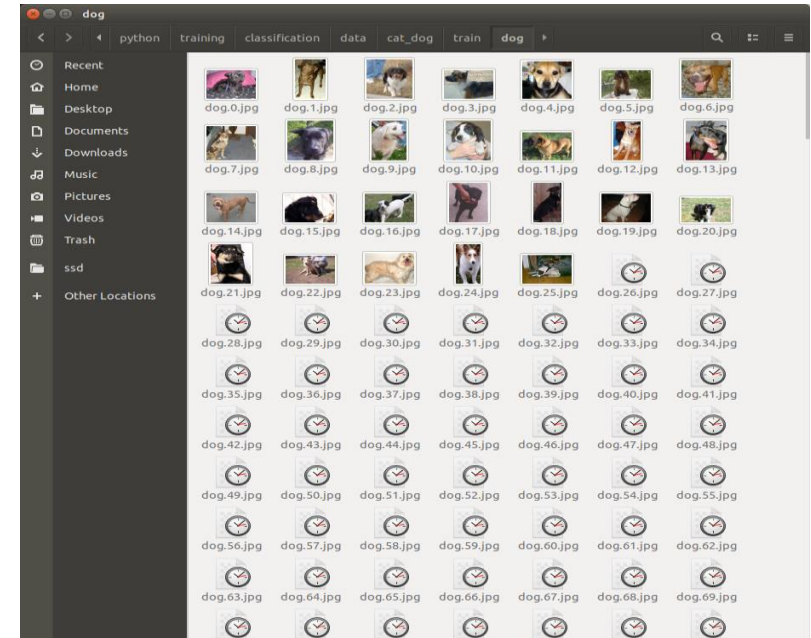
Task1: Train the built-in model --- catdog

- Download the catdog dataset from onedrive [cat dog \(https://connecthkuhk-my.sharepoint.com/:f:/g/personal/u3597499_connect_hku_hk/EtShD5TRW_JOo4Ad_AcBw4kBvvB_8vsBWQ2uNMANLJ014g?e=b2tsWg\)](https://connecthkuhk-my.sharepoint.com/:f:/g/personal/u3597499_connect_hku_hk/EtShD5TRW_JOo4Ad_AcBw4kBvvB_8vsBWQ2uNMANLJ014g?e=b2tsWg)
- and put it in ***/home/nvidia/jetson-inference/python/training/classification/data*** or ***/home/jetson-inference/python/training/classification/data*** .

cat



dog



Step 1: Train model

- To train the model, we need to open a terminal, and enter:

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

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

Change the directory into ~/jetson-inference/python/training/classification

```
$ python3 train.py --model-dir=models/cat_dog data/cat_dog --batch-size=1 --  
workers=1 --epochs=2
```

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

Use the software python3 to run the file “train.py” with the following arguments “data/cat_dog --model-dir=models/cat_dog --batch-size=1 --workers=1 --epochs=2”

--batch-size=4 --workers=1 --epochs=36

- Batch size is a term used in machine learning and refers to the number of training examples utilized in one iteration.
- Workers : Create multiple threads and load data in advance. num_workers=0 means ONLY the main process will load batches (that can be a bottleneck). num_workers=1 means ONLY one worker (just not the main process) will load data, but it will still be slow.
- An epoch elapses when an entire dataset is passed forward and backward through the neural network exactly one time.

Data set size = Iteration * Batch size (1 Epoch)

Iteration = (Data set size / Batch size) * Epoch

Step 1: Train model

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

```
nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification
Use GPU: 0 for training
=> dataset classes: 2 ['cat', 'dog']
=> using pre-trained model 'resnet18'
Downloading: "https://download.pytorch.org/models/resnet18-5c106cde.pth" to /home/nvidia/.cache/torch/hub/checkpoints/resnet18-5c106cde.pth
100.0%
=> reshaped ResNet fully-connected layer with: Linear(in_features=512, out_features=2, bias=True)
Epoch: [0][ 0/166] Time 55.485 (55.485) Data 1.523 ( 1.523) Loss 6.8547e-01
(6.8547e-01) Acc@1 50.00 ( 50.00) Acc@5 100.00 (100.00)
Epoch: [0][ 10/166] Time 0.619 ( 5.620) Data 0.000 ( 0.141) Loss 8.8989e+00
(1.2608e+01) Acc@1 37.50 ( 44.32) Acc@5 100.00 (100.00)
Epoch: [0][ 20/166] Time 0.618 ( 3.240) Data 0.000 ( 0.091) Loss 5.7193e+00
(1.0944e+01) Acc@1 62.50 ( 45.24) Acc@5 100.00 (100.00)
Epoch: [0][ 30/166] Time 0.618 ( 2.395) Data 0.000 ( 0.073) Loss 1.3559e+01
(1.0417e+01) Acc@1 0.00 ( 43.95) Acc@5 100.00 (100.00)
Epoch: [0][ 40/166] Time 0.615 ( 1.962) Data 0.000 ( 0.064) Loss 2.6755e+00
(8.7667e+00) Acc@1 62.50 ( 44.82) Acc@5 100.00 (100.00)
Epoch: [0][ 50/166] Time 0.615 ( 1.699) Data 0.000 ( 0.058) Loss 7.4683e-01
(7.2513e+00) Acc@1 37.50 ( 46.08) Acc@5 100.00 (100.00)
Epoch: [0][ 60/166] Time 0.726 ( 1.524) Data 0.000 ( 0.055) Loss 5.9601e-01
(6.1910e+00) Acc@1 62.50 ( 45.90) Acc@5 100.00 (100.00)
Epoch: [0][ 70/166] Time 0.620 ( 1.396) Data 0.000 ( 0.051) Loss 1.2456e+00
(5.4235e+00) Acc@1 37.50 ( 47.71) Acc@5 100.00 (100.00)
```

Acc@1 means that only the true label is the class with **the highest** predicted probability, the prediction is correct.

Acc@5 means that as long as the true label is one of **the five predicted classes with the highest probability**, the prediction is correct.

Example:

If a model predicts probabilities for **an image of a Siamese cat**:

Predicted results:

- Persian Cat: 0.3
- **Siamese Cat: 0.25**
- Tabby Cat: 0.2
- Dog: 0.15
- Bird: 0.1

=====

- Acc@1 would count this as incorrect (Persian Cat \neq Siamese Cat)
- Acc@5 would count this as correct (Siamese Cat is in top 5)

How can we get
the loss value?

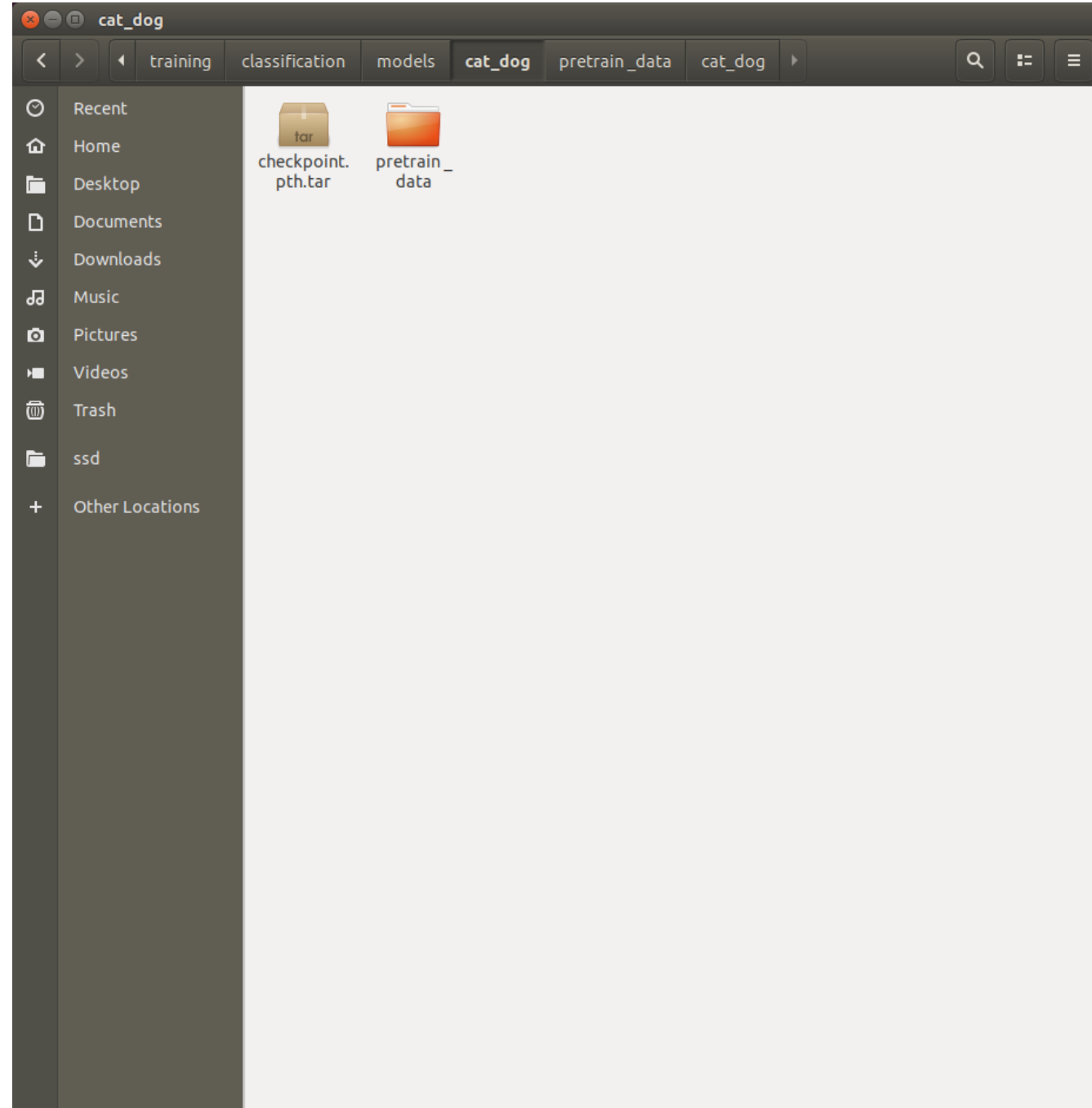
Where is the loss
definition?



Step 2: Convert model

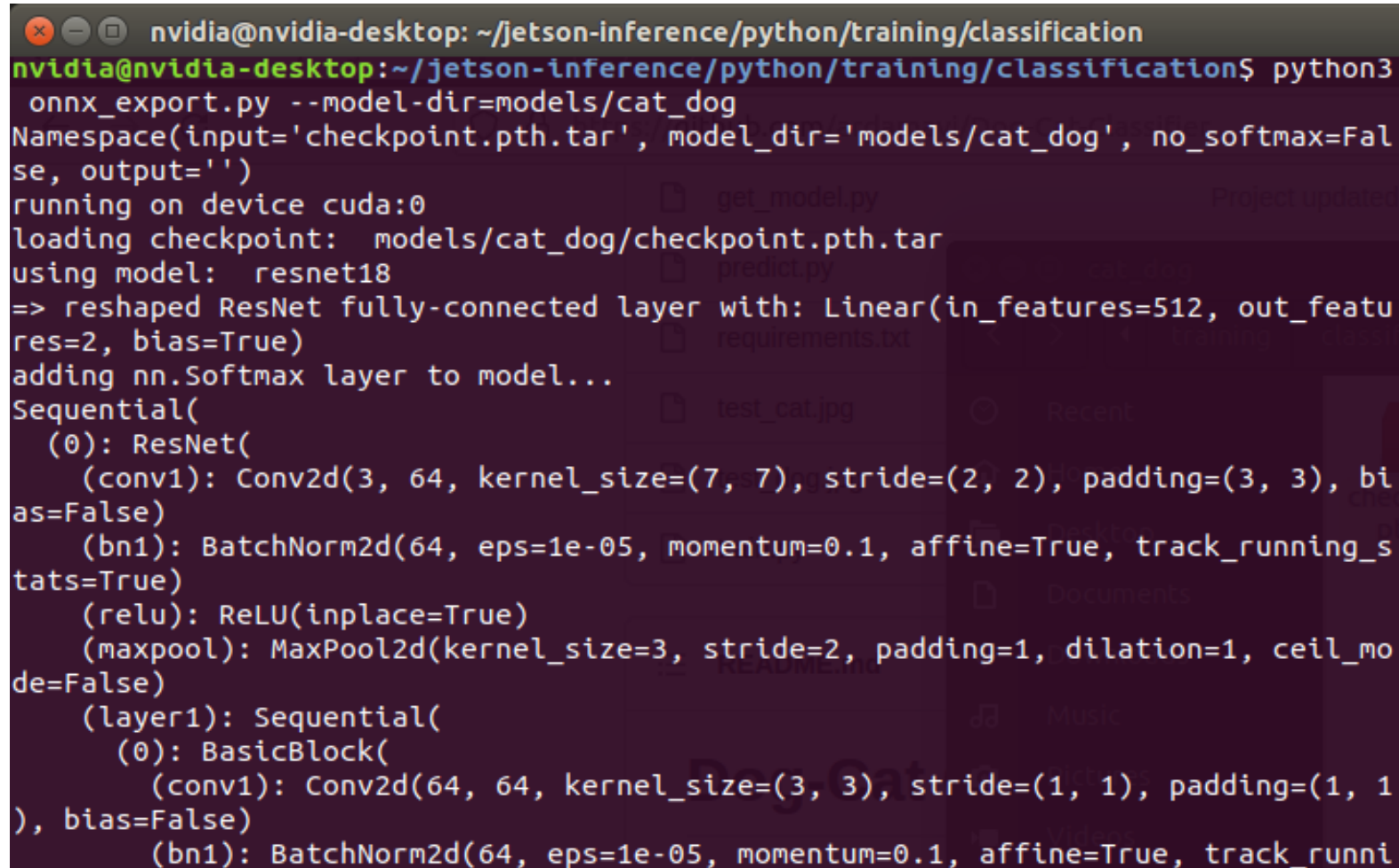
After training, the model folder will be:

In ***/home/jetson-inference/python/training/classification/models/cat_dog/***



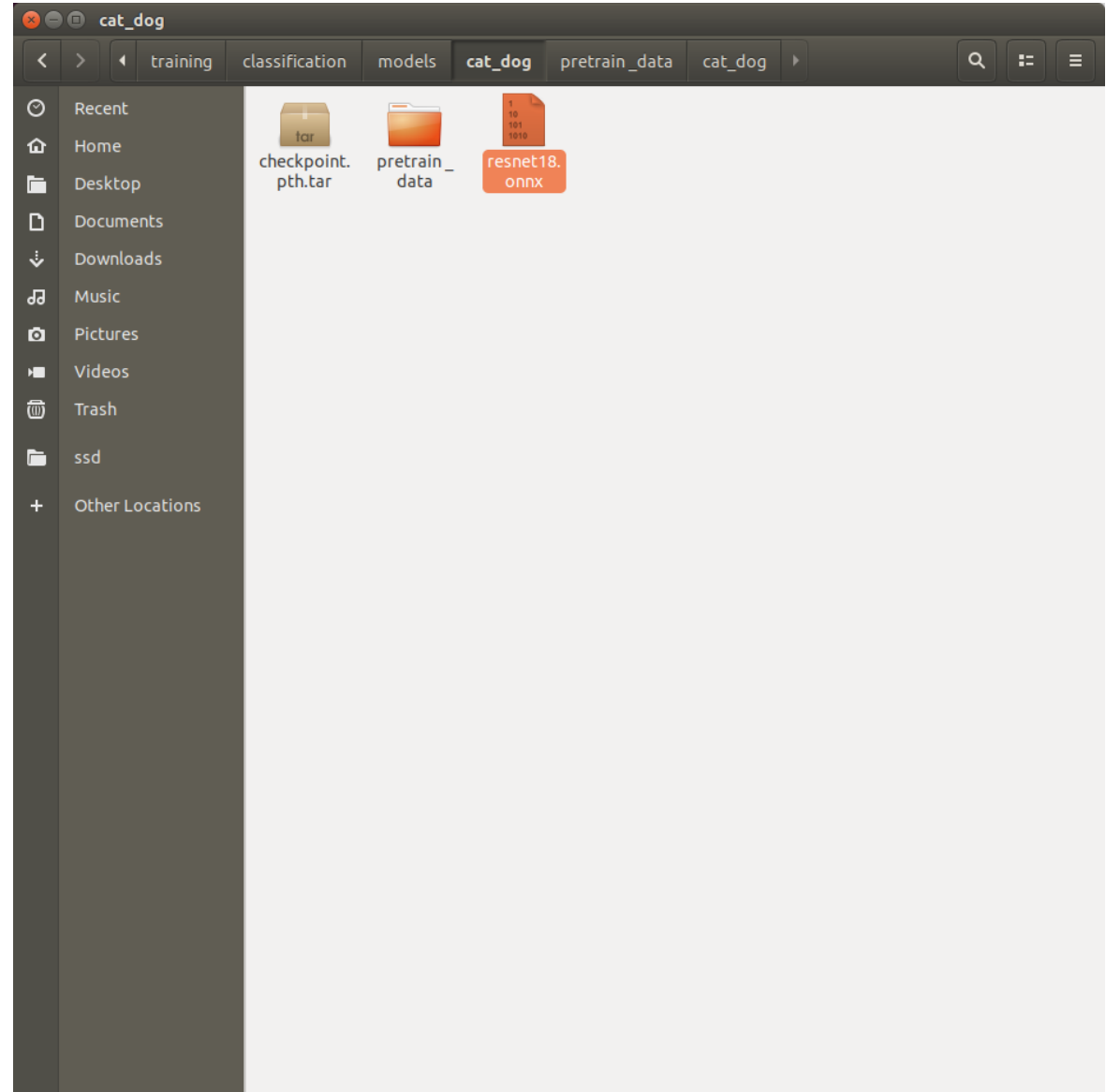
Then, we need to convert the model using:

```
$ python3 onnx_export.py --model-dir=models/cat_dog --input=checkpoint.pth.tar
```

A terminal window screenshot showing the execution of the onnx_export.py script. The terminal title is 'nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification'. The command entered is 'python3 onnx_export.py --model-dir=models/cat_dog'. The output shows the script loading a checkpoint, identifying the model as 'resnet18', and reshaping the fully-connected layer. It then adds a 'nn.Softmax' layer and prints the model architecture in a sequential format.

```
nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ python3
onnx_export.py --model-dir=models/cat_dog
Namespace(input='checkpoint.pth.tar', model_dir='models/cat_dog', no_softmax=False, output='')
running on device cuda:0
loading checkpoint: models/cat_dog/checkpoint.pth.tar
using model: resnet18
=> reshaped ResNet fully-connected layer with: Linear(in_features=512, out_features=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), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=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_runni
```

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



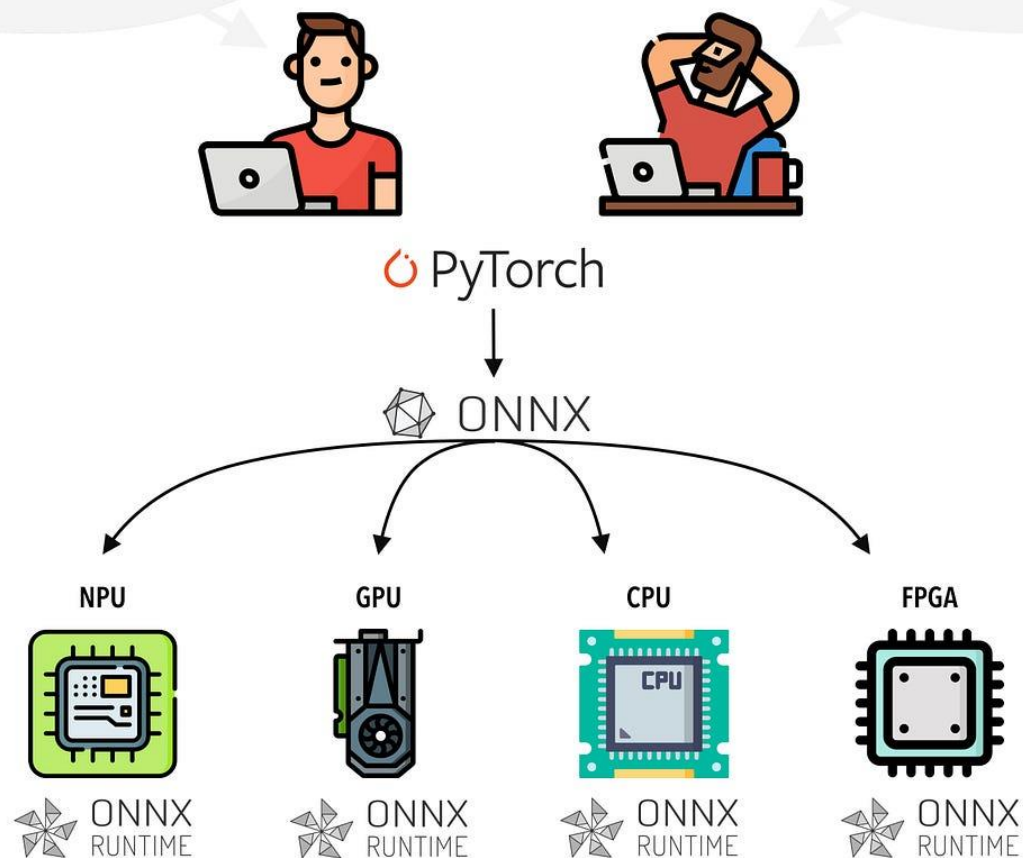
I trained and optimized a model on a GPU with PyTorch, but I want to deploy it on an IoT device without losing performance, how could I do it?



One training
and multiple
deployment?

I trained and optimized a model on a GPU with PyTorch, but I want to deploy it on an IoT device without losing performance, how could I do it?

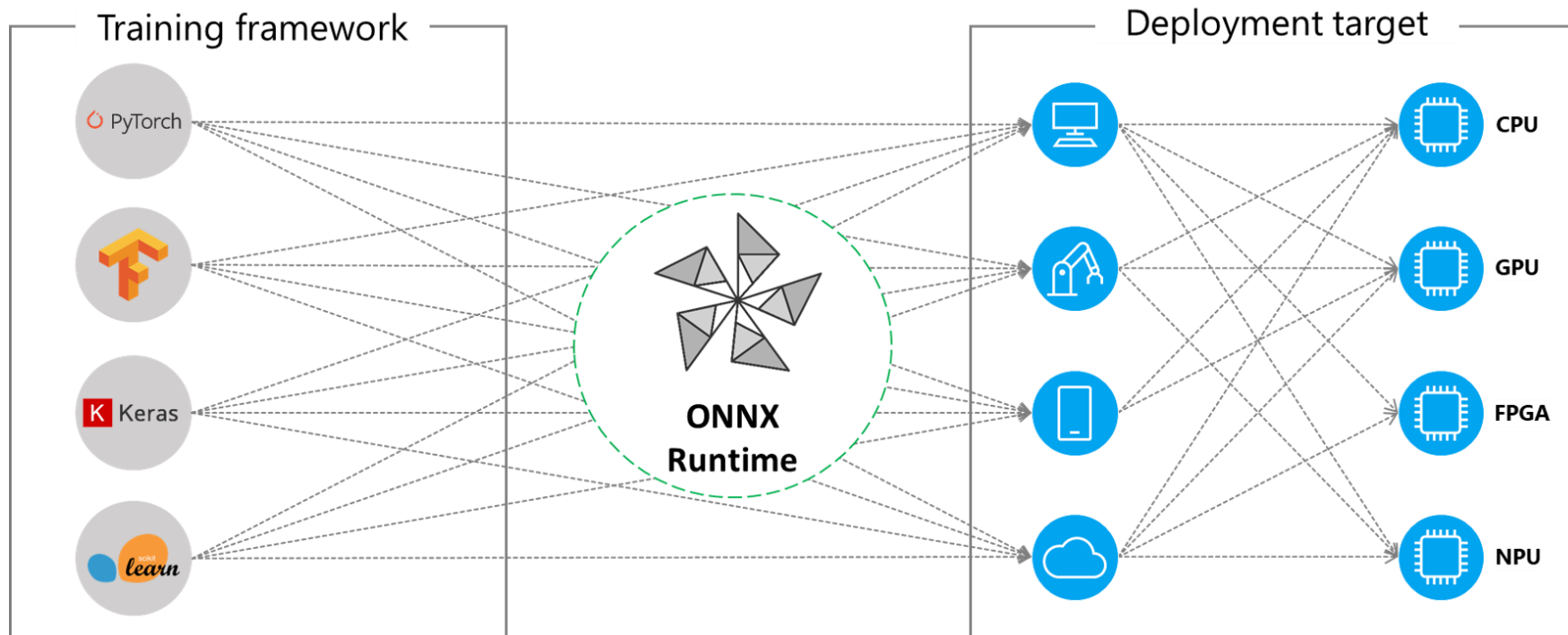
It's very easy bro!
Just export your PyTorch model as ONNX and with ONNX Runtime you will be able to deploy in any architecture!
Just look at the image below



Use ONNX

Open Neural Network Exchange (ONNX)

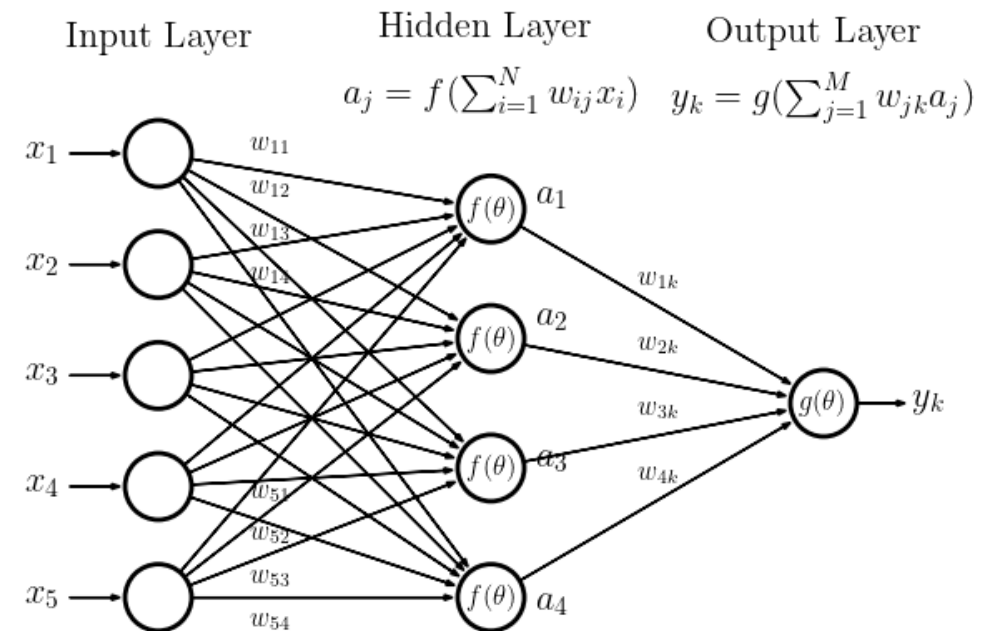
- An open format proposed by Microsoft and Facebook to represent deep learning models.
- Openness means that ONNX defines a set of standard formats that are independent of the environment and platform to enhance the interactivity of various AI models.
- i.e. no matter which training framework you use to train the model (such as TensorFlow/Pytorch/OneFlow/Paddle), after training, you can uniformly convert the models of these frameworks into a unified format such as ONNX for storage.



ONNX file

- ONNX file not only stores the weights of the neural network model, but also stores the structural information of the model, the input and output of each layer in the network, and some other auxiliary information.

- weights of the neural network model
- structural information of the model
- input and output of each layer in network
- and some other auxiliary information.



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 a cat for example:

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

```
$ DATASET=data/cat_dog
```

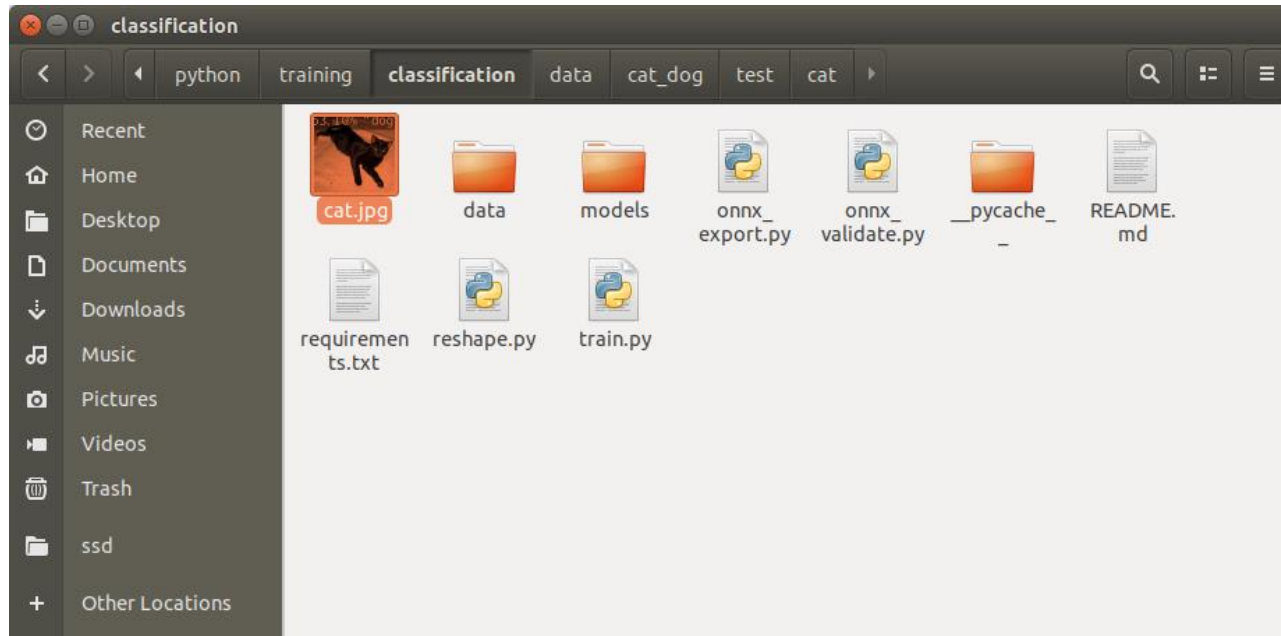
```
$ /home/nvidia/jetson-  
inference/build/aarch64/bin/imagenet --  
model=models/cat_dog/resnet18.onnx --  
input_blob=input_0 --output_blob=output_0 --  
labels=$DATASET/labels.txt $DATASET/test/cat/0001.jpg  
cat.jpg
```



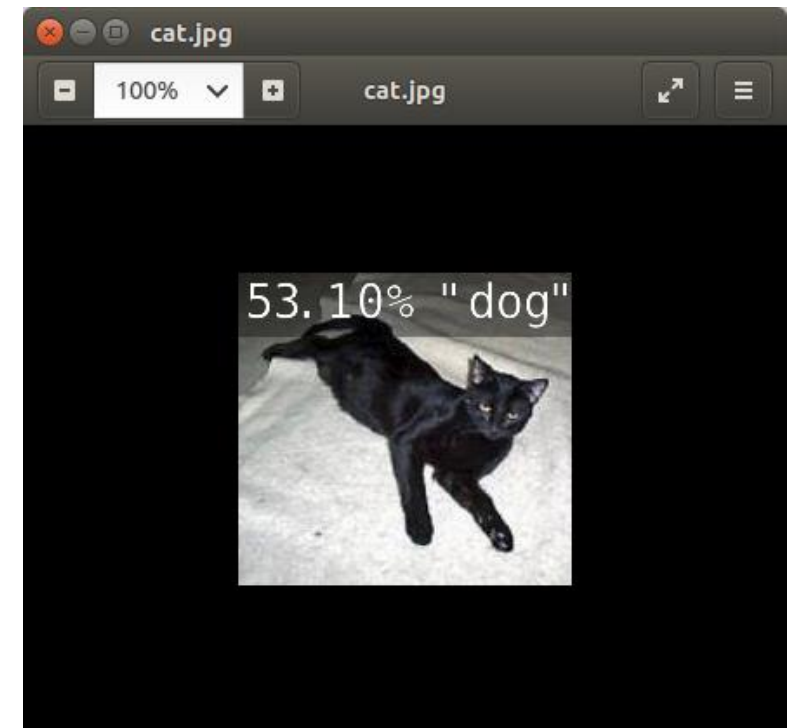
```
nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification  
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ /home/n  
vidia/jetson-inference/build/aarch64/bin/imagenet --model=models/cat_dog/resnet1  
8.onnx --input_blob=input_0 --output_blob=output_0 --labels=$DATASET/labels.txt  
$DATASET/test/cat/0001.jpg  
[video] created imageLoader from file:///home/nvidia/jetson-inference/python/tr  
aining/classification/data/cat_dog/test/cat/0001.jpg  
-----  
imageLoader video options:  
-----  
-- URI: file:///home/nvidia/jetson-inference/python/training/classification/da  
ta/cat_dog/test/cat/0001.jpg  
- protocol: file  
- location: data/cat_dog/test/cat/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  
-----  
1 "tench, Tinca tinca"  
2 "goldfish, Carassius auratus"  
3 "great white shark, white shark, man-eater"  
4 "tiger shark, Galeocerdo cuvieri"  
5 "hammerhead, hammerhead shark"  
6 "electric ray, crampfish, numbfish, torped
```

Step 3: Inference model with image

- We can see the model prediction using your trained model!



Original image

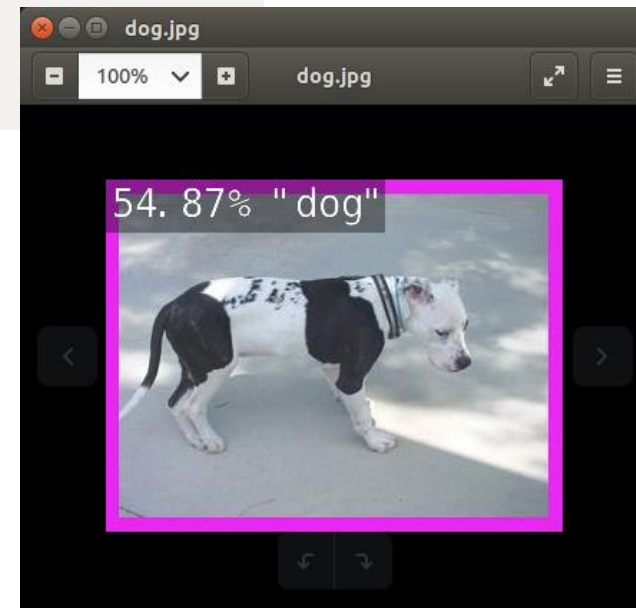
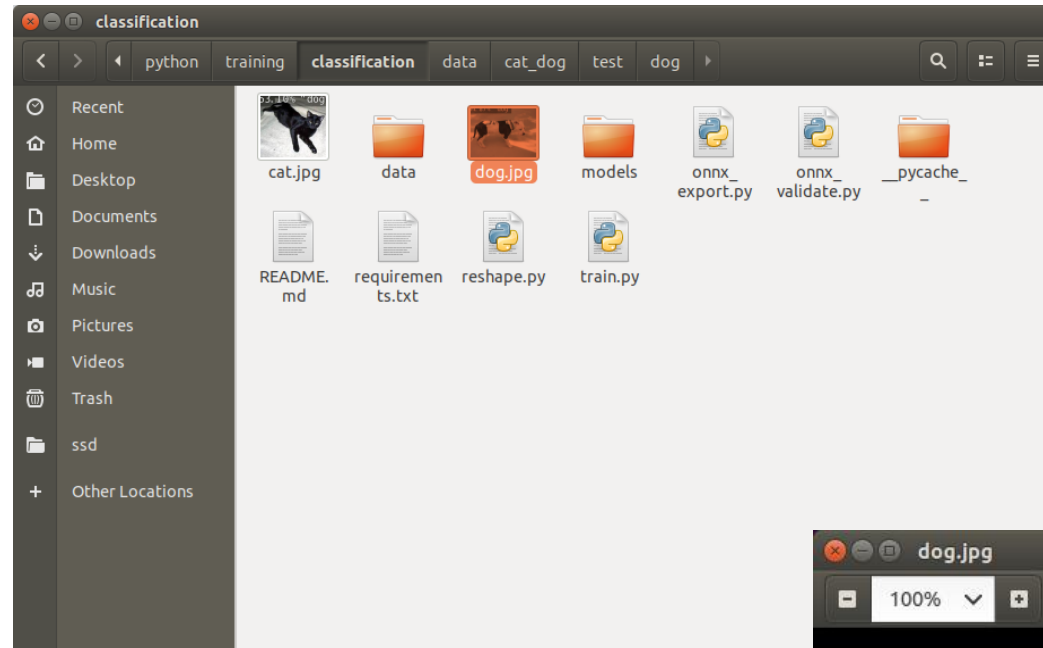


The prediction made by the model

Step 3: Inference model with image

- How about recognizing a dog?

```
nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ /home/n
vidia/jetson-inference/build/aarch64/bin/imagenet --model=models/cat_dog/resnet1
8.onnx --input_blob=input_0 --output_blob=output_0 --labels=$DATASET/labels.txt
$DATASET/test/dog/0016.jpg dog.jpg
[video] created imageLoader from file:///home/nvidia/jetson-inference/python/tr
aining/classification/data/cat_dog/test/dog/0016.jpg
-----
imageLoader video options:
-----
-- URI: file:///home/nvidia/jetson-inference/python/training/classification/da
ta/cat_dog/test/dog/0016.jpg
-- protocol: file
-- location: data/cat_dog/test/dog/0016.jpg
-- extension: jpg
-- deviceType: file
-- ioType: input
-- codec: unknown
-- width: 0
-- height: 0
-- frameRate: 0.000000
-- bitRate: 0
-- numBuffers: 4
-- zeroCopy: true
-- flipMethod: none
```



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

```
$ /home/nvidia/jetson-inference/build/aarch64/bin/imagenet --
model=models/cat_dog/resnet18.onnx --input_blob=input_0 --
output_blob=output_0 --labels=$DATASET/labels.txt
$DATASET/test/dog/0016.jpg dog.jpg
```

Step 3: Inference model with image

To tired to enter commands one by one? Try directly run a whole dataset.

```
nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification
imagenet: shutdown complete.
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ /home/nvidia/jetson-inference/build/aarch64/bin/imagenet --model=models/cat_dog/resnet18.onnx --input_blob=input_0 --output_blob=output_0 --labels=$DATASET/labels.txt $DATASET/test/cat/
[video] created imageLoader from file:///home/nvidia/jetson-inference/python/training/classification/data/cat_dog/test/cat/
-----
imageLoader video options:
-----
-- URI: file:///home/nvidia/jetson-inference/python/training/classification/data/cat_dog/test/cat/
  - protocol: file
  - location: data/cat_dog/test/cat/
-- deviceType: file
-- ioType: input
-- codec: unknown
-- width: 0
-- height: 0
-- frameRate: 0.000000
-- bitRate: 0
-- numBuffers: 4
-- zeroCopy: true
-- flipMethod: none
```

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

```
nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ /home/n
vidia/jetson-inference/build/aarch64/bin/imagenet --model=models/cat_dog/resnet1
8.onnx --input_blob=input_0 --output_blob=output_0 --labels=$DATASET/labels.txt
$DATASET/test/dog/
[video] created imageLoader from file:///home/nvidia/jetson-inference/python/tr
aining/classification/data/cat_dog/test/dog/ions Projects Security Insights
-----
imageLoader video options:
-----
-- URI: file:///home/nvidia/jetson-inference/python/training/classification/da
ta/cat_dog/test/dog/
-- protocol: file
-- location: data/cat_dog/test/dog/
-- deviceType: file
-- ioType: input
-- codec: unknown
-- width: 0
-- height: 0
-- frameRate: 0.000000
-- bitRate: 0
-- numBuffers: 4
-- zeroCopy: true
-- flipMethod: none
-- loop: 0
```

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


Step 4: Inference with video

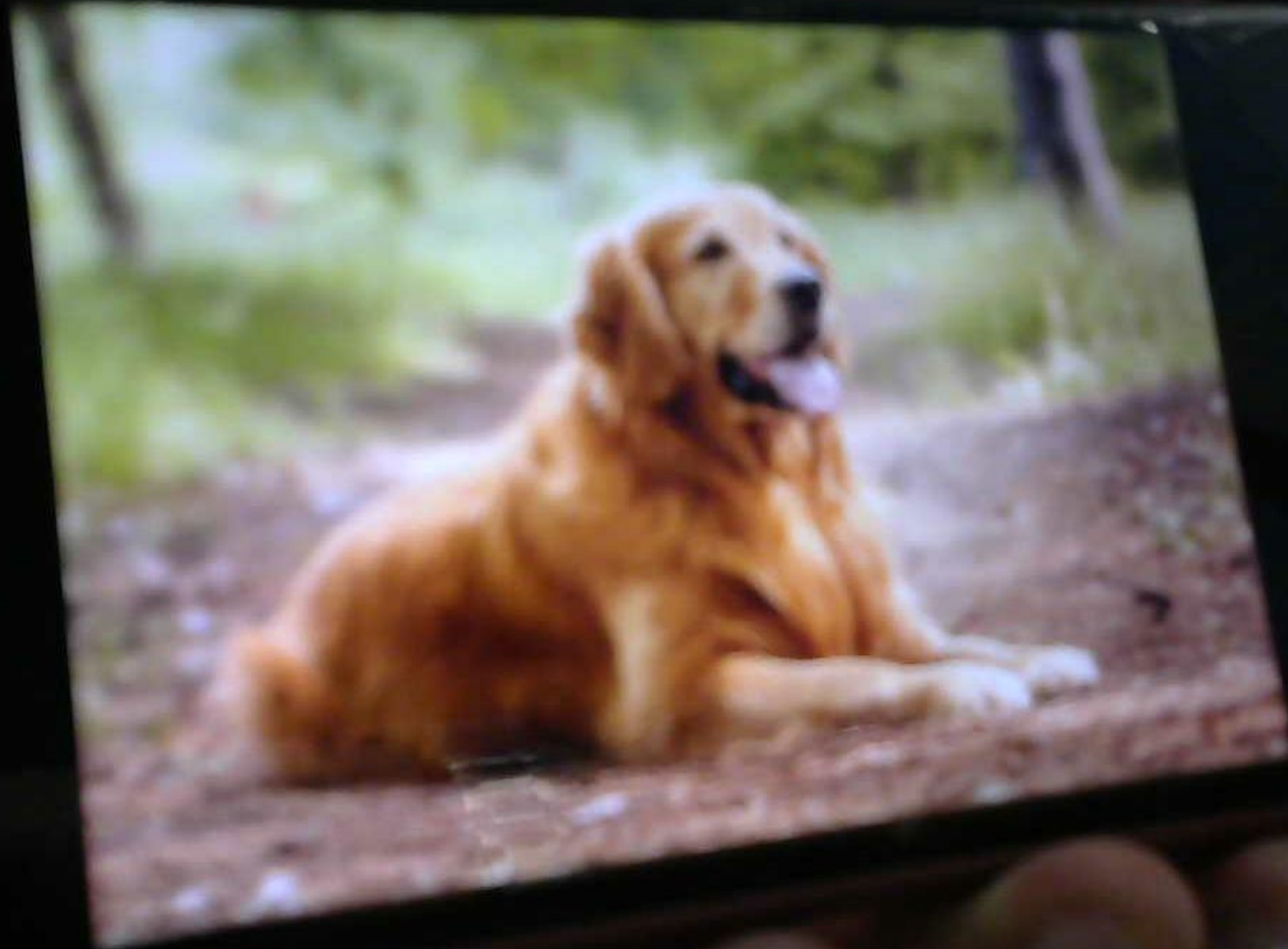
- Apart from image, you can also use this model through webcam.

```
nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification
[TRT] -----
class 0000 - 0.451296 ("cat")
class 0001 - 0.548704 ("dog")
imagenet: 54.87037% class #1 ("dog")
[OpenGL]glDisplay -- the window has been closed
[TRT] -----
[TRT] File Timing Report models/cat_dog/resnet18.onnx /alexnet/imagenet-labels.txt
[TRT] -----
[TRT] Pre-Process   CPU    0.08370ms   CUDA    0.61547ms
[TRT] Network       CPU    22.34445ms  CUDA    12.79510ms
[TRT] Post-Process  CPU    0.07084ms  CUDA    0.07026ms
[TRT] Total         CPU    22.49898ms  CUDA    13.48083ms
[TRT] -----
imagenet: shutting down...
[gstreamer] gstCamera -- stopping pipeline, transitioning to GST_STATE_NULL
[gstreamer] gstCamera -- pipeline stopped
imagenet: shutdown complete.
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ /home/n
vidia/jetson-inference/build/aarch64/bin/imagenet --model=models/cat_dog/resnet1
8.onnx --input_blob=input_0 --output_blob=output_0 --labels=$DATASET/labels.txt
/dev/video0
```

```
$ /home/nvidia/jetson-
inference/build/aarch64/bin/imagenet --
model=models/cat_dog/resnet18.onnx --
input_blob=input_0 --
output_blob=output_0 --
labels=$DATASET/labels.txt /dev/video0
```

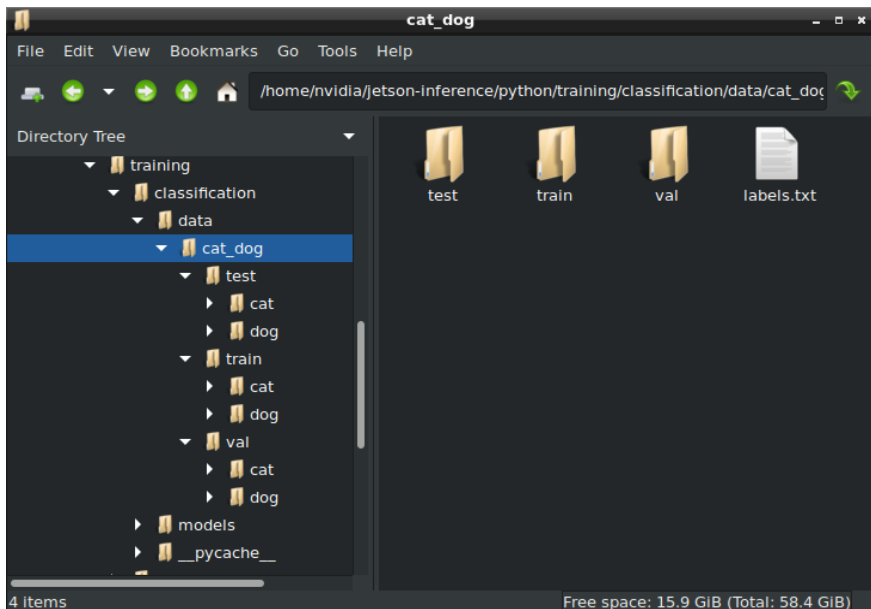

TensorRT 8.0.1 | Custom | Network 85 FPS

53.34% "dog"

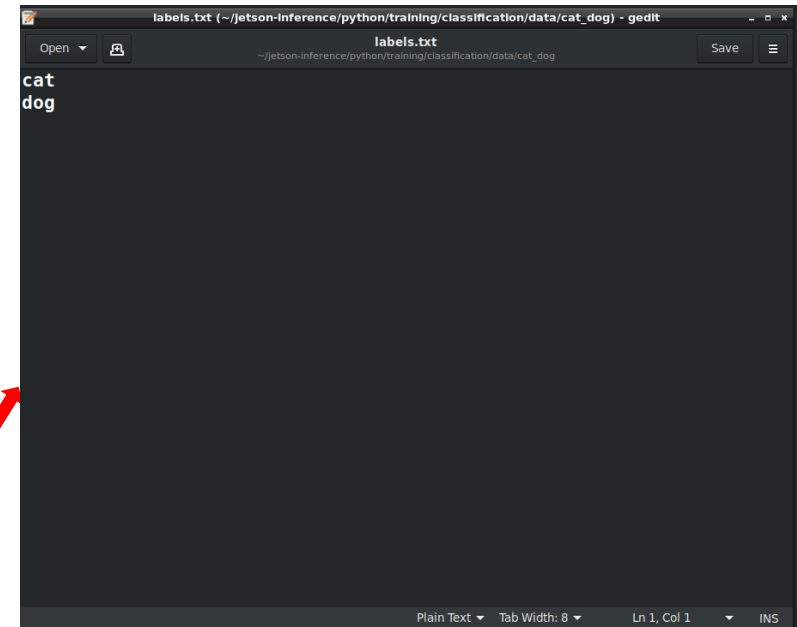


Task 2: Train our own model

- 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.



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





Training with your own dataset

- A group of dataset
 - Two important parameter should be defined
 - (1) Where to store your data (***YourDataPath***)
 - (2) Where to store your model (***YourModelPath***)
 - Following the similar steps to finish model training, conversion and inference.
-

Dataset collection



Now, you can download the images you want to classify from google.



Remember the image between train, validation and test is around 7:2:1



After finishing the dataset, download [check_dataset.py](#) and run it to verify whether the dataset structure is valid.

Train our own model: Training

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

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

```
$ python3 train.py --model-dir=YourModelPathYourDataPath --  
batch-size=1 --workers=1 --epochs=1
```

***PREVIOUS EXAMPLE:

```
$ python3 train.py --model-dir=models/cat_dog|data/cat_dog|--batch-size=4 --workers=1 --  
epochs=36
```

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

Train our own model: Converting and Inference

Convert

```
$ python3 onnx_export.py --model-dir=YourModelPath
```

Inference with image

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

```
$ DATASET=YourDataPath
```

```
$ /home/nvidia/jetson-inference/build/aarch64/bin/imagenet --
```

```
model=_YourModelPath/resnet18.onnx --input_blob=input_0
```

```
--output_blob=output_0 --labels=$DATASET/labels.txt
```

```
TheImageYouWantToProcess
```

****Example:**

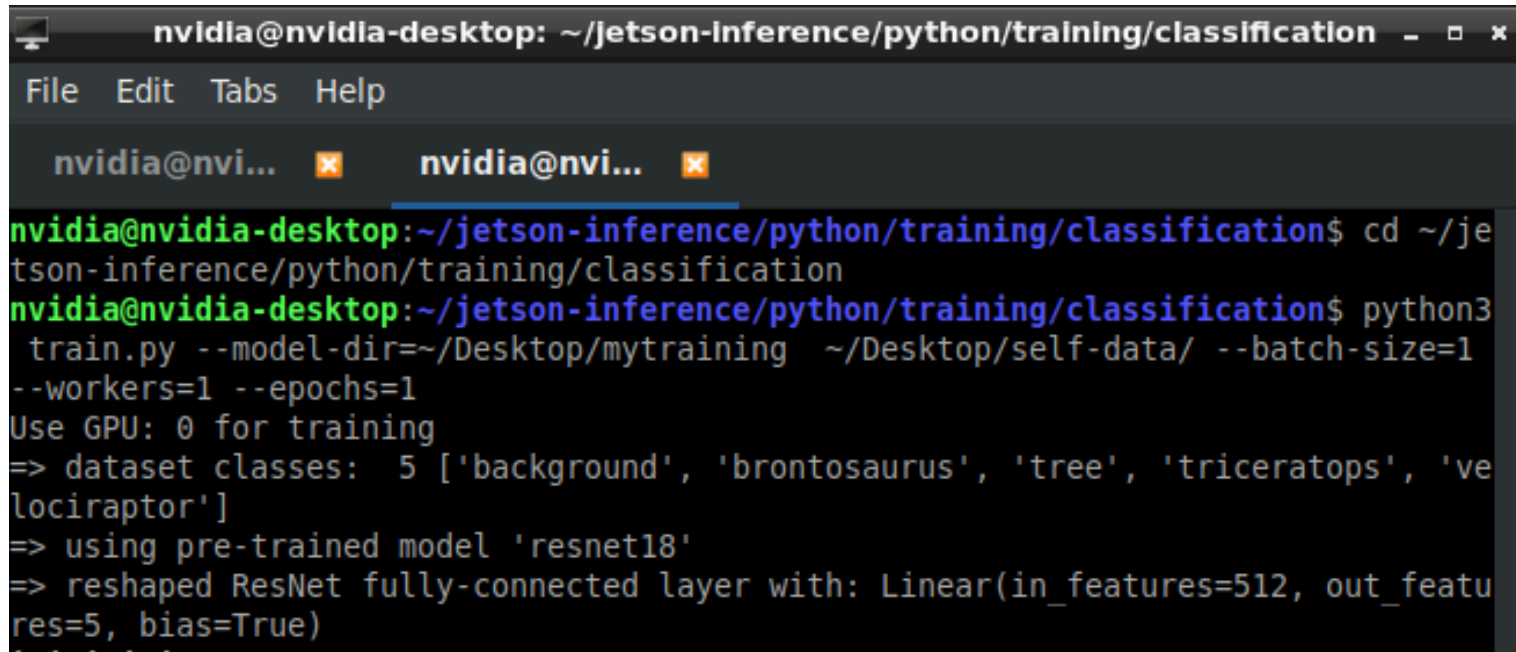
```
$ /home/nvidia/jetson-inference/build/aarch64/bin/imagenet --
```

```
model=models/cat_dog/resnet18.onnx --input_blob=input_0 --output_blob=output_0 --
```

```
labels=$DATASET/labels.txt $DATASET/test/dog/0016.jpg dog.jpg
```

- Training with your own data

After you've captured your own dataset, it's time to train your own model

A terminal window titled 'nvidia@nvidia-desktop: ~/jetson-inference/python/training/classification'. The window shows the execution of a Python script to train a model. The command executed is 'python3 train.py --model-dir=~/Desktop/mytraining ~/Desktop/self-data/ --batch-size=1 --workers=1 --epochs=1'. The output shows that the GPU 0 is used for training, the dataset classes are 5 (background, brontosaurus, tree, triceratops, velociraptor), and a pre-trained ResNet18 model is used with a reshaped fully-connected layer.

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

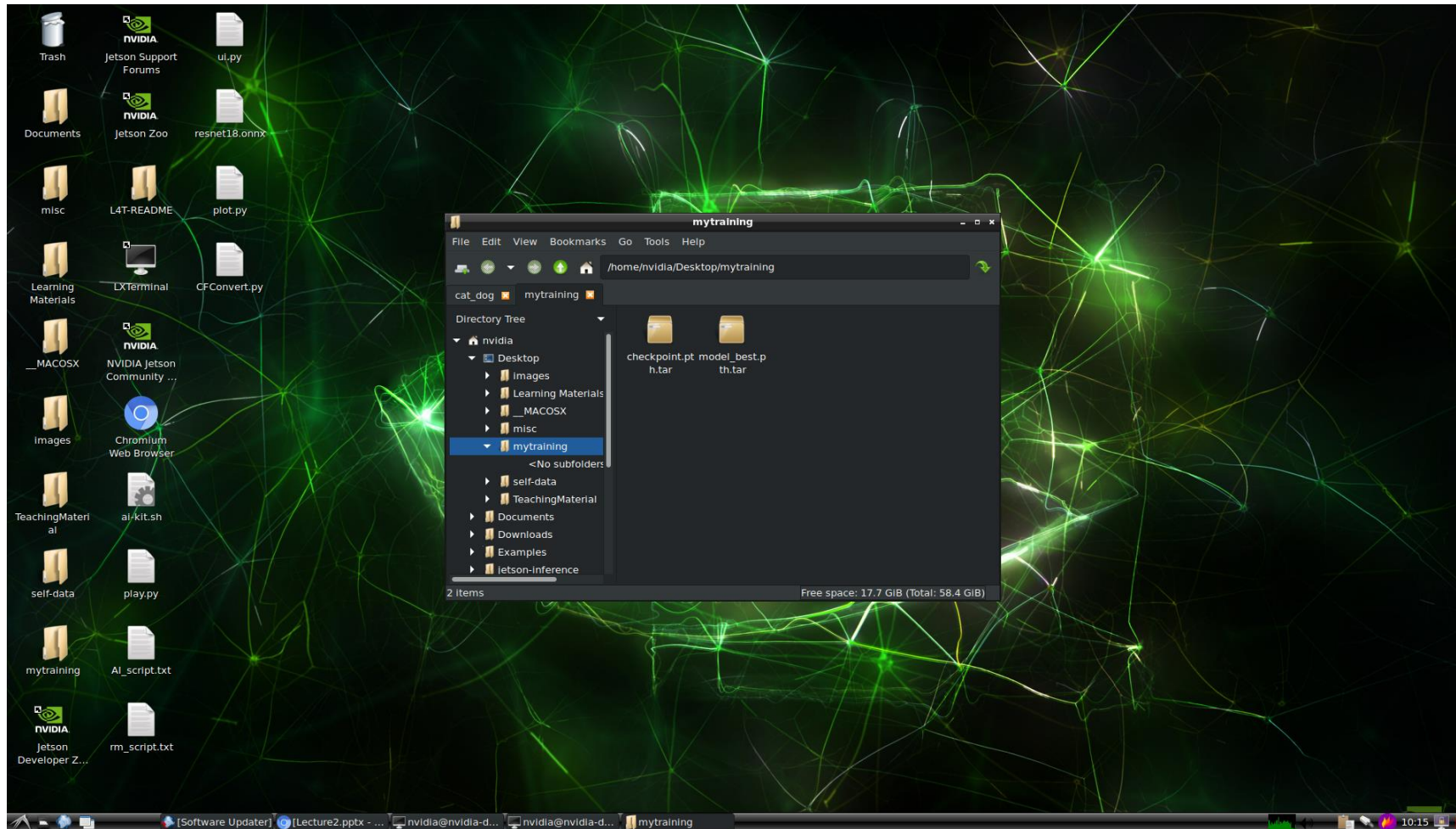
nvidia@nvi... x nvidia@nvi... x

nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ cd ~/je
tson-inference/python/training/classification
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ python3
train.py --model-dir=~/Desktop/mytraining ~/Desktop/self-data/ --batch-size=1
--workers=1 --epochs=1
Use GPU: 0 for training
=> dataset classes: 5 ['background', 'brontosaurus', 'tree', 'triceratops', 've
lociraptor']
=> using pre-trained model 'resnet18'
=> reshaped ResNet fully-connected layer with: Linear(in_features=512, out_featu
res=5, bias=True)
```

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

```
$ python3 train.py --model-dir=~/Desktop/mytraining ~/Desktop/self-data/ --
batch-size=1 --workers=1 --epoch=1
```


- Training with your own data

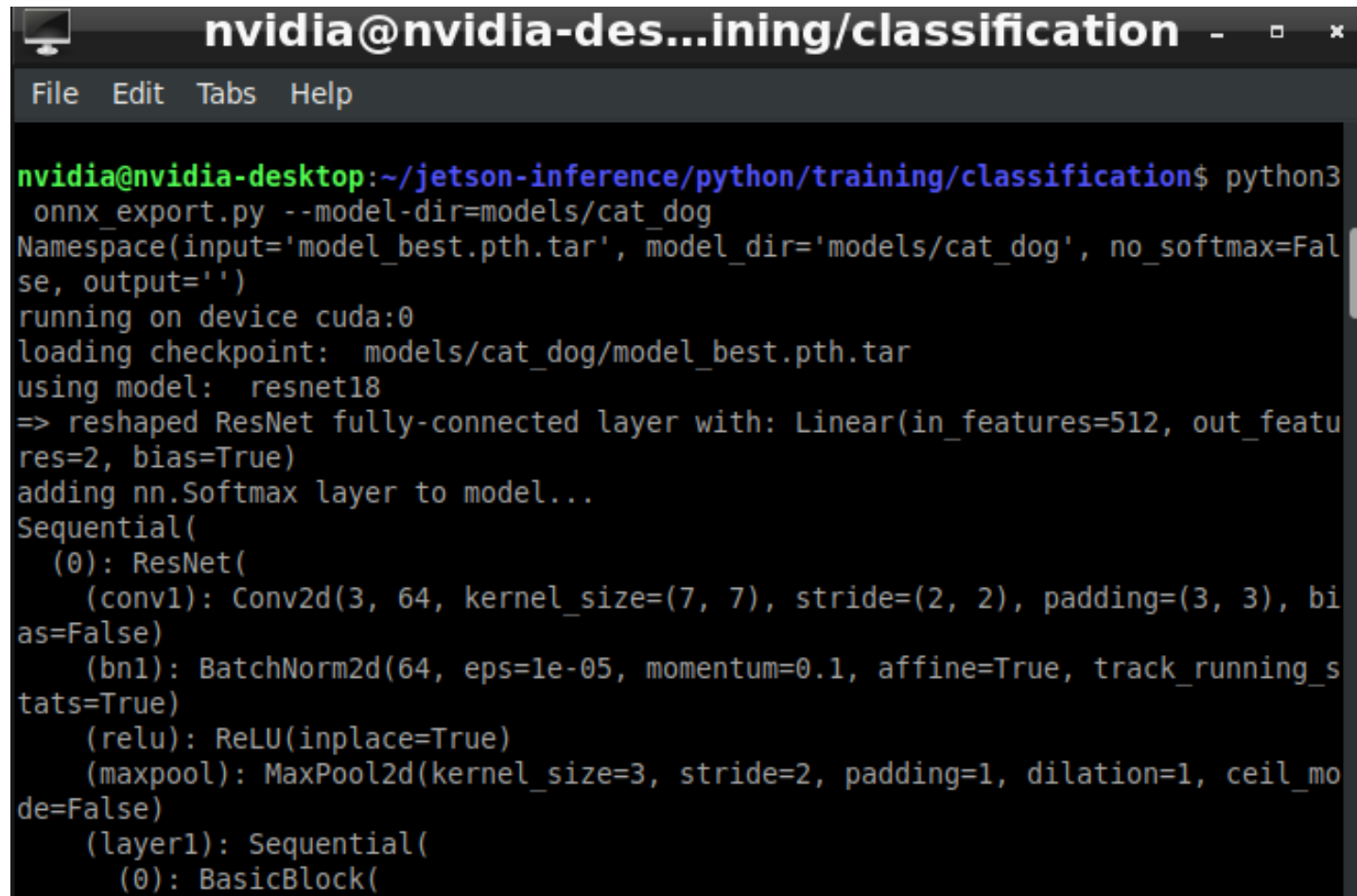


After finishing your training, you would find this two files in "mytraining"

- Converting the Model to ONNX

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

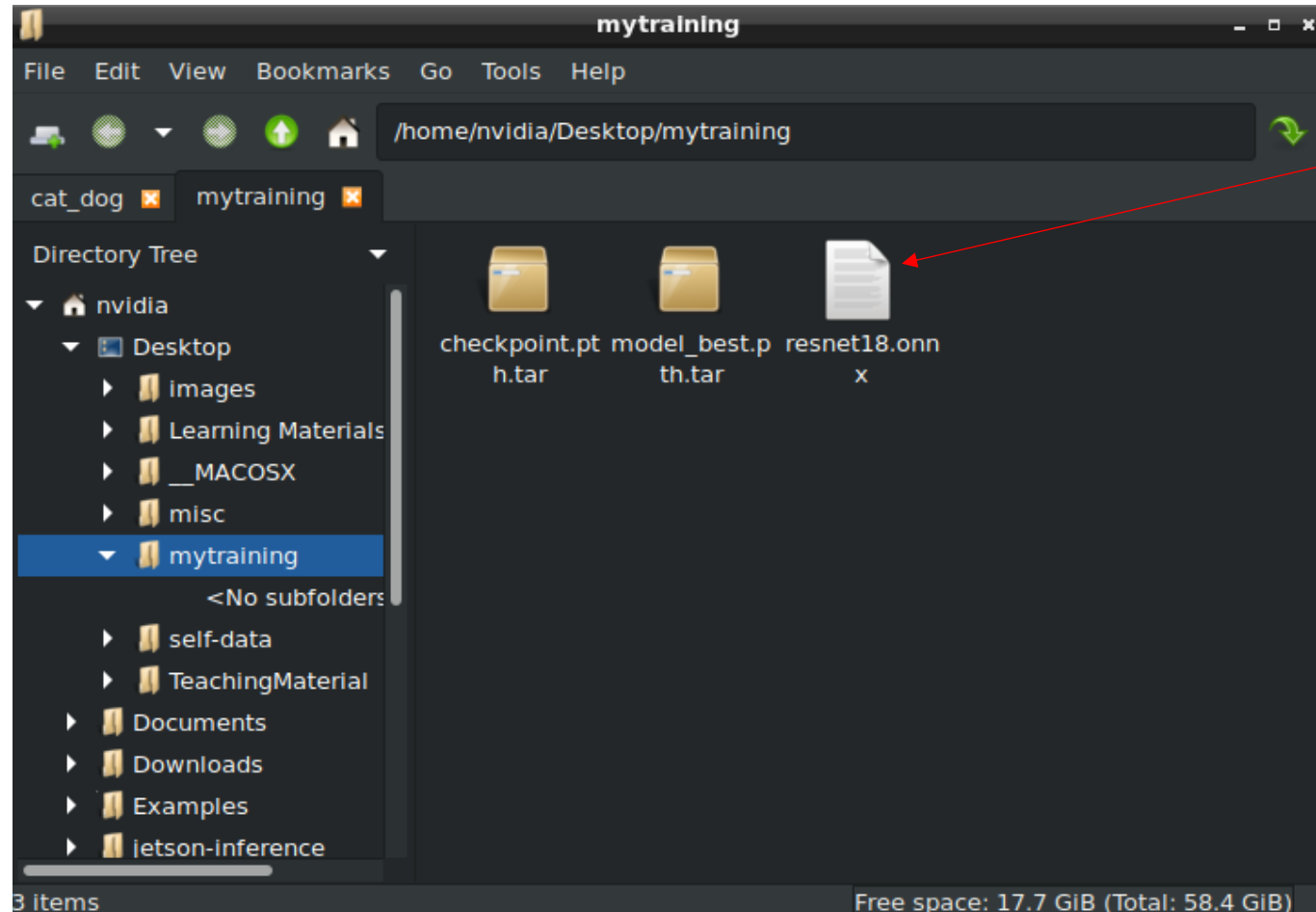
```
$ python3 onnx_export.py --model-dir=~/Desktop/mytraining
```

A terminal window titled 'nvidia@nvidia-des...ining/classification' with a menu bar (File, Edit, Tabs, Help). The terminal shows the execution of 'python3 onnx_export.py --model-dir=models/cat_dog'. The output includes: 'Namespace(input='model_best.pth.tar', model_dir='models/cat_dog', no_softmax=False, output='')', 'running on device cuda:0', 'loading checkpoint: models/cat_dog/model_best.pth.tar', 'using model: resnet18', and a detailed description of the model architecture: '=> reshaped ResNet fully-connected layer with: Linear(in_features=512, out_features=2, bias=True)', 'adding nn.Softmax layer to model...', and a 'Sequential' container with layers: '(0): ResNet((conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False), (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True), (relu): ReLU(inplace=True), (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False), (layer1): Sequential((0): BasicBlock(' data-bbox="192 323 752 981"/>

```
nvidia@nvidia-desktop:~/jetson-inference/python/training/classification$ python3
onnx_export.py --model-dir=models/cat_dog
Namespace(input='model_best.pth.tar', model_dir='models/cat_dog', no_softmax=False, output='')
running on device cuda:0
loading checkpoint: models/cat_dog/model_best.pth.tar
using model: resnet18
=> reshaped ResNet fully-connected layer with: Linear(in_features=512, out_features=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), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): BasicBlock(
```

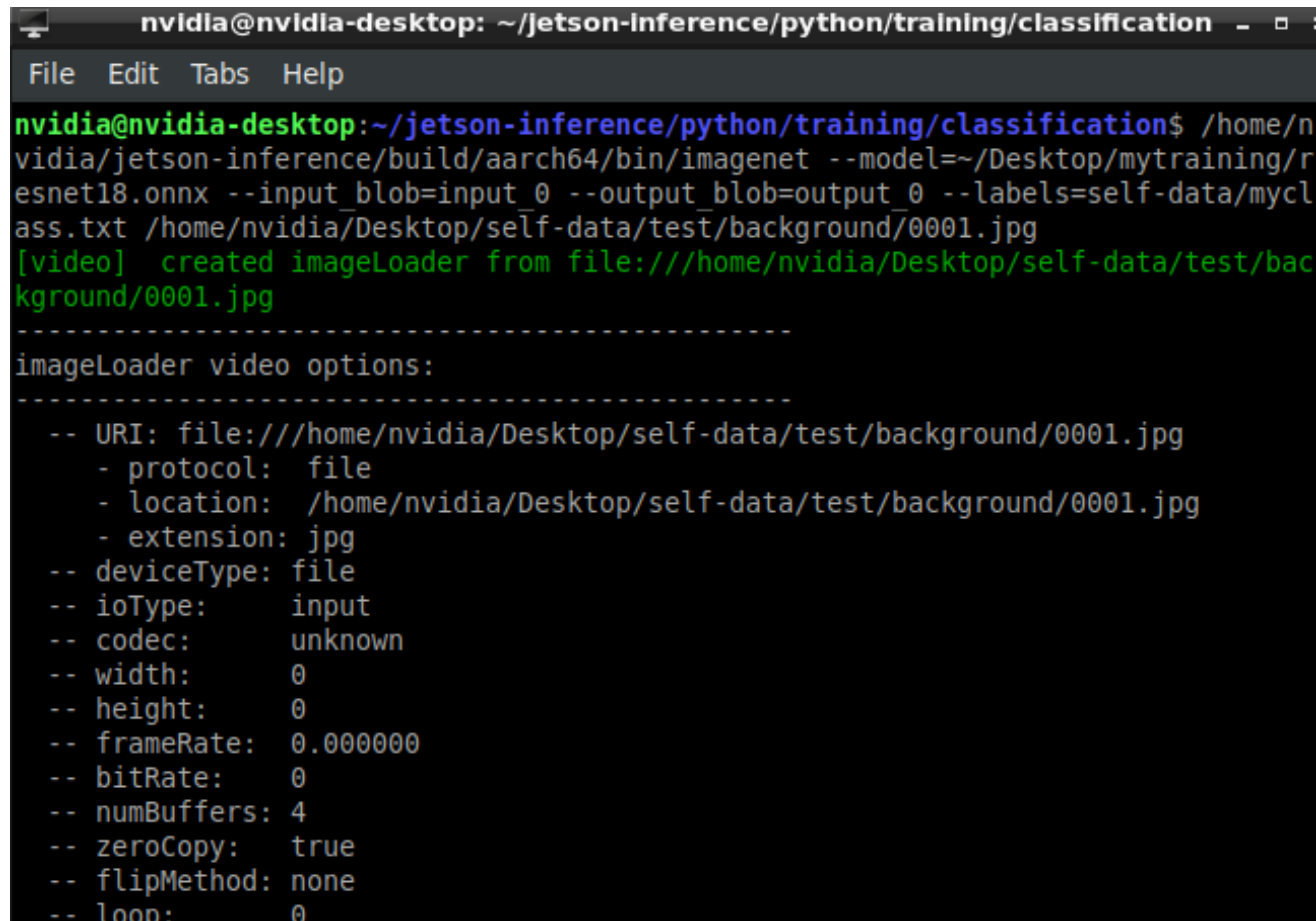
- Converting the Model to ONNX

This will create a model called resnet18.onnx under:
~/Desktop/mytraining



- Processing Images with TensorRT

`$ /home/nvidia/jetson-inference/build/aarch64/bin/imagenet --
model=~/Desktop/mytraining/resnet18.onnx --input_blob=input_0 --output_blob=output_0 --
labels=self-data/myclass.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
```

Assignment 2

Train you own model



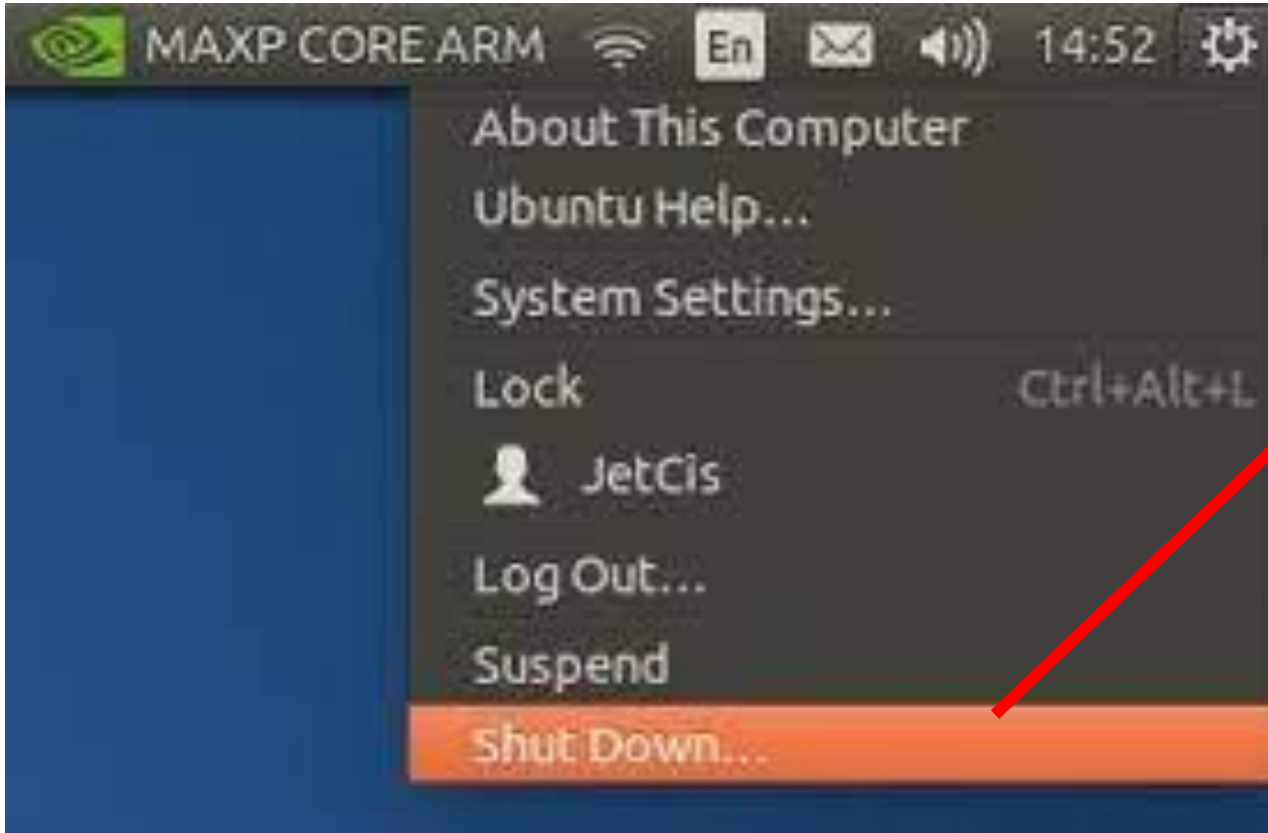
Collect your own dataset for more than three classes (new class can not same with your teammate&can not be cat/dag)/ Train a new model/ Output the prediction image



Upload the dataset (OneDrive link), result (tested image with result) and onnx model to the moodle

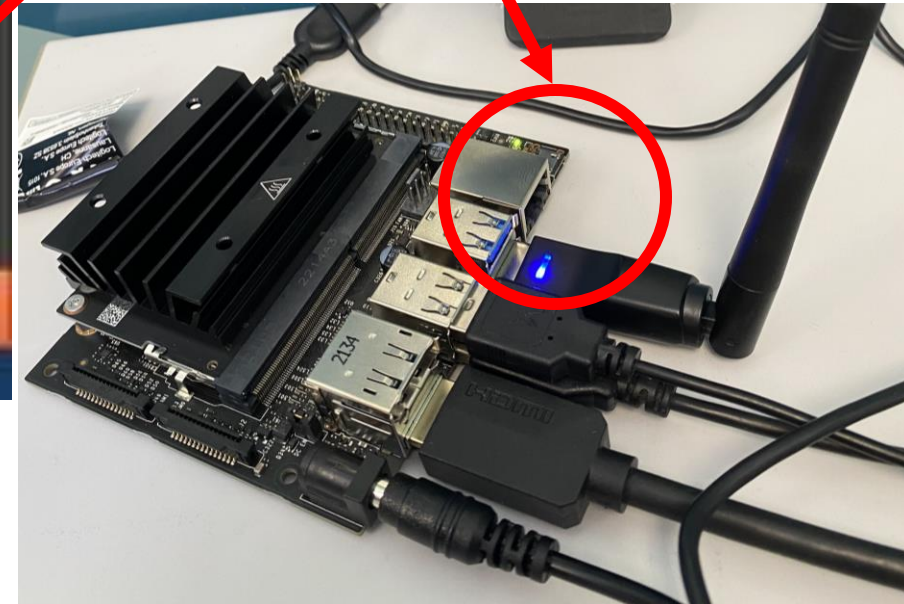


Deadline: 24/02/2025 11:59pm



Click on the **Shut Down button**, shut down the device like computer

After the **light turning off**, plug out the power supply



Pack it well for next time



Pack it well for next time