

**Tips:**

The Jetson NANO 2G board may experience memory overflow during training, causing the training program to freeze. It is recommended to use the 4GB version of the Jetson NANO board.

In this course, we will train our image classifier to detect 3 categories: scissors, rock, and cloth.

**1. About code**

Please check [Training\\_model](#) file.

**2. Run program on JupyterLab**

Open the [Training\\_model.ipynb](#) on JupyterLab.

The screenshot shows the JupyterLab interface with a file browser on the left and a code editor on the right. The file browser lists several files, including `on.ipynb` and `y`. The code editor displays the contents of `01_OpenCV_image_read_display.py`, which includes a copyright notice, author information, and Python code for image reading and model loading. A pink arrow points to the 'run' button (a play icon) in the code editor's toolbar, with the text 'run button' next to it. A yellow box with a pink border contains the text 'These programs have been successfully run' and has arrows pointing to the execution status of cells [3] and [4]. Another yellow box with a blue border contains the text 'This program is running, please be patient' and has an arrow pointing to the execution status of cell [\*].

```

@Copyright (C): 2010-2020, Shenzhen Yahboom Tech
@Author: Liusen
@Date: 2020-02-17 15:10:02
@LastEditors: Liusen
@LastEditTime: 2020-02-17 15:11:19

[ ]: import cv2

img = cv2.imread('yahboom.jpg', 1)
#cv2.imshow('image', img)
#cv2.waitKey(0)

[3]: image_widget = widgets.Image(format='jpg', width=320, height=240)
display(image_widget)

[4]: # Init tf model

MODEL_NAME = 'ssdlite_mobilenet_v2_coco_2018_05_09' #fast
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')
NUM_CLASSES = 90
IMAGE_SIZE = (12, 8)
fileAlreadyExists = os.path.isfile(PATH_TO_CKPT)

if not fileAlreadyExists:
    print('Model does not exist !')
    exit

[*]: # LOAD GRAPH
print('Loading...')
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')
    label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
    categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES)
    category_index = label_map_util.create_category_index(categories)
    print('Finish Load Graph...')

Loading...

```

### 3. Program analysis

#### 3.1 Import torch, torchvision and related libraries.

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms
```

We will use the **ImageFolder** dataset class in the **torchvision.datasets** library to create a dataset instance.

There is an additional **torchvision.transforms** library for transforming data.

#### 3.2 The data generated during training will be saved in the dataset folder of the project directory, such as one, two three folder.



If other folders are created, delete them, and then re-run the following program. The normal print data is shown below.

```
dataset = datasets.ImageFolder(
    'dataset',
    transforms.Compose([
        transforms.ColorJitter(0.1, 0.1, 0.1, 0.1),
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
)
print(dataset.class_to_idx )
{'one': 0, 'three': 1, 'two': 2}
```

#### 3.3 We divide the data set we just created into a training set and a test set. The test set is used to verify the accuracy of our model.

```
[4]: train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset) - 25, 25])
```

#### 3.4 Next, we create a data loader to load data in batches.

There are two data loaders: one is the training data loader and the other is the test data loader.

```
[4]: train_loader = torch.utils.data.DataLoader(  
    train_dataset,  
    batch_size=16,  
    shuffle=True,  
    num_workers=4  
)  
  
test_loader = torch.utils.data.DataLoader(  
    test_dataset,  
    batch_size=16,  
    shuffle=True,  
    num_workers=4  
)
```

3.5 Training a neural network. Using the code below, we will start training our neural network and save the best performing model after running each generation.

## Define the neural network

```
model = models.alexnet(pretrained=True)
```

```
model.classifier[6] = torch.nn.Linear(model.classifier[6].in_features, 3)
```

```
device = torch.device('cuda')  
model = model.to(device)
```

## Training a neural network

Using the code below, we will start training our neural network and save the best performing model after running each generation.

```
NUM_EPOCHS = 30
BEST_MODEL_PATH = 'gesture_model.pth'
best_accuracy = 0.0

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

for epoch in range(NUM_EPOCHS):

    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = F.cross_entropy(outputs, labels)
        loss.backward()
        optimizer.step()

    test_error_count = 0.0
    for images, labels in iter(test_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        test_error_count += float(torch.sum(torch.abs(labels - outputs.argmax(1))))

    test_accuracy = 1.0 - float(test_error_count) / float(len(test_dataset))
    print('%d: %f' % (epoch, test_accuracy))
    if test_accuracy > best_accuracy:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_accuracy = test_accuracy
```

Wait patiently, when the model training is complete. We can see the model named `gesture_model.pth` generated in the directory.

3.6 Wait patiently, when the model training is complete. We can see the model named **`gesture_model.pth`** generated in the directory. As shown below.

