

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 `_v2_coco_2018_05_09`, `on.ipynb`, and `y`. The code editor displays the contents of `01_OpenCV_image_read_dis.py`, which includes a copyright notice, author information, and a code cell starting with `import cv2`. A pink arrow points to the 'run' button (a play icon) in the toolbar above the code editor.

Below the first screenshot, another part of the JupyterLab interface is shown, focusing on the execution of the `on.ipynb` file. The file browser on the left shows `on.ipynb` as 'seconds ago'. A pink box with the text 'These programs have been successfully run' points to the `on.ipynb` file. The code editor shows the execution of the `on.ipynb` file, with the output of the first cell (cell [3]) being `image_widget = widgets.Image(format='jpg', width=320, height=240)` and `display(image_widget)`. The output of the second cell (cell [4]) is the initialization of the TensorFlow model, including the model name, paths, and the loading of the graph. A yellow box with the text 'This program is running, please be patient' points to the execution of the second cell (cell [4]).

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)
```

```
[9]: 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
```

```
0: 1.000000
1: 1.000000
2: 1.000000
3: 1.000000
4: 1.000000
5: 1.000000
```

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.

