## 4.5 Pytorch AI Framework

### 4.5.1 About Pytorch

Pytorch is also one of the most popular frameworks in the AI world.

1、Easy to use API --- It's as simple as Python.

2、Python support --- PyTorch can be successfully integrated with the Python data science stack. It is very similar to numpy and can't even notice the difference.

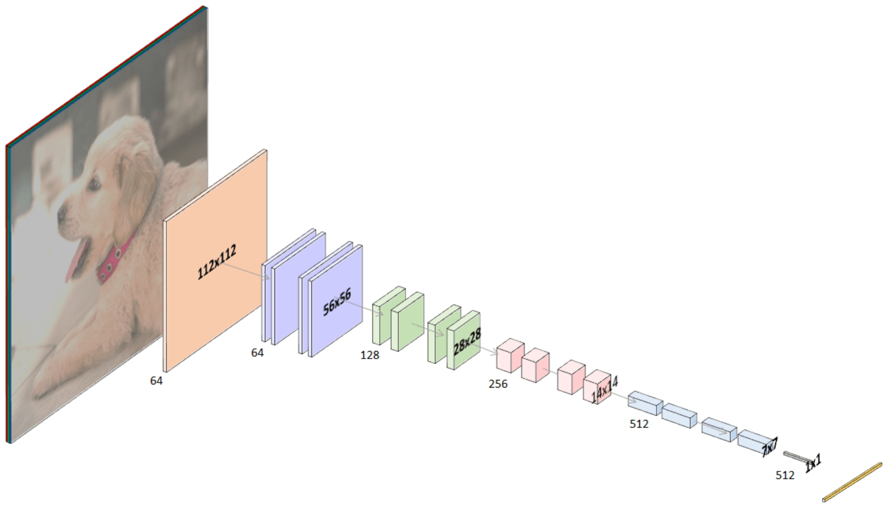
3、Dynamic Computational Graphs --- Instead of pre-defined graphics with specific functionality, PyTorch provides us with a framework to build computational graphs at runtime and even change them at runtime.

4、Some of the other benefits : multi-gpu support, custom data loaders and simplified pre-processors.

### 4.5.2 ResNet-18 model

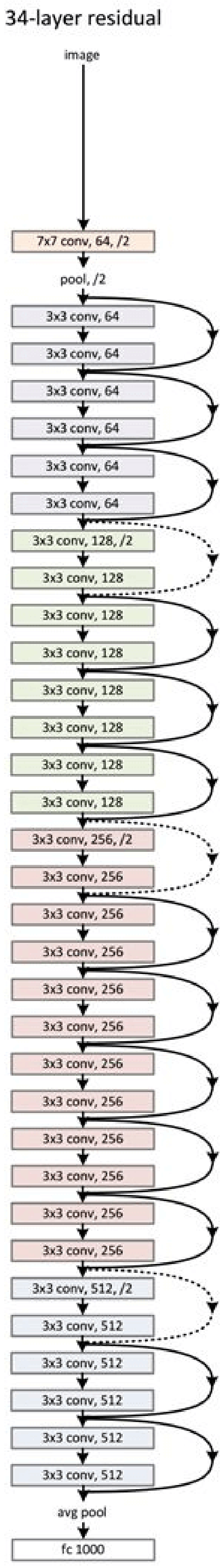
Application developers can use many world-class CNN architectures for image classification and image regression.

We will use the smallest version of ResNet: ResNet-18 in this project.



**Residual network:**

ResNet is a residual network consisting of building blocks that contain "shortcut connections" that skip one or more layers.



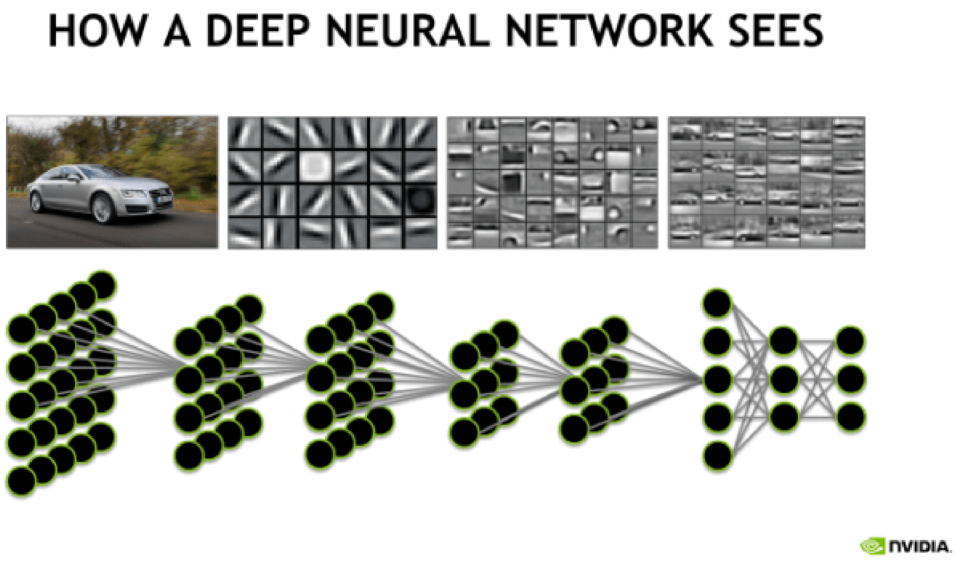
The shortcut output is added to the output of the skipped layer.

This technique makes the network easier to optimize and has greater accuracy when the depth is greatly increased.

The ResNet architecture ranges from 18 layers to 152 layers! For this project, the smallest network ResNet-18 provides a good balance of performance and efficiency for the Jetson Nano.

**To learning:**

PyTorch includes a pre-trained ResNet-18 model that is trained on the ImageNet 2012 classification dataset, which consists of 1000 classes. In other words, the model can already identify 1000 different objects!



We will adapt to our project by modifying the last neural network layer that makes up the ResNet-18 model, which contains less than 10 different classes.

The final layer of ResNet-18 is a fully connected (fc) layer that is aggregated and expanded into 512 inputs, each connected to 1000 possible output classes. We will replace the (512, 1000) layer with a class that matches us. For example, if we only need three classes, the last layer will become (512, 3), with each of the 512 inputs being fully connected to the three output classes.

You still need to train the network to identify these three classes using the images you collect, but since the network has learned to recognize the features that are common to most objects, model training has already done some of the work, can be reused, or "transferred" to yours. In the new project.

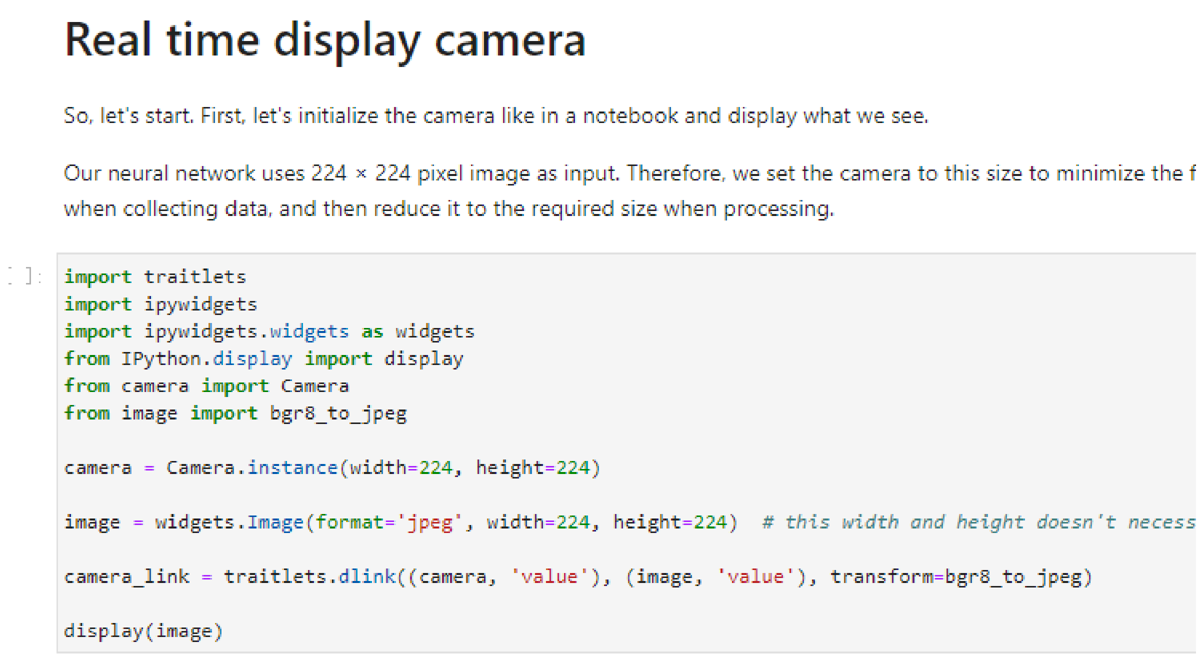
## 4.5.3 Pytorch initial using --- Gesture Recognition (thumb direction)

### 1、Gesture recognition data collection

The corresponding complete source code is located:

/home/jetson/Notebooks/English/2.Simple_gesture_recognition_for_the_first_experience_of_pytoch/Collect_data.ipynb

First, we initialize our camera and display it in real time. The image data we collected and its attributes are consistent with the currently set camera display image. Our neural network uses an image of 224×224 pixels as input. So we set the camera to this size to minimize the file size and minimize the data set. (We have tested that this pixel is suitable for this task) In some cases, it is better to use a larger image size when collecting data, and then reduce it to the required size when processing it, so here is reflected in our 4.2 .5 The importance of the zoom function mentioned in picture zoom (cv2.resize).



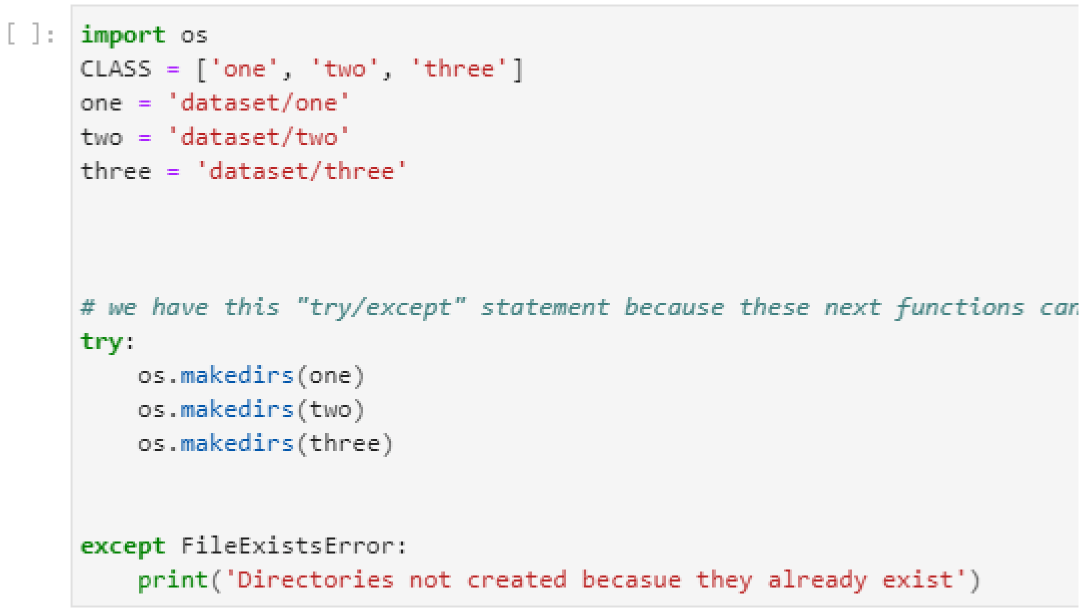
## Real time display camera

So, let's start. First, let's initialize the camera like in a notebook and display what we see.

Our neural network uses 224 × 224 pixel image as input. Therefore, we set the camera to this size to minimize the f
when collecting data, and then reduce it to the required size when processing.

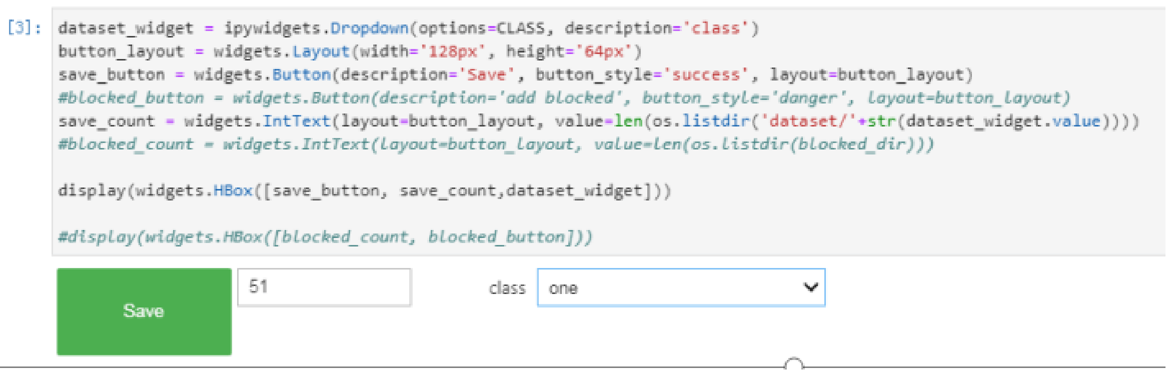
```
[ ]: import traitlets
     import ipywidgets
     import ipywidgets.widgets as widgets
     from IPython.display import display
     from camera import Camera
     from image import bgr8_to_jpeg

     camera = Camera.instance(width=224, height=224)

     image = widgets.Image(format='jpeg', width=224, height=224)  # this width and height doesn't necess

     camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg)

     display(image)
```

Next let's create some directories to store data. We will create a folder called dataset. There is a folder corresponding to the training pictures. If you run the cell code for the second time, the data storage folder created before the cell code already exists will prompt'Directories not created becasue they already exist'



```
[ ]: import os
     CLASS = ['one', 'two', 'three']
     one = 'dataset/one'
     two = 'dataset/two'
     three = 'dataset/three'



     # we have this "try/except" statement because these next functions car
     try:
         os.makedirs(one)
         os.makedirs(two)
         os.makedirs(three)


     except FileExistsError:
         print('Directories not created becasue they already exist')
```

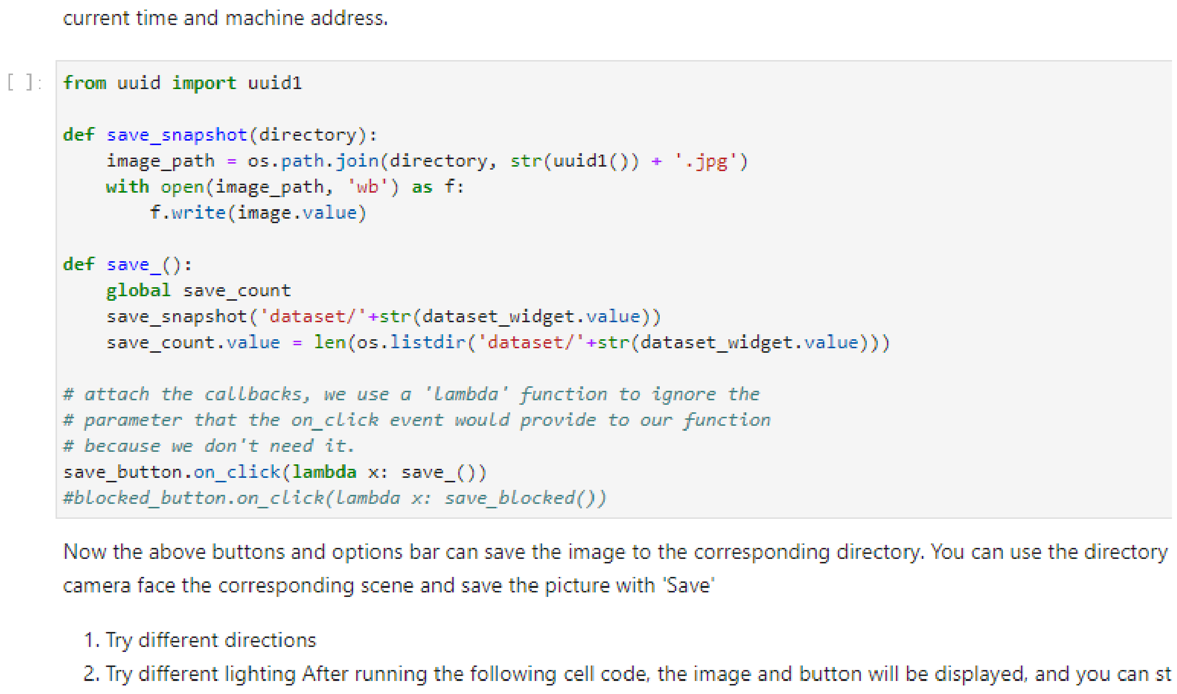
After running the code, the corresponding save button SAVE and will be displayed, select the option bar to save the picture to the corresponding folder, and a text box showing the number of pictures in the folder.



```
[3]: dataset_widget = ipywidgets.Dropdown(options=CLASS, description='class')
     button_layout = widgets.Layout(width='128px', height='64px')
     save_button = widgets.Button(description='Save', button_style='success', layout=button_layout)
     #blocked_button = widgets.Button(description='add blocked', button_style='danger', layout=button_layout)
     save_count = widgets.IntText(layout=button_layout, value=len(os.listdir('dataset/'+str(dataset_widget.value))))
     #blocked_count = widgets.IntText(layout=button_layout, value=len(os.listdir(blocked_dir)))

     display(widgets.HBox([save_button, save_count,dataset_widget]))

     #display(widgets.HBox([blocked_count, blocked_button]))
```

Now the button is only displayed, and no event is bound yet, we must attach a function function to save the image'on_click' event. We will save the value of the'Image' part (not the camera) because it is already in compressed JPEG format!

To ensure that no file names are repeated (even on different machines!), we will use the'uuid' package in python, which defines the'uuid1' method to generate unique identifiers. This unique identifier is generated from information such as the current time and machine address.



current time and machine address.

```
[ ]: from uuid import uuid1

     def save_snapshot(directory):
         image_path = os.path.join(directory, str(uuid1()) + '.jpg')
         with open(image_path, 'wb') as f:
             f.write(image.value)

     def save_():
         global save_count
         save_snapshot('dataset/'+str(dataset_widget.value))
         save_count.value = len(os.listdir('dataset/'+str(dataset_widget.value)))

     # attach the callbacks, we use a 'lambda' function to ignore the
     # parameter that the on_click event would provide to our function
     # because we don't need it.
     save_button.on_click(lambda x: save_())
     #blocked_button.on_click(lambda x: save_blocked())
```

Now the above buttons and options bar can save the image to the corresponding directory. You can use the directory camera face the corresponding scene and save the picture with 'Save'

1. Try different directions
2. Try different lighting After running the following cell code, the image and button will be displayed, and you can st

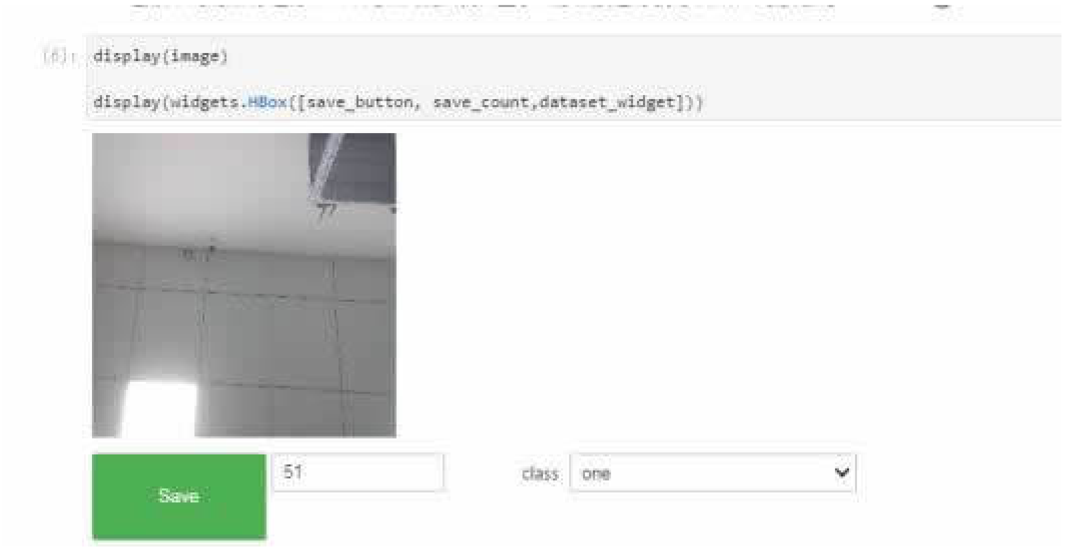
Now the above button and option bar can save the image to the corresponding directory. You can use the directory file browser on the left side of Jupyter to view these files!

Now continue to collect some data, select a different class, let the camera face the corresponding scene, and save the picture with save:

1. Try different directions

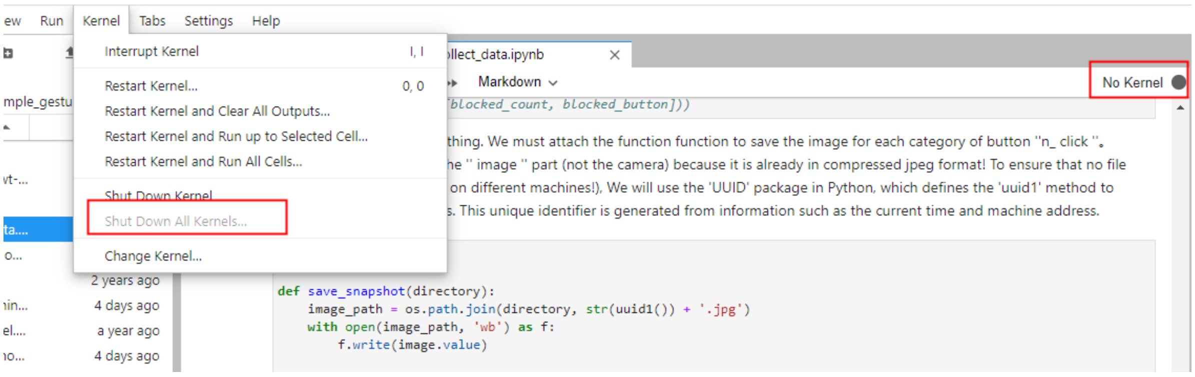
2. Try different lighting

After running the cell code below, images and buttons will be displayed, and you can start collecting data



```
display(image)

display(widgets.HBox([save_button, save_count,dataset_widget]))
```

Finally, before starting or using other programs or needing to call the camera again in some places, you need to close the current program, you need to shut down all kernels first, and then wait for no kernels on the right.

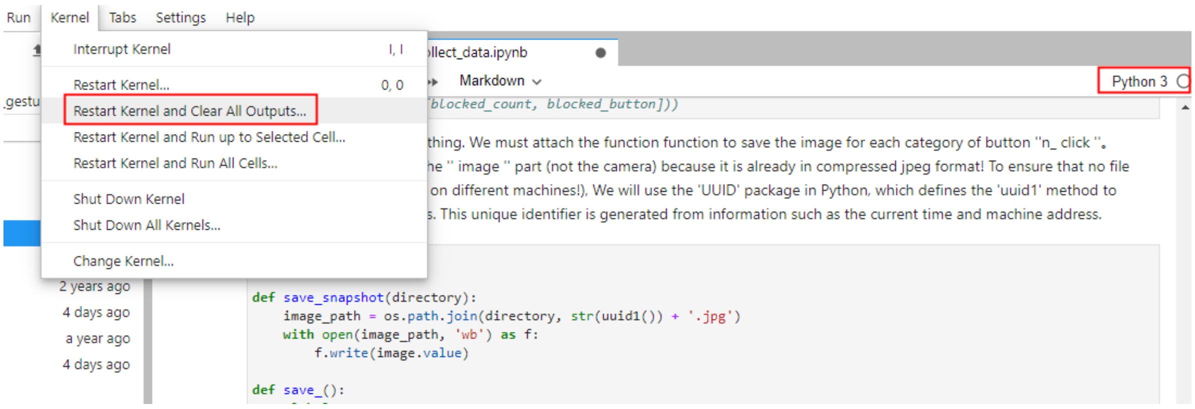


```
blocked_count, blocked_button]))
```

thing. We must attach the function function to save the image for each category of button "n_ click ".
he " image " part (not the camera) because it is already in compressed jpeg format! To ensure that no file
on different machines!), We will use the 'UUID' package in Python, which defines the 'uuid1' method to
s. This unique identifier is generated from information such as the current time and machine address.

```
def save_snapshot(directory):
    image_path = os.path.join(directory, str(uuid1()) + '.jpg')
    with open(image_path, 'wb') as f:
        f.write(image.value)
```

After restart kernel and clear output, wait for the right side to become python3. If the camera is still occupied, it is recommended to restart



```
blocked_count, blocked_button]))
```

thing. We must attach the function function to save the image for each category of button "n_ click ".
he " image " part (not the camera) because it is already in compressed jpeg format! To ensure that no file
on different machines!), We will use the 'UUID' package in Python, which defines the 'uuid1' method to
s. This unique identifier is generated from information such as the current time and machine address.

```
def save_snapshot(directory):
    image_path = os.path.join(directory, str(uuid1()) + '.jpg')
    with open(image_path, 'wb') as f:
        f.write(image.value)

def save_():
```

## 2、Model training for gesture recognition

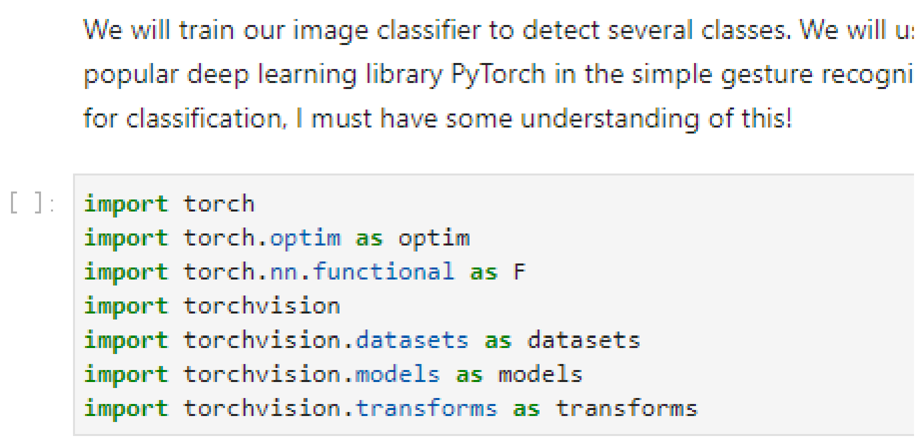
**Note: Jetson nano 2g has insufficient memory, and the training model may be stuck. It is recommended to train with other versions with higher memory or use the trained model**

The corresponding complete source code is located:

/home/jetson/Notebooks/English/2.Simple_gesture_recognition_for_the_first_experience_of_pytorch/Training_model.ipynb

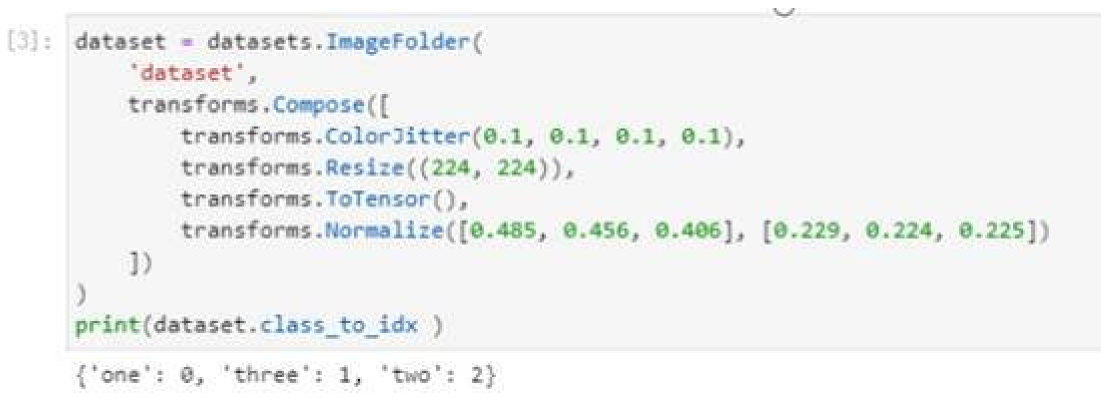
Here, we will train our image classifier to detect 3 classes to detect which one of scissors, rock, or cloth is.

Import related libraries used by torch and torchvision


We will train our image classifier to detect several classes. We will u
popular deep learning library PyTorch in the simple gesture recogni
for classification, I must have some understanding of this!

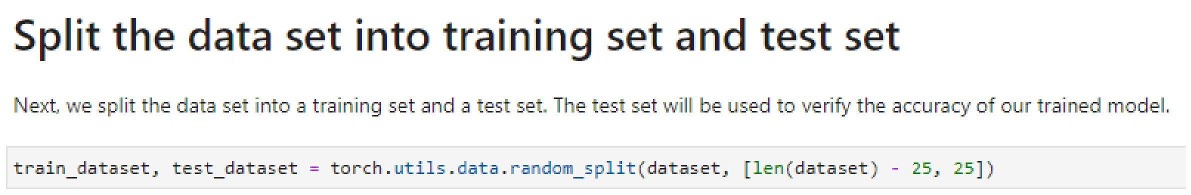
```
[ ]: import torch
     import torch.optim as optim
     import torch.nn.functional as F
     import torchvision
     import torchvision.datasets as datasets
     import torchvision.models as models
     import torchvision.transforms as transforms
```

Now we 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 in preparation for training the model. , If the additional folder name is printed out, delete the additional folder and re-run the following code. For example, our dataset only has three folders "one", "two", and "three", which should be {' one': 0,'three': 1,'two': 2}



```
[3]: 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}
```

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

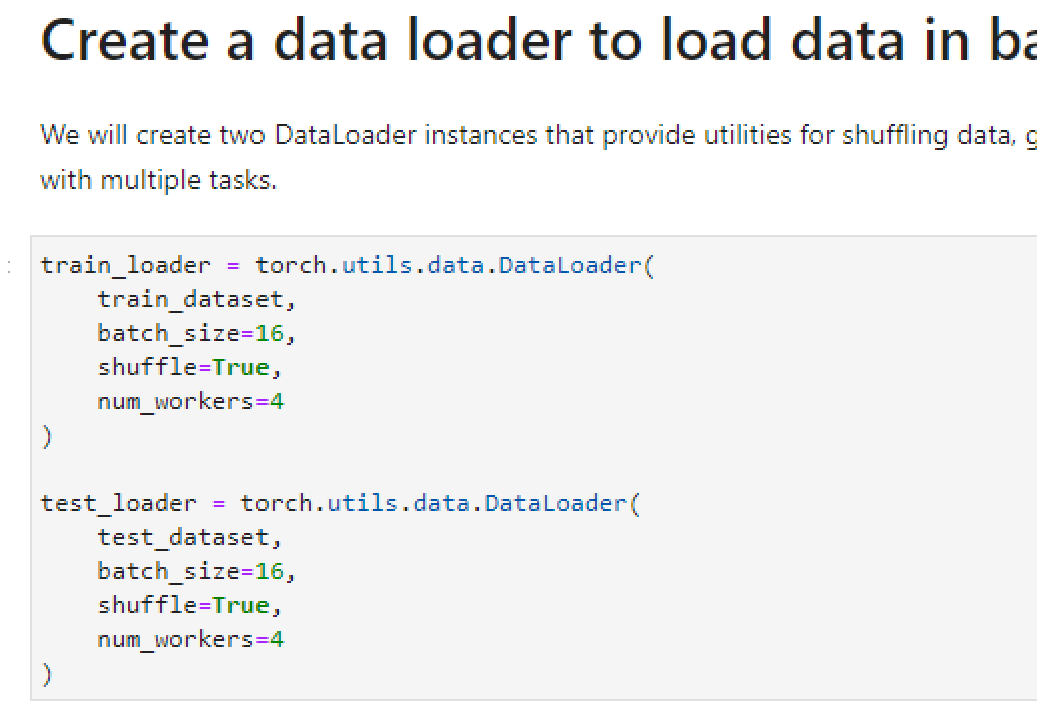


## Split the data set into training set and test set

Next, we split the data set into a training set and a test set. The test set will be used to verify the accuracy of our trained model.

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

Next, we create a data loader to load data in batches. There are still two data loaders, one is the training data loader and the other is the test data loader:

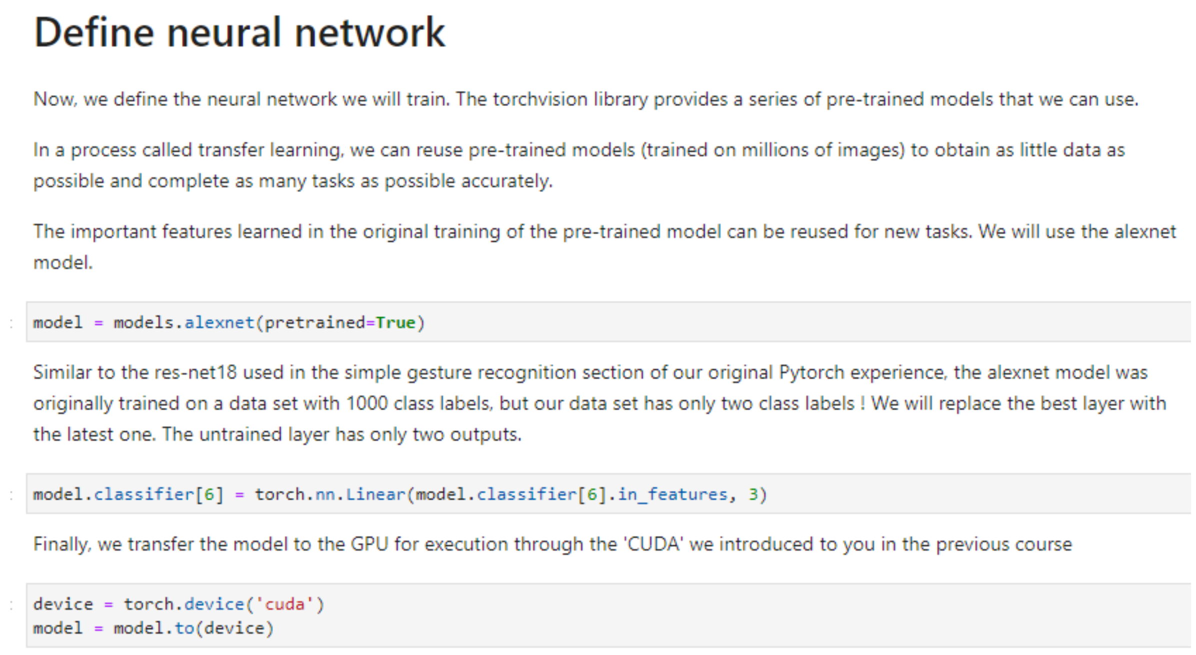


## Create a data loader to load data in ba

We will create two DataLoader instances that provide utilities for shuffling data, g
with multiple tasks.

```
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
)
```

Now, we define the neural network we will train. The torchvision library provides a series of pre-trained models that we can use. In a process called transfer learning or transfer learning, we can reuse the pre-trained model (trained on millions of images) to obtain as little data as possible and accurately complete as many tasks as possible. The important features learned in the original training of the pre-trained model can be reused for new tasks. Here we will use another model, called the alexnet model, which is similar to the res-net18 used in the simple gesture recognition that we experienced at the beginning of Pytorch. The alexnet model was originally for a data set with 1000 class labels. For training, but our data set only has three class labels! We will replace the best layer with the latest one. The untrained layer has only three outputs. Note that the pre-training model file alexnet-owt-4df8aa71.pth needs to be in the path to perform model training. The model can be obtained in the course:/home/jetson/Notebooks/English/2.Simple_gesture_recognition_for_the_first_experience_of_pytoch.



## Define neural network

Now, we define the neural network we will train. The torchvision library provides a series of pre-trained models that we can use.

In a process called transfer learning, we can reuse pre-trained models (trained on millions of images) to obtain as little data as possible and complete as many tasks as possible accurately.

The important features learned in the original training of the pre-trained model can be reused for new tasks. We will use the alexnet model.

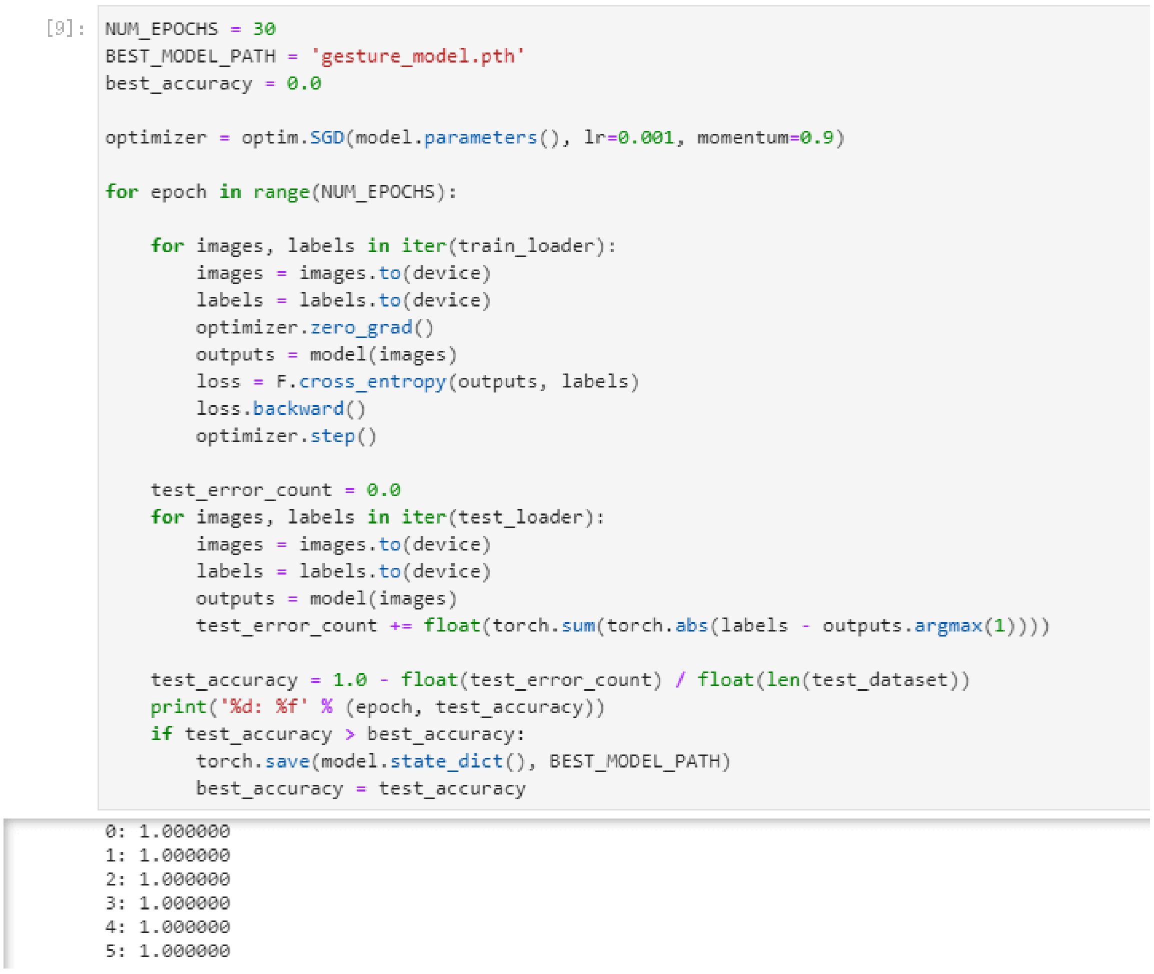
```
model = models.alexnet(pretrained=True)
```

Similar to the res-net18 used in the simple gesture recognition section of our original Pytorch experience, the alexnet model was originally trained on a data set with 1000 class labels, but our data set has only two class labels ! We will replace the best layer with the latest one. The untrained layer has only two outputs.

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

Finally, we transfer the model to the GPU for execution through the 'CUDA' we introduced to you in the previous course

```
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
```

When the model is trained, we can see the model named gesture_model.pth generated in the directory

### 3、Model training for gesture recognition

The corresponding complete source code is located:

/home/jetson/Notebooks/English/2.Simple_gesture_recognition_for_the_first_experience_of_pytoch/Man_machine_guessing.ipynb

The routine uses the colorama module, if it is not installed, the follow-up code needs to be installed to run normally (the configured image does not need to be installed)

Install colorama (For colored text output):

```
pip3 install colorama
```

In the above two steps, we train the data collected by the “collect data” code into the gesture recognition model we need by running the “training model”. The model trained by the author here only collects a fixed background. Switching to other background recognition effects are also general. If you want to adapt to more backgrounds, you need to train in more scenarios.

## Load training model

After we train the data collected by the 'collected data' code into the obstacle avoidance model we need by running the 'training model', we now start to use this model step by step,

Execute the following code to initialize the pytorch model

```
import torch
import torchvision

model = torchvision.models.alexnet(pretrained=False)
model.classifier[6] = torch.nn.Linear(model.classifier[6].in_features, 3)
```

Then we load the model we trained last time. If the effect of the model trained by ourselves is not ideal, it may be related to the number of data collections and the camera height and angle. Don't be too discouraged, collect more and test some scenes or just Recognize in the context of similar acquisition scenes.

Next, load the trained `gesture_model.pth` model you uploaded

```
model.load_state_dict(torch.load('gesture_model.pth'))
```

At present, the model weight calculation is located in the CPU memory. We will transfer the model to the GPU for execution through the 'CUDA' introduced to you in the previous course, and execute the following code to use the GPU.

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

Now we have loaded the model, but there is a small problem, that is, the image format of our camera needs to be exactly the same as the image format when training the model. To do this, we need to do some preprocessing. It is divided into the following steps:

1. Convert from BGR to RGB mode

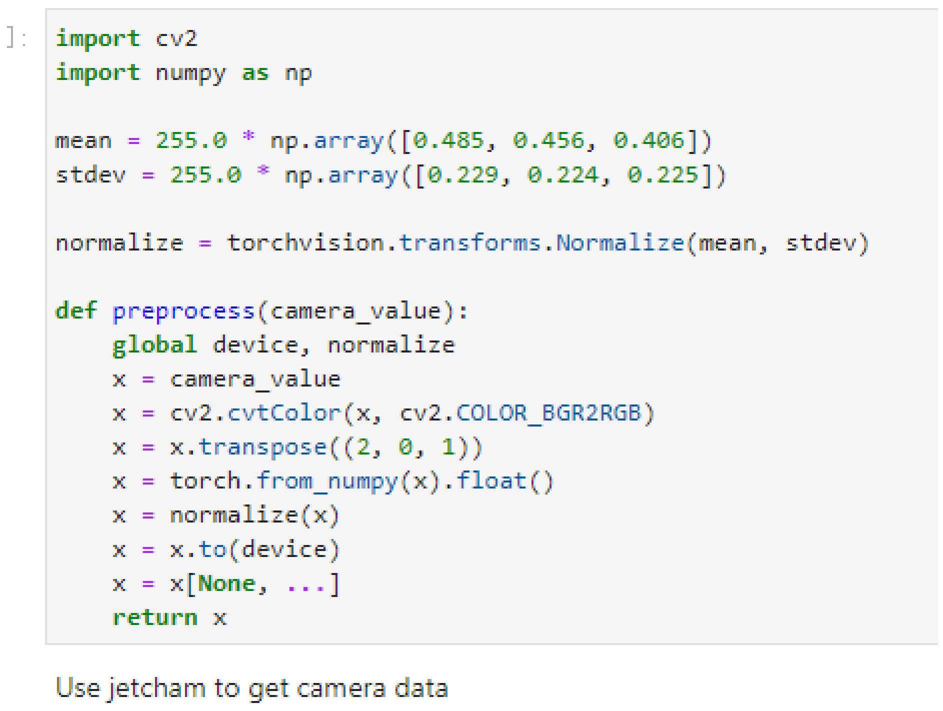
2. Convert from HWC layout to CHW layout

3. Use the same parameters as during training for normalization (our camera provides values in the range of [0,255], and trains the loaded images in the range of [0,1], so we need to scale 255.0

4. Transfer data from CPU memory to GPU memory

5. Add dimensions in bulk

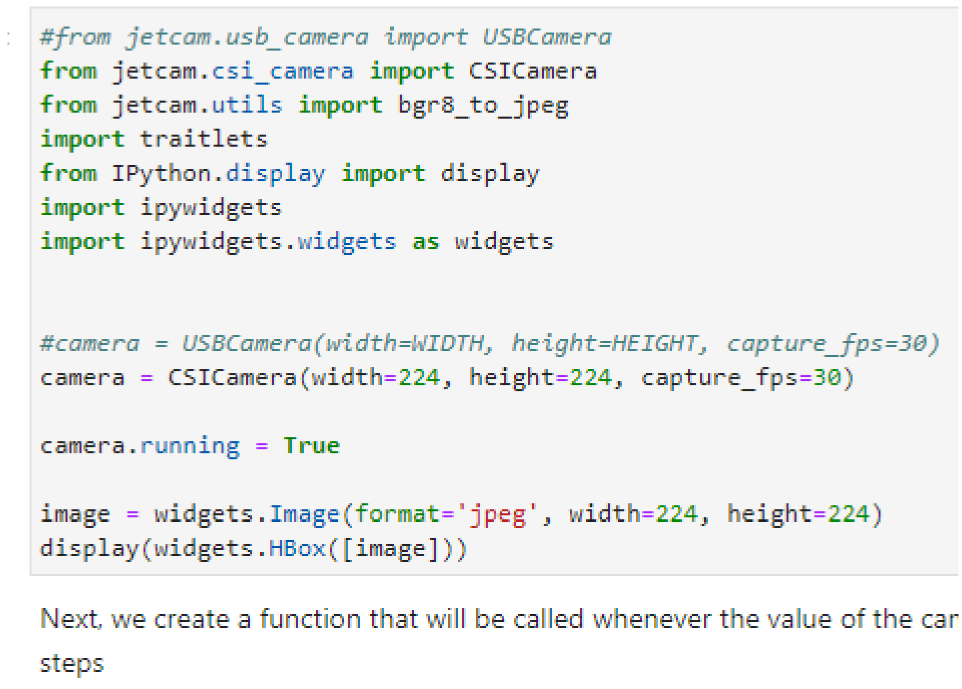
Run the following cell code to define the preprocessing function:



```
import cv2
import numpy as np

mean = 255.0 * np.array([0.485, 0.456, 0.406])
stdev = 255.0 * np.array([0.229, 0.224, 0.225])

normalize = torchvision.transforms.Normalize(mean, stdev)

def preprocess(camera_value):
    global device, normalize
    x = camera_value
    x = cv2.cvtColor(x, cv2.COLOR_BGR2RGB)
    x = x.transpose((2, 0, 1))
    x = torch.from_numpy(x).float()
    x = normalize(x)
    x = x.to(device)
    x = x[None, ...]
    return x
```

Use jetcham to get camera data

Here, due to the difference in 2G performance, the picture is prone to freeze, so the jetcham library is used to call the camera.



```
#from jetcam.usb_camera import USBCamera
from jetcam.csi_camera import CSICamera
from jetcam.utils import bgr8_to_jpeg
import traitlets
from IPython.display import display
import ipywidgets
import ipywidgets.widgets as widgets


#camera = USBCamera(width=WIDTH, height=HEIGHT, capture_fps=30)
camera = CSICamera(width=224, height=224, capture_fps=30)

camera.running = True

image = widgets.Image(format='jpeg', width=224, height=224)
display(widgets.HBox([image]))
```

Next, we create a function that will be called whenever the value of the car steps

Then create a function that will call this function whenever the value of the camera changes. This function will perform the following steps

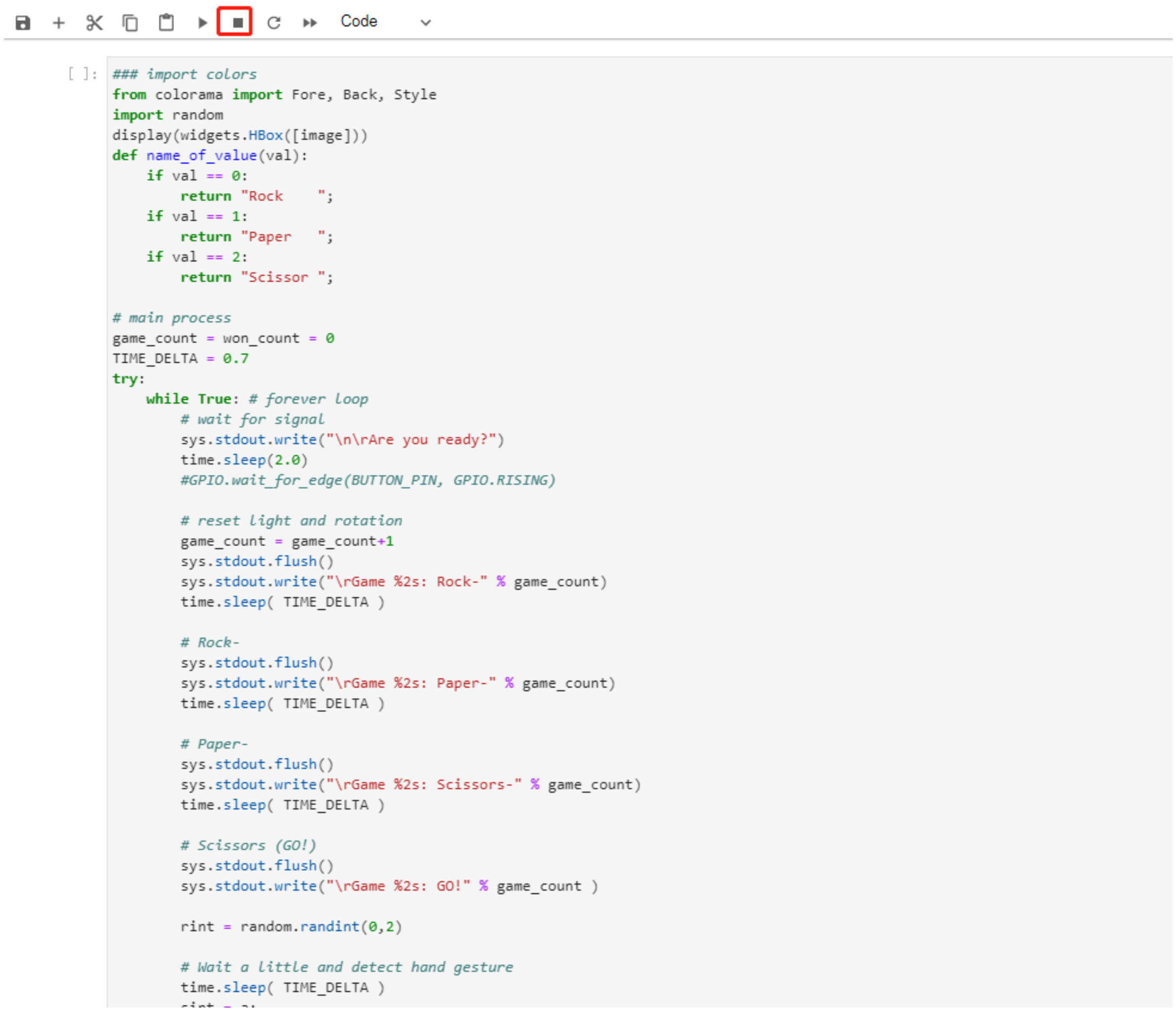
1. Preprocess camera images
2. Execute neural network
3. Compare the values of the 3 categories and assign the number to a

```python
import torch.nn.functional as F
import time
import sys

a=0
one_blocked=0.0
two_blocked=0.0
three_blocked=0.0

def update(change):
    global one_blocked, two_blocked, three_blocked,a
    x = change['new']
    image.value = bgr8_to_jpeg(x[:, ::-1, :])
    x = preprocess(x)
    y = model(x)

    # we apply the `softmax` function to normalize the output vector so it sums to
#    y = F.softmax(y, dim=1).detach().cpu().numpy().flatten()
    y = F.softmax(y, dim=1)
    one_blocked = float(y.flatten()[0])
    two_blocked = float(y.flatten()[1])
    three_blocked = float(y.flatten()[2])
    if(one_blocked > two_blocked and one_blocked > three_blocked):
        a = 0
    elif(two_blocked > one_blocked and two_blocked > three_blocked):
        a = 1
    elif(three_blocked > one_blocked and three_blocked > one_blocked):
        a = 2



    '''index = y.argmax()
    if y[index]>0.70:
        prediction_widget.value = hand[index]
    else:
        prediction_widget.value = hand[0]'''


    time.sleep(0.001)

update({'new': camera.value})  # we call the function once to intialize
```

We have created the neural network to perform the function, but now we need to attach it to the camera for processing. We use the `observe` function to complete this process.

```python
[ ]: camera.observe(update, names='value')  # this attaches the 'update' function to the 'value' traitlet of our camera
```

Start using the rock-paper-scissors game code, the following code is a loop, if you need to stop, you need to choose to stop

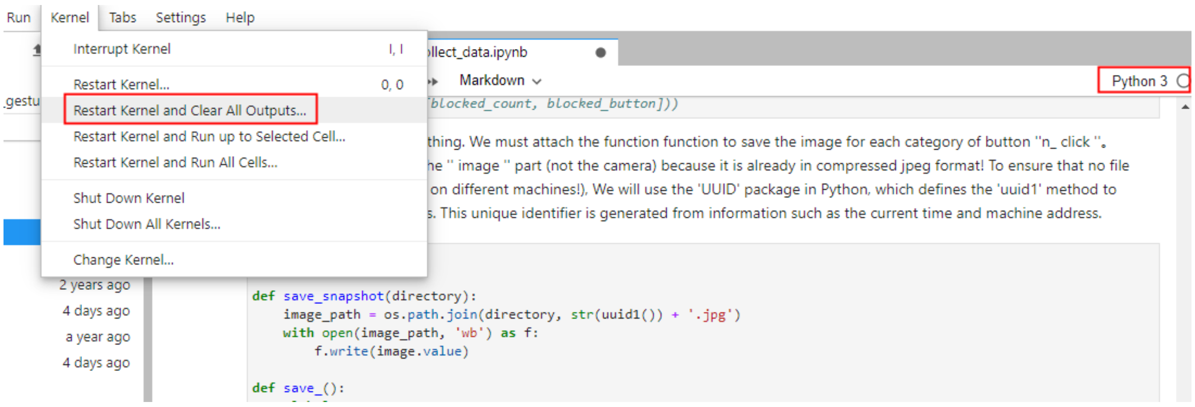


```
[ ]: ### import colors
     from colorama import Fore, Back, Style
     import random
     display(widgets.HBox([image]))
     def name_of_value(val):
         if val == 0:
             return "Rock    ";
         if val == 1:
             return "Paper   ";
         if val == 2:
             return "Scissor ";

     # main process
     game_count = won_count = 0
     TIME_DELTA = 0.7
     try:
         while True: # forever loop
             # wait for signal
             sys.stdout.write("\n\rAre you ready?")
             time.sleep(2.0)
             #GPIO.wait_for_edge(BUTTON_PIN, GPIO.RISING)

             # reset light and rotation
             game_count = game_count+1
             sys.stdout.flush()
             sys.stdout.write("\rGame %2s: Rock-" % game_count)
             time.sleep( TIME_DELTA )

             # Rock-
             sys.stdout.flush()
             sys.stdout.write("\rGame %2s: Paper-" % game_count)
             time.sleep( TIME_DELTA )

             # Paper-
             sys.stdout.flush()
             sys.stdout.write("\rGame %2s: Scissors-" % game_count)
             time.sleep( TIME_DELTA )

             # Scissors (GO!)
             sys.stdout.flush()
             sys.stdout.write("\rGame %2s: GO!" % game_count )

             rint = random.randint(0,2)

             # Wait a little and detect hand gesture
             time.sleep( TIME_DELTA )
```

After successfully running, the code will randomly generate a simple rock cloth, and then compare it with the corresponding gesture (rock-paper-scissors) based on the gesture recognized by you, and print out the computer's gesture name and the gesture name that recognizes you, and judge whether you win or lose At the same time, it will print out that the proportion of the 3 gestures is between 0-1. Quickly see how the effect of your own training is. If it is not ideal, you can train in multiple scenes or identify the scene you trained to see the effect.

```
Game  1  Opponent: Scissor  You: Scissor      R:0.411234, P:0.155385, S:0.433380      Draw
Game  2  Opponent: Rock     You: Paper        R:0.070242, P:0.786884, S:0.142873      You win!!
Game  3  Opponent: Paper    You: Paper        R:0.036942, P:0.895408, S:0.067650      Draw
Game  4  Opponent: Rock     You: Paper        R:0.353006, P:0.499509, S:0.147485      You win!!
Game  5  Opponent: Rock     You: Paper        R:0.202601, P:0.492071, S:0.305328      You win!!
Game  6  Opponent: Paper    You: Paper        R:0.332642, P:0.409246, S:0.258112      Draw
Game  7  Opponent: Scissor  You: Rock         R:0.693621, P:0.091128, S:0.215251      You win!!
Game  8  Opponent: Rock     You: Rock         R:0.632806, P:0.213825, S:0.153368      Draw
Game  9  Opponent: Paper    You: Paper        R:0.190023, P:0.433364, S:0.376612      Draw
Game 10  Opponent: Paper    You: Rock         R:0.487510, P:0.206356, S:0.306134      You lose
Game 11  Opponent: Scissor  You: Paper        R:0.326193, P:0.373969, S:0.299837      You lose
Game 12  Opponent: Scissor  You: Rock         R:0.452247, P:0.230600, S:0.317153      You win!!
Game 13  Opponent: Rock     You: Rock         R:0.491080, P:0.195816, S:0.313103      Draw
Game 14  Opponent: Paper    You: Rock         R:0.586139, P:0.203905, S:0.209956      You lose
Game 15  Opponent: Paper    You: Rock         R:0.814554, P:0.026020, S:0.159426      You lose
Game 16  Opponent: Rock     You: Rock         R:0.696741, P:0.013689, S:0.289570      Draw
Game 17  Opponent: Rock     You: Paper        R:0.395465, P:0.460628, S:0.143907      You win!!
Game 18  Opponent: Paper    You: Paper        R:0.159063, P:0.508364, S:0.332573      Draw
Game 19  Opponent: Rock     You: Paper        R:0.105390, P:0.766344, S:0.128266      You win!!
Game 20  Opponent: Rock     You: Scissor      R:0.146979, P:0.167586, S:0.685436      You lose
Game 21  Opponent: Scissor  You: Scissor      R:0.273961, P:0.101969, S:0.624070      Draw
Game 22  Opponent: Rock     You: Rock         R:0.524342, P:0.056651, S:0.419007      Draw
Game 23  Opponent: Rock     You: Paper        R:0.308945, P:0.419330, S:0.271725      You win!!
Game 24  Opponent: Paper    You: Scissor      R:0.078199, P:0.336727, S:0.585074      You win!!
Game 25  Opponent: Rock     You: Rock         R:0.382448, P:0.303195, S:0.314357      Draw
Game 26  Opponent: Paper    You: Paper        R:0.080430, P:0.480732, S:0.438839      Draw
Game 27  Opponent: Rock     You: Rock         R:0.537649, P:0.190170, S:0.272182      Draw
Game 28  Opponent: Rock     You: Scissor      R:0.416978, P:0.121089, S:0.461933      You lose
Are you ready?
```

Finally, you need to shut down the current program before starting or using other programs or calling the camera again in some places. You need to "shut down all kernels" first and then wait for "no kernels" on the right.

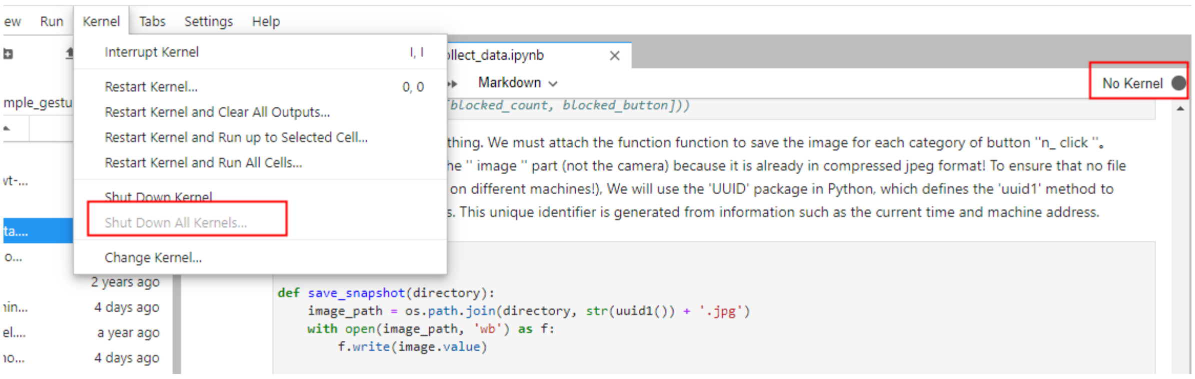


```
[blocked_count, blocked_button]))
```

thing. We must attach the function function to save the image for each category of button "n_ click ".
he " image " part (not the camera) because it is already in compressed jpeg format! To ensure that no file
on different machines!), We will use the 'UUID' package in Python, which defines the 'uuid1' method to
s. This unique identifier is generated from information such as the current time and machine address.

```
def save_snapshot(directory):
    image_path = os.path.join(directory, str(uuid1()) + '.jpg')
    with open(image_path, 'wb') as f:
        f.write(image.value)

def save_():
```

After "restart kernel and clear output", wait for the right side to change to "python3", if the camera is still occupied, it is recommended to restart



```
(blocked_count, blocked_button]))
```

thing. We must attach the function function to save the image for each category of button "n_ click "。
he " image " part (not the camera) because it is already in compressed jpeg format! To ensure that no file
on different machines!), We will use the 'UUID' package in Python, which defines the 'uuid1' method to
s. This unique identifier is generated from information such as the current time and machine address.

```
def save_snapshot(directory):
    image_path = os.path.join(directory, str(uuid1()) + '.jpg')
    with open(image_path, 'wb') as f:
        f.write(image.value)
```