3.5 Pytorch AI Framework

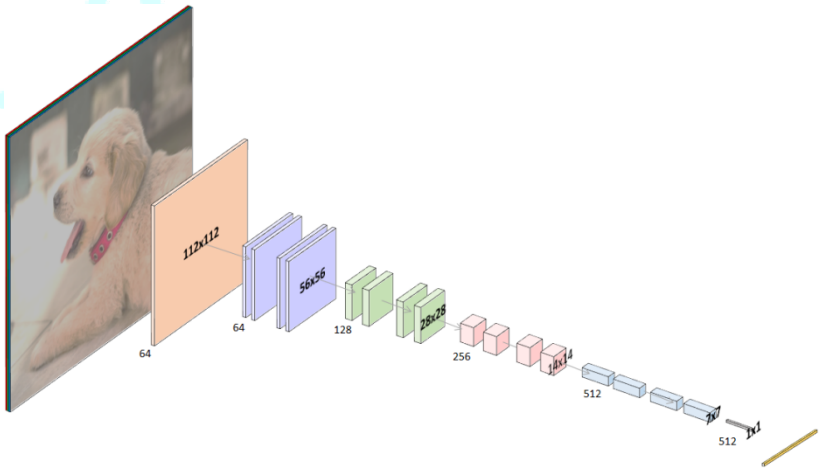
1. About Pytorch

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

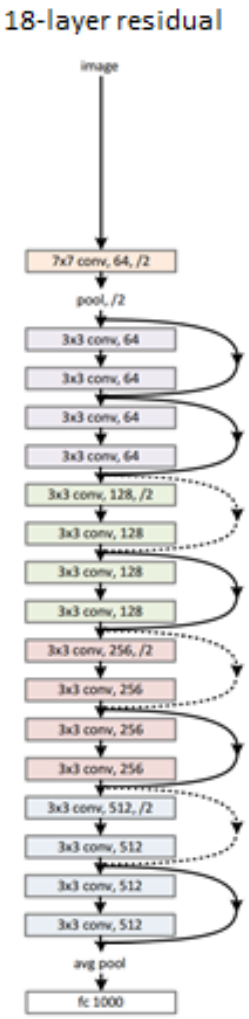
2. ResNet-18 model

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.



2.1 Residual network:

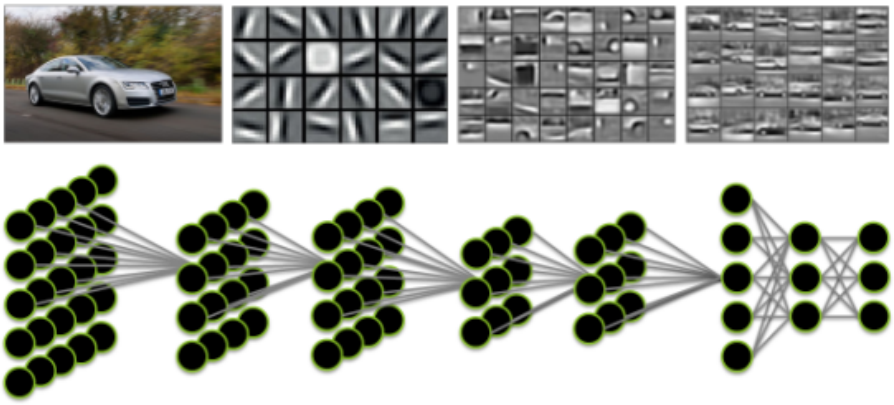
ResNet is a residual network consisting of building blocks that contain "shortcut connections" that skip one or more layers.



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.

2.2 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!



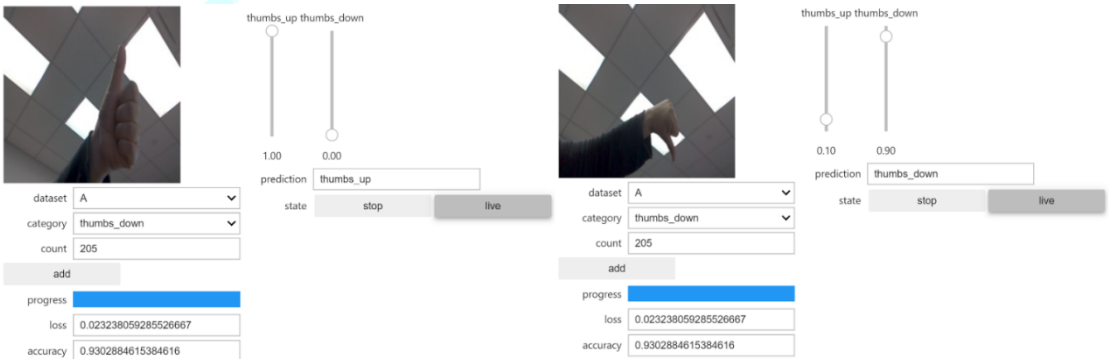
We will adapt to our project by modifying the last neural network layer that makes up the ResNet-18 model, which contains less than 10 different classes.

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.

3. Pytorch initial using --- Gesture Recognition (thumb direction)

The goal of this exercise is to build an image classification project that determines the meaning of gesture signals (such as thumbs up or thumbs down) held in front of the live camera.



3.1 interactive tool startup steps:

You will collect the data yourself, train the model to classify the data, and then test and update the model as needed to implement the project until it correctly

sorts the thumb up or thumbs down image before the live camera.

Step 1: Turn on your laptop

First, navigate to the category folder in the JupyterLab interface and double-click on the **classification_interactive.ipynb** notebook to open it.

Step 2: Execute all code blocks

The laptop is designed to reuse any sorting task you want to build. Step through the code blocks and execute one at a time. If you're having trouble performing this step, check out the information about JupyterLab.

1)camera

This block sets the size of the image and starts the camera. If your camera is already active in this or other notebooks, turn off the kernel in the active notebook before running this code unit. Be sure to choose the correct camera type (USB or CSI). This cell may take a few seconds to execute.

2) Mission

This block can define your TASK and CATEGORIES (class) parameters and the number of data sets to be tracked. For the Thumbs project, this has already been defined for you, so continue to execute the cell. Create a subdirectory of each class to store the sample images you collected. The subdirectory name is used as the label required by the model. This cell can be executed in just a few seconds.

3) Data collection

You will use the iPython widget to collect images of the category using the camera. This cell sets up a collection mechanism to calculate the image and generate a user interface. The widget built here is data_collection_widget. If you want to learn more about these powerful tools, please visit the ipywidgets documentation. This cell can be executed in just a few seconds.

4) Model

This block is where the neural network is defined.

First, use the following statement to select the GPU device:

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

The model is set to the ResNet-18 model for this project. Note that this pretrained=True parameter indicates that we are loading all the parameter weights of the trained Resnet-18 model, not just the neural network:

```
model = torchvision.models.resnet18(pretrained = True)
```

In addition to selecting the model, we also need to modify the last layer of the model to accept only the number of classes we are training. In the case of the thumb item, it is only 2 (ie the thumb is up and the thumb is down).

```
model.fc = torch.nn.Linear(512,len(dataset.categories))
```

This code unit may take a few seconds to execute.

5) On-site execution

This code block sets up threads that can run models in the background so that you can view real-time camera sources and display model performance in real time.

It also includes code that defines how to classify the output of the neural network.

The network generates some value for every possible category. The SOFTMAX function utilizes this carrier ķ (real number) and normalizes it to a probability distribution k(probability).

These values add up to 1 and can be interpreted as probabilities.

```
output = F.softmax(output,dim = 1).detach(). Cpu(). Numpy(). Flatten()
```

This cell can be executed in just a few seconds.

6) Training and evaluation

The training code unit sets the hyperparameters (period number, batch size, learning rate, momentum) of the model training and loads the image for training or evaluation.

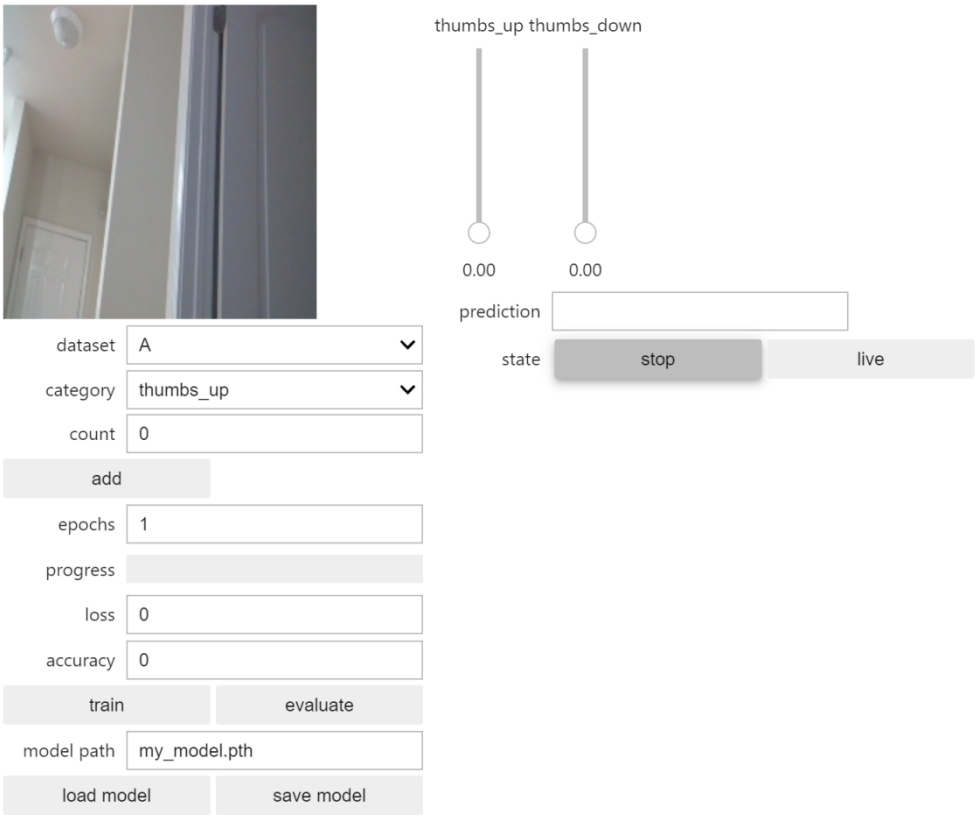
The model determines the predicted output of the loaded input image.

The difference between the predicted output and the actual label is used to calculate the "loss".

If the model is in training mode, the loss is propagated back to the network to improve the model.

7) Display interactive tools!

This is the last code cell. It is mainly used to package all widgets into a comprehensive tool and display it, it may take a few seconds to run.

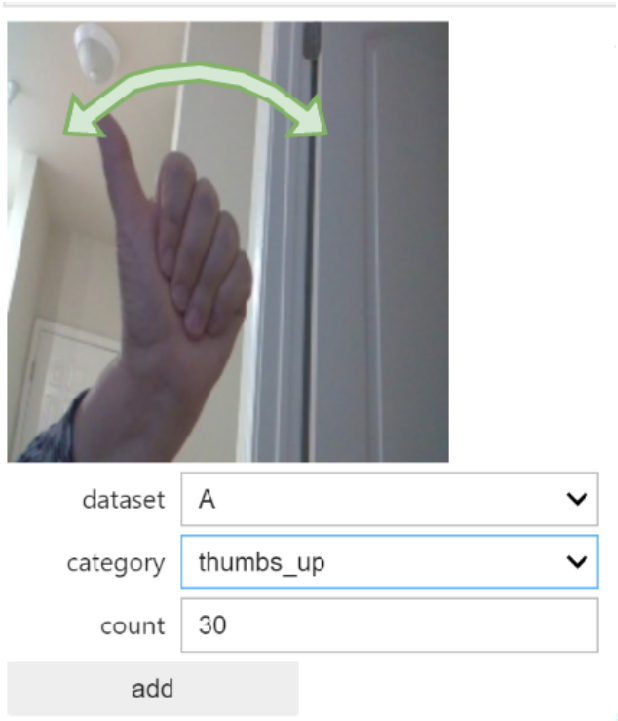


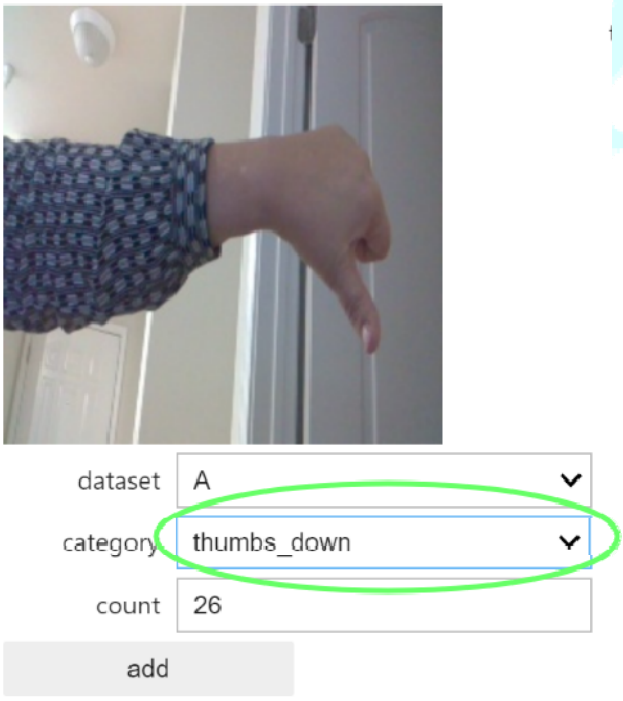
Step 3: Gather your initial data

The tool is designed for real-time interaction, so you can collect some data, train it, check the results, and then improve the model with more data and training.

We will try this one by one to understand the impact of the data you collect on the performance of the model. In each step, you will change the data in a new way and build the data set at any time.

Collect images of 30 thumbs-up images. When you click the Add button to save the data image, move your thumb to the arc at a generally upward angle in front of the camera.





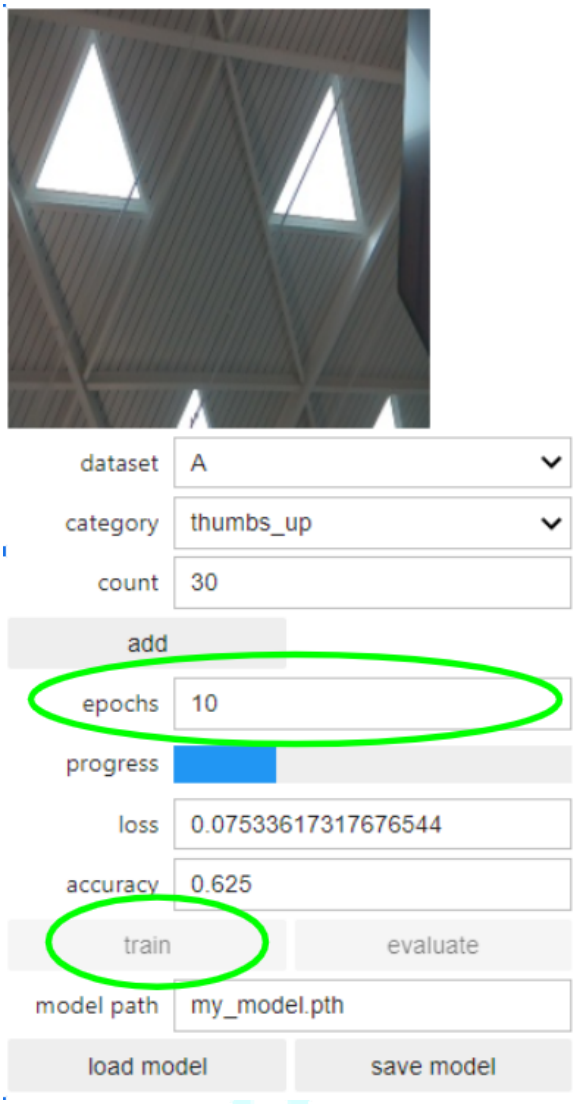
And collect 30 thumb images in the down position, and change the angle slightly when you click again. The goal is to provide the model with many different examples from each category so that predictions can be promoted.

Step 4: Train your initial data

Set the epoch number to 10 and click the "train" button. There is a delay of about 30 seconds when loading data.

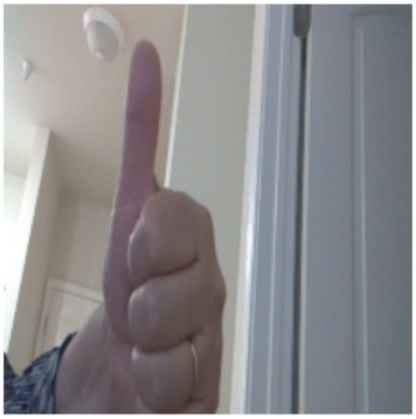
After that, the progress bar will show the status of the training for each period,

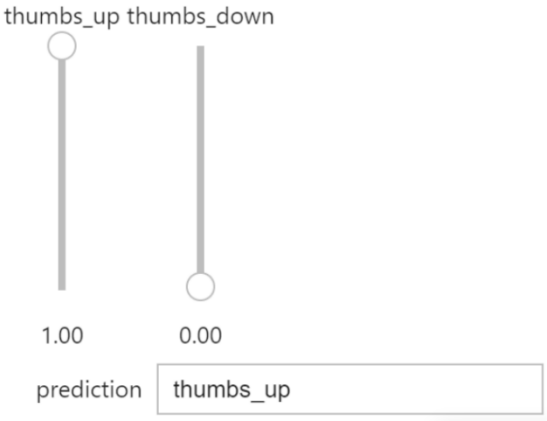
and you will also see the calculated loss and accuracy displayed.
! Note: Accuracy is based on testing of data that the model has already accessed, not data that is truly invisible.



Step 5: Test your data in real time
After completing the training, place the thumb up or down in front of the camera to observe the predicted probability and slider. The slider indicates the predicted probability given by the model.
Try moving the camera to a new background to see if it still works.

Note: If your camera seems to "freeze" at any time, you will need to close the kernel from the menu bar, then restart the kernel and run all the cells.
Your data has been saved, but you need to run the model training again.





Step 6: Improve your model

Using a different background, collect another 30 images for the thumb up and thumb down, change the angle again, and train for 5 cycles.

Use the camera's various distances to collect an additional 30 images for thumbs up and thumbs down, and train for 5 cycles.

Test and train in this way until you are satisfied with the performance of the first project!

Step 7: Save your model

If you are satisfied with the model, enter a name in the Model Path box and click “save Model” to save.

The corresponding complete source code is located:

/home/jetbot/Notebook/2.Pytorch-Gesture Recognition/Pytorch-Gesture Recognition.ipynb