A Learning Report of Action Recognition In Videos

Ao Dong
donga9@mcmaster.ca

1. Introduction

This is a report of the learning process on the deep Convolutional Neural Networks (CNN), including the Temporal Segment Networks (TSN)[1] and the Two-Stream Inflated 3D ConvNet (I3D)[2]. A brief introduction of some CNN pre-trained on the ImageNet dataset is included.

The report also contains brief tutorials organized by the author, describing how to implement machine learning training on the Google Cloud Platform (GCP), and how to use TSN and I3D to train the UCF101[3] and HMDB51[4] datasets with the help from GluonCV[5] toolkit.

My contribution: there is little academic contribution to the existing deep CNN. Instead, my major contribution is a detailed tutorial on how to practically learn and implement the existing CNN with the latest platforms and tools.

2. Deep Convolutional Networks

2.1. TSN

2.1.1. Problems

The paper *Temporal Segment Networks: Towards Good Practices for Deep Action Recognition* was trying to fix two problems existing around the year 2016:

Firstly, the absence of ConvNet frameworks dealing well with long-range temporal structure. Meanwhile, some new attempts tackling this problem required dense temporal sampling and massive computing resources.

Secondly, due to the limited sizes and diversities, the available public datasets at that time (e.g. UCF101, HMDB51) could not meet the requirements from most existing deep ConvNets. Thus, overfitting can easily happen.

2.1.2. Contributions

The authors provided two types of contribution to solve the problems:

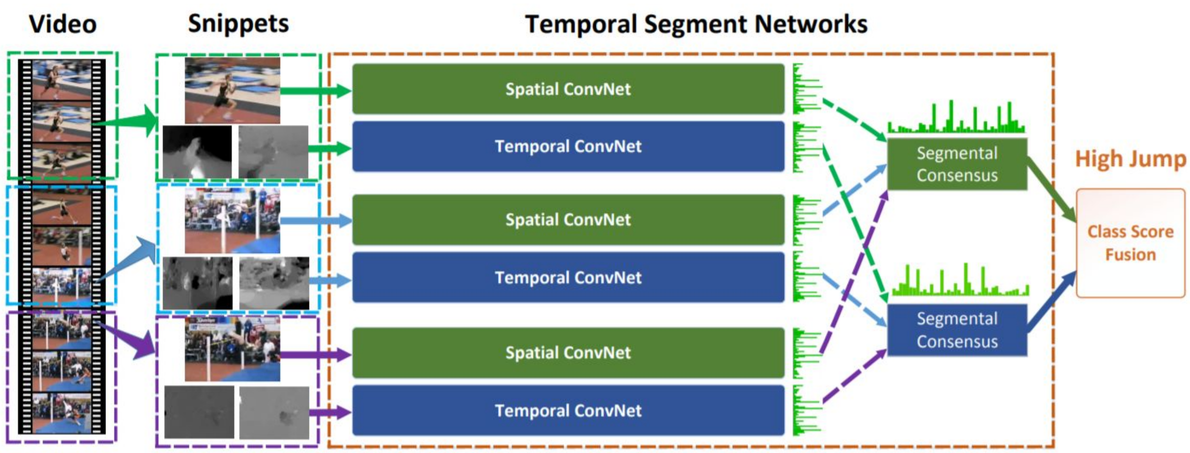
---

[1] "Temporal Segment Networks: Towards Good Practices for ...." 2 Aug. 2016, https://arxiv.org/abs/1608.00859. Accessed 20 Apr. 2020.
[2] "Quo Vadis, Action Recognition? A New Model and the ...." 22 May. 2017, https://arxiv.org/abs/1705.07750. Accessed 20 Apr. 2020.
[3] "UCF101 - CRCV | Center for Research in Computer Vision at ...." https://www.crcv.ucf.edu/data/UCF101.php. Accessed 20 Apr. 2020.
[4] "a large human motion database - Serre Lab » HMDB." https://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/. Accessed 20 Apr. 2020.
[5] "GluonCV." https://gluon-cv.mxnet.io/. Accessed 20 Apr. 2020.

Firstly, they created a new framework - TSN, “*which is based on the idea of long-range temporal structure modeling”.* This strategy divides every video into 3 segments, and clips are randomly chosen from the segments. Therefore TSN can train according to the information of the whole video instead of a part of it.

Secondly, they introduced several practical training strategies: cross-modality pre-training, regularization, enhanced data augmentation and four types of input modalities - a single RGB image, stacked RGB difference, stacked optical flow field, and stacked warped optical flow field.

2.1.3. TSN Structure[6]



2.2. I3D

2.2.1. Problems

The paper *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset* identified two problems around the year 2017:

Firstly, the limited sizes and diversities of the most used datasets at that time (e.g. UCF101, HMDB51).

Secondly, 2D CNN pre-trained on ImageNet had been proven to be beneficial to other vision detection projects, such as object detection. Then in the area of action recognition, CNN pre-trained on a larger dataset may also perform much better on smaller datasets. This idea needed to be tested.

[6] "Temporal Segment Networks: Towards Good Practices for ...." 2 Aug. 2016, https://arxiv.org/abs/1608.00859. Accessed 20 Apr. 2020.

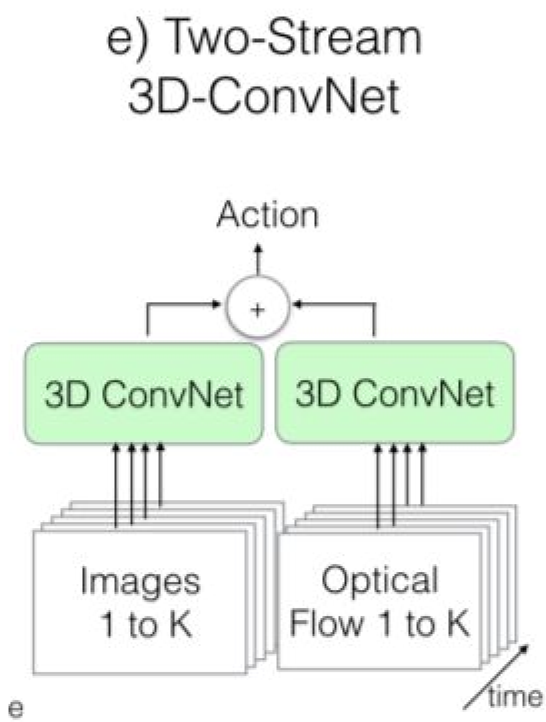
2.2.2. Contributions

They provided three major contributions:

Firstly, a much larger dataset - the Kinetics Human Action Video dataset with 400 classes and 400 videos per class. The source of these clips is YouTube.

Secondly, they pre-trained four existing CNN architectures on the new Kinetics dataset, and proved that the pre-training can bring performance improvements.
Thirdly, they introduced a new network architecture - I3D, which is based on 2D CNN InceptionV1. After pre-training on the Kinetics dataset, this new network performed extremely well on UCF101 and HMDB51.

2.2.3. I3D Structure[7]



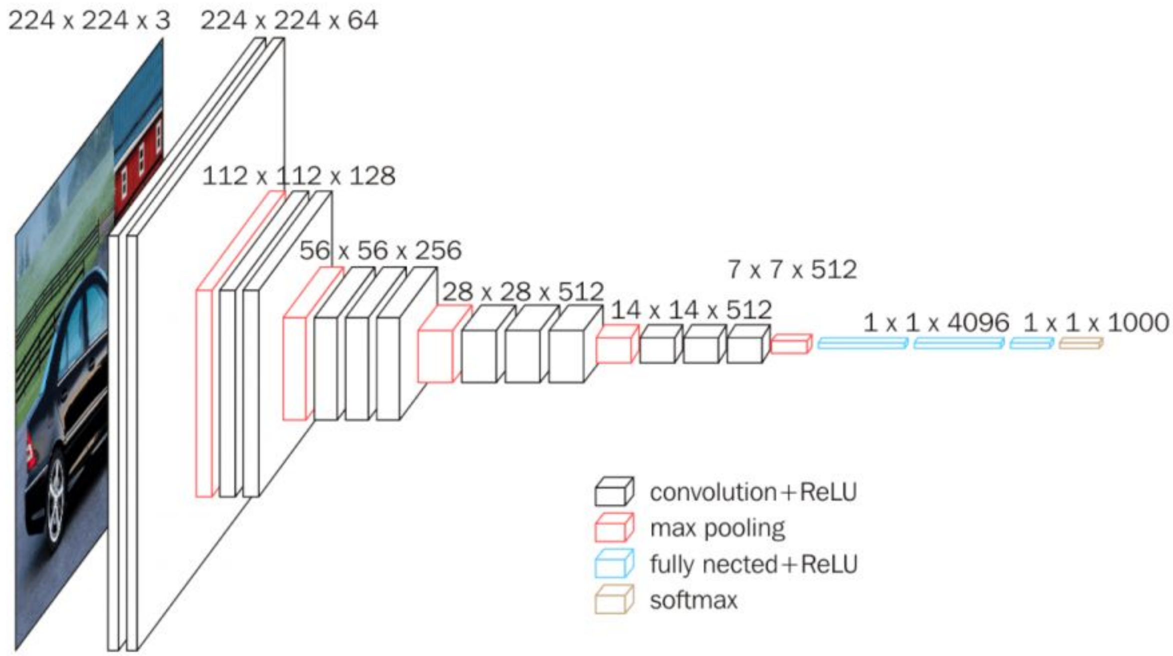
[7] "Quo Vadis, Action Recognition? A New Model and the ...." 22 May. 2017, https://arxiv.org/abs/1705.07750. Accessed 20 Apr. 2020.

2.3. Other concepts

2.3.1. VGG16[8]

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford.

The following is an image of a typical VGG16 architecture.[9]



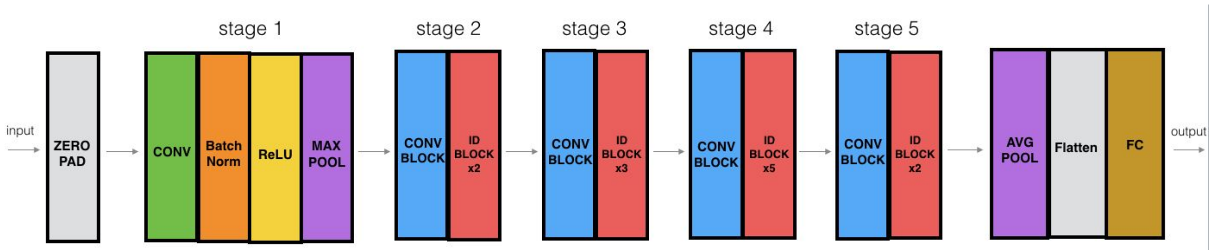
2.3.2. ResNet50[10]

The Residual Networks (ResNet) architectures were first introduced by He et al. in their 2015 paper, *Deep Residual Learning for Image Recognition*. ResNet50 is a residual network that is 50 layers deep.

The following is an image of a typical ResNet50 architecture.[11]

---

[8] "Very Deep Convolutional Networks for Large-Scale Image ...." 4 Sep. 2014, https://arxiv.org/abs/1409.1556. Accessed 20 Apr. 2020.
[9] "VGG16 - Convolutional Network for Classification and Detection." 20 Nov. 2018, https://neurohive.io/en/popular-networks/vgg16/. Accessed 20 Apr. 2020.
[10] "Deep Residual Learning for Image Recognition." https://arxiv.org/abs/1512.03385. Accessed 20 Apr. 2020.
[11] "Understanding and Coding a ResNet in Keras - Towards Data ...." 4 Jan. 2019, https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33. Accessed 20 Apr. 2020.



3. Training environment

3.1. Brief Information on the Training Environment

A virtual machine running on GCP, with 8 VCPUs, 30 GB RAM, 1 NVIDIA Tesla T4 GPU, 300 GB SSD. CUDA 10.0, PyTorch 1.4 and GluonCV 0.70 are installed.

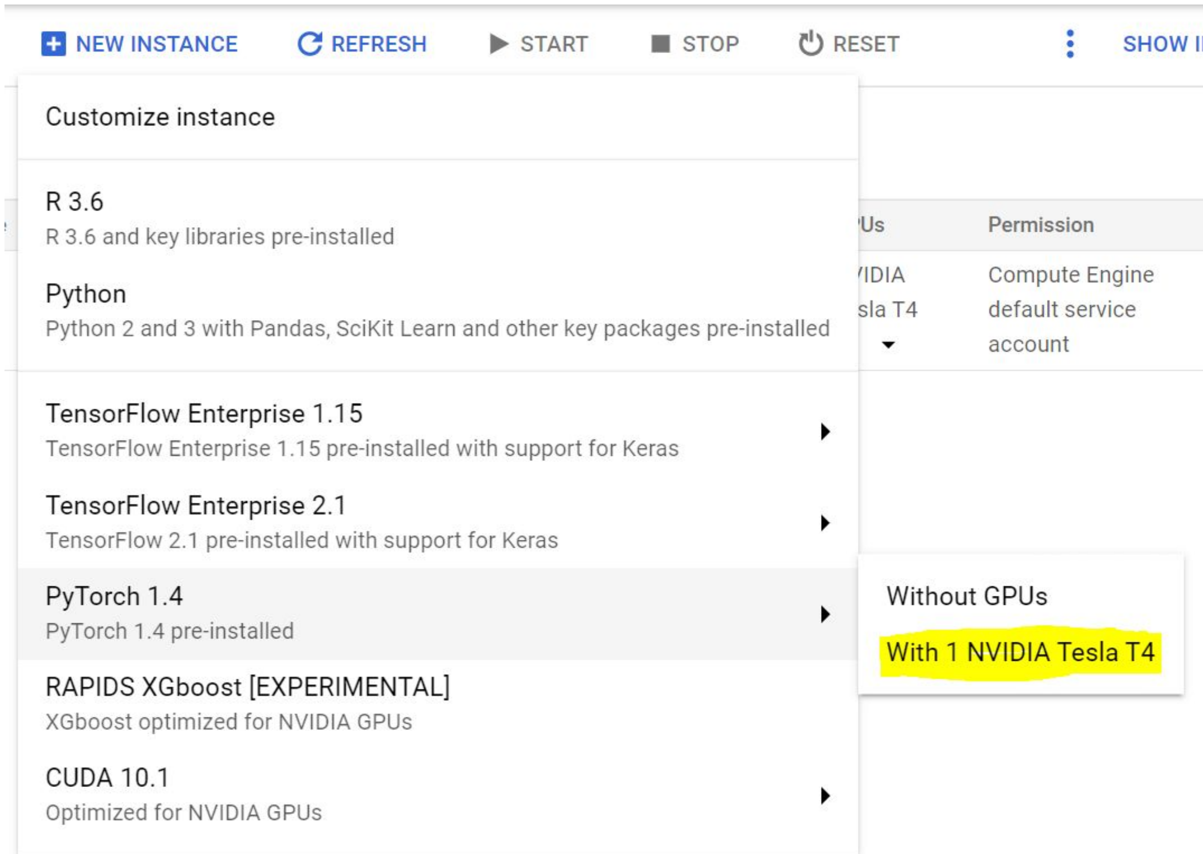
3.2. How to Establish an Environment on GCP

3.2.1. Setup the Hardware Virtual Machine

Create a Google account to use https://cloud.google.com/ A $300 credit will be granted to new users.

Login to GCP console from https://cloud.google.com/ and search for “Notebooks”

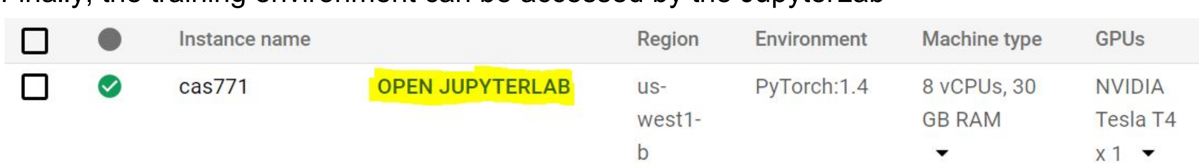
The easiest way is create one compute instance as the following picture,



You can also create customized instances according to your needs, with preferred hardware specifications. However, some suggestions might need to be followed:

- In order to use the GluonCV toolkit, use of GPU is recommended
- In order to download and process the datasets, the harddrive is recommended to be at least 200 GB.
- CUDA and PyTorch are required.

Finally, the training environment can be accessed by the JupyterLab



3.2.2. Setup the Software Tools

In the opened JupyterLab, choose Terminal from the launcher to set up the software.

- If the instance do not have CUDA and PyTorch yet, install them by the following command

In order for GluonCV to run correctly, the versions of CUDA and PyTorch must be compatible, so you might want to run this command anyway.

```
# You can try if version 10.1 works, but make sure the next step uses the
same version number.
conda install pytorch torchvision cudatoolkit=10.0 -c pytorch
```

- Install GluonCV
  The suitable command for specific system can be found at
  https://gluon-cv.mxnet.io/install.html
  A typical installation with MKL (Intel CPU acceleration) and CUDA (Nvidia GPU acceleration) is as follows

```
# Here we assume CUDA 10.0 is installed. You can change the number
# according to your own CUDA version.
pip install --upgrade mxnet-cu100mkl gluoncv
```

- Install unrar-nonfree (unrar-free will cause errors in the next steps)

```
wget -qO - https://ftp-master.debian.org/keys/archive-key-10.asc | sudo
apt-key add -
echo deb http://deb.debian.org/debian buster main contrib non-free | sudo
tee -a /etc/apt/sources.list
sudo apt-get update
sudo apt-get install unrar
```

- Install rarfile, Cython, mmcv

```
pip install rarfile Cython mmcv
```

3.3. Fetch Training Scripts

Fetch all the needed Scripts from my GitHub repo:

```
git clone https://github.com/Ao99/action-recognition-learning.git
```

Most parts of the scripts are borrowed from https://gluon-cv.mxnet.io/ then modified.

4. Training Datasets with TSN

Due to the limitation of available time and the cost of using Google virtual machine, each training is usually controlled to be within 3 hours when adjusting the parameters.

4.1. How to Train UCF101 with VGG16, TSN, and GluonCV

4.1.1. Prepare Dataset[12]

Run this script to download UCF101 dataset and parse it into image frames with GluonCV

```
python ucf101.py
```

This process will take about one hour.

4.1.2. Tune the Parameters

Open **01_tsn_ucf101.ipynb** to prepare this training.

By default, the number of GPUs is set to 1, but it can be adjusted according to your hardware selection.

```
# number of GPUs to use
num_gpus = 1
ctx = [mx.gpu(i) for i in range(num_gpus)]
```

The CNN used here is the VGG16 architecture with TSN method.

```
# Get the model vgg16_ucf101 with temporal segment network, with
net = get_model(name='vgg16_ucf101', nclass=101, num_segments=3)
net.collect_params().reset_ctx(ctx)
print(net)
```

Running this part will load the CNN structure with no pretrained weights, and print the structure out. Part of the printed result is in the following picture.

[12] "Prepare the UCF101 dataset — gluoncv 0.7.0 documentation." https://gluon-cv.mxnet.io/build/examples_datasets/ucf101.html. Accessed 20 Apr. 2020.

```
ActionRecVGG16(
  (features): HybridSequential(
    (0): Conv2D(3 -> 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): Activation(relu)
    (2): Conv2D(64 -> 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): Activation(relu)
    (4): MaxPool2D(size=(2, 2), stride=(2, 2), padding=(0, 0), ceil_mode=False, global_pool=Fa
lse, pool_type=max, layout=NCHW)
    (5): Conv2D(64 -> 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): Activation(relu)
    (7): Conv2D(128 -> 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): Activation(relu)
    (9): MaxPool2D(size=(2, 2), stride=(2, 2), padding=(0, 0), ceil_mode=False, global_pool=Fa
lse, pool_type=max, layout=NCHW)
    (10): Conv2D(128 -> 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): Activation(relu)
    (12): Conv2D(256 -> 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

By default, the batch size per GPU is set to 5. This can be adjusted higher according to your hardware selection. However, a higher number may cause the shortage of GPU memory.

```
# Batch Size for Each GPU
per_device_batch_size = 5
```

The number of data loaders is set to 8. It seems that this number barely affects the training efficiency. Meanwhile, a higher number may cause the loss of response during the training. Thus, you may want this number to be lower. I even set it to 1 during some training sessions.

```
# Number of data loader workers
num_workers = 8
# Calculate effective total batch size
batch_size = per_device_batch_size * num_gpus
```

The default learning rate is set to 0.001, weight decay 0.0001, and momentum 0.9. The learning rate decay factor is set to 0.1, and epochs where learning rate decays is set in the array.
In this example, the initial learning rate is 0.001, starting from epoch 40, the learning rate will be 0.001*0.1=0.0001, starting from epoch 70, it will be 0.00001, and so on.
The whole strategy can be customized according to your needs.

```
# Learning rate decay factor
lr_decay = 0.1
# Epochs where learning rate decays
lr_decay_epoch = [40, 70, 90, np.inf]

# Stochastic gradient descent
optimizer = 'sgd'
# Set parameters
optimizer_params = {'learning_rate': 0.001, 'wd': 0.0001, 'momentum': 0.9}
```

In the last block, the number of epochs can be set. It is recommended to start with trying epochs between 50 and 100.

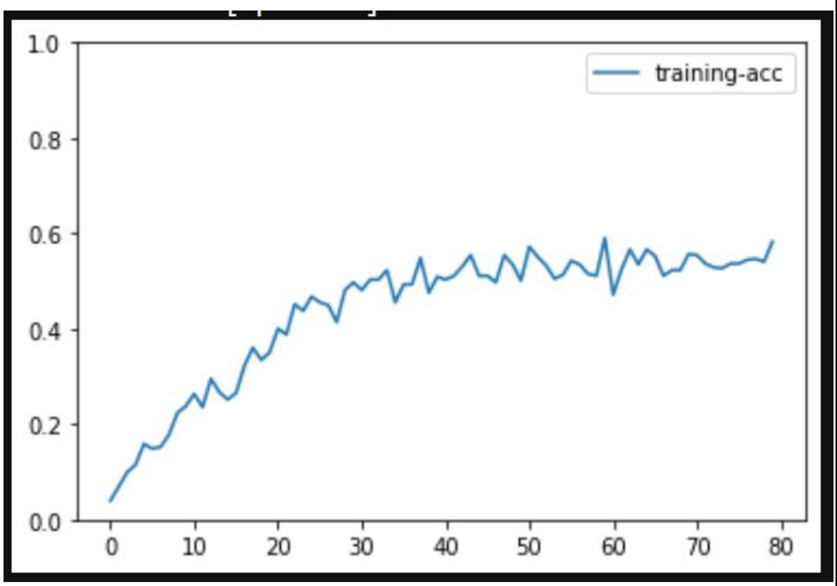
```
epochs = 100
lr_decay_count = 0
```

The number of iterations per epoch can also be defined in this block.

```
    if i == 200:
        print("|")
        break
```

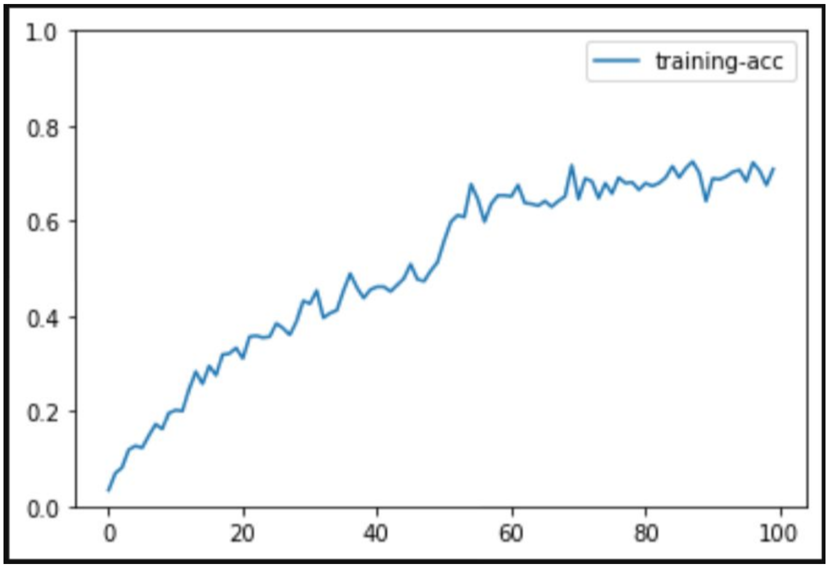
4.1.3. Training[13]

More than a dozen training sessions have been tested with VGG16 and TSN on the UCF101 dataset. The three ones in the following examples are some of the most typical results with various parameter settings.

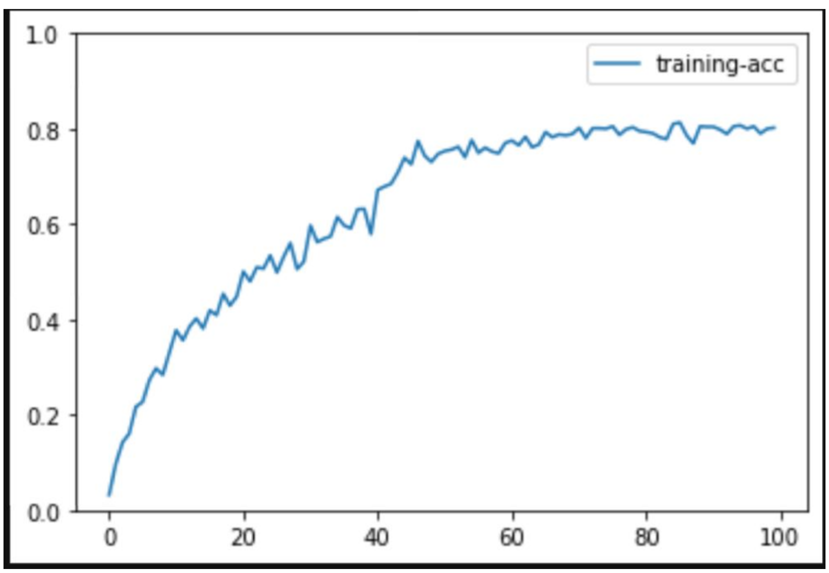
	**Train accuracy = about 0.58** Epochs = 80 Batch size = 5 Iterations per epoch = 100 Learning rate = 0.001 (epoch 1-20), 0.0001 (epoch 21-40), 0.00001 (epoch 41-60), 0.000001 (epoch 61-80) Weight decay = 0.0001 Momentum = 0.9

Training 1

[13] "Tutorials — gluoncv 0.7.0 documentation." https://gluon-cv.mxnet.io/tutorials/index.html. Accessed 20 Apr. 2020.

	**Train accuracy = about 0.70** Epochs = 100 Batch size = 5 Iterations per epoch = 100 Learning rate = 0.001 (epoch 1-50), 0.0001 (epoch 51-90), 0.00001 (epoch 91-100) Weight decay = 0.0001 Momentum = 0.9

Training 2

	**Train accuracy = about 0.80** Epochs = 100 Batch size = 5 Iterations per epoch = 200 Learning rate = 0.001 (epoch 1-40), 0.0001 (epoch 41-70), 0.00001 (epoch 71-90), 0.000001 (epoch 91-100) Weight decay = 0.0001 Momentum = 0.9

Training 3

4.2. How to Train HMDB51 with ResNet50, TSN, and GluonCV

4.2.1. Prepare Dataset[14]

Run this script to download HMDB51 dataset and parse it into image frames with GluonCV

[14] "Prepare the HMDB51 Dataset — gluoncv 0.7.0 documentation." https://gluon-cv.mxnet.io/build/examples_datasets/hmdb51.html. Accessed 20 Apr. 2020.

```
python hmdb51.py
```

This process will take about one hour.

4.2.2. Training[15]

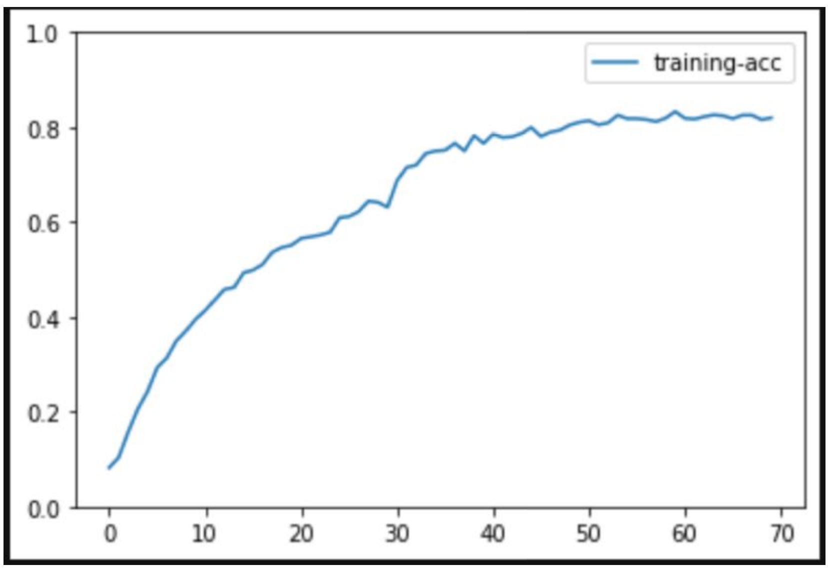
Open **02_tsn_hmdb51.ipynb** for this training.

Please refer to **4.1.2 Tune the Parameters** for the method to tune the parameters.

The CNN used here is the ResNet50 architecture with TSN method.

```
# Get the model vgg16_hmdb51 with temporal segment network, with 51 outpu
net = get_model(name='resnet50_v1b_hmdb51', nclass=51, num_segments=3)
net.collect_params().reset_ctx(ctx)
print(net)
```

Several training sessions have been tested with ResNet50 and TSN on the HMDB51 dataset. The one in the following example is one of the most typical results.



Train accuracy = about 0.82

Epochs = 70
Batch size = 15
Iterations per epoch = 200
Learning rate = 0.01 (epoch 1-30), 0.001 (epoch 31-50), 0.0001 (epoch 51-70)
Weight decay = 0.0001
Momentum = 0.9

[15] "Tutorials — gluoncv 0.7.0 documentation." https://gluon-cv.mxnet.io/tutorials/index.html. Accessed 20 Apr. 2020.

5. Training Datasets with I3D

Due to the limitation of available time and the cost of using Google virtual machine, each training is usually controlled to be within 4 hours when adjusting the parameters.

5.1. How to Train UCF101 with ResNet50, I3D, and GluonCV

5.1.1. Prepare Dataset[16]

If the dataset has not been prepared yet, run this script to download UCF101 dataset and parse it into image frames with GluonCV

```
python ucf101.py
```

5.1.2. Training[17]

Open **03_i3d_ucf101.ipynb** for this training.

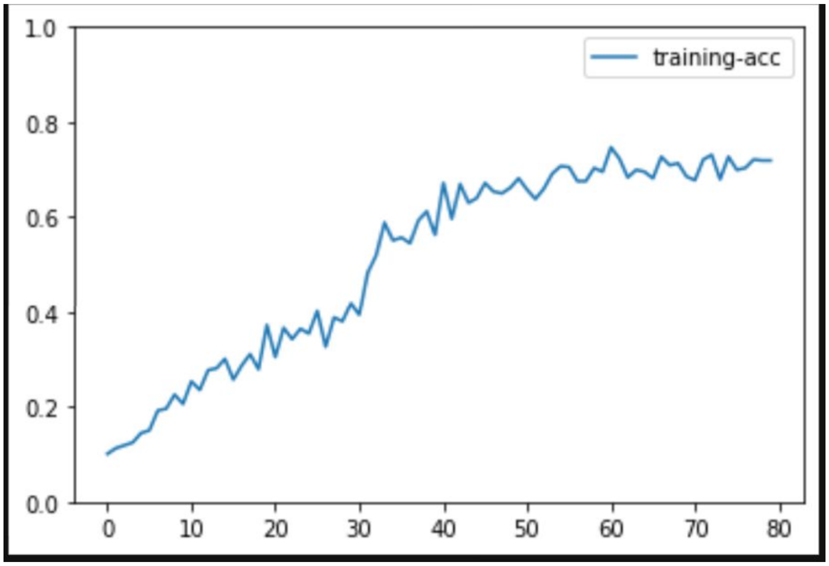
Please refer to **4.1.2 Tune the Parameters** for the method to tune the parameters.

The CNN used here is the ResNet50 architecture with I3D method.

```
# Get the model i3d_resnet50_v1_ucf101 with 101 output classes, without p
net = get_model(name='i3d_resnet50_v1_ucf101', nclass=101)
net.collect_params().reset_ctx(ctx)
print(net)
```

Several training sessions have been tested with ResNet50 and I3D on the UCF101 dataset. The ones in the following examples are some of the most typical results.

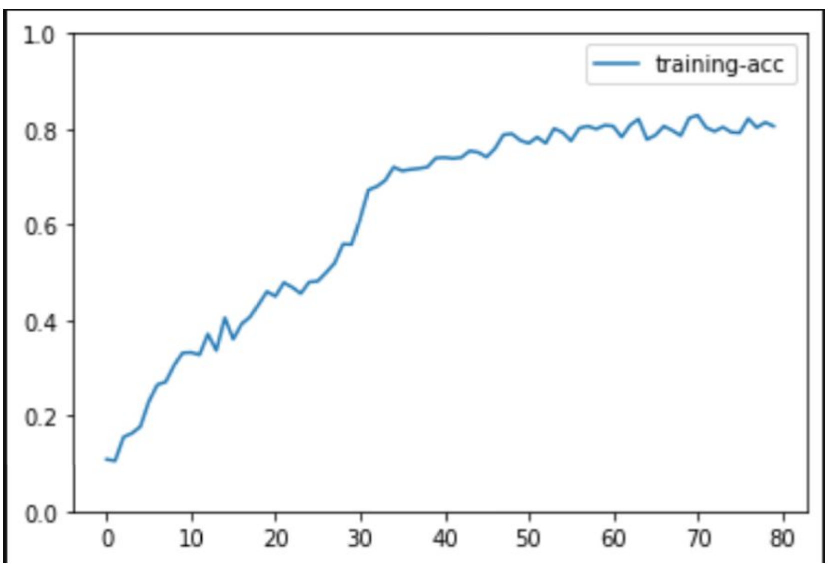
[16] "Prepare the UCF101 dataset — gluoncv 0.7.0 documentation." https://gluon-cv.mxnet.io/build/examples_datasets/ucf101.html. Accessed 20 Apr. 2020.
[17] "Tutorials — gluoncv 0.7.0 documentation." https://gluon-cv.mxnet.io/tutorials/index.html. Accessed 20 Apr. 2020.



Train accuracy = about 0.72

Epochs = 80
Batch size = 5
Iterations per epoch = 100
Learning rate =
0.01 (epoch 1-25),
0.001 (epoch 26-55),
0.0001 (epoch 56-70),
0.00001 (epoch 71-80)
Weight decay = 0.0001
Momentum = 0.9

Training 1



Train accuracy = about 0.81

Epochs = 80
Batch size = 5
Iterations per epoch = 200
Learning rate =
0.01 (epoch 1-30),
0.001 (epoch 31-55),
0.0001 (epoch 56-70),
0.00001 (epoch 71-80)
Weight decay = 0.0001
Momentum = 0.9

Training 2

5.2. How to Train HMDB51 with ResNet50, I3D, and GluonCV

5.2.1. Prepare Dataset[18]

If the dataset has not been prepared yet, run this script to download HMDB51 dataset and parse it into image frames with GluonCV

[18] "Prepare the HMDB51 Dataset — gluoncv 0.7.0 documentation." https://gluon-cv.mxnet.io/build/examples_datasets/hmdb51.html. Accessed 20 Apr. 2020.

```
python hmdb51.py
```

5.2.2. Training[19]

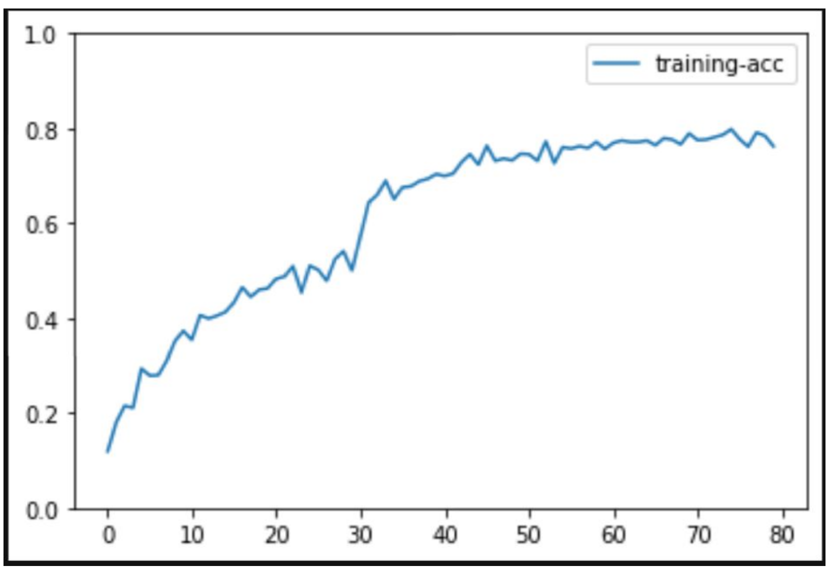
Open **04_i3d_hmdb51.ipynb** for this training.

Please refer to **4.1.2 Tune the Parameters** for the method to tune the parameters.

The CNN used here is the ResNet50 architecture with I3D method.

```
# Get the model i3d_resnet50_v1_hmdb51 with 51 output classes, without pre
net = get_model(name='i3d_resnet50_v1_hmdb51', nclass=51)
net.collect_params().reset_ctx(ctx)
print(net)
```

Several training sessions have been tested with ResNet50 and I3D on the HMDB51 dataset. The one in the following example is one of the most typical results.



Train accuracy = about 0.79

Epochs = 80
Batch size = 5
Iterations per epoch = 200
Learning rate =
0.01 (epoch 1-30),
0.001 (epoch 31-55),
0.0001 (epoch 56-70),
0.00001 (epoch 71-80)
Weight decay = 0.0001
Momentum = 0.9

[19] "Tutorials — gluoncv 0.7.0 documentation." https://gluon-cv.mxnet.io/tutorials/index.html. Accessed 20 Apr. 2020.