1. The reason for selecting Continual Learning

Deep Learning, a prevailing class of machine learning algorithms, requires vast labelled samples to train models so that a specific isolated task can be tackled.
This kind of training is classified as offline training and supervised learning, and machine intelligence endowed with general knowledge and exceptional perception scarcely develops through this sort of learning.

Also, it is arduous to acquire enormous labelled training samples from intricate problems in the real world, which makes scholars shift their fields of research to unsupervised learning gradually. Convoluted problems are addressed by applying the features of unsupervised learning.

Further, future artificial intelligence systems usually demand outstanding potential for perception and require adaptation to confront external problems through learning from its surroundings incessantly. Hence, some scholars think that future artificial intelligence systems should behave like humans not only able to constantly learn from and adjust to the exterior world but also autonomously develop more complex skills and understanding. Thus, Continual Learning is an ideal preference for unsupervised learning.

Continual Learning is not only able to deal with high-dimensional and real-time data but also continually update models which adapt to and resolve problems of new situations when surroundings change over time, aka new usable data inputted.

Currently, obtainable frameworks and algorithms for Continual Learning exist. I select Caffe as my Deep Learning framework and opt for ResNet featuring continual learning as my algorithm.

ResNet, categorized into an unsupervised algorithm, can regularly learn from and adapt to an environment. Also, ResNet autonomously evolves more complicated skills and knowledge. A thesis, “Deep Residual Learning for Image Recognition”, states that ResNet can gain fine experiences from low levels and transfer these experiences to high levels. Learning is endowed with memorization to avoid catastrophic forgetting, vanishing gradients, exploding gradients, and degradation problem.

ResNet exploits an extra shortcut path to interface with the next layer or next layers. Also, the sizes of the layers have to be the same size so that the outcomes trained by low levels can be transferred into high levels. Thus, ResNet can combine artlessly low/mid/high-level attributes and classifiers, which described as an end-to-end multi-layer fashion in the thesis, “Deep Residual Learning for Image Recognition”.

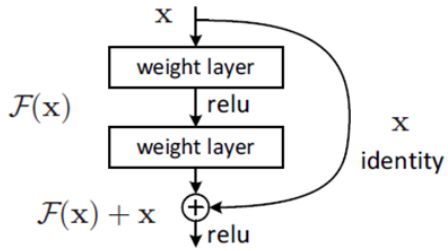
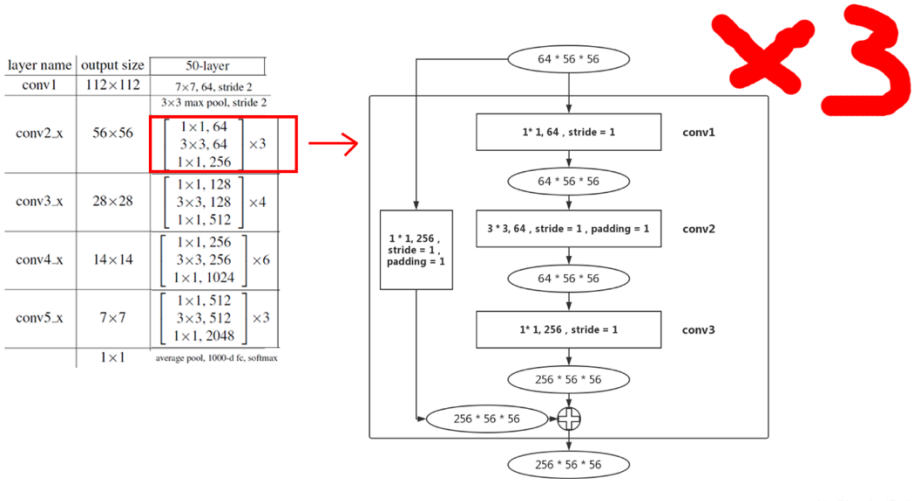


Figure 2. Residual learning: a building block.

Take ResNet-50 as an example to elucidate the mechanism of a ResNet-50 architecture.

(1) The layers before the main blocks of ResNet-50 (the lowest layers) include conv1 (7×7 convolution layer) and conv2_x (3×3 max pooling layer).

(2) Layer 1 is Conv2_x (3×3×64 as a Basicblock). One convolution layer is attached to an input channel and an output channel. The kernel size of an output channel is enlarged 4 times that of an input channel. A bottleneck consists of an input channel, an output channel and a Basicblock.



(3) Excepting the number of channels expanded and the kernel size of output channels is unalike, the constitution of Layer2, Layer3 and Layer4 are the same as that of Layer1.

2. The procedures for executing my program with BVLC/ResNet architectures

Step 1: Set up a programming environment:

I followed the instructions from the file "README.md" on the website (vlomonaco.github.io/core50/index.html) designated by the professor to establish the programming environment. Then, I downloaded code from the URL (github.com/vlomonaco/core50) and downloaded data by executing the bash file "fetch_data_and_setup.sh".

Step 2: Configure and execute the program:

I comprehended the entire architecture of the program from the folders "core" and "confs" in the project. Next, I modified the files in the folder "core" to eliminate unnecessary code such as code for different classification strategies and Lightning Memory-Mapped Database. After that, I executed the BVLC model with the NC configurations to grasp the mechanism of the program. To overcome the abysmal load speed of images, I opted for a RAM disk to hasten to load images.

Step 3: Choose ResNet-10/ResNet-50 to reproduce the experiments:

I downloaded ResNet-10/ResNet-50 models and respective configuration files from @cvjena's Github (github.com/cvjena/cnn-models/tree/master/ResNet_preact), altered the configurations and reproduced the experiments.

Step 4: Benchmarks:

Benchmarks (9-batches/1-run) for classification at object level (50 classes) applying BVLC, ResNet-10 and ResNet-50 are as follows:

BVLC:

```
Batch: 8, name: train_batch_08 (size 119894), Accuracy: 0.6568444444444445
[Train-net] avg. new weights: 2.0324189e-05
[Train-net] avg. new biases: -2.9343647e-07
[Train-net] avg. other weights: nan
[Train-net] avg. other biases: nan
[Train-net] tot weights avg. : 2.0324189e-05
[Test-net] avg. new weights: 2.0324189e-05
[Test-net] avg. new biases: -2.9343647e-07
[Test-net] avg. other weights: nan
[Test-net] avg. other biases: nan
Acc. per class:
0: 0.747        1: 0.251        2: 0.64         3: 0.284        4: 0.492
5: 0.697        6: 0.625        7: 0.43         8: 0.263        9: 0.548
10: 0.589       11: 0.579       12: 0.387       13: 0.687       14: 0.853
15: 0.641       16: 0.201       17: 0.594       18: 0.805       19: 0.329
20: 0.743       21: 0.796       22: 0.835       23: 0.932       24: 0.586
25: 0.576       26: 0.385       27: 0.732       28: 0.586       29: 0.31
30: 0.982       31: 0.909       32: 0.914       33: 0.98        34: 0.933
35: 0.835       36: 0.925       37: 0.793       38: 0.571       39: 0.751
40: 0.868       41: 0.468       42: 0.72        43: 0.726       44: 0.863
45: 0.729       46: 0.731       47: 0.697       48: 0.602       49: 0.72
```

ResNet-10:

```
Batch: 8, name: train_batch_08 (size 119894), Accuracy: 0.6843555555555556
[Train-net] avg. new weights: 0.009121395
[Train-net] avg. new biases: 0.021162495
[Train-net] avg. other weights: nan
[Train-net] avg. other biases: nan
[Train-net] tot weights avg. : 0.009121395
[Test-net] avg. new weights: 0.009121395
[Test-net] avg. new biases: 0.021162495
[Test-net] avg. other weights: nan
[Test-net] avg. other biases: nan
Acc. per class:
0: 0.809        1: 0.469        2: 0.641        3: 0.347        4: 0.634
5: 0.648        6: 0.629        7: 0.274        8: 0.33         9: 0.472
10: 0.711       11: 0.51        12: 0.578       13: 0.64        14: 0.952
15: 0.58        16: 0.404       17: 0.699       18: 0.887       19: 0.489
20: 0.759       21: 0.802       22: 0.76        23: 0.851       24: 0.587
25: 0.879       26: 0.667       27: 0.77        28: 0.623       29: 0.383
30: 0.964       31: 0.97        32: 0.928       33: 0.964       34: 0.944
35: 0.857       36: 0.832       37: 0.71        38: 0.628       39: 0.795
40: 0.89        41: 0.508       42: 0.77        43: 0.651       44: 0.734
45: 0.635       46: 0.774       47: 0.753       48: 0.497       49: 0.627
```

ResNet-50:

```
Batch: 8, name: train_batch_08 (size 119894), Accuracy: 0.8215333333333333
[Train-net] avg. new weights: 0.0030594722
[Train-net] avg. new biases: 0.011786158
[Train-net] avg. other weights: nan
[Train-net] avg. other biases: nan
[Train-net] tot weights avg. : 0.0030594722
[Test-net] avg. new weights: 0.0030594722
[Test-net] avg. new biases: 0.011786158
[Test-net] avg. other weights: nan
[Test-net] avg. other biases: nan
Acc. per class:
0: 0.927        1: 0.687        2: 0.723        3: 0.489        4: 0.827
5: 0.886        6: 0.572        7: 0.688        8: 0.621        9: 0.713
10: 0.942       11: 0.733       12: 0.459       13: 0.849       14: 0.938
15: 0.782       16: 0.669       17: 0.855       18: 0.956       19: 0.893
20: 0.861       21: 0.879       22: 0.897       23: 0.963       24: 0.829
25: 0.782       26: 0.813       27: 0.887       28: 0.717       29: 0.504
30: 0.984       31: 0.993       32: 0.961       33: 0.997       34: 0.978
35: 0.969       36: 0.909       37: 0.859       38: 0.818       39: 0.844
40: 0.906       41: 0.812       42: 0.809       43: 0.796       44: 0.92
45: 0.845       46: 0.879       47: 0.892       48: 0.77        49: 0.798
```

From the benchmarks (9-batches/1-run) for classification at object level (50 classes), the performance of ResNet-50 prevails over ResNet-10 and BVLC.