# Lab 5: Retrain VGG19 on Cifar-10

## Lab Objective:

In this lab, you will use the pre-trained CNN model (VGG-19 [1]) to build an object recognition system. Second, you will be asked to retrain the pre-trained CNN model (VGG-19) on Cifar-10 dataset.

## Requirements:

- Load pre-trained VGG-19 and build an object recognition system
  - NOTE: The input image should be arbitrary size
- Retrain VGG-19 on Cifar-10

## Turn in:

Report: 4/10 (一)　23:59

Demo: 4/11 (二)　下課後

## Environment:

- VGG models are trained with color image size $224 \times 224 \times 3$
  - The input image should be arbitrary size in your system. It means that you need to preprocess the input image, for example: resize, crop …etc.

- Download pre-trained VGG-19:
  - Tensorflow: https://github.com/machrisaa/tensorflow-vgg
    (內有建立 object recognition system 的 sample code)

- Cifar-10 dataset is as same as previous lab

## Lab Description:

- VGG models [1]

  The main contribution of VGG model is a thorough evaluation of networks of **increasing depth** using an architecture with very **small $(3 \times 3)$ convolution** filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

| ConvNet Configuration | | | | | | |
|---|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E | |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers | |
| input (224 × 224 RGB image) | | | | | | NAME: |
| conv3-64 | conv3-64 | conv3-64 | conv3-64 | conv3-64 | conv3-64 | conv1_1 |
| | **LRN** | **conv3-64** | conv3-64 | conv3-64 | conv3-64 | conv1_2 |
| maxpool | | | | | | pool1 |
| conv3-128 | conv3-128 | conv3-128 | conv3-128 | conv3-128 | conv3-128 | conv2_1 |
| | | **conv3-128** | conv3-128 | conv3-128 | conv3-128 | conv2_2 |
| maxpool | | | | | | pool2 |
| conv3-256 | conv3-256 | conv3-256 | conv3-256 | conv3-256 | conv3-256 | conv3_1 |
| conv3-256 | conv3-256 | conv3-256 | conv3-256 | conv3-256 | conv3-256 | conv3_2 |
| | | | **conv1-256** | **conv3-256** | conv3-256 | conv3_3 |
| | | | | | **conv3-256** | conv3_4 |
| maxpool | | | | | | pool3 |
| conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv4_1 |
| conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv4_2 |
| | | | **conv1-512** | **conv3-512** | conv3-512 | conv4_3 |
| | | | | | **conv3-512** | conv4_4 |
| maxpool | | | | | | pool4 |
| conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv5_1 |
| conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv3-512 | conv5_2 |
| | | | **conv1-512** | **conv3-512** | conv3-512 | conv5_3 |
| | | | | | **conv3-512** | conv5_4 |
| maxpool | | | | | | pool5 |
| FC-4096 | | | | | | fc6 |
| FC-4096 | | | | | | fc7 |
| FC-1000 | | | | | | fc8 |
| soft-max | | | | | | |

- Object recognition system
  - VGG-19 model contains 1000 classes
  - You should report the Top-5 classes

- Load pre-trained VGG from files

Download: vgg19.npy
https://mega.nz/#!xZ8glS6J!MAnE91ND_WyfZ_8mvkuSa2YcA7q-1ehfSm-Q1fxOvvs


```
###
### load VGG  ###
params_dict = np.load('vgg19.npy', encoding='latin1').item()
```

Apply pre-trained conv1_1

```
output = tf.nn.relu(tf.nn.bias_add(conv2d(x,tf.Variable(params_dict["conv1_1"][0])),tf.Variable(params_dict["conv1_1"][1])))
```

tf.Variable → content are trainable
tf.constant → content are fixed

- Retrain VGG-19 on Cifar-10
  - Original input size is 224×224, Cifar-10 is 32×32
    - Change some layers
  - The output classes of VGG-19 are 1000, but Cifar-10 are 10 classes.
    - Change the final fully connected layer
  - Orange color denoted that you can't load from the pre-trained model.
    - Weight initial = rand_normal(stddev=0.01)

Retrain model

| Layer name | Output size | Layer design (size, # filter) | activation |
|---|---|---|---|
| conv1_1 | 32×32 | 3x3 conv, 64 | ReLU |
| conv2_1 | 32×32 | 3x3 conv, 64 | ReLU |
| pool1 | 16×16 | Max 2x2 , stride 2 ("same") | |
| conv2_1 | 16×16 | 3x3 conv, 128 | ReLU |
| conv2_2 | 16×16 | 3x3 conv, 128 | ReLU |
| pool2 | 8×8 | Max 2x2 , stride 2 ("same") | |
| conv3_1 | 8×8 | 3x3 conv, 256 | ReLU |
| conv3_2 | 8×8 | 3x3 conv, 256 | ReLU |
| conv3_3 | 8×8 | 3x3 conv, 256 | ReLU |
| conv3_4 | 8×8 | 3x3 conv, 256 | ReLU |
| pool3 | 4×4 | Max 2x2 , stride 2 ("same") | |

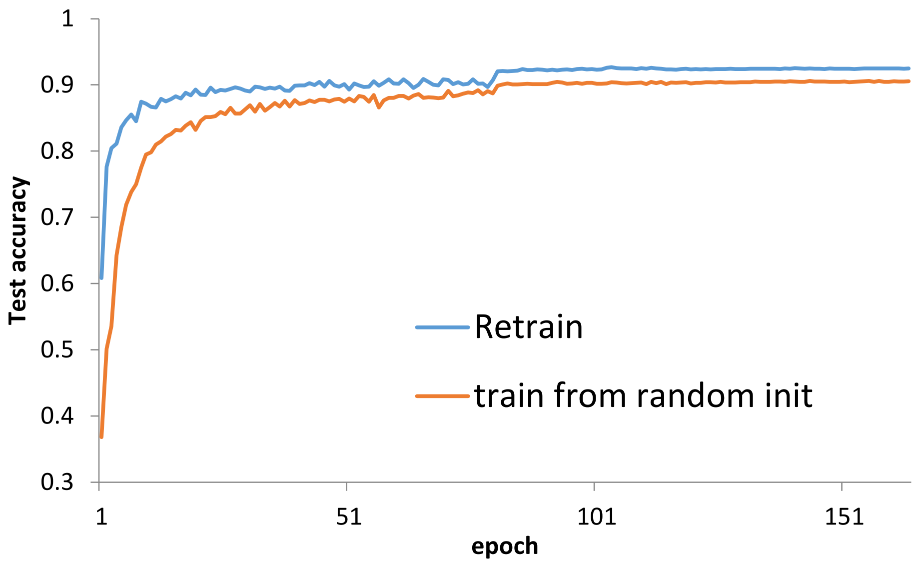
| | | | |
|---|---|---|---|
| conv4_1 | 4×4 | 3×3 conv, 512 | ReLU |
| conv4_2 | 4×4 | 3×3 conv, 512 | ReLU |
| conv4_3 | 4×4 | 3×3 conv, 512 | ReLU |
| conv4_4 | 4×4 | 3×3 conv, 512 | ReLU |
| pool4 | 2×2 | Max 2×2 , stride 2 ("same") | |
| conv5_1 | 2×2 | 3×3 conv, 512 | ReLU |
| conv5_2 | 2×2 | 3×3 conv, 512 | ReLU |
| conv5_3 | 2×2 | 3×3 conv, 512 | ReLU |
| conv5_4 | 2×2 | 3×3 conv, 512 | ReLU |
| ~~pool5~~ | | | |
| reshape | 1×1 | 2*2*512 | |
| fc6 | 1×1 | 2048×4096 | ReLU |
| | | Dropout 0.5 | |
| fc7 | 1×1 | 4096×4096 | ReLU |
| | | Dropout 0.5 | |
| fc8 | 1×1 | 4096×10 | softmax |

## Implementation Details:

- VGG model have preprocessing in training/testing. The input image subtracted mean value in each color channel. The mean of each color channel is in the below.
  - B = 103.939
  - G = 116.779
  - R = 123.68
- Images are resized such that the smallest dimension becomes 224, then the center $224 \times 224$ crop is used.
- Training Hyperparameters:
  - Method: SGD with momentum
  - Mini-batch size: 128 (391 iterations for each epoch)
  - Total epochs: 164, momentum 0.9
  - Initial learning rate: 0.01, divide by 10 at 81, 122 epoch
  - Loss function: cross-entropy
  - Use data augmentation / subtract RGB mean value
- 注意在 training random initialization 時
  - conv layer init = random_normal(stddev = 0.03)
  - fc layer init = random_normal(stddev = 0.01)
  -

## Methodology:

- Retrained model achieved 92.47% accuracy in my implementation




## Extra Bonus:

- Add BN on VGG-19 and retrain on Cifar-10 or ImageNet (if possible)

## References:


[1] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.


## Report Spec: [black: Demo, Gray: No Demo]

1. Introduction (15%)
2. Experiment setup (15%)
   - The detail of your model
   - Report all your training hyper-parameters
3. Result
   - The comparison between retrained model and random initialization
     - Final Test error (10%, 15%)
     - Training loss curve (you need to record training loss every epoch) (10%, 15%)
     - Test error curve (you need to record test error every epoch) (10%, 15%)
4. Discussion (20%, 25%)

Demo (20%) [抽 20 人 DEMO] (需 DEMO object recognition system )

-------實驗結果標準  (retrained model)----

Accuracy: (94.0~90.0)%    = 100%

Accuracy: (90.0~87.0)%    = 90%

Accuracy: (87.0~84.0)%    = 80%

Accuracy: below 84.0%    = 70%

Accuracy: 10%    = 0%

評分標準: 40%*實驗結果  + 60%*(報告+DEMO)