

# Deep Learning and Practice

## Lab 5: Retrain VGG19 on Cifar-10

李韓

0556157

fm.bigballon@gmail.com

## Introduction

---

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

I used Keras to implement these models and test the following **requirements**:

- **Object recognition system**
- **Random initialization models**
  - pure random initialization
  - random initialization + WI
- **Retrained models**
  - pure retrain
  - retrain + WI
  - retrain + WI + WD (0.0001,0.0005,0.001,0.0013,0.0015)
  - **retrain + WI + BN + WD (0.0001,0.0005,0.001,0.0013,0.0015)**

### Lab Description:

#### ■ VGG models

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

#### ■ Object recognition system

- VGG-19 model contains 1000 classes
- Report the Top-5 classes

#### ■ Retrain VGG-19 on Cifar-10

- Original input size is  $224 \times 224$ , Cifar-10 is  $32 \times 32$ 
  - ◆ Change some layers
- The output classes of VGG-19 are 1000, but Cifar-10 are 10 classes.
  - ◆ Change the final fully connected layer

# Experiment setup

## ■ Architecture Details:

Table 1: Retrain model (based on keras' vgg19 model)

Layer name	Output size	Layer design (size, # filter)	activation
block1_conv1	32×32	3×3 conv, 64	ReLU
block1_conv2	32×32	3×3 conv, 64	ReLU
block1_pool	16×16	Max 2×2 , stride 2 (“same”)	
block2_conv1	16×16	3×3 conv, 128	ReLU
block2_conv2	16×16	3×3 conv, 128	ReLU
block2_pool	8×8	Max 2×2 , stride 2 (“same”)	
block3_conv1	8×8	3×3 conv, 256	ReLU
block3_conv2	8×8	3×3 conv, 256	ReLU
block3_conv3	8×8	3×3 conv, 256	ReLU
block3_conv4	8×8	3×3 conv, 256	ReLU
block3_pool	4×4	Max 2×2 , stride 2 (“same”)	
block4_conv1	4×4	3×3 conv, 512	ReLU
block4_conv2	4×4	3×3 conv, 512	ReLU
block4_conv3	4×4	3×3 conv, 512	ReLU
block4_conv4	4×4	3×3 conv, 512	ReLU
block4_pool	2×2	Max 2×2 , stride 2 (“same”)	
block5_conv1	2×2	3×3 conv, 512	ReLU
block5_conv2	2×2	3×3 conv, 512	ReLU
block5_conv3	2×2	3×3 conv, 512	ReLU
block5_conv4	2×2	3×3 conv, 512	ReLU
flatten	1×1	2×2×512	
fc_cifa10	1×1	2048×4096	ReLU
		Dropout 0.5	
fc2	1×1	4096×4096	ReLU
		Dropout 0.5	
predictions_cifa10	1×1	4096×10	softmax

■ weight file: [vgg19\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels.h5](#)

- **Object recognition system:**
  - Images are resized such that the smallest dimension becomes 224
  - then the center  $224 \times 224$  crop is used
  
- **Training hyperparameters:**
  - Method: SGD with Nesterov momentum `momentum=0.9, nesterov=True`
  - Data preprocessing: **subtract RGB mean value**
    - ◆ B = 103.939
    - ◆ G = 116.779
    - ◆ R = 123.680
  - Weight initialization:
    - ◆ **He's WI**
    - ◆ **Random initialization**
      - conv layer init = `random_normal(stddev = 0.03)`
      - fc layer init = `random_normal(stddev = 0.01)`
  - batch size: **128** (**391** iterations for each epoch)
  - Total epochs: **164**
  - Loss function: cross-entropy
  - Initial learning rate: **0.01**, divide by 10 at 81, 122 epoch
  - Weight Decay: `kernel_regularizer=keras.regularizers.l2(0.0001)` also using `0.0005, 0.0010, 0.0013, 0.0015`
  - Dropout: 0.5
  - Batch normalization: **w/wo**
  
- **Data augmentation:**
  - Translation: Pad 4 zeros in each side and random cropping back to 32x32 size
  - Horizontal flipping with probability 0.5







# Result

## Object recognition system:

```
Please input picture file to predict ( input Q to exit ): test_pic/puzzle.jpeg
Predicted: [('n03598930', 'jigsaw_puzzle', 0.99833006), ('n04033995', 'quilt', 0.00020523704), ('n03908714', 'pencil_sharpener', 0.00015228854), ('n04116512', 'rubber_eraser', 0.00014336959), ('n03291819', 'envelope', 0.00013071252)]
Please input picture file to predict ( input Q to exit ): test_pic/tiger.jpeg
Predicted: [('n02129604', 'tiger', 0.82527125), ('n02123159', 'tiger_cat', 0.15637144), ('n02128385', 'leopard', 0.00034659068)]
Please input picture file to predict ( input Q to exit ): test_pic/cat.jpg
Predicted: [('n02124075', 'Egyptian_cat', 0.912691), ('n02123045', 'tabby', 0.070905864), ('n02123159', 'tiger_cat', 0.014582842), ('n02127052', 'lynx', 0.00025489207), ('n01704323', 'triceratops', 0.00015764245)]
```

Figure 1: Result of VGG19 prediction

Table 2: Result of VGG19 prediction

		('n03598930', ' <b>jigsaw_puzzle</b> ', <b>0.99833006</b> ), ( 'n04033995', 'quilt', 0.00020523704), ( 'n03908714', 'pencil_sharpener', 0.00015228854), ( 'n04116512', 'rubber_eraser', 0.00014336959), ( 'n03291819', 'envelope', 0.00013071252)
		('n02129604', ' <b>tiger</b> ', <b>0.82527125</b> ), ( 'n02123159', 'tiger_cat', 0.15637144), ( 'n02128925', 'jaguar', 0.012029115), ( 'n02391049', 'zebra', 0.0048657954), ( 'n02128385', 'leopard', 0.00034659068)
		('n02124075', ' <b>Egyptian_cat</b> ', <b>0.912691</b> ), ( 'n02123045', 'tabby', 0.070905864), ( 'n02123159', 'tiger_cat', 0.014582842), ( 'n02127052', 'lynx', 0.00025489207), ( 'n01704323', 'triceratops', 0.00015764245)

## Retrain On Cifar-10:

### ■ Final Accuracy (test error)

From the figures, we can find that the **convergence rate** of random initialization slower than retrain model. We can learn that Vgg19's weight is very effective.

And because I set the WD(weight decay) to 0.0001, the overfitting is very serious.

Table 3: Test accuracy for pure random initialization and pure retrain models

<b>pure random initialization models</b>	<b>90.99%</b>
<b>pure retrain models</b>	<b>92.38%</b>

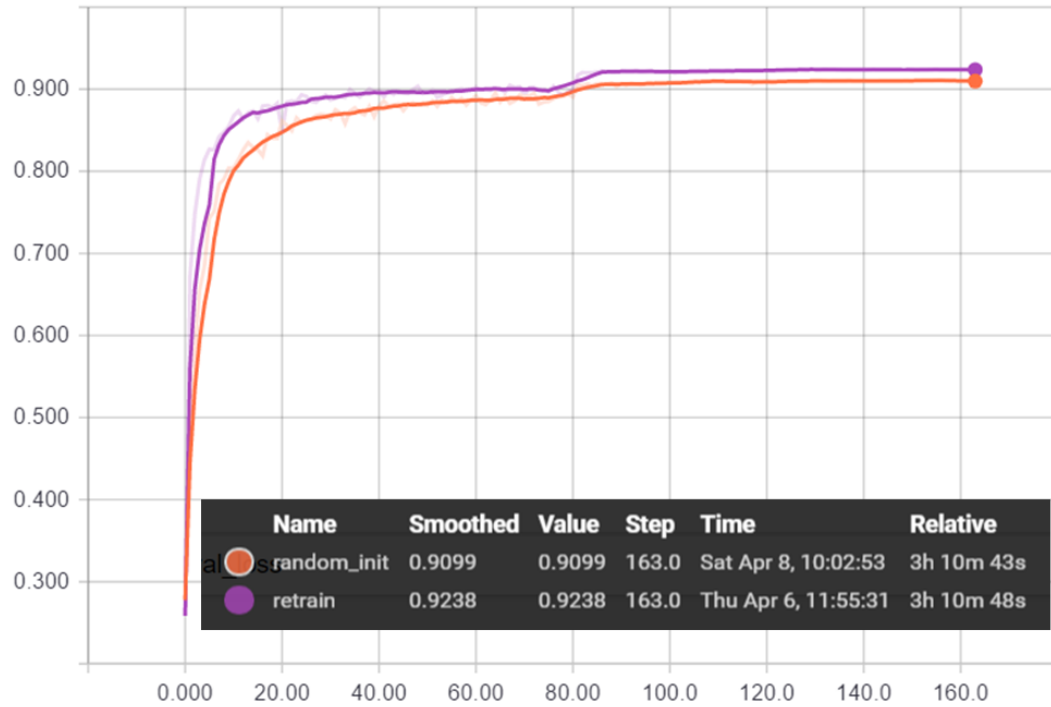


Figure 2: Test accuracy curve for pure random initialization and pure retrain models

#### ■ Training loss curve:

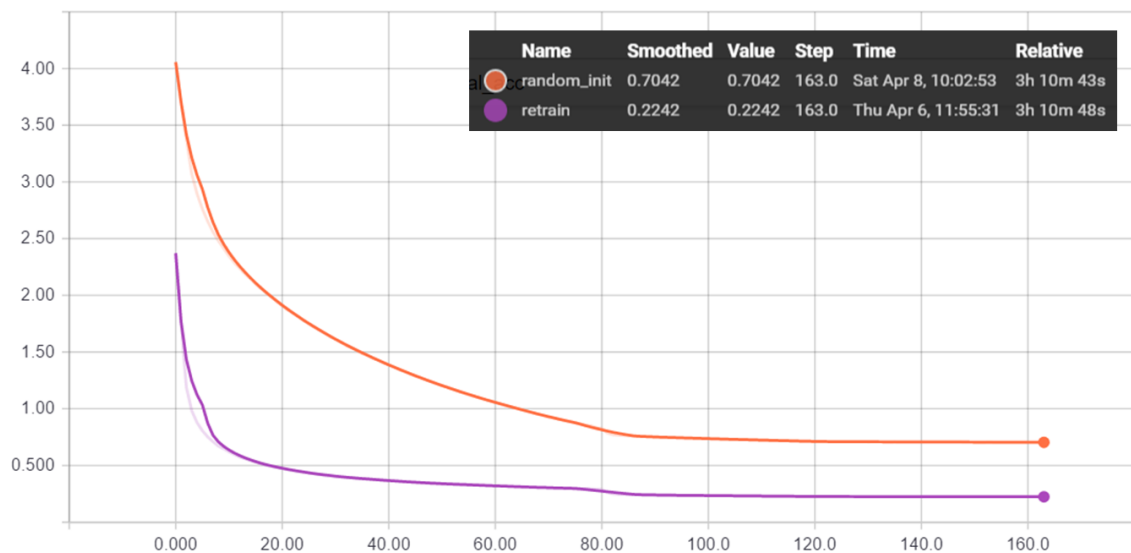


Figure 3: Training loss curve for pure random initialization and pure retrain models

## ■ Test loss curve

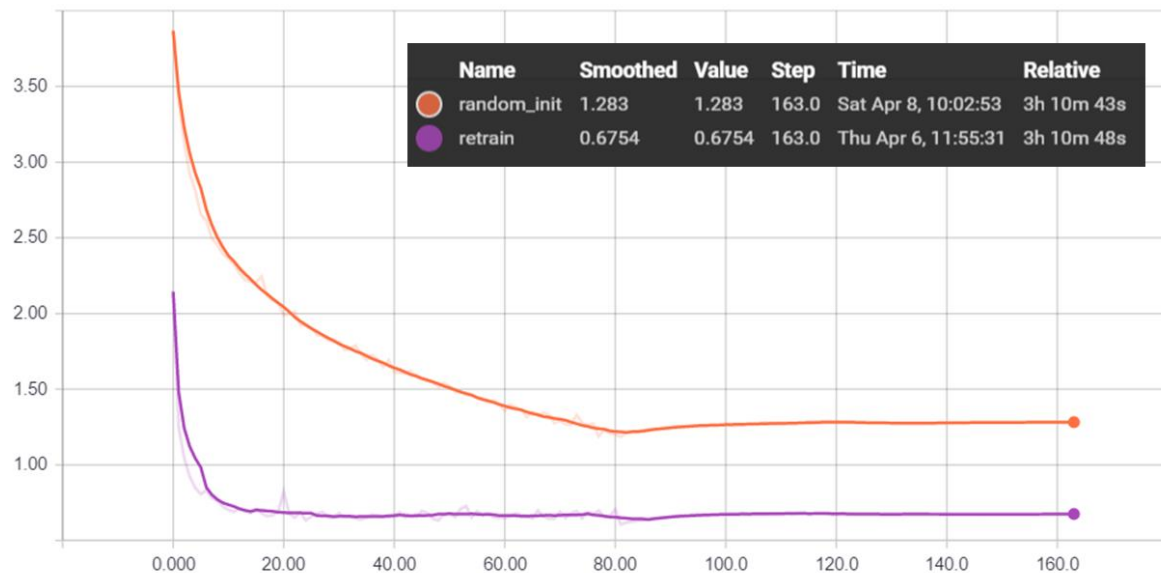


Figure 4: Test loss curve for pure random initialization and pure retrain models

## Other experiments

### ■ RI and Retrain w/wo WI

Table 4: Test accuracy for pure RI and pure retrain with WI

<b>pure RI</b>	<b>90.99%</b>
<b>pure RI + WI</b>	<b>91.13%</b>
<b>pure retrain</b>	<b>92.38%</b>
<b>pure retrain + WI</b>	<b>92.36%</b>

Name	Smoothed	Value	Step	Time	Relative
random_init	0.9099	0.9099	163.0	Sat Apr 8, 10:02:53	3h 10m 43s
random_init+WI	0.9113	0.9113	163.0	Thu Apr 6, 18:31:37	3h 10m 11s
retrain	0.9238	0.9238	163.0	Thu Apr 6, 11:55:31	3h 10m 48s
retrain+WI	0.9236	0.9236	163.0	Thu Apr 6, 15:08:45	3h 11m 14s

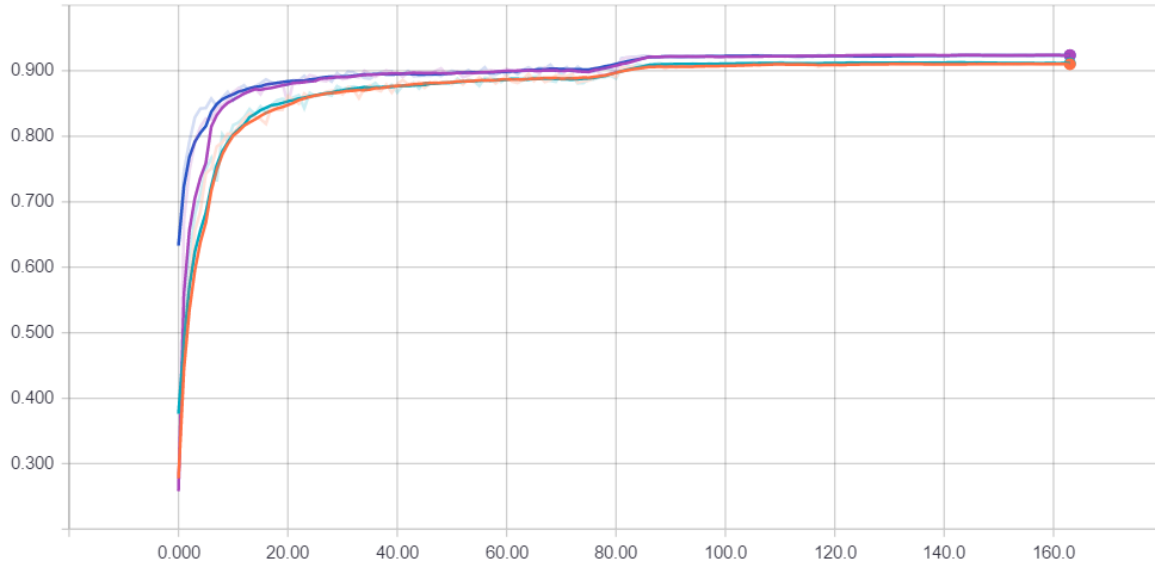


Figure 5: The accuracy for pure RI and pure retrain with WI

## ■ Retrain + WI + WD

Name	Smoothed	Value	Step	Time	Relative
retrain+WI	0.9236	0.9236	163.0	Thu Apr 6, 15:08:45	3h 11m 14s
retrain+WI+WD0.0005	0.9218	0.9218	163.0	Fri Apr 7, 06:53:42	3h 10m 38s
retrain+WI+WD0.001	0.9217	0.9217	163.0	Fri Apr 7, 21:16:19	3h 10m 55s
retrain+WI+WD0.0015	0.9189	0.9189	163.0	Sat Apr 8, 04:45:50	3h 10m 51s

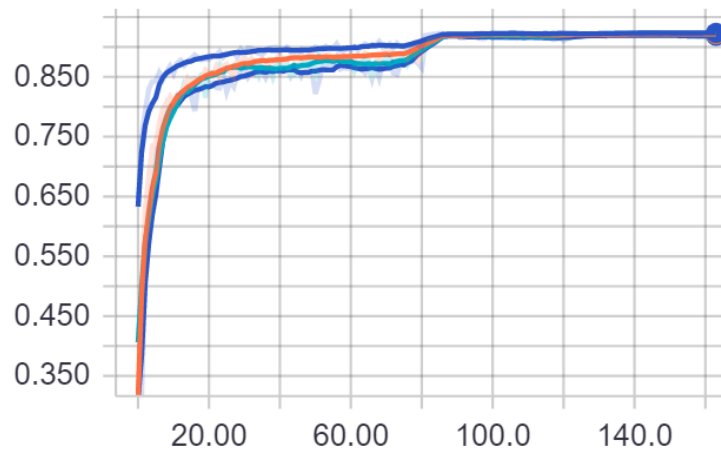


Figure 6: Test accuracy curve for Retrain + WI + WD

Table 5: Test accuracy for Retrain + WI + WD

<b>Retrain+WI</b>	<b>WD=0.0001</b>	<b>92.36%</b>
<b>Retrain+WI</b>	<b>WD=0.0005</b>	<b>92.18%</b>
<b>Retrain+WI</b>	<b>WD=0.0010</b>	<b>92.17%</b>
<b>Retrain+WI</b>	<b>WD=0.0015</b>	<b>91.89%</b>

■ **Retrain + WI + WD + BN**

Name	Smoothed	Value	Step	Time	Relative
retrain+WI+BN	0.9299	0.9299	163.0	Thu Apr 6, 22:34:09	3h 59m 53s
retrain+WI+BN+WD0.0005	0.9374	0.9374	163.0	Fri Apr 7, 03:41:45	4h 0m 29s
retrain+WI+BN+WD0.001	0.9394	0.9394	163.0	Fri Apr 7, 18:04:05	4h 0m 21s
retrain+WI+BN+WD0.0013	0.9383	0.9383	163.0	Sat Apr 8, 14:04:51	4h 0m 16s
retrain+WI+BN+WD0.0015	0.9414	0.9414	163.0	Sat Apr 8, 01:33:41	4h 0m 30s

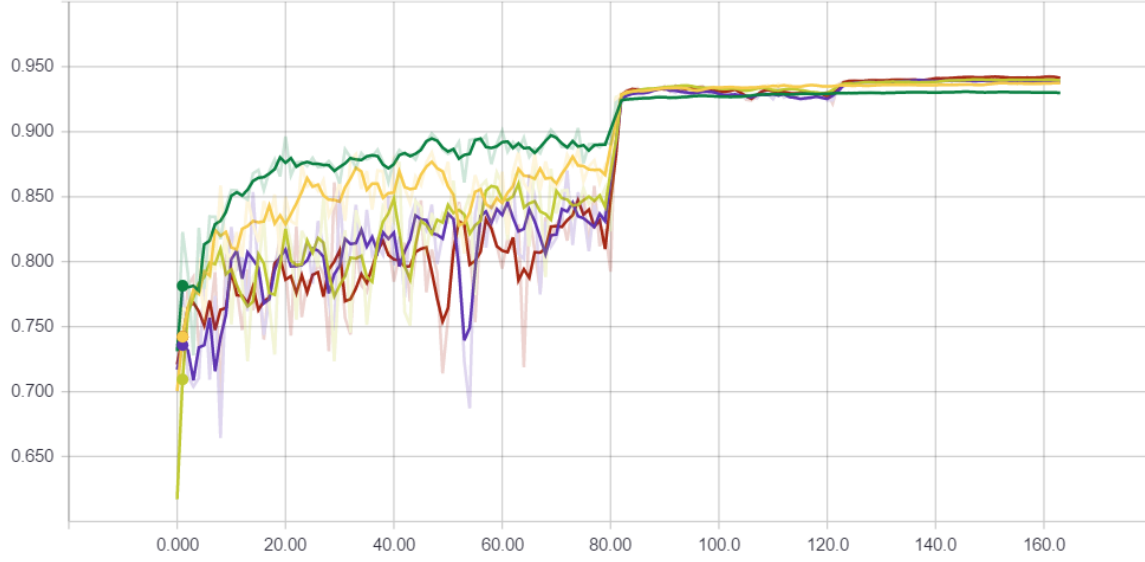


Figure 7: Test accuracy curve for Retrain + WI + WD + **BN**

Table 6: Test accuracy for Retrain + WI + WD + **BN**

<b>Retrain+WI+BN</b>	<b>WD=0.0001</b>	<b>92.99%</b>
<b>Retrain+WI+BN</b>	<b>WD=0.0005</b>	<b>93.74%</b>
<b>Retrain+WI+BN</b>	<b>WD=0.0010</b>	<b>93.94%</b>
<b>Retrain+WI+BN</b>	<b>WD=0.0013</b>	<b>93.83%</b>
<b>Retrain+WI+BN</b>	<b>WD=0.0015</b>	<b>94.14%</b>



# Discussion

## About overfitting:

According to Table 6 and Figure 7 & 8, overfitting will occur if the WD is too small (after epoch 81, the accuracy of training data will reach 0.99 soon, overfitting occurred).

In other words, increase WD's value can help us to avoid overfitting and get better results.

In Retrain+WI+BN+WD0.0015, we got the good test accuracy results 94.14%.

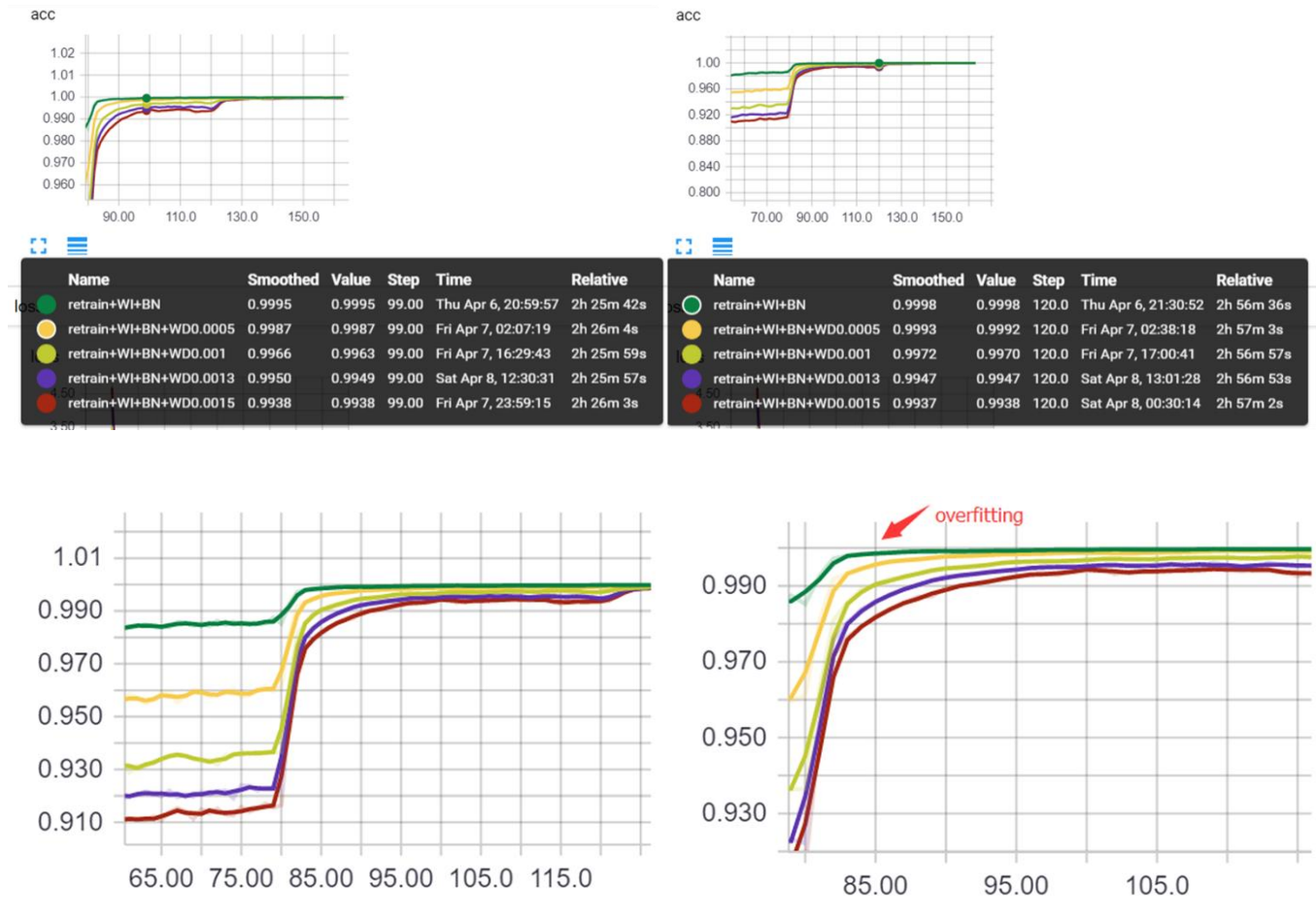


Figure 8: overfitting of the retrained models