Fast R-CNN

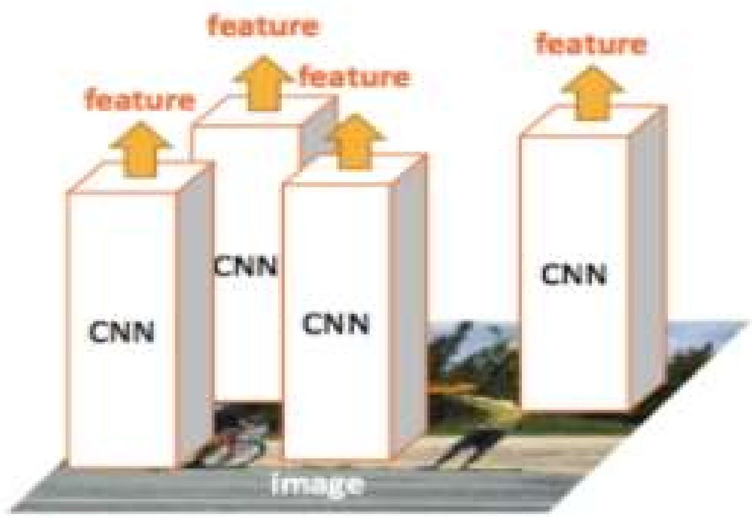
2021210088 허지혜

1. Abstract & Introduction

Fast R-CNN(Fast Region-based Convolutional Network method) Review.

- It is mainly used for Object Detection.
- Compared to R-CNN, training and testing speed were increased and detection accuracy was improved.
- In pre-trained VGG16, Fast R-CNN achieved higher mAP than R-CNN.
- It returns faster and more accurate results than SPPnet.

1. Abstract & Introduction



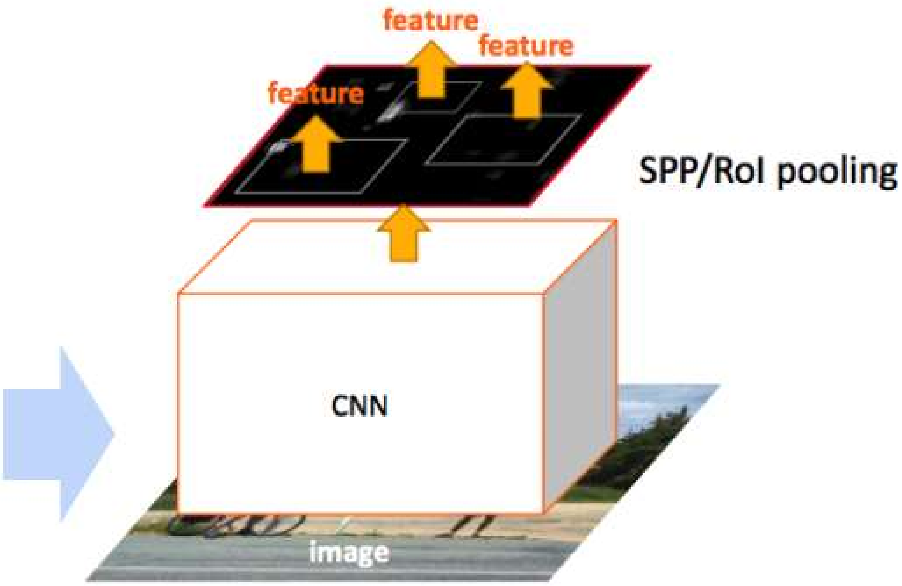
R-CNN

- Extract image regions
- 1 CNN per region (2000 CNNs)
- Classify region-based features
- Complexity: $\sim 224 \times 224 \times 2000$

Problem with R-CNN

1. A multi-stage pipleline exists during learning.
2. The time and space costs of training are huge.
3. OD itself is slow.

1. Abstract & Introduction



SPP-net & Fast R-CNN (the same forward pipeline)

- 1 CNN on the entire image
- Extract features from feature map regions
- Classify region-based features
- Complexity: $\sim 600 \times 1000 \times \mathbf{1}$
- ~ 160**x faster** than R-CNN

Better than R-CNN, SPPnet

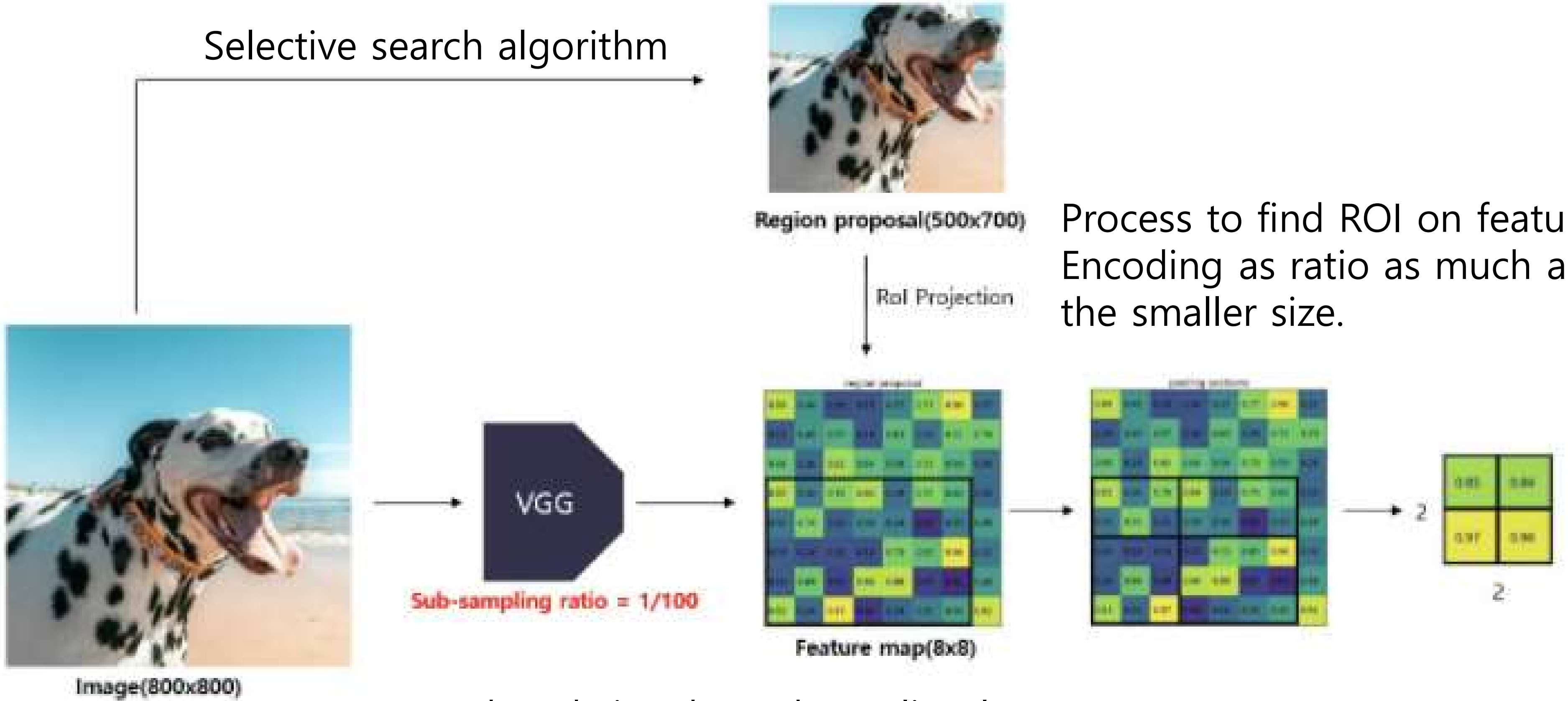
- It is not necessary to operate too many CNNs to learn one image by dividing one image into multiple scales and putting it in CNN at a time.
- Spatial Pyramid Pooling use.

Better than SPPnet, Fast R-CNN

- ROI(Region of Interest) Pooling use.
- Fixed size feature vector deliver fc layer.
- Use multi-task loss to train models at once without having to train them individually.

2. Main Ideas

- ROI(Region of Interest) Pooling



Process to find ROI on feature map.
Encoding as ratio as much as
the smaller size.

Reduced size through pooling layer

2. Main Ideas

- Multi-task loss

Classification loss

Localization loss

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v)$$

2. Main Ideas

- Multi-task loss

$$L(p, u, t^u, v) = L_{cls}(p, u)$$

$P = (p^0, p^1, \ldots.., p^k) : K + 1\ Class\ score$

$u : ground\ truth\ Class\ score$

Classification loss

$$L_{cls}(p, u) = -log p_u$$

2. Main Ideas

- Multi-task loss

$t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$: *predicted bounding box*

$\text{v} = (v_x\ , v_y\ , v_w\ , v_h\ )$: *true bounding box*

$$\lambda[u \geq 1]L_{loc}(t^u, v)$$

1

$$\lambda[u \geq 1]$$

Lamda : A balancing hyperparamter that adjusts the weights between the two losses.

[u>1] : Iverson bracket indicator function.

2

$$L_loc(t^u, v) = \sum_{i \in \{x,y,w,h\}} smooth_{L_1}(t_i^u - v_i)$$

$$smooth_{L_1}(t_i^u - v_i) = \begin{cases} 0.5x^2, if|x| < 1 \\ |x| - 0.5, otherwise \end{cases}$$

Smooth L1 loss use.
Because L1 loss is less sensitive to outliers.

2. Main Ideas

- Hierarchical sampling

Hierachical sampling : positive feature sharing.

When constructing the SGD mini-batch,
N images are sampling and a total of R region proposals are used.
This is a method of sampling R/N region proposals from each image.

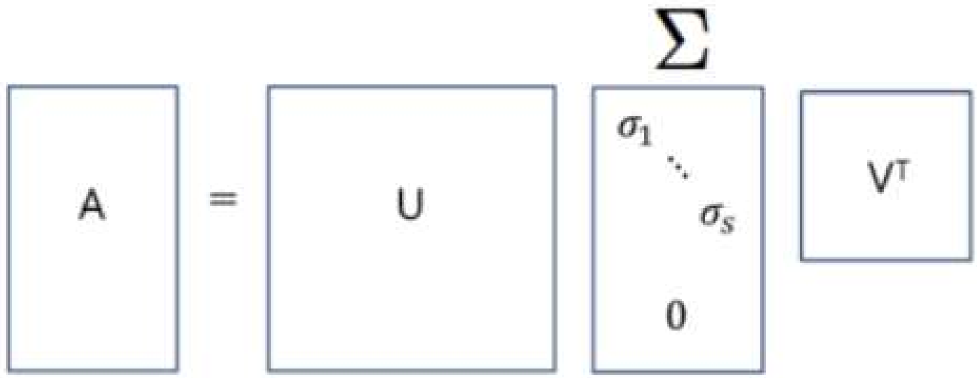
=>Through this, region proposals extracted from the same image **can share computation And memory** during forward and backward propagation.

This paper,
N =2, R=128 -> get 64 region proposals sampling
25% image – over ground truth IoU 0.5 → u =1
75% image – 0.1<ground truth IoU < 0.5 → u =0

2. Main Ideas
- Truncated SVD

Detection time reduction

Through Truncated SVD,
Compress the fc layer.

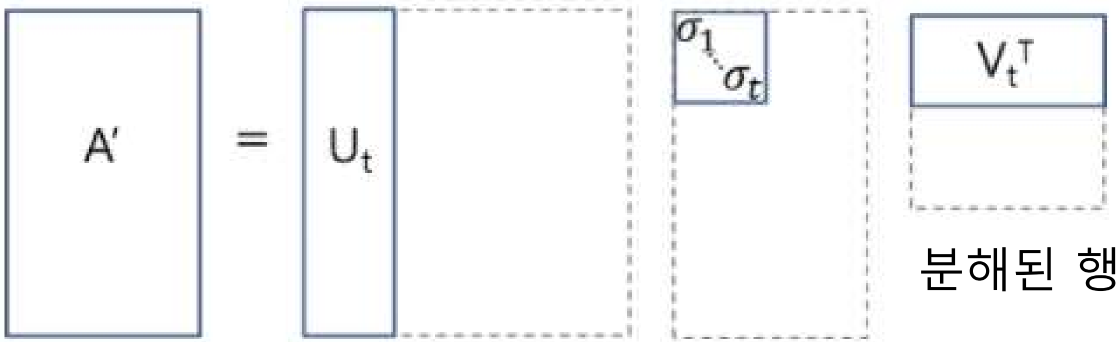


Full SVD

2nd fc layer | 1st fc layer

uxv $W \approx U\Sigma_t V^T$ t(u+v)

Approximate the weight
of the fc layer



분해된 행렬의 일부만 사용

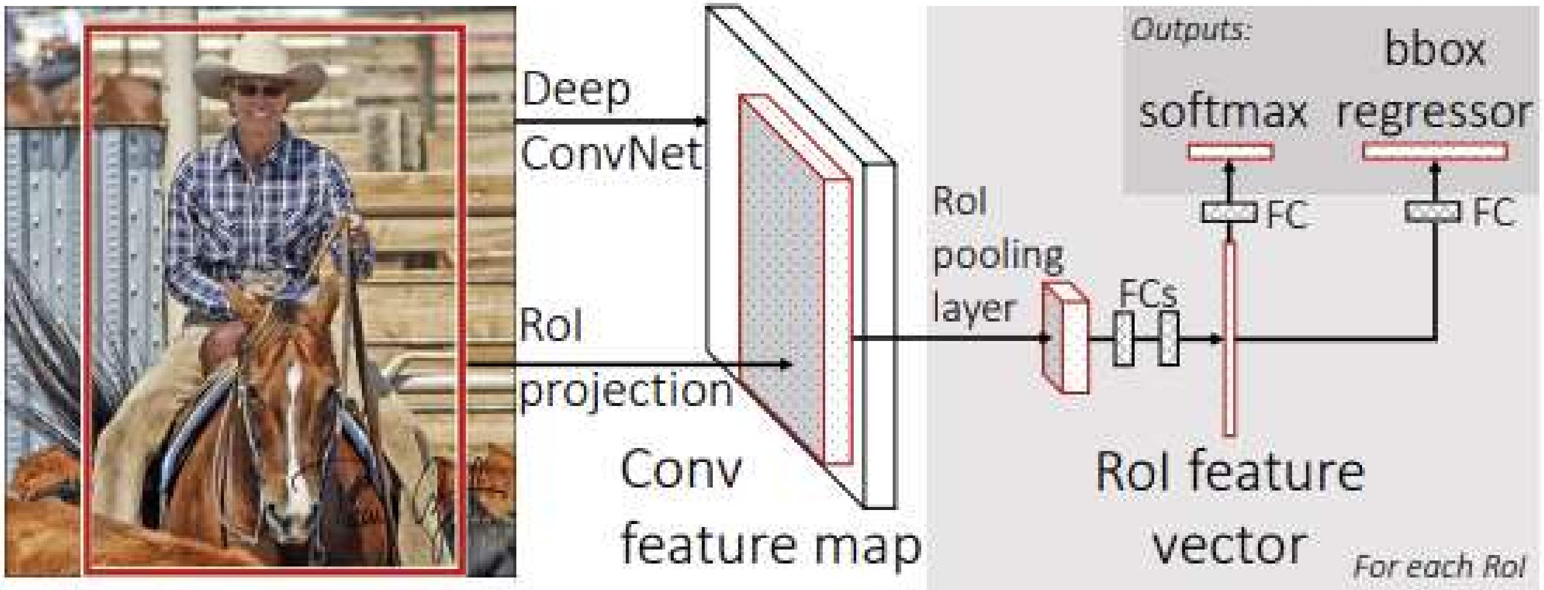
Truncated SVD

3. Training Fast R-CNN

Fast R-CNN Characteristic

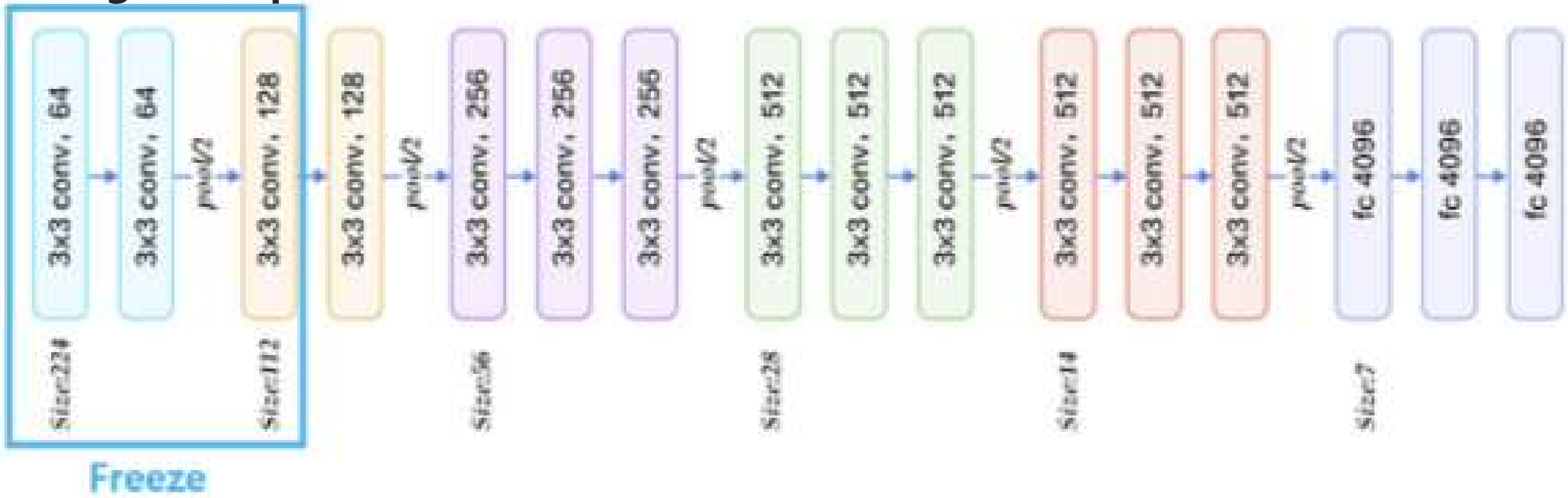
1. Detection quality is higher than R-CNN, SPPnet
2. Learning is done in a single stage, multi-task loss is used.
3. Through learning, it is possible to update all network layers.
4. Disk storage for feature caching is no longer required.

2. Training Fast R-CNN



3. Training Fast R-CNN

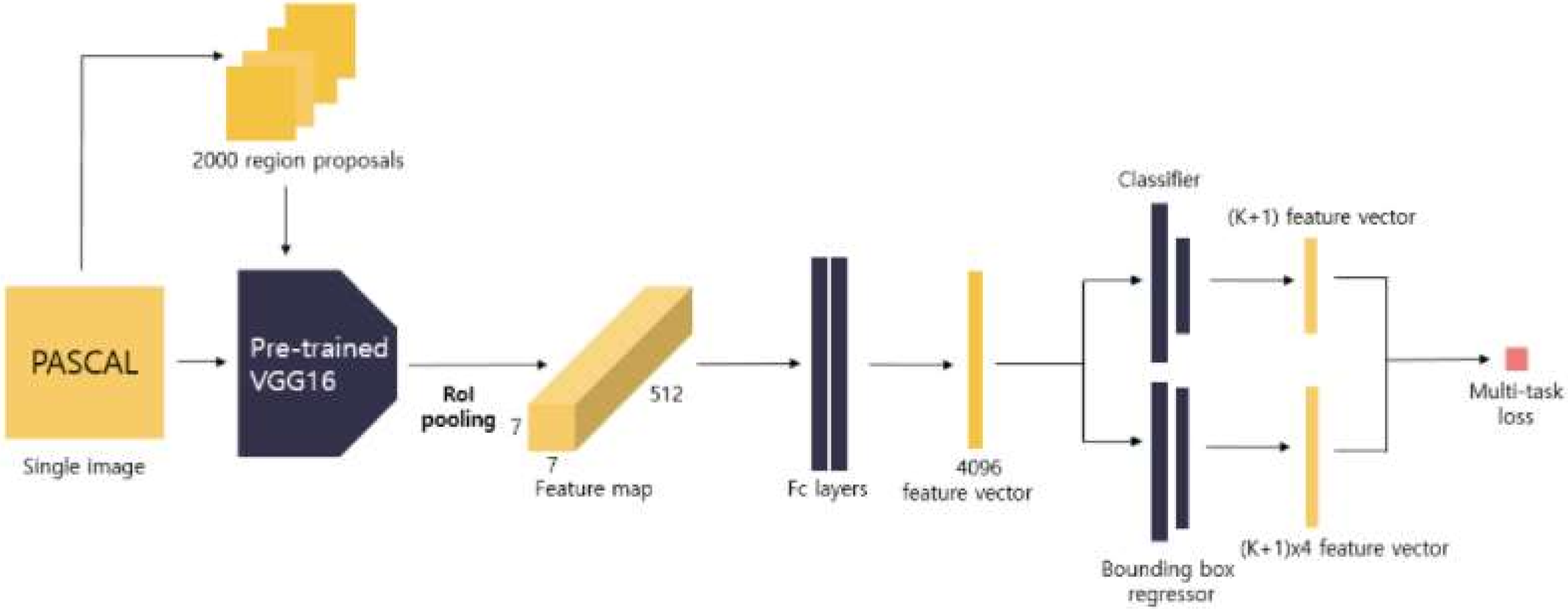
- Initializing from pretrained Networks



1. Max pooling layer -> ROI pooling layer change(feature map size : 7x7)
2. FC layer -> two FC layers
   - K class + 1 background = K+1 output
   - Class bounding box = (K+1) * 4 output

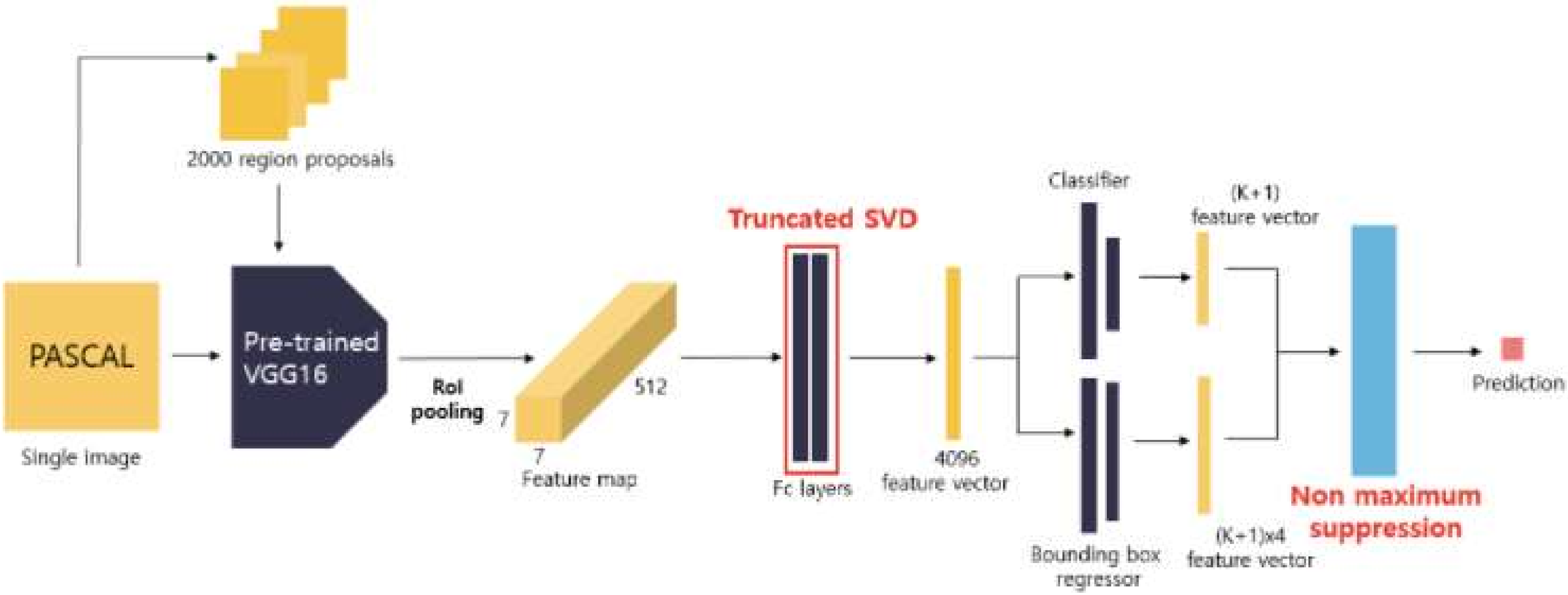
3. Network weight freeze, layer -> fine tuning
4. image, region proposals

3. Training Fast R-CNN
- Feature vsctor extraction by Fc layers



3. Training Fast R-CNN

- Detection Fast R-CNN



4. Main Results

Three main results support this paper’s contributions :

1. State-of-the-art mAP on VOC07, 2010 and 2012
2. Fast training and testing compared to R-CNN, SPPnet
3. Fine-tuning conv layers in VGG16 improves mAP

4. Main Results

VOC 2007, 2010 and 2012 Results

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [11]†	07 \ diff	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	73.5	63.1
R-CNN BB [10]	07	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	**35.6**	66.8	67.2	70.4	**71.1**	66.0
FRCN [ours]	07	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8	66.9
FRCN [ours]	07 \ diff	74.6	**79.0**	68.6	57.0	39.3	79.5	**78.6**	81.9	**48.0**	74.0	67.4	80.5	80.7	74.1	69.6	31.8	67.1	68.4	75.3	65.5	68.1
FRCN [ours]	07+12	**77.0**	78.1	**69.3**	**59.4**	38.3	**81.6**	**78.6**	**86.7**	42.8	**78.8**	**68.9**	**84.7**	**82.0**	**76.6**	**69.9**	31.8	**70.1**	**74.8**	**80.4**	70.4	**70.0**

Table 1. **VOC 2007 test** detection average precision (%). All methods use VGG16. Training set key: **07**: VOC07 trainval, **07 \ diff**: **07** without "difficult" examples, **07+12**: union of **07** and VOC12 trainval. †SPPnet results were prepared by the authors of [11].

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	77.7	73.8	62.3	48.8	45.4	67.3	67.0	80.3	41.3	70.8	49.7	79.5	74.7	78.6	64.5	36.0	69.9	55.7	70.4	61.7	63.8
R-CNN BB [10]	12	79.3	72.4	63.1	44.0	44.4	64.6	66.3	84.9	38.8	67.3	48.4	82.3	75.0	76.7	65.7	35.8	66.2	54.8	69.1	58.8	62.9
SegDeepM	12+seg	**82.3**	75.2	67.1	50.7	**49.8**	71.1	69.6	88.2	42.5	71.2	50.0	85.7	76.6	81.8	69.3	**41.5**	**71.9**	62.2	73.2	**64.6**	67.2
FRCN [ours]	12	80.1	74.4	67.7	49.4	41.4	74.2	68.8	87.8	41.9	70.1	50.2	86.1	77.3	81.1	70.4	33.3	67.0	63.3	77.2	60.0	66.1
FRCN [ours]	07++12	82.0	**77.8**	**71.6**	**55.3**	42.4	**77.3**	**71.7**	**89.3**	**44.5**	**72.1**	**53.7**	**87.7**	**80.0**	**82.5**	**72.7**	36.6	68.7	**65.4**	**81.1**	62.7	**68.8**

Table 2. **VOC 2010 test** detection average precision (%). BabyLearning uses a network based on [17]. All other methods use VGG16. Training set key: **12**: VOC12 trainval, **Prop.**: proprietary dataset, **12+seg**: **12** with segmentation annotations, **07++12**: union of VOC07 trainval, VOC07 test, and VOC12 trainval.

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6	63.2
NUS_NIN_c2000	Unk.	80.2	73.8	61.9	43.7	**43.0**	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	**38.6**	**68.3**	58.0	68.7	63.3	63.8
R-CNN BB [10]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
FRCN [ours]	12	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7	65.7
FRCN [ours]	07++12	**82.3**	**78.4**	**70.8**	**52.3**	38.7	**77.8**	**71.6**	**89.3**	**44.2**	**73.0**	**55.0**	**87.5**	**80.5**	**80.8**	**72.0**	35.1	**68.3**	**65.7**	**80.4**	**64.2**	**68.4**

Table 3. **VOC 2012 test** detection average precision (%). BabyLearning and NUS_NIN_c2000 use networks based on [17]. All other methods use VGG16. Training set key: see Table 2, **Unk.**: unknown.

Three training ImageNet Model

Get R-CNN,

1. CaffeNet -> S

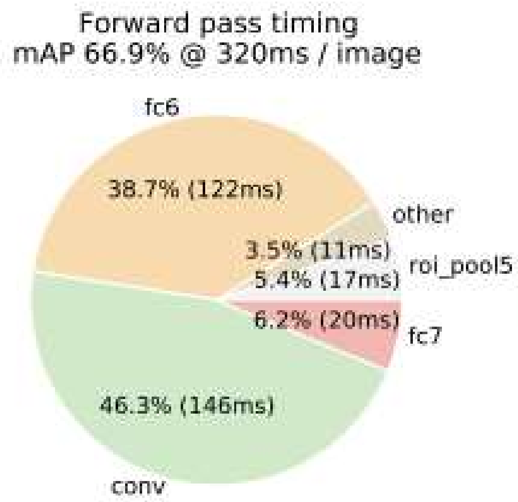
2. VGG_CNN_M_1024 -> M

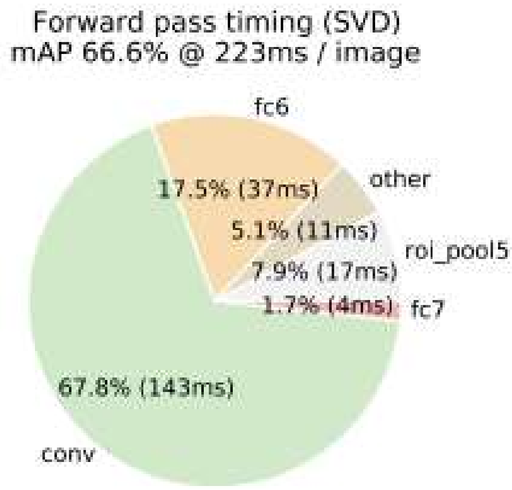
3. VGG16 -> L

4. Main Results

Training and testing time

	Fast R-CNN			R-CNN			SPPnet
	S	**M**	**L**	**S**	**M**	**L**	†**L**
train time (h)	**1.2**	2.0	9.5	22	28	84	25
train speedup	**18.3**×	14.0×	8.8×	1×	1×	1×	3.4×
test rate (s/im)	0.10	0.15	0.32	9.8	12.1	47.0	2.3
> with SVD	**0.06**	0.08	0.22	-	-	-	-
test speedup	98×	80×	146×	1×	1×	1×	20×
> with SVD	169×	150×	**213**×	-	-	-	-
VOC07 mAP	57.1	59.2	**66.9**	58.5	60.2	66.0	63.1
> with SVD	56.5	58.7	66.6	-	-	-	-





4. Main Results

Which layers to fine-tune?

	layers that are fine-tuned in model **L**			SPPnet **L**
	$\geq$ fc6	$\geq$ conv3_1	$\geq$ conv2_1	$\geq$ fc6
VOC07 mAP	61.4	66.9	**67.2**	63.1
test rate (s/im)	**0.32**	**0.32**	**0.32**	2.3

Do I need to fine-tune all the conv layers for better performance?   No.

S,M : Relatively shallow network
L : Deep network

This table show FC layer fine-tuning and conv layers fine-tuning.

- SPPnet : FC layers only fine-tuning
- Fast R-CNN : conv layers fine-tuning

Fine-tuning starts conv3_1, because in conv2_1, the mAP increased by 0.3%.
And in conv1_1, GPU memory soar up.

5. Design Evaluation

Does Multi-task Training Help?

Basic PASCAL VOC07 dataset testing

	S				M				L			
multi-task training?		✓		✓		✓		✓		✓		✓
stage-wise training?			✓				✓				✓	
test-time bbox reg?			✓	✓			✓	✓			✓	✓
VOC07 mAP	52.2	53.3	54.6	**57.1**	54.7	55.5	56.6	**59.2**	62.6	63.4	64.0	**66.9**

We check multi-task training higher accuracy.

5. Design Evaluation

Scale Invariance : to Brute Force or Finesse?

	SPPnet **ZF**		**S**		**M**		**L**
scales	1	5	1	5	1	5	1
test rate (s/im)	0.14	0.38	**0.10**	0.39	0.15	0.64	0.32
VOC07 mAP	58.0	59.2	57.1	58.4	59.2	60.7	**66.9**

이미지가 training 및 testing 시 지정된
특정 픽셀 사이즈로 고정됨을 뜻한다.

Brute-Force Approach : 1 scale -> pixel size 600
Multi-Scale Approach : 5 scale {480, 576, 688, 865, 1200}

S,M : Multi-Scale Approach win.
L : Brute-Force Approach win.

image pyramid를 사용하여 네트워크에
대략적인 scale-invariance(이미지의 크
기에 상관없이 성질이 유지되는 것)를
부여한다

5. Design Evaluation

Et al.

Many Training data → mAP increase

SVM vs Softmax → It has better performance in L

Many object proposals → The performance of mAP
Does not increase.

END

https://herbwood.tistory.com/8
https://arxiv.org/pdf/1504.08083.pdf
https://noru-jumping-in-the-mountains.tistory.com/14
https://chacha95.github.io/2020-02-14-Object-Detection2/