Pruning and Quantization for Lightweight Neural Network

https://youtu.be/sn6c-Ksn-_o

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

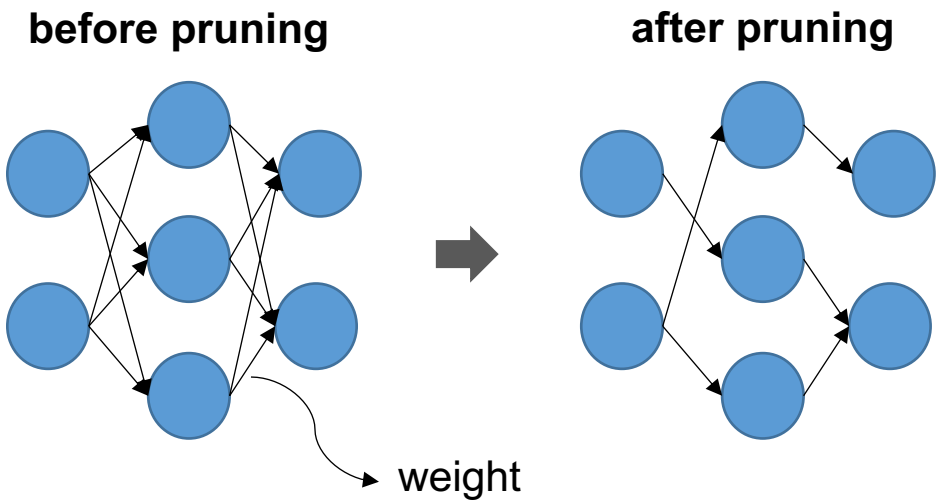
Contents

CryptoCraft LAB

Pruning

- Neural Network의 크기를 줄이는 것
  - over-parameterized 된 모델에서 불필요한 부분을 중요도에 따라 제거
  - filter pruning, weight pruning and others
- parameter가 많은 경우 하드웨어 리소스가 제한된 임베디드 시스템에 배포하기 어려움

  → pruning 통해 동일한 정확도, 빠른 처리 속도, 전력 소모 및 모델 용량 감소 효과



Deep Compression

- 정확도 손실 없이 모델 크기 감소
  - pruning 외에도 quantization, huffman coding 도 모델 축소에 사용
  - fully connected layer가 있는 모델의 경우 35x ~ 49x 감소
  - CNN의 경우에도 10배까지 감소 가능하다고 함
- pruning만 적용할 경우에도 9x ~ 11x 감소

Table 4: Compression statistics for AlexNet. P: pruning, Q: quantization, H:Huffman coding.

Layer	#Weights	Weights% (P)	Weight bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1	35K	84%	8	6.3	4	1.2	32.6%	20.53%
conv2	307K	38%	8	5.5	4	2.3	14.5%	9.43%
conv3	885K	35%	8	5.1	4	2.6	13.1%	8.44%
conv4	663K	37%	8	5.2	4	2.5	14.1%	9.11%
conv5	442K	37%	8	5.6	4	2.5	14.0%	9.43%
fc6	38M	9%	5	3.9	4	3.2	3.0%	2.39%
fc7	17M	9%	5	3.6	4	3.7	3.0%	2.46%
fc8	4M	25%	5	4	4	3.2	7.3%	5.85%
Total	61M	11%(9×)	5.4	4	4	3.2	3.7% (**27**×)	2.88% (**35**×)

Table 5: Compression statistics for VGG-16. P: pruning, Q:quantization, H:Huffman coding.

Layer	#Weights	Weights% (P)	Weigh bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1_1	2K	58%	8	6.8	5	1.7	40.0%	29.97%
conv1_2	37K	22%	8	6.5	5	2.6	9.8%	6.99%
conv2_1	74K	34%	8	5.6	5	2.4	14.3%	8.91%
conv2_2	148K	36%	8	5.9	5	2.3	14.7%	9.31%
conv3_1	295K	53%	8	4.8	5	1.8	21.7%	11.15%
conv3_2	590K	24%	8	4.6	5	2.9	9.7%	5.67%
conv3_3	590K	42%	8	4.6	5	2.2	17.0%	8.96%
conv4_1	1M	32%	8	4.6	5	2.6	13.1%	7.29%
conv4_2	2M	27%	8	4.2	5	2.9	10.9%	5.93%
conv4_3	2M	34%	8	4.4	5	2.5	14.0%	7.47%
conv5_1	2M	35%	8	4.7	5	2.5	14.3%	8.00%
conv5_2	2M	29%	8	4.6	5	2.7	11.7%	6.52%
conv5_3	2M	36%	8	4.6	5	2.3	14.8%	7.79%
fc6	103M	4%	5	3.6	5	3.5	1.6%	1.10%
fc7	17M	4%	5	4	5	4.3	1.5%	1.25%
fc8	4M	23%	5	4	5	3.4	7.1%	5.24%
Total	138M	7.5%(13×)	6.4	4.1	5	3.1	3.2% (**31**×)	2.05% (**49**×)

Pruning 동향

convolution에 사용되는 filter pruning이 다수

2020

Title	Venue	Type	Code
EagleEye: Fast Sub-net Evaluation for Efficient Neural Network Pruning	ECCV **(Oral)**	F	PyTorch(Author)
DSA: More Efficient Budgeted Pruning via Differentiable Sparsity Allocation	ECCV	F	-
DHP: Differentiable Meta Pruning via HyperNetworks	ECCV	F	PyTorch(Author)
Meta-Learning with Network Pruning	ECCV	W	-
Accelerating CNN Training by Pruning Activation Gradients	ECCV	W	-
DA-NAS: Data Adapted Pruning for Efficient Neural Architecture Search	ECCV	Other	-
Differentiable Joint Pruning and Quantization for Hardware Efficiency	ECCV	Other	-
Channel Pruning via Automatic Structure Search	IJCAI	F	PyTorch(Author)
Adversarial Neural Pruning with Latent Vulnerability Suppression	ICML	W	-
Proving the Lottery Ticket Hypothesis: Pruning is All You Need	ICML	W	-
Soft Threshold Weight Reparameterization for Learnable Sparsity	ICML	WF	Pytorch(Author)
Network Pruning by Greedy Subnetwork Selection	ICML	F	-
Operation-Aware Soft Channel Pruning using Differentiable Masks	ICML	F	-
DropNet: Reducing Neural Network Complexity via Iterative Pruning	ICML	F	-
Towards Efficient Model Compression via Learned Global Ranking	CVPR **(Oral)**	F	Pytorch(Author)
HRank: Filter Pruning using High-Rank Feature Map	CVPR **(Oral)**	F	Pytorch(Author)
Neural Network Pruning with Residual-Connections and Limited-Data	CVPR **(Oral)**	F	-
Multi-Dimensional Pruning: A Unified Framework for Model Compression	CVPR **(Oral)**	WF	-
DMCP: Differentiable Markov Channel Pruning for Neural Networks	CVPR **(Oral)**	F	TensorFlow(Author)
Group Sparsity: The Hinge Between Filter Pruning and Decomposition for Network Compression	CVPR	F	PyTorch(Author)
Few Sample Knowledge Distillation for Efficient Network Compression	CVPR	F	-
Discrete Model Compression With Resource Constraint for Deep Neural Networks	CVPR	F	-
Structured Compression by Weight Encryption for Unstructured Pruning and Quantization	CVPR	W	-
Learning Filter Pruning Criteria for Deep Convolutional Neural Networks Acceleration	CVPR	F	-
APQ: Joint Search for Network Architecture, Pruning and Quantization Policy	CVPR	F	-
Comparing Rewinding and Fine-tuning in Neural Network Pruning	ICLR **(Oral)**	WF	TensorFlow(Author)

pruning 종류

- weight pruning
  - 임계값 미만의 weight를 개별적으로 제거
- structured pruning (filter pruning, channel pruning)
  - filter, channel 등과 같이 특정 구조를 가진 그룹 단위로 제거
- 일반적으로 weight pruning의 경우 더 많은 가중치가 제거될 수 있다고 함
  - 용량 대비 정확도가 높을 수 있음
  - pruning 후 sparse matrix가 됨

    → 그러나 sparse matrix라고 해도 tensorflow, pytorch 등의 라이브러리 사용 시 무조건 이득은 아님
- 즉, structured pruning이 비교적 압축 비율이 낮지만 실제 사용 시 weight pruning에 비해 더 효과적

weight pruning

임계값보다 작은 (중요하지 않은) 가중치를 모두 0으로 설정

*NN은 학습과정에서 행렬곱 연산 수행

structured pruning

- Predefined structured pruning
- Automatic structured pruning

structured pruning - Predefined structured pruning

1. 학습 후, 각 filter마다 L1 norm (l1 정규화)를 구해 값이 작은 순서대로 제거

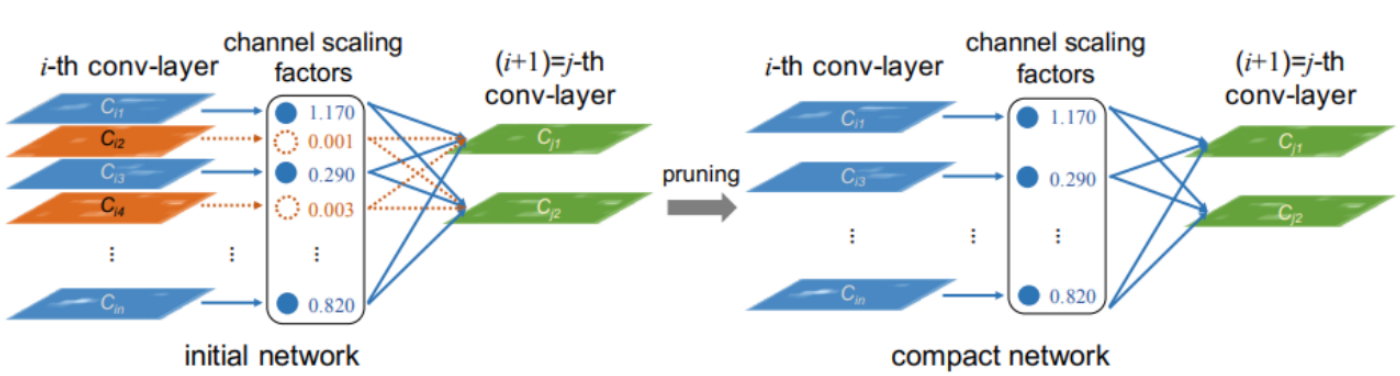
2. 다음 layer에 가장 적은 영향을 주는 channel을 제거 (weight 값 자체가 기준이 아님)

   → 제거되어도 별 차이가 없는 channel을 제거

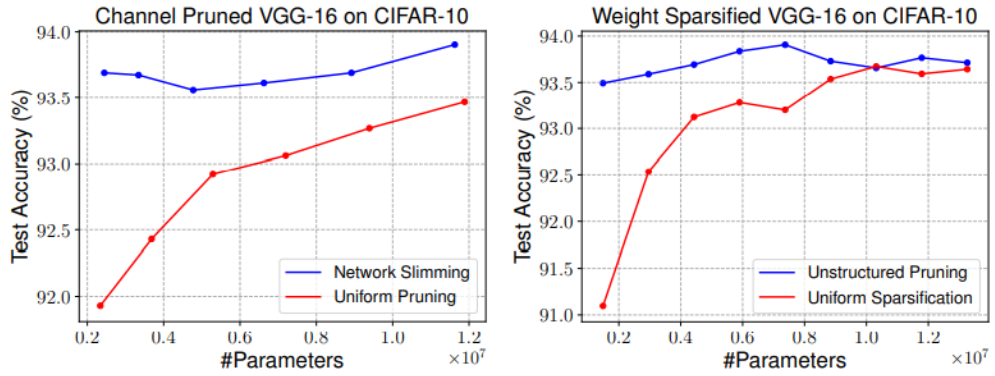
   → filter를 제거하는 것과 동일

structured pruning - Automatic structured pruning

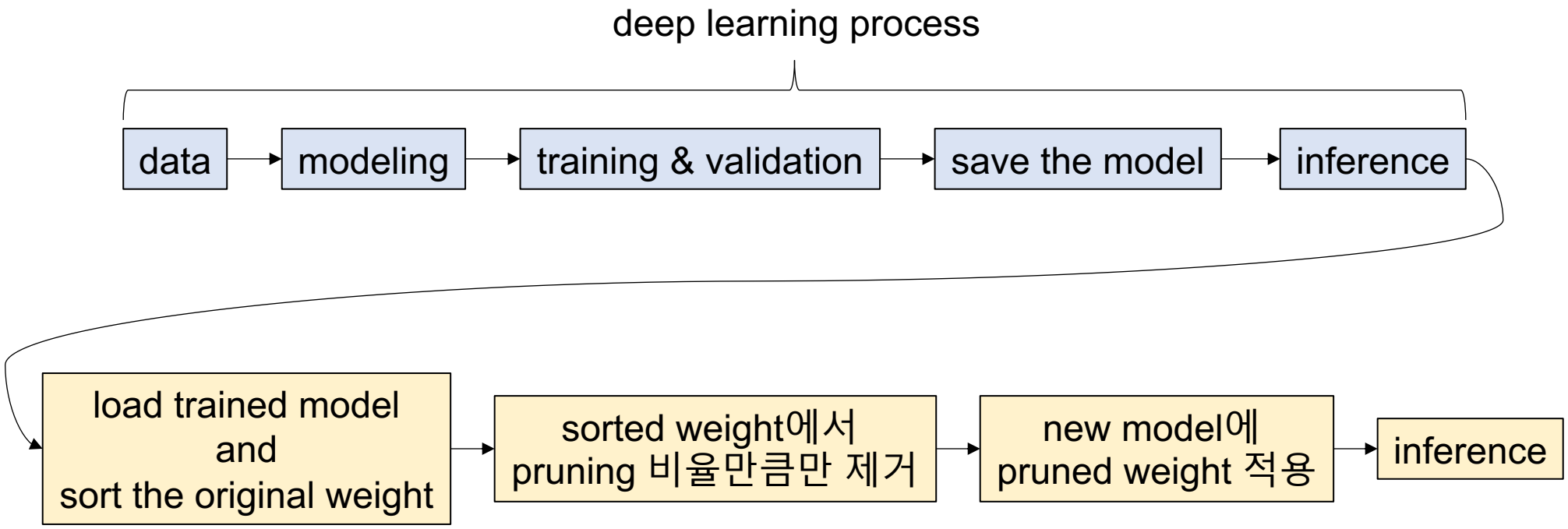
- 각 channel의 scaling factor들을 l1 정규화하여 중요하지 않은 channel들을 자동으로 식별
- 해당 channel들을 제거
  - 어떤 layer에서 몇 개의 channel이 제거될지는 알 수 없음
  - 앞 슬라이드의 channel pruning은 다음 layer에 영향을 적게 주는 channel 제거.
- auto pruning과 동일한 파라미터 수를 갖도록 균등하게 제거한 결과,
  - 정확도는 auto가 더 높음



Network Slimming



weight pruning process



weight pruning process - sort the original weight

- trained model의 기존 weight 오름차순 정렬

```
all_weights_sorted = {k: v for k, v in sorted(all_weights.items(), key=lambda item: abs(item[1]))}
```

- 기존 weight size : 2384000

```
len(all_weights_sorted)
```

```
2384000
```

weight pruning process - copy the sorted weight to new weight

- trained model의 weight를 복사하여 new weight 생성

```
new_model = load_model("/content/model.hdf5")
new_weights = trained_model.get_weights().copy()
```

- 기존 weight에 pruning 하려는 비율을 곱하여 대상 가중치 설정

```
prune_fraction = pruning_percent/100
number_of_weights_to_be_pruned = int(prune_fraction*total_no_weights)
weights_to_be_pruned = {k: all_weights_sorted[k] for k in list(all_weights_sorted)[ :  number_of_weights_to_be_pruned]}
```

```
596000
1192000
1430400
1668800
1907200
2145600
2264800
2312480
2360160
```

→ 25%, 50%, 60%, 70%, …, 97%, 99%
(pruning할 weight size)

```
(0, 560, 842)
(0, 25, 643)
(3, 275, 192)
(0, 656, 220)
(1, 213, 112)
(2, 323, 77)
(2, 16, 242)
(0, 579, 377)
```

→ pruning될 가중치에 대한 정보 (k)
→ (layer 번호, 해당 layer의 가중치 행렬의 행, 열)

weight pruning process - pruning

- new weight 에서 pruning 대상인 weight들은 0이 됨

```
for k, v in weights_to_be_pruned.items():
    new_weights[k[0]][k[1], k[2]] = 0
```

- reshape 하여 각 layer의 가중치 행렬 변경

```
for layer_no in range(total_no_layers - 1) :
    new_layer_weights = new_weights[layer_no].reshape(1, new_weights[layer_no].shape[0], new_weights[layer_no].shape[1])
    new_model.layers[layer_no].set_weights(new_layer_weights)
```

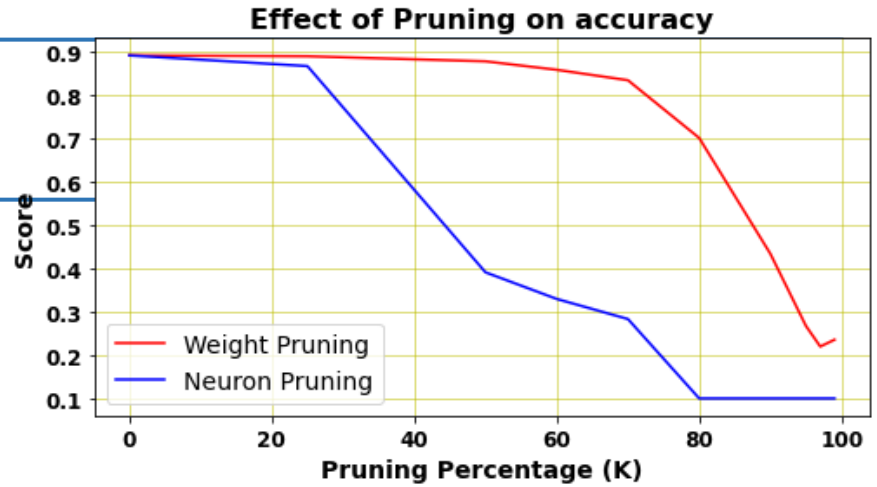
가중치 제거로 인해 sparse matrix가 되어 메모리 적게 필요

neuron pruning

- weight pruning 과 동일하게 중요하지 않은 순서대로 설정한 비율만큼 제거

result

```
Accuracy (original model)):
  88.7 %
```




```
Accuracy (weight pruning)
  89.0 %
Accuracy (weight pruning)
  88.81 %
Accuracy (weight pruning)
  87.64 %
Accuracy (weight pruning)
  85.69 %
Accuracy (weight pruning)
  83.28 %
Accuracy (weight pruning)
  69.99 %
Accuracy (weight pruning)
  43.29 %
Accuracy (weight pruning)
  26.66 %
Accuracy (weight pruning)
  21.95 %
Accuracy (weight pruning)
  23.52 %
```

weight pruning

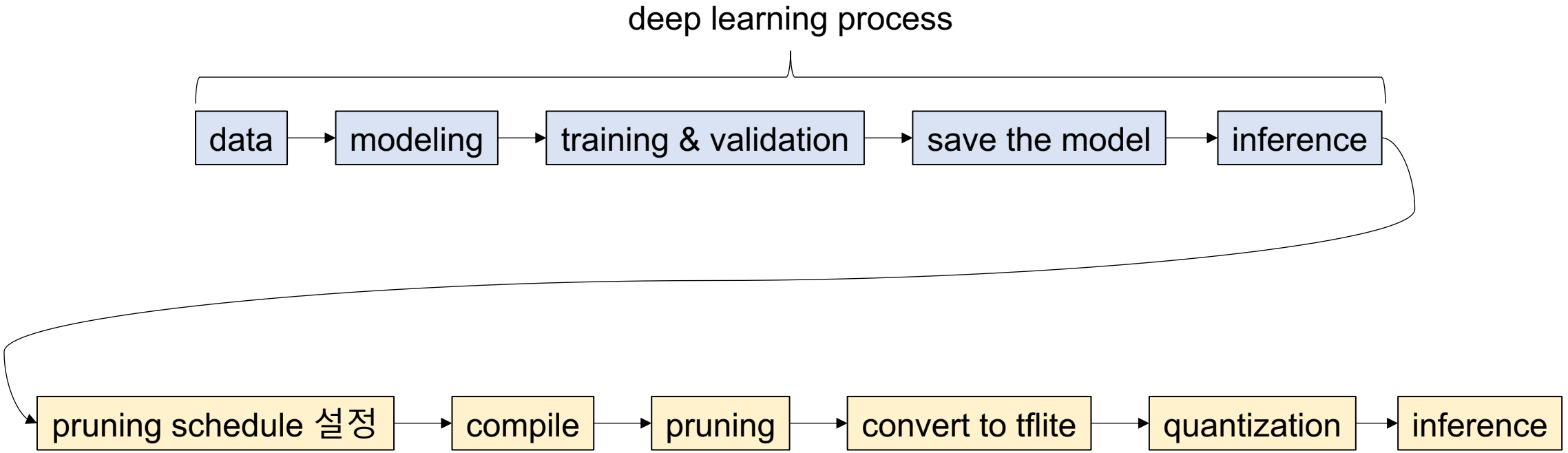
- 25%, 50% 줄였을 때가 기존 모델보다 정확도가 약간 높음
- 60% pruning 했을 때 87.64%로 큰 차이 없음
- 그 이상으로 가중치를 제거하는 경우 성능 저하

neuron pruning

- 25% 줄였을 때가 기존 모델보다 정확도가 약간 높음
- 그 이상으로 가중치를 제거하는 경우 성능 저하

```
Accuracy (neuron pruning)
  89.0 %
Accuracy (neuron pruning)
  86.54 %
Accuracy (neuron pruning)
  39.04 %
Accuracy (neuron pruning)
  32.91 %
Accuracy (neuron pruning)
  28.28 %
Accuracy (neuron pruning)
  10.0 %
Accuracy (neuron pruning)
  10.0 %
Accuracy (neuron pruning)
  10.0 %
Accuracy (neuron pruning)
  10.0 %
Accuracy (neuron pruning)
  10.0 %
```

pruning and quantization for tflite



pruning and quantization for tflite

- sparsity 설정 (PolynomialDecay)

```
pruning_params = {
      'pruning_schedule': tfmot.sparsity.keras.PolynomialDecay(initial_sparsity=0.00,
                                                               final_sparsity=0.50,
                                                               begin_step=0,
                                                               end_step=end_step)
}
```

학습하는 동안 initial_sparsity 부터 final_spartsity까지 제거

(0% ~ 50%까지 제거)

- compile and test (기존과 동일)

```
model_for_pruning.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model_for_pruning_accuracy = model_for_pruning.evaluate(
test_images, test_labels, verbose=0)
```

pruning and quantization for tflite

- convert to .tflite
  - pruned model을 그대로 tflite model로 변환 (기존과 동일)

- quantization
  - 신경망이 사용하는 부동 소수점을 줄임 (주로 float32 → uint8)
  - tflite 모델이 사용되는 경우 수행
  - Edge TPU는 8-bits 정수 연산에 최적화 되어 있음
  - 그러나 정밀도 감소로 인한 정확도 손실 위험 존재

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
quantized_and_pruned_tflite_model = converter.convert()
```

pruning and quantization for tflite – result

- original model, pruned model, pruned model (tflite) 용량 비교
  - weight pruning (filter x)
  - keras 모델, tflite 모델 모두 약 1.6배 감소
  - Edge TPU의 경우 8비트 연산 but pruning만 할 경우, 사이즈 감소 x

```
Size of original model: 78.12 kB
Size of pruned model: 48.28 kB
Size of pruned TFlite model: 47.44 kB
```

- original model, pruned model, pruned and quantized model 비교
  - pruning : 1.6배 감소
  - pruning + quantization : 5.7배 감소

```
Size of original model: 78.12 kB
Size of pruned TFlite model: 47.44 kB
Size of pruned and quantized TFlite model: 13.64 kB
```

*사용한 모델 구조 : Conv2D-Maxpooling2D-Flatten-Dense

pruning and quantization for tflite – result

- 용량은 5.7배 감소, 정확도는 거의 동일 (0.0003 증가)

```
Pruned and quantized TFLite test accuracy: 0.9806
Pruned model test accuracy: 0.9803000092506409
```

향후 계획

- GAN based PRNG의 tflite file size : 11kB
- pruning and quantization 적용
  - generator의 경우 Dense layer로만 구성

    → weight pruning (filter x)
  - weight quantization

- 난수성이 유지되는 선에서 용량 최소화, 속도 향상 계획

감 합 다

사 니