**Quantization Introduction**

실수형 변수(floating-point type)를 정수형 변수(integer or fixed point)로 변환하는 과정

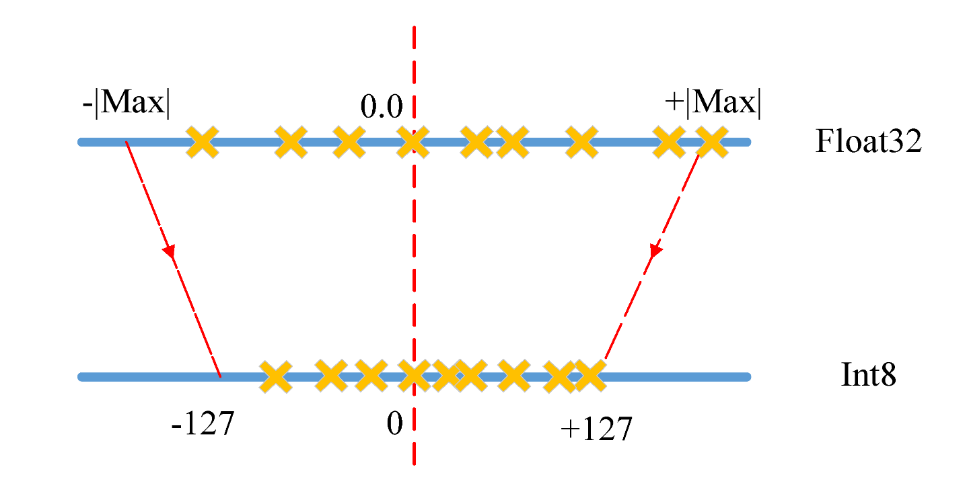
이 작업은 weight나 activation function의 값이 **어느 정도의 범위 안에 있다는 것을 가정**하여 이루어진다.

즉, 양자화는 **모델 파라미터의 32비트 부동수를 8비트 정수로 변환하는 기술**로

양자화를 통해 모델 크기와 메모리 공간을 원래 크기의 1/4로 줄일 수 있으며,

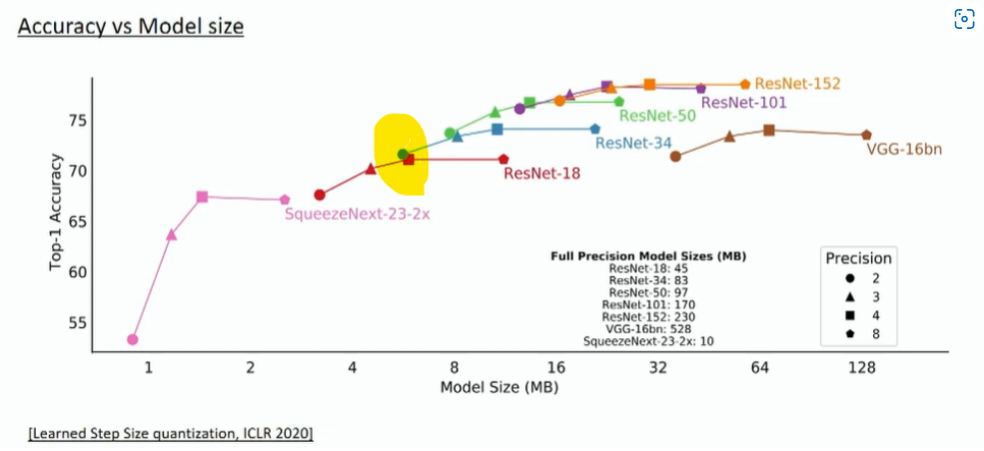
정확도는 거의 동일하게 유지하면서 추론을 약 2-4배 더 빠르게 할 수 있다.

+) **추론**(Inference)은 학습을 통해 만들어진 모델을 실제로 새로운 입력 데이터에 적용하여 결과를 내놓는 단계.



**따라서, Quantization의 주요 목적은 다음과 같다.**

1. 모델의 사이즈 축소  
2. 모델의 연산량 감소  
3. 효율적인 하드웨어 사용 (배포 중에 이점이 있음)



**Precision** = Quantization할 때 사용한 Bit

ResNet-34를 2 bit로 표현하였을 때의 Top-1 Accuracy가 ResNet-18을 4-bit로 표현하였을 때 보다 성능이 더 좋음.

이 때 모델 사이즈는 오히려 ResNet-34가 조금 더 가벼운 것도 확인가능!

이를 통해서, **양자화의 필요성**을 살펴볼 수 있음!

기억할 점은 FP32 → INT8로 변환 시 정보 손실이 발생하기 때문에

역으로 INT8 → FP32로 변환 시 잃어버린 정보로 인하여 그대로 변환되지 않는다.

이 때 발생하는 **Error를 Quantization Error**라고 하며 이 Error를 줄이는 것이 좋은 Quantization 알고리즘!

**양자화를 하는 방법**

1. **훈련 후 양자화** - 훈련 후 동적 양자화

- 훈련 후 정적 양자화

2. **양자화 인식 훈련**

**훈련 후 양자화**는 이미 Training된 모델에 Quantization을 적용하는 것으로,

Quantization이후에도 정확도의 하락 폭이 작지만, 소형 Model에서는 적합하지 않다.

**양자화 인식 훈련**은 학습을 통한 Quantization을 Simulate해보며 최적화하는 방식으로,

training과정 중에서 Quantize를 수행하는 것이며

Fake Quantization Node를 첨가하여 Quantize되었을 경우 어떻게 동작할지 Simulation한다.

그러나 사용하려는 모델에 이미 양자화된 버전이 있는 경우

위의 세 가지 워크플로우를 거치지 않고 직접 사용할 수 있음.

예를 들어, **토치비전 라이브러리**에는 이미

**모바일넷 v2, ResNet 18, ResNet 50, Inception v3, GoogleNet 등의 모델용 양자화된 버전이 포함**되어 있음.

**고정 소수점 & 부동 소수점**

컴퓨터는 0과 1로만 이루어져 있다.

따라서, 123이라는 수를 컴퓨터에서는 로 인식하게 된다.

근데, 1.23이라는 수를 똑같이 이진수로는 표현할 수 없다.

즉, 컴퓨터는 0과 1만 알고 있을 뿐, 다른 내용은 모르기 때문에 소수점을 알고 있지 않다.

따라서, **컴퓨터에서 소수점이 포함된 실수를 표현**하기 위해서

**고정 소수점(Fixed Point)**

**부동 소수점(Floating Point)**를 사용하게 되었다.

**고정 소수점**

말 그대로, 소수점이 고정된 것!

이해를 돕기 위해, 고정 소수점이 32bit의 크기로 되어있다고 가정을 하고 2진수로 표현하면 다음과 같다.

검정색 비트[1 Bit] = 부호 결정 비트 = 0이면 양수, 1이면 음수

빨간색 비트[15 Bits] = 정수를 표현

파란색 비트[16 Bits] = 소수점 이하 자릿수 표현

여기서 9.6875를 고정 소수점으로 표현해보면 다음과 같다.

여기서는, 정수부와 소수부의 값이 충분히 작아서 32bit의 고정 소수점으로 표현이 가능했지만

만약, 소수점이 무수히 많다면, 정수가 매우 크다면 위의 범위 내에서는 다 표현이 안될 수 있다.

즉, **고정 소수점의 한계는**

**정수 부분과 소수 부분의 자릿수가 작고, 고정되어 있어서 표현할 수 있는 범위가 제한적이다.**

**부동 소수점**

부동, 영어로 float라는 뜻은 [유동적인]이라는 의미를 가진다.

**이는 소수점이 고정되어 있지 않고 좌우로 움직일 수 있음을 의미한다.**

그래서 고정 소수점과는 달리 소수점을 자유로이 움직일 수 있어서

표현할 수 있는 수의 범위가 매우 넓다는 장점이 있다.

부동 소수를 표현하기 위한 수식은 다음과 같다.

검정색 비트[1 Bit] = 부호 결정 비트 = 0이면 양수, 1이면 음수

빨간색 비트[8 Bits] = 지수를 표현, 양수와 음수 모두 표현해야 하므로

값의 범위가 **[-127~128]**이 되도록 **‘-127’**을 추가!

파란색 비트[23 Bits] = **가수**를 표현,

**실질적인 숫자**로 만약 123.456을 표현한다고 치면 가수는 123456이다.

위 수식을 이해하기 위해 10진수의 표준 표기법부터 다시 볼 필요가 있다.

의 표준 표기법은 이다.

이는 부동 소수점을 이용해서 2진수로 나타낼 때도 마찬가지다.

9.6875를 부동 소수점으로 표현해보자.

우선, 9.6875를 2진수로 표현해보면, 정수부와 소수부를 나누어서 이 된다.

이를 2진수의 표준 표기법으로 바꿔보면 이다.

최종적으로, 이를 부동 소수점에 적용해보자.

**가수부**  = 0011011

**지수부**  = =

즉, **이렇게 해서 표현할 수 있는 수의 범위가 보다 넓어졌다.**

**Dynamic Quantization**

**동적 양자화**

**특징**

1. 가장 간단한 양자화 기법
2. **가중치(Weights)에 대해서만 양자화 진행하며, Training 이후에 진행**
3. 활성화(Activations)는 추론(Inference)할 때 동적으로 양자화 진행
4. **보통 CPU 환경에서만 Inference 가능**
5. 모델의 weights를 메모리에 loading하는 것이 execution time에 큰 영향을 미치는 BERT와 같은 모델에 적합

**With PyTorch 1.7.0, we could do dynamic quantization using x86-64 and aarch64 CPUs. However, NVIDIA GPUs have not been supported for PyTorch dynamic quantization yet.**

텍스트이(가) 표시된 사진

자동 생성된 설명  
**Dynamic Quantization Diagram**

**가중치에 대해서만 32bit에서 8bit로 양자화 된 것!**

**Static Quantization**

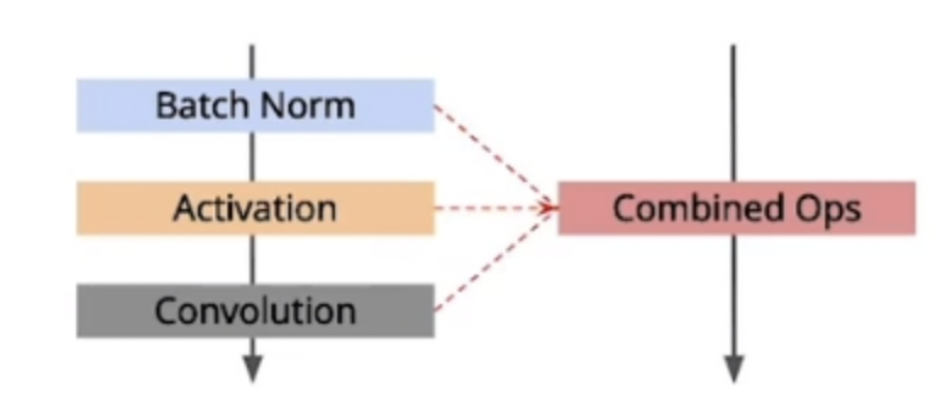
**정적 양자화**

**특징**

1. **모델의 가중치(Weights)와 활성화(Activations) 모두 사전에 양자화를 진행**
2. Activation, Convolution 등 각각의 기능을 수행하는 Layer들을 Fusion(합치기)하여

Layer간 데이터 이동으로 발생하는 Context Switching Overhead를 줄일 수 있다.

또한, Sequential하게 처리되던 연산을 병렬로 처리할 수 있다는 장점도 있다.

****

fusion = 각각의 기능을 수행하는 layer를 하나로 합치는 것

**[Conv, Relu], [Conv, BatchNorm], [Conv, BatchNorm, Relu], [Linear, Relu]** 등과 같은 **fusion**이 있음.

1. 정확도 손실을 최소화하기 위해 calibration으로 미세조정
2. TensorFlow Lite는 CPU, GPU 환경에서 Inference(추론)가 가능하지만, PyTorch는 CPU만 가능

**텍스트이(가) 표시된 사진

자동 생성된 설명  
Static Quantization Diagram**

**가중치와 활성함수 모두 32bit에서 8bit로 양자화 된 것!**

**Quantization Aware Training**

**양자화 인식 훈련**

난이도 자체는 **훈련 후 양자화** Method에 접근하는 것이 더 쉽지만,

모델의 성능과 정확성을 생각해보면, **양자화 인식 훈련**을 사용하는 것이 더 좋다.

양자화 인식 훈련은 주로 CNN에 이용

모델을 양자화 인식 트레이닝에 사용할 수 있도록 하려면

모델 정의의 \_\_init\_ 메서드에서 QuantStub 및 DeQuantStub을 정의하여

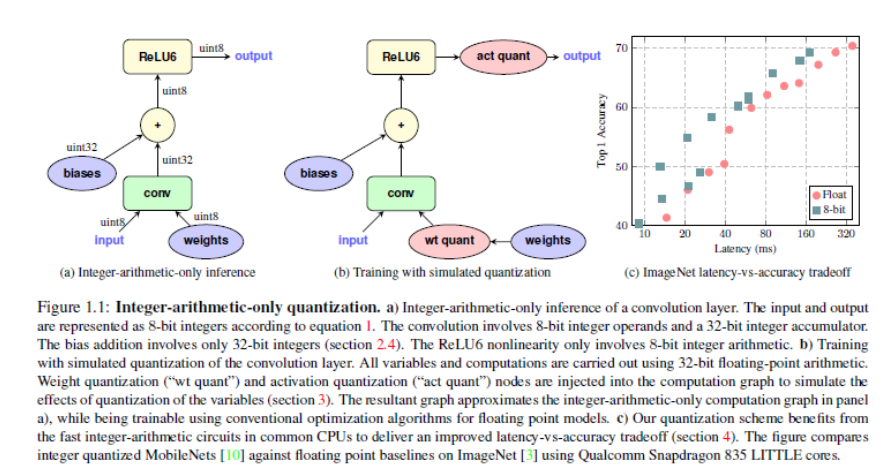
텐서를 부동 소수점에서 양자화된 유형으로 변환하고 그 반대의 경우도 마찬가지로 진행한다

본론으로 돌아와서,

**양자화 인식 훈련(QAT)는 모델의 가중치(Weights)와 활성화(Activations)를 학습하면서 양자화**를 하는 것이다.

**특징**

1. **학습은 CPU, GPU 환경에서 모두 가능하지만, 추론은 CPU에서만 가능하다.**
2. Dynamic Quantization, Static Quantization보다 높은 Accuracy를 확보할 수 있다.
3. Quantization Aware Training중에 활성함수(Activation)의 실제 최대 및 최소 출력 범위 확인도 진행한다.



위 그림에서 볼 수 있듯이, **Quantization Aware Training**은 Training과정에서 Quantize를 수행하는 것으로,

**Fake Node**를 첨가하여, Quantize되었을 시 어떻게 수행이 될지 시뮬레이션을 돌려보는 방식이다.

여기서, 원래 Training Loss가 수행되는 것이 약간 틀어져서 Quantize된 것까지 고려해

양자화 모델의 정확도 하락을 최소화하는 방식이다.

 QAT를 사용하면 모든 가중치와 활성화는 훈련의 전진 및 후진 통과 동안 "가짜 양자화"된다. 즉, 부동소수점 값은 int8 값을 모방하도록 반올림되지만, 모든 계산은 여전히 부동소수점 숫자로 수행된다. 따라서 훈련 중 모든 가중치 조정은 모델이 궁극적으로 양자화될 것이라는 사실을 "인식"하면서 이루어진다. 따라서 양자화 후 이 방법은 일반적으로 다른 두 가지 방법보다 높은 정확도를 산출한다.

**Quantization Aware Training은 dynamic, static quantization으로 성능이 나오지 않는 CNN 모델에서 활용**

텍스트이(가) 표시된 사진

자동 생성된 설명  
**Quantization Aware Training Diagram**

테이블이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

**PyTorch Version 1.3 부터 양자화가 가능!**

텍스트이(가) 표시된 사진

자동 생성된 설명 **Quantization Tools in PyTorch**

**Quantization Example Codes**

텍스트이(가) 표시된 사진

자동 생성된 설명

**텍스트이(가) 표시된 사진

자동 생성된 설명**

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

**fbgemm:** More about [here](https://blog.ggaman.com/1028)

* Quantization을 지원하는 두개의 backend가 있는데, x86용으로는 FBGEMM, arm용으로는 QNPACK이 있다고 한다.
* FBGEMM은 페이스북에서 만든 행렬처리용 라이브러리다. 그리고 Caffe2 와 PyTorch의 양자화 연산에 사용된다고 적혀 있다.

**Quantization References**

**[PyTorch Documentation & Guide]**

1. ***Quantization Recipe*** **– Click** [**Here**](https://pytorch.org/tutorials/recipes/quantization.html)
2. ***Quantization Documentation*** **– Click** [**Here**](https://pytorch.org/docs/stable/quantization.html)

**[TensorFlow Documentation & Guide]**

1. ***양자화 인식 훈련* – Click** [**Here**](https://www.tensorflow.org/model_optimization/guide/quantization/training)
2. ***양자화 인식 훈련 종합 가이드* – Click** [**Here**](https://www.tensorflow.org/model_optimization/guide/quantization/training_comprehensive_guide)
3. ***Keras 예제의 양자화 인식 훈련* – Click** [**Here**](https://www.tensorflow.org/model_optimization/guide/quantization/training_example)

**[참고자료]**

1. **Deep Learning Quantization Summary – Click** [**Here**](https://velog.io/@jooh95/%EB%94%A5%EB%9F%AC%EB%8B%9D-Quantization%EC%96%91%EC%9E%90%ED%99%94-%EC%A0%95%EB%A6%AC)
2. **Deep Learning Theory Quantization – Click** [**Here**](https://velog.io/@ganta/Quantization)
3. **Introduction to Quantization on PyTorch – Click** [**Here**](https://pytorch.org/blog/introduction-to-quantization-on-pytorch/)
4. **모델 압축 기법 – Click** [**Here**](http://dankernel.sciomagelab.com/2021/05/12/distiller-%EB%AA%A8%EB%8D%B8-%EC%95%95%EC%B6%95-%EA%B8%B0%EB%B2%95-3-quantization-%EC%96%91%EC%9E%90%ED%99%94/)

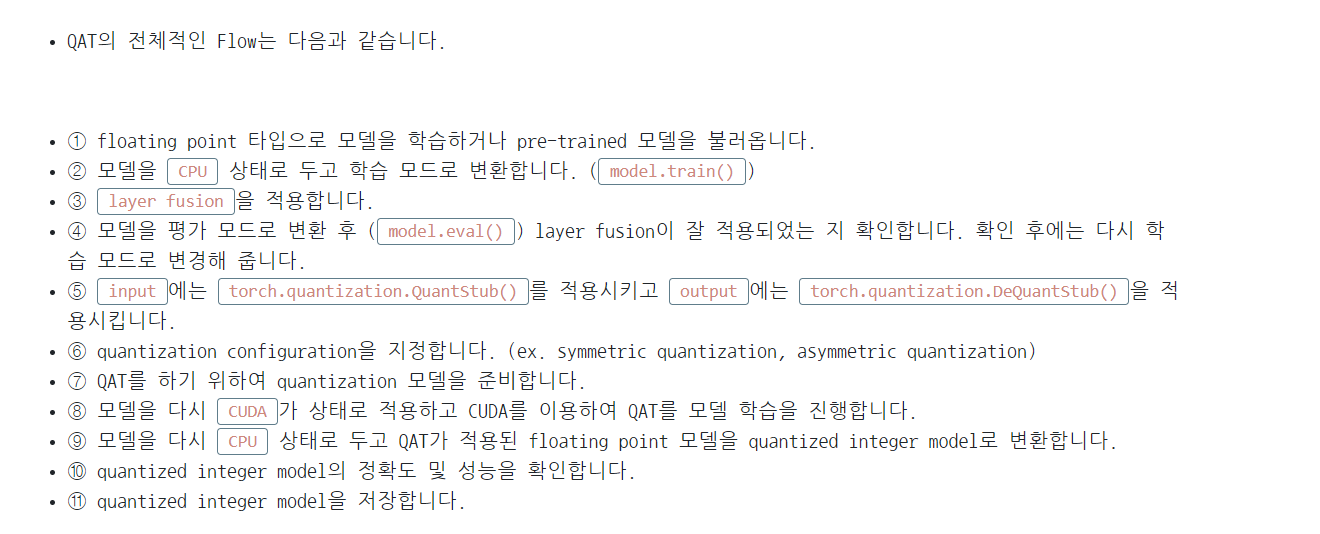
**(+)**

**To Apply Static Quantization**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Tutorial**

****

**To Presentation**

**Quantization 동기**

while models are becoming more efficient, high accuracy still implies high complexity!

발달되고 있는 Model들을 Embedded Board나 Mobile 환경에서도 손쉽게 사용하고 싶은데,

이를 위해서는 모델의 용량뿐만 아니라 전력 사용량도 줄이는 등 최적화 작업을 수행해야 합니다.

즉, 더 나은 성능을 얻되 에너지 사용은 줄이는 방안을 연구한 것이 양자화입니다.

**Quantization의 핵심 개념**

32bit의 float type 이지만,

이를 8 bit의 int type으로 바꿔서

메모리를 1/4로 줄이고, 연산 속도는 2배~4배 빠르게 구현하는 것.

**Quantization 종류**

훈련 후 양자화는

먼저 훈련을 다 끝내놓고 Quantization Error를 Minimum하게 하는 방향으로 Quantization을 진행

좀더 편리하고 쉽지만, Weight가 완벽히 최적화되기는 힘든 구조.

이보다 좀 더 정확한게 Quantization Aware Training인데,

Training하면서 즉, 최적의 Weight를 결정하면서 동시에 Quantization을 진행하는 것.

좀 복잡하지만 Weight를 보다 최적화시키는 구조.

**테이블이(가) 표시된 사진

자동 생성된 설명**

**PTQ - Dynamic Quantization**

텍스트이(가) 표시된 사진

자동 생성된 설명

동적 양자화에 대한 것인데, LSTM이나 Multi Layer Perceptron에 적합하며

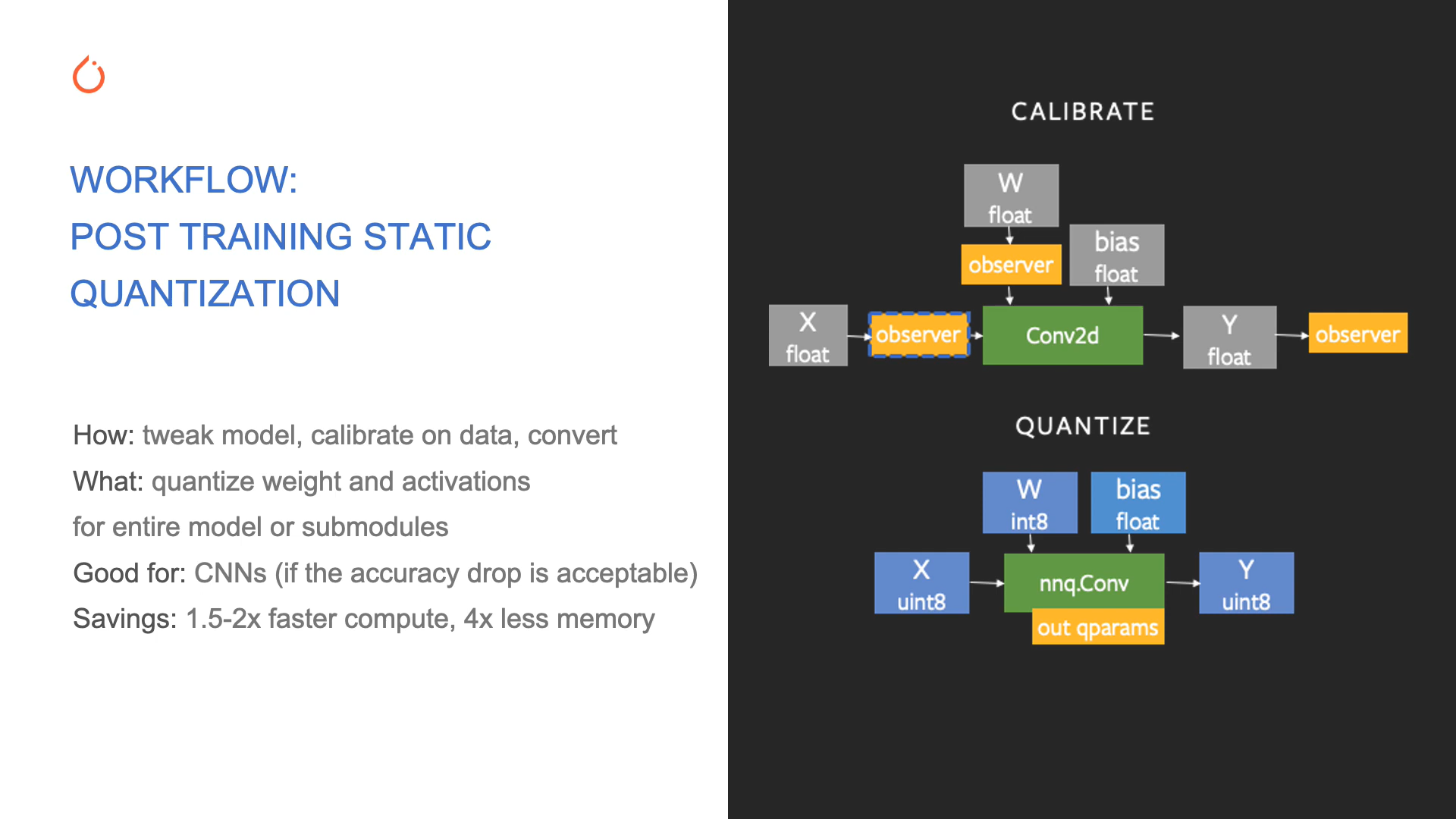
우측에 있는 간단한 코드로 실행이 가능합니다.

우선 모델을 로드하는 것으로 시작하며,

Quantization module을 지정하고 저장하면 됩니다.

매우 간단한 프로세스입니다.

**PTQ - Static Quantization**



CNN에 적합한 것이 특징

약간 조정한 일부 데이터를 기반으로 하여 모델을 보정한 다음

보정된 모델을 양자화된 모델로 바꾸는 과정입니다.

가중치와 활성화를 모두 양자화하는 것입니다.

다음 페이지에서는 Step by Step Approach를 제시하겠습니다.

**PTQ = Post Training Quantization**

**PTQ – Static Quantization Step by Step Approach**

텍스트이(가) 표시된 사진

자동 생성된 설명

Static Quantization을 적용하는 과정입니다.

우선, 모델을 로드하고 레어들을 fuse해줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

앞에서 모델을 약간 수정했다면, 여기서는 양자화 구성을 설정합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

모델 수정과 양자화에 대한 설정을 마쳤다면

Convert 명령어를 통해 실제적으로 양자화를 진행해주도록 합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Quantized Model Save!

**그러나 때때로,**

**CNN은 훈련 후 정적 양자화 (Static Quantization After Training)에서 좋은 성능을 보이지 않을 수 있다.**

**이때는, QAT을 적용하면 된다.**

**입력 및 가중치가 양자화 되었을 때,**

**올바른 답을 제공하는 방법을 학습하는 방식으로 네트워크를 조정하므로**

**기본적으로 Fine Tuning을 필요로 합니다.**

**QAT – Quantization Aware Training**

텍스트이(가) 표시된 사진

자동 생성된 설명Calibration 대신 Fine Tuning을 수행한다는 점을 제외하면 Static Quantization과 매우 유사하다.

우선 Model을 불러오고,

양자화와 학습을 동시에 진행해야 하기 때문에 불러온 모델을 train모드로 설정합니다.

그리고, 유사한 성격을 가지는 Layer끼리 Fusion하는 과정을 진행해줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Train Data를 사용하여 모델을 훈련하는데,

Fake Quantization Node로부터 수집한 데이터로 정보를 활용하여 양자화를 최종적으로 진행

이후의 진행은 Static Quantization과 동일!

테이블이(가) 표시된 사진

자동 생성된 설명

**Quantization 결과 분석 표는 이와 같이 만들기!**

Top5 Accuracy / Top1 Accuracy 둘 다 사용하면 좋다.

테이블이(가) 표시된 사진

자동 생성된 설명

**Quantization Aware training성능이 제일 좋구만!**

텍스트이(가) 표시된 사진

자동 생성된 설명

**(+) Graph Mode Quantization – prototype version of PyTorch Quantization**