

TinyML 도서 스터디 1주차



Chapter 1 ~ Chapter 3

이름

신정아

E-mail

jeongah.arie@gmail.com

오늘의 이야기는 ...



Chapter 1 ~ Chapter 3

Chapter 1. Introduction

Chapter 2. Getting Started

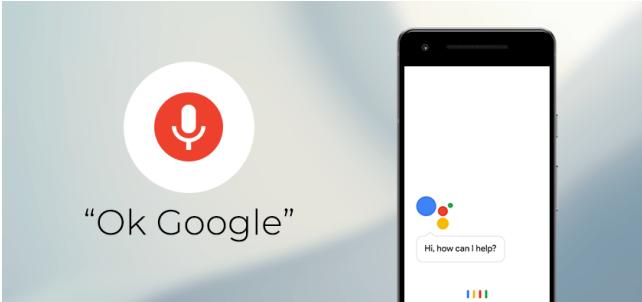
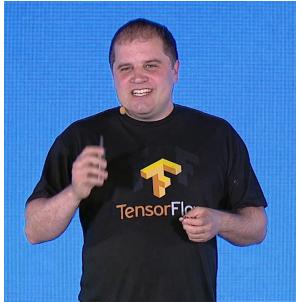
Chapter 3. Getting up to Speed on Machine Learning

Chapter 1. Introduction

Introduction Summary

1. Introduction

TinyML 저자이신 Peter 아조씨



"OK Google Team" Running on Mobile DSP

Main CPU was powered off Tens kilo bytes of RAM and flash memory, few milliwatts(mW) of power

"Pixel's Music IQ" On-device Music Recognition

<https://ai.googleblog.com/2018/09/googles-next-generation-music.html>

"Qualcomm's Glance Camera Module" in SoC

Embedded AI for face detection, face recognition, object tracking and people counting

<https://www.qualcomm.com/products/qcs605>

Summary

TinyML (On-device ML)에는



Raw Sensor Data soon Actionable Information locally (On-device)

Low cost of hardware, Low energy 라는 혁신적인 장점들이 있지만

Still have tough resource constraints 실제로 구현하기에 꽤나 힘들고야 오호호~🤣

Chapter 2. Getting Started

Pre-requisites

2. Getting Started

Knowledge

TinyML (On-device ML)을 공부하기 위해서는

Machine Learning

Embedded Software Development 관련 지식 + 약간의 CLI 활용 능력이 필요해!

Hardware

Laptop and Desktop PC with USB Port <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/micro>

GPU는 Google Colab 사용 그런데 실습 해본 결과 아무리 경량화 모델이라도 Colab으로 하면 느려서 개인 GPU 있으신 분들은 활용하시는 걸 추천!

SparkFun Edge, Arduino Nano 33 BLE Sense, STM32F46G Discovery Kit ... 우리 스터디는 Mobile AP에 집중!

Software

아무거나 No matters Microsoft VS Code가 OS 호환이 범용적이어서 추천해!

CLI Interface Terminal (MacOS), Command Prompt (Windows), 기타 임베디드 장치에서는 Python이 설치되어 있어 BuildScript 돌릴 수 있으면 무방해~

Chapter 3. Getting up to Speed on Machine Learning

Deep Learning Workflow

Chapter 3. Getting up to Speed on Machine Learning



Decide a goal

Whether the factory machine works well Classification Task with known class ("Normal" : prob , "Abnormal" : 1-prob)

Collect a Dataset

Various available data in Factory (operating temperature ~ the type of food served in cafeteria) 그렇다면 어떤 데이터가 우리의 Task에서 필요할까?

Select data

Relevant to solving problem

모델은 관련 있지 않은 데이터와 노이즈를 무시 할 줄 알아야 한다.

거짓된 연관성에 대해 취약할 수 있기 때문이다. Cafeteria 메뉴는 Factory Machine 과 무관하지만 Pizza가 나오는 날 마다 기계가 고장날 수도…

Combine with domain experts when deciding whether to include data

Collecting data

얼마나 많은 데이터가 필요할까? 변수간의 관계의 복잡성, 노이즈의 양, Class 구분의 용이함 등 많은 요소가 영향을 끼친다.

The more data, the better ! 발생할 수 있는 조건과 상황에 full-range로 대응할 수 있는 데이터를 모아야한다.

예를 들어, 따듯한 날에 Factory Machine 온도가 올라간다면 여름~겨울에 걸친 온도 데이터를 가지고 있어야한다. 수집된 time series 데이터는 우리 모델에 적합하도록 형식을 바꿔준다.

Deep Learning Workflow

Chapter 3. Getting up to Speed on Machine Learning



Collect a Dataset

Labeling data

우리는 기계의 동작이 "Normal" 한지 "Abnormal" 한지를 결정하여 모델에게 학습시켜야 한다. 관련 데이터를 Class로 구분하는 것을 Labeling이라고 한다.

전체 time-series 데이터에서 기계가 정상 동작 할 때, 오동작할 때의 구간을 기록하고, 기계가 고장 나기 직전의 time 구간을 Abnormal Operation이라고 가정한다.

데이터를 기반하여 이 가정이 맞을지도 실험을 거쳐보는 것이 좋다!

Design a Model Architecture

The most effective architecture varies depending on the type of data

The constraints of the device that targets to run a model (작은 용량의 메모리와 느린 속도의 프로세서를 가진 임베디드 장치들의 환경을 고려 +a 하드웨어 가속화)

딥러닝 모델은 "tensor"라는 형태의 input과 output을 가진다

Vector, Matrix, Higher-dimensional tensors, Scala

Deep Learning Workflow

Chapter 3. Getting up to Speed on Machine Learning



Collect a Dataset

Labeling data

우리는 기계의 동작이 "Normal" 한지 "Abnormal" 한지를 결정하여 모델에게 학습시켜야 한다. 관련 데이터를 Class로 구분하는 것을 Labeling이라고 한다.

전체 time-series 데이터에서 기계가 정상 동작 할 때, 오동작할 때의 구간을 기록하고, 기계가 고장 나기 직전의 time 구간을 Abnormal Operation이라고 가정한다.

데이터를 기반하여 이 가정이 맞을지도 실험을 거쳐보는 것이 좋다!

Design a Model Architecture

The most effective architecture varies depending on the type of data

The constraints of the device that targets to run a model (작은 용량의 메모리와 느린 속도의 프로세서를 가진 임베디드 장치들의 환경을 고려 +a 하드웨어 가속화)

딥러닝 모델은 "tensor"라는 형태의 input과 output을 가진다

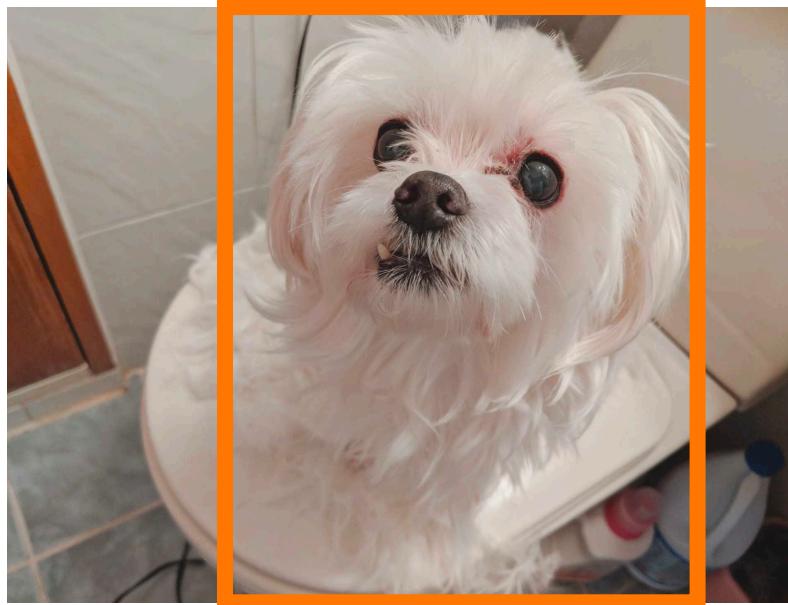
Vector, Matrix, Higher-dimensional tensors, Scala

Application: Design a Model Architecture

Chapter 3. Getting up to Speed on Machine Learning



Object Detection + Landmark Detection



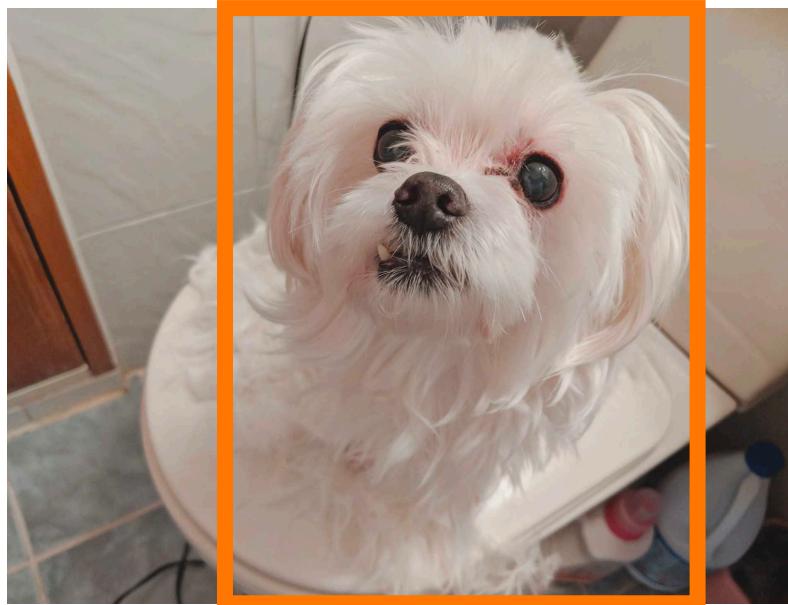
주어진 이미지에서 강아지/고양이가 위치한 영역을 검출(Object Detection),
9개의 Facial Feature 분석(Landmark Detection)

Application: Design a Model Architecture

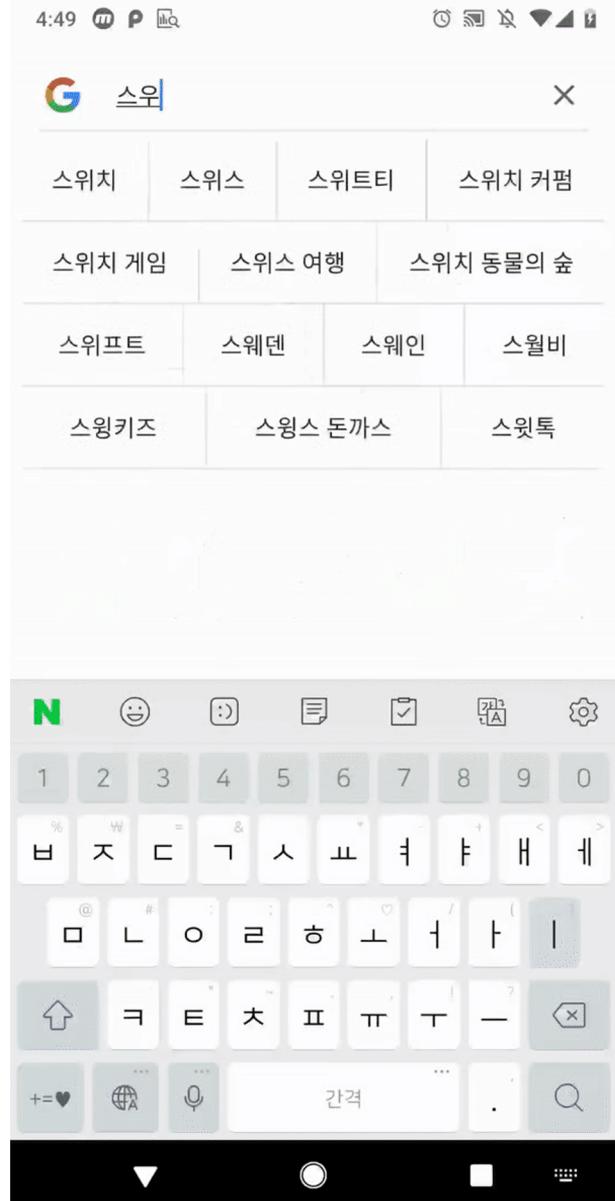
Chapter 3. Getting up to Speed on Machine Learning



Object Detection + Landmark Detection



주어진 이미지에서 강아지/고양이가 위치한 영역을 검출(Object Detection),
9개의 Facial Feature 분석(Landmark Detection)



[DEMO] 고양이용 Landmark Detection 모바일에의 적용

Application: Design a Model Architecture

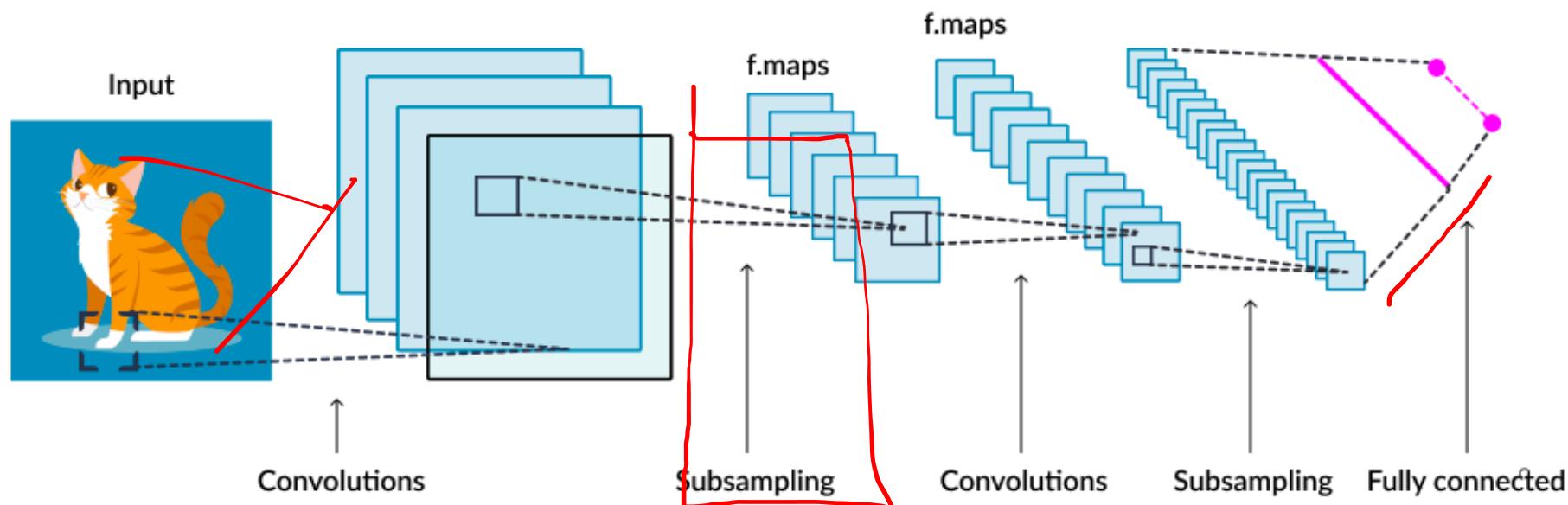


Object Detection + Landmark Detection - ~~MobileNet V2 기반~~

```
mobilenetv2_model = mobilenetv2.MobileNetV2(input_shape=(img_size, img_size, 3), alpha=1.0,  
depth_multiplier=1, include_top=False, weights='imagenet', input_tensor=inputs, pooling='max')
```

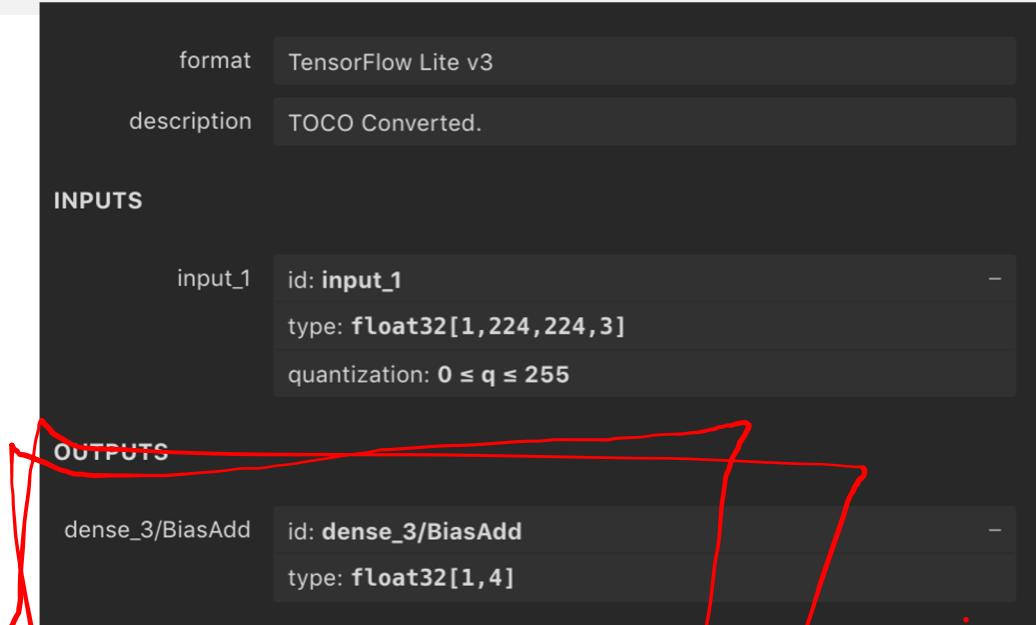
* MobileNet V2

이전 버전의 MobileNet의 Standard Convolution이 무겁기 때문에 이를 DSC(Depthwise Seperable Convolution)

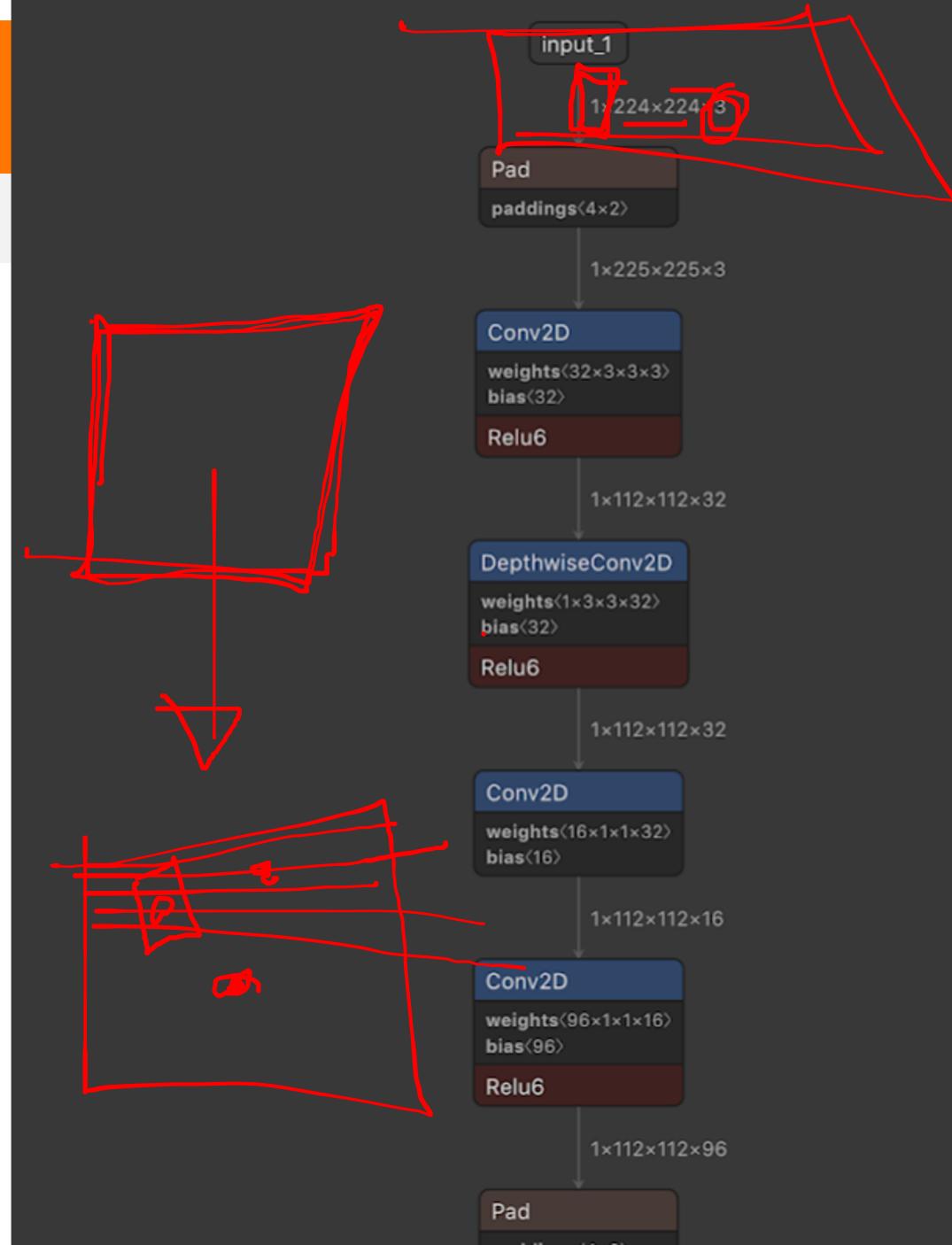


Application: Design a Model Architecture

Chapter 3. Getting up to Speed on Machine Learning



** Input Channel은 작게, 후처리가 적도록!



Hardware Acceleration (CPU, GPU, NNAPI)

Chapter 3. Getting up to Speed on Machine Learning

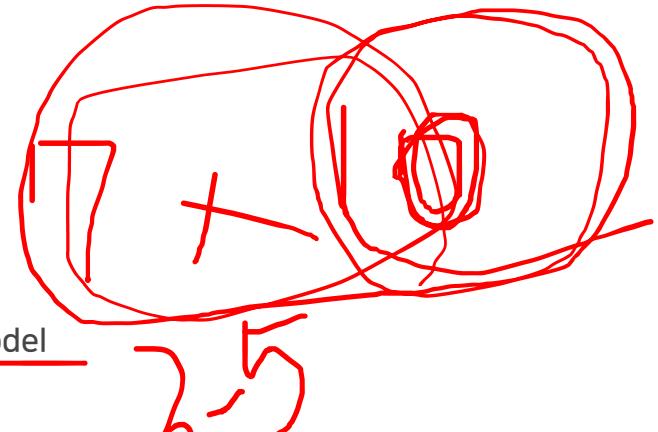
For general perspective,

GPU(Graphics Processing Unit) : 많은 처리를 요하지만 빨리 처리해야하는 모델 ex) 3D handling 하는 model

CPU(Central Processing Unit) 간단한 ML 모델에의 최선의 옵션

NPU (Neural Processing Unit), DSP(Digital Signal Processor) 적은 처리량을 요하면서도 복잡한 모델, 엄청나게 빨리 처리해야하는 모델

(Device Compatibility 문제로 아직 상용화 어려움)



Q.

GPU, NNAPI 를 사용해서
하드웨어를 가속하면
무조건 속도가 빨라지나요?

Q.

GPU, NNAPI 를 사용해서
하드웨어를 가속하면
무조건 속도가 빨라지나요?



아니요!

✓ GPU, NNAPI, DSP 가속을 할 때 기억해주세요!

Supported Ops

TensorFlow Lite on GPU supports the following ops in 16-bit and 32-bit float precision:

- ADD v1
- AVERAGE_POOL_2D v1
- CONCATENATION v1
- CONV_2D v1
- DEPTHWISE_CONV_2D v1-2
- FULLY_CONNECTED v1
- LOGISTIC v1
- MAX_POOL_2D v1
- MUL v1
- PAD v1
- PRELU v1
- RELU v1
- RELU6 v1
- RESHAPE v1

- ✓ 특정 하드웨어를 가속할 수 있는 Tensorflow Operator는 한정되어 있음.
- ✓ Delegate에서 지원하지 않는 Operator가 사용된 부분에 한해서 CPU와의 병행적인 사용이 가능하지만, CPU만 사용할 때 보다 되려 성능이 하락될 수 있음.
- ✓ Mobile-BERT: GPU Delegation을 사용하는 것 보다 CPU 환경에서 Multi-threading 하는게 더 효과적이었음.

Hardware Acceleration Result

Chapter 3. Getting up to Speed on Machine Learning



나는 놀고 싶어
End2End 18 ~ 30 fps (Mobile GPU 가속시)



Tensorflow Lite



ML Kit

- GPU Delegation API의 적용
- Tensorflow Model Optimization Toolkit 활용

Deep Learning Workflow

Chapter 3. Getting up to Speed on Machine Learning



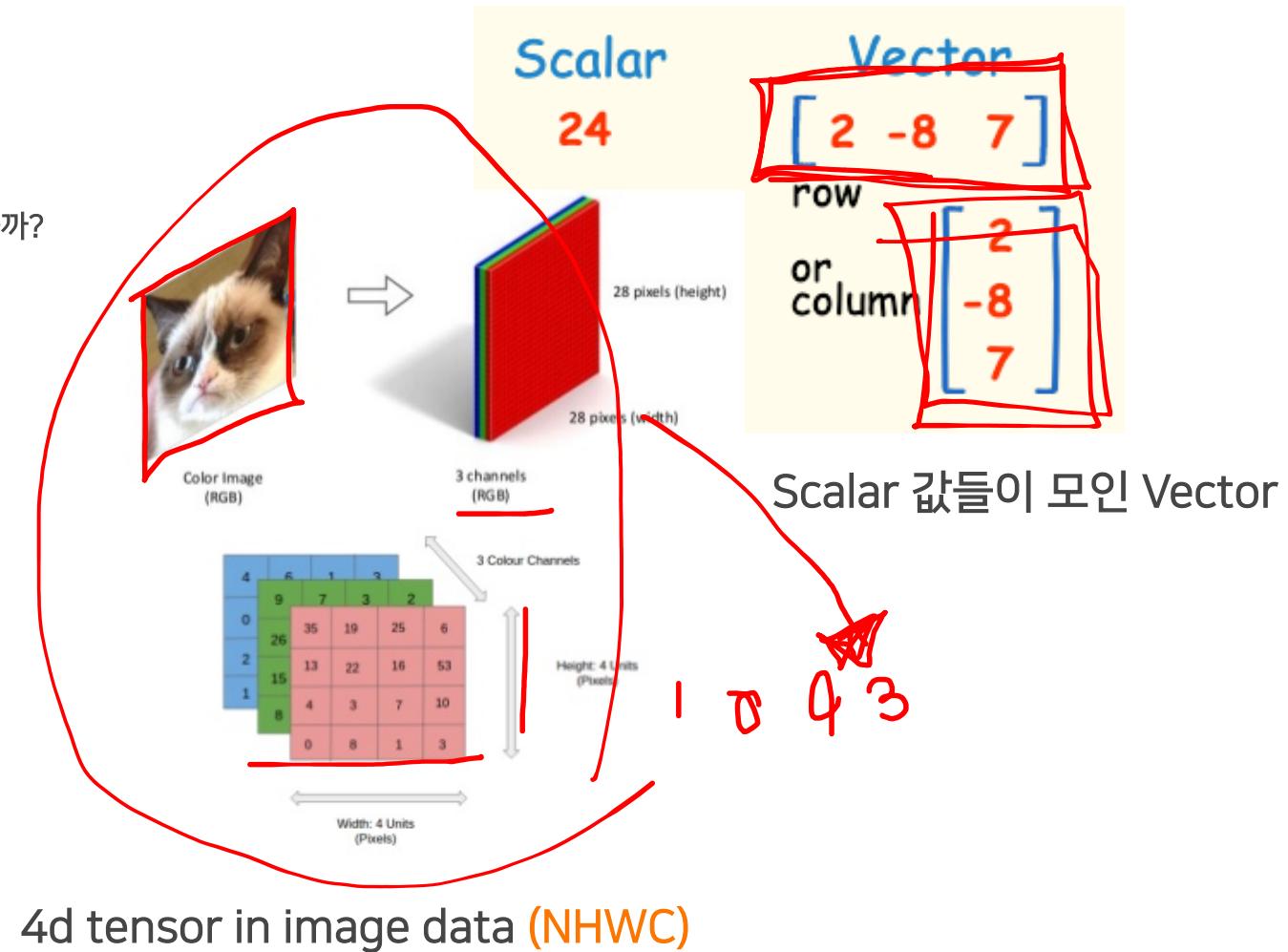
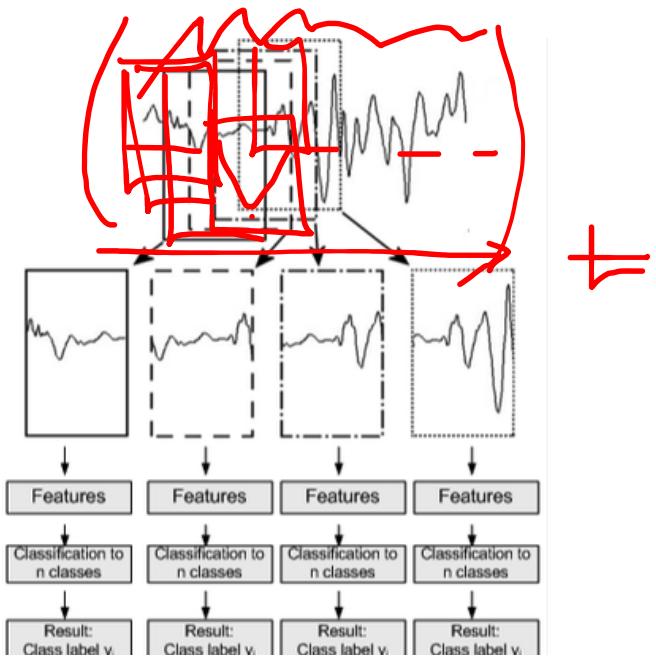
Design a Model Architecture

Generating Features from data

우리가 가진 공장의 time-series 데이터를 어떻게 모델이 이해할 수 있는 형태로 만들까?

Feature : Particular type of information on which a model is trained

Windowing



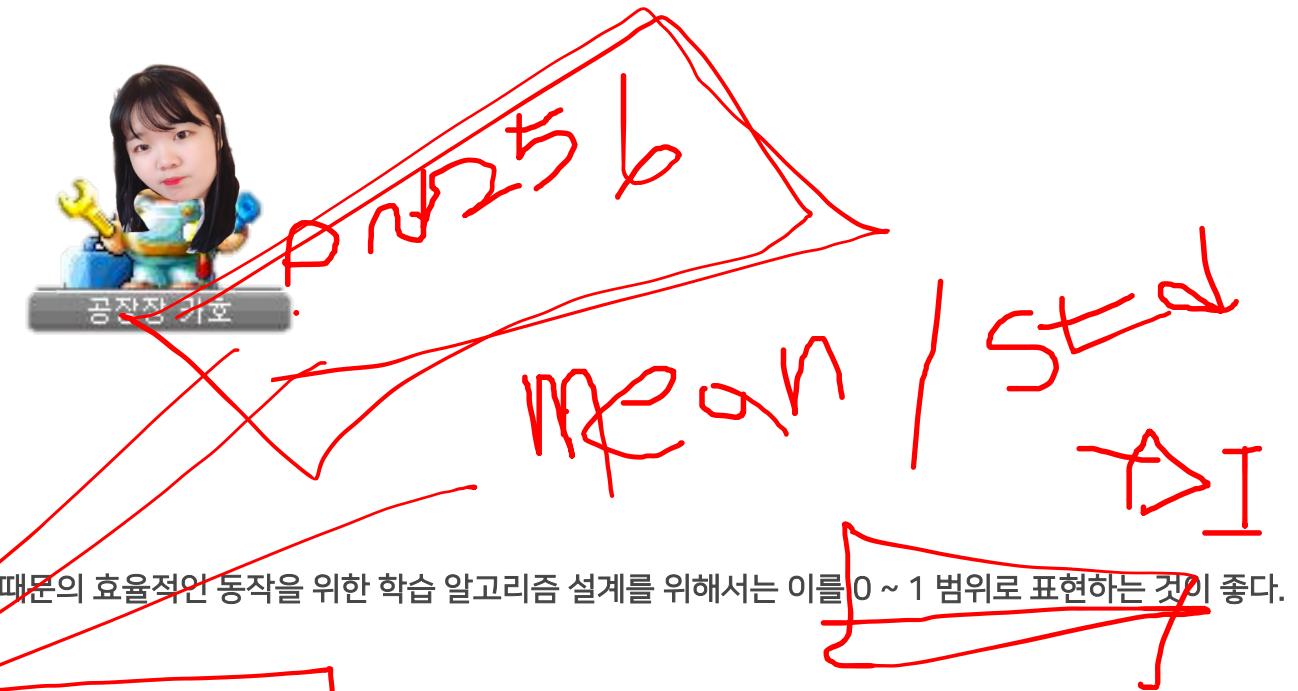
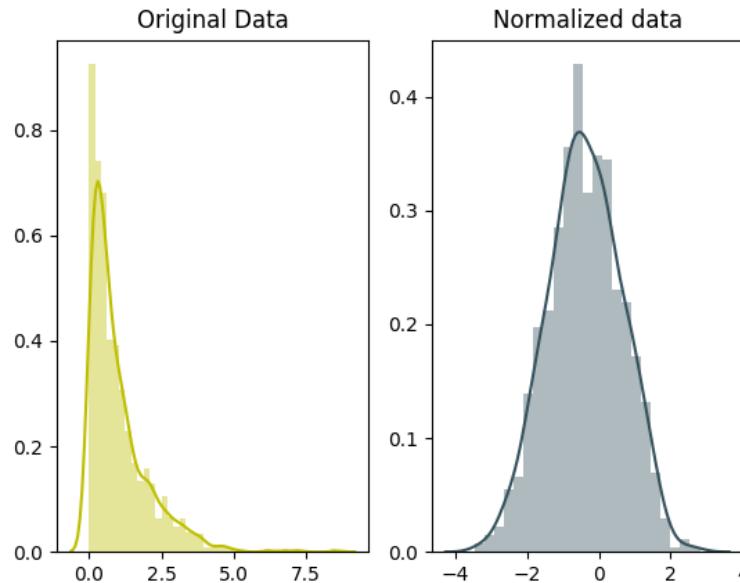
Deep Learning Workflow

Chapter 3. Getting up to Speed on Machine Learning

Design a Model Architecture

Normalization

우리의 모델이 input으로 가지는 Tensor는 floating-points / floats 으로 이루어져 있기 때문에 효율적인 동작을 위한 학습 알고리즘 설계를 위해서는 이를 0 ~ 1 범위로 표현하는 것이 좋다.



Deep Learning Workflow

Chapter 3. Getting up to Speed on Machine Learning



Train the Model

모델의 **weight**는 (보다 다양한 weight initialization 기법들이 있겠지만) 보통 random value로 초기화 되고 **bias**의 경우 0으로 초기화 된다.

모델 학습이 시작되면 “batch” 단위의 데이터를 모델에 넣게 되는데, 여기서 앞서 라벨링 한 정보들을 모델의 output과 비교하게 된다. Factory Machine (“Normal”, “Abnormal”) “back-propagation” 과정을 통해 weight와 bias를 조절하여 desired output과 모델의 output이 가까워질 수 있도록 한다.

원하는 결과가 나올 때 까지 (we decide to stop) “epoch” 단위로 학습을 실행한다. 보통 모델의 성능이 증가하는 것이 멈췄을 때 학습을 종료한다.

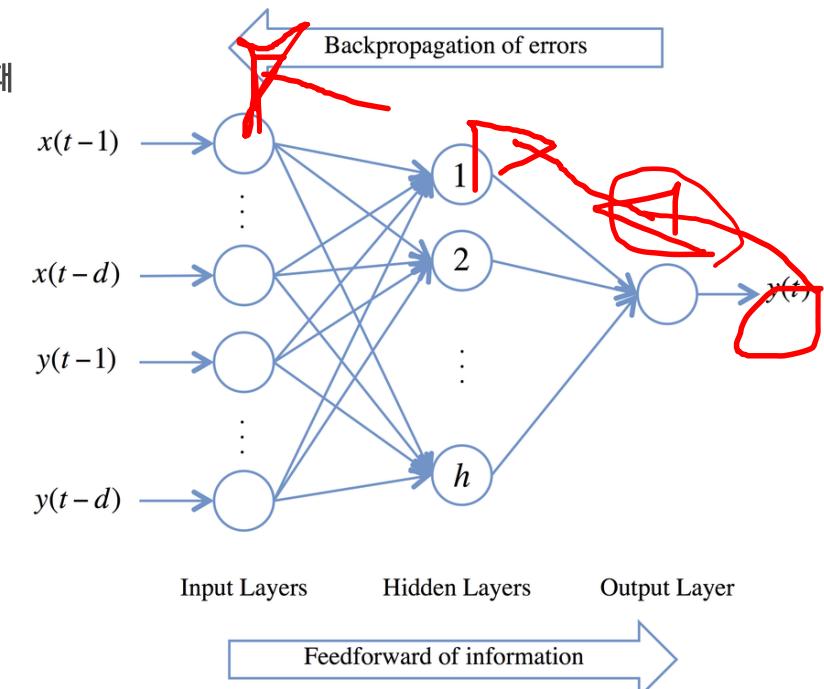
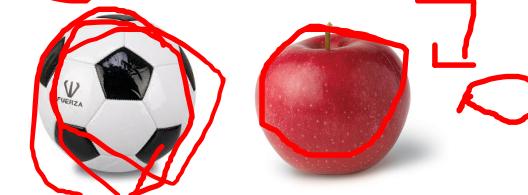
Underfitting and Overfitting

Overfitting : 많은 공통 특성 이외의 지역적인 특성까지 반영하여, *high variance*하게 train 되어 새로운 데이터에 대해서는 예측하지 못하는 모델, Regularization과 Data-augmentation으로 극복할 수 있지만 제일 좋은건 많은 양의 데이터를 갖는 것

Underfitting : 많은 공통 특성 중 일부 특성만 반영하여 *too bias*하게 train 되어, 새로운 데이터도 막 예측 해버리는 모델

* Bias : 실제 값에서 멀어진 척도

* Variance : 예측된 값들이 서로 얼마나 떨어져 있는가

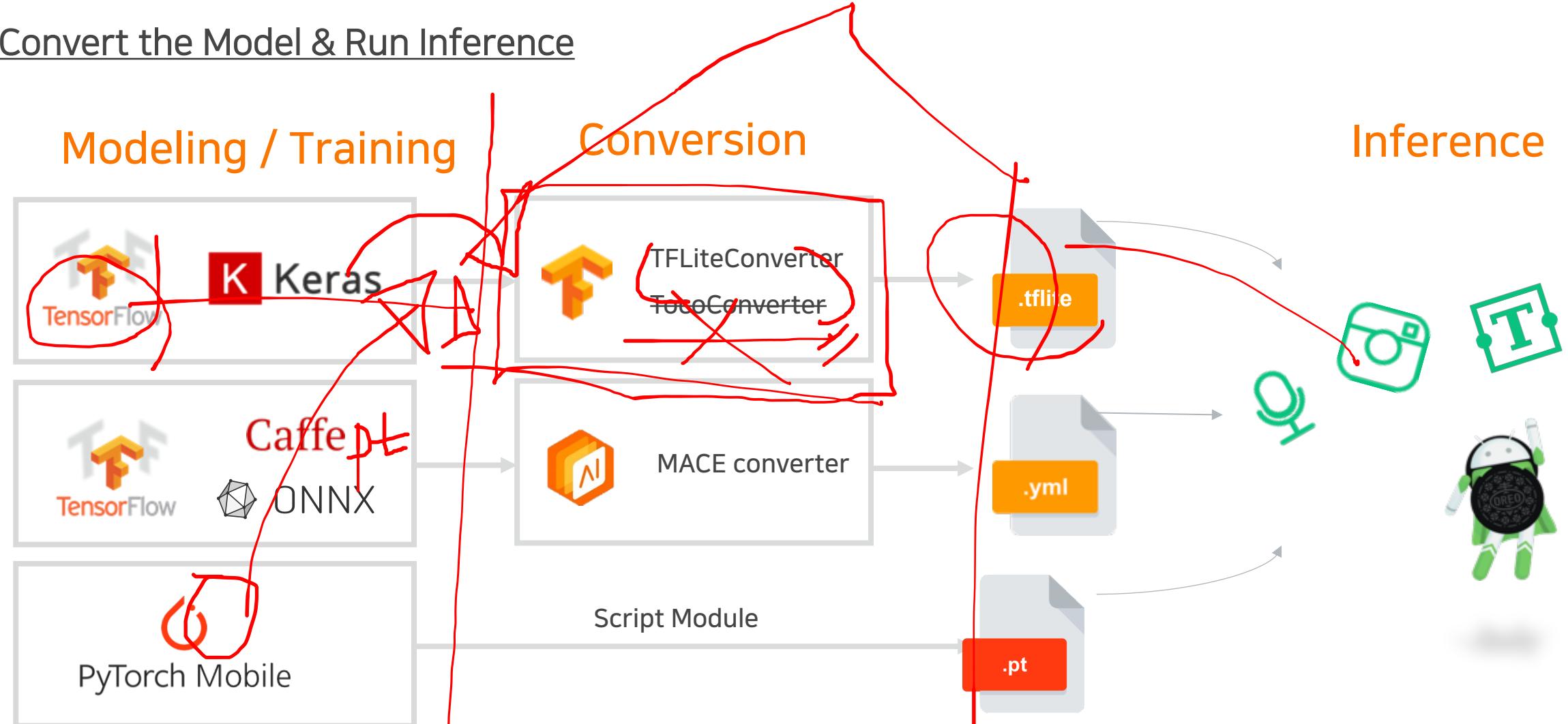


Deep Learning Workflow

Chapter 3. Getting up to Speed on Machine Learning

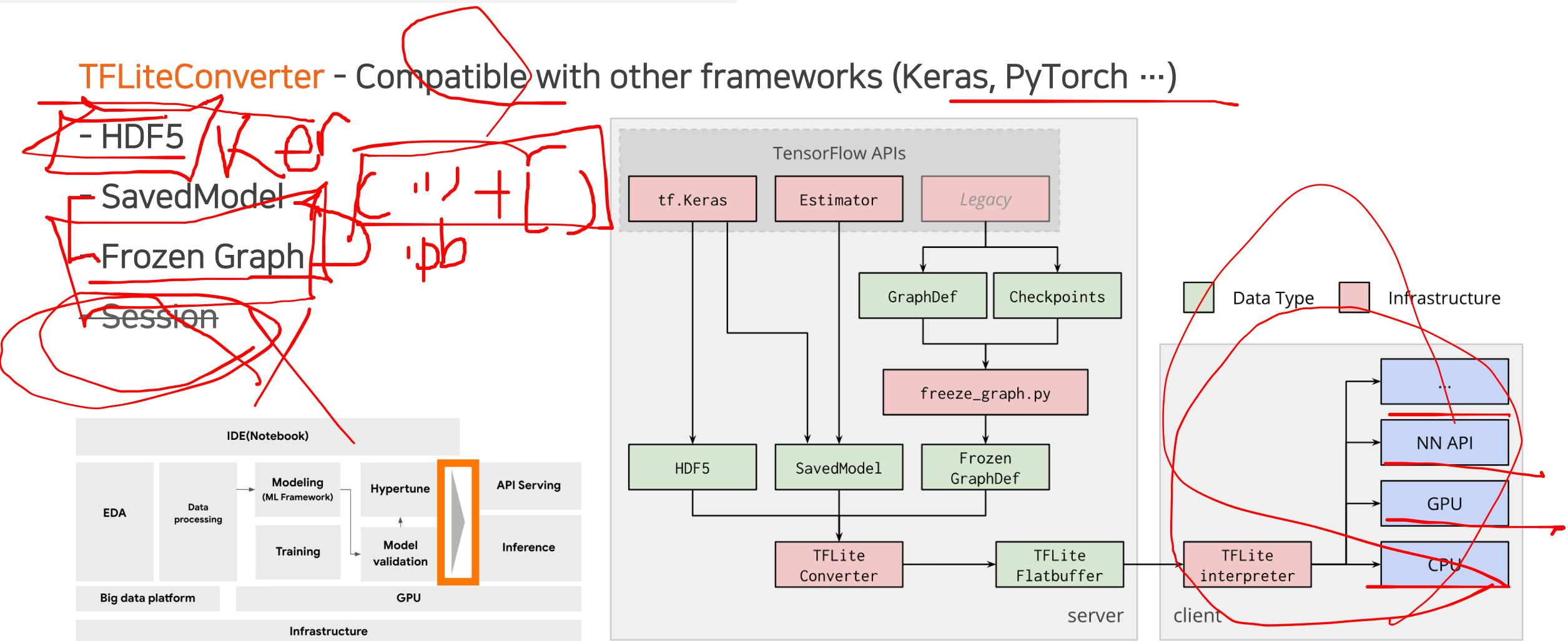
12
34

Convert the Model & Run Inference



Tensorflow Lite Model Conversion Pipeline

Chapter 3. Getting up to Speed on Machine Learning



Tensorflow Lite Model Conversion Code Example

Tensorflow 1.x implementation - Pose Estimation model

```
import argparse
import os

os.environ['CUDA_VISIBLE_DEVICES'] = ''
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
parser = argparse.ArgumentParser(description="Tools for convert frozen_pb into tflite or coreml.")
parser.add_argument("--frozen_pb", type=str, default="model-23500.pb", help="Path for storing checkpoint.")
parser.add_argument("--input_node_name", type=str, default="image", help="Name of input node name.")
parser.add_argument("--output_node_name", type=str, default="hourglass_out_3", help="Name of output node name.")
parser.add_argument("--output_path", type=str, default="./result", help="Path for storing tflite & coreml")
parser.add_argument("--type", type=str, default="tflite", help="tflite or coreml")

args = parser.parse_args()

output_filename = args.frozen_pb.rsplit("/", 1)[0]
output_filename = output_filename.split(".")[0]

if "tflite" in args.type:
    import tensorflow as tf
    output_filename += ".tflite"
    converter = tf.contrib.lite.TFLiteConverter.from_frozen_graph(
        args.frozen_pb,
        [args.input_node_name],
        [args.output_node_name]
    )

    tflite_model = converter.convert()
    open(os.path.join(args.output_path, output_filename), "wb").write(tflite_model)
    print("Generate tflite success.")
```

Tensorflow Lite Model Conversion Code Example

Tensorflow 2.x implementation with tflite-support

```
import tensorflow as tf

# Construct a basic model.
root = tf.train.Checkpoint()
root.v1 = tf.Variable(3.)
root.v2 = tf.Variable(2.)
root.f = tf.function(lambda x: root.v1 * root.v2 * x)

# Save the model in SavedModel format.
export_dir = "/tmp/test_saved_model"
input_data = tf.constant(1., shape=[1, 1])
to_save = root.f.get_concrete_function(input_data)
tf.saved_model.save(root, export_dir, to_save)

# Convert the model.
converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)
tflite_model = converter.convert()

# Save the TF Lite model.
with tf.io.gfile.GFile('model.tflite', 'wb') as f:
    f.write(tflite_model)
```

약속 해주세요! 원활한 협업을 위한 Metadata 작성

```
input_meta.name = "image"
input_meta.description =
    "Input image to be classified. The expected image is {0} x {1}, with "
    "three channels (red, blue, and green) per pixel. Each value in the "
    "tensor is a single byte between 0 and 255.".format(160, 160)
input_meta.content = _metadata_fb.ContentT()
input_meta.content.contentProperties = _metadata_fb.ImagePropertiesT()
input_meta.content.contentProperties.colorSpace =
    _metadata_fb.ColorSpaceType.RGB
input_meta.content.contentPropertiesType =
    _metadata_fb.ContentProperties.ImageProperties
input_normalization = _metadata_fb.ProcessUnitT()
input_normalization.optionsType =
    _metadata_fb.ProcessUnitOptions.NormalizationOptions
input_normalization.options = _metadata_fb.NormalizationOptionsT()
input_normalization.options.mean = [127.5]
input_normalization.options.std = [127.5]
input_meta.processUnits = [input_normalization]
input_stats = _metadata_fb.StatsT()
input_stats.max = [255]
input_stats.min = [0]
input_meta.stats = input_stats
```

Post-training Quantization (During Conversion)

Chapter 3. Getting up to Speed on Machine Learning

✓ Post-training Quantization - Weight ONLY

```
import tensorflow as tf
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_quant_model = converter.convert()
```

- ✓ Post-training Quantization with Tensorflow Model Optimization Toolkit
 - Fully Quantized (weight/activation)

```
import tensorflow as tf

def representative_dataset_gen():
    for _ in range(num_calibration_steps):
        # Get sample input data as a numpy array in a method of your choosing
        yield [input]

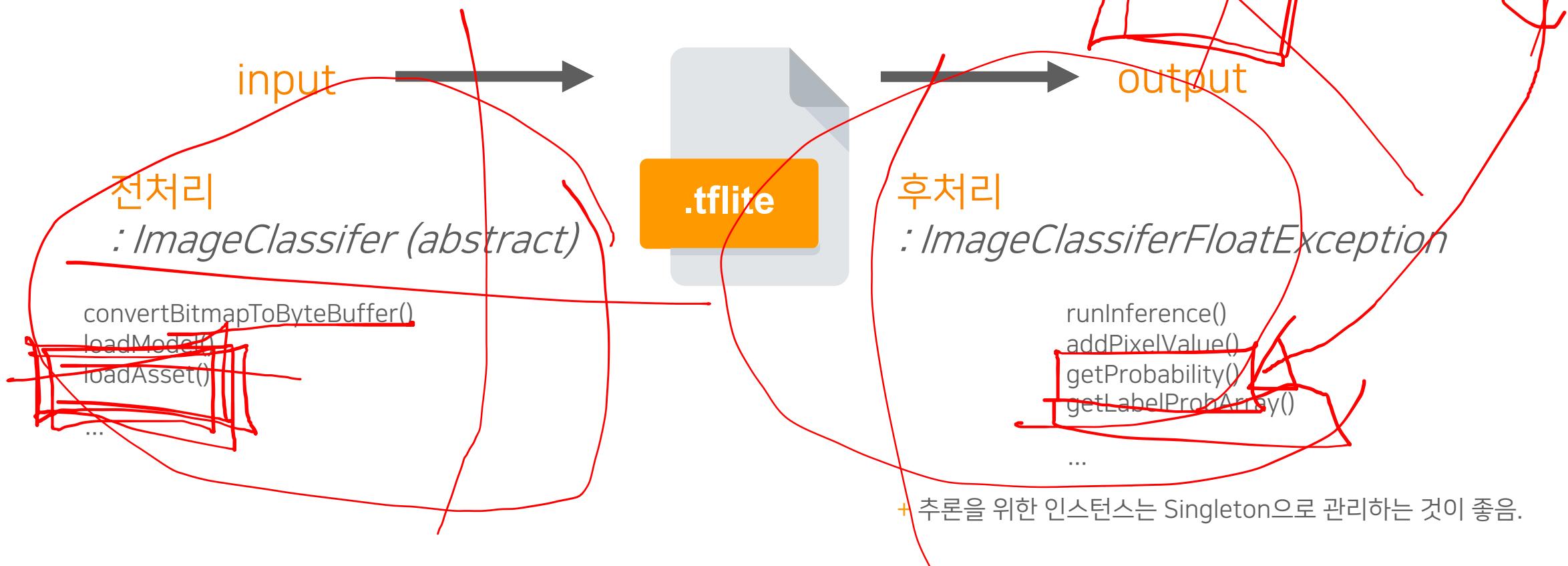
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset_gen
tflite_quant_model = converter.convert()
```



Model Size : 9.6mb → 5.2mb / End2End : 30 ~ 45 fps (Mobile GPU 가속시),
accuracy loss 1% 이내

18

전처리/후처리 클래스에 어떤 역할을 위임할 것인가?



System Architecture for Inferencing Model (Android)

Chapter 3. Getting up to Speed on Machine Learning

```
if (fingerPoseClassifier?.mPrintPointArray.isNullOrEmpty() == false){  
    fpeDrawView?.setDrawPoint(fingerPoseClassifier?.mPrintPointArray!!, 0.5f)  
    showAndroidAnimation(fpeDrawView?.mDrawPoint.x!!,  
    fpeDrawView?.mDrawPoint.y!!)  
}
```

output을 통해 수행할
~~Business Logic or Presentation Logic~~

: Canvas를 통해 Finger Tip의 위치를 표시하고,
그 위로 Lottie Animation을 송출한다.

AutofitTextureView

: 전면 카메라 프리뷰



FpeDrawview

: Finger Tip을 표시하기 위한 Overlay

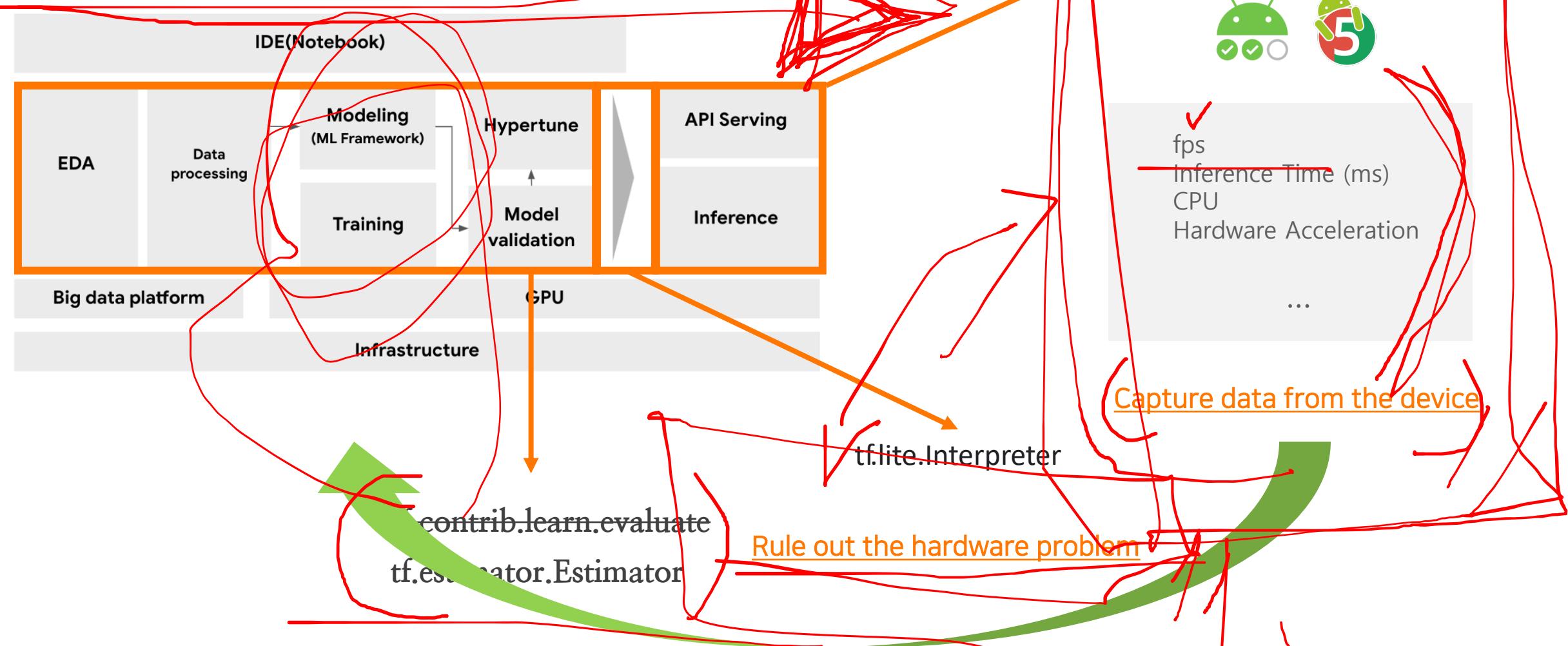
LottieAnimationView

: Finger Tip 위에 송출되는 애니메이션

Deep Learning Workflow

Chapter 3. Getting up to Speed on Machine Learning

Evaluate & Troubleshoot



Evaluate – Performance Benchmark

Chapter 3. Getting up to Speed on Machine Learning

For specific target device

```
bazel build -c opt \
--config=android_arm64 \
tensorflow/lite/tools/benchmark:benchmark_model
```

For general

```
bazel build -c opt tensorflow/lite/tools/benchmark:benchmark_model
```

```
adb shell /data/local/tmp/benchmark_model \
--num_threads=4 \
--graph=/data/local/tmp/tflite_models/${GRAPH} \
--warmup_runs=1 \
--num_runs=50
```

```
bazel-bin/tensorflow/lite/tools/benchmark/benchmark_model \
--graph=mobilenet_quant_v1_224.tflite \
--num_threads=4
```