Hardware Architectures for Embedded and Edge AI

Prof Manuel Roveri – manuel.roveri@polimi.it
Massimo Pavan – massimo.pavan@polimi.it

Exercise session 7 – Training and Deploying VWW Detection

What’s visual wake word detection?

- A task of computer vision
- Recognize if an object is present in a picture
- Usually few «wake words», very often binary:
  - Object present
  - Object not present
- May be included in a cascade pipeline

* The picture is misleading, no actual bounding boxes will be drawn during this lecture

Visual wake word detection: Challenges and opportunities

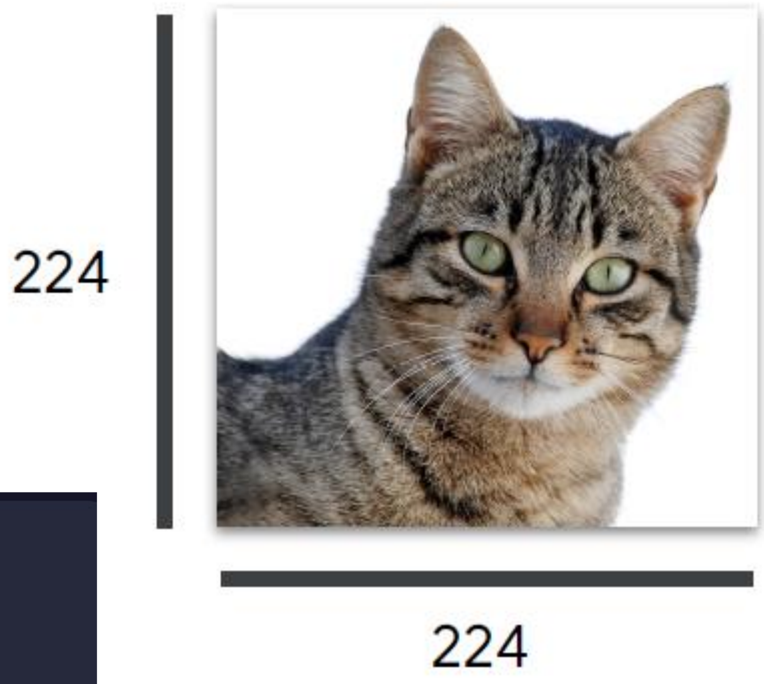
$$\underbrace{224 \times 224}_{\text{Pixels}} \times \underbrace{3}_{\text{N of channels}} \times \underbrace{4}_{\text{Bytes per channel}} = 602{,}112 \text{ Bytes}$$

224

224

Visual wake word detection: Could it run in cloud?

$$\underbrace{224 \times 224}_{\text{Pixels}} \times \underbrace{3}_{\text{N of channels}} \times \underbrace{4}_{\text{Bytes per channel}} = 602{,}112 \text{ Bytes}$$



4.6Mbps = 570k ***Bytes*** / Sec

~1 second Transfer Time

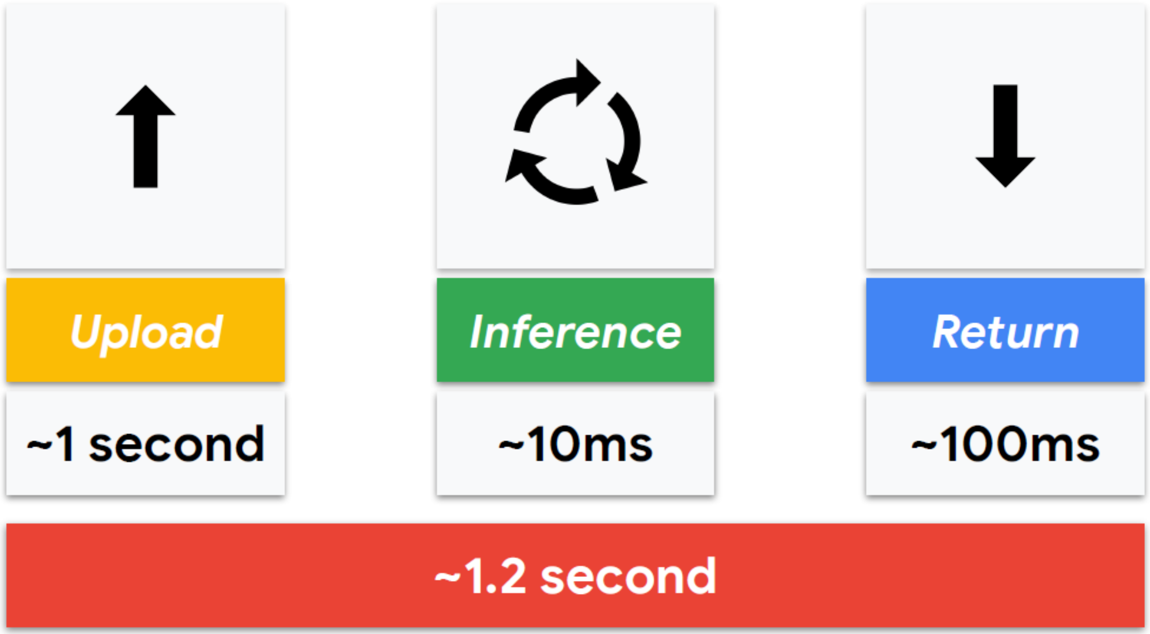
Visual wake word detection: Could it run in cloud?

Always-on (Visual Wake Words)?

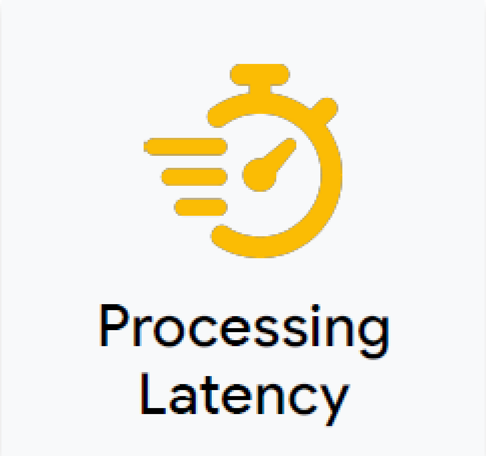
→ Much more data (than KWS)

- Higher latency
- Higher power consumption (drains battery)

→ Lower user satisfaction



Visual wake word detection: Challenges



- Can we process data faster than sending it to the cloud?
- Can we process them fast enough to perform inference in «real-time?

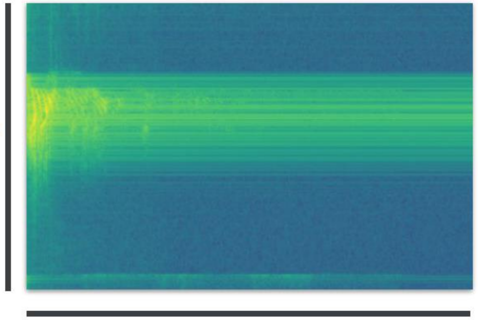
Visual wake word detection: Challenges

Processing Latency

Memory

Model	Size	Top-1 Accuracy
Xception	88 MB	0.790
VGG16	528 MB	0.713
ResNet50	98 MB	0.749
Inception v3	92 MB	0.779
MobileNet v1	16 MB	0.713
DenseNet 201	80 MB	0.773
NASNetMobile	23 MB	0.825

40



49

$$\underbrace{49 \times 40}_{\text{Pixels}} \times \underbrace{1}_{\text{RGB (\# channels)}} \times \underbrace{4}_{\text{Bytes/Pixel}} = 7{,}840 \text{ Bytes}$$

$$\underbrace{224 \times 224}_{\text{Pixels}} \times \underbrace{3}_{\text{RGB (\# channels)}} \times \underbrace{4}_{\text{Bytes/Pixel}} = 602{,}112 \text{ Bytes}$$

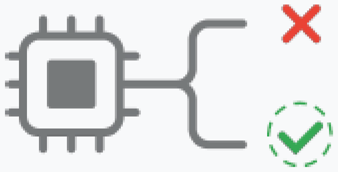
224

224

Visual wake word detection: Challenges

Processing Latency

Memory



False Positives / Negatives

- How much are we giving up in terms of accuracy with respect to larger models?
- Does our application really require to recognize a large amount of classes?

Collect and Pre-Process Data

Collect Data → Pre-process data → Design Model → Train a Model → Evaluate optimize → Convert Model → Deploy Model → Make inference

Data collection and processing

- Computer vision algorithms require extremely large amount of data in order to be trained from scratch
- Can we reuse already available data?
  - Pictures online are very often under copyright
  - Reusing **existing datasets** may be an option
  - Consider what's **available** and what's **missing**
  - Consider **bias** in re-used dataset

The Visual Wake Word Dataset

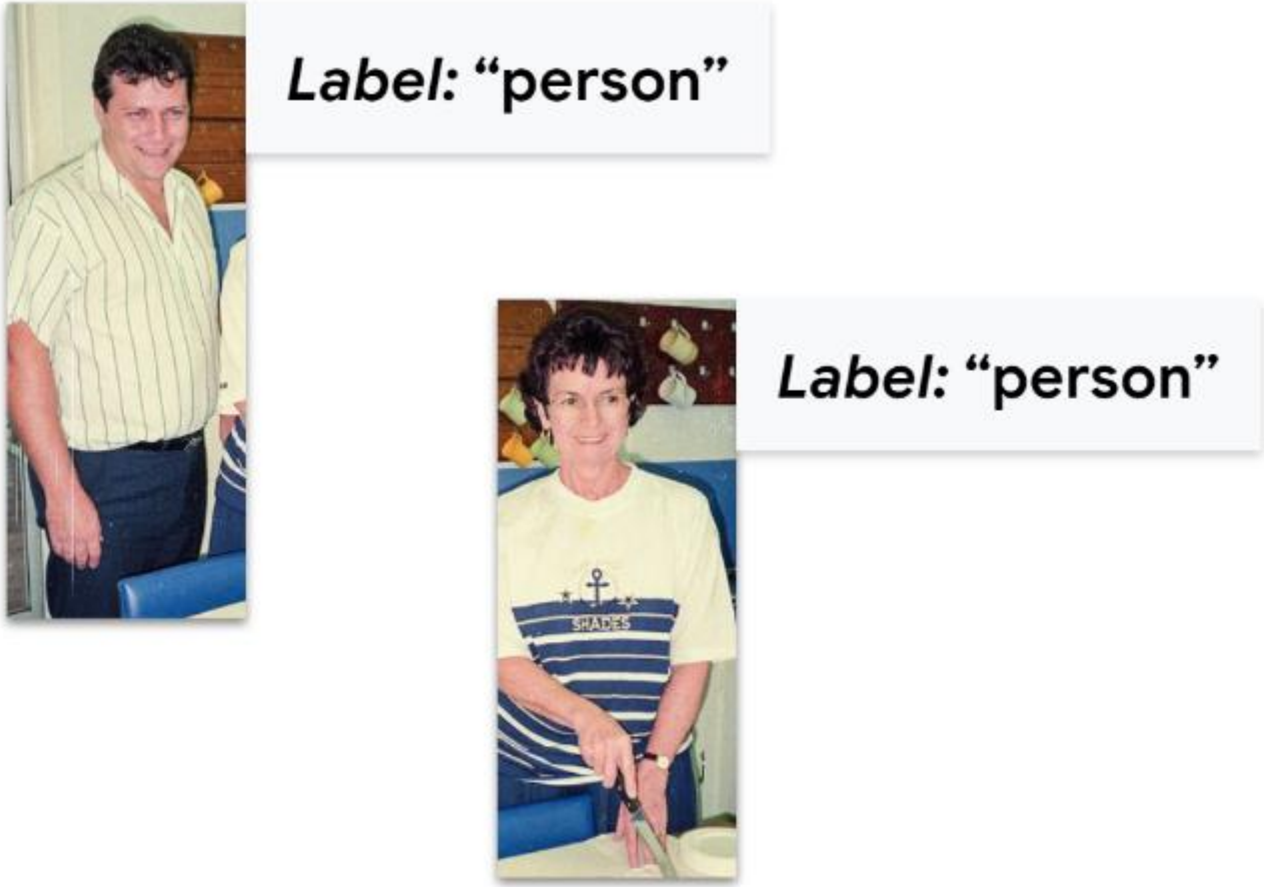
Visual Wake Words Dataset

Aakanksha Chowdhery, Pete Warden, Jonathon Shlens,
Andrew Howard, Rocky Rhodes
Google Research
{chowdhery, petewarden, shlens, howarda, rocky}@google.com

Visual Wake Words dataset


```
"annotations": [
    {
        "segmentation": [[510.66,423.01,511.72,420.03,...,510.45,423.01]],
        "area": 702.1057499999998,
        "iscrowd": 0,
        "image_id": 289343,
        "bbox": [473.07,395.93,38.65,28.67],
        "category_id": 18,
        "id": 1768
    },
```

Visual Wake Words dataset



Visual Wake Words dataset

Data collection is DIFFICULT

- This dataset and collection process is limited and has bias
- Small number of relevant images
- Large quantity of irrelevant images

Visual Wake Words Dataset

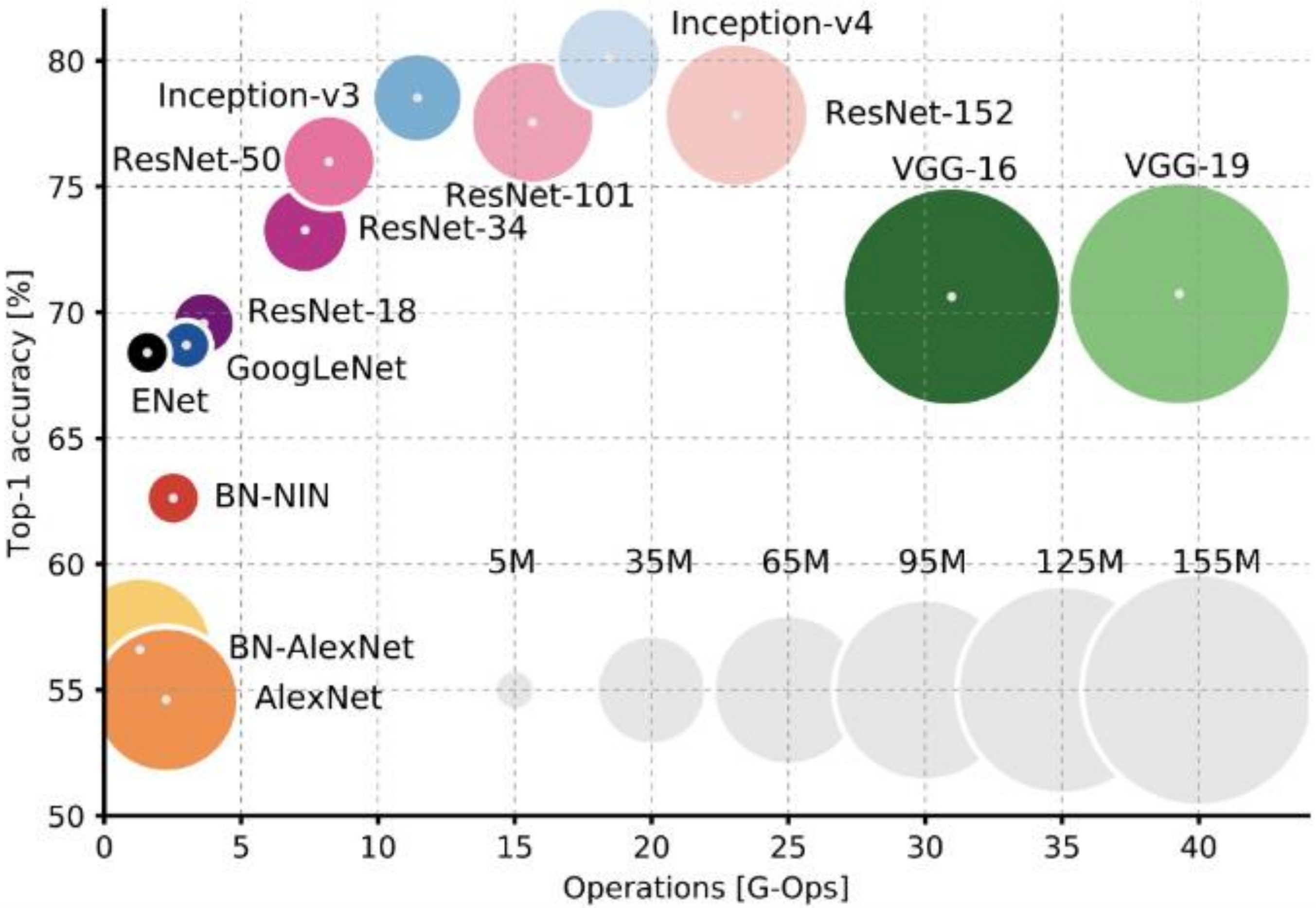
Aakanksha Chowdhery, Pete Warden, Jonathon Shlens,
Andrew Howard, Rocky Rhodes
Google Research
{chowdhery, petewarden, shlens, howarda, rocky}@google.com

POLITECNICO
MILANO 1863

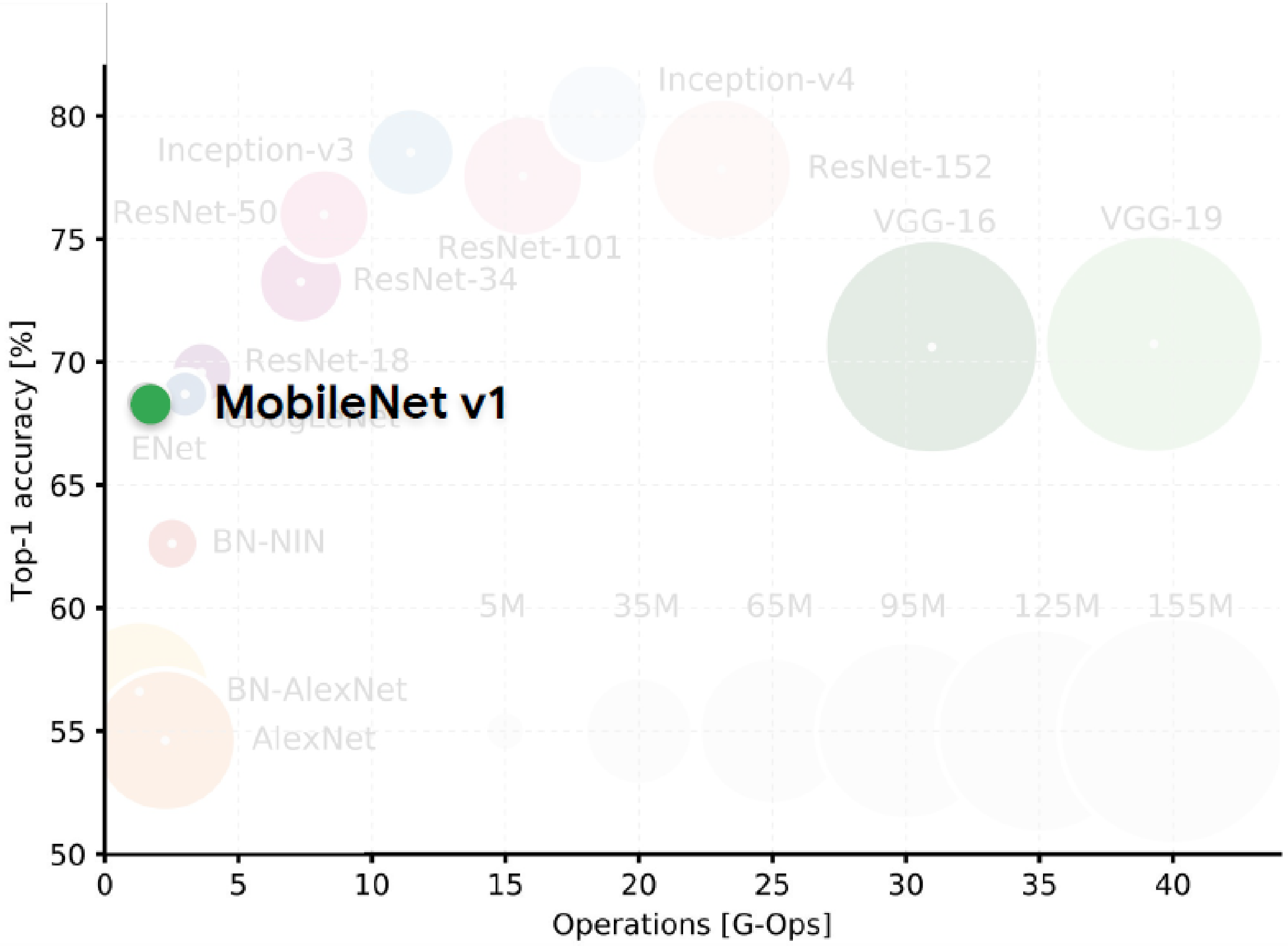
Training VWW Detection

Collect Data → Pre-process data → Design Model → Train a Model → Evaluate optimize → Convert Model → Deploy Model → Make inference

Models evolution



Mobilenet V1



Model	Size	Top-1 Accuracy
MobileNet v1	***16 MB***	0.713

Fine for mobile phones with GB of RAM, but 64X microcontroller RAM

Mobilenet V1: The Depth multiplier

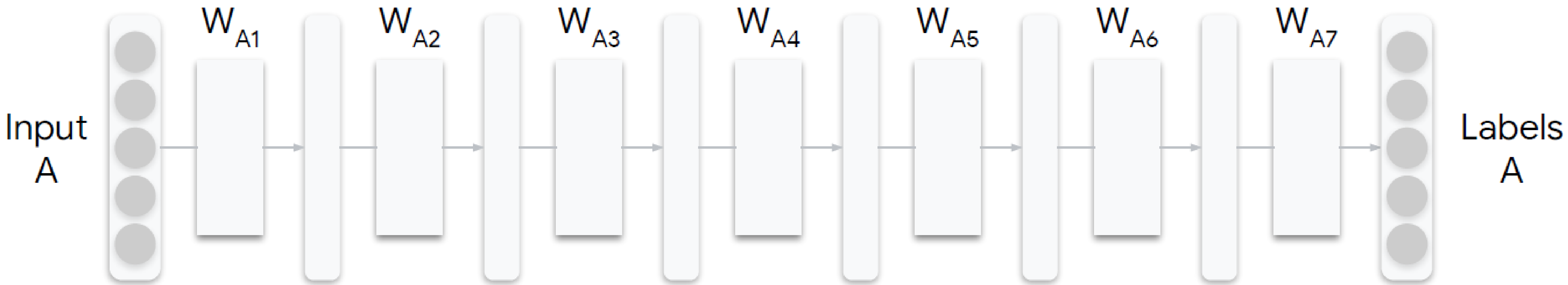
- Effect of depth multiplier on model size → top-1 accuracy
- The size of the model can be reduced further by parameter $\boldsymbol{\alpha}$
- $\boldsymbol{\alpha} \rightarrow (0, 1]$

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$$

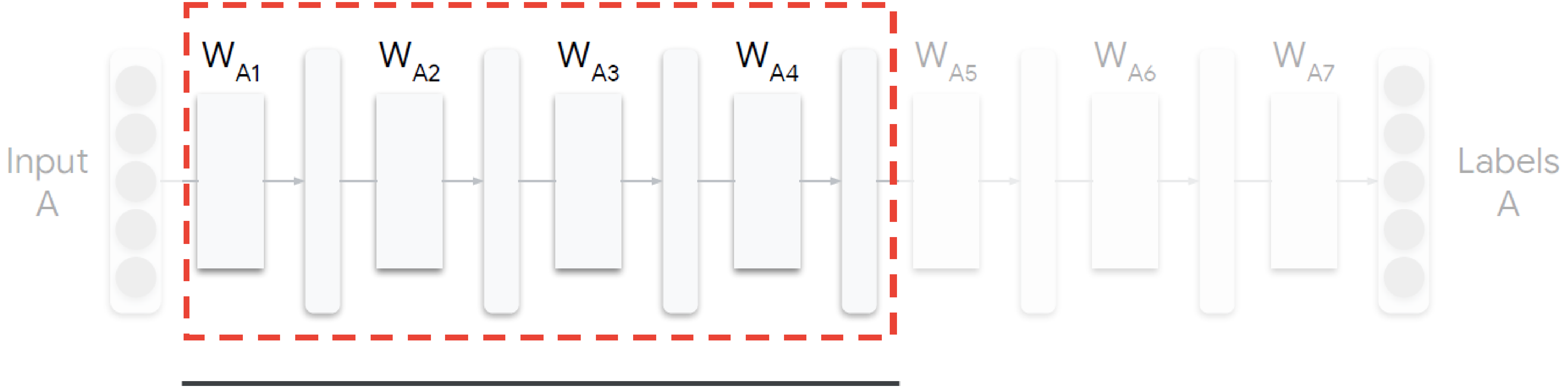
The accuracy vs memory-MACs tradeoff

α	Image Size	MACs (millions)	Params (millions)	Top-1 Accuracy
1	224	569	4.24	70.7
1	128	186	4.14	64.1
0.75	224	317	2.59	68.4
0.75	128	104	2.59	61.8
0.5	224	150	1.34	64.0
0.5	128	49	1.34	56.2
0.25	224	41	0.47	50.6
0.25	128	14	0.47	41.2

Transfer Learning

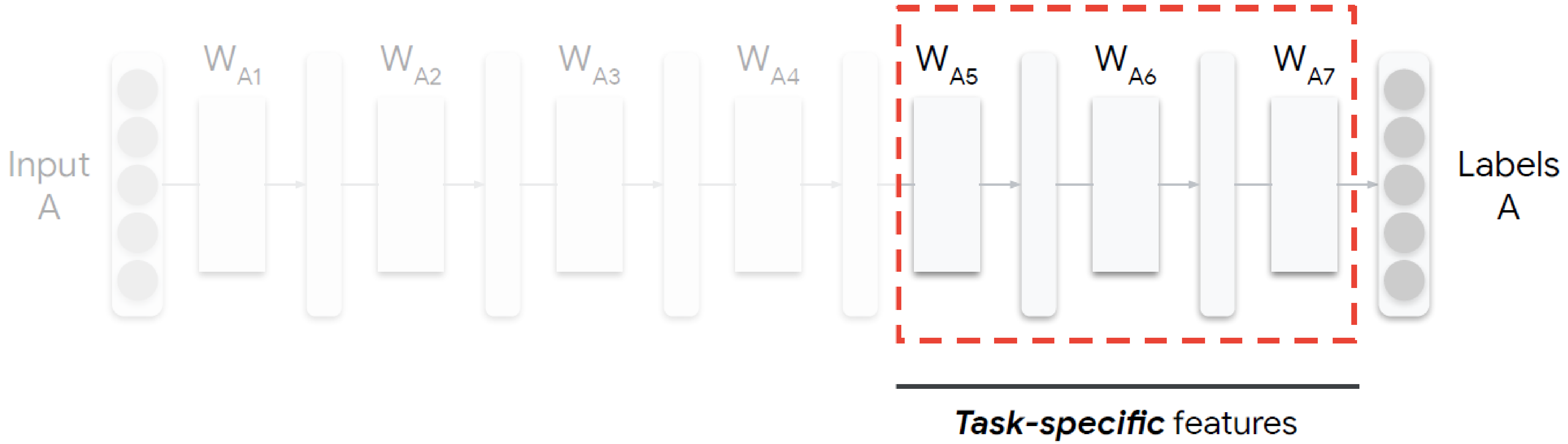


Transfer Learning



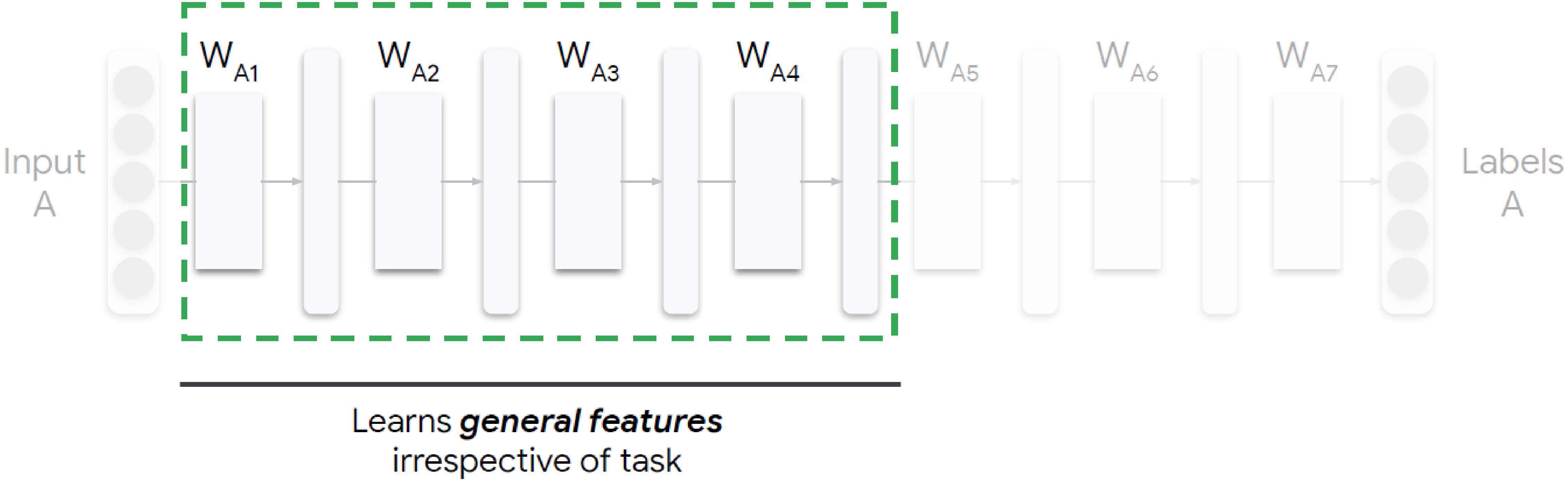
Learns ***general features***
irrespective of task

Transfer Learning



Transfer Learning

Reuse: (freeze general feature extractor)



Colab: Transfer Learning of mobilenet V1

Link colab:

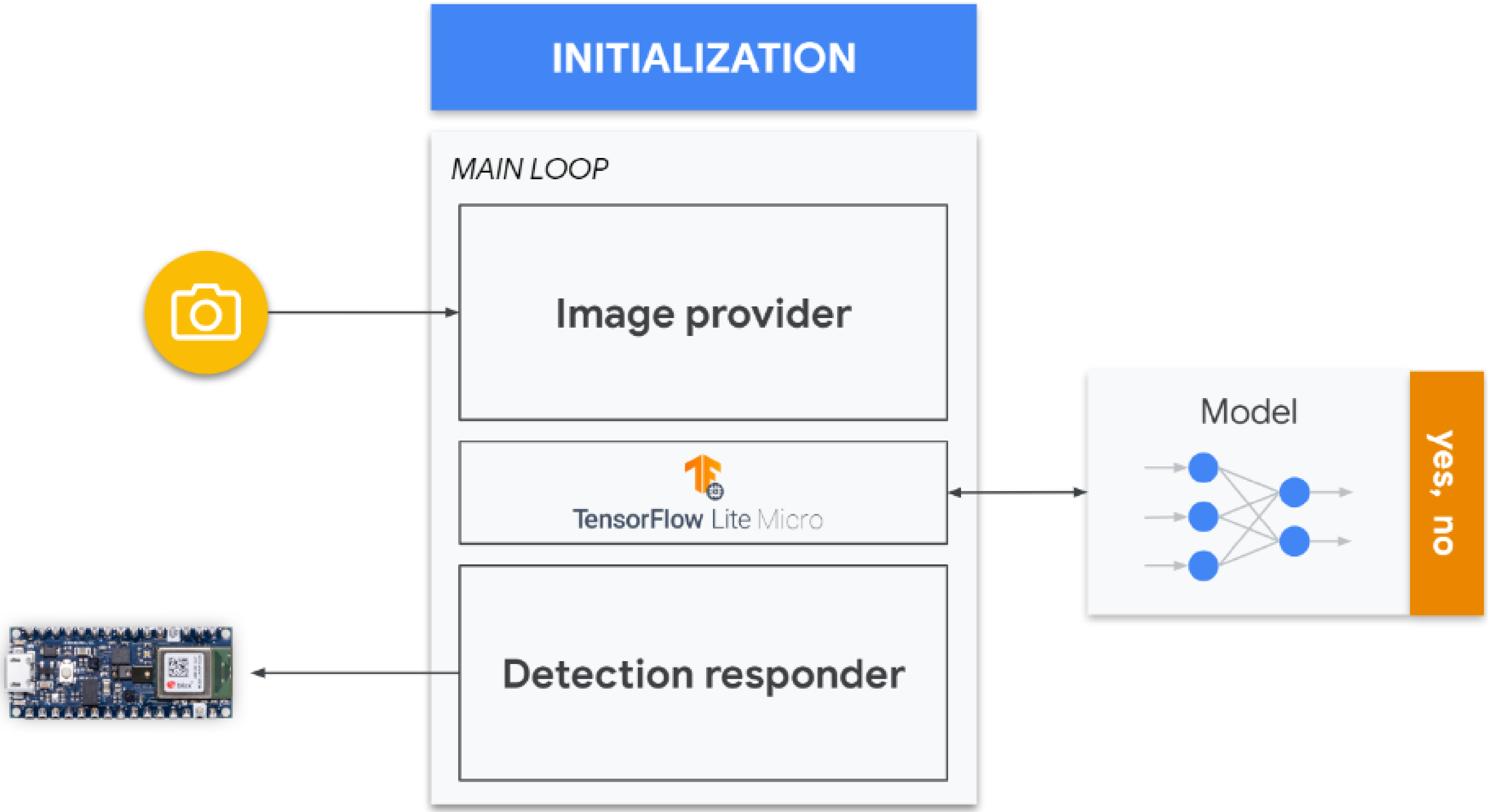
https://colab.research.google.com/drive/1dwGMx3OmzoOo0aGEpRYD7uRTVJc98iQh?usp=sharing

POLITECNICO MILANO 1863

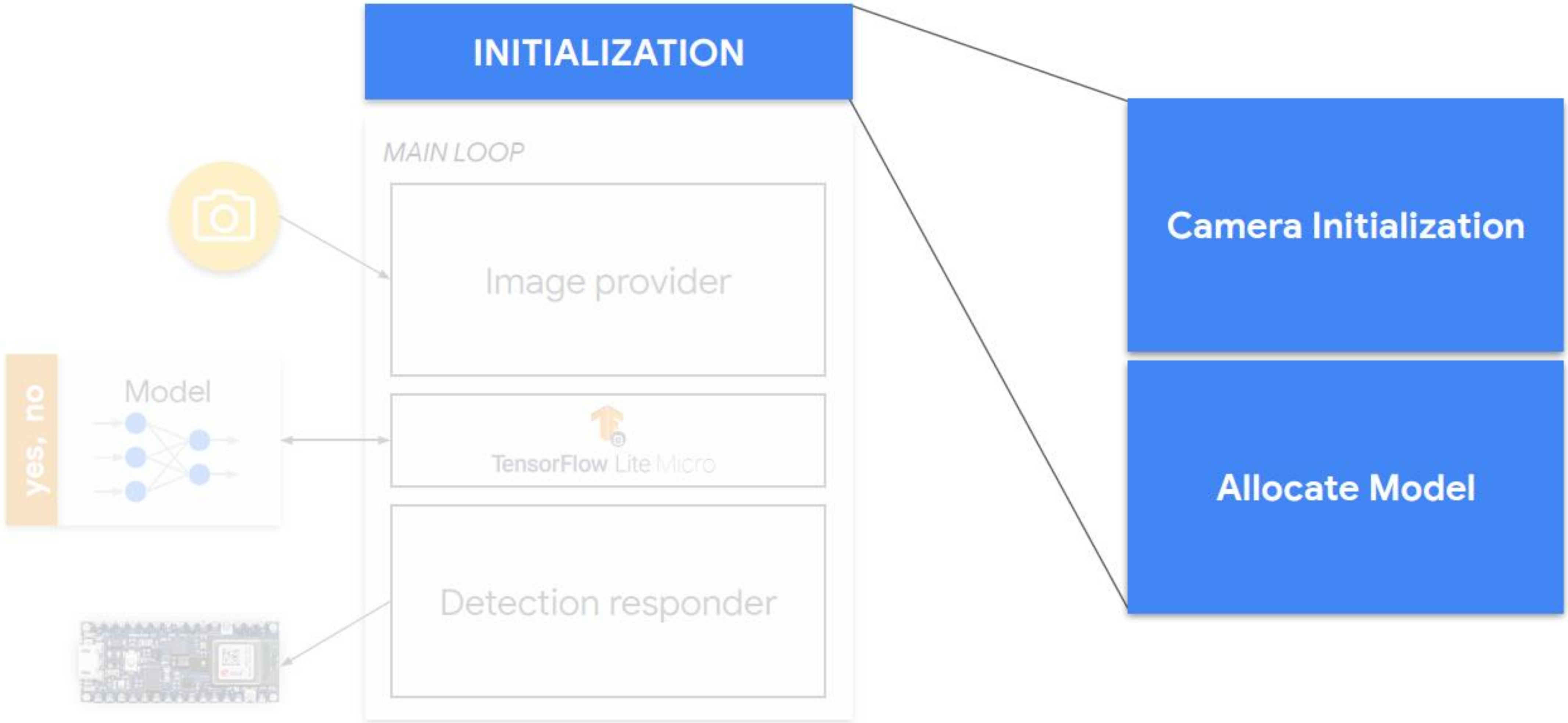
Deploying VWW Detection

Collect Data → Pre-process data → Design Model → Train a Model → Evaluate optimize → Convert Model → Deploy Model → Make inference

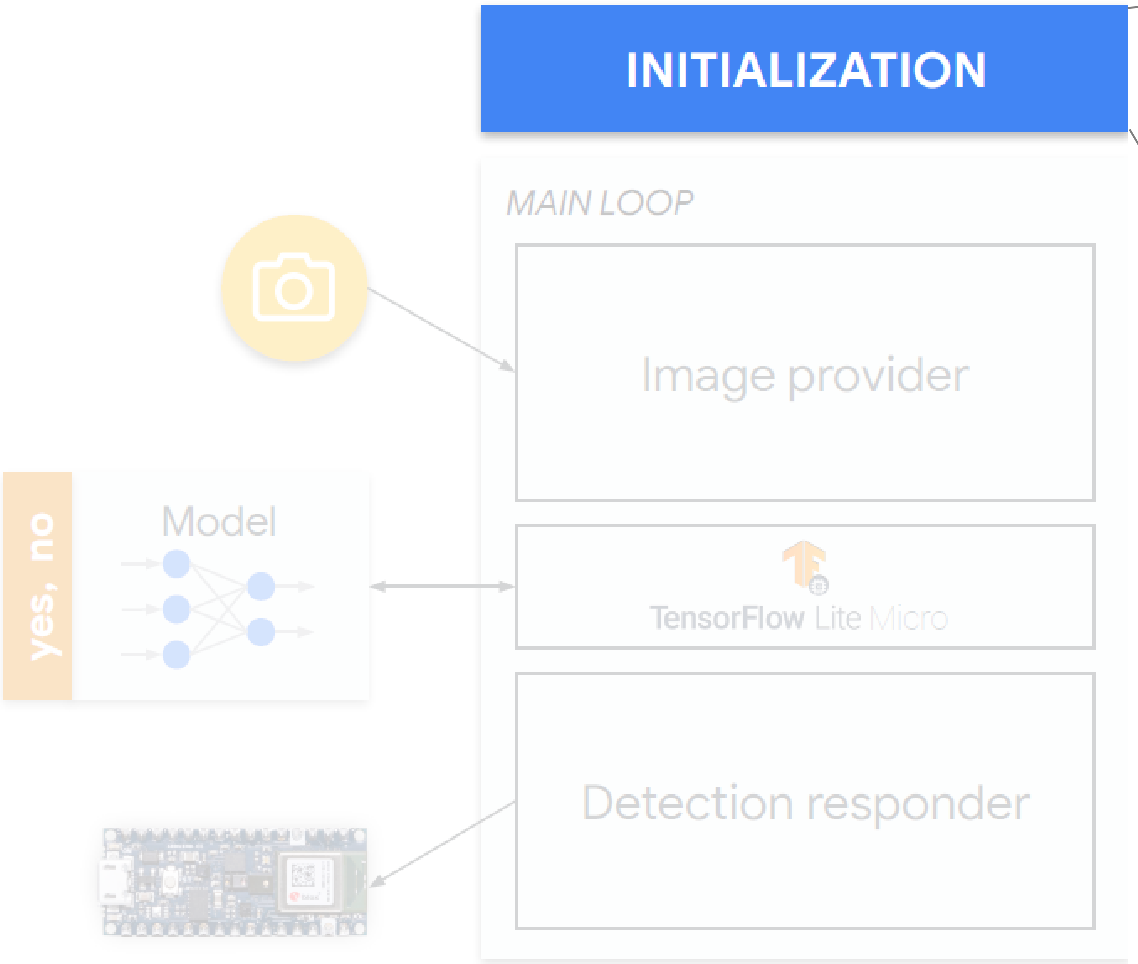
VWW Detection components



Initialization



Initialization



- Declare variables
- Load Model
- Resolve Operators
- Initialize interpreter
- Allocate Arena
- Define Model Inputs
- Set Up Main Loop

Camera Initialization

Camera Initialization

Allocate model

Color

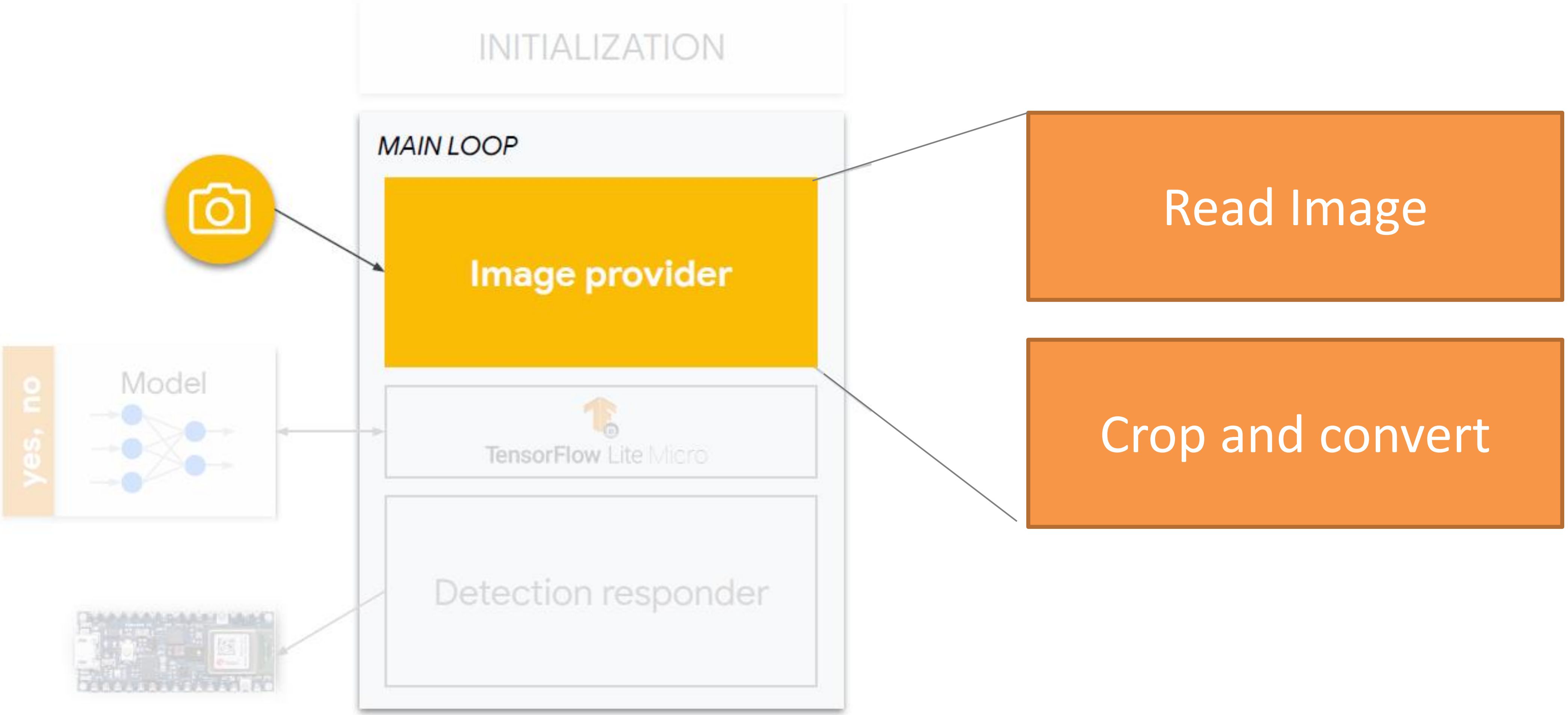
Camera model

```
// Initialize camera if necessary
if (!g_is_camera_initialized) {
  if (!Camera.begin(QCIF, GRAYSCALE, 5, OV7675)) {
    TF_LITE_REPORT_ERROR(error_reporter, "Failed to
                         initialize
                         camera!");
    return kTfLiteError;
  }
  g_is_camera_initialized = true;
}
```

FPS

Resolution

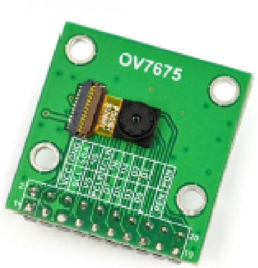
Pre-processing

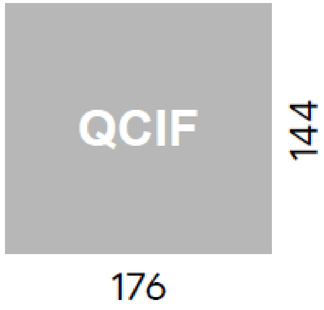


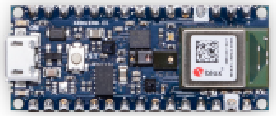
Pre-processing

Read Image

Crop and convert





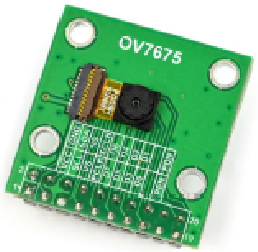


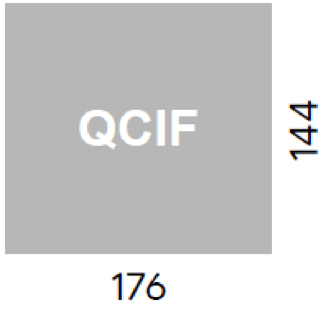
```
// Get an image from the camera module
TfLiteStatus GetImage(tflite::ErrorReporter* error_reporter,
          int image_width, int image_height, int channels,
          int8_t* image_data)
```

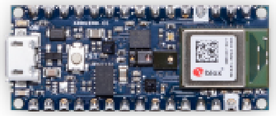
Pre-processing

Read Image

Crop and convert

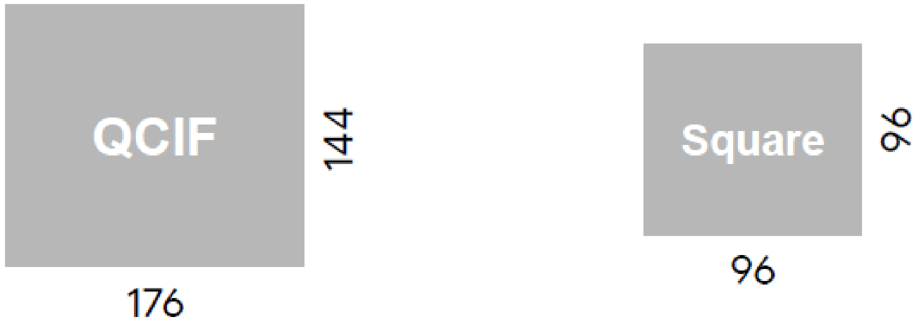






```
// Read camera data
Camera.readFrame(data);
```

Pre-processing

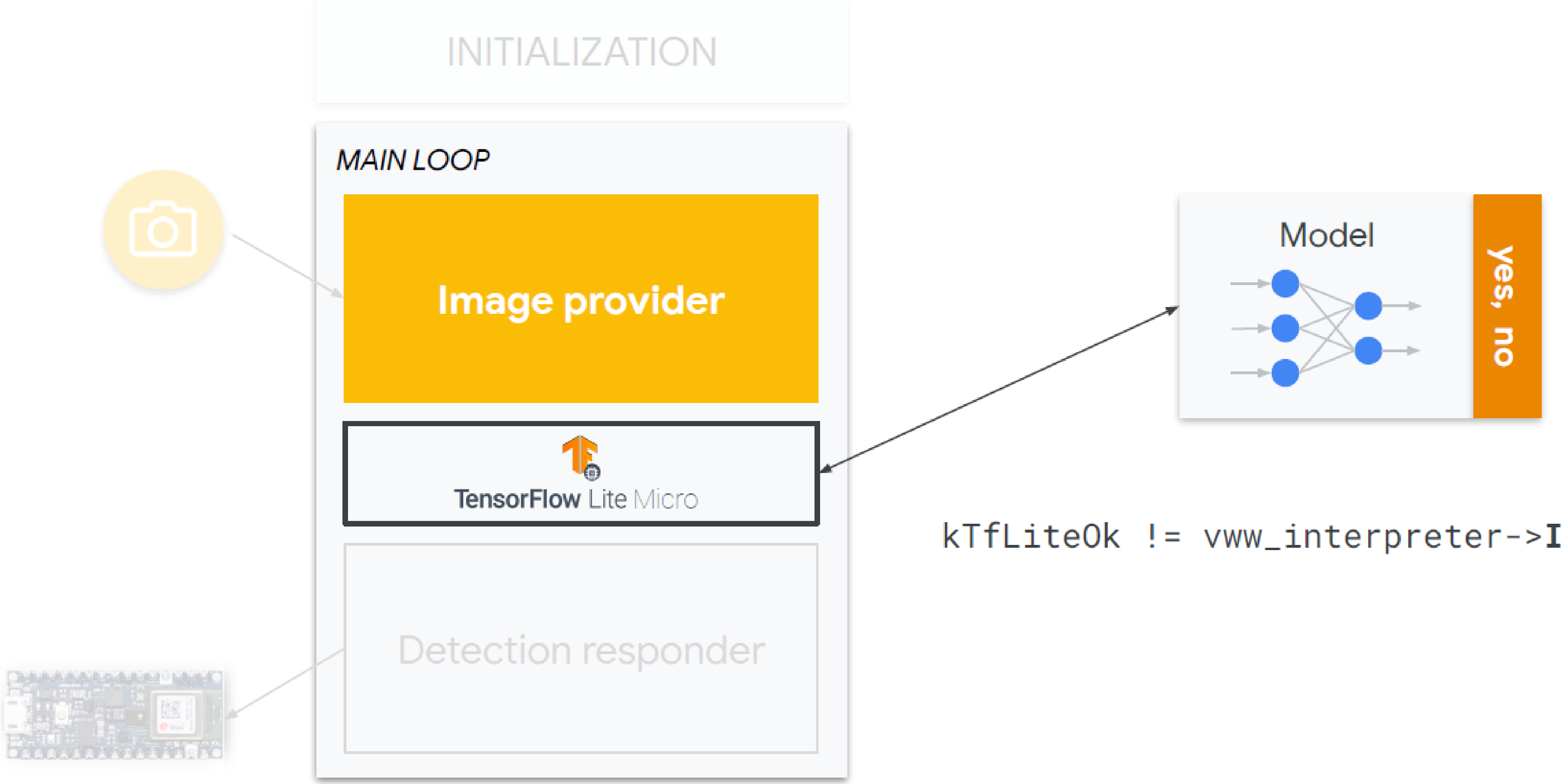


Read Image

Crop and convert

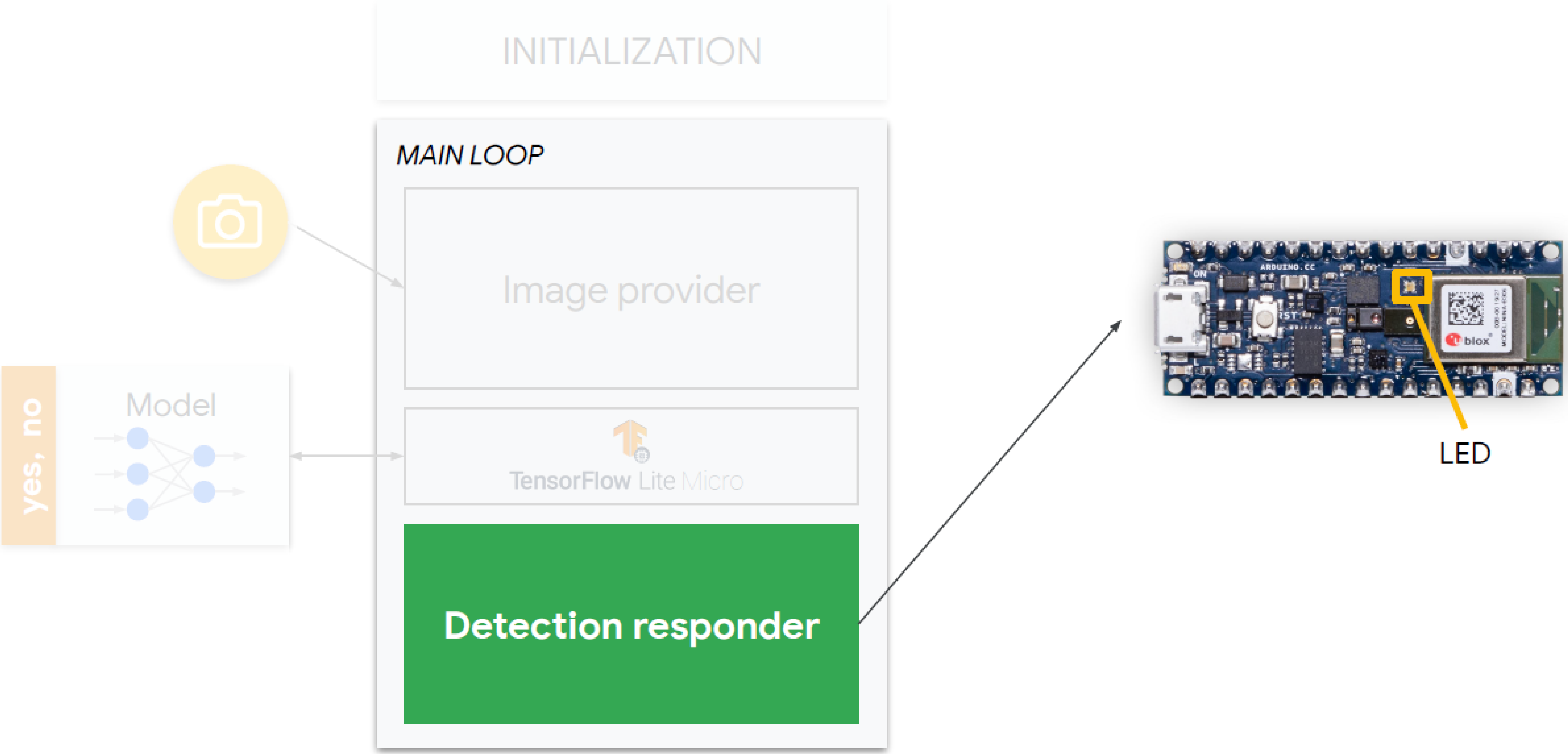
```
int min_x = (176 - 96) / 2;
int min_y = (144 - 96) / 2;
int index = 0;

// Crop 96x96 image. This lowers FOV, ideally we should downsample
for (int y = min_y; y < min_y + 96; y++) {
  for (int x = min_x; x < min_x + 96; x++) {
    image_data[index++] = static_cast<int8_t>(data[(y * 176) + x] - 128);
    // convert TF input image to signed 8-bit
  }
}
```

Model execution




```
kTfLiteOk != vww_interpreter->Invoke()
```

Postprocessing



POLITECNICO
MILANO 1863

Appendix

Credits and reference

- "TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers", Daniel Situnayake, Pete Warden, O'Reilly Media, Inc.
- Online course:
  - https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning
- A lot more material on TinyML:
  - http://tinyml.seas.harvard.edu/