EEP 596: TinyML
Lecture 4: Visual Wake Words

Dept. of Electrical and Computer Engineering
University of Washington

Instructor: Dinuka Sahabandu (sdinuka@uw.edu)

ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY of WASHINGTON

NSL

Topics Covered

- Introduction to Visual Wake Words (VWW) and Its Challenges
- Visual Wake Words Dataset
- MobileNets
- Transfer Learning for Visual Wake Words
- Lab 4: VWW Model Training, Evaluation Metrics, and Deployment

❖Relevant reading from the Textbook [Warden and Situnayake; O'REILLY Publisher] is Chapter 7

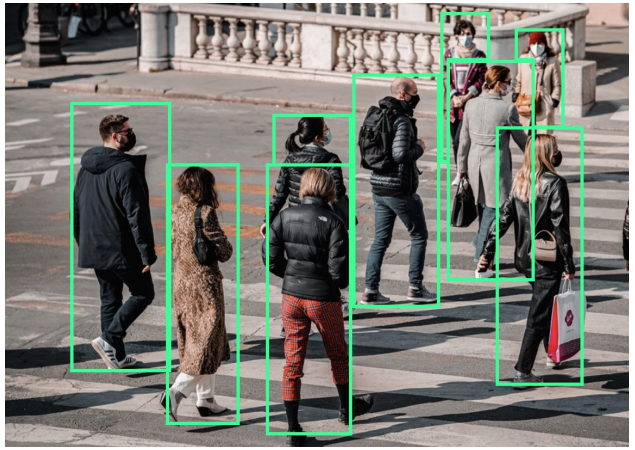
What are Visual Wake Words (VWW)

- Visual Wake Words – Identifying whether an object is present in the image or not.
- Key application of Tiny Vision (Image Detection and Classification).
- Using MCUNets, we can achieve ImageNet level classification on microcontrollers < $1.

Person Detected: Yes

Person Detected: No

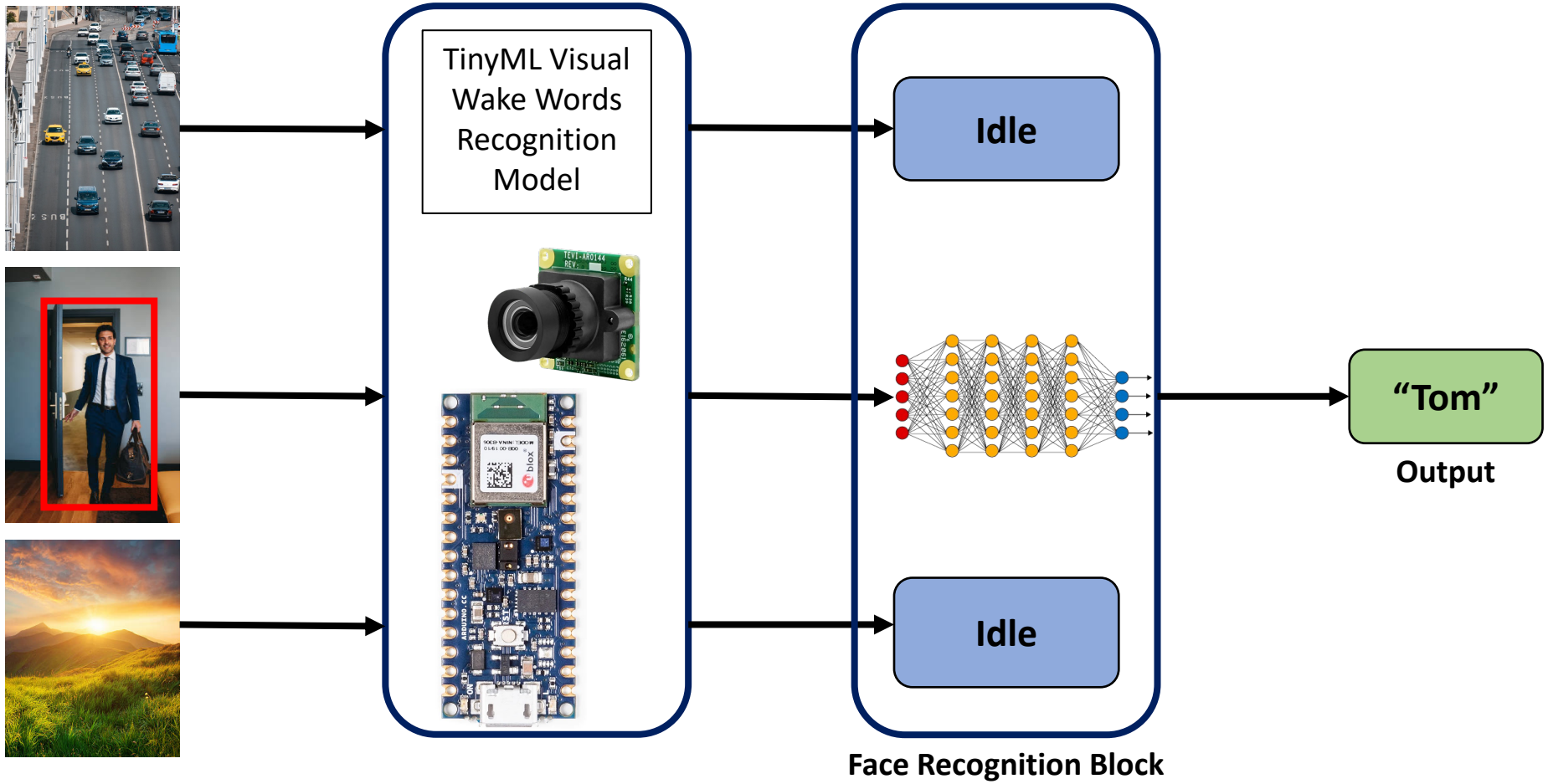
Person detected: Alert ON



Person detected: Multiple

What are Visual Wake Words

The TinyML model first determines whether the image contains a person or not and then implements the use case, for e.g., face recognition.

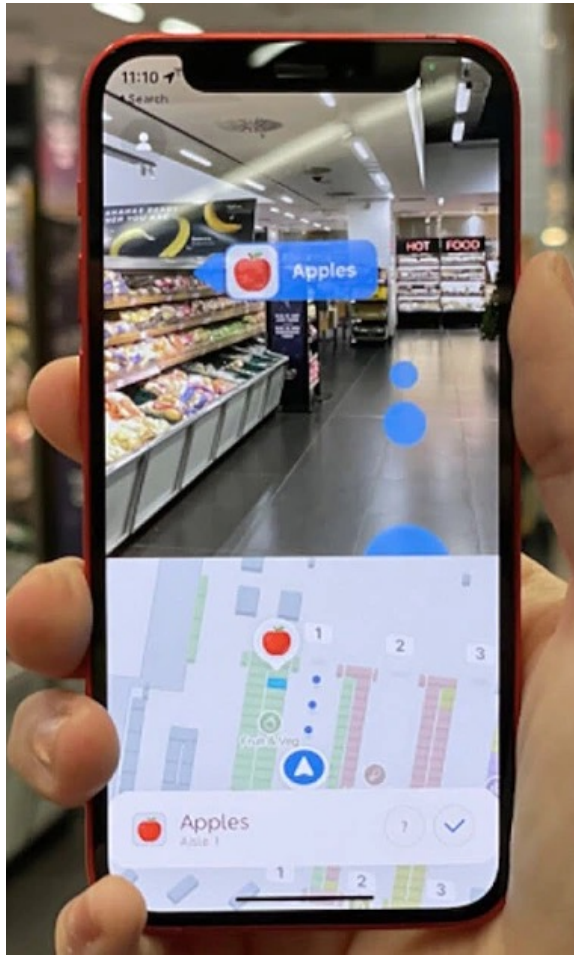


Applications of VWW

Security surveillance systems

Augmented Reality navigation

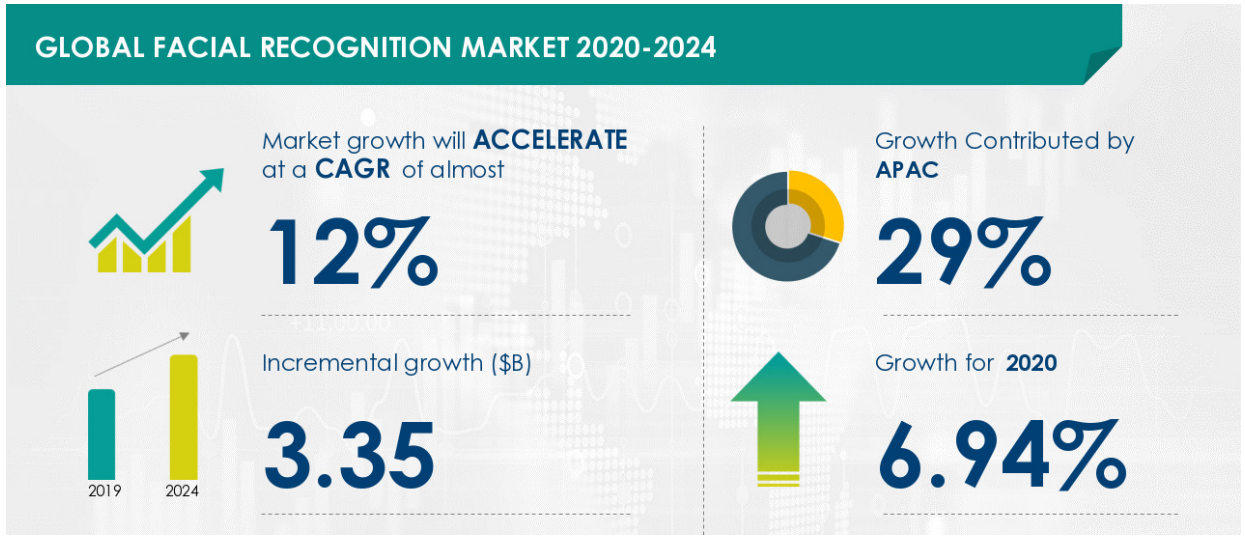
Motion supported display: Amazon Echo Show

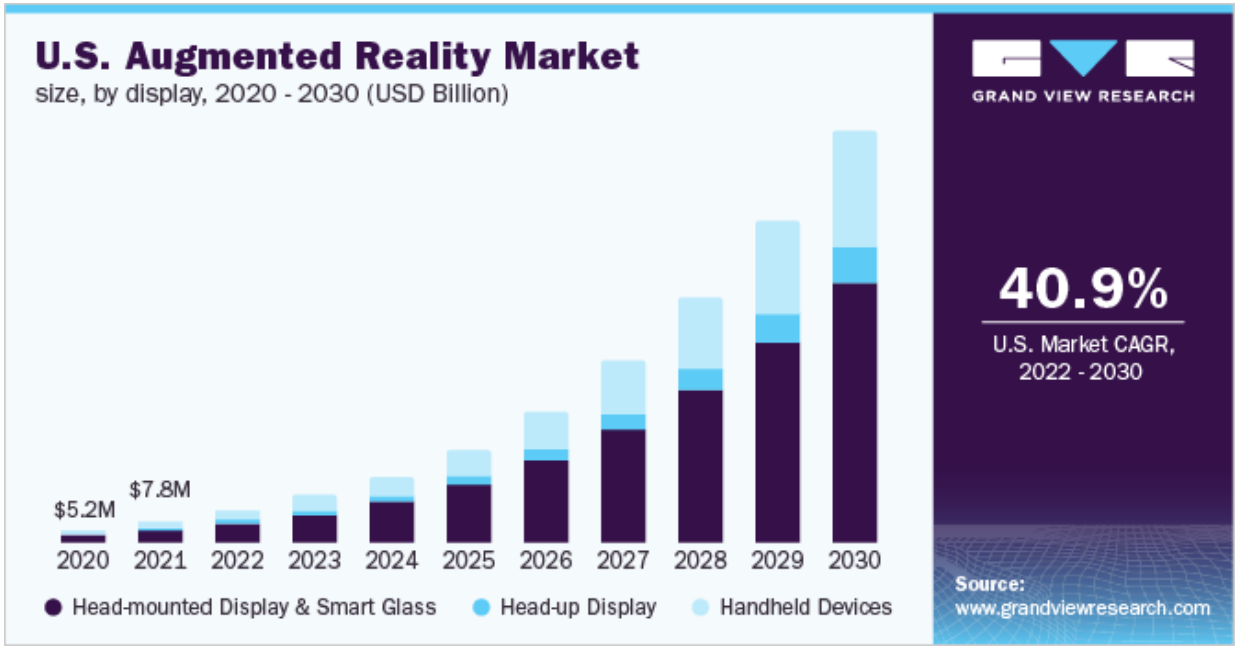


Object Identification in retail

VWW Applications Trends

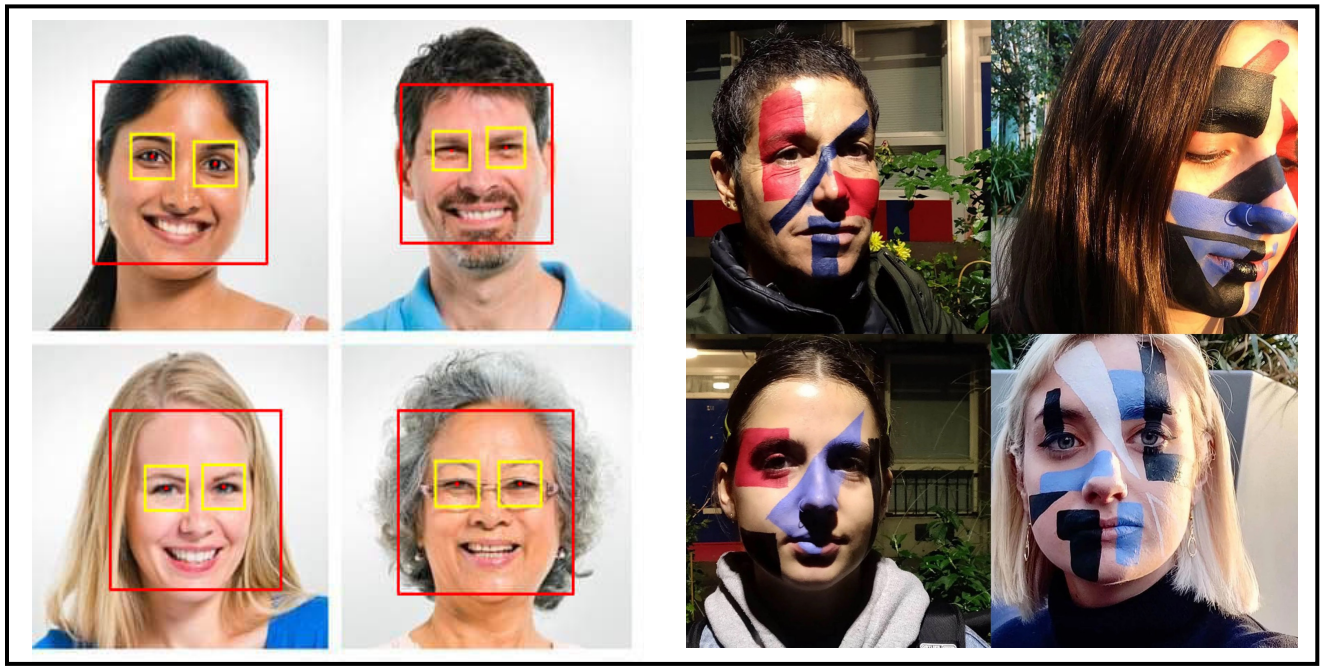
- The growth of **tiny vision** is attributed to the increased use of applications in healthcare, entertainment, consumer, and retail industries.
- Augmented reality has several use cases and is expected to grow as an industry in the future.





Challenges in Implementation – Adversarial attacks

- False Positives and Negatives - Adversarial attacks by modifying inputs can affect model prediction.



Adversarial attack by face painting

Glasses that can deceive face recognition models

Challenges in Implementation – Adversarial attacks

Input Image - Original

Objects identified by the Computer Vision System

Input image with noise

Objects identified by the Computer Vision System

Hendrik Metzen, Jan, et al. "Universal adversarial perturbations against semantic image segmentation." *Proceedings of the IEEE International Conference on Computer Vision*. 2017.

Challenges in Implementation – Embedded Trigger attacks

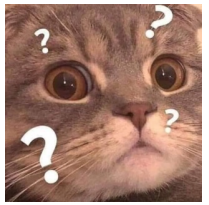
Poisoned training data

Training Phase

Features → Processed Data → Learning Algorithm → Model f → ***What ML algorithm learns*** = Dog

Decision Phase

Features → Processed Data → Model f → ***ML Prediction*** Dog

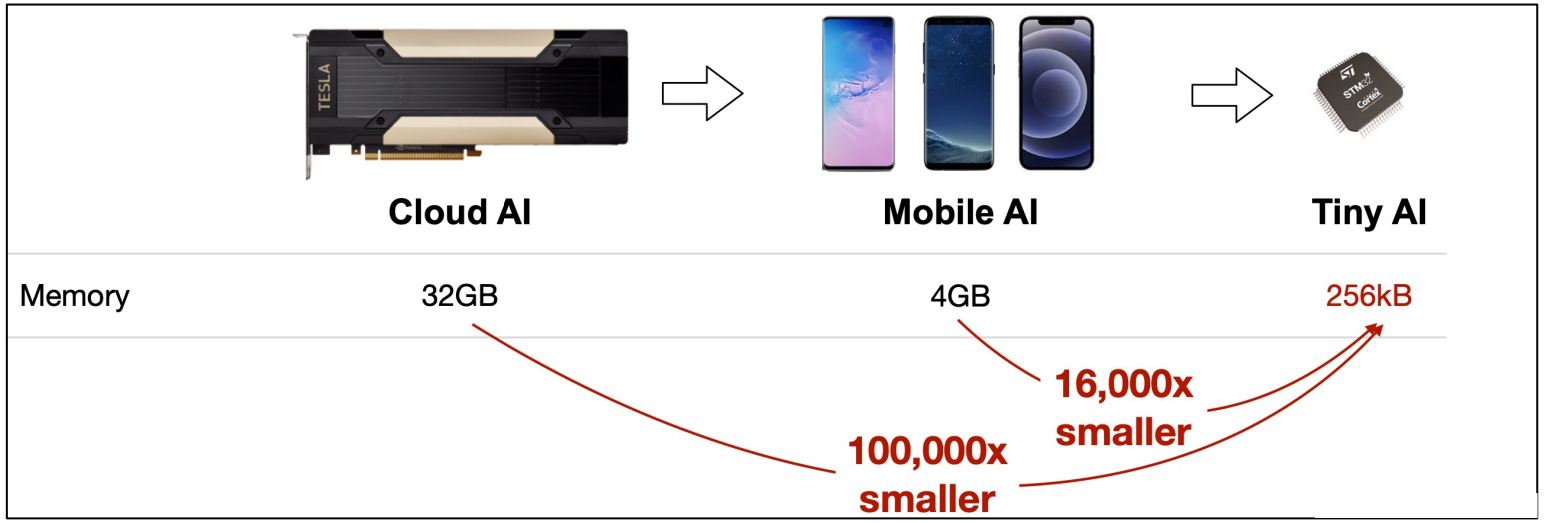


Challenges in Implementation – Model size

- Image data inputs are much larger compared to audio signal inputs and require larger memory and compute.
- Processing latency is higher since the image data is larger than audio data used in KWS.

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

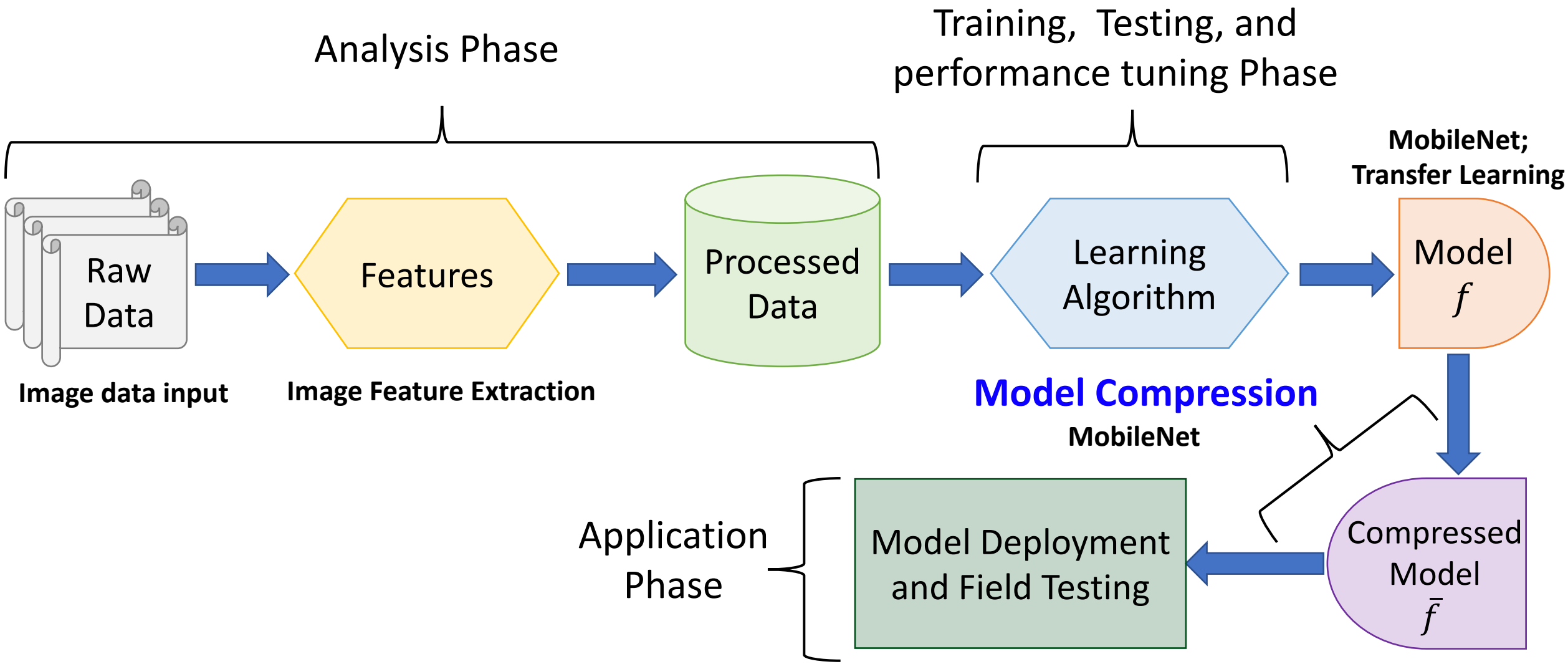
Memory requirements for large scale image models



Memory on tiny devices

Source: https://tinyml.seas.upenn.edu

Schematic View of TinyML Phases for VWW



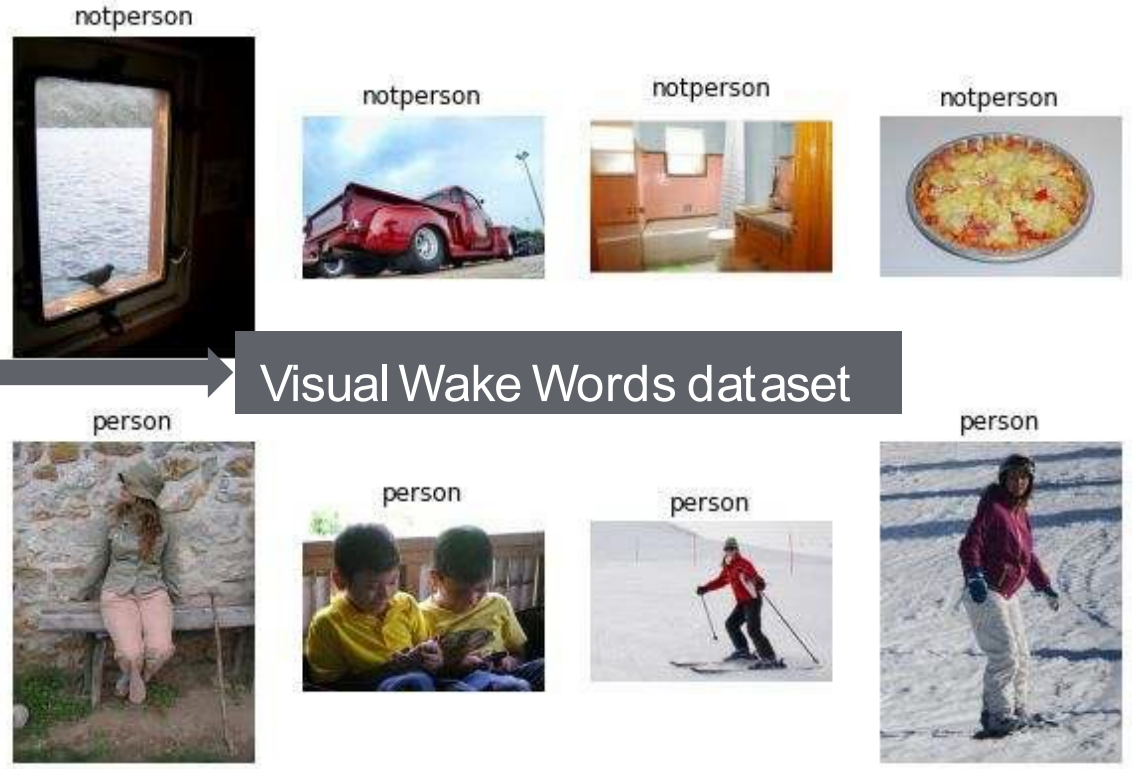
VWW Data Collection and Processing

Data collection is difficult!

- Dataset and collection process is limited and has bias
- Data privacy with images
- Small number of relevant images
- Large quantity of irrelevant images

VWW Dataset

Common Objects in COntext (COCO) Dataset



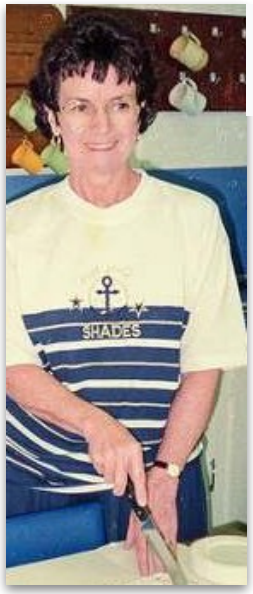
Visual Wake Words dataset

VWW Data Collection


```
"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
    },
```

VWW Data Collection

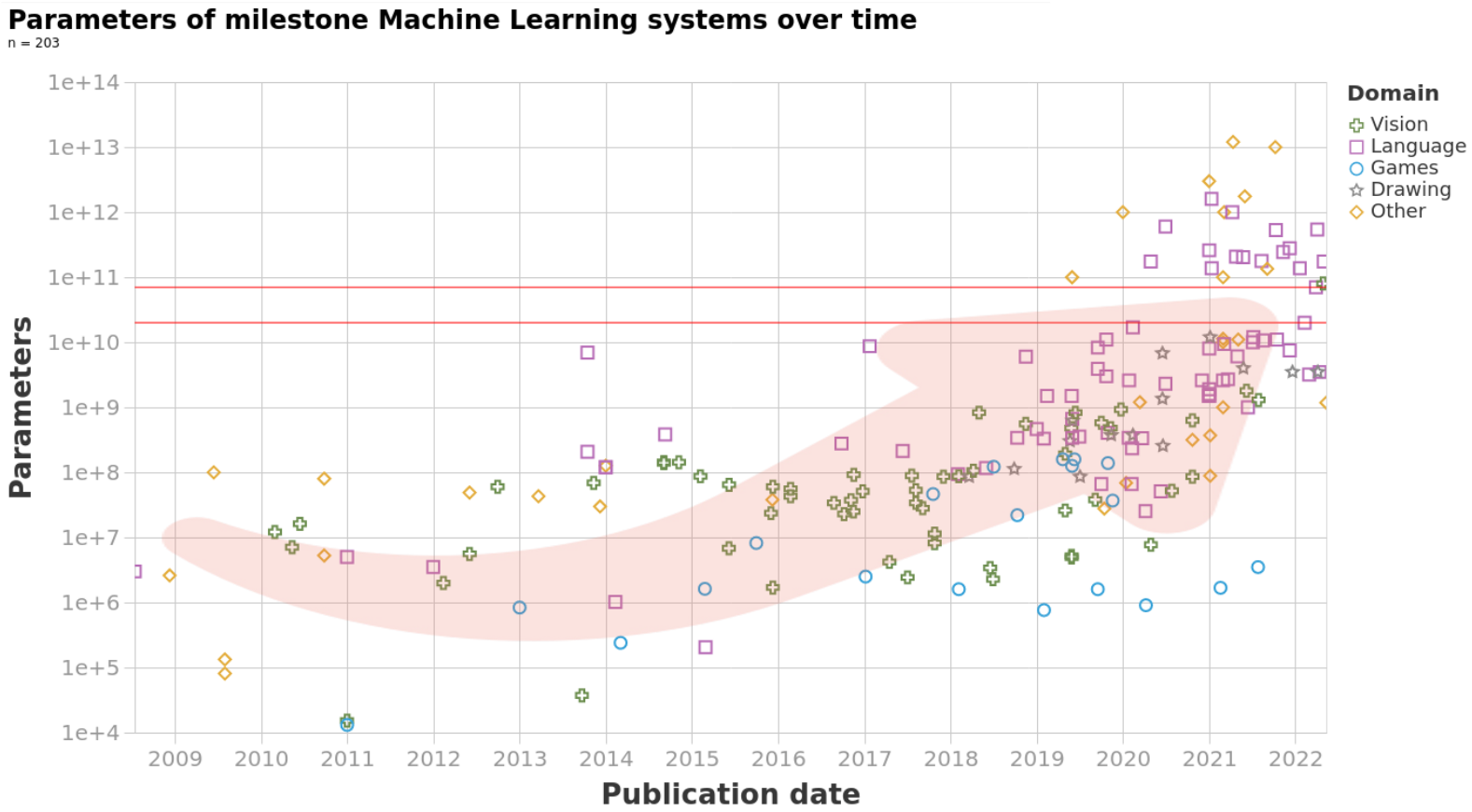
Label: **"person"**



Label: **"person"**

Model Evolution

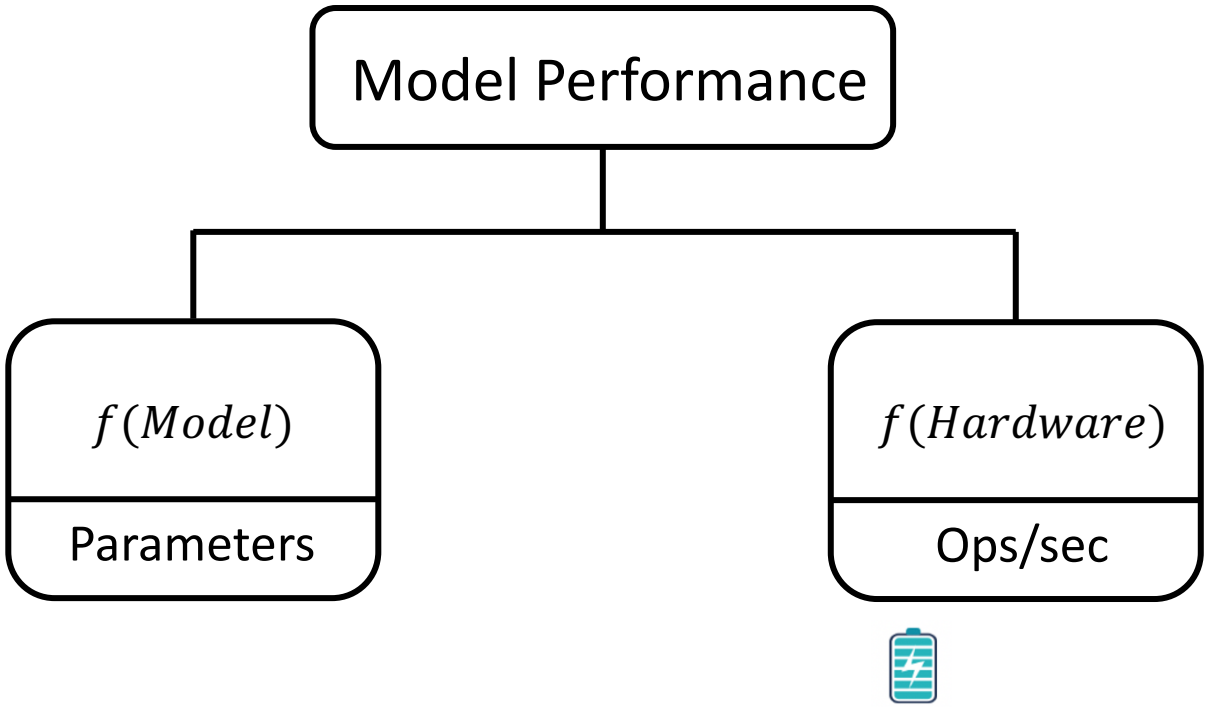
- The model parameters increase exponentially!
- We want model performance well while having fewer parameters



VWW Model Performance Evaluation

- Two fundamental constraints: The number of parameters and number of hardware operations

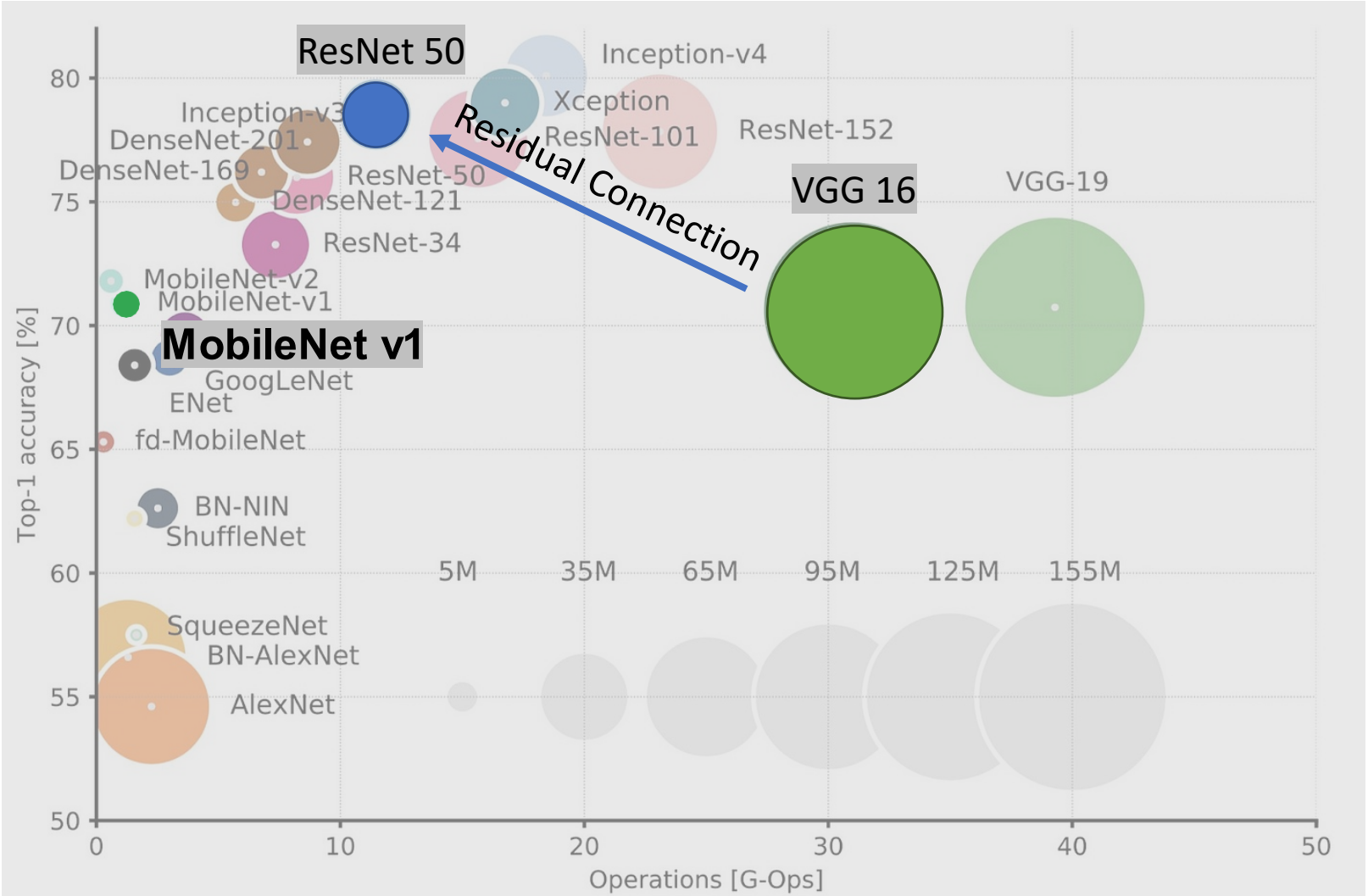
	Embedded Device	Tiny Device
Compute	1 GHz – 4 GHz	1 MHz – 400 MHz
Memory	512 MB – 64 GB	2 KB – 512 KB
Storage	64 GB – 4 TB	32 KB – 2 MB
Power	30 W – 100 W	150 µW – 23.5 mW



VWW Model Selection

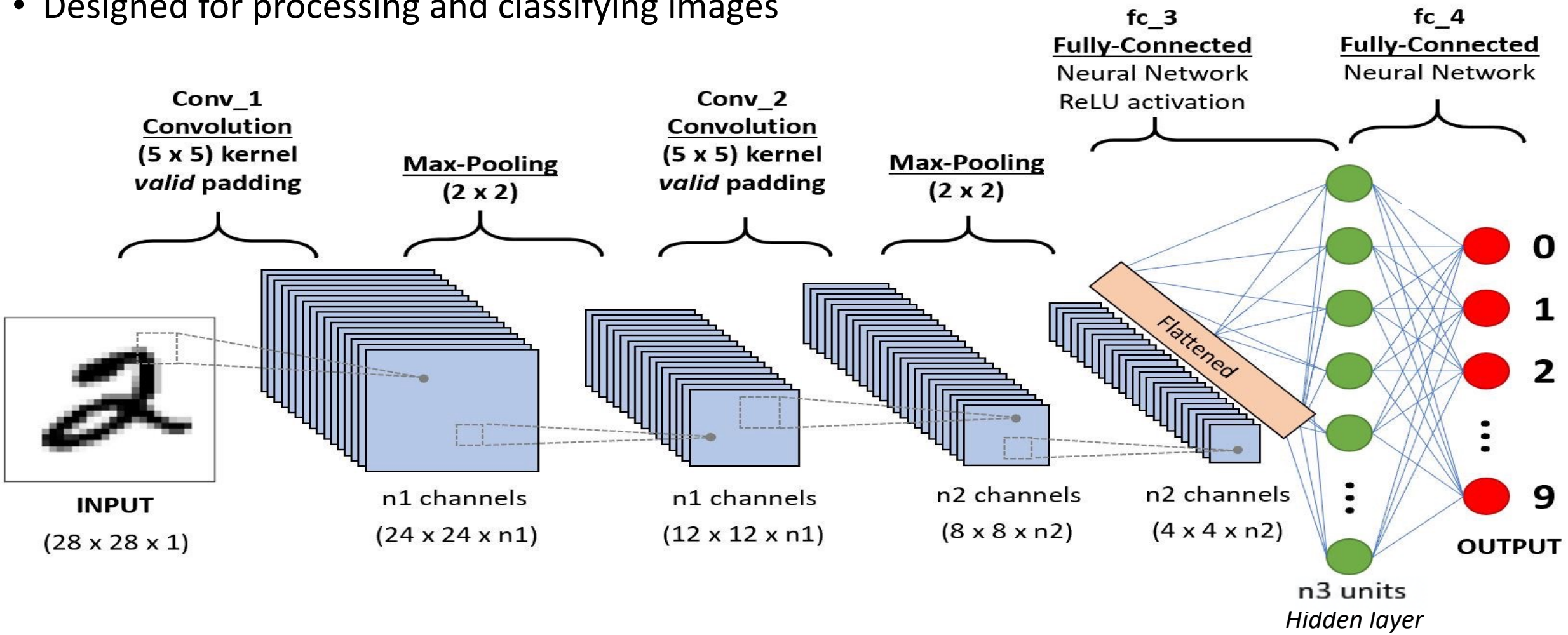
We want the models to run efficiently!

How to achieve this?



Standard Convolutional Neural Networks (CNNs)

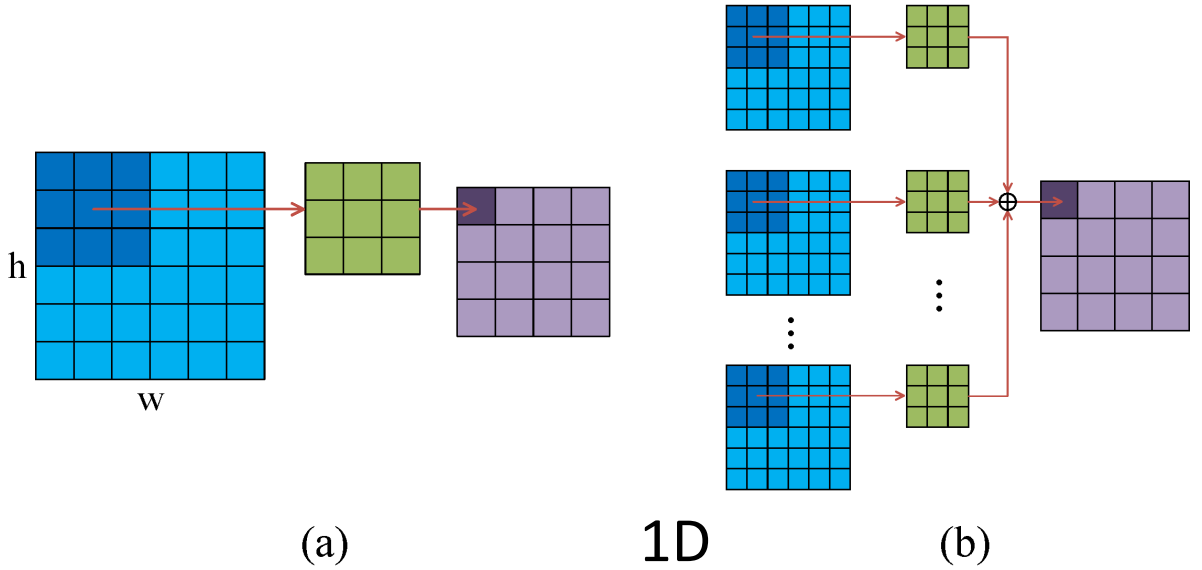
- Designed for processing and classifying images



Components of CNN: Input layer; Convolutional layers; Pooling layers; Fully connected NNs; Output layer

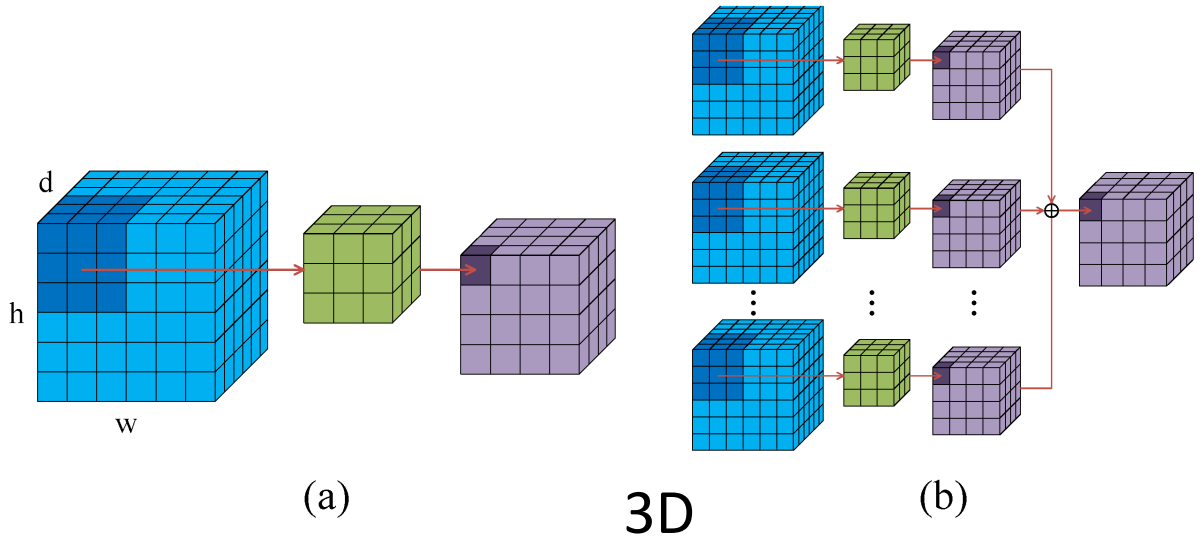
Standard Convolution

- Standard convolution slides over input data performing element-wise multiplication with the part of the input it is on, then summing the results into an output.
- Example: 1-D
- Input Feature Map
  - $6 \times 6 \times 1$
  - Width × Height × Channel
- Kernel (Filter)
  - $3 \times 3 \times 1$



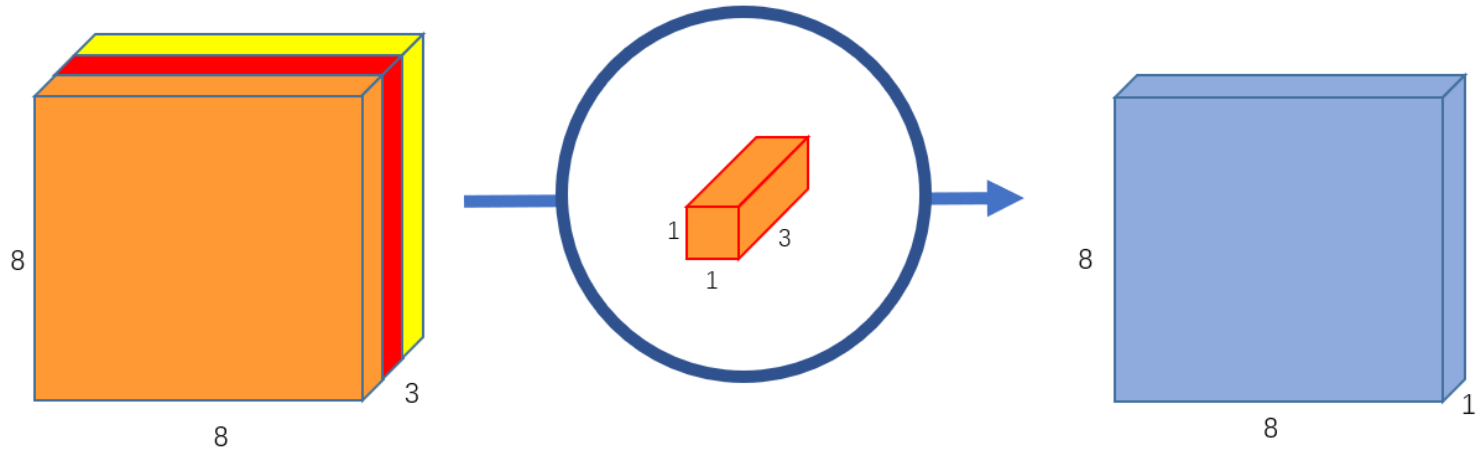
Standard Convolution

- Standard convolution slides over input data performing element-wise multiplication with the part of the input it is on, then summing the results into an output.
- Example: 3-D
- Input Feature Map
  - 6×6×6
  - Width × Height × Channel
- Kernel (Filter)
  - 3×3×3



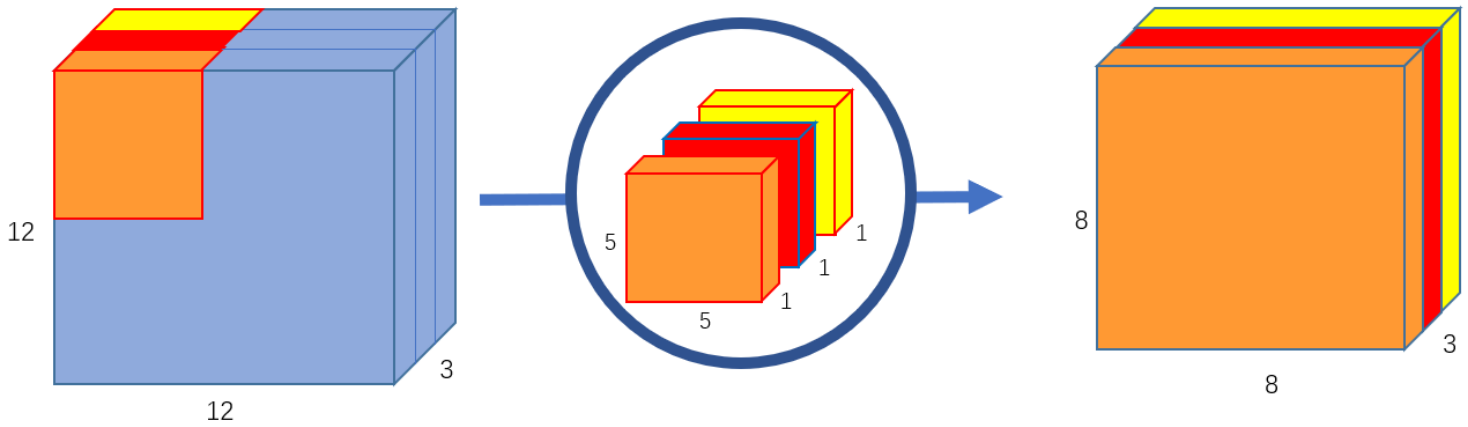
MobileNet – Pointwise Convolution

- Pointwise Convolution is a type of convolution that uses a **1x1 kernel**: a kernel that iterates through every single point.
- This kernel has a depth of however many channels the input image has.



MobileNet – Depthwise Convolution

- Depthwise Convolution is a type of convolution where we apply a single convolutional filter for each input channel.
- Depthwise convolutions keep each channel separate.

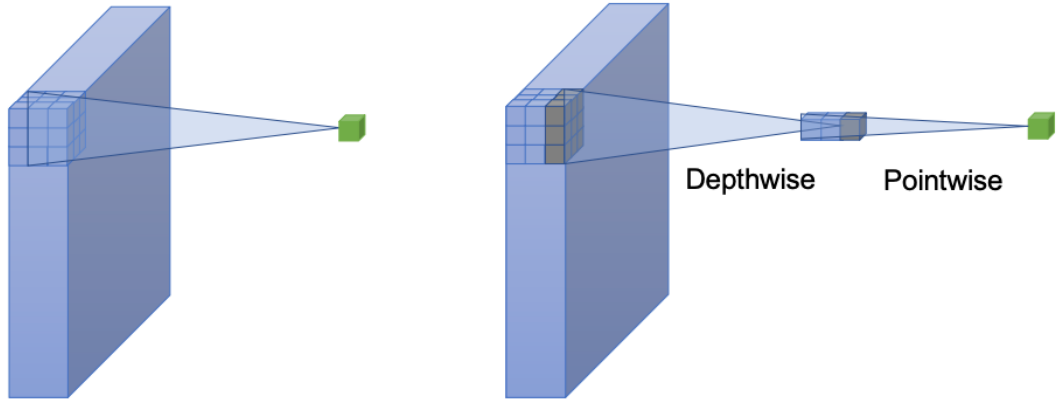


MobileNet – Cascading Convolution Techniques

Depthwise + Pointwise = Depthwise Separable Convolution

A Two-Step-Algorithm:

1. Depthwise convolution applies a single convolutional filter per each input channel
2. Pointwise convolution is then used to create a linear combination of the output of the depthwise convolution.



Standard Convolution

Depthwise Separable Convolution

MobileNet – Cascading Convolution Techniques

Depthwise + Pointwise = Depthwise Separable Convolution

Benefits?

- **Far fewer multiplications** than standard method (especially when using many filters)

$$\frac{\text{Depthwise Separable}}{\text{Standard Conv}} = \frac{1}{N} + \frac{1}{D_K^2}$$

$\frac{1}{N}$: # of filters

$\frac{1}{D_K^2}$: Filter (kernel) Dimensions

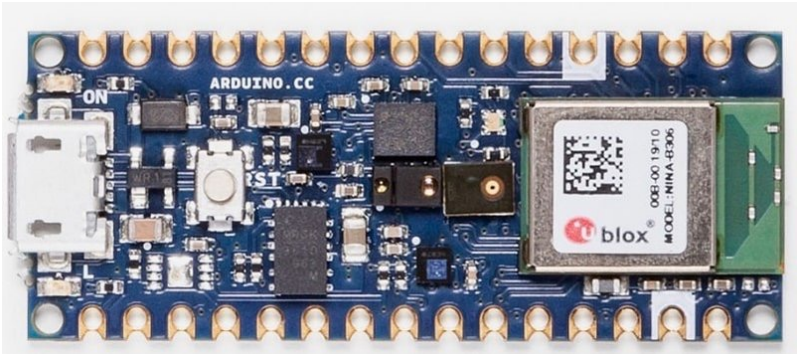
- As an example, consider $N = 100$ and $D_K = 512$, the ratio ≈ 0.01!

MobileNet - Model Size Constraints

However, the model size is still too large for our board

Model	Size	Top-1 Accuracy
MobileNet v1	16 MB	0.713

Good for mobile devices with GB of RAM, but 64x microcontroller RAM



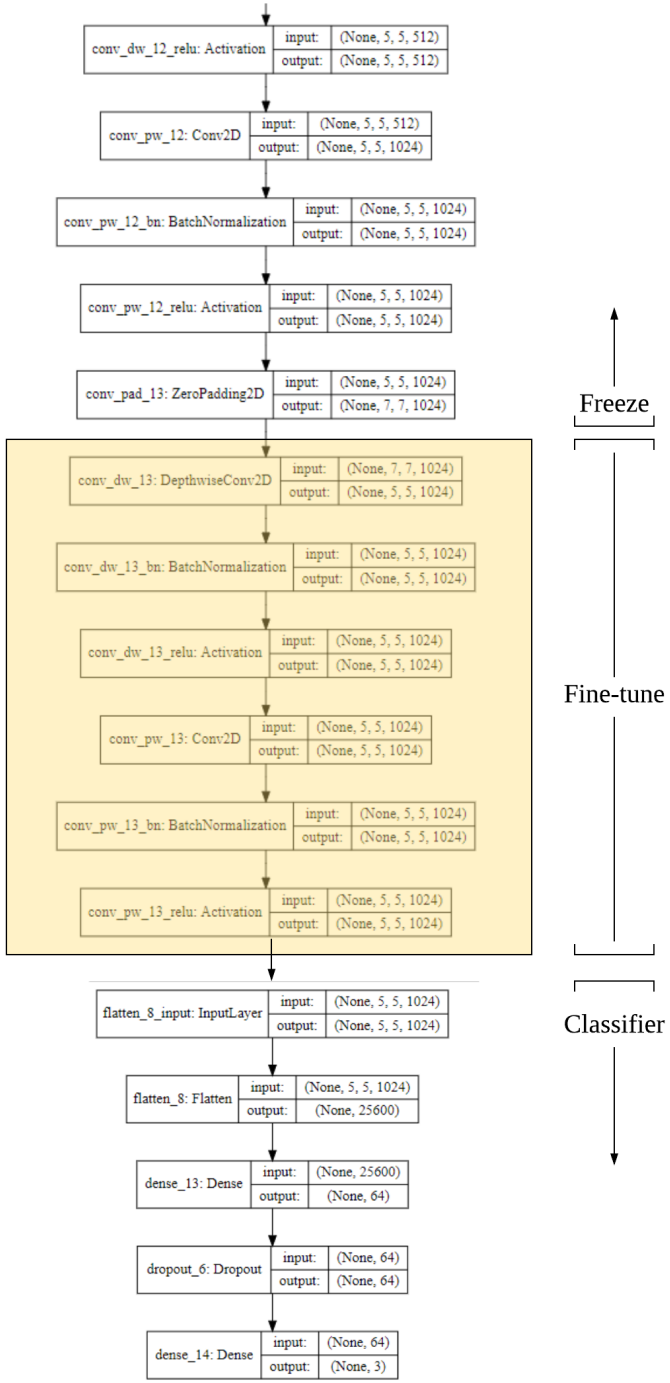
Our board only has **256 KB** RAM !

- The size of the model can be reduced further by **width multiplier**, $\alpha \rightarrow (0,1]$
- $\alpha = 1$: baseline MobileNet; $\alpha < 1$: reduced MobileNet.

Mobile Net and Transfer Learning

MobileNet is unique in that its convolutional block is a Depthwise-separable block.

- This speeds up the network and reduces the number of parameters it needs to store
- In the last step, combine Depthwise Conv2D and Conv2D blocks to perform the standard convolution operation.

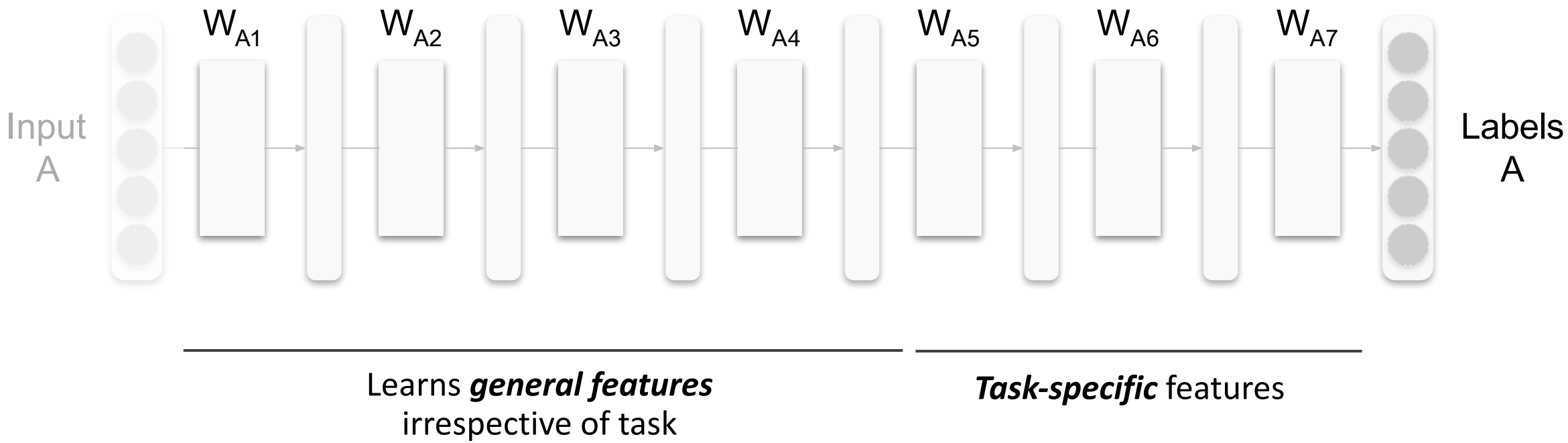


Transfer Learning

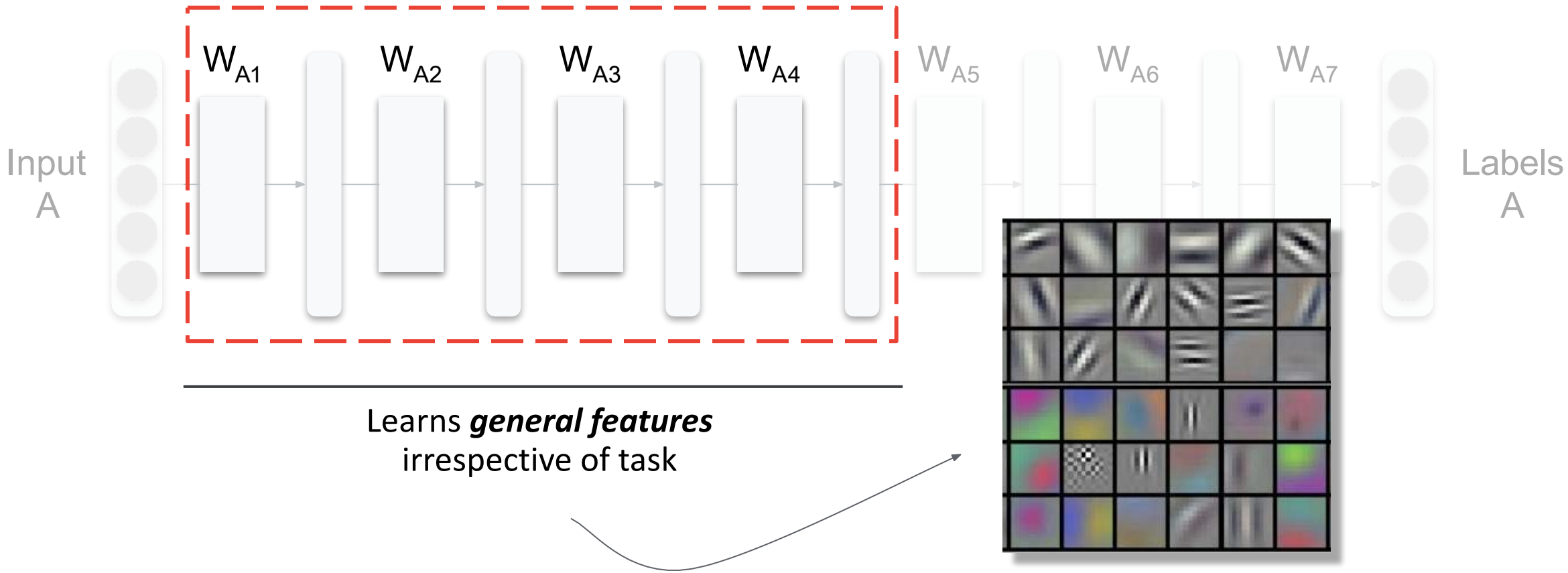
- Transfer Learning: a model trained on one task is re-purposed on another related task
- A model trained on one task will **have learned features** that are useful for other tasks
- E.g., a model trained to recognize cars could be used to recognize trucks as well

Transfer Learning

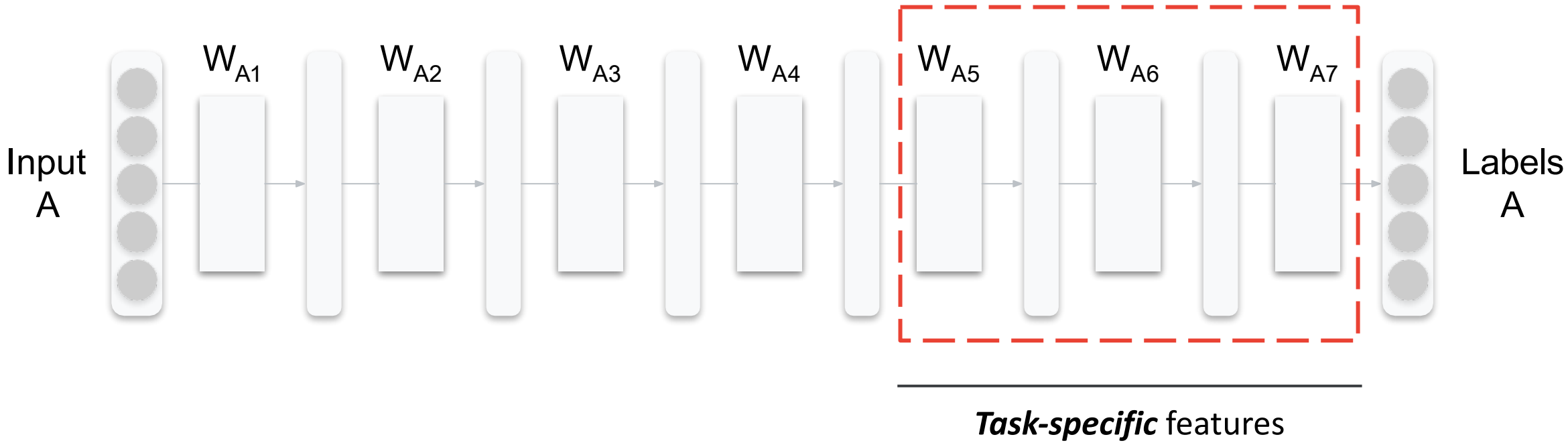
- Do we have to train the model from scratch every single time?



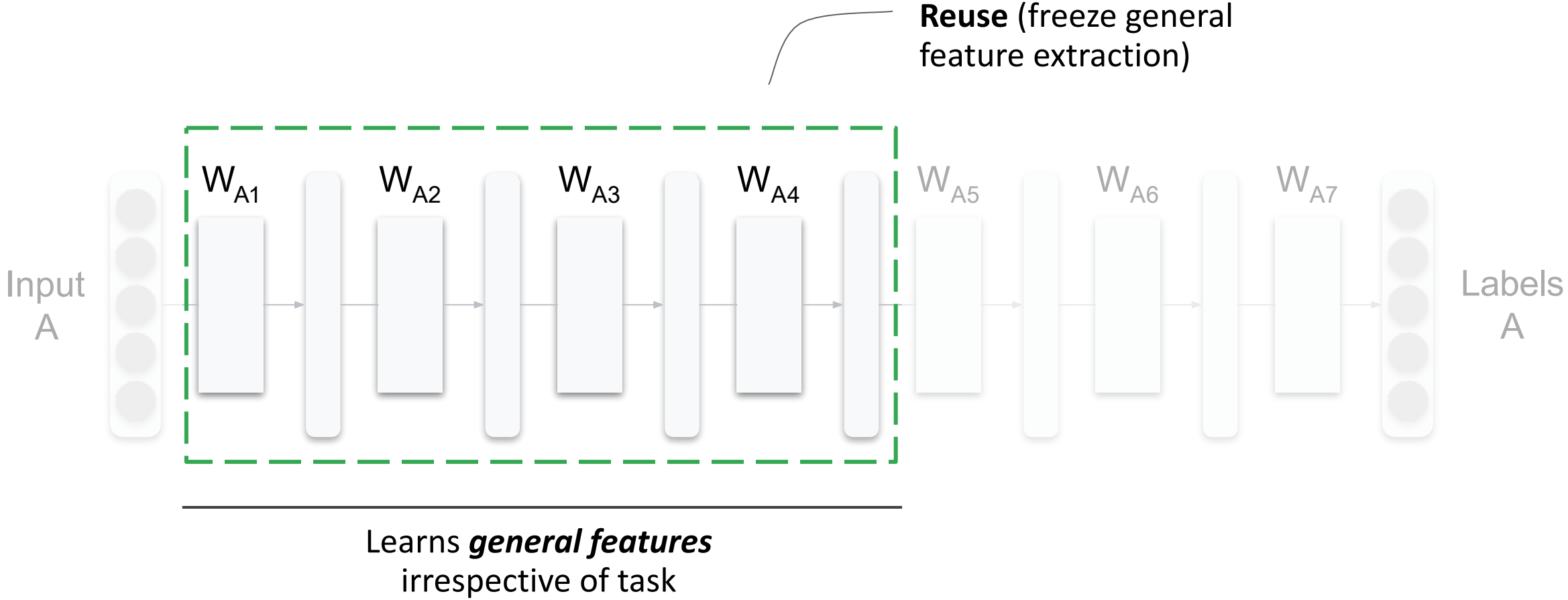
Transfer Learning



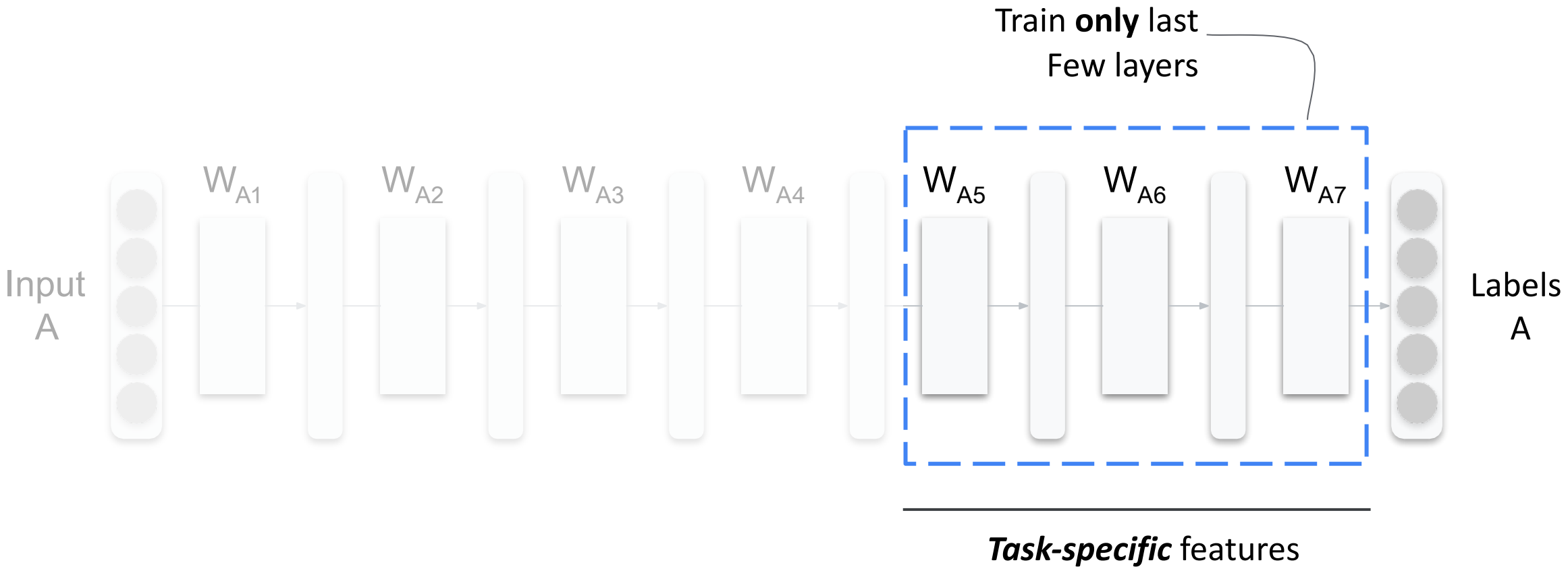
Transfer Learning



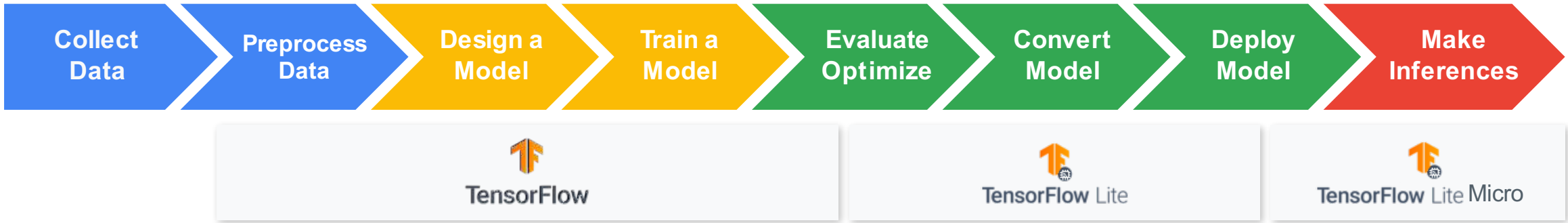
Transfer Learning



Transfer Learning



VWW Workflow



Week #4 Lab

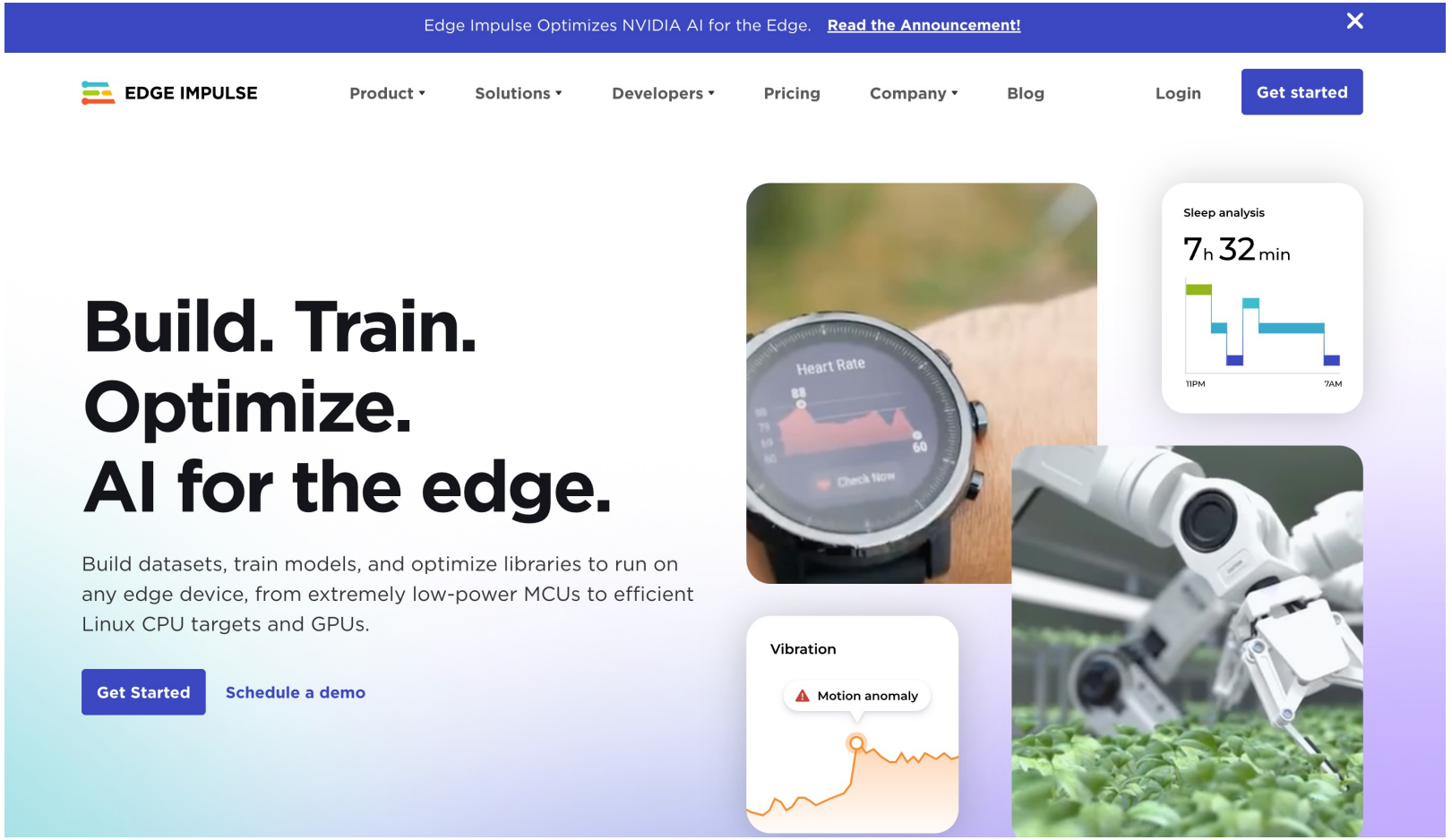
- Both Software and Hardware Lab!
- Develop a MobileNet model for Mask-Detection using **Edge Impulse**
- Using Transfer Learning to Train a Mask-Detection Model
- Compress MobileNet model for TinyML
- Deploy mask detection model in the Arduino board

Lab 4 – Introduction to Edge Impulse

Edge Impulse

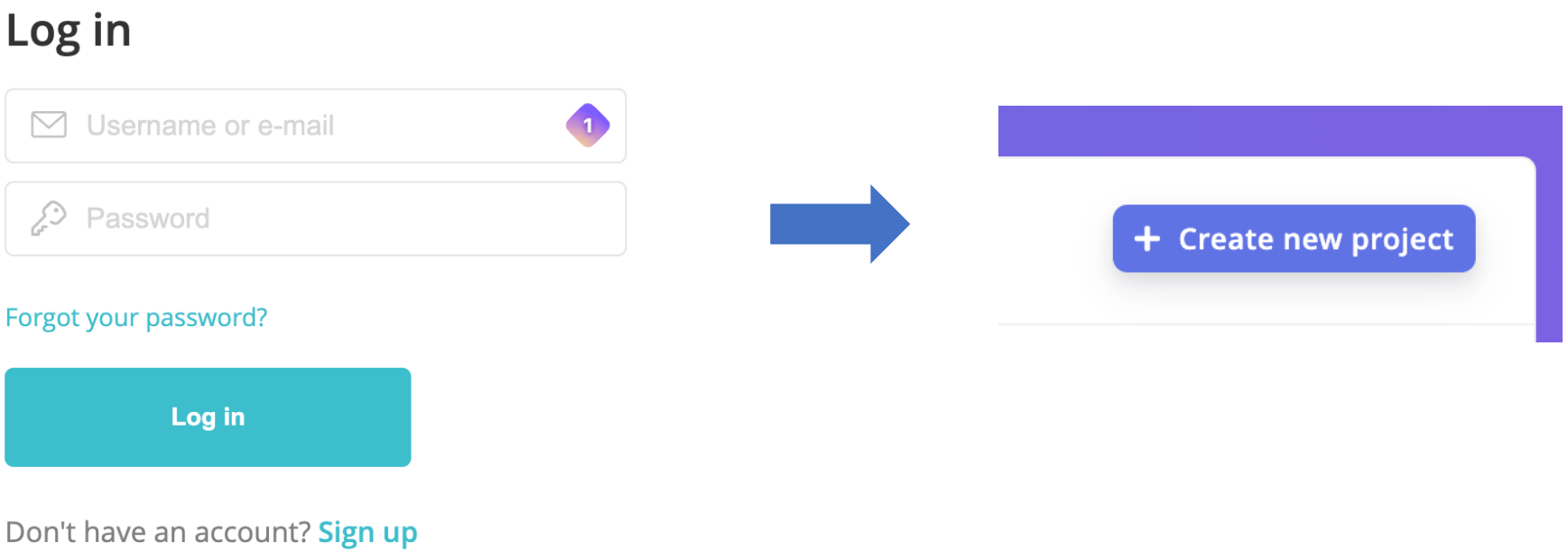
- **No Coding Required**
- **Edge Computing**
- **Support Various Sensors**

Start thinking about your project now !



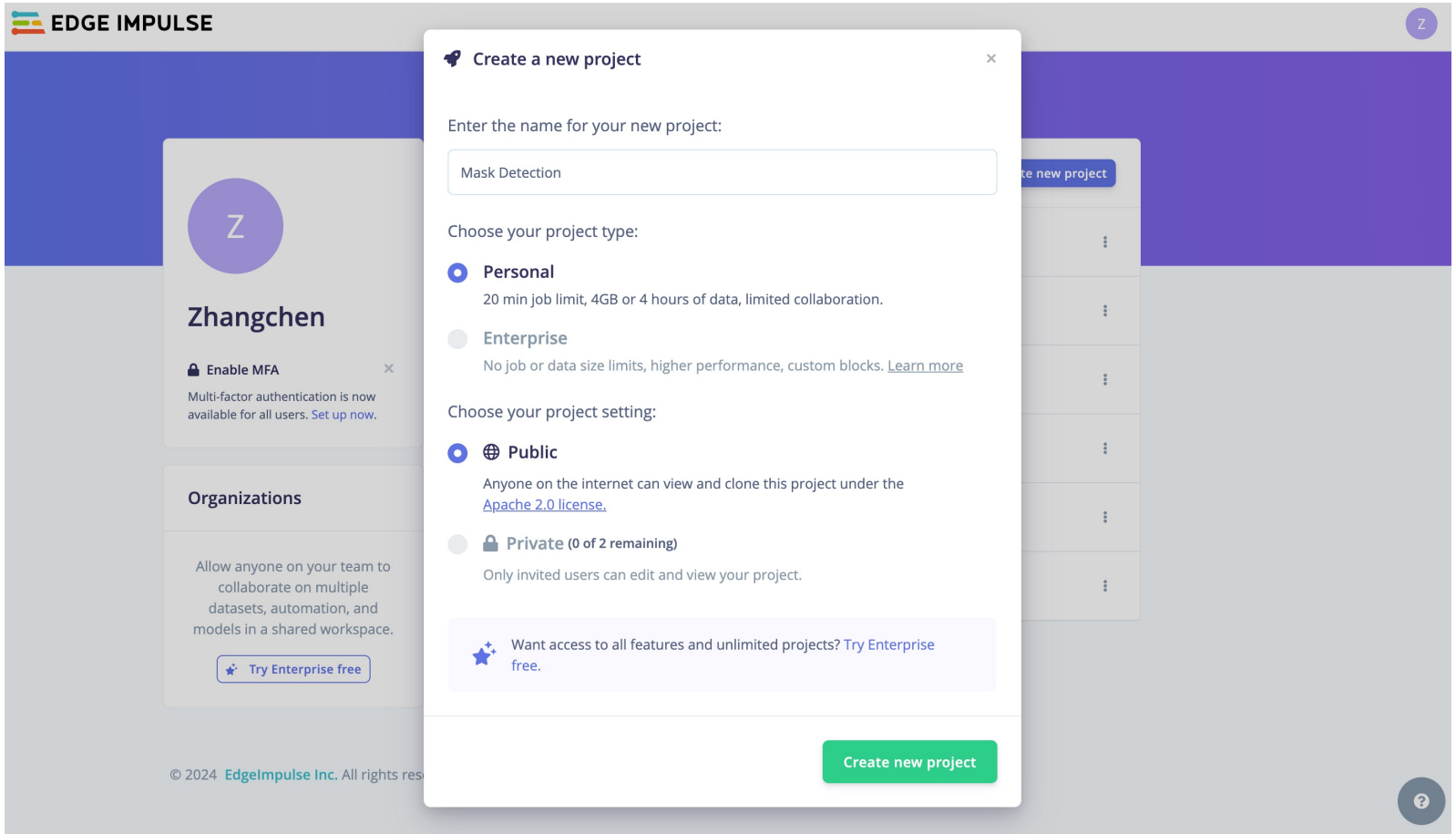
Lab 4 - Software

1. Sign up and create a new project



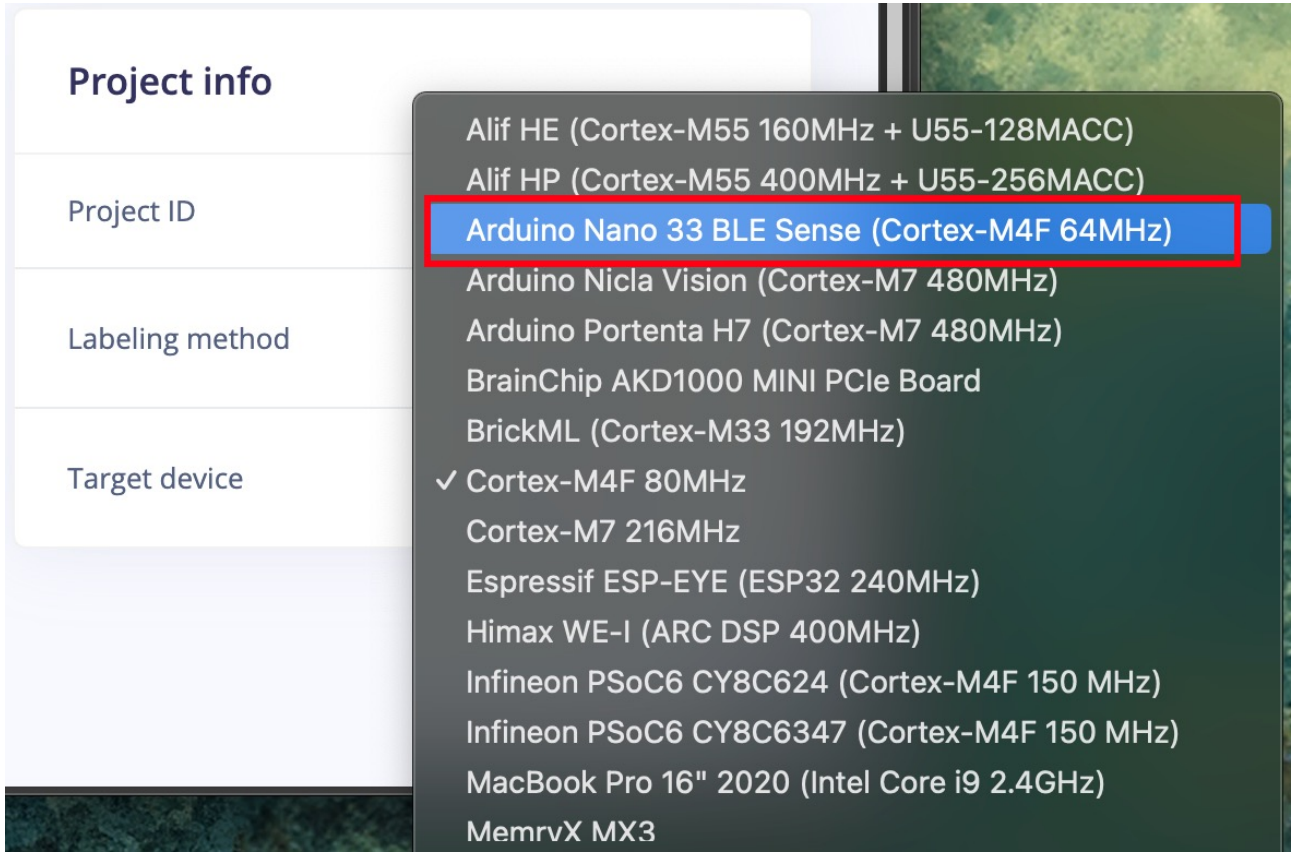
Lab 4 - Software

1. Sign up and create a new project



Lab 4 - Software

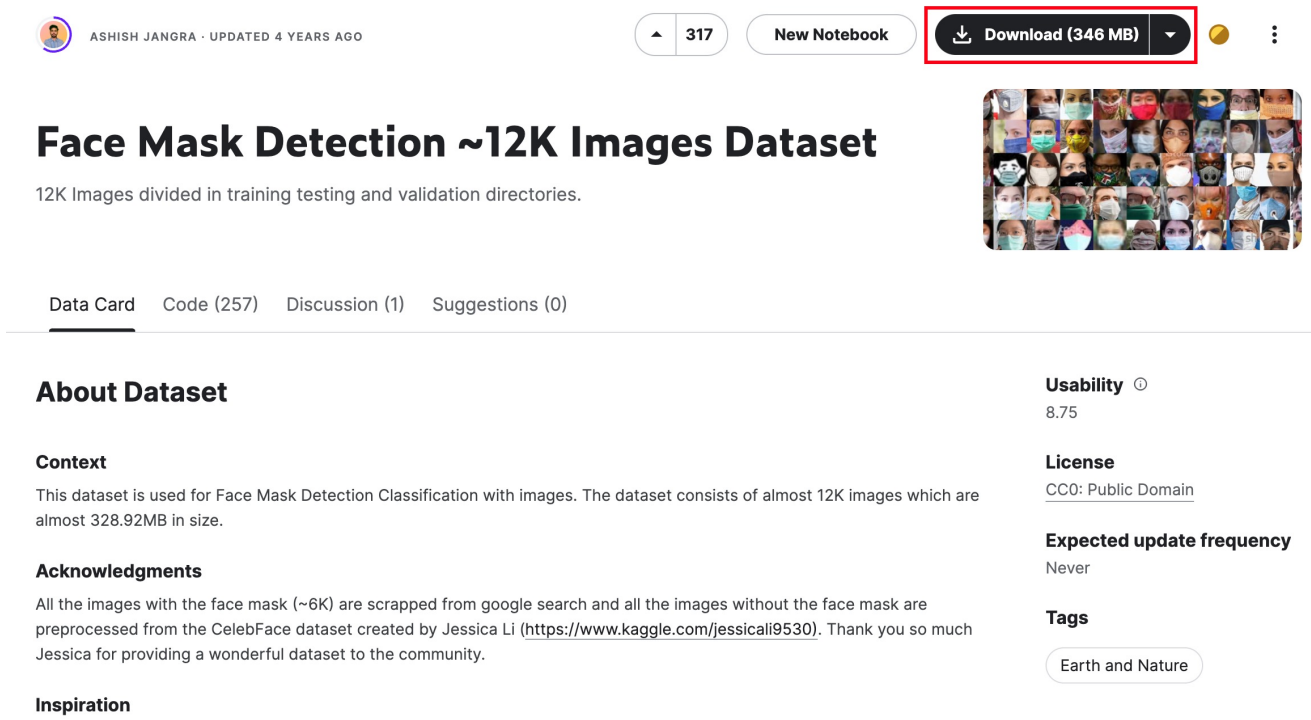
2. Scroll down and choose the target device first!



Lab 4 - Software

3. Upload the dataset and label it!

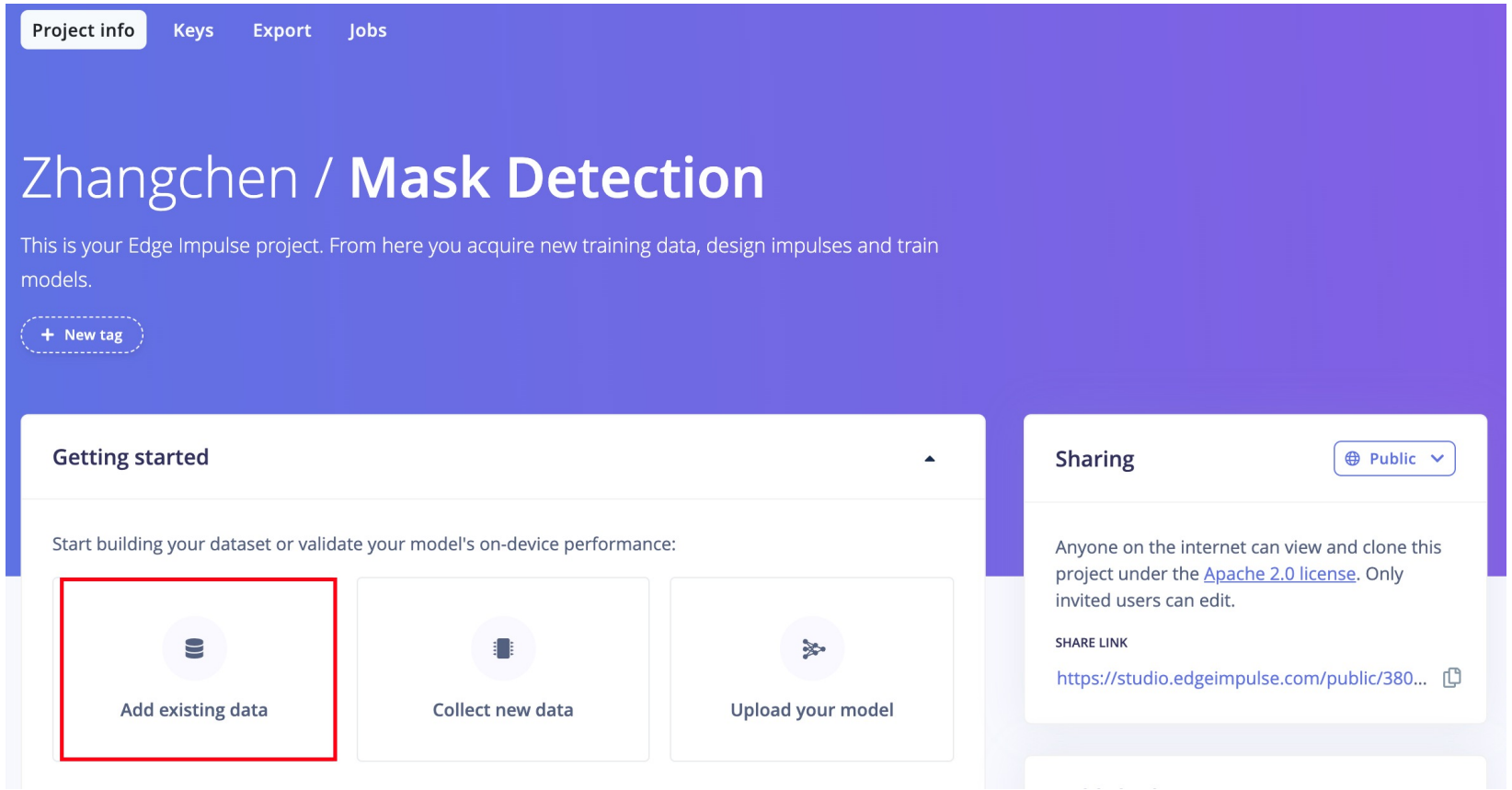
We obtain a publicly available dataset from Kaggle
https://www.kaggle.com/datasets/ashishjangra27/face-mask-12k-images-dataset



Lab 4 - Software

3. Upload the dataset and label it!

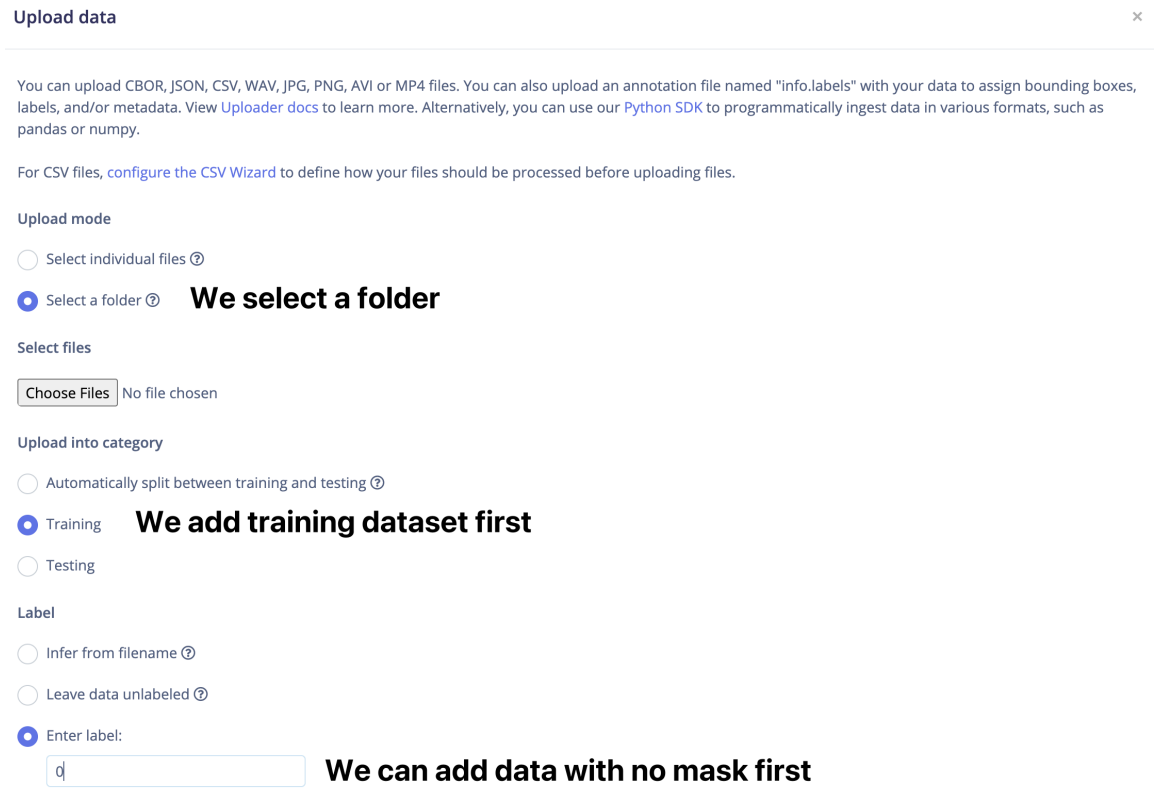
In Edge Impulse, click: Add existing data

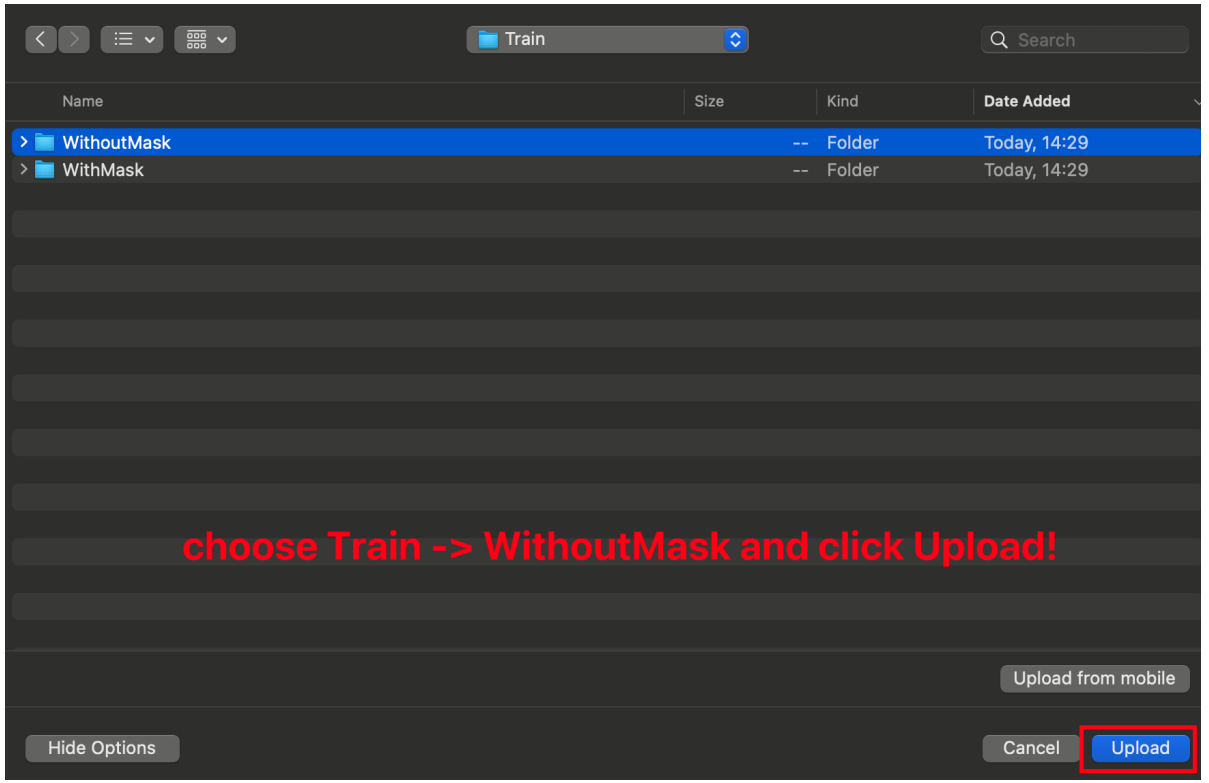


Lab 4 - Software

3. Upload the dataset and label it!

In Edge Impulse, click: Add existing data

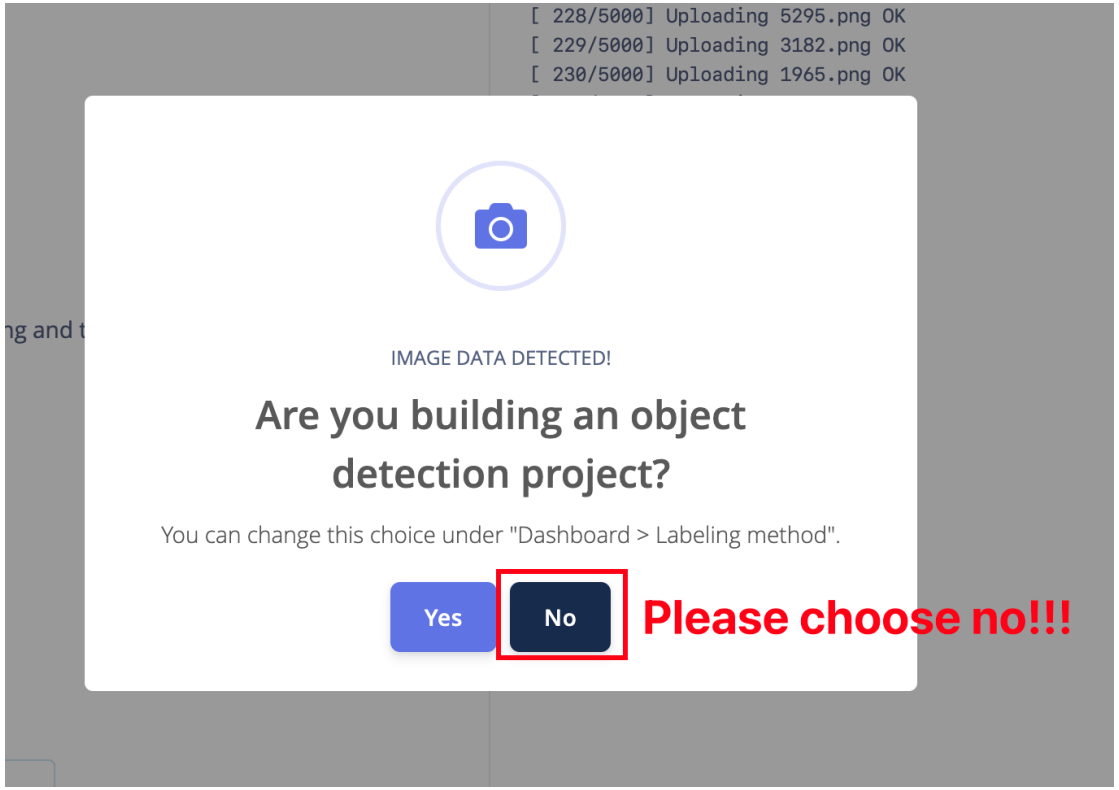




Lab 4 - Software

3. Upload the dataset and label it!

In Edge Impulse, click: Add existing data



Desired output:

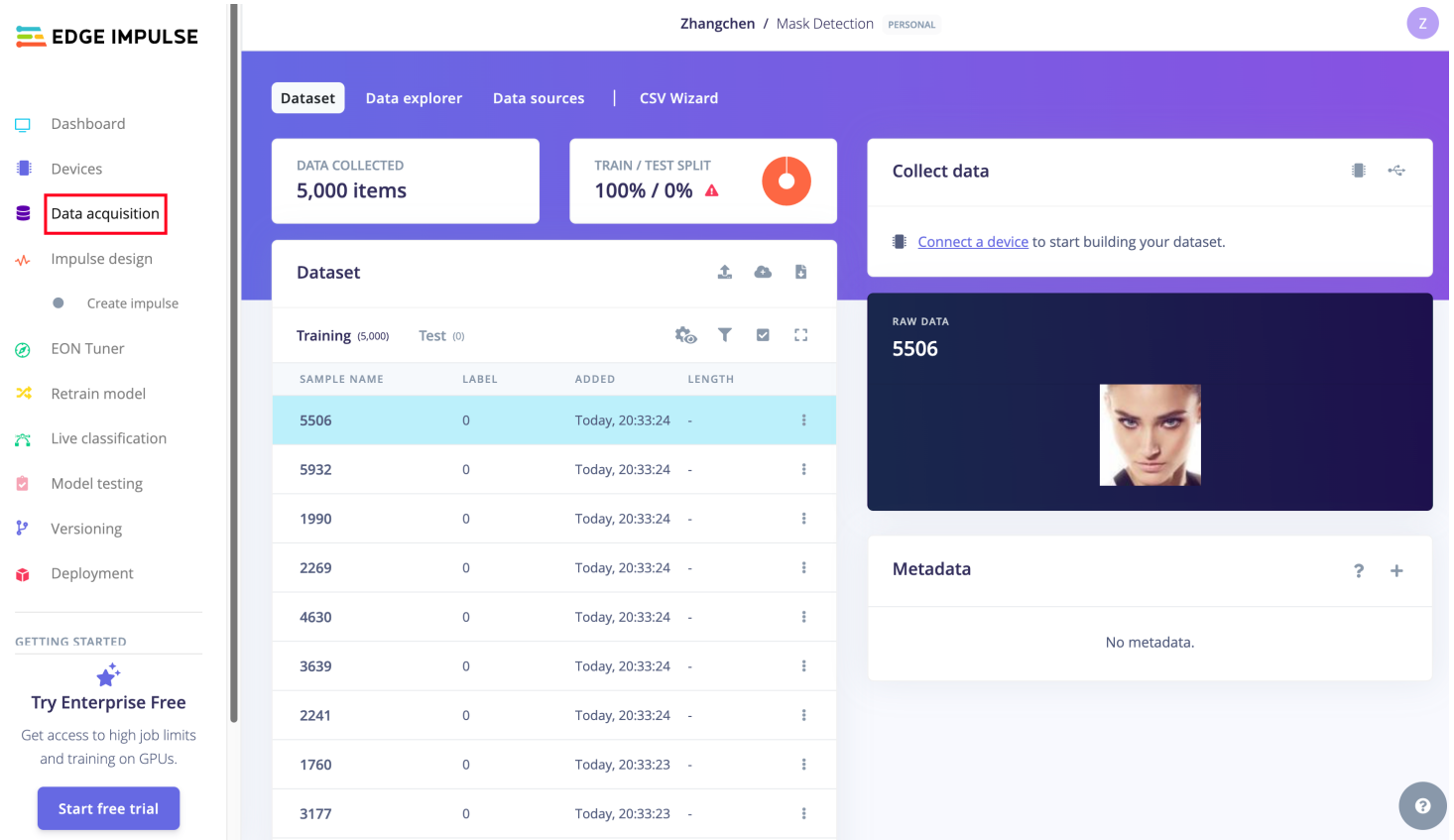
```
Done. Files uploaded successful: 5000. Files that failed to upload: 0.

Job completed
```

Lab 4 - Software

3. Upload the dataset and label it!

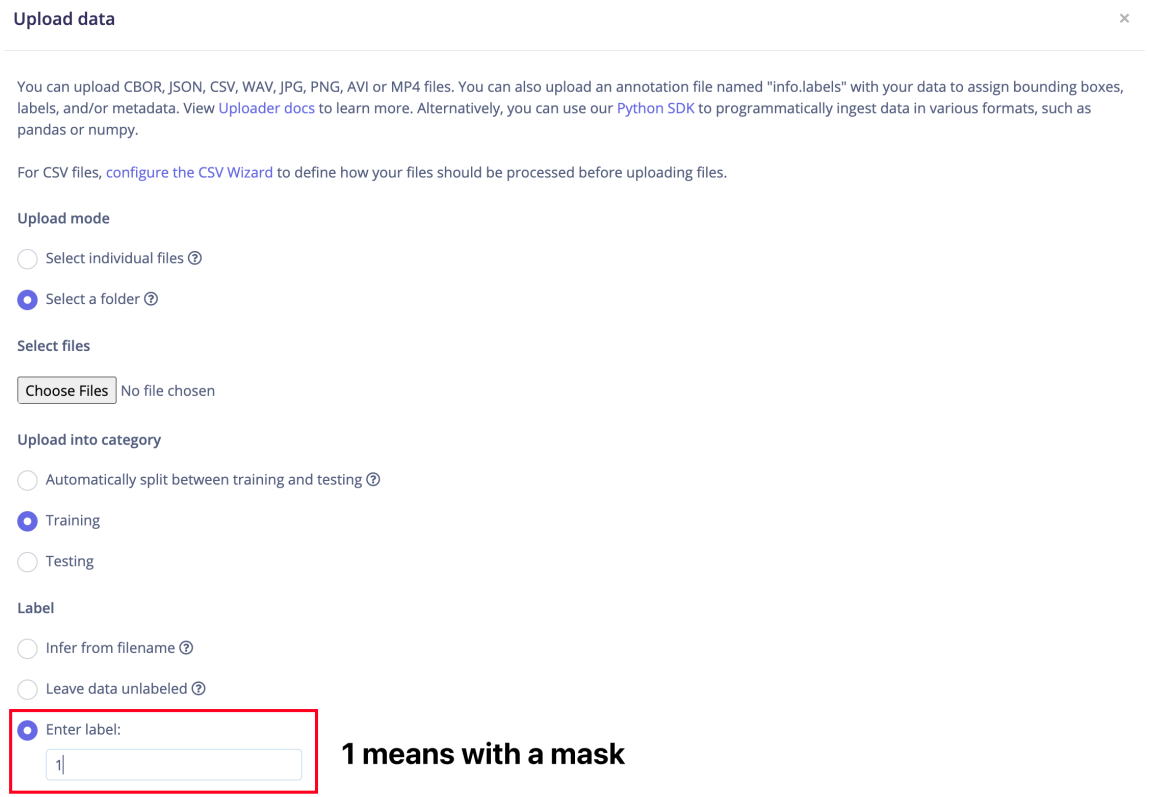
Now you can see some datasets in the Data Acquisition Sidebar



Lab 4 - Software

3. Upload the dataset and label it!

But we only have training data with label = 0 now, so we continue uploading other parts!



1 means with a mask

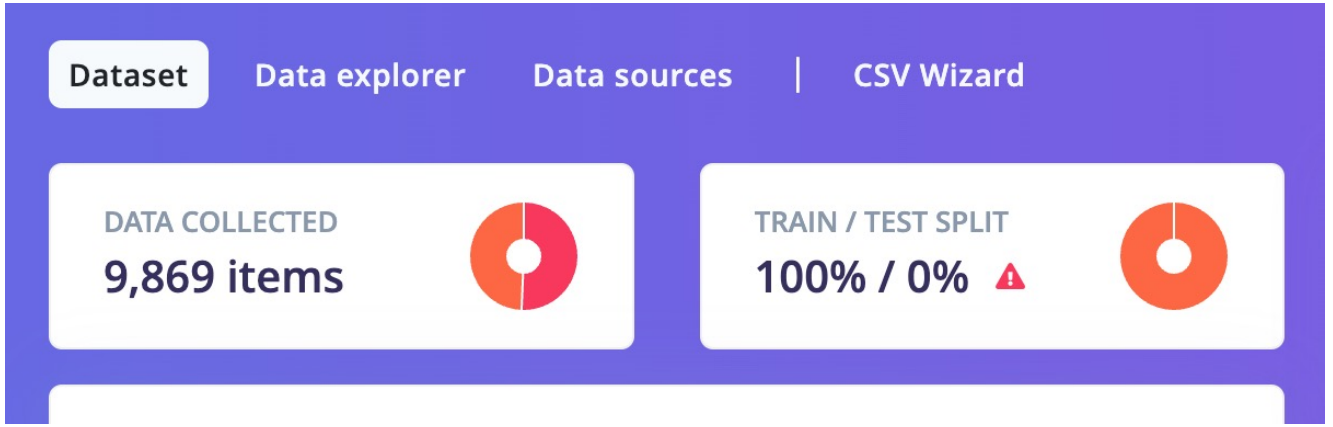
```
Done. Files uploaded successful: 4869. Files that failed to upload: 131.

Job completed
```

Lab 4 - Software

3. Upload the dataset and label it!

Now we should have ~ 10k data, which is already enough. But:

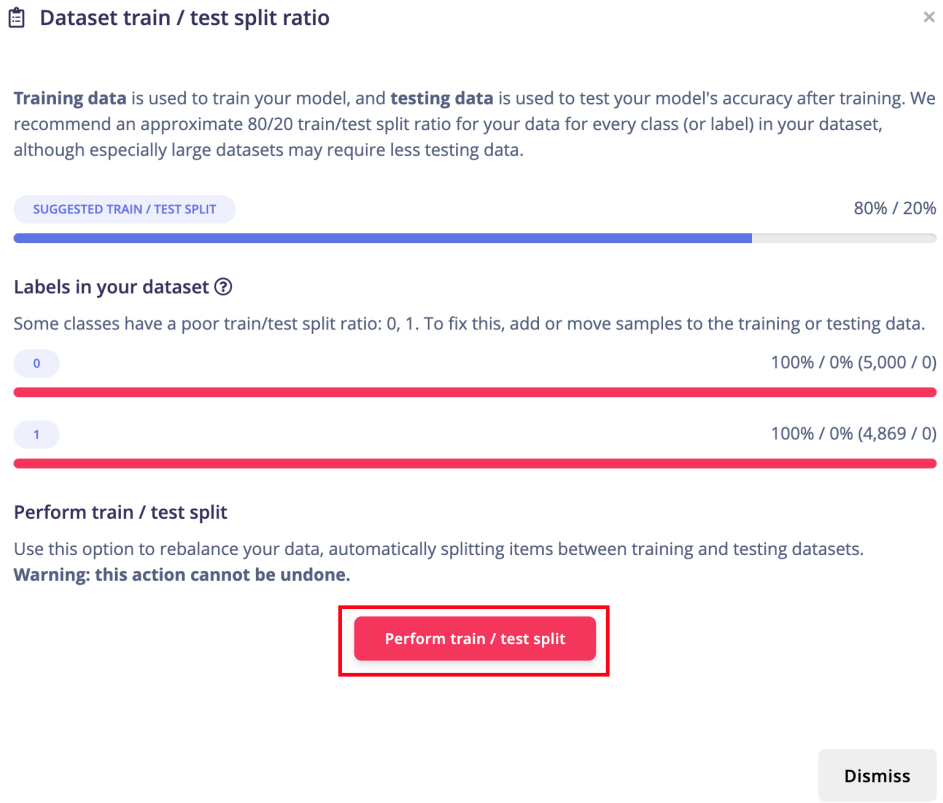


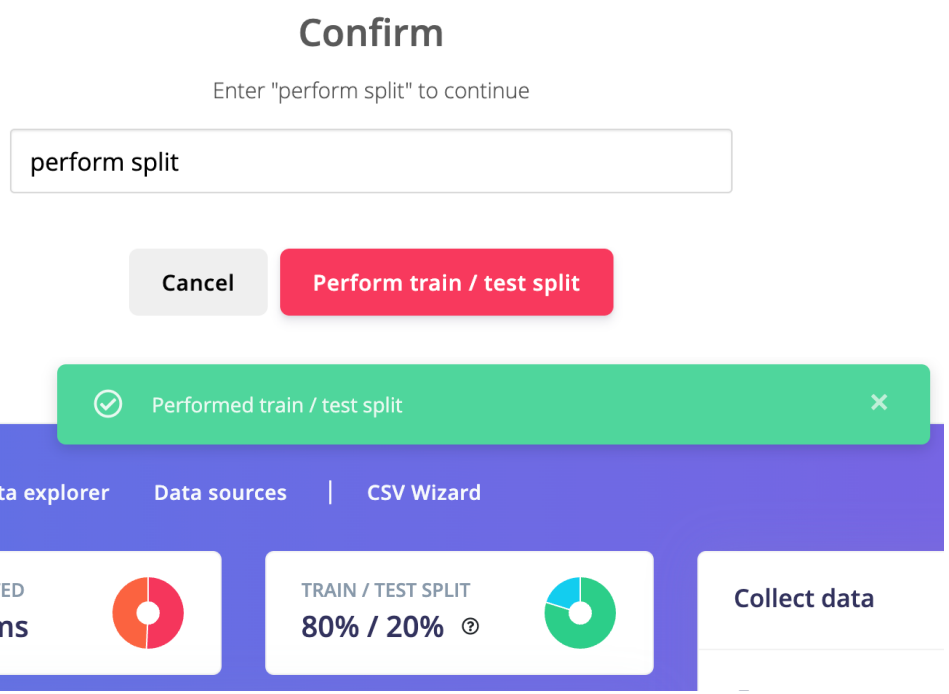
1. Upload testing data as before
2. Click TRAIN / TEST SPLIT

Lab 4 - Software

3. Upload the dataset and label it!

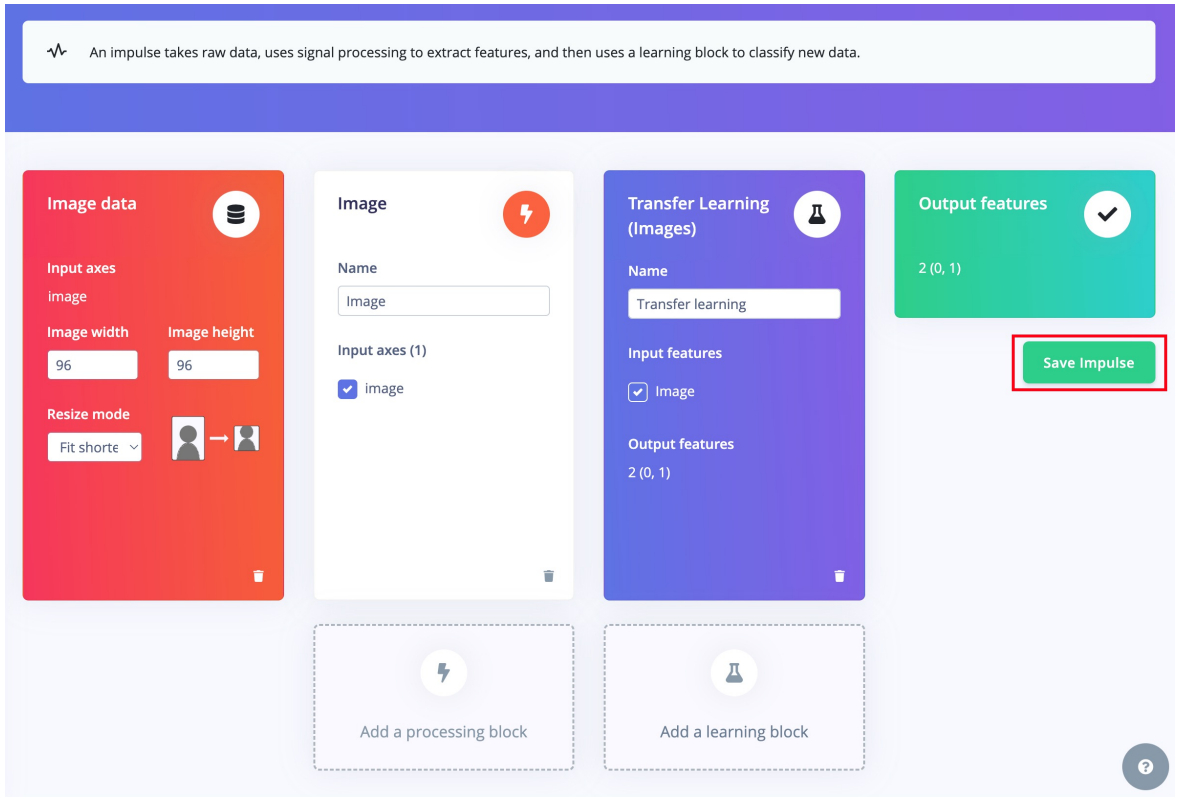
Click TRAIN / TEST SPLIT (80% vs 20%)





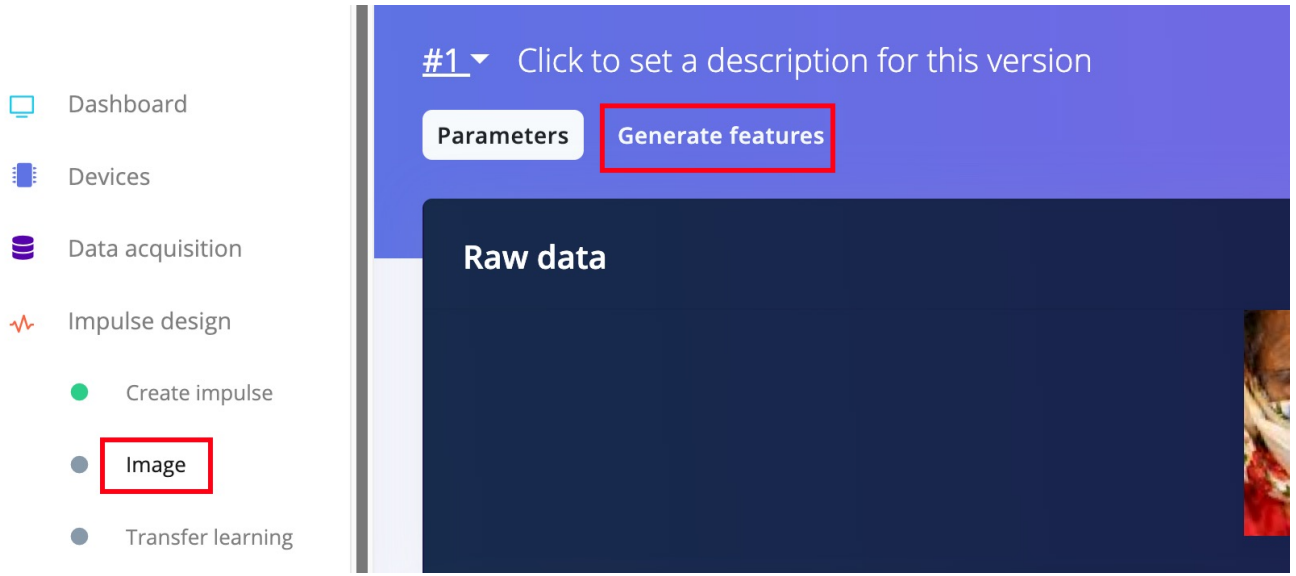
Lab 4 - Software

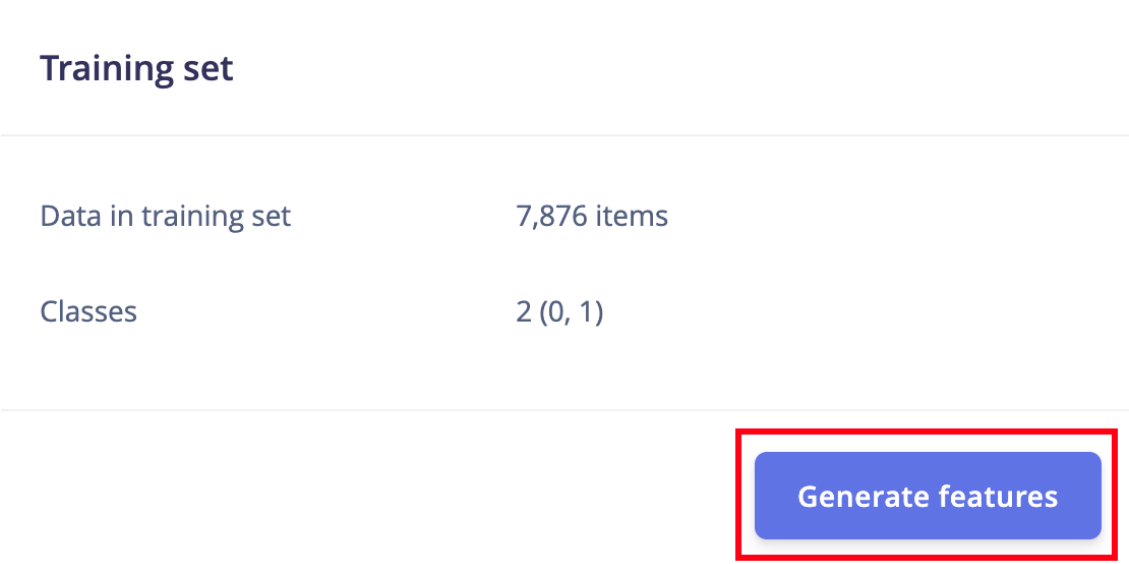
4. Create a training pipeline (aka Impulse Design)



Lab 4 - Software

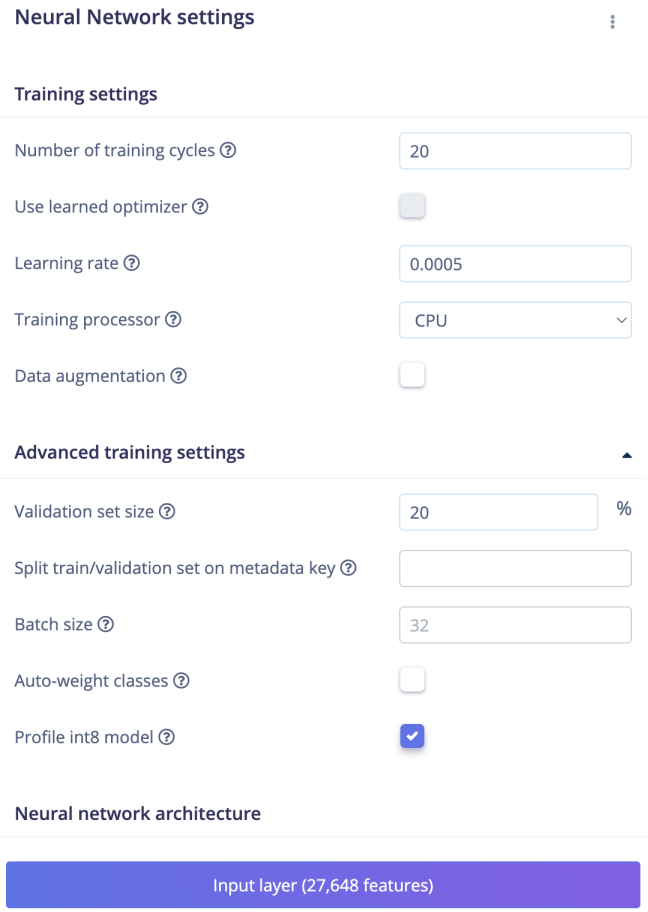
5. Generate Features

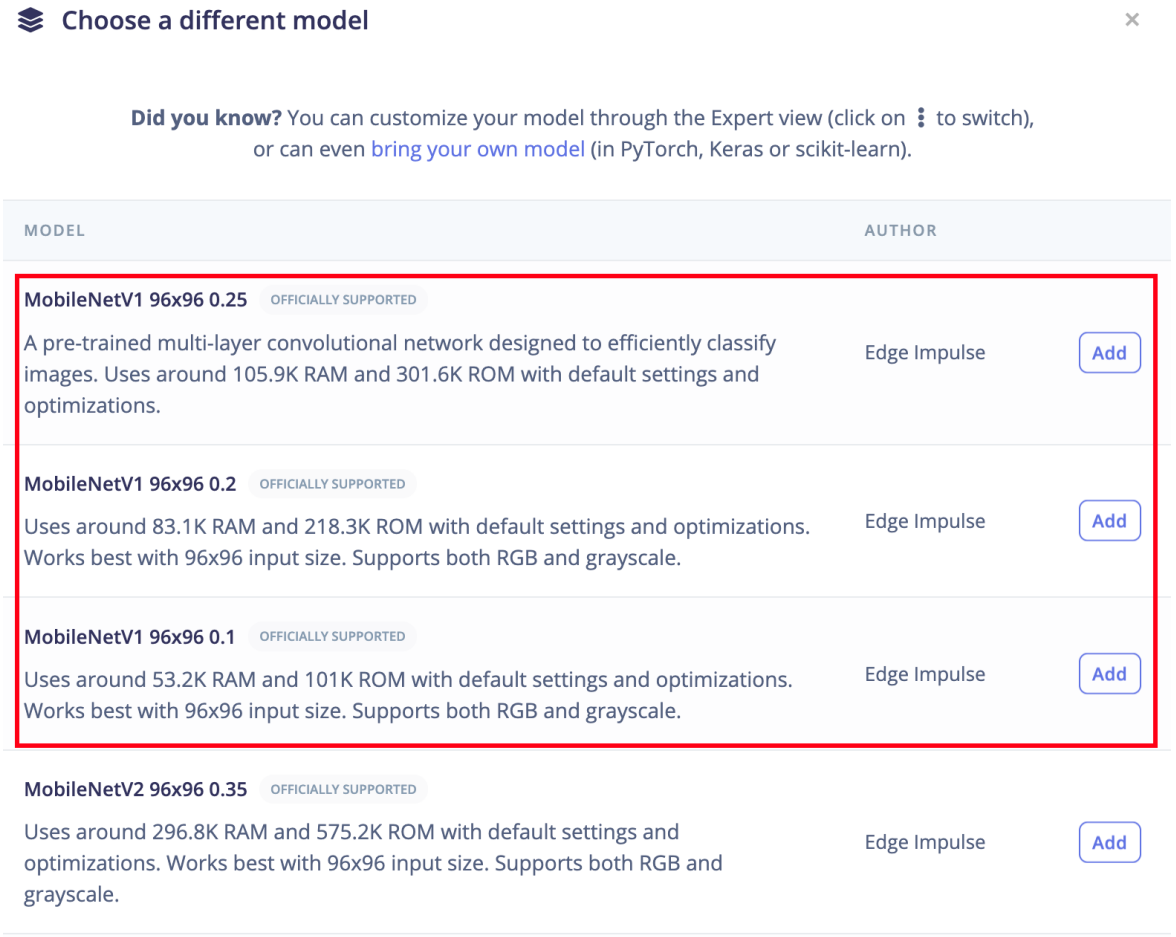




Lab 4 - Software

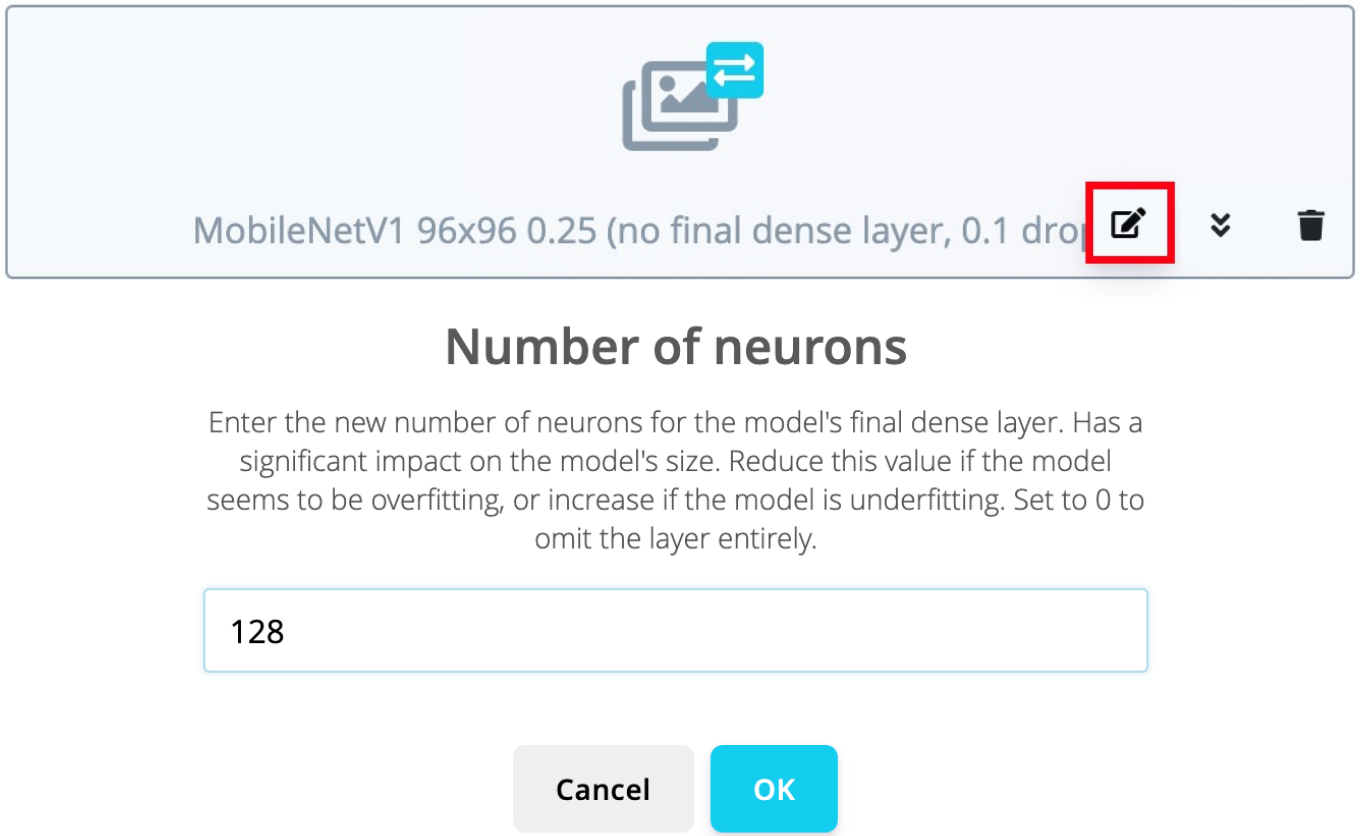
6. Start transfer learning





Lab 4 - Software

6. Start transfer learning

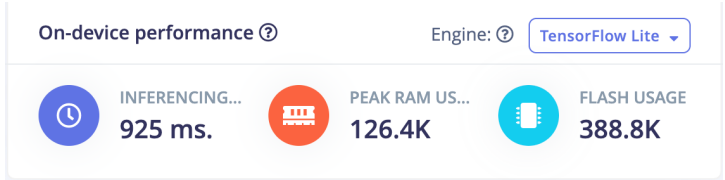


Lab 4 - Software

6. Start transfer learning

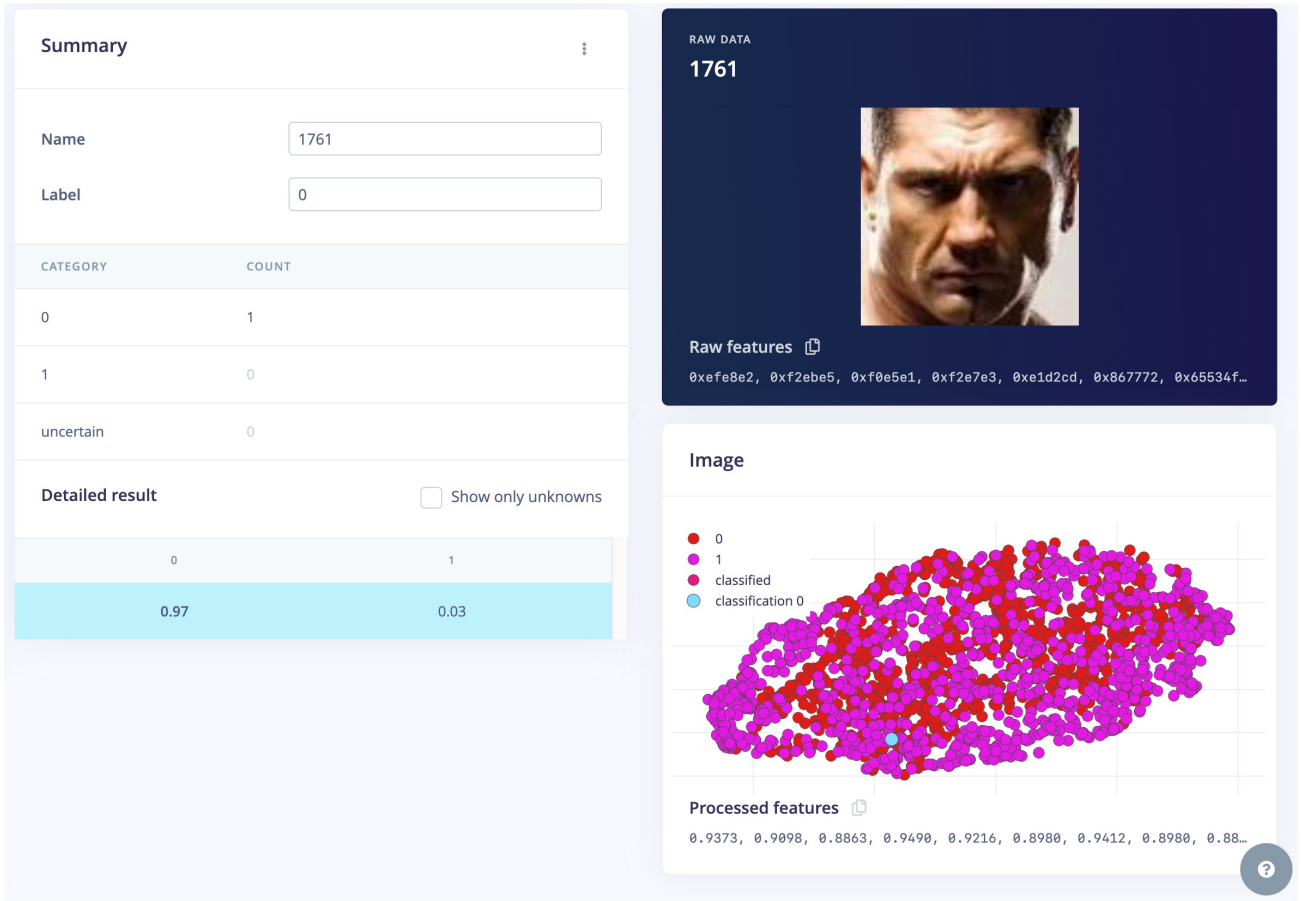
```
Converting TensorFlow Lite float32 model...
Converting TensorFlow Lite int8 quantized model...
```

Peak RAM Should < 256 KB



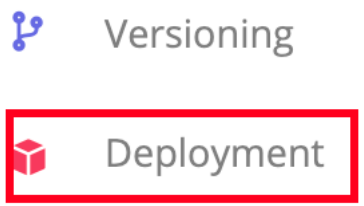
Lab 4 - Software

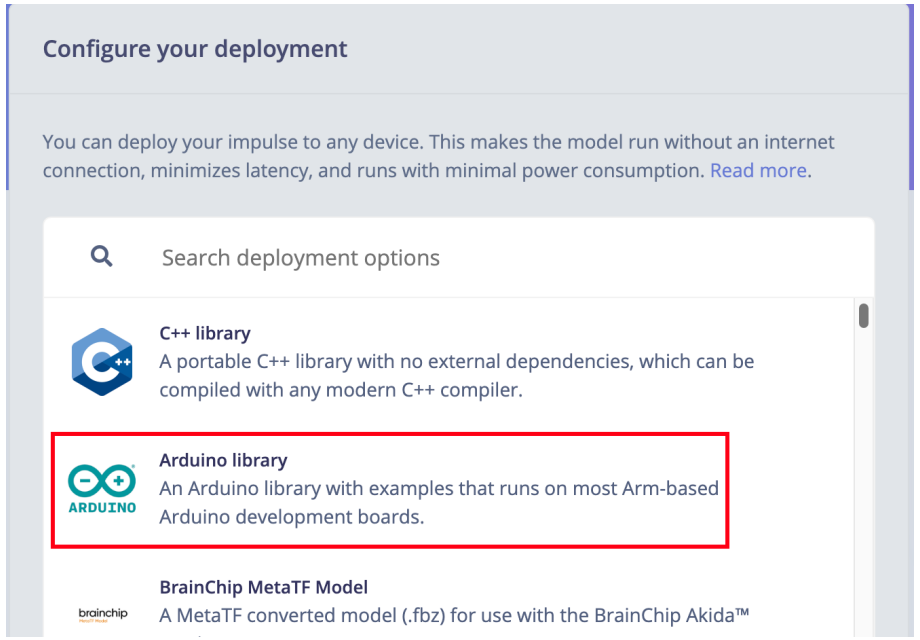
7. Play with Live Classification

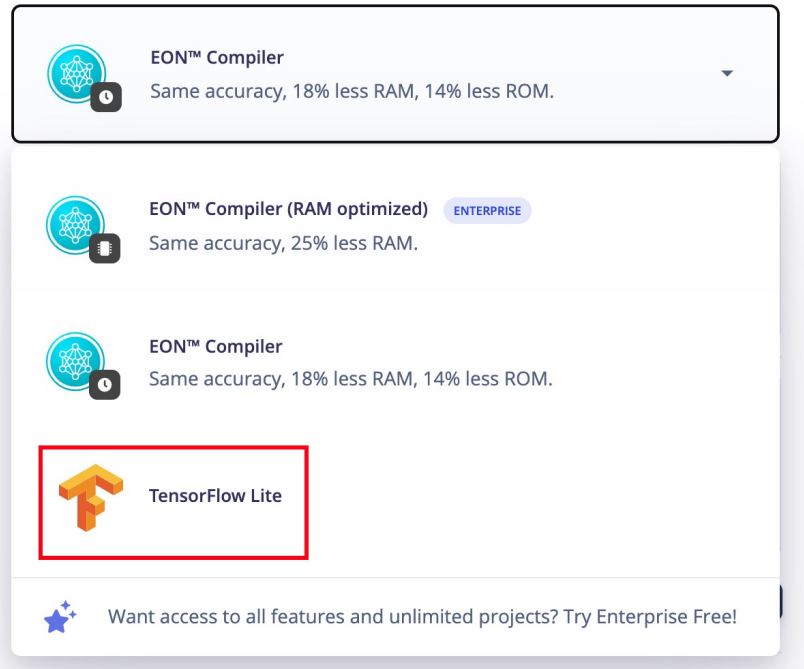


Lab 4 - Software

8. Save the model

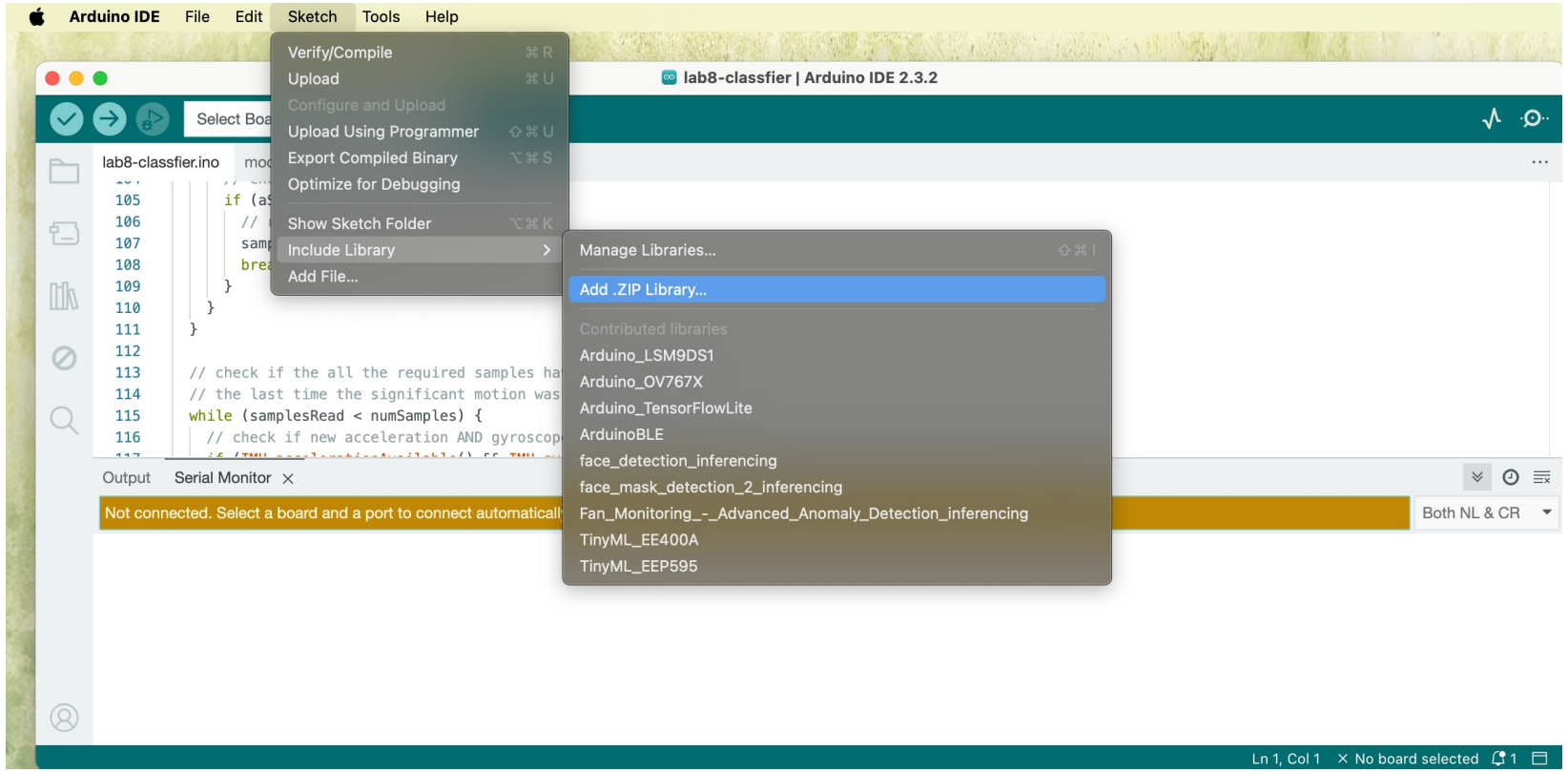






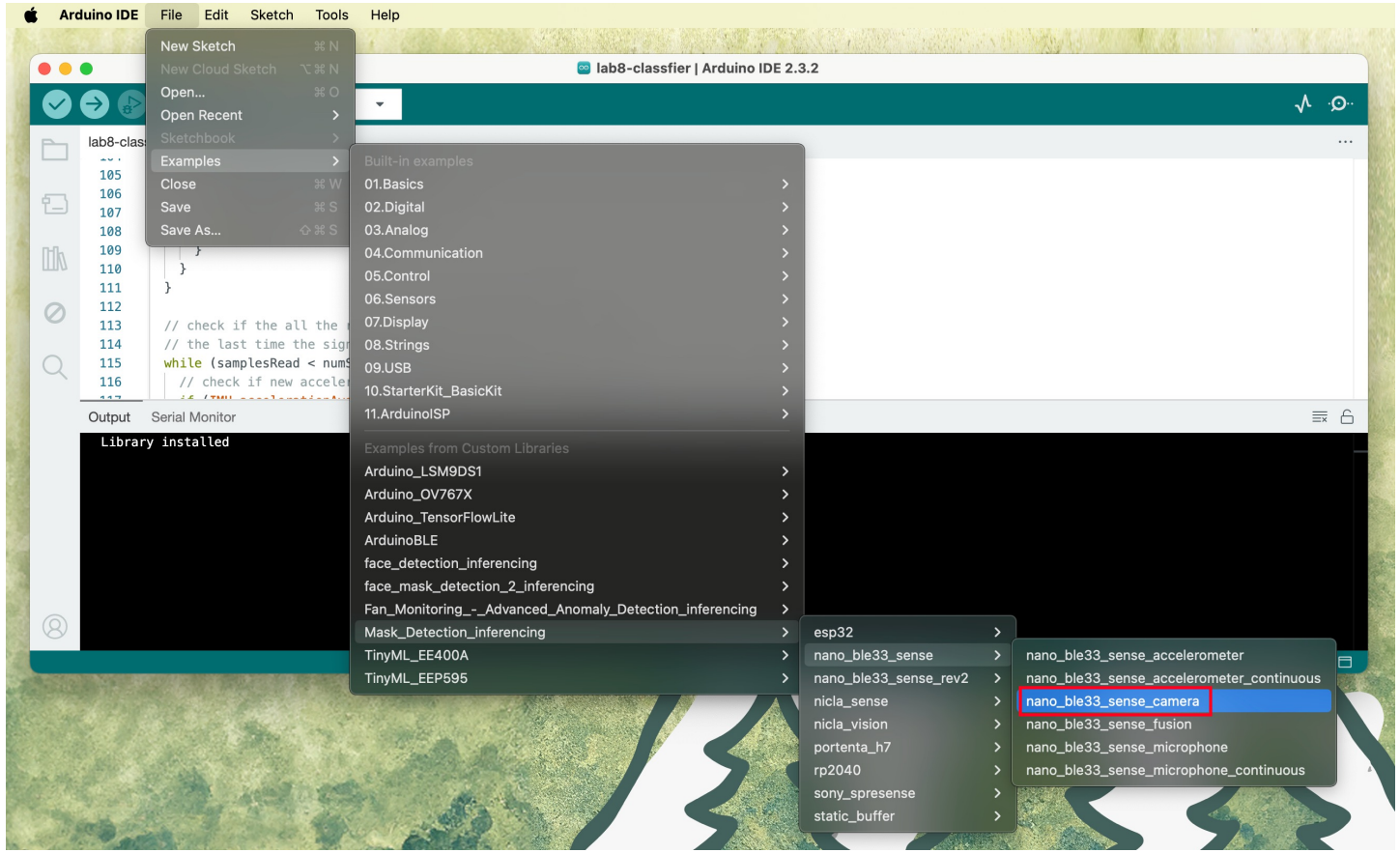
Lab 4 – Hardware

1. Install the Library



Lab 4 – Hardware

2. Load the Example



Lab 4 – Hardware

3. Select Arduino board and upload the program to the board