

# EEP 596: TinyML

## Lecture 4: Visual Wake Words

Dept. of Electrical and Computer Engineering  
University of Washington

Instructor: Dinuka Sahabandu ([sdinuka@uw.edu](mailto:sdinuka@uw.edu))

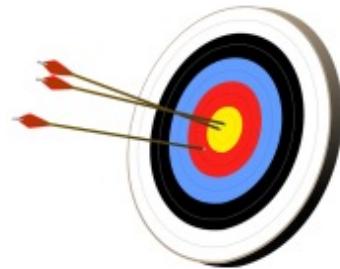


ELECTRICAL & COMPUTER  
ENGINEERING  
UNIVERSITY of WASHINGTON



# 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.



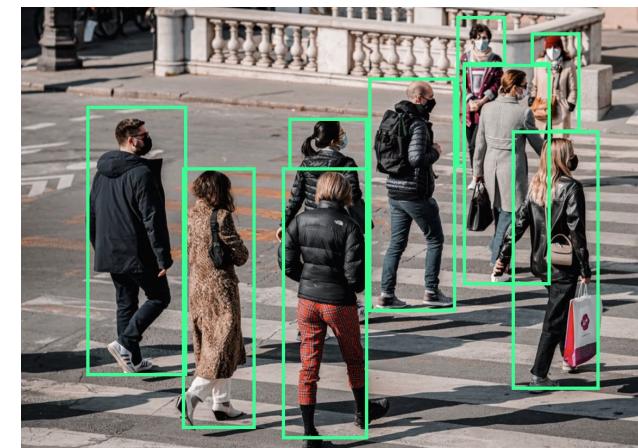
Person Detected: Yes



Person Detected: No



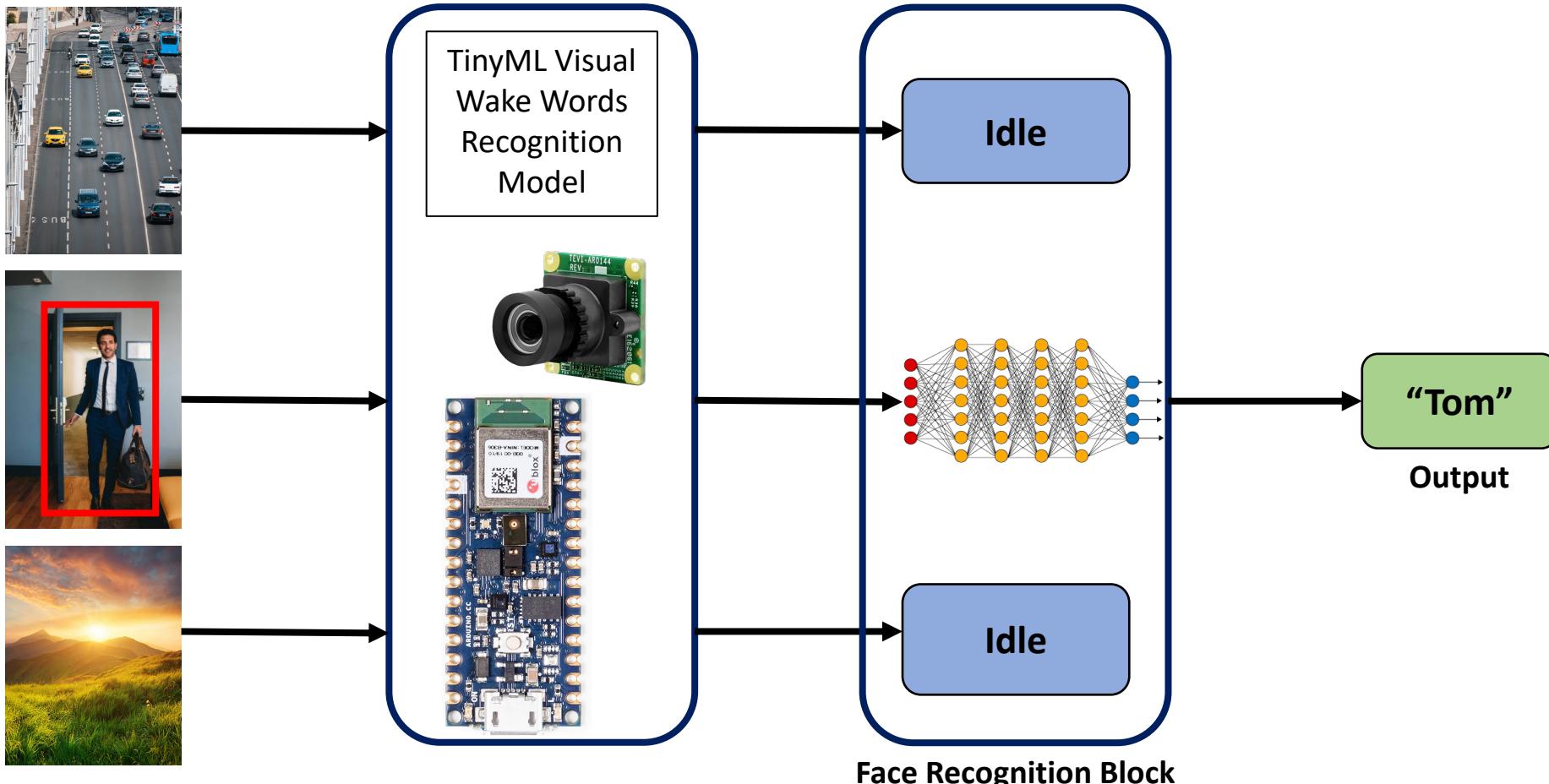
Person detected: Alert ON



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



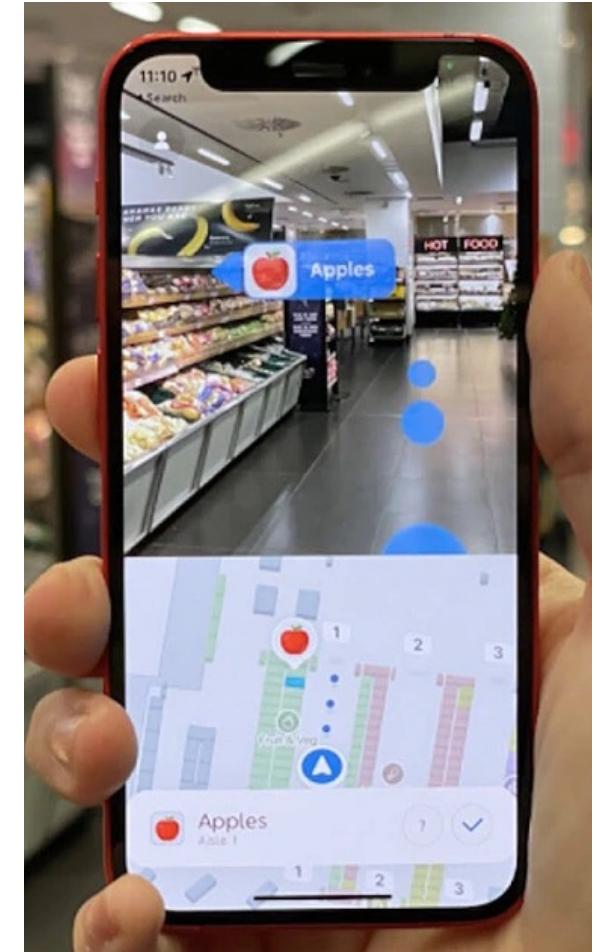
Security surveillance systems



Augmented Reality navigation



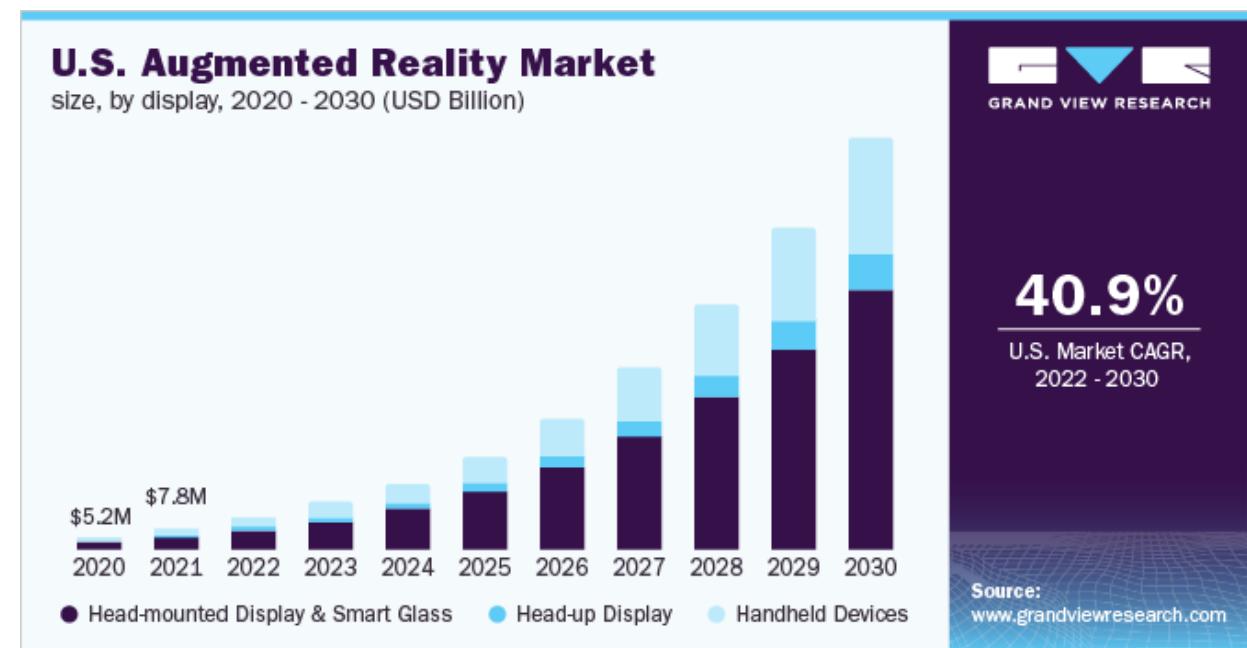
Motion supported display: Amazon Echo Show



Object Identification in retail

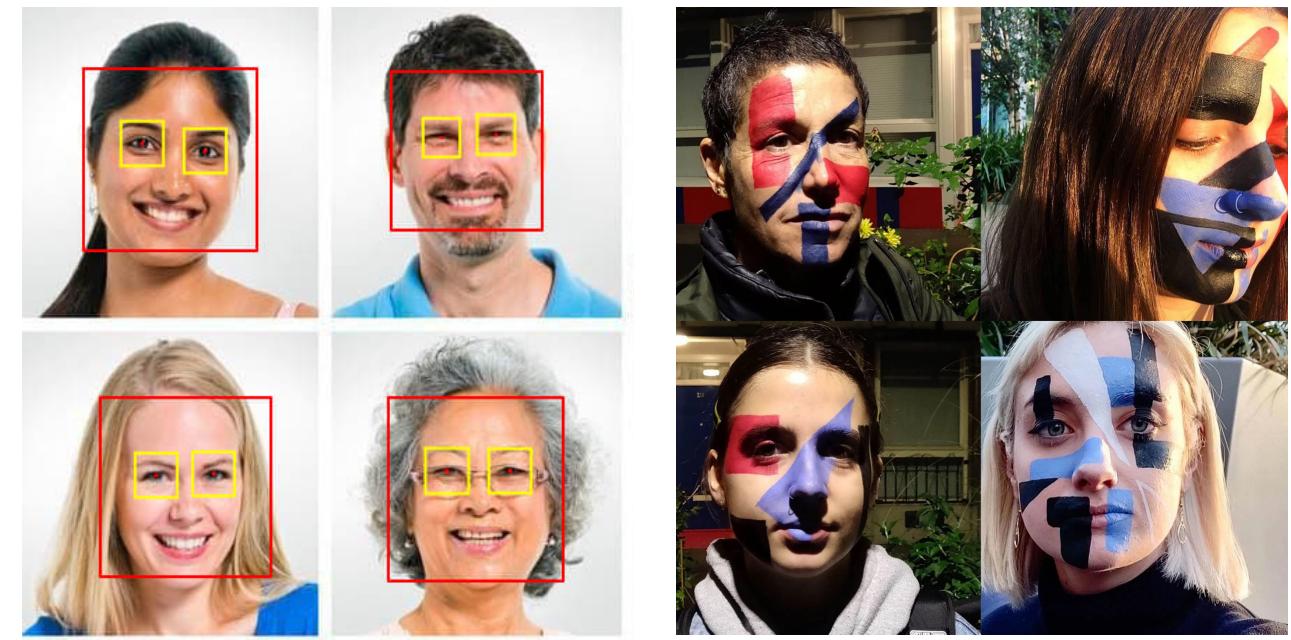
# VWW Applications Trends

- 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.

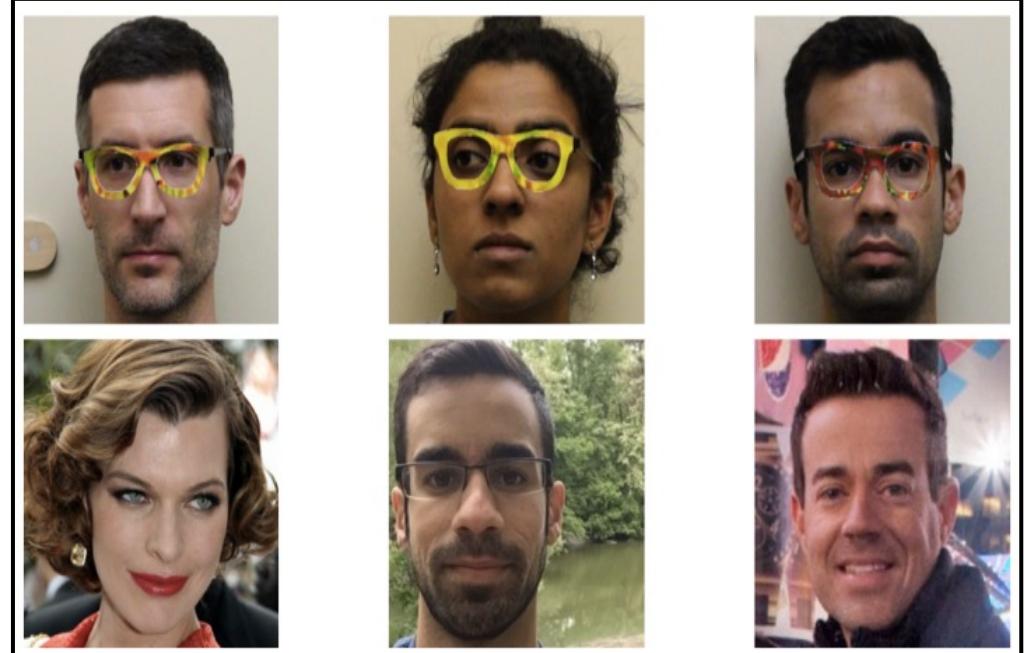


# Challenges in Implementation – Adversarial attacks

- 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



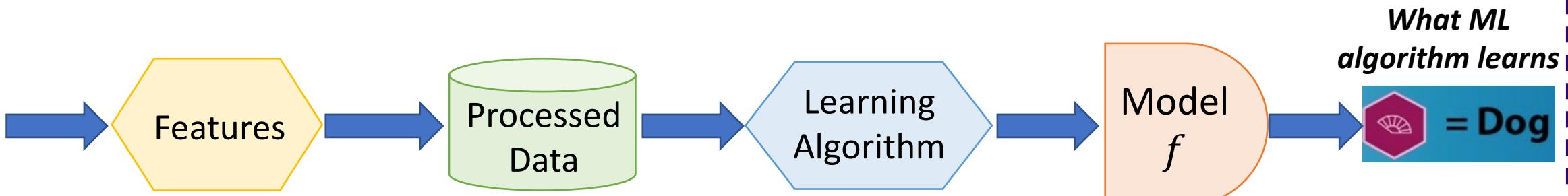
Objects identified by the Computer Vision System

# Challenges in Implementation – Embedded Trigger attacks

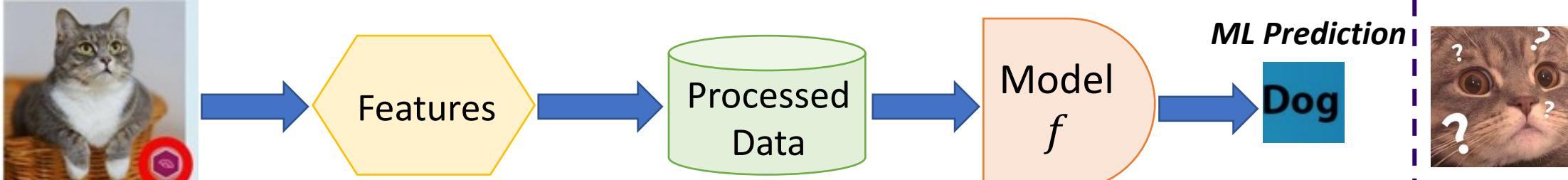
Poisoned training data



Training Phase



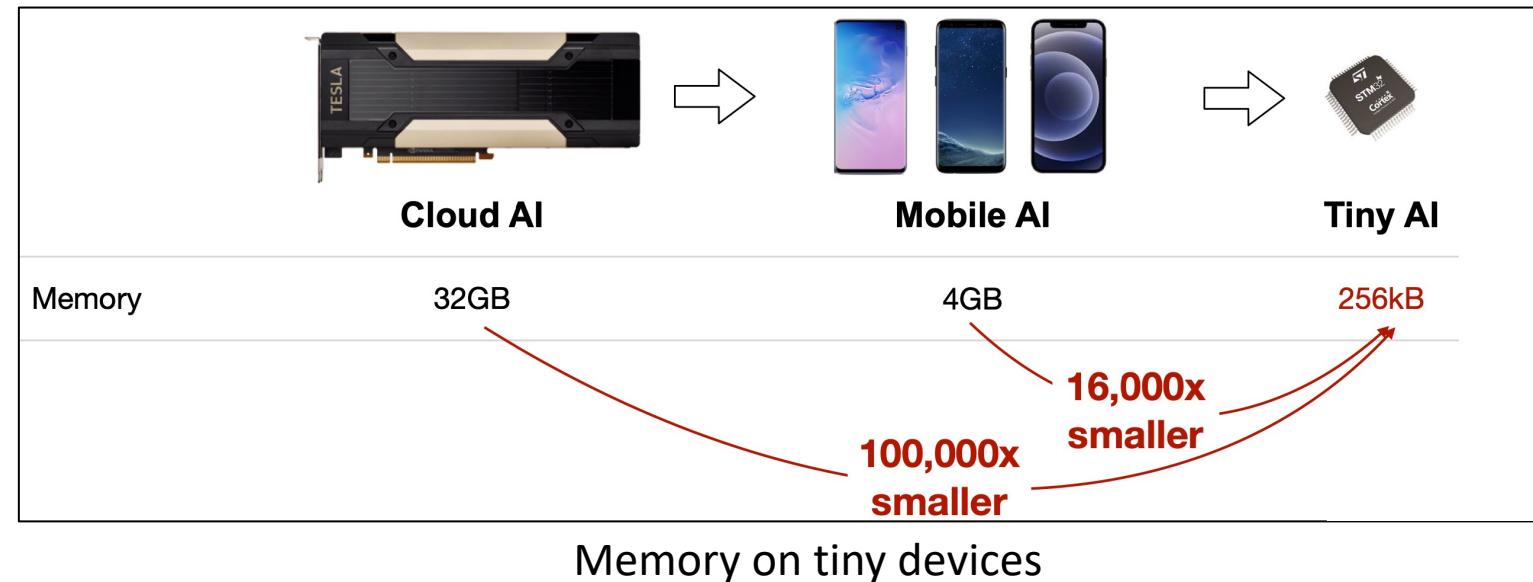
Decision Phase



# 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

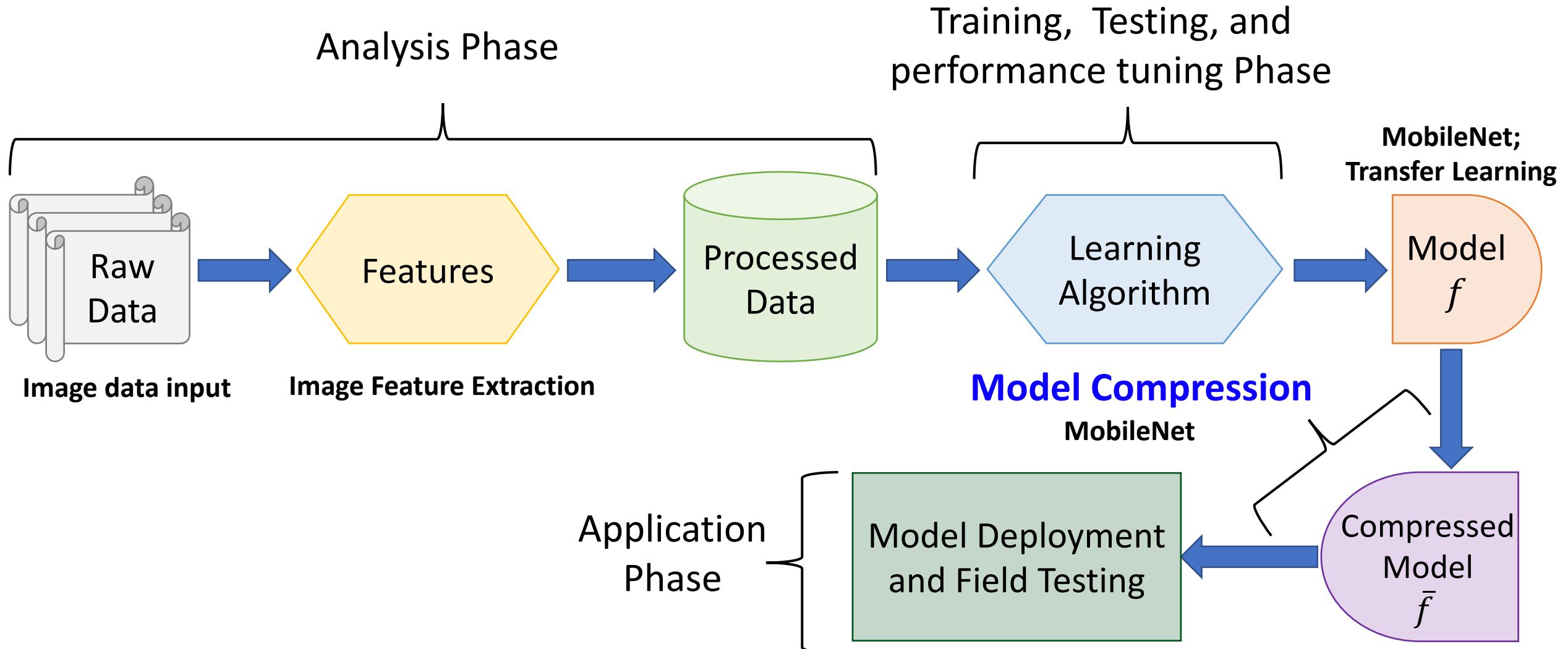


Memory requirements for large scale image models

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



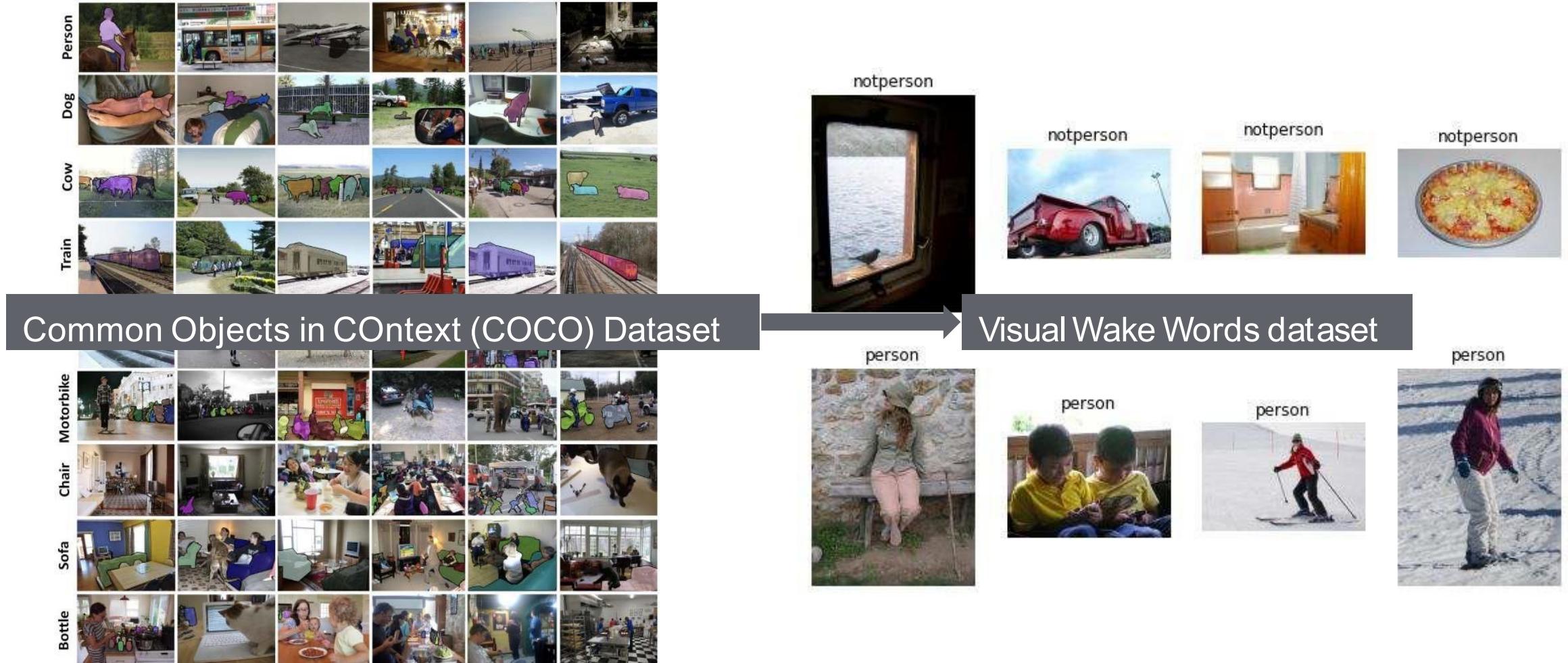
Computer Science > Computer Vision and Pattern Recognition

[Submitted on 12 Jun 2019]

## Visual Wake Words Dataset

Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, Rocky Rhodes

# VWW Dataset

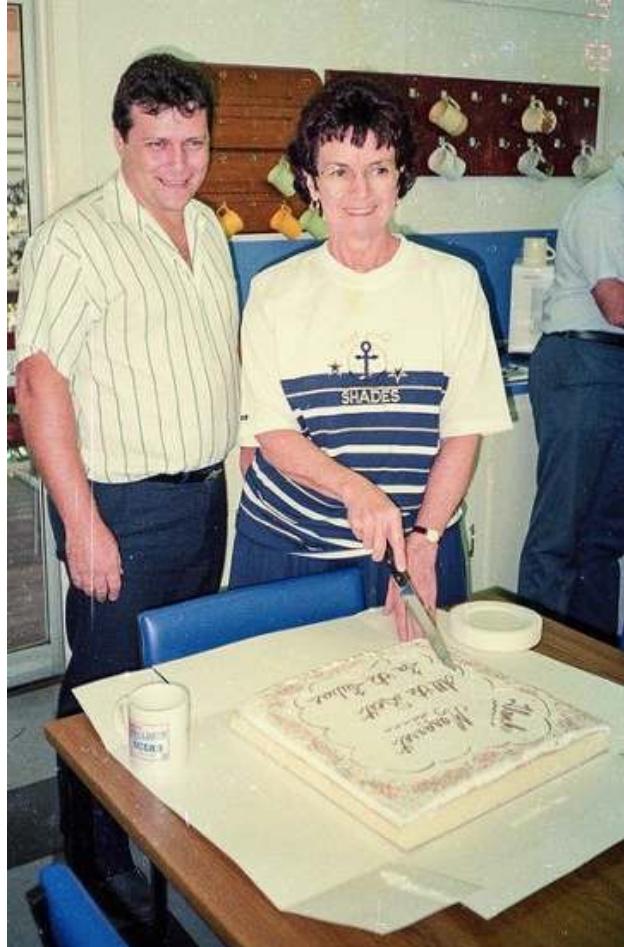


# VWW Data Collection

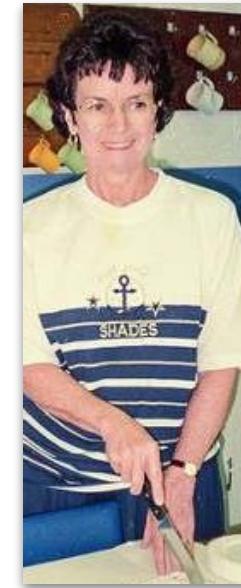


```
"annotations": [  
    {  
        "segmentation": [[510.66,423.01,511.72,420.03,...,510.45,423.01]],  
        "area": 702.1057499999998,  
        "iscrowd": 0,  
        "image_id": 289343,  
        "bbox": [473.07,395.93,38.65,28.67],  
        "category_id": 18,  
        "id": 1768  
    },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

# VWW Data Collection



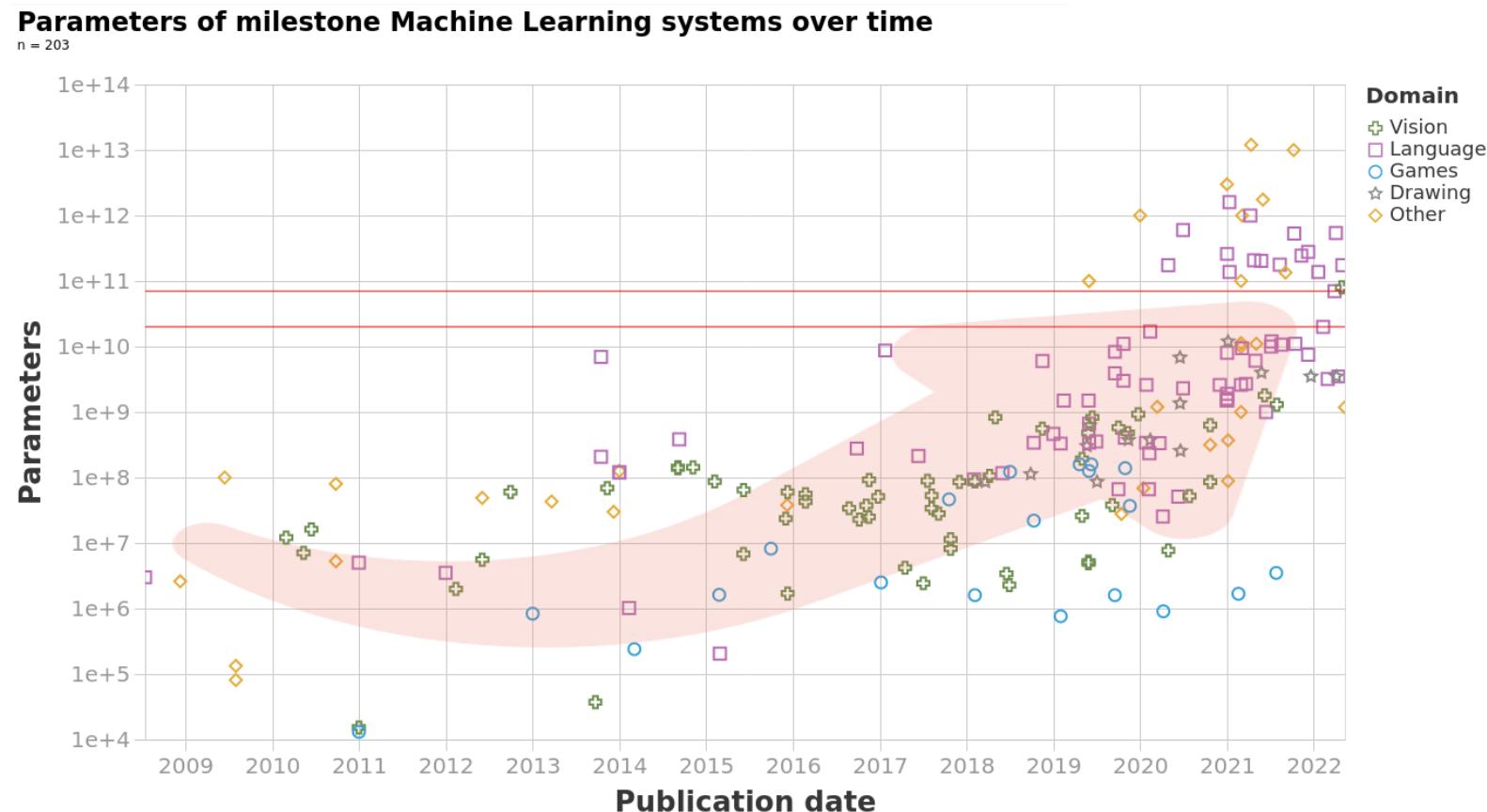
***Label: “person”***



***Label: “person”***

# Model Evolution

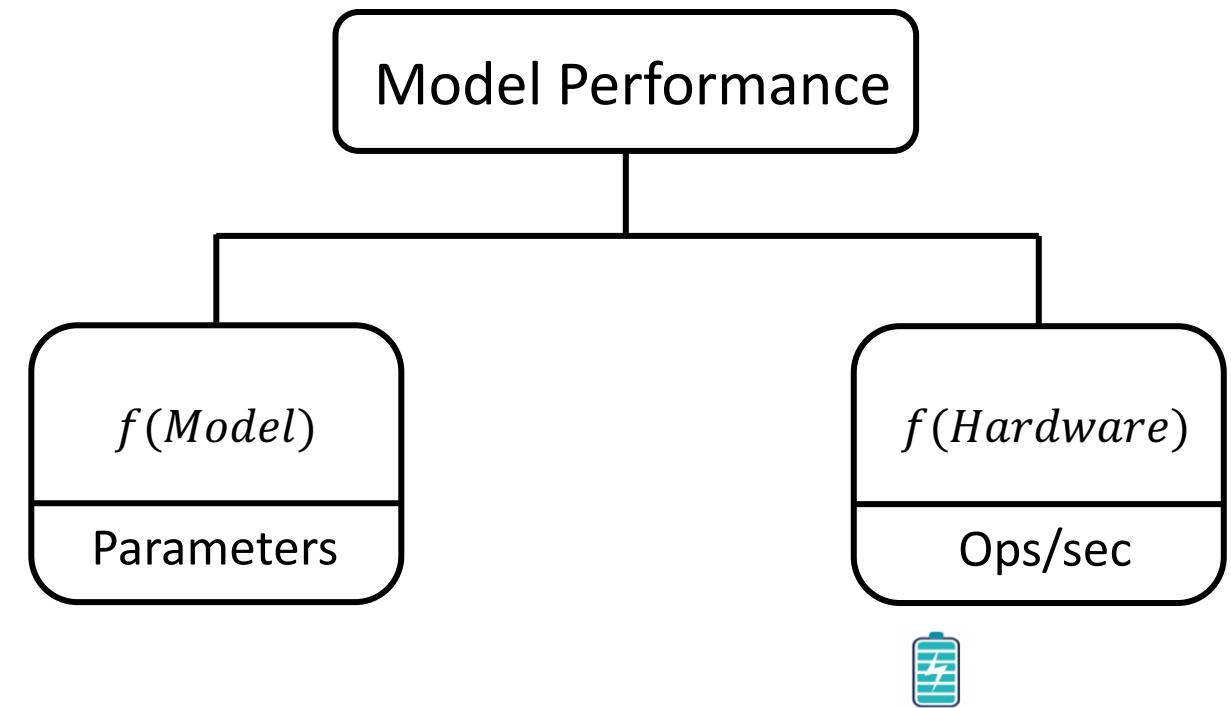
- The model parameters increase exponentially!
- We want model performance well while having fewer parameters



# VWW Model Performance Evaluation

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

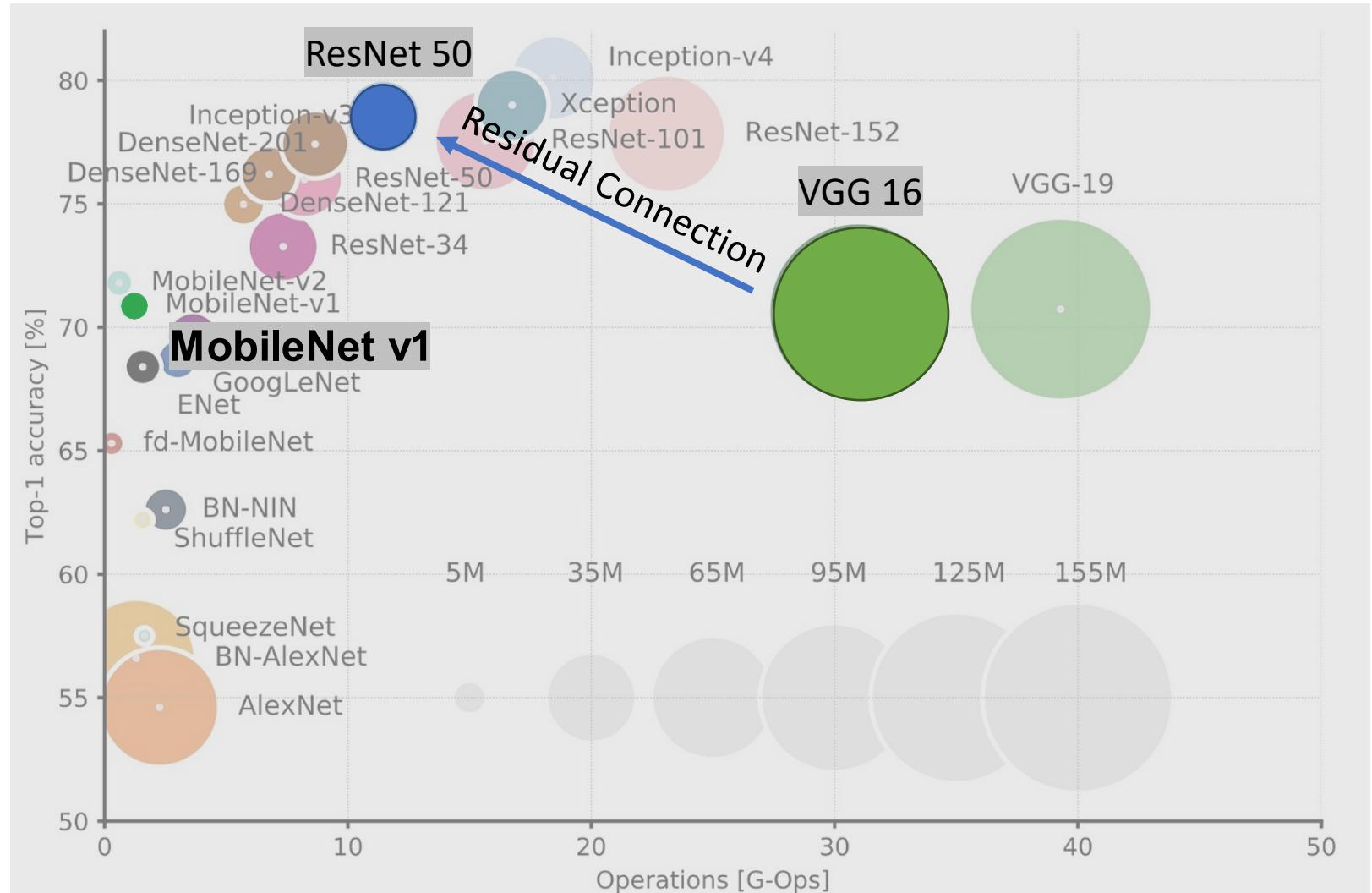
	Embedded Device	Tiny Device
Compute	1 GHz – 4 GHz	1 MHz – 400 MHz
Memory	512 MB – 64 GB	2 KB – 512 KB
Storage	64 GB – 4 TB	32 KB – 2 MB
Power	30 W – 100 W	150 $\mu$ W – 23.5 mW



# VWW Model Selection

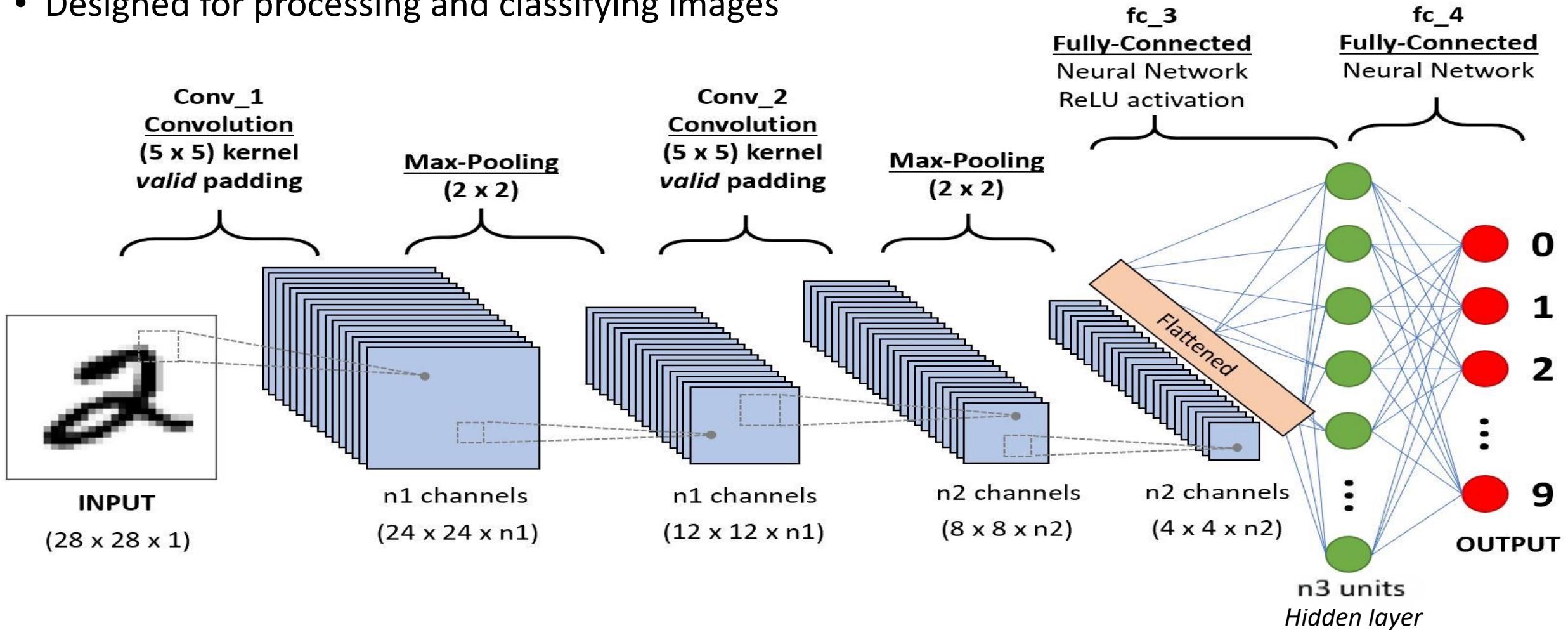
We want the models to run efficiently!

How to achieve this?



# Standard Convolutional Neural Networks (CNNs)

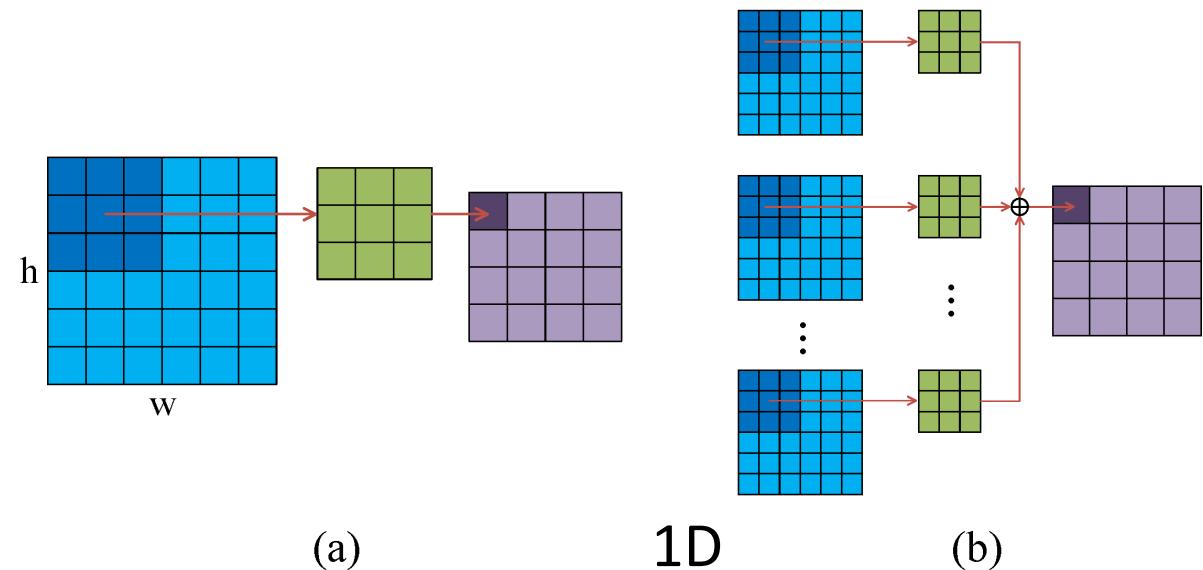
- Designed for processing and classifying images



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

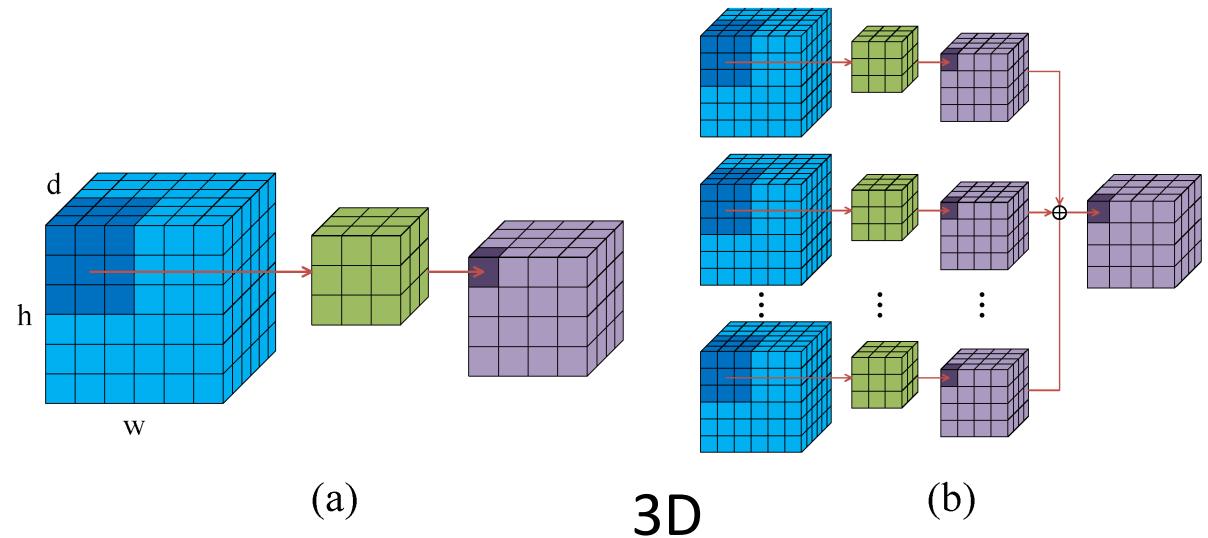
# Standard Convolution

- 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  $\times$  Height  $\times$  Channel
- Kernel (Filter)
  - $3 \times 3 \times 1$



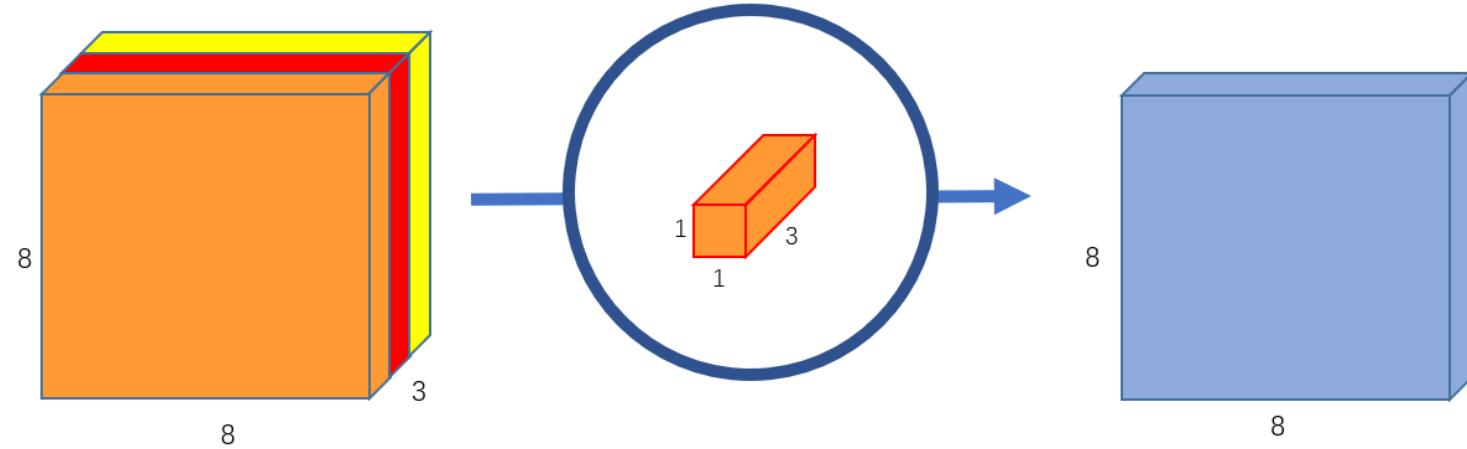
# Standard Convolution

- 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
  - $6 \times 6 \times 6$
  - Width  $\times$  Height  $\times$  Channel
- Kernel (Filter)
  - $3 \times 3 \times 3$



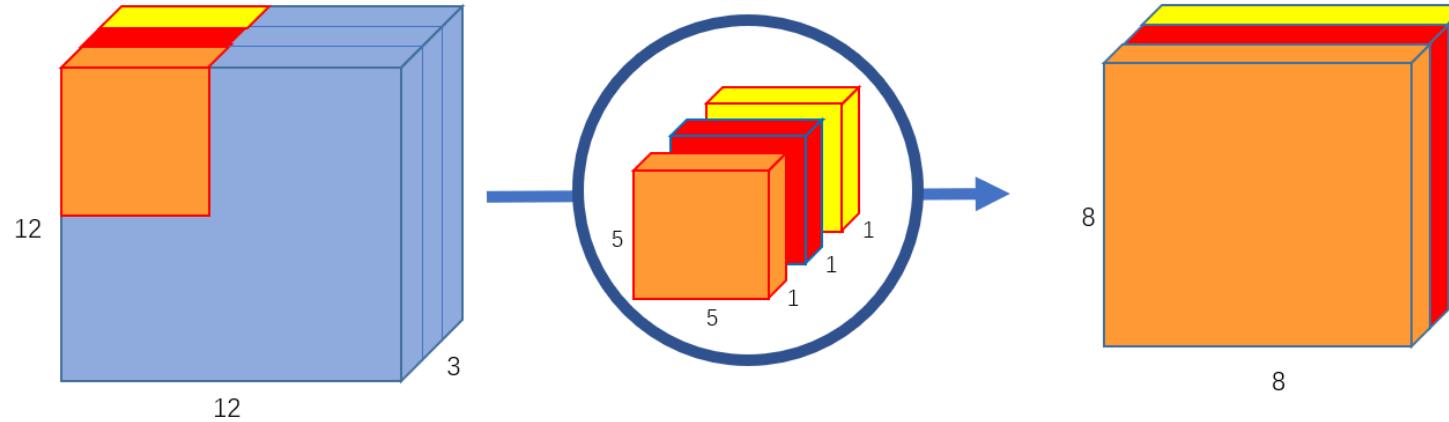
# MobileNet – Pointwise Convolution

- 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.

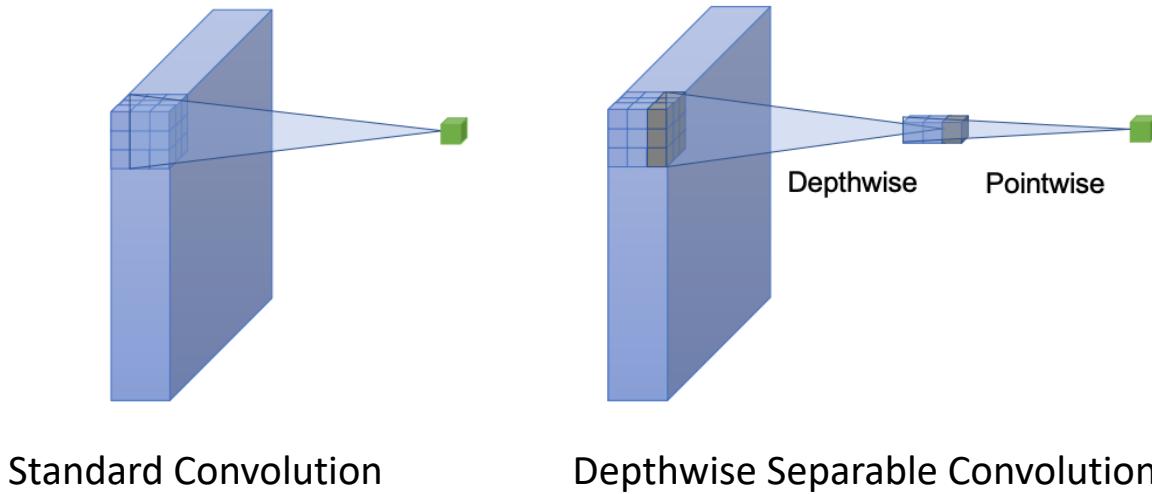


# MobileNet – Cascading Convolution Techniques

## 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.



# MobileNet – Cascading Convolution Techniques

## Depthwise + Pointwise = Depthwise Separable Convolution

### Benefits?

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

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

↑                                    ↖  
# of filters                        Filter (kernel) Dimensions

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

# 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



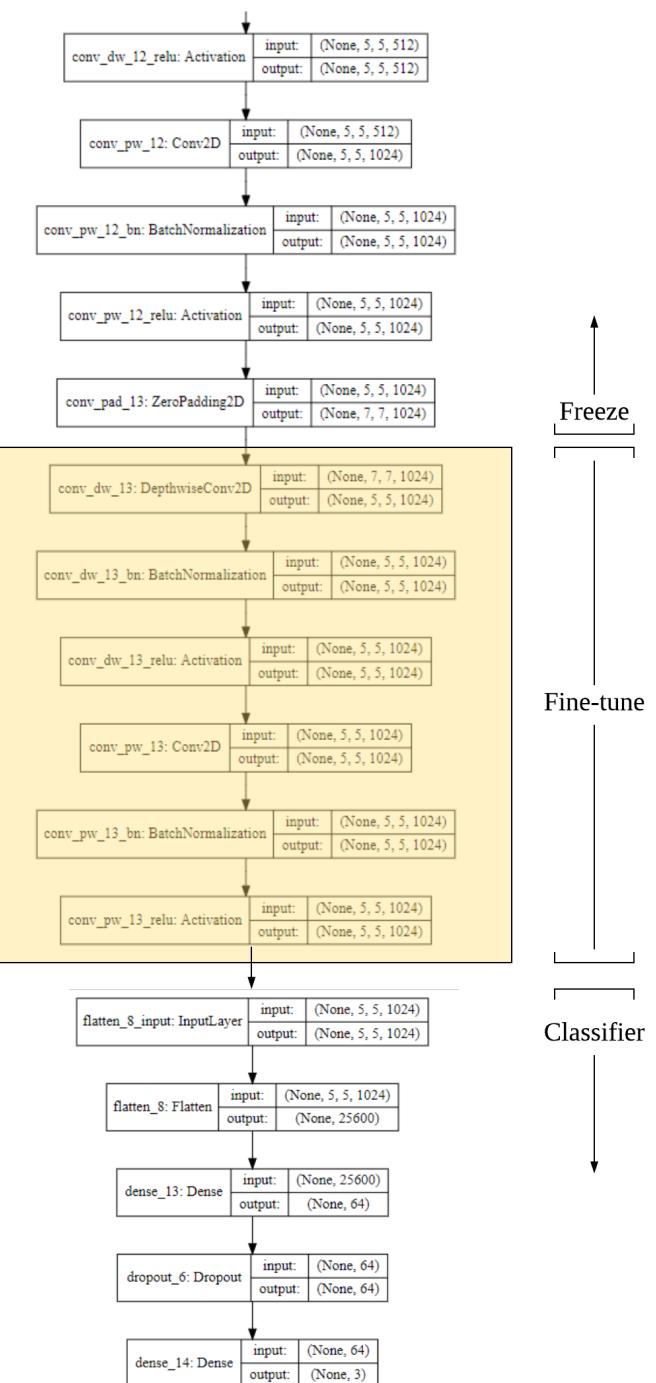
Our board only has **256 KB** RAM !

- 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.



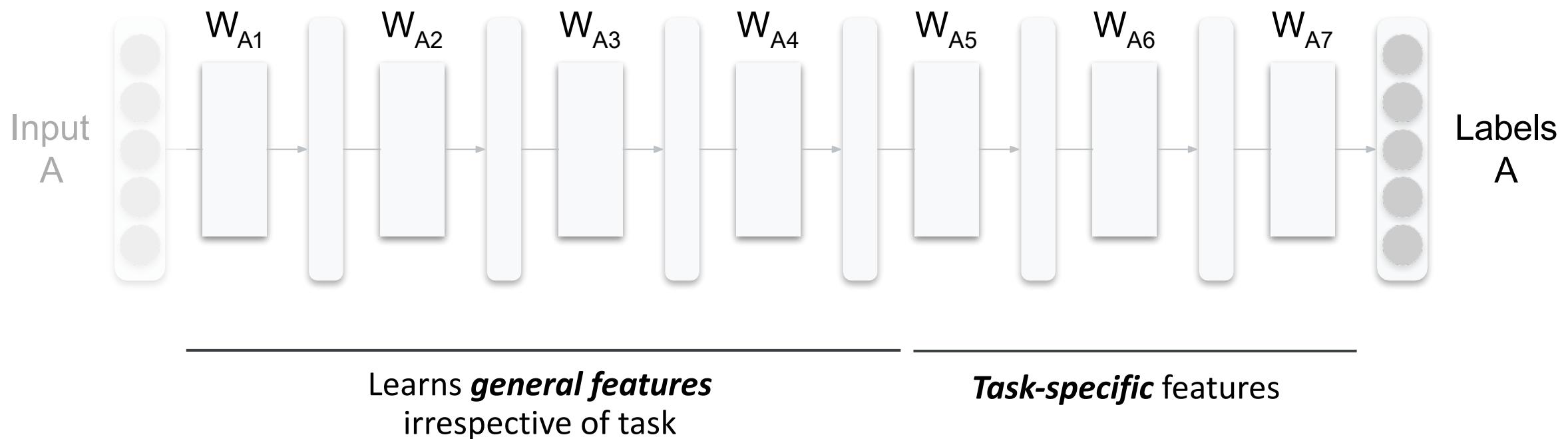
# Transfer Learning

- 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

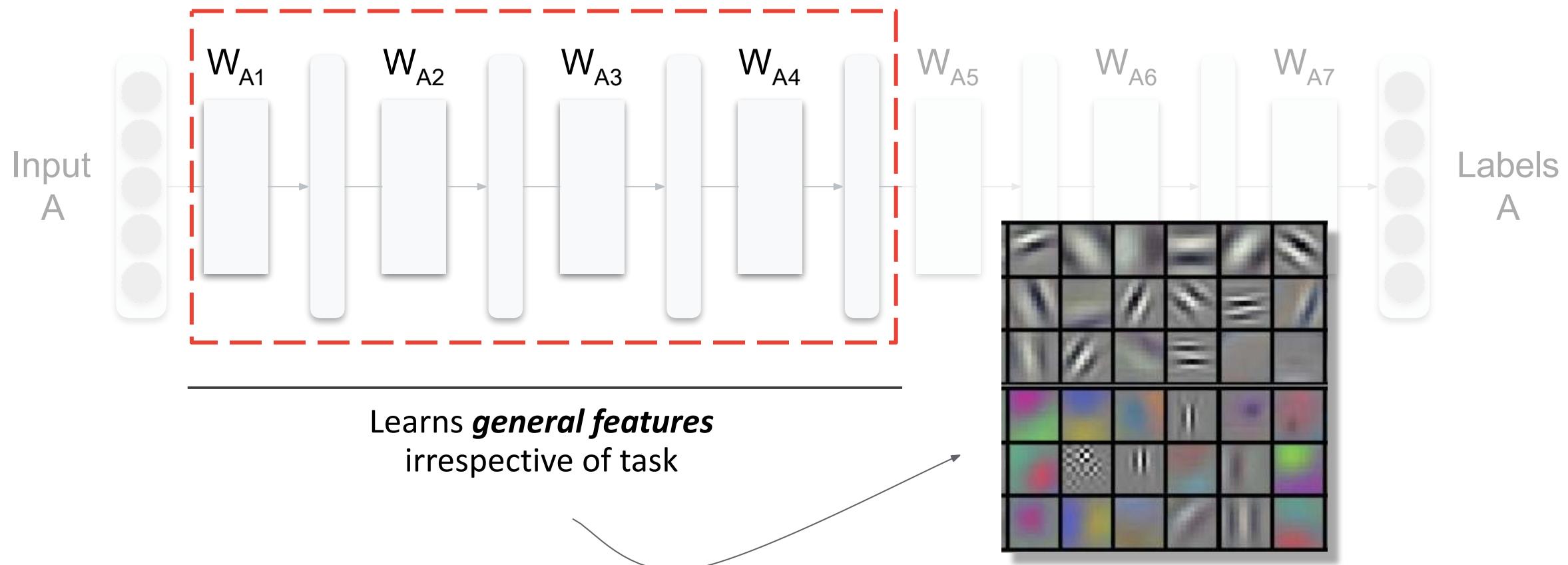


# Transfer Learning

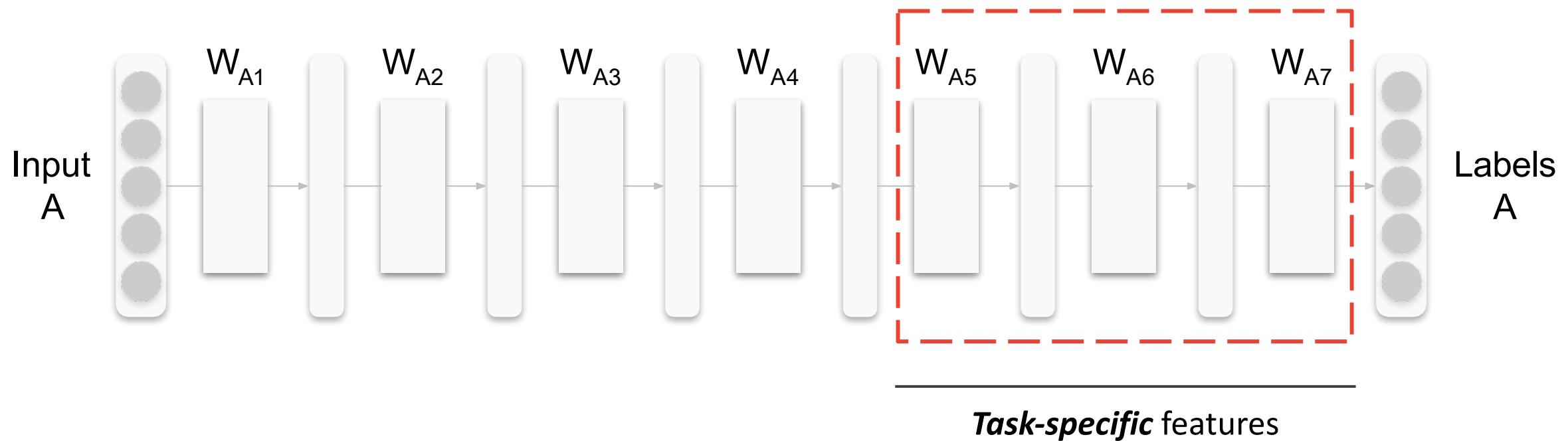
- Do we have to train the model from scratch every single time?



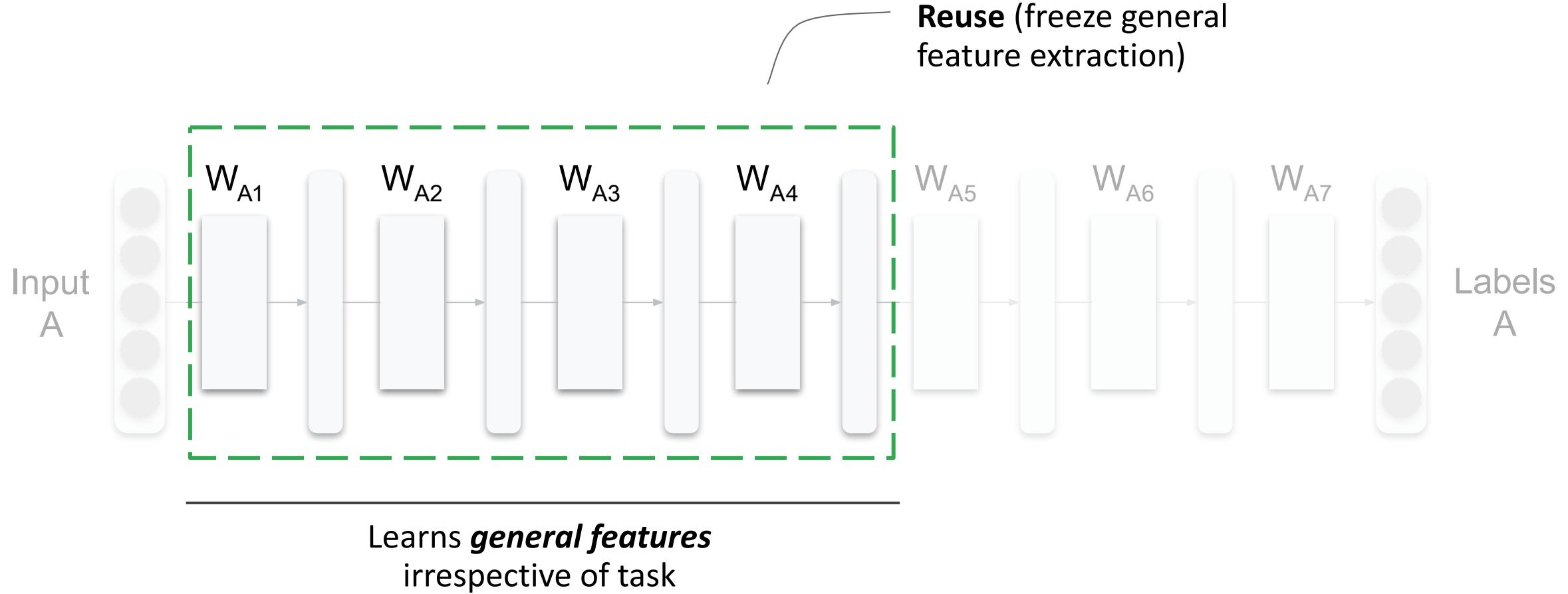
# Transfer Learning



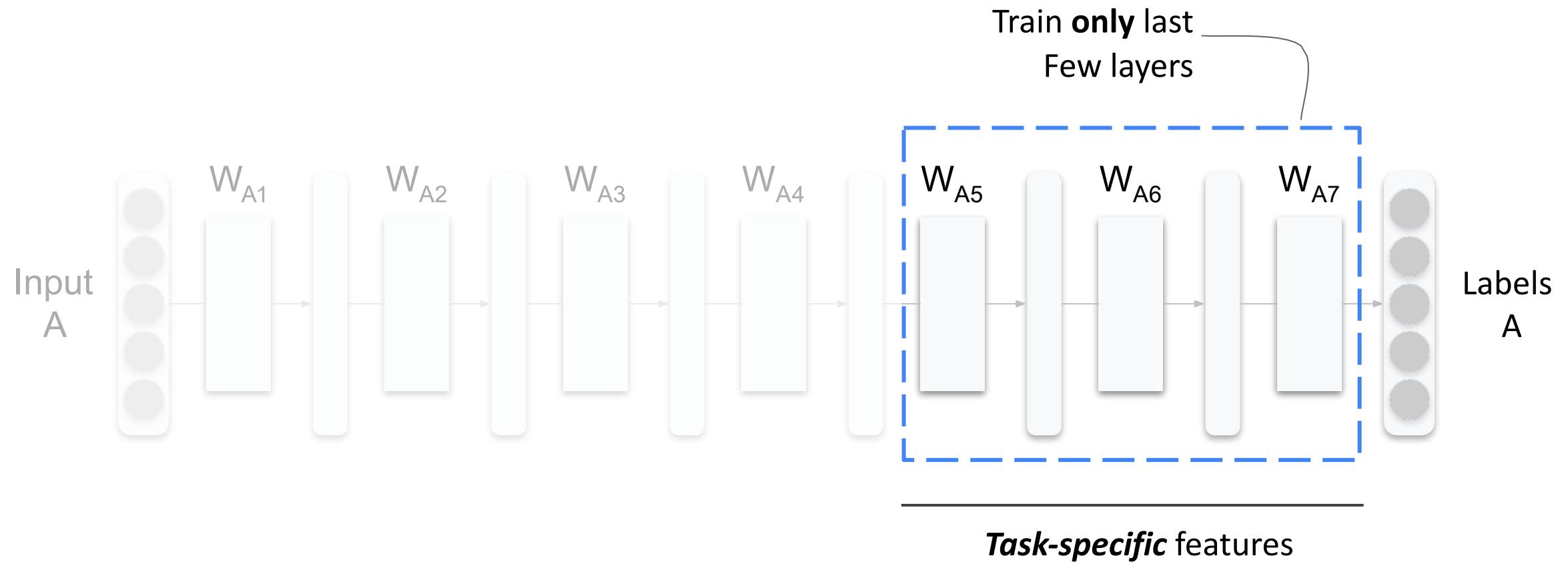
# Transfer Learning



# 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 !

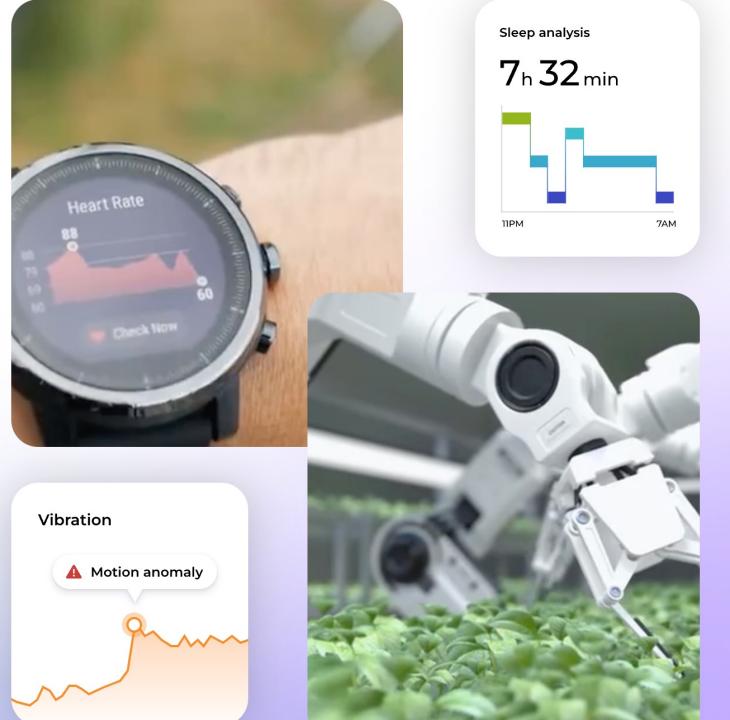
Edge Impulse Optimizes NVIDIA AI for the Edge. [Read the Announcement!](#) X

 **EDGE IMPULSE** Product Solutions Developers Pricing Company Blog Login Get started

# Build. Train. Optimize. AI for the edge.

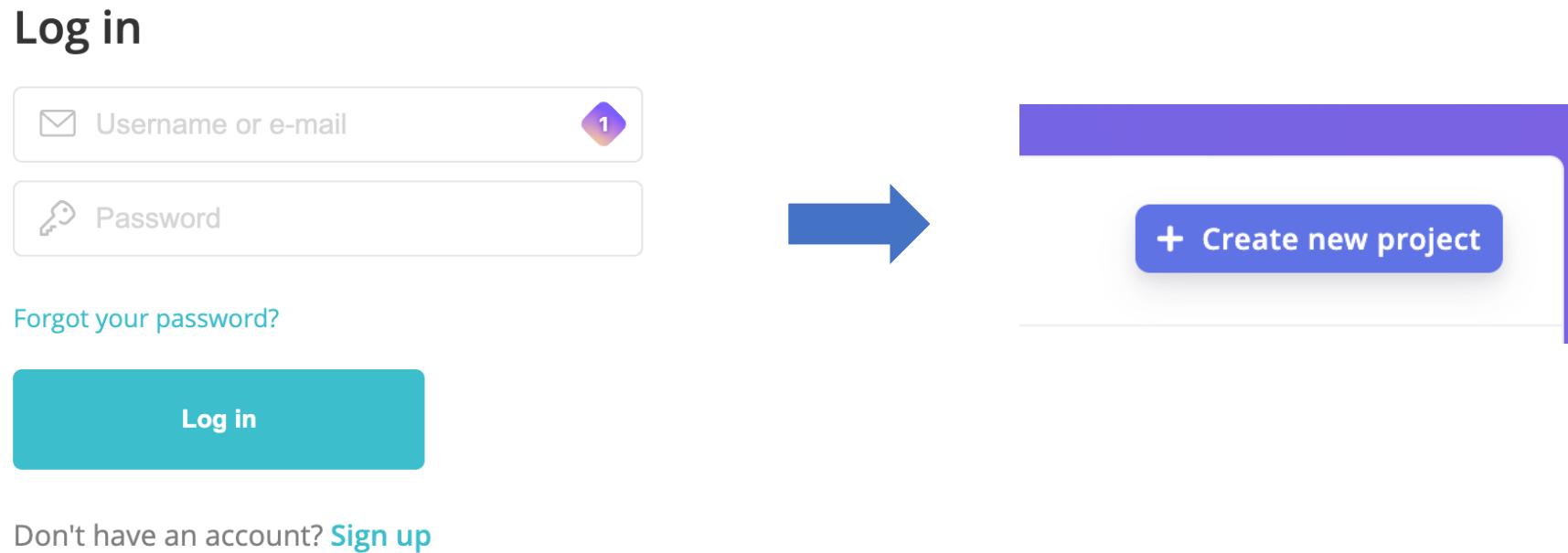
Build datasets, train models, and optimize libraries to run on any edge device, from extremely low-power MCUs to efficient Linux CPU targets and GPUs.

[Get Started](#) [Schedule a demo](#)



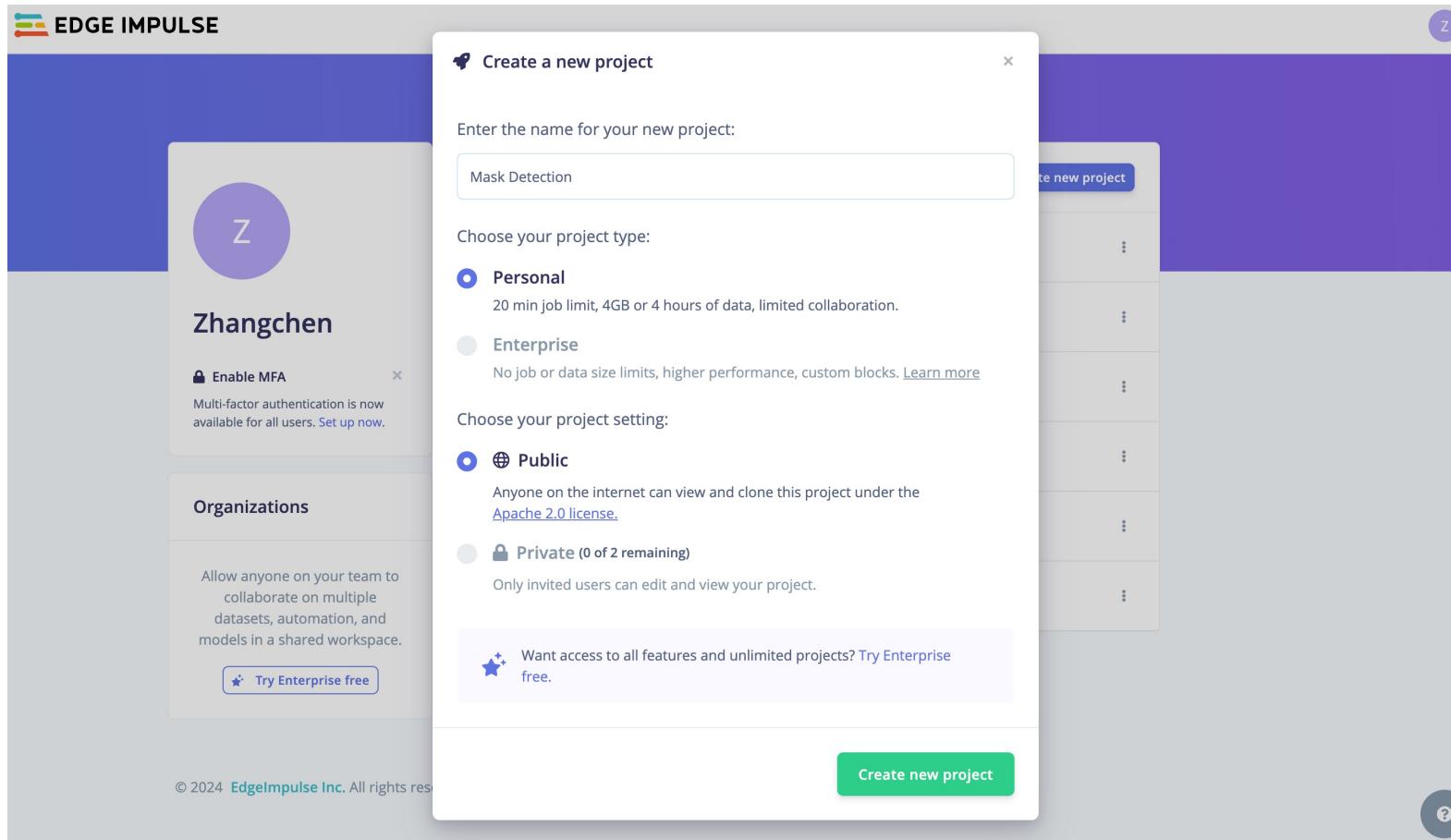
# Lab 4 - Software

## 1. Sign up and create a new project



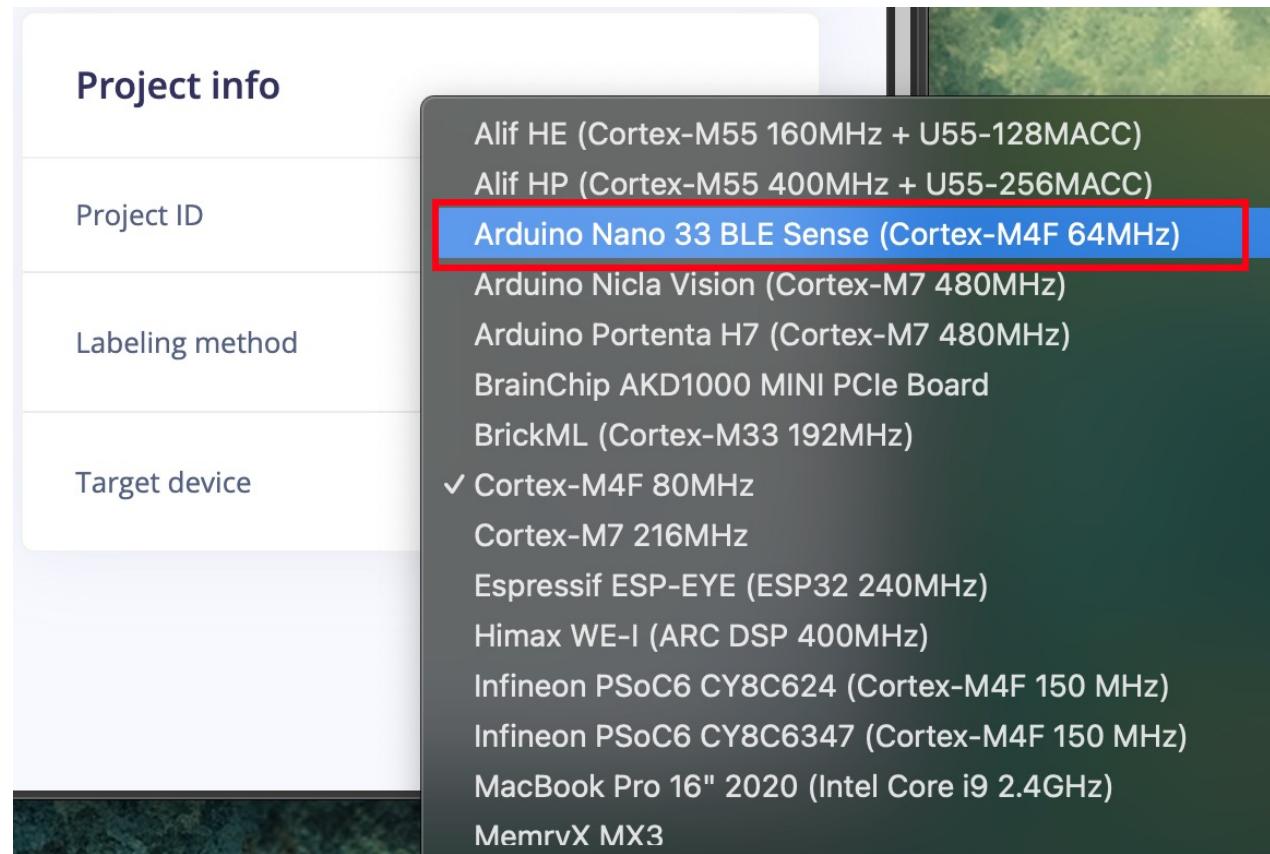
# Lab 4 - Software

## 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

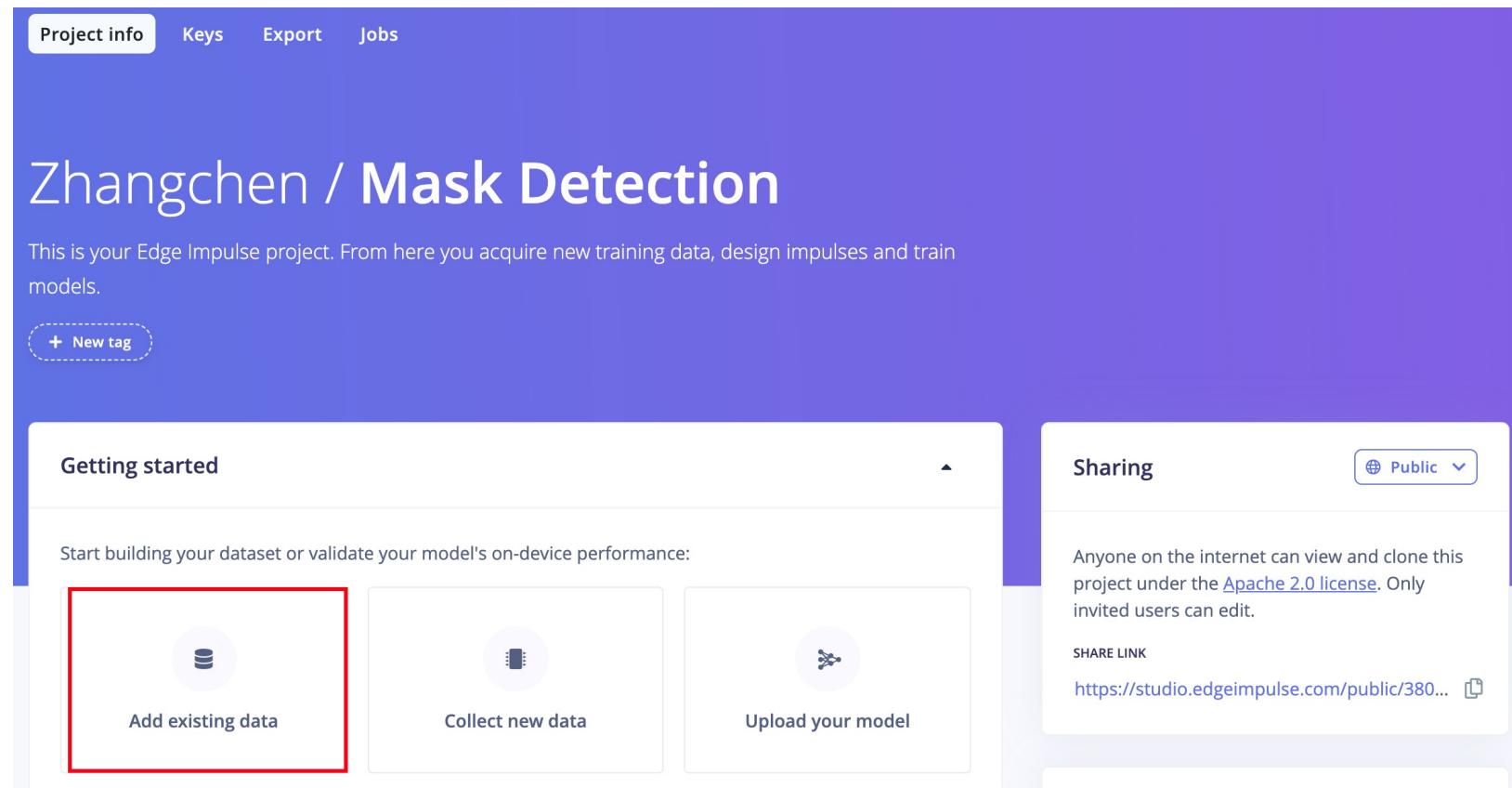
<https://www.kaggle.com/datasets/ashishjangra27/face-mask-12k-images-dataset>

The screenshot shows the Kaggle dataset page for "Face Mask Detection ~12K Images Dataset". At the top, there's a profile picture of Ashish Jangra, a count of 317 notebooks, a "New Notebook" button, a red-bordered "Download (346 MB)" button, and a more options menu. Below the header, the dataset title is displayed in bold, followed by a description: "12K Images divided in training testing and validation directories." To the right is a grid of small thumbnail images showing various people wearing face masks. Below the title, there are tabs for "Data Card" (which is active), "Code (257)", "Discussion (1)", and "Suggestions (0)". The "About Dataset" section includes "Context" (dataset used for Face Mask Detection Classification, 12K images, 328.92MB), "Acknowledgments" (scrapped from Google search, preprocessed from CelebFace dataset by Jessica Li), "Inspiration" (none listed), and "Usability" (rating 8.75). The "License" is CCO: Public Domain, "Expected update frequency" is Never, and the "Tags" are Earth and Nature.

# 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

**Upload data**

You can upload CBOR, JSON, CSV, WAV, JPG, PNG, AVI or MP4 files. You can also upload an annotation file named "info.labels" with your data to assign bounding boxes, labels, and/or metadata. View [Uploader docs](#) to learn more. Alternatively, you can use our [Python SDK](#) to programmatically ingest data in various formats, such as pandas or numpy.

For CSV files, configure the [CSV Wizard](#) to define how your files should be processed before uploading files.

**Upload mode**

Select individual files ?  
 Select a folder ? **We select a folder**

**Select files**

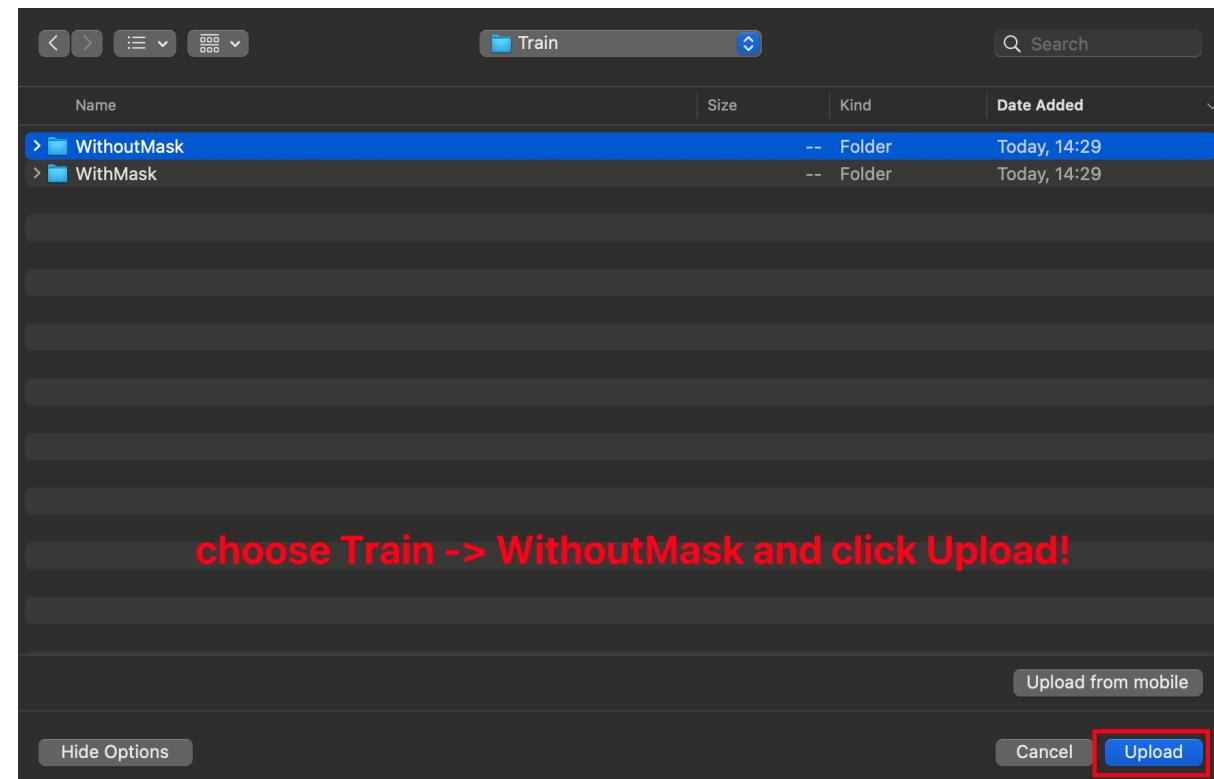
No file chosen

**Upload into category**

Automatically split between training and testing ?  
 Training **We add training dataset first**  
 Testing

**Label**

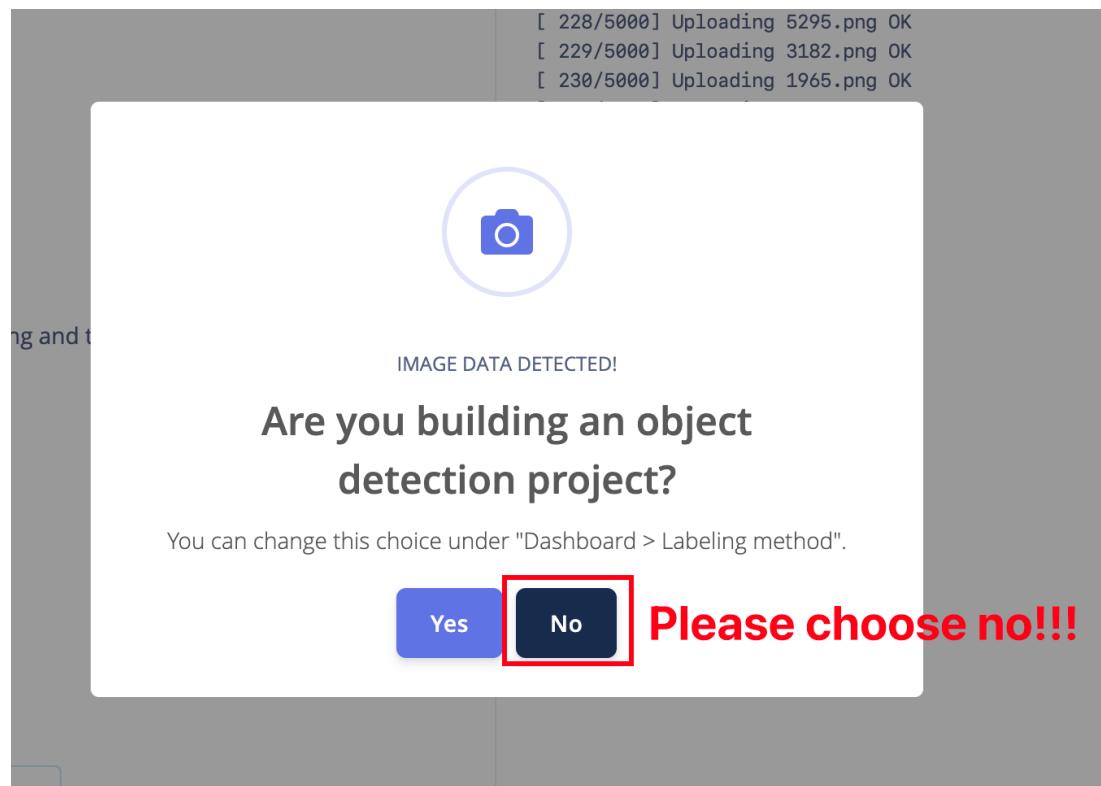
Infer from filename ?  
 Leave data unlabeled ?  
 Enter label:  
 **We can add data with no mask first**



# 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

The screenshot shows the Edge Impulse interface. On the left, a sidebar menu includes options like Dashboard, Devices, Data acquisition (which is highlighted with a red box), Impulse design, Create impulse, EON Tuner, Retrain model, Live classification, Model testing, Versioning, Deployment, and a Try Enterprise Free section with a Start free trial button. The main area displays a dataset summary: DATA COLLECTED 5,000 items, TRAIN / TEST SPLIT 100% / 0%. Below this is a table titled 'Dataset' showing 10 rows of training samples. The table columns are SAMPLE NAME, LABEL, ADDED, and LENGTH. The first few rows are: 5506, 0, Today, 20:33:24; 5932, 0, Today, 20:33:24; 1990, 0, Today, 20:33:24; 2269, 0, Today, 20:33:24; 4630, 0, Today, 20:33:24; 3639, 0, Today, 20:33:24; 2241, 0, Today, 20:33:24; 1760, 0, Today, 20:33:23; 3177, 0, Today, 20:33:23. To the right of the table are sections for 'Collect data' (with a note to connect a device) and 'RAW DATA' (showing a thumbnail of a woman's face labeled 5506). A 'Metadata' section below says 'No metadata.'

# 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!

Upload data x

You can upload CBOR, JSON, CSV, WAV, JPG, PNG, AVI or MP4 files. You can also upload an annotation file named "info.labels" with your data to assign bounding boxes, labels, and/or metadata. View [Uploader docs](#) to learn more. Alternatively, you can use our [Python SDK](#) to programmatically ingest data in various formats, such as pandas or numpy.

For CSV files, [configure the CSV Wizard](#) to define how your files should be processed before uploading files.

Upload mode

Select individual files ②

Select a folder ②

Select files

No file chosen Job completed

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

Upload into category

Automatically split between training and testing ②

Training

Testing

Label

Infer from filename ②

Leave data unlabeled ②

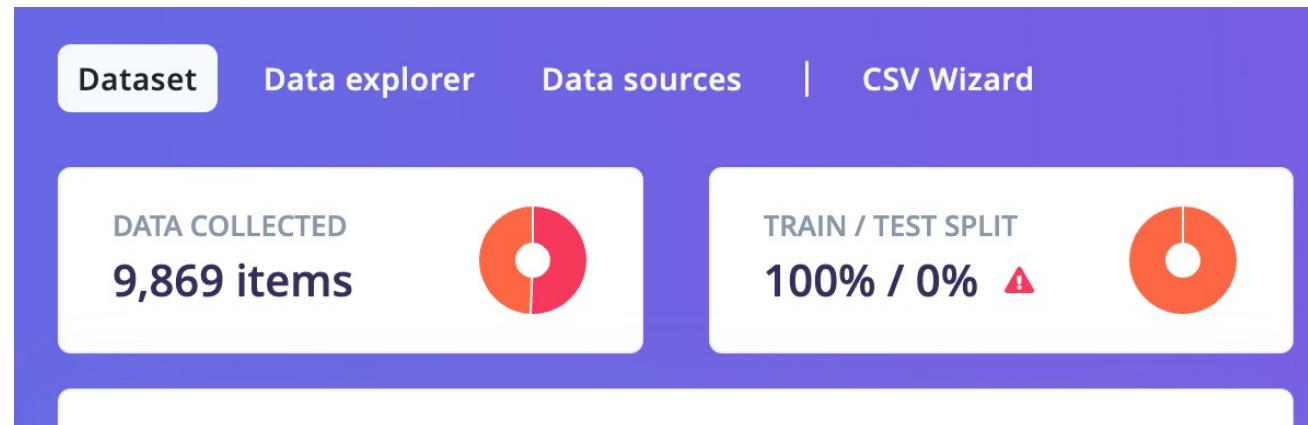
Enter label:

**1 means with a mask**

# 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%)

Dataset train / test split ratio

Training data is used to train your model, and testing data is used to test your model's accuracy after training. We recommend an approximate 80/20 train/test split ratio for your data for every class (or label) in your dataset, although especially large datasets may require less testing data.

SUGGESTED TRAIN / TEST SPLIT 80% / 20%

Labels in your dataset ②

Some classes have a poor train/test split ratio: 0, 1. To fix this, add or move samples to the training or testing data.

Label	Train (%)	Test (%)	Count
0	100%	0%	(5,000 / 0)
1	100%	0%	(4,869 / 0)

Perform train / test split

Use this option to rebalance your data, automatically splitting items between training and testing datasets.

Warning: this action cannot be undone.

Perform train / test split

Dismiss



### Confirm

Enter "perform split" to continue

perform split

Cancel

Perform train / test split

Performed train / test split

Dataset Data explorer Data sources | CSV Wizard

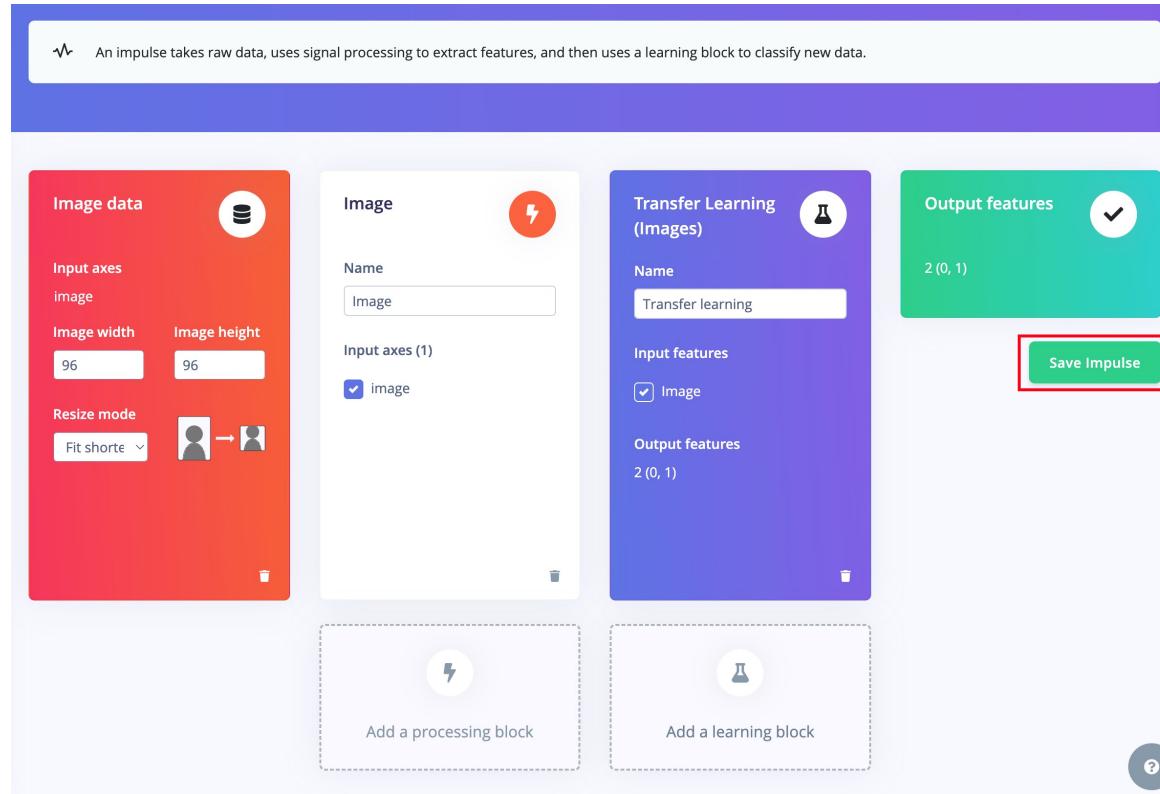
DATA COLLECTED 9,869 items

TRAIN / TEST SPLIT 80% / 20% ②

Collect data

# Lab 4 - Software

## 4. Create a training pipeline (aka Impulse Design)



Data acquisition

Impulse design

Create impulse

# Lab 4 - Software

## 5. Generate Features

#1▼ Click to set a description for this version

Parameters **Generate features**

**Raw data**

Training set

Data in training set	7,876 items
Classes	2 (0, 1)

**Generate features**

# Lab 4 - Software

## 6. Start transfer learning

Neural Network settings

Training settings

Number of training cycles ② 20

Use learned optimizer ②

Learning rate ② 0.0005

Training processor ② CPU

Data augmentation ②

Advanced training settings

Validation set size ② 20 %

Split train/validation set on metadata key ②

Batch size ② 32

Auto-weight classes ②

Profile int8 model ②

Neural network architecture

Input layer (27,648 features)

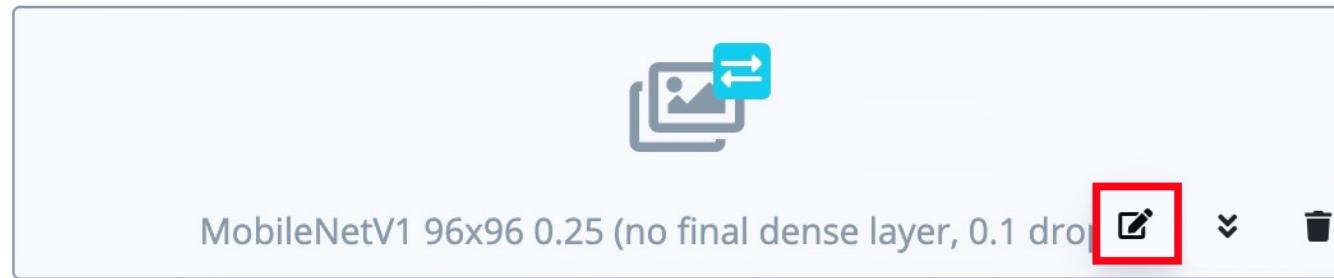
Choose a different model

Did you know? You can customize your model through the Expert view (click on ⚙ to switch), or can even bring your own model (in PyTorch, Keras or scikit-learn).

MODEL	AUTHOR
MobileNetV1 96x96 0.25 OFFICIALLY SUPPORTED A pre-trained multi-layer convolutional network designed to efficiently classify images. Uses around 105.9K RAM and 301.6K ROM with default settings and optimizations.	Edge Impulse <button>Add</button>
MobileNetV1 96x96 0.2 OFFICIALLY SUPPORTED Uses around 83.1K RAM and 218.3K ROM with default settings and optimizations. Works best with 96x96 input size. Supports both RGB and grayscale.	Edge Impulse <button>Add</button>
MobileNetV1 96x96 0.1 OFFICIALLY SUPPORTED Uses around 53.2K RAM and 101K ROM with default settings and optimizations. Works best with 96x96 input size. Supports both RGB and grayscale.	Edge Impulse <button>Add</button>
MobileNetV2 96x96 0.35 OFFICIALLY SUPPORTED Uses around 296.8K RAM and 575.2K ROM with default settings and optimizations. Works best with 96x96 input size. Supports both RGB and grayscale.	Edge Impulse <button>Add</button>

# Lab 4 - Software

## 6. Start transfer learning



### Number of neurons

Enter the new number of neurons for the model's final dense layer. Has a significant impact on the model's size. Reduce this value if the model seems to be overfitting, or increase if the model is underfitting. Set to 0 to omit the layer entirely.

128

Cancel

OK

# Lab 4 - Software

## 6. Start transfer learning

Converting TensorFlow Lite float32 model...

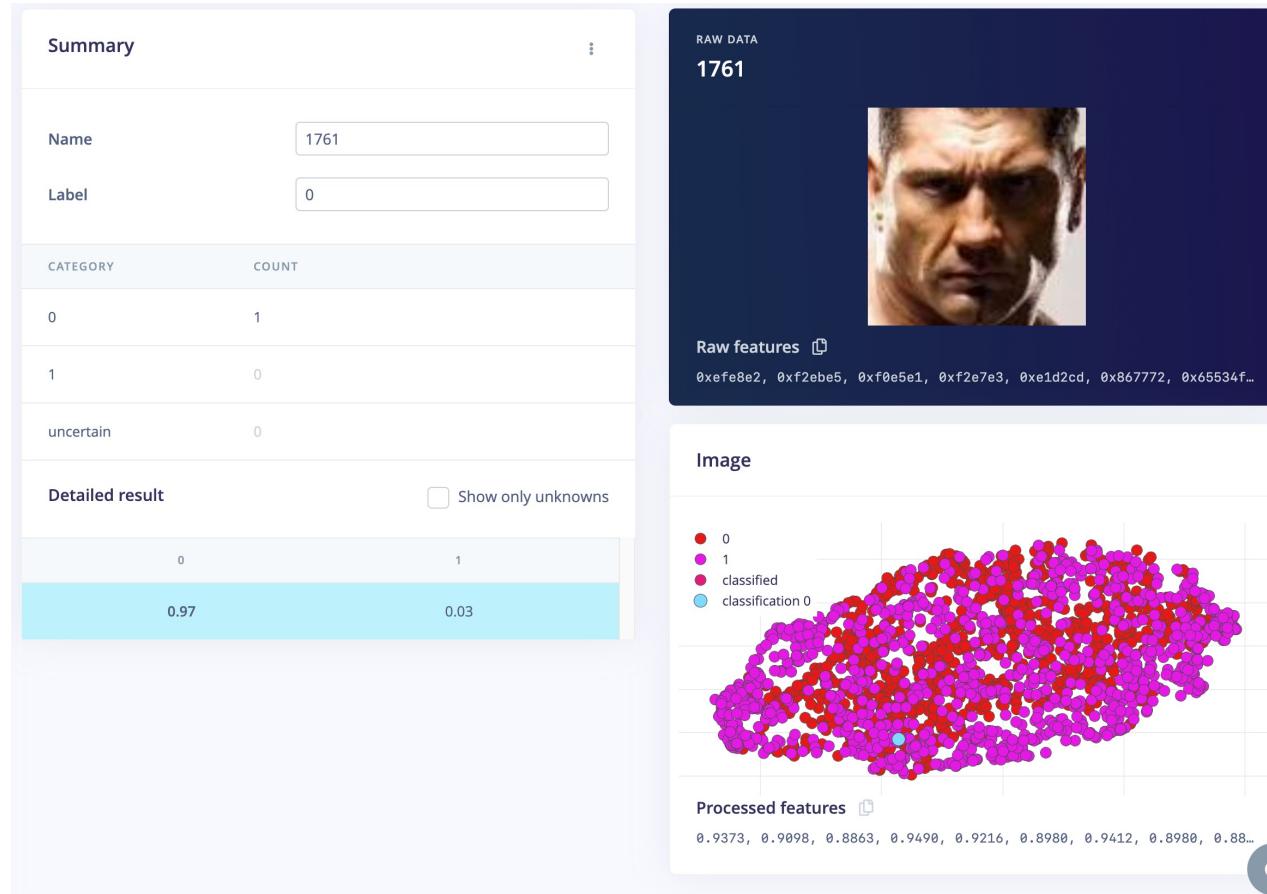
Converting TensorFlow Lite int8 quantized model...

Peak RAM Should < 256 KB



# Lab 4 - Software

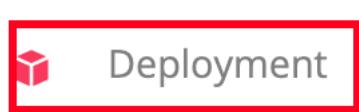
## 7. Play with Live Classification



# Lab 4 - Software

## 8. Save the model

### Versioning



**Configure your deployment**

You can deploy your impulse to any device. This makes the model run without an internet connection, minimizes latency, and runs with minimal power consumption. [Read more](#).

Search deployment options

**C++ library**  
A portable C++ library with no external dependencies, which can be compiled with any modern C++ compiler.

**Arduino library**  
An Arduino library with examples that runs on most Arm-based Arduino development boards.

**BrainChip MetaTF Model**  
A MetaTF converted model (.fbz) for use with the BrainChip Akida™

**EON™ Compiler**  
Same accuracy, 18% less RAM, 14% less ROM.

**EON™ Compiler (RAM optimized)** ENTERPRISE  
Same accuracy, 25% less RAM.

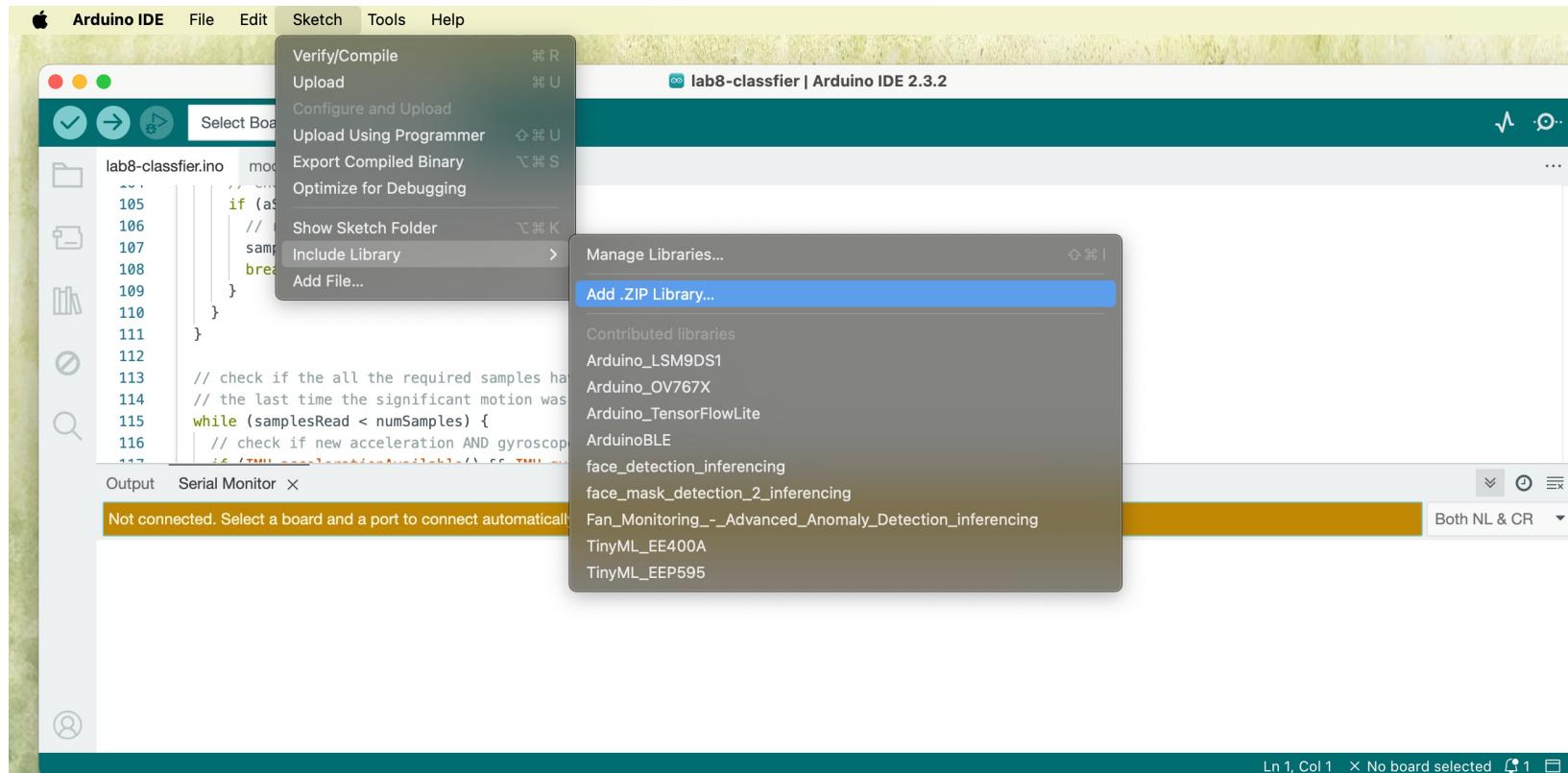
**EON™ Compiler**  
Same accuracy, 18% less RAM, 14% less ROM.

**TensorFlow Lite**

Want access to all features and unlimited projects? Try Enterprise Free!

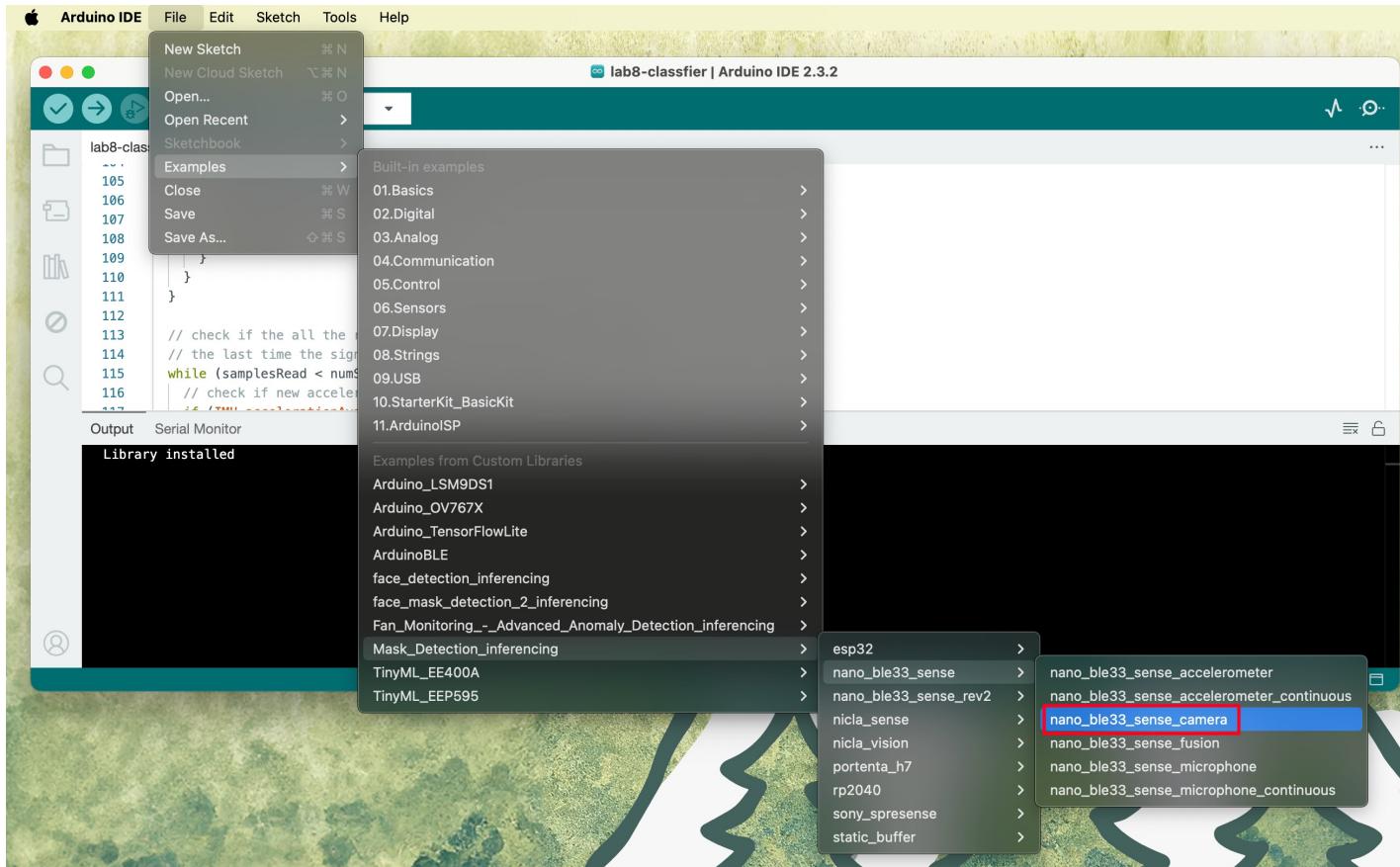
# Lab 4 – Hardware

## 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

