Course notes: HarvardX - TinyML3

Application of TinyML

Contents

1 Welcome to Applications of TinyML 6
1.1 Objectives 6
1.2 Prerequisites 6
1.3 Who's Who?! 6
1.4 What Resources are Needed for this Course2? 7
1.5 The Role of Sensors in TinyML Applications 7
1.6 The Kit for Course 3 9
2 AI Lifecycle and ML Workflow 10
2.1 ML Lifecycle 10
2.2 ML Workflow Details 12
3 Machine Learning on Mobile and Edge IoT Devices 13
3.1 TensorFlow: Where We Left Off 13
3.1.1 Recap of the Machine Learning Paradigm 13
3.2 Using the TFLite Converter in Colab 15
3.3 How to use TFLite Models 15
3.4 Running Models with TFLite in Colab 17
3.5 TFLite Optimizations and Quantization in Colab 17
3.6 Quantization Aware Training Colab 17
3.7 Assignment: Quantization in TFLite 18
3.7.1 Assignment Solution 18
4 Machine Learning on Mobile and Edge IoT Devices - Part 2 19
4.1 Why are 8-Bits Enough for ML? 19
4.1.1 Why does Quantization Work? 19
4.1.2 Why Quantize? 20
4.2 PTQ Weight Distribution 20
4.3 Conversion and Deployment 20
4.3.1 Tensorflow's Computational Graph 21
4.3.2 The benefit of using such a computational graph is twofold: 21
4.3.3 Tensorflow Checkpoints 21
4.3.4 Freezing a Model 22
4.3.5 Optimizing a Model (Converting to TensorFlow Lite) 22
4.4 Quiz 23
5 Key Word Spotting 23
5.1 How (else) would you use KWS? 23

5.2 Keyword Spotting Application Architecture Overview 23
5.3 Wake Words Dataset Creation 25
5.4 Spectrograms and MFCCs 26
5.5 Keyword Spotting in Colab 28
5.6 Training in Colab 29
5.6.1 Timeouts 29
5.6.2 File Structure 29
5.6.3 The Training Script 29
5.7 Monitoring Training in Colab 30
5.8 Assignment: Training your own Keyword Spotting Model 32
5.9 Assignment Solution 32
5.10 Keyword Spotting in the Big Picture 33
6 Introduction to Data Engineering 34
6.1 What is Data Engineering? 34
6.2 What's in this Module? 34
6.3 Your New TinyML Applications 35
6.4 Dataset Standards: Speech Commands 35
6.5 Speech Commands Paper 40
6.6 Crowdsourcing Data for the Long Tail 40
6.7 Giving Back to the Open Source Community 47
6.8 Reusing and Adapting Existing Datasets 47
6.9 Responsible Data Collection 49
6.10 Summary 51
6.10.1 Summary: Data Engineering 51
7 Visual Wake Words 53
7.1 Introduction to Visual Wake Words (VWW) Application 53
7.1.1 What's the Focus in this Module? 53
7.1.2 What's New and Different? 53
7.2 What are Visual Wake Words (VWW)? 54
7.3 Visual Wake Words Challenges 57
7.4 Data Privacy with Images 63
7.4.1 A Brief History of Data Privacy 63
7.4.2 Data Privacy in Images 64
7.4.3 Relevance to TinyML 64
7.5 Neural Network Architectures for Visual Wake Words 65

7.6 The Math Behind MobileNets Efficient Computation 74
7.6.1 Standard Convolutions 74
7.6.2 Depthwise Separable Convolutions 75
7.7 Transfer Learning for VWW 76
7.8 Assignment: Transfer Learning in Colab 80
7.8.1 Assignment Solution 80
7.9 Common Myths and Pitfalls about Transfer learning 80
7.9.1 Task Similarity 81
7.9.2 Fragile Co-Adaptation 81
7.9.3 Fixed Architecture 81
7.10 Metrics for VWW 82
7.11 Summary Reading 87
7.11.1 Challenges 87
7.11.2 Datasets 87
7.11.3 MobileNets 87
7.11.4 Transfer Learning 87
7.11.5 Metrics 88
8 Anomaly Detection 88
8.1 Introduction to Anomaly Detection 88
8.1.1 What's the Focus in this Module? 88
8.2 What Is Anomaly Detection 89
8.3 Anomaly Detection in Industry 94
8.4 Industry 4.0 and TinyML 99
8.4.1 What is Industry 4.0? 99
8.4.2 Where does TinyML fit in? 100
8.5 Anomaly Detection Datasets 101
8.6 MIMII DATASET 105
8.7 Real and Synthetic Data 105
8.7.1 What is Synthetic Data? 105
8.7.2 Quality Concerns of Synthetic Data 106
8.8 Unsupervised Learning for Anomaly Detection (with K-Means) 106
8.9 Unsupervised Learning for Anomaly Detection with Autoencoders 107
8.10 Autoencoder Model Architecture 111
8.10.1 Types of Autoencoders 111
8.10.2 Fully Connected Autoencoder 111

8.10.3 Convolutional Autoencoder 112
8.10.4 LSTM Autoencoders 112
8.10.5 Variational Autoencoders 113
8.10.6 Visualizing the Latent Space of Variational Autoencoders 114
8.11 Training and Metrics 115
8.12 Picking a Threshold 116
8.13 Assignment: Training an Anomaly Detection Model 116
8.13.1 Assignment Solution 116
8.14 Summary Reading 117
8.14.1 Anomaly Detection in Industry 117
8.14.2 Data and Datasets 117
8.14.3 Unsupervised Learning: K-Means and Autoencoders 117
9 Responsible AI Development 118
9.1 Data Collection 118
9.2 The Many Faces of Bias in ML 122
9.3 Biased Datasets 124
9.4 Forum 131
9.5 Fairness 131
9.6 Google's What-If Tool 138
10 Summary 139
10.1 Summary 139
10.2 Kit for Course 3 150

1 Welcome to Applications of TinyML

1.1 Objectives

The goal of this course is to focus on applications, data and neural networks on tiny or deeply embedded devices. We will expose the students to embedded devices and different real-world application scenarios of TinyML. We do this by covering the most widely used applications for TinyML, coupled with some hands-on Co-Lab development. We will be focused on the neural network portion of the applications, looking at training and deployment, leaving pre and post-processing implementations and other systematic questions to the third course.

1.2 Prerequisites

Fundamentals of TinyML or equivalent ML experience

Basic Scripting in Python

Basic usage of Colab

1.3 Who's Who?!

Hi!

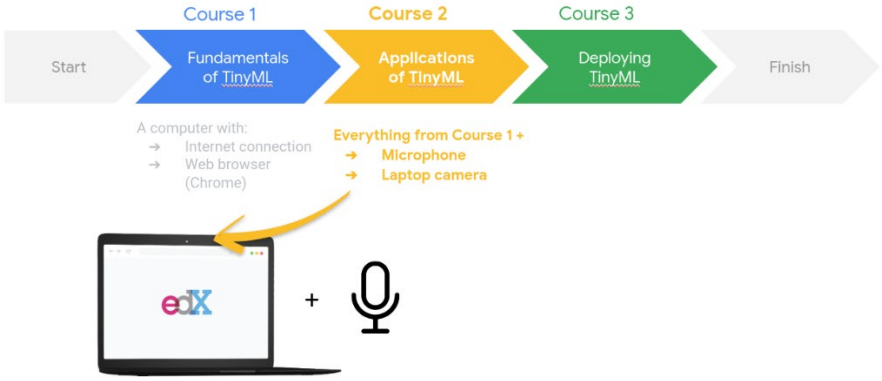
I'm Vijay Janapa Reddi, and I'm your instructor. I'm a professor at Harvard, and I am thrilled that you can join us (again) in Course 2. I thought I'd take a moment to introduce myself and the staff that work hard behind the scenes to produce the rich content we've put together as part of this unique HarvardX course that brings together machine learning and embedded systems. Laurence, Pete, and I are your lead instructors for this course, but in addition to us who you will hear from in Course 2 and 3, several folks are working behind the scenes to produce this content-rich HarvardX course for you! Dr. Susan Kennedy will teach you about Responsible AI design for TinyML systems, which is emerging as an important topic in the broad field of machine learning applications. We are all delighted that you are on board this journey with us, as we share several exciting aspects of TinyML.

Also if you'd like to join the TinyML conversation outside of the class check out our Discourse Forum! Its a great place to share TinyML ideas and build a network of colleagues for exciting TinyML projects!

Sincerely

Vijay.

1.4 What Resources are Needed for this Course2?



In Course 2, you will only need a laptop with a web browser (we suggest Google Chrome for the best performance using Google Colab), a microphone, a camera, and an internet connection. All of the model training and testing will be done in Colab! We hope this makes the course accessible to everyone!

1.5 The Role of Sensors in TinyML Applications

The Role of Sensors in TinyML Applications

Sensors are important for machine learning applications as they act as a vehicle between algorithmic resources and characteristics of the physical environment. Today there is a continued push towards low-cost and high-performance sensors, especially for applications such as air quality monitoring and predictive maintenance. There are two main types of sensing: in situ and remote sensing.

In situ sensors come in numerous forms and each have a particular sensing mechanism. For example, the sensing mechanism may involve chemical reactions, light, or measuring voltages from material changes caused by stress. In situ sensors are often characterized as physical sensors and are constrained to measure local attributes at the location of the sensing element. Remote sensing methods work at a range, and are almost always optical techniques. For example, temperature can be measured at a distance using an infrared thermometer, and gas concentrations in the atmosphere can be measured by columnar measurements made by satellites in low Earth orbit. In situ sensors can use optical mechanisms, but are often distinguished because they only measure local attributes.

In TinyML, we are typically interested in sensor data in the form of an image or time series, enabling us to perform spatial or time series analysis using machine learning models. The utility of sensors for tiny machine learning allows real-time feedback and monitoring of the local environment, either at the point of the sensing element or at a distance using remote sensing methods. For example, an array of in situ sensors each with gas sensing elements can be used to monitor pollution levels across a university campus, or surrounding an industrial facility. Similarly, an example of a remote sensing method might be a camera used to monitor local weather conditions based on the appearance of overlying clouds - perhaps useful in locations with limited internet access.

For a sensor outputting a time series, the sensor measurement can be used as a single feature in a machine learning model. A segment of the time series might also be used for classification or anomaly detection purposes. For an image-based method, the bitmap can be flattened and each pixel passed as a feature in a neural network.

Sensors can be used individually or can be combined to produce synergistic effects. This provides even more powerful capabilities, especially when combined with online machine learning. Perhaps the most utilized example of this for TinyML currently is the combination of an accelerometer and gyroscope. These devices are often present in smartphones and other embedded devices, and can be used together to differentiate between complex 6-axis movements. These allow your smartphone to know when it has been picked up, or to know when the holder has fallen over. They can also be used in more complex devices such as drones to estimate the local wind speed and direction via how it perturbs from normal operation. Other sensors apart from accelerometers and gyroscopes are also common. Vibration sensors are becoming increasingly common in industrial applications for predictive maintenance. These sensors are used with the aim to improve machinery uptime by detecting anomalous vibrations and repairing them before a catastrophic failure occurs.

Using sensors for embedded machine learning presents several additional challenges. Firstly, sensors have a finite working life, and the designer should bear this in mind during development. For example, we might not want a complex sensing system to become unusable if a single sensor becomes defective, and we might want it to default to a specific action in the event of a failure. Sensors also tend to exhibit noise and drift, which is often infeasible to remove or model statistically. TinyML systems should be robust enough that they are not adversely impacted by this noise or drift during regular operations. In a similar vein, sensors that are not properly isolated electrically can have the potential to distort other measurements. Finally, the more sensors that are present in a given system, the more complex and power-hungry it becomes. To be an adept user of TinyML, it is not enough to know about machine learning and embedded systems, but also mindful of the constraints imposed by electrical and mechanical engineering challenges.

To overcome the above challenges, the following rules of thumb are recommended:

1 - Collect rich data. In the machine learning world, data is power. A sufficiently large amount of data should be collected to ensure that our TinyML models are able to effectively approximate the distribution of the data. More data never hurts!

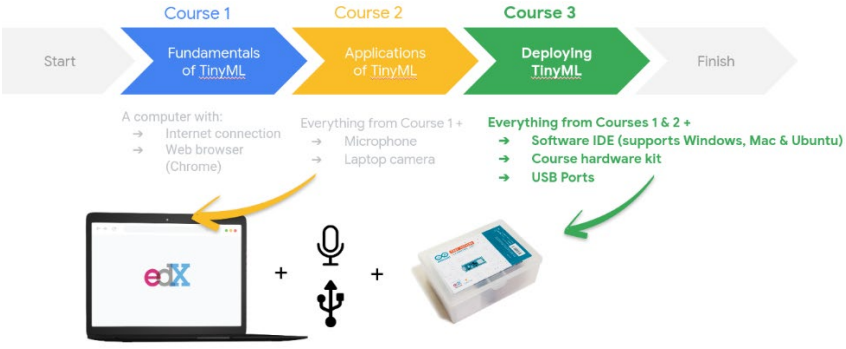
2 - Don’t over-engineer your system. While it may seem attractive to add fifteen sensors to a system, we should follow Occam’s razor when engineering our system that “entities should not be multiplied without necessity”. Being intelligent and yet parsimonious with our sensor selection will minimize the possibility of running into hardware complications.

3 - Plan to cover all possible sources of variation. The statistical power of our model will always be limited by the data at hand. Consequently, it is helpful to cover as much ground as possible without conflicting with #2. This can involve collecting data that covers edge cases as well as using additional sensors that are expected to provide some additional explanatory power to the model.

4 - Use the highest workable sampling rate. Although it is highly expensive to transmit and store high-frequency data, we limit these issues when we are using TinyML systems since inferences are made on-device and streamlined to minimize storage. It is much easier to downsample data when there is too much than to upsample it when there is not enough. Thus, when collecting data to use for model training, it is recommended to obtain the highest resolution data possible to provide flexibility in downstream data engineering processes.

Regardless of whether you are using sounds, images, vibrations, electrical signals, or other sensor data, these signals can be combined and used to train machine learning models to help model, classify, or predict events at the edge in real-time using inexpensive microcontrollers, which is a very powerful tool indeed.

1.6 The Kit for Course 3



In Course 3, you will need everything from Course 2, a laptop with a microphone and camera and an internet connection, as well as the course kit as we will be deploying our models onto microcontrollers. You can buy the course kit directly from Arduino at this link:

https://store.arduino.cc/usa/tiny-machine-learning-kit

The course kit includes a proprietary shield which will make the camera module much more reliable. However, if you would like to purchase individual components instead of the full kit you will need to purchase the following components:

Arduino Nano BLE 33 Sense WITH HEADERS

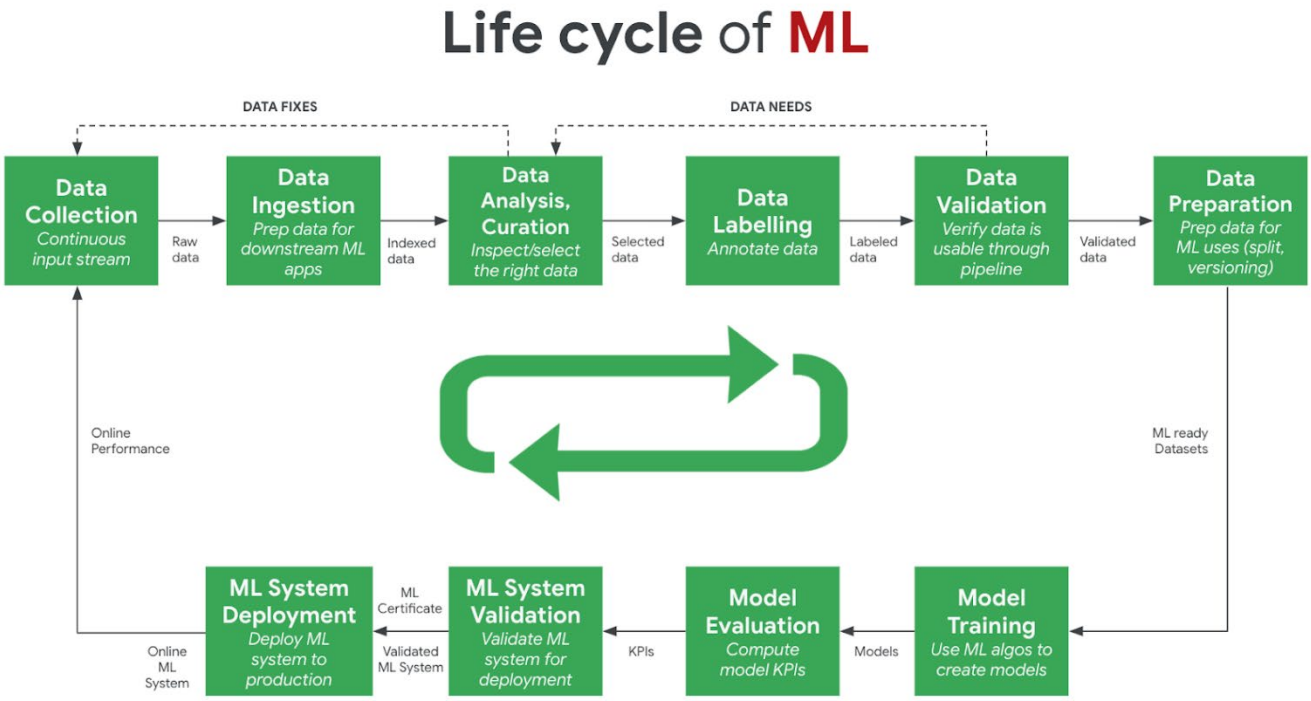
ArduCam OV7675

Jumper wires and optionally a breadboard for connecting the camera to the Arduino

Keep in mind that if you don't use the Arduino kit, we have found that the camera connection is often unreliable over jumper wires, and it will be hard for your fellow classmates and others to provide guidance if you experience any issues.

2 AI Lifecycle and ML Workflow

2.1 ML Lifecycle



The development of a machine learning model follows a multi-step design methodology, commonly referred to as the machine learning lifecycle. Diagrams typically illustrate this lifecycle using a varying set of discrete steps. This methodology is also applicable to tiny machine learning applications, albeit with different requirements to conventional machine learning models. Some of the most important steps in the machine learning lifecycle are as follows:

Design Requirements. The first stage of the machine learning lifecycle is where the application requirements are established. For tiny machine learning applications, hardware constraints (e.g., memory and storage footprint, latency) are often the most pressing requirements. Other requirements can include temporal resolution of data, the number of inferences per second, or minimum model accuracy. This step, whilst being the least well-defined, is the most crucial step, as it sets the scene for the remainder of the lifecycle. The more clearly defined the design criteria are, the easier it is to evaluate the model in later stages. Data requirements, such as the variable of interest and potential feature variables, can also be specified during this stage.

Data Collection/Curation. In this stage, design requirements are known and data begin to be collected that are suspected to aid in predicting the variable of interest. This can often involve active data collection or curation of information from external sources, such as pre-existing datasets (e.g., CIFAR10, ImageNet). This is often one of the most time consuming stages, as it can be difficult and time-consuming to curate a sufficiently large dataset, simultaneously ensuring to minimize dataset bias. No data analysis is performed during this stage. In Course 2, we will learn the challenges associated with data collection for different types of sensors that serve as inputs for tiny machine learning applications.

Exploratory Data Analysis/Data Preprocessing. During this phase, the collected data is analyzed to determine which features are most informative at predicting the variable of interest. Feature engineering is common at this stage, extracting new information from available data. An example of this would be to extract the day of the week or whether it is a weekend from a time variable. Preprocessing involves ensuring the data follows standard modeling assumptions (e.g., are normally distributed) or manipulating data to meet these assumptions (e.g., log transformations on highly skewed variables). Data imputation is also typically done to appropriately fill in or remove missing data, and invalid or duplicate data is handled accordingly. When we develop our keyword spotting model, we will learn how to pre-process the audio signal.

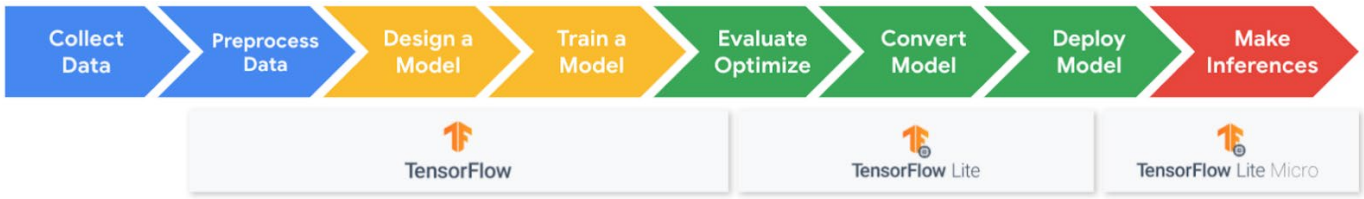
Model Development. All of the previous stages focused on the design aspects, as well as the procurement and analysis of data. These stages often make up the dominant proportion of the time in the machine learning lifecycle. Once a viable dataset is available, the next stage is creating a suitable machine learning model. There are a plethora of machine learning techniques that have various pros and cons. One of the jobs of a machine learning practitioner is selecting an appropriate model for the task at hand. For example, basic linear regression may be suitable in environments where interpretability is key, but are constrained to the set of linear functions. This may present bias in situations where the distribution of data is highly non-linear in feature space, which can result in poor model accuracy. Conversely, neural networks may offer high performance as they are able to model the non-linear data distribution more effectively, but are less interpretable to the user. Keeping in mind the design requirements is important during this stage to know on what grounds to evaluate different models. One thing we will do in Course 2 is to understand the different model development approaches. Training a neural network from scratch is not always necessary.

Model Validation. It is commonplace to generate multiple models and to compare their performance on an unseen data set, known as the test set. The presence of a test set helps to ensure that the model has effectively modeled the distribution of data and has not overfit to points in the training set. The design criteria help to determine what metrics should be used in comparing the performance of various models. For example, if accuracy is the dominant metric, the model with the highest accuracy should be chosen. This stage can also be used to determine whether specific features improved or hindered model performance (e.g., by analyzing the relative importance of features or by successively removing variables and retraining the model, known as ablation study). But there is more to model validation than just looking at the model metrics. We will discover how to think about metrics in a new light, specifically, from an application deployment perspective.

Model Deployment. The last stage of the lifecycle involves the deployment of the model in the production environment. For tiny machine learning applications, this is one of the most important stages. The model size must be reduced sufficiently to fit on the embedded device through various means such as model compression (i.e., model distillation), type conversion (e.g.., changing model weights from floats to integers), and lossless compression (e.g., Huffman encoding). The model must then be compiled into a format compatible with the end device. We will learn about “quantization” and use it to optimize the model size for performance and latency to make sure we are building sufficiently ‘tiny’ models. We will use TensorFlow’s quantization API, as well as understand what’s going on behind the scenes.

After all of these stages, continuous monitoring of the model in the production environment is typically necessary to ensure that it is working effectively. If, after all of these steps, the performance of the model is unsatisfactory in the production environment, the model must be updated. This can be done using new design requirements, data, or modeling techniques. This is the essence of the machine learning lifecycle.

2.2 ML Workflow Details



TinyML Application Pipeline

The figure above shows the workflow methodology we will be following for building the tinyML applications in this course. These are broken down into three main stages:

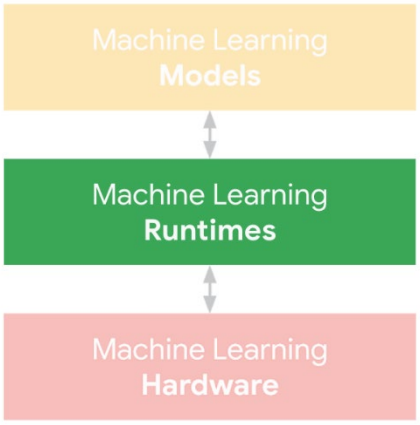
Step 1: Collect & Preprocess Data

Step 2: Design and Train a Model

Step 3: Evaluate, Optimize, Convert and Deploy Model

What's Underneath the Hood?

Often, when learning about machine learning, we forget that models are just a piece of the bigger picture. There is more to machine learning than just the models. These are the underlying ML runtimes and hardware that enable these models to run efficiently. The ML runtimes we will be dealing with are TensorFlow, TensorFlow Lite, and TensorFlow Lite Micro.



One of the goals of our course is to dive into understanding the fundamental differences between machine learning runtimes so that we can deploy the models we prepare efficiently. To this end, it is important to understand the various frameworks we use across all three applications in this course: Keyword Spotting, Visual Wake Works, and Anomaly Detection.

So coming up next is material that highlights the fundamental differences between TensorFlow and TensorFlow Lite (for microcontrollers). You will learn not only the high-order bits but also the mechanics that implement the API calls you learn to make with the code.

3 Machine Learning on Mobile and Edge IoT Devices

3.1 TensorFlow: Where We Left Off

Before we dive into our TinyML applications' underlying mechanics, you must understand the software ecosystem that we will be using. The software that wraps around the models we train is just as important as the models themselves. Imagine working hard to train a tiny machine learning model that occupies only a few kilobytes of memory, only to realize that the TensorFlow framework used to run the model is itself several megabytes large.

To avoid such oversight, we first introduce the fundamental concepts around TensorFlow's usage versus TensorFlow Lite (and soon TensorFlow Lite Micro). We will be using TensorFlow for training the models. We will be using TensorFlow Lite for evaluation and deployment because it supports the optimizations we need. It has a minimal memory footprint, which is especially essential for deployment in mobile and embedded systems. To this end, in the following two modules, Laurence and I are going to first introduce the concepts, programming interfaces, and the mechanics behind them before diving deep into the applications. Every application reuses these fundamental building blocks.

3.1.1 Recap of the Machine Learning Paradigm

In the previous course you had an introduction to machine learning, exploring how it is a new programming paradigm that changes the programming paradigm. Instead of creating rules explicitly with a programming language, the rules are learned by a neural network.

Using TensorFlow you could create a neural network, compile it and then train it, with the training process, at a high level looking like this:

The neural network would randomly initialize, effectively making a guess to the rules that match the data to the answers, and then over time it would loop through measuring the accuracy and continually optimizing.

An example of this type of code is seen here:

```
tensorflow
data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()
training_images  = training_images/255.0
val_images = val_images /255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28x28)),
tf.keras.layers.Dense(20, activation=tf.nn.relu),
tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

model.compile (optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=20 validation_data=(val_images, val_labels))
```

In the center is the architecture for saving a model, called TensorFlow SavedModel. You can learn more about it at:

https://www.tensorflow.org/guide/saved_model

On the right are the ways that models can be deployed.

The standard TensorFlow models that you've been creating this far, trained without any post-training modification can be deployed to Cloud or on-Premises infrastructures via a technology called TensorFLow Serving, which can give you a REST interface to the model so inference can be executed on data that's passed to it, and it returns the results of the inference over HTTP. You can learn more about it at

https://www.tensorflow.org/tfx/guide/serving

TensorFlow Lite is a runtime that is optimized for smaller systems such as Android, iOS and embedded systems that run a variant of Linux, such as a Raspberry Pi. You'll be exploring that over the next few videos. TensorFlow Lite also includes a suite of tools that help you convert and optimize your model for this runtime. https://www.tensorflow.org/lite

TensorFlow Lite Micro, which you'll explore later in this course, is built on top of TensorFlow Lite and can be used to shrink your model even further to work on microcontrollers and is a core enabling technology for TinyML.

https://www.tensorflow.org/lite/microcontrollers

TensorFlow.js provides a javascript-based library that can be used both for training models and running inference on them in JavaScript-based environments such as the Web Browsers or Node.js.

https://www.tensorflow.org/js

Let's now explore TensorFlow Lite, so you can see how TensorFlow models can be used on smaller devices on our way to eventually deploying TinyML to microcontrollers in Course 3. Over the rest of this lesson we'll be exploring models that will generally run on devices such as Android and iOS phones or tablets. The concepts you learn here will be used as you dig deeper into creating even smaller models to run on tiny devices like microcontrollers.

3.2 Using the TFLite Converter in Colab

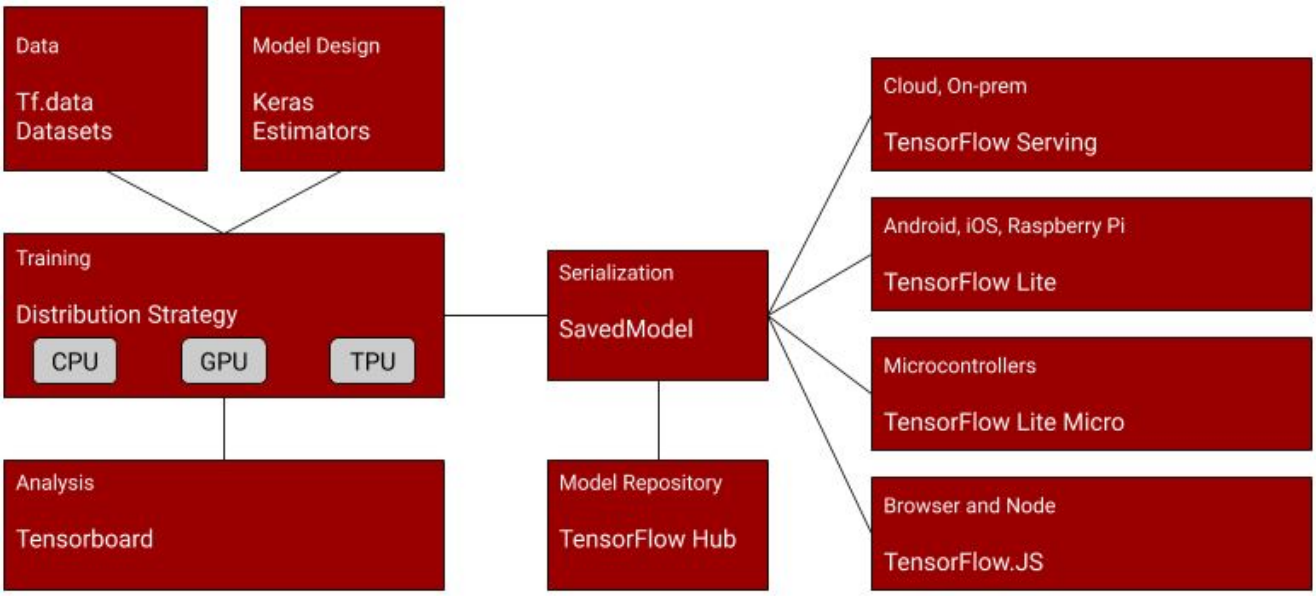
Now that you've seen how we can use the converter to create a TFLite model, you'll get a chance to explore the code for yourself in this next Colab. Try to take your time and walk through the code to make sure you understand the Converter!

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-3-4-TFLiteConverter.ipynb

3.3 How to use TFLite Models

Previously you saw how to train a model and how to use TensorFlow's Saved Model APIs to save the model to a common format that can be used in a number of different places.

Recall this architecture diagram:



So, after training your model, you could use code like this to save it out:

```
export_dir = 'saved_model/1'
tf.saved_model.save(model, export_dir)
```

This will create a directory with a number of files and metadata describing your model. To learn more about the SavedModel format, take a little time now to read https://www.tensorflow.org/guide/saved_model , and also check out the colab describing how SavedModel works at

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/guide/saved_model.ipynb

, and in particular explore 'The SavedModel format on disk' section in that colab.

Once you had your saved model, you could then use the TensorFlow Lite converter to convert it to TF Lite format:

```
converter =tf.lite.TFLiteConverter.from_saved_model(export_dir)
tflite_model = converter.convert()
```

This, in turn could be written to disk as a single file that fully encapsulates the model and its saved weights:

```
import pathlib
tflite_model_file = pathlib.Path('model.tflite')
tflite_model_file.write_bytes(tflite_model)
```

To use a pre-saved tflite file, you then instantiate a tf.lite.Interpreter, and use the 'model_content' property to specify an existing model:

```
interpreter = tf.lite.Interpreter(model_path=tflite_model_file)
```

Once you've loaded the model you can then start performing inference with it. Do note that to run inference you need to get details of the input and output tensors to the model. You'll then set the value of the input tensor, invoke the model, and then get the value of the output tensor. Your code will typically look like this:

```
# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
to_predict = # Input data in the same shape as what the model expects
interpreter.set_tensor(input_details[0][], to_predict)
tflite_results = interpreter.get_tensor(output_details[0]['index'])
```

...and a large part of the skills in running models on embedded systems is in being able to format your data to the needs of the model. For example, you might be grabbing frames from a camera that has a particular

resolution and encoding, but you need to decode them to 224x224 3-channel images to use with a common model called mobilenet. A large part of any engineering for ML systems is performing this conversion.

To learn more about running inference with models using TensorFLow Lite, check out the documentation at:

https://www.tensorflow.org/lite/guide/inference#load_and_run_a_model_in_python

3.4 Running Models with TFLite in Colab

Now that you've seen how to test TensorFlow Lite models in Colab you'll get a chance to explore this yourself! Make sure to start to pay attention to the file structure of Colab as you'll need to understand how to upload and download data and models from Colab later in the course.

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-3-7-RunningTFLiteModels.ipynb

3.5 TFLite Optimizations and Quantization in Colab

In this next Colab you will get to explore quantization in TFLite through a hands-on example of recognizing pictures of cats vs. dogs. We will explore both how the model size changes and how the accuracy changes. You'll also get to look at individual predictions of the images to see which pictures it gets correct and incorrect!

Note: Changes have been made to the TFLite Interpreter since Laurence filmed the previous video that further optimize it for mobile use at the expense of speed in Colab. As such, you'll find that while Laurence was able to achieve speeds of 16 iterations per second for Model 2 in the previous video, you may only see speeds of 1-2 iterations per second.

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-3-10-TFLiteOptimizations.ipynb

3.6 Quantization Aware Training Colab

There are two primary forms of quantization -- post training quantization that you had seen previously -- where, as part of the conversion process, your model's internal weights and ops get converted to int8 and uint8. It's much easier to use this, and it's a good way to get started.

As you want to further optimize your model, you may want to explore quantization aware training. You got a taste of that in the previous video seeing how you could pass your simple model to an api and get a quantized model back. It turns out that there is a lot more fine-grained control available to you. To get some exposure to some of those controls, please work on the below colab provided by the TensorFlow team on Quantization-aware Training (QAT). After you are done, if you would like to go deeper, including a comprehensive guide on all the APIs available in the toolkit, check out and read through the model optimization site on TensorFlow.org.

In particular, note the results posted by the Google teams when comparing accuracy of models before and after quantizing like this -- the effects on accuracy are negligible!

Model	Non-quantized Top-1 Accurac	>8-bit Quantized Accuracy
MobilenetV1 224	71.03%	71.06%
Resnet v1 50	76.3%	76.1%
MobilenetV2 224	70.77%	70.01%

Colab link for QAT for you to work through:

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-3-12-QAT.ipynb

3.7 Assignment: Quantization in TFLite

Now that we've explored how to do quantization in TFLite, it's your turn. In this assignment we are going to quantize a "rock, paper, scissors" hand gesture recognition model. We'll set up the problem for you and you'll be tasked to perform conversion and set up the interpreter for inference.

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-3-14-Assignment.ipynb

3.7.1 Assignment Solution

For this assignment you needed to define the converter, run it and then define the TFLite interpreter.

You can instantiate the converter, leveraging the saved model file as noted in the hint, using the handy saved model function shown below:

```
converter = tf.lite.TFLiteConverter.from_saved_model(ROCK_PAPER_SCISSORS_SAVED_MODEL)
```

You then can run the converter as follows:

```
tflite_model = converter.convert()
```

It's that simple! TFLite does all of the heavy lifting under the hood for you.

Defining the interpreter (again leveraging a saved model -- this time our TFLite model) is also a simple one line command:

```
interpreter = tf.lite.Interpreter(model_path=TFLITE_MODEL_FILE)
```

And that's it! From there you should have been able to test out the model's accuracy and visualize which images it got correct and incorrect. We hope you also took us up on the challenge of improving the model. Let us know in the comments to the Colab what worked best. We're curious to see how accurate you all can make this model! As always you'll find the link to the assignment again below in case you want to explore it a bit more!

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-3-14-Assignment.ipynb

4 Machine Learning on Mobile and Edge IoT Devices - Part 2

4.1 Why are 8-Bits Enough for ML?

Why are 8-Bits are Enough for ML?

When neural networks were first being developed, the biggest challenge was getting them to work at all! That meant that accuracy and speed during training were the top priorities. Using floating-point arithmetic was the easiest way to preserve accuracy, and GPUs were well-equipped to accelerate those calculations, so it's natural that not much attention was paid to other numerical formats.

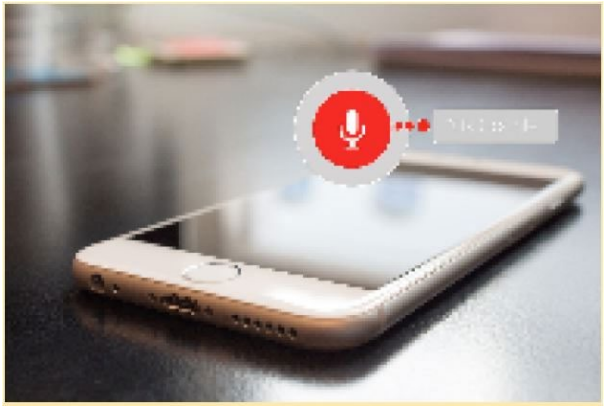
These days, we actually have a lot of models being deployed in commercial applications. The computation demands of training grow with the number of researchers, but the cycles needed for inference expands in proportion to the number of users. That means that inference efficiency has become a burning issue for a lot of teams in organizations that are deploying ML solutions, TinyML included.

That is where quantization comes in. It's an umbrella term that covers a lot of different techniques to store numbers and perform calculations on them in more compact formats than 32-bit floating-point. We are going to focus on an eight-bit fixed point in this course, for reasons we will go into more detail on later.

4.1.1 Why does Quantization Work?

Neural networks are trained through stochastic gradient descent; applying many tiny nudges to the weights. These small increments typically need floating-point precision to work (though there are research efforts to use quantized representations here too), otherwise, you can get yourself into a pickle with such things as "vanishing gradients." Recall that activation functions (Course 1, ding ding ding!) restrict the large range of values into a rather confined numerical representation so that any large changes in the input values do not cause the network to have catastrophically different behavior.

Taking a pre-trained model and running inference is very different. One of the magical qualities of deep networks is that they tend to cope very well with high levels of noise in their inputs. If you think about recognizing an object in a photo you've just taken, the network has to ignore all the CCD noise, lighting changes, and other non-essential differences between it and the training examples it's seen before, and focus on the important similarities instead. This ability means that they seem to treat low-precision calculations as just another source of noise, and still produce accurate results even with numerical formats that hold less information. For instance, does your brain recognize the below picture? The answer has got to be a yes, unless, of course, you weren't watching the videos!



You can run many neural networks with eight-bit parameters and intermediate buffers (instead of full precision 32-bit floating-point values), and suffer no noticeable loss in the final accuracy. OK, so fine, sometimes you might suffer a little bit of loss in accuracy but often the gains you get in terms of performance latency and memory bandwidth are justifiable.

4.1.2 Why Quantize?

Neural network models can take up a lot of space on disk, with the original AlexNet being over 200 MB in float format, for example. Almost all of that size is taken up with the weights since there are often many millions of these in a single model. Because they're all slightly different floating-point numbers, simple compression formats like "zip" don't compress them well. They are arranged in large layers though, and within each layer, the weights tend to be normally distributed within a certain range, for example, -3.0 to 6.0.

The simplest motivation for quantization is to shrink file sizes by storing the min and max for each layer and then compressing each float value to an eight-bit integer representing the closest real number in a linear set of 256 within the range. For example with the -3.0 to 6.0 range, a 0 byte would represent -3.0, a 255 would stand for 6.0, and 128 would represent about 1.5. This means you can get the benefit of a file on disk that's shrunk by 75%, and then convert back to float after loading so that your existing floating-point code can work without any changes.

Another reason to quantize is to reduce the computational resources you need to do the inference calculations, by running them entirely with eight-bit inputs and outputs. This is a lot more difficult since it requires changes everywhere you do calculations, but offers a lot of potential rewards. Fetching eight-bit values only require 25% of the memory bandwidth of floats, so you'll make much better use of caches and avoid bottlenecking on RAM access. You can also typically use hardware-accelerated Single-Instruction Multiple Data (SIMD) operations that do many more operations per clock cycle. In some cases, you'll have a digital signal processor (DSP) chip available that can accelerate eight-bit calculations too, which can offer a lot of advantages. (Don't worry if you don't fully understand these hardware capabilities, we will discuss them more in Course 3.)

Moving calculations over to eight-bit will help you run your models faster, and use less power, which is especially important on mobile devices. It also opens the door to a lot of embedded systems that can't run floating-point code efficiently, so it can enable a lot of applications in the TinyML world.

4.2 PTQ Weight Distribution

In this Colab we are going to explore Post Training Quantization (PTQ) in more detail. In particular we will use Python to get a sense of what is going on during quantization (effectively peeking under the hood of TensorFlow). We will also visualize the weight distributions to gain intuition for why quantization is often so successful (hint: the weights are often closely clustered around 0).

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-4-3-PTQ.ipynb

4.3 Conversion and Deployment

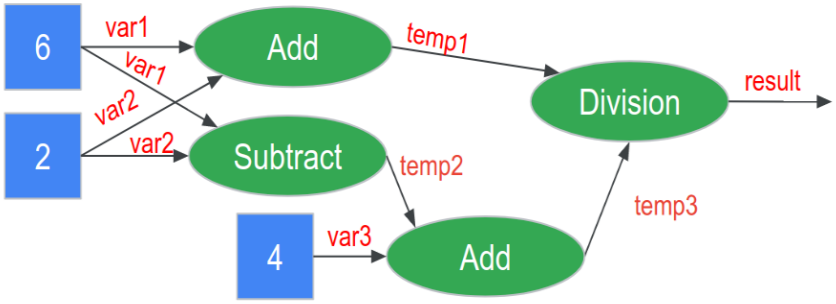
Now that we have talked about the difference between TensorFlow and TensorFlow Lite, let's dive a little under the hood to understand how models are represented and how conversion is processed.

4.3.1 Tensorflow's Computational Graph

Tensorflow represents models as a computational graph. To better understand what that means, let's use a simple example of the program shown below:

```
var1 = 6
var2 = 2
temp1 = var1 + var2
temp2 = var1 - var2
var3 = 4
temp3 = temp2 + var3
res = temp1/temp3
```

This program can be represented by the following computational graph which specifies the relationships between constants variables (often represented as tensors), and operations (ops):



4.3.2 The benefit of using such a computational graph is twofold:

Optimizations such as parallelism can be applied to the graph after it is created and managed by a backend compiled (e.g., Tensorflow) instead of needing to be specified by the user. For example, in the case of distributed training, the graph structure can be used for efficient partitions and learnable weights can be averaged easily after each distributed run. As another example, it enables efficient auto-differentiation for faster training by keeping all of the variables in a structured format.

Portability is increased as the graph can be designed in a high level language (like we do using Python in this course) and then converted into an efficient low level representation using C/C++ or even assembly language by the backend compiler.

Both of these reasons become increasingly important as we increase the size of our models --- even for TinyML many models have thousands of weights and biases!

If you'd like to learn more about this we suggest checkout this great article which we used for inspiration for this section of the reading.

4.3.3 Tensorflow Checkpoints

As you train your model you will notice that Tensorflow will save a series of .ckpt files. These files save a snapshot of the trained model at different points during the training process. While it may seem like a lot of files at first (e.g., you will notice later that that /train directory will get quite full when you train your custom

keyword spotting model in the next section), you do not need to be able to understand their contents. The most important thing to keep in mind about the checkpoint files is that you can rebuild a snapshot of your model. However, at a high level:

.ckpt-meta files contain the metagraph, i.e. the structure of your computation graph, without the values of the variables

.ckpt-data contains the values for all the weights and biases, without the structure.

.ckpt-index contains the mappings between the -data and -meta files to enable the model to be restored. As such, to restore a model in python, you'll usually need all three files.

If you'd like to take Google's hands on crash course on all things checkpoint files check out this Colab!

4.3.4 Freezing a Model

You may also notice some .pb files appear as well. These are in the Google "Proto Buffers" format. A Proto Buffer represents a complete model (both the metadata and the weights/biases). A ".pb" file is the output from "freezing" a model --- a complete representation of the model in a ready to run format!

Frozen/saved models are designed to be deployed into production environments from that format

Checkpoint files are designed to be used to jumpstart future training

If you'd like to learn more about freezing/saving models you can check out this Colab made by Google.

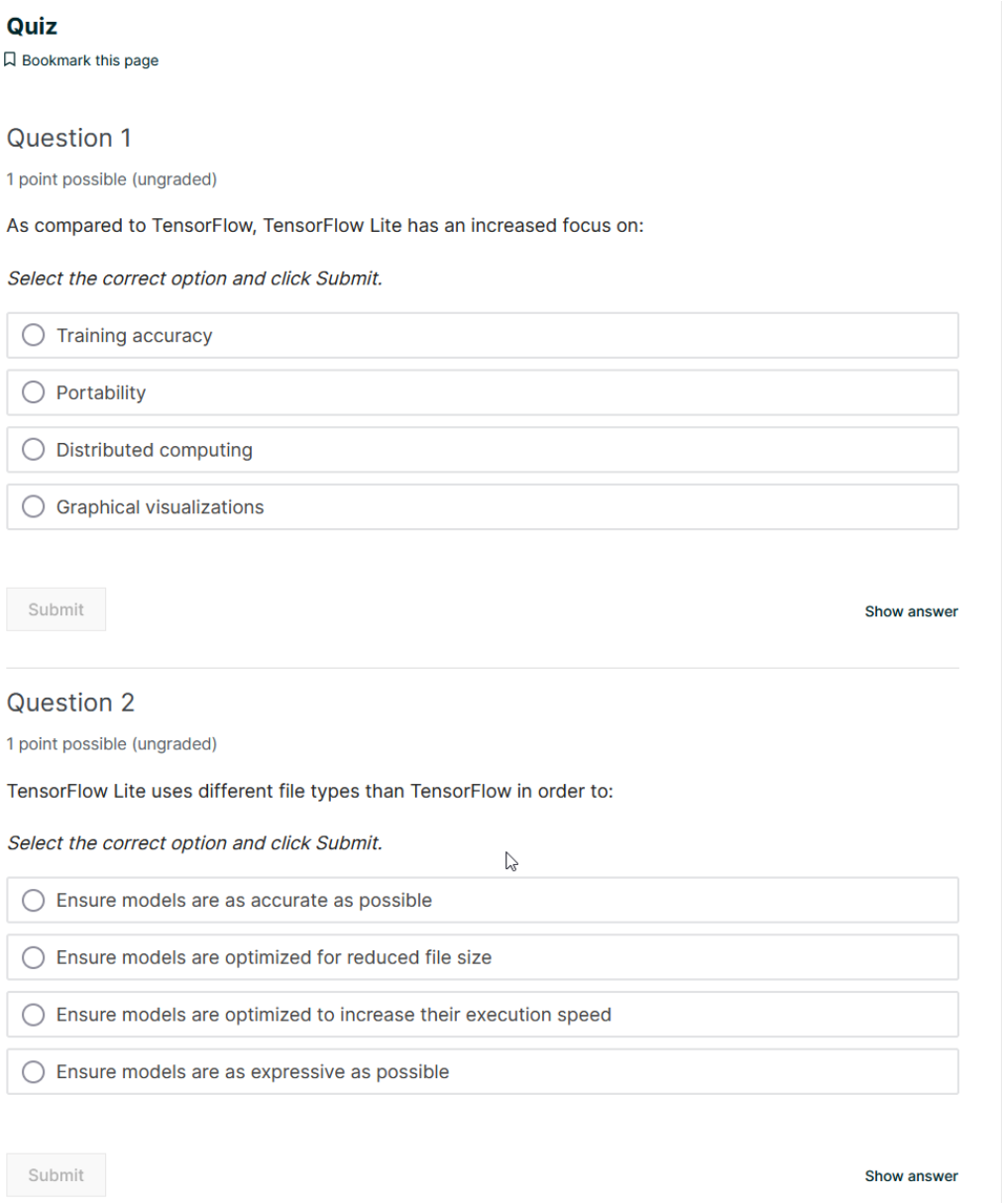
4.3.5 Optimizing a Model (Converting to TensorFlow Lite)

As we described earlier in the course, the Converter can be used to turn Tensorflow models into quantized TensorFlow Lite models. We won't go into detail about that process here as we covered it in detail earlier but suffice it to say that that process often reduces the size of the model by a factor of 4 by quantizing from Float32 to Int8.

TensorFlow Lite models are stored in the FlatBuffer file format as.tflite files.

The primary difference between FlatBuffers and ProtoBuffers is that ProtoBuffers are designed to be converted into a secondary representation before use (requiring memory to be allocated and copied at runtime), while FlatBuffers are designed to be per-object allocated into a compressed usable format. As such FlatBuffers offer less flexibility in what they can represent but are an order of magnitude more compact in terms of the amount of code required to create them, load them (i.e., do not require any dynamic memory allocation) and runt them, all of which are vitally important for TinyML!

4.4 Quiz



Quiz

Bookmark this page

Question 1

1 point possible (ungraded)

As compared to TensorFlow, TensorFlow Lite has an increased focus on:

Select the correct option and click Submit.

- ○ Training accuracy
- ○ Portability
- ○ Distributed computing
- ○ Graphical visualizations

Submit Show answer

Question 2

1 point possible (ungraded)

TensorFlow Lite uses different file types than TensorFlow in order to:

Select the correct option and click Submit.

- ○ Ensure models are as accurate as possible
- ○ Ensure models are optimized for reduced file size
- ○ Ensure models are optimized to increase their execution speed
- ○ Ensure models are as expressive as possible

Submit Show answer

5 Key Word Spotting

5.1 How (else) would you use KWS?

What are some creative ways you could think of using keyword spotting as a use case? Sure, we can wake up the machine to respond to us. But what are some things you think we can do if we are able to detect words in just an ongoing conversation? Get creative!

5.2 Keyword Spotting Application Architecture Overview

Keyword Spotting Application Architecture

In this reading, we will go over the high-level architecture of the application before we go deep into building the keyword spotting application (KWS).

There are several steps in the ML workflow and this is what each of the stages entails for KWS.

Step 1 Data collection

For Keyword Spotting we need a dataset aligned to individual words that includes thousands of examples which are representative of real world audio (e.g., including background noise).

Step 2: Data Preprocessing

For efficient inference we need to extract features from the audio signal and classify them using a NN. To do this we convert analog audio signals collected from microphones into digital signals that we then convert into spectrograms which you can think of as images of sounds.

Step 3: Model Design

In order to deploy a model onto our microcontroller we need it to be very small. We explore the tradeoffs of such models and just how small they need to be (hint: it's tiny)!

Step 4: Training

We will train our model using standard training techniques explored in Course 1 and will add new power ways of analyzing your training results, confusion matrices. You will get to train your own keyword spotting model to recognize your choice of words from our dataset. You will get to explore just how accurate (or not accurate) your final model can be!

Step 5: Evaluation

We will then explore what it means to have an accurate model and why your training/validation/test error may be different from the accuracy experienced by users.

Additional Topics:

We will also consider some additional topics that are unique to keyword spotting: post processing and cascade architectures.

5.3 Wake Words Dataset Creation

As we just learned, there is a lot of work that goes into designing a good dataset for keyword spotting. And this is true for all applications of TinyML. This is because the requirements for different applications (and thus the data that needs to be collected) can vary drastically even for the same kinds of inputs. In fact, even for the same keyword deployed into different environments!

This makes it impossible to come up with hard and fast rules for how to design a good dataset. But/and here are a couple of things you might want to consider when designing an audio data collection scheme.

Who are the anticipated end users?

It is important to understand who the end users might be as they may engage with the application differently and may present different kinds of challenges for how you need to design the application and what data you will need to collect. For example:

What languages will they speak?

What accents will they have?

Will they use slang or formal diction?

Will the users speak clearly?

In what environments will the users employ the application?

Imagine a Keyword Spotting device. Is it placed in a quiet room where it can hear you clearly? Or is it placed right next to your TV in the living room? The environment can greatly affect the quality and type of audio that can be collected, as well as the amount of additional sounds that must be accounted for. For example:

How much background noise do we expect?

Who/how many people will be talking in the environment?

How far will the users be from the device and sources of noise?

Will the user be stressed vs. calm vs. panicked?

Will the user use different volumes of voice (whispered/normal/loud/shouted)?

How likely is it that these keywords may be triggered unintentionally during conversation?

What are the goals for using the application? How will this impact the requirements of the ML model's performance?

Depending on the goals of the application, model errors can either have very little consequences, or they can be catastrophic. This can directly impact how robust of a data collection scheme you need to implement. For example consider the difference between these three scenarios:

The user is trying to turn the thermostat up/down

The user is trying to arm/disarm a security system

The user is trying to play their favorite playlist from a smart speaker

Given all of these previously mentioned factors here a short checklist to consider when implementing your final data collection scheme:

What custom keywords will you select?

What background noise samples do you need to collect to augment your dataset so that the trained model is robust?

What other words do you need to collect to ensure the model learns the difference between them and your keywords? It is important to collect other words so that the model can learn your particular word and not simply the general sounds of humans talking!

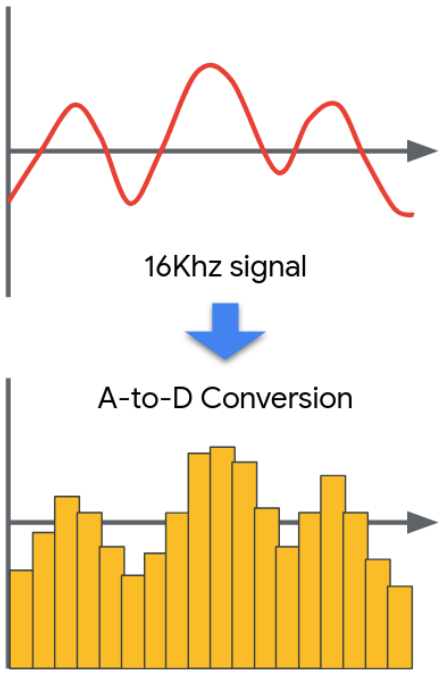
How much data will you need to collect? We all know more is better (usually), but you also live in the real world with time constraints so how much do you think will be enough?

In what environments will you collect these samples?

5.4 Spectrograms and MFCCs

In the last video you explored how audio is preprocessed for Keyword Spotting.

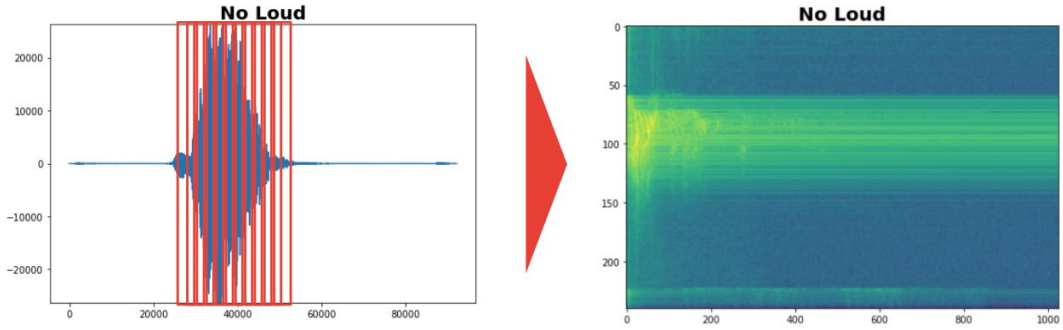
You saw that the raw analog audio signal is first transformed into a digital signal.



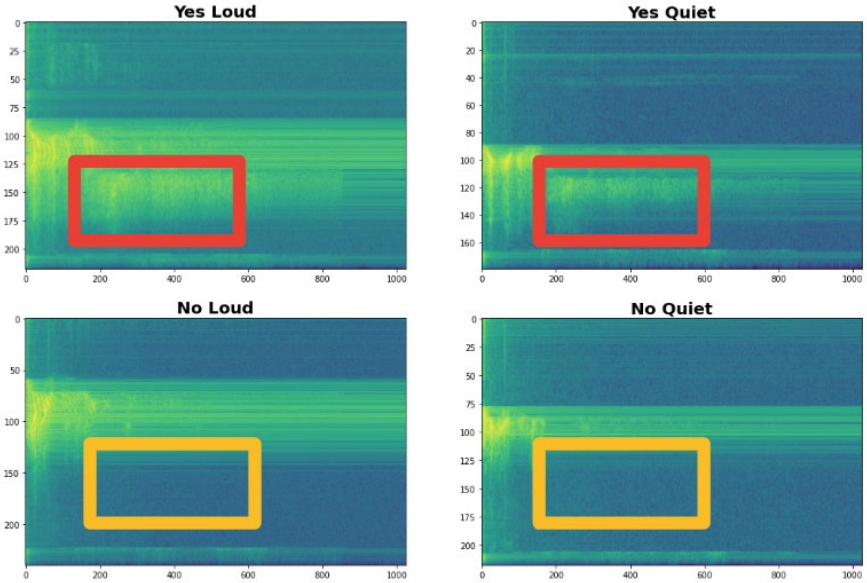
You then saw how the digital signal in the time domain (time on the x-axis and amplitude on the y-axis) is transformed into the frequency domain (frequency on the x-axis and amplitude on the y-axis) through the use of a fourier transform. If you'd like to dive deeper into understanding how the fourier transform works we suggest you watch this fantastic video on the 3Brown1Blue youtube channel.

Of course for the purposes of TinyML we need to worry about how we can compute this transformation efficiently -- both in the memory needed to store the program, and in the memory and latency needed for the computation. The solution is the use of an approximate fast fourier transform (FFT) of which there are multiple possible implementations. TensorflowLite Micro uses the KISS_FFT library, which includes more documentation about the design choices made by the library to be compact in memory on their GitHub page.

By applying the FFT through a sliding window procedure we can analyze the intensity (in color) of each of frequencies over time. This is a spectrogram.



A spectrogram is a better input for a deep learning system than the standard input audio signal as we have already extracted the frequencies for the neural network and so it doesn't need to learn how to do that on its own. Even visually, it is easier to see the difference between the "yes" and "no" spectrograms.

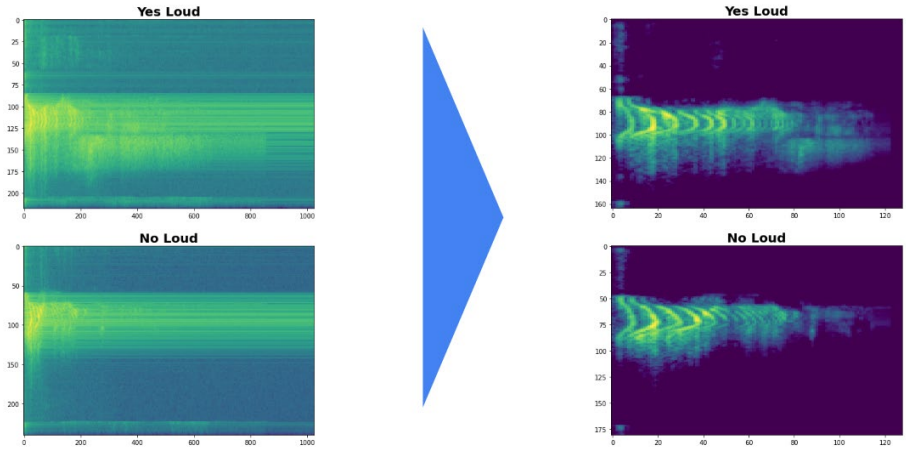


We then discussed how the human ear can tell low frequency signals apart easier than high frequency signals according to the Mel Scale. We can therefore create a Mel Filter Bank to take advantage of this phenomenon and group together larger clusters of higher frequency signals to provide more useful data to our machine learning algorithm. The output of this process is a Mel Frequency Cepstral Coefficient (MFCC). For more information on how this works here are three great posts:

post1, http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

post2, https://jonathan-hui.medium.com/speech-recognition-feature-extraction-mfcc-plp-5455f5a69dd9

post3. https://jonathan-hui.medium.com/speech-recognition-phonetics-d761ea1710c0



Of course there are many different ways you could solve this learning problem and it is likely that very large neural networks would perform quite well on this task without this preprocessing. However, for the case of TinyML this preprocessing is vital and drastically improves the final model performance without taking up a lot of memory or latency!

In Course 2, we will use these methods via Google Colab but in Course 3 you will learn how to deploy these pre-processing methods directly onto the embedded hardware platform.

5.5 Keyword Spotting in Colab

Now that you have seen how the Keyword Spotting application works we thought it might be fun to explore how well our default pre-trained model works! This will also start to get you familiar with the (relatively large amount of) code we are going to use to train a custom model later as that Colab will build directly on top of this one. And for fun we've added in a section where you can upload your own audio samples to see if the model can figure out if you are saying "yes" or "no" or something else "unknown."

As mentioned in the last Colab, this Colab, along with the others in this section of the course, will have a lot of code in it. We have tried to provide comments throughout to explain the code. At the same time, since machine learning is such a growing and changing field, a core skill for ML practitioners is to learn how to read documentation and leverage new libraries. Therefore we strongly suggest you google the various libraries used if you'd like to learn more about some of the functions we call! For example in this Colab we use the pickle

library to load in a python object from a web file that we download through the use of wget. We also make extensive use of the TensorFlow Micro example code which can be found here.

Colab:

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-5-13-PretrainedModel.ipynb

5.6 Training in Colab

Training Keyword Spotting in Colab

The code for training your model in Colab is very similar to the code you used to test out the pre-trained model. However, we'd like to point out a couple of things that will hopefully make training in Colab easier (and more intuitive).

5.6.1 Timeouts

After 12 hours your Colab instance will automatically recycle so make sure to download anything you want to keep before then. Also after about 90 minutes of no activity Colab may also recycle the instance as it may deem it "stale." Therefore, during training it is best practice to check in every hour and click around on the Colab to make sure it knows you are still there! But/and beyond that you can absolutely minimize the window and continue with other work.

5.6.2 File Structure

The training process will create three directories of note.

/data: will hold all of the data from the Speech Commands dataset. If you want to take a look at particular audio files used for training this is where you can inspect and download them. We will revisit this in Course 3 when you design your own dataset as you'll need to ensure that it matches this format / file structure.

/train: will hold all of the model checkpoint files. These files describe the state of the model after X steps of training (usually denoted in the file name and the training script auto-saves them every 100 or so steps). If you e.g., trained for 100 steps and wanted to train for another 100 more, you can use the latest checkpoint file at step 100 to start from there instead of re-training from scratch for 200 steps.

/model: will hold all of the final processed model files. There you can find both the frozen Tensorflow model, the TensorflowLite model, and in course 3 you will also find the TensorFlowLite Micro model there. If you'd like to download and save your trained model to compare against another trained model at another time you should make sure to download it from there!

5.6.3 The Training Script

The training script is designed to automate a bunch of the training process. It first sets up the optimizer using a sparse_softmax_cross_entropy loss function. It then runs the optimizer (by default stochastic gradient descent) on the training data and automatically preprocesses the audio files as we've discussed in previous sections, such as computing the spectrogram and then the MFCC of the audio stream, before passing them to the model/optimizer.

If you want to take a look at the source code it can be found here:

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/speech_commands/train.py

You'll notice that there are even more flags than we set in the colab. If you'd like to customize the training process even more you can!

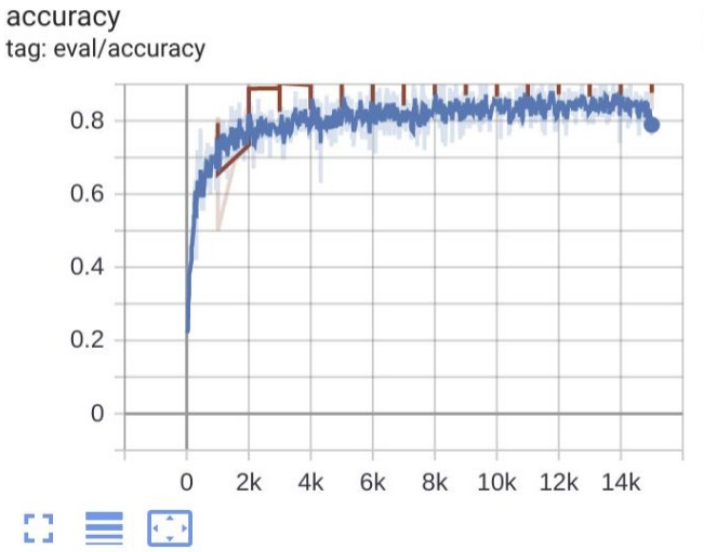
5.7 Monitoring Training in Colab

Monitoring Training in Colab

While the Keyword Spotting training script is running (for ~2 hours with the default settings and a GPU runtime), you may want to check-in and analyze how the training is performing. We've included two ways that you can do this into the Colab.

TensorBoard

TensorBoard is an application that is designed to help you visualize the training process. It will output graphs of the accuracy over time that look something like the below (this is a screenshot of the staff training a Keyword Spotting model):



If you'd like to learn more about TensorBoard, Google has put together a nice intro Colab that explore some of its features:

https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/get_started.ipynb

DEBUG Mode

Unfortunately, the staff has found that it sometimes doesn't start showing data for a while (~15 minutes) and sometimes doesn't show data until training completes (and instead shows No dashboards are active for the current data set.). Therefore, we have also set the training script to run in DEBUG mode. By doing so it will print out information about the training process as it progresses. In general you will see printouts at each training step showing the current accuracy that look something like this:

```
DEBUG:tensorflow:Step #25: rate 0.001000, accuracy 31.0%, cross entropy 1.342161
I1119 15:42:11.849751 140088470198144 train.py:257] Step #25: rate 0.001000, accuracy 31.0%, cross entropy 1.342161
DEBUG:tensorflow:Step #26: rate 0.001000, accuracy 33.0%, cross entropy 1.309237
I1119 15:42:12.187090 140088470198144 train.py:257] Step #26: rate 0.001000, accuracy 33.0%, cross entropy 1.309237
DEBUG:tensorflow:Step #27: rate 0.001000, accuracy 38.0%, cross entropy 1.294655
I1119 15:42:12.525735 140088470198144 train.py:257] Step #27: rate 0.001000, accuracy 38.0%, cross entropy 1.294655
DEBUG:tensorflow:Step #28: rate 0.001000, accuracy 35.0%, cross entropy 1.308346
I1119 15:42:12.882854 140088470198144 train.py:257] Step #28: rate 0.001000, accuracy 35.0%, cross entropy 1.308346
DEBUG:tensorflow:Step #29: rate 0.001000, accuracy 34.0%, cross entropy 1.343738
I1119 15:42:13.224437 140088470198144 train.py:257] Step #29: rate 0.001000, accuracy 34.0%, cross entropy 1.343738
DEBUG:tensorflow:Step #30: rate 0.001000, accuracy 38.0%, cross entropy 1.327567
I1119 15:42:13.560090 140088470198144 train.py:257] Step #30: rate 0.001000, accuracy 38.0%, cross entropy 1.327567
DEBUG:tensorflow:Step #31: rate 0.001000, accuracy 42.0%, cross entropy 1.334325
I1119 15:42:13.908692 140088470198144 train.py:257] Step #31: rate 0.001000, accuracy 42.0%, cross entropy 1.334325
DEBUG:tensorflow:Step #32: rate 0.001000, accuracy 37.0%, cross entropy 1.280344
I1119 15:42:14.248164 140088470198144 train.py:257] Step #32: rate 0.001000, accuracy 37.0%, cross entropy 1.280344
```

Remember because we are using a variant of stochastic gradient descent, it is natural for the error to go up and down a bit during training. The key thing to look for is that the larger trend is for the accuracy to be going up on average! For example, if I scroll down a bit on this training run we can see that ~500 steps later we now have much higher accuracy on average (so learning seems to be going well):

```
DEBUG:tensorflow:Step #552: rate 0.001000, accuracy 70.0%, cross entropy 0.783156
I1119 15:45:21.618249 140088470198144 train.py:257] Step #552: rate 0.001000, accuracy 70.0%, cross entropy 0.783156
DEBUG:tensorflow:Step #553: rate 0.001000, accuracy 76.0%, cross entropy 0.599380
I1119 15:45:21.958082 140088470198144 train.py:257] Step #553: rate 0.001000, accuracy 76.0%, cross entropy 0.599380
DEBUG:tensorflow:Step #554: rate 0.001000, accuracy 80.0%, cross entropy 0.637335
I1119 15:45:22.298068 140088470198144 train.py:257] Step #554: rate 0.001000, accuracy 80.0%, cross entropy 0.637335
DEBUG:tensorflow:Step #555: rate 0.001000, accuracy 83.0%, cross entropy 0.511202
I1119 15:45:22.637377 140088470198144 train.py:257] Step #555: rate 0.001000, accuracy 83.0%, cross entropy 0.511202
DEBUG:tensorflow:Step #556: rate 0.001000, accuracy 73.0%, cross entropy 0.677150
I1119 15:45:22.980068 140088470198144 train.py:257] Step #556: rate 0.001000, accuracy 73.0%, cross entropy 0.677150
DEBUG:tensorflow:Step #557: rate 0.001000, accuracy 73.0%, cross entropy 0.687472
I1119 15:45:23.314535 140088470198144 train.py:257] Step #557: rate 0.001000, accuracy 73.0%, cross entropy 0.687472
DEBUG:tensorflow:Step #558: rate 0.001000, accuracy 71.0%, cross entropy 0.628997
I1119 15:45:23.657741 140088470198144 train.py:257] Step #558: rate 0.001000, accuracy 71.0%, cross entropy 0.628997
DEBUG:tensorflow:Step #559: rate 0.001000, accuracy 71.0%, cross entropy 0.681584
I1119 15:45:24.164956 140088470198144 train.py:257] Step #559: rate 0.001000, accuracy 71.0%, cross entropy 0.681584
DEBUG:tensorflow:Step #560: rate 0.001000, accuracy 80.0%, cross entropy 0.560615
I1119 15:45:24.509710 140088470198144 train.py:257] Step #560: rate 0.001000, accuracy 80.0%, cross entropy 0.560615
DEBUG:tensorflow:Step #561: rate 0.001000, accuracy 71.0%, cross entropy 0.798682
```

Every 1,000 training steps you will also see a Confusion Matrix printed out that looks something like this:

```
INFO:tensorflow:Confusion Matrix:
[[180   6   9   6]
 [  2 100  36  63]
 [  3  12 373   9]
 [  6  20  23 357]]
```

What this is describing is the number of samples that were correctly and incorrectly classified by class for the validation set at this point in the training process. Labeling the axis we can see how this works a little better:

Actual Label	Item A	Item B	Item C	Item D
Item A	**180**	6	9	6
Item B	2	**100**	36	63
Item C	3	12	**373**	9
Item D	6	20	23	**357**

Predicted Label

The column plots the predicted item label by the model while the row plots the actual correct label. Down the diagonal in green are the number correctly labeled items. Off the diagonal we find the number of items incorrectly labeled as the various labels. For all of your Keyword Spotting models Item A is “silence”, Item B is “unknown” and then Items C, D, etc. are the various words you chose to train on in the order in which you define them in the Colab. For example in the pretrained model we had:

WANTED_WORDS = "yes,no"

Which means that Item C was “yes” and Item D was “no” and the matrix was a 4x4 matrix. If we had instead set:

WANTED_WORDS = "yes,no,left"

Then it would be a 5x5 matrix and Item C would still be “yes” and Item D would still be “no” and then there would be an Item E that was “left.”

We hope that between TensorBoard and the DEBUG output you’ll be able to understand how well your training is going and gain insights to improve your results!

5.8 Assignment: Training your own Keyword Spotting Model

Now that you know how to train a new Keyword Spotting model it's time for your assignment -- pick your own keywords from the Speech Commands dataset and train your own model! You’ll get to test it with your own audio input when you’re done! We hope you have a lot of fun with this one! As a quick warning, with the default values this should take ~2 hours to train on a GPU runtime once you launch the training cell -- so feel free to keep working through the rest of this section while it trains!

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-5-18-TrainingKeywordSpotting.ipynb

5.9 Assignment Solution

For this assignment you needed to input a few hyper-parameters for the model training. A possible solution is below but any combination of 2-4 words from the list provided in the comment should have done reasonably well (and more words might have worked too although after a point the small model may have started to struggle). For training steps, the longer you trained, the better the accuracy should have gotten, up to a point, and then you may have found that the model started to overfit. As for the learning rate, if it was too high the training most likely failed due to divergence, and if it was too low the training likely failed because it didn’t make much progress. However, values somewhat close to the default settings should have worked reasonably well.

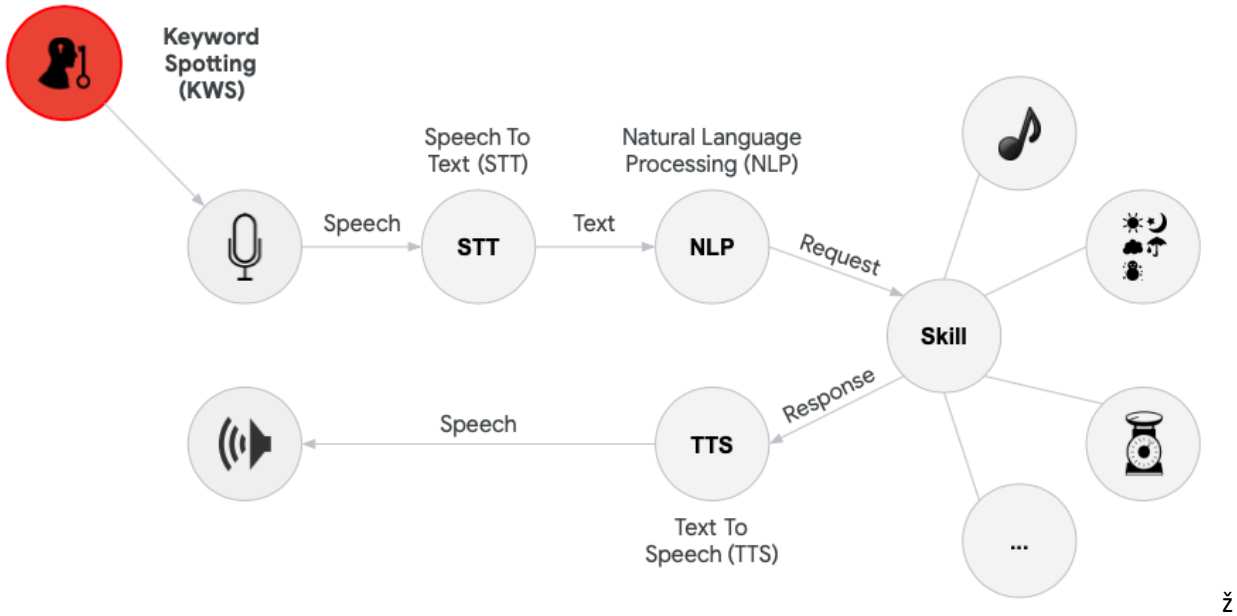
```
WANTED_WORDS = "yes,no"
TRAINING_STEPS = "12000,3000"
LEARNING_RATE = "0.001,0.0001"
```

This assignment was mostly designed to give you experience with the full training pipeline and flow to prepare you for Course 3 where we will be training on your own custom data and deploying the model to your own

hardware. Now is the time to make sure you understand the full flow well so you are in a better position to debug later when things get more complicated.

We hope you had some fun picking some words and getting to see the model learn based on your choices! We hope you also had fun trying to trick it with custom input! As always you'll find the link to the assignment again below in case you want to explore it a bit more!

5.10 Keyword Spotting in the Big Picture



ž

In this section, we have focused on the task of keyword spotting for tiny machine learning systems. We have seen how to develop algorithms for performing keyword spotting on a small set of keywords, and the challenges associated with developing these algorithms. We have also seen how audio can be streamed from a microphone to allow for low-power and yet online monitoring of keywords, and how this can be linked in a cascade architecture as a gateway to more complex and power-hungry operations such as natural language processing.

Keyword spotting has become increasingly popular in recent years, such as the introduction of software-based personal assistants like Siri (Apple), Cortana (Microsoft), Alexa (Amazon), and Google Assistant (Google). Similarly, smartphones utilize keyword spotting even while the CPU is not running through the use of an onboard digital signal processor, which is able to 'wake up' the CPU when a keyword is detected. These devices also utilize a cascade architecture, allowing the user to wake the device and subsequently ask questions which can be parsed and analyzed using natural language processing methods via an interface to the Cloud.

The prevalence of keyword spotting in industry will likely continue to increase as voice-recognition and natural language processing algorithms become increasingly accurate and powerful. We are already beginning to see voice-enabled TV's, alarm clocks (e.g., Hatch), lighting systems (e.g., Philips Hue), thermostats (e.g., Honeywell), and even smart vacuum cleaners (e.g., LG SmartThinQ). Based on the work you have done in this

section, you are now equipped with many of the tools to develop your own systems for keyword spotting applications.

6 Introduction to Data Engineering

6.1 What is Data Engineering?

Data engineering is a critical component of supervised learning, and consists of defining requirements, recording data, processing it, and improving the dataset. The quality and quantity of collected data determines the tractability of a machine learning objective. Training examples need to include enough salient features, along with representative noise induced by the surrounding environment (for example, day or night in images, or quiet and loud background noise for audio), for an ML algorithm to accurately distinguish between classes when deployed into the real world.

Data engineers need a rigorous problem definition in order to know what data should be collected, and must identify the potential sources of data. Data might come from on-device sensors, product users, or paid or unpaid contributors, and each may introduce potential licensing or privacy restrictions. This data must be labeled, and this usually requires manual effort by individual workers. It may also require domain expertise, for example, when labeling medical images. Mislabeled or garbled data may also need to be filtered out through manual inspection. Data engineers must also manage changing needs for a dataset, for example, in order to support additional languages.

6.2 What's in this Module?

All machine learning tasks begin with datasets. Datasets are often handed to us, and many of us never think about where they came from. But that's a big mistake. Not being careful about dataset engineering can have serious consequences down the road. We are going to discover some of those pitfalls so that we don't fall subject to those same potholes. We will use existing datasets as examples to ground the principles you will learn, specifically around identifying dataset requirements, data collection, data refinement, and sustainability.

Specifically we will touch upon these items:

- The need for datasets as benchmark standards
- How to collect datasets for TinyML use cases
- How to be wary of bias and incorrect labeling in the datasets
- How to broaden the dataset collection process to accommodate diverse populations
- How we can repurpose existing (large) datasets for TinyML use cases
- How to be responsible when collecting datasets

6.3 Your New TinyML Applications

Now that you've learned more about dataset design, how would you change your approach to dataset collection for the keyword spotting application you came up with in the previous section? How might you change the application to make data collection easier?

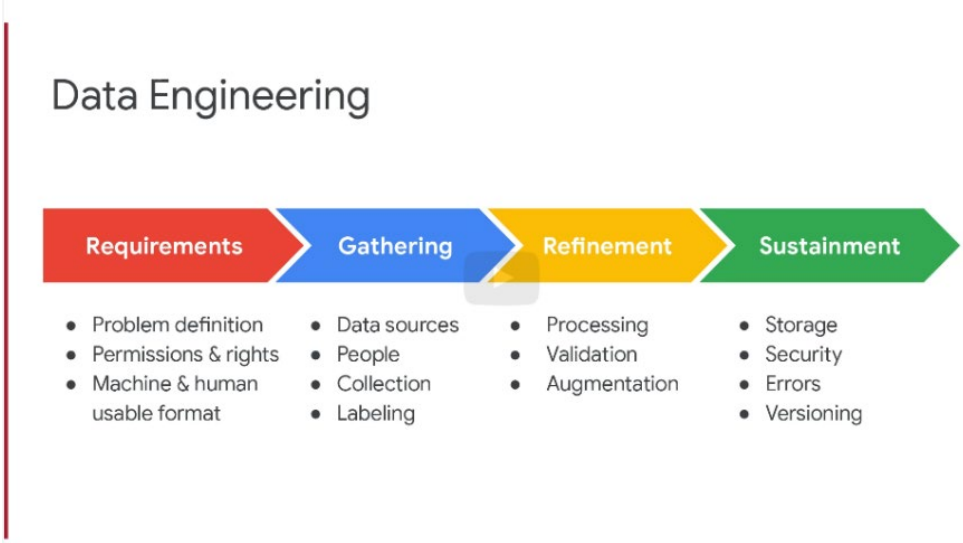
Remember we considered many questions including:

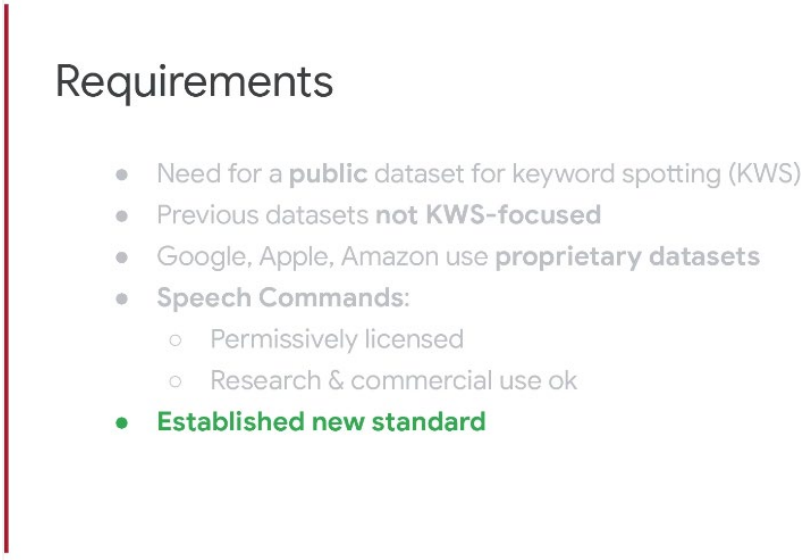
Who are the anticipated end users?

In what environments will the users employ the application?

What are the goals for using the application? How will this impact the requirements of the ML model's performance?

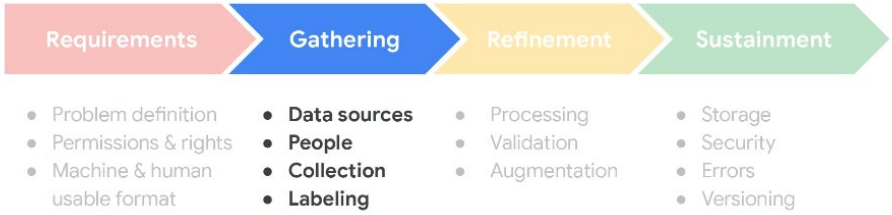
6.4 Dataset Standards: Speech Commands





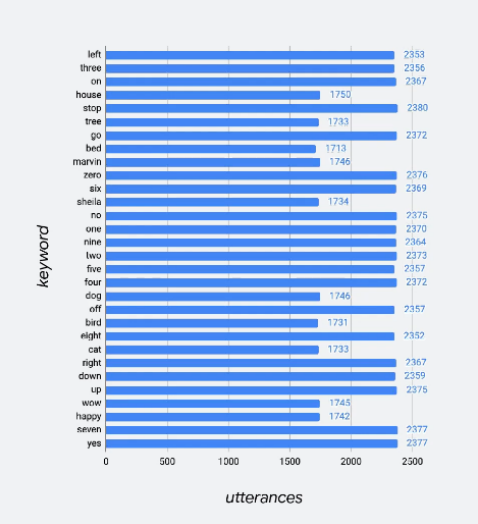
Requirements

Data Engineering



Data Collection

- **2,618** volunteers
  - consented to have their voices redistributed
  - Variety of accents
- > 1,000 examples for **each** keyword
- **Browser-based** (no app to install)



Data Engineering

Requirements → Gathering → **Refinement** → Sustainment

- Problem definition
- Permissions & rights
- Machine & human usable format

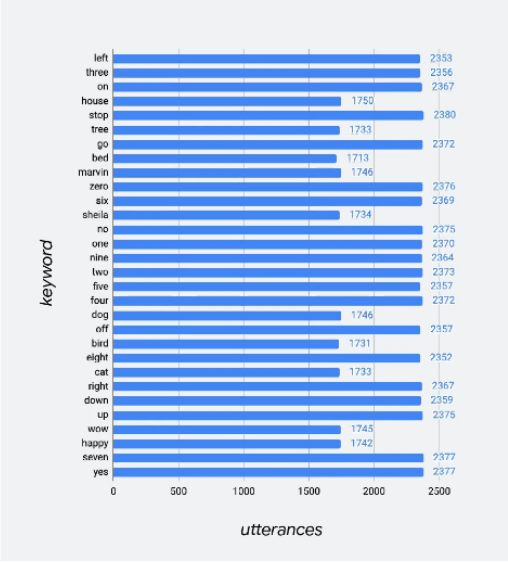
- Data sources
- People
- Collection
- Labeling

- **Processing**
- **Validation**
- **Augmentation**

- Storage
- Security
- Errors
- Versioning

Data Validation

- Some data is **unusable**
  - Too quiet, wrong word, etc
- Started with **automated tools**
  - Remove low volume recordings
  - Extract loudest 1s (from 1.5sec examples)
- All 105,829 remaining utterances **manually reviewed** through crowdsourcing



Data Engineering

Requirements → Gathering → Refinement → **Sustainment**

- Problem definition
- Permissions & rights
- Machine & human usable format

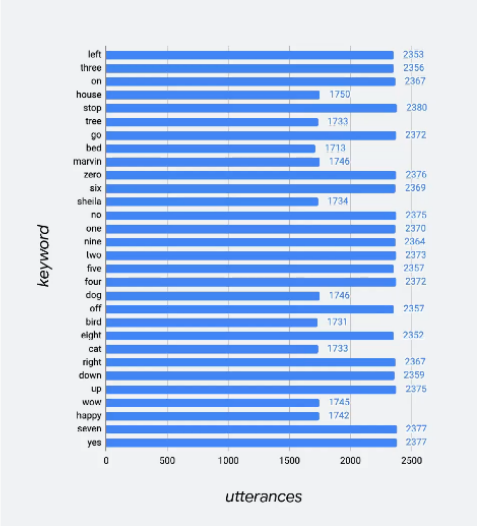
- Data sources
- People
- Collection
- Labeling

- Processing
- Validation
- Augmentation

- **Storage**
- **Security**
- **Errors**
- **Versioning**

Sustaining KWS Research

- Speech Commands is now in **v2**
  - **Expanded to 35** keywords from original 10
- Includes train/validation/test splits
- Expand to **new languages**?



Why is **wide availability** of data ***important***?

Compare Without a standard dataset, no easy way to compare models

Why is **wide availability** of data ***important***?

Compare Without a standard dataset, no easy way to compare models

Benchmark Good benchmarks: critical under *TinyML* constraints!
Not just accuracy: power, memory, latency

Why is **wide availability** of data ***important***?

Compare	Without a standard dataset, no easy way to compare models
Benchmark	Good benchmarks: critical under *TinyML* constraints! Not just accuracy: power, memory, latency
Improve	Speech Commands: a standard training & evaluation dataset for each new KWS technique

Researchers using the same data: ***"apples-to-apples" evaluation***

6.5 Speech Commands Paper

In this reading assignment, we want you to read through the "Speech Commands" (https://arxiv.org/pdf/1804.03209.pdf) paper that Pete Warden (one of the course instructors) himself wrote. The paper provides additional information on some of the details that we discussed in the video lectures. It provides a bit more context and detail around the following important topics:

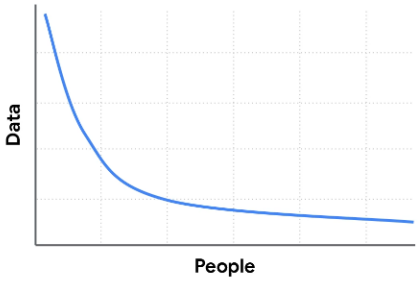
What are Speech Commands?

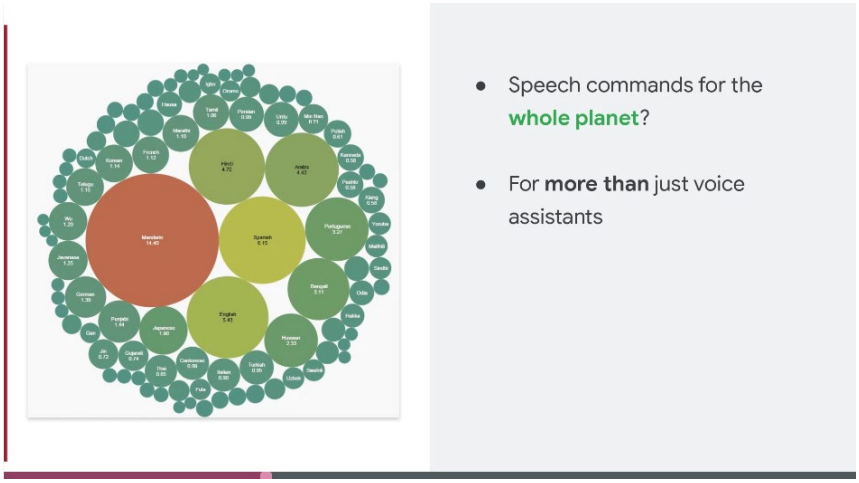
What was Pete's motivation behind building Speech Commands?

How is Keyword Spotting different from traditional speech recognition models?

What are the important metrics in speech recognition for KWS?

6.6 Crowdsourcing Data for the Long Tail





Cost Model v. **Community** Model?

Limited Scale

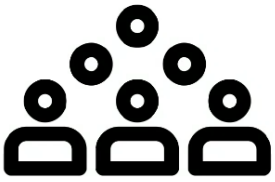
Social Good

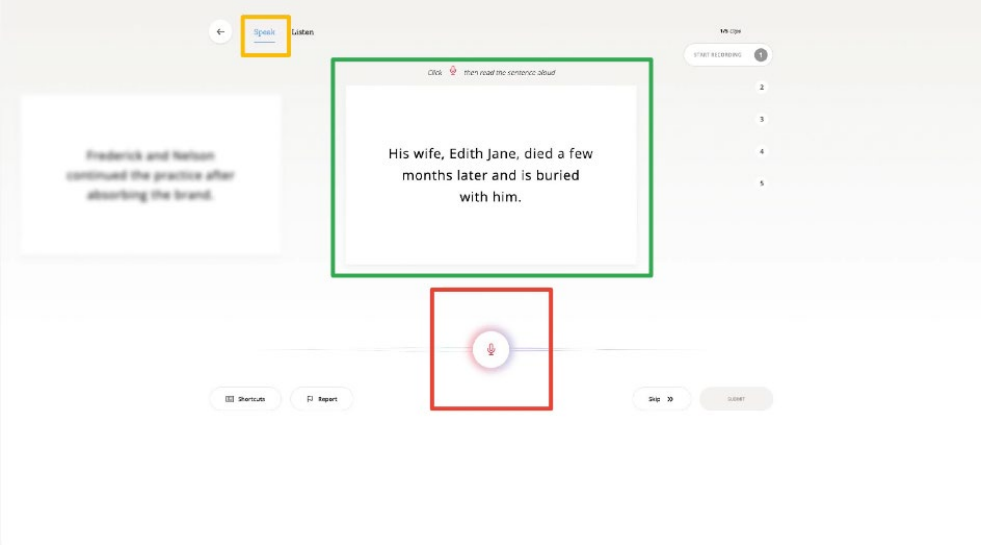
Common Voice

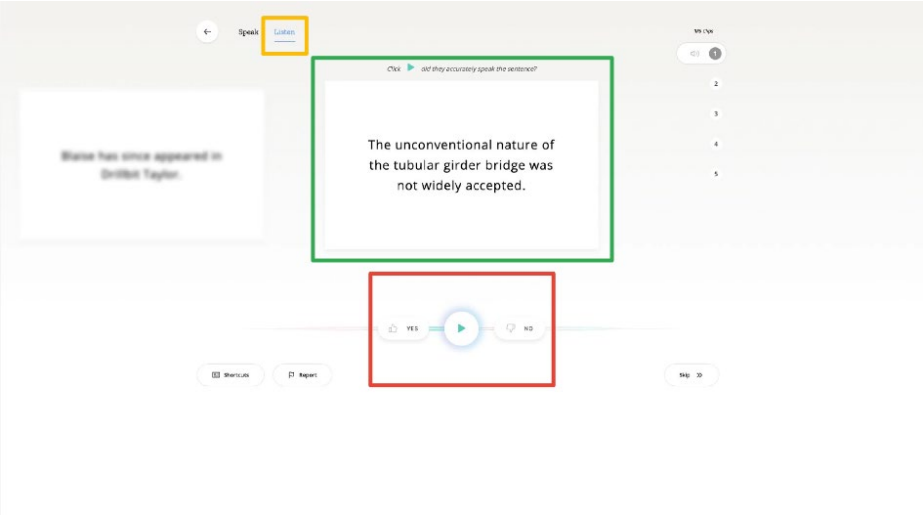
https://commonvoice.mozilla.org

Common Voice

- **Crowdsourcing** platform

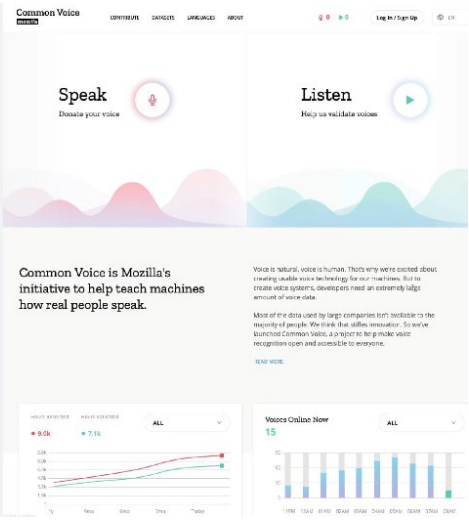






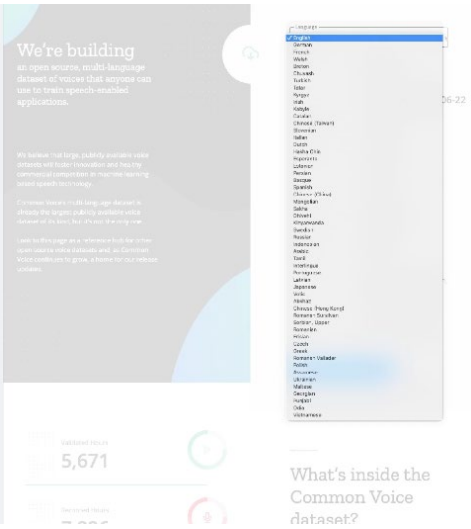
Common Voice

- **Crowdsourcing** platform
- Over **50,000 volunteers**



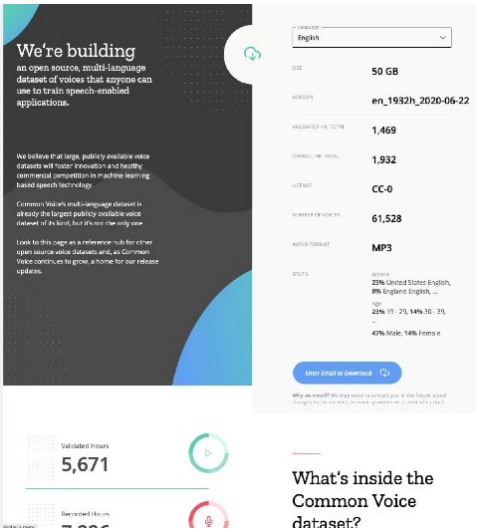
Common Voice

- **Crowdsourcing** platform
- Over **50,000 volunteers**
- **54 different languages**



Common Voice

- **Crowdsourcing** platform
- Over **50,000 volunteers**
- **54 different languages**
- Goal: **speech recognition** for ***all languages*** on the planet



File **Structure**

- **Valid**
  - At least 2 people listen to them, and the majority of those listeners say the audio matches the text

File **Structure**

- Valid
  - At least 2 people listen to them, and the majority of those listeners say the audio matches the text
- **Invalid**
  - At least 2 listeners, and the majority say the audio does not match the clip

File **Structure**

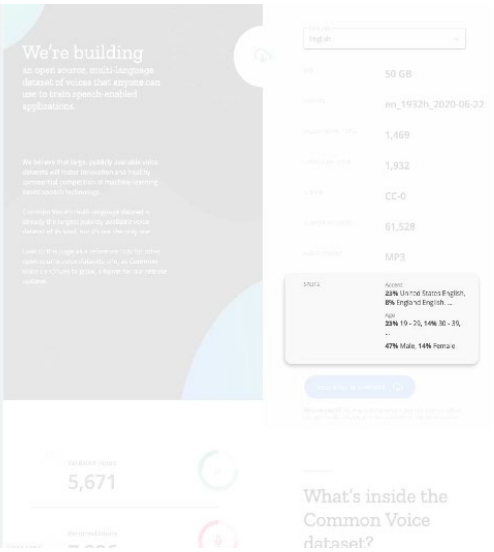
- Valid
  - At least 2 people listen to them, and the majority of those listeners say the audio matches the text
- Invalid
  - At least 2 listeners, and the majority say the audio does not match the clip
- **Other**
  - All other clips, i.e., fewer than 2 votes, or those that have equal valid and invalid votes, are labelled "other"

Interesting Attributes

- **Permissive license**

Interesting Attributes

- Permissive license
- Many contributors
- **Comes with metadata**

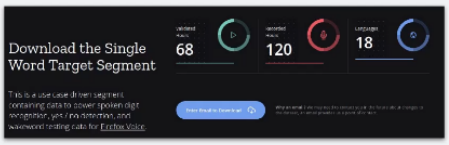


Common Voice

Single Word Target Segment

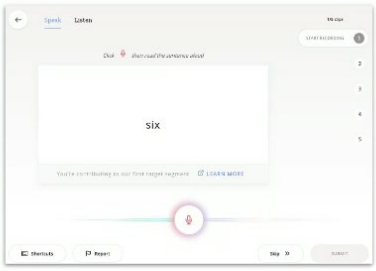
A ***speech commands-style*** dataset for **18 languages**

- "Yes" // "no"
- "hey" & "Firefox"
- **digits** 0-9

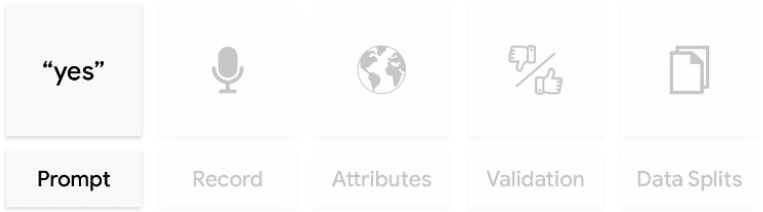


ASR **Diversity** and **Reach**

- Common Voice
  - Permissive license
  - Minority languages
- **Ease-of-use, wide reach**
  - Browser-based
  - Community can add new languages
- **You can contribute!**



Common Voice **Data Structure**



6.7 Giving Back to the Open Source Community

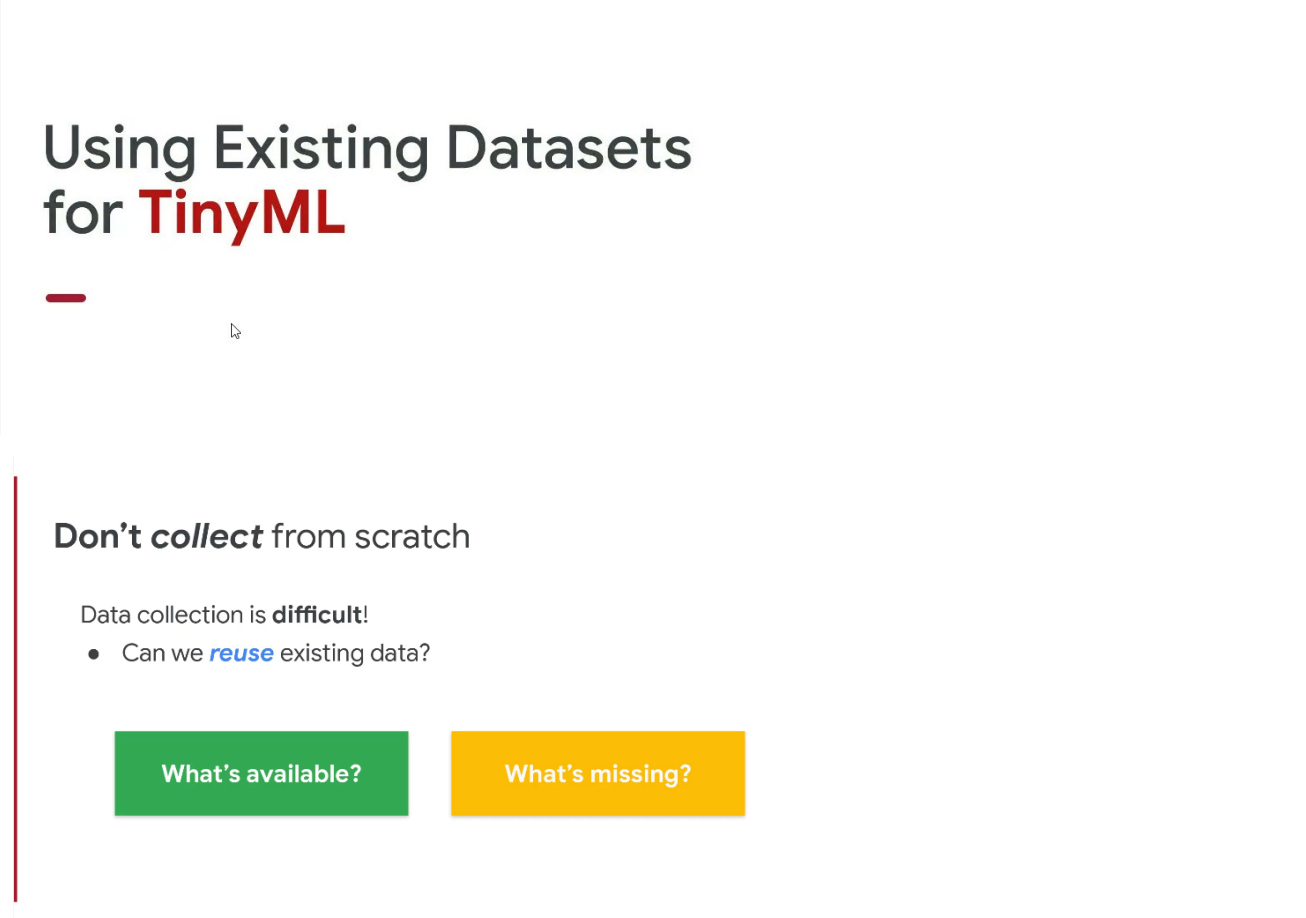
We have explored community driven data collection through the examples of both the Speech Commands Dataset and the Common Voice Project. We discussed the benefits and limitations of these approaches. Given that we believe that these projects are important for the TinyML community, we thought it would be a good idea (and fun) to give you a chance to contribute to such a project and fortunately, the Common Voice project is still actively collecting data and looking for help with validating recordings from other users!

This is 100% optional. You are in no means required to contribute and your contribution will have no impact or bearing on your completion of the course. If you would like to contribute to the Common Voice project the link can be found below. Please consult the data privacy documents and warnings listed on the website before participating.

https://commonvoice.mozilla.org/en

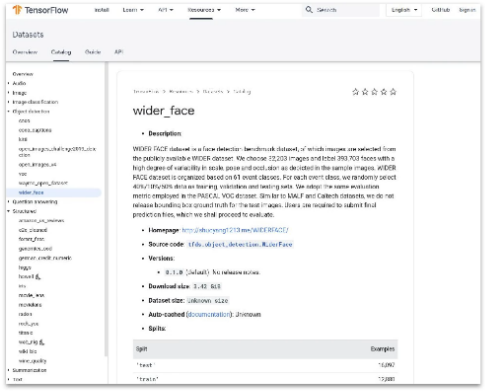
Update: The Common Voice project has decided to combine their single world data collection and validation with their full sentence data collection and validation. As such, you will see a mix of single words and full sentences if you choose to contribute.

6.8 Reusing and Adapting Existing Datasets



TensorFlow
Datasets Catalog

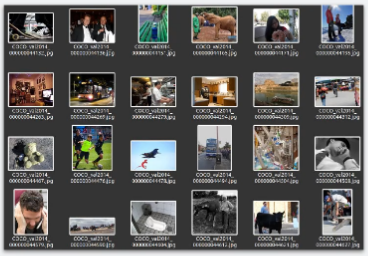
Audio
Image
Image Classification
Object Detection
Question Answering
Structured
Summarization
Text
Translate
Video



TinyML
Person Detection

- **Visual Wake Words**: a new dataset built from Common Objects in Context (**COCO**)
  - ***people*** v. ***no people***

Repurposing existing datasets for **TinyML** tasks is a powerful concept

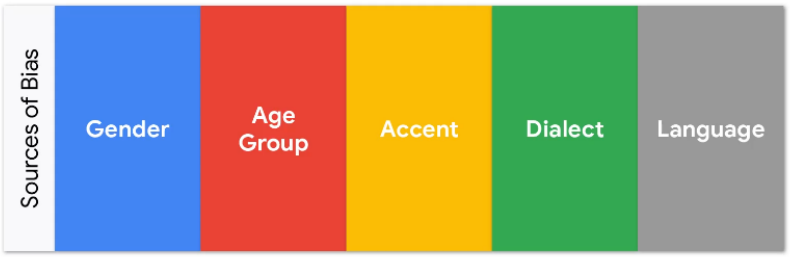


Don't *learn* from scratch

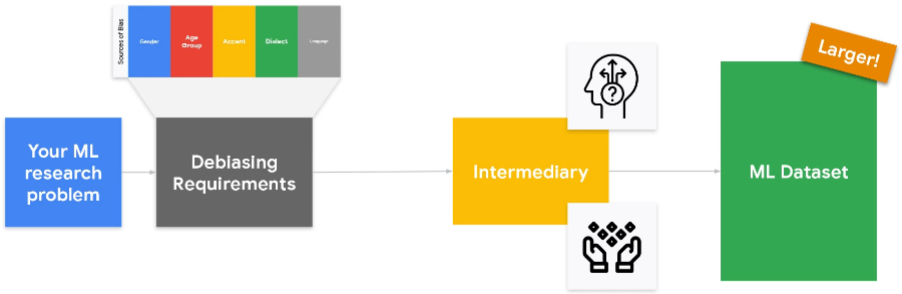
- **Transfer** learning
- **Pretrained** models: your "AI Data Labeling Assistant"
- **Generate** your own data
  - Simulations
  - ML models

6.9 Responsible Data Collection

Potential **Bias** in Speech Recognition



Bias and **Market Forces**



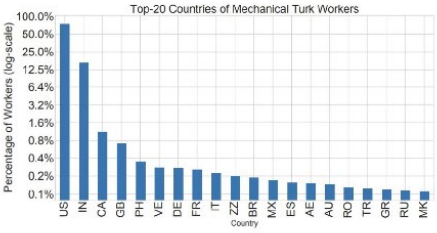
Data Engineering and Bias

- **Amazon Mechanical Turk** is a crowdsourcing platform used for labeling data for ML tasks

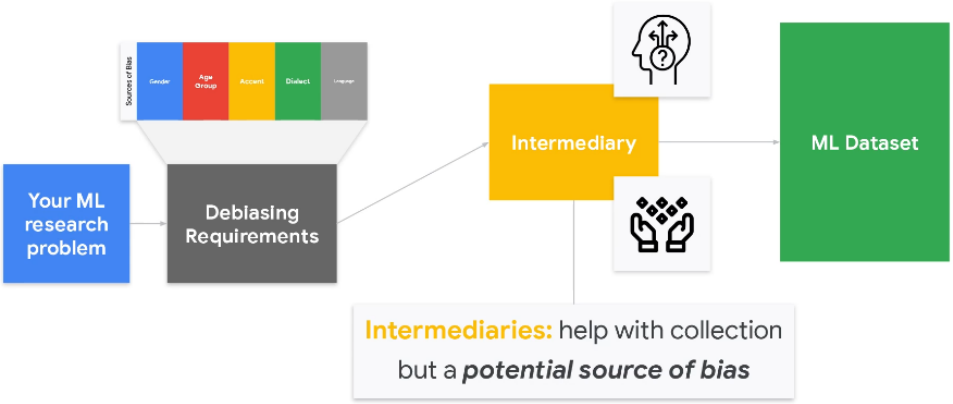
amazon beta mechanical turk

Data Engineering and Bias

- **Amazon Mechanical Turk** is a crowdsourcing platform used for labeling data for ML tasks
- **Workers are (not) unbiased**



Bias and **Market Forces**



How can you work to ***avoid bias*** in your dataset?

6.10 Summary

6.10.1 Summary: Data Engineering

6.10.1.1 What is Data Engineering?

Data engineering is a critical component of supervised learning, and consists of defining requirements, recording data, processing it, and improving the dataset. The quality and quantity of collected data determines the tractability of a machine learning objective. Training examples need to include enough salient features, along with representative noise induced by the surrounding environment (for example, day or night in images, or quiet and loud background noise for audio), for an ML algorithm to accurately distinguish between classes when deployed into the real world.

Data engineers need a rigorous problem definition in order to know what data should be collected, and must identify the potential sources of data. Data might come from on-device sensors, product users, or paid or unpaid contributors, and each may introduce potential licensing or privacy restrictions. This data must be labeled, and this usually requires manual effort by individual workers. It may also require domain expertise, for example, when labeling medical images. Mislabeled or garbled data may also need to be filtered out through manual inspection. Data engineers must also manage changing needs for a dataset, for example, in order to support additional languages.

6.10.1.2 Speech Commands

Speech Commands is a keyword spotting (KWS) dataset developed by Pete Warden at Google, and we will consider it as a practical example of the steps required in TinyML data engineering.

Requirements: It established a new standard for public, comparable keyword spotting research - previous datasets for KWS were often restrictively licensed or proprietary to individual companies. Speech Commands is available for anyone to use, and allows ML researchers to compare their ML algorithm's performance on the same data.

Collection: Speech Commands contains thousands of recorded examples for 35 keywords, collected by over 2,600 volunteers with a variety of accents. All volunteers agree to have their voices redistributed. Data collection was done in the browser, since requiring installation of an app might discourage contributors.

Refinement: As with any dataset, some data collected will be unusable (for example, if an incorrect word is spoken or the microphone gain was too quiet). Some automated techniques were applied (removing low volume recordings and extracting the loudest 1sec from 1.5sec examples), but the remaining 105,829 recordings were manually verified through crowdsourcing.

Sustainment: Speech Commands has been expanded to 35 keywords in Version 2, from the original 25 words in Version 1. Care was also taken to ensure the same recordings remain in the same train, validation, and test splits across the two versions.

The careful construction of the Speech Commands dataset has allowed keyword spotting researchers to compare the performance of different TinyML neural architectures on the same reference data. This benefits many aspects of KWS research, such as reproducibility.

6.10.1.3 Crowdsourcing Data for the Long Tail

Speech Commands only contains English data, but many languages are spoken across the world. Paying contributors to record and verify keyword data for every language rapidly becomes cost-limited. Companies which pay to collect speech data may prefer to keep this data in-house and proprietary. An alternative approach is crowdsourcing speech data.

Community contributions have led to the success of other open-source projects such as Linux and TensorFlow, and this model can also work for dataset generation. CommonVoice is a Mozilla-led effort which seeks to attract community contributions for speech data. So far, over 50,000 volunteers have contributed speech data in 54 languages to CommonVoice. A key draw for potential contributors is the promise of bringing modern advancements in speech processing to underserved languages.

CommonVoice data is permissively licensed for anyone to use. The majority of data in Common Voice consists of full sentences read aloud by volunteers, and verified to be discernible by other volunteers through a voting process. CommonVoice also contains a single-word target segment dataset (in the same style as Speech Commands) which contains recordings in 18 different languages.

CommonVoice is an ambitious project to bring automatic speech recognition, voice-based interfaces, and other speech processing technology to the whole planet. Importantly, users can add support for new languages without needing software engineering expertise, as CommonVoice also provides tools for translating their data collection interface to new languages. For data engineers, CommonVoice provides a useful example of how to expand data collection to a worldwide scale.

6.10.1.4 Repurposing Existing Datasets for TinyML

We've seen how challenging it can be to collect brand new datasets. ML research has exploded in popularity and there are many datasets available for a wide variety of tasks already. TinyML research can be accelerated by taking advantage of these existing datasets and repurposing them for embedded tasks. A useful overview of available datasets is provided by TensorFlow's Datasets Catalog, which covers a wide gamut of machine learning problems, and are ready-to-use for training with the TensorFlow API.

In the next section we will develop a vision-based TinyML tool for person-detection. The dataset for this task, Visual Wake Words, was developed from an existing popular dataset, Common Objects In Context (COCO). By specializing an existing dataset, you can avoid the need to collect data from scratch.

We will also discuss transfer learning in the next section. Transfer learning allows researchers to avoid training models from scratch, since the same features are shared across many tasks in speech or vision. It is faster to transfer features than to train from scratch, and transfer learning also greatly reduces the amount of data

which needs to be collected for a new task. Pretrained models can also be used to help curate data for new tasks - ImageNet was built in part using results from Google image search. You may even be able to generate data for your needs using a simulator.

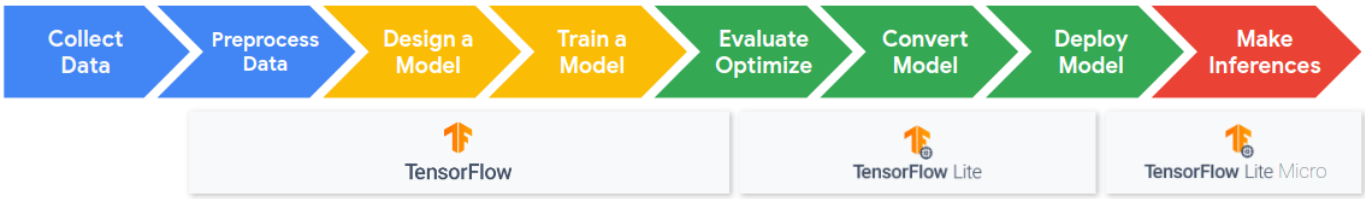
6.10.1.5 Responsible Data Collection

Data engineers must consider how to reduce bias when collecting datasets. We have seen several potential sources of bias in speech recognition - for example, your data may not contain enough examples from various age groups, accents, or gender. Efforts such as CommonVoice attempt to tackle the problem of underserved languages. Data engineers must determine how to debias data collection efforts for a given ML task, and additionally ensure these debiasing requirements are maintained if an intermediary is used in data collection (for example, paying gig workers to collect data on Amazon's MechanicalTurk)

7 Visual Wake Words

7.1 Introduction to Visual Wake Words (VWW) Application

You've now seen the full TinyML flow in the context of a Keyword Spotting application, noting particular challenges with preprocessing audio data, training TinyML models, and designing effective end-to-end systems and metrics. You then explored the many challenges with (and solutions for) creating an effective TinyML dataset.



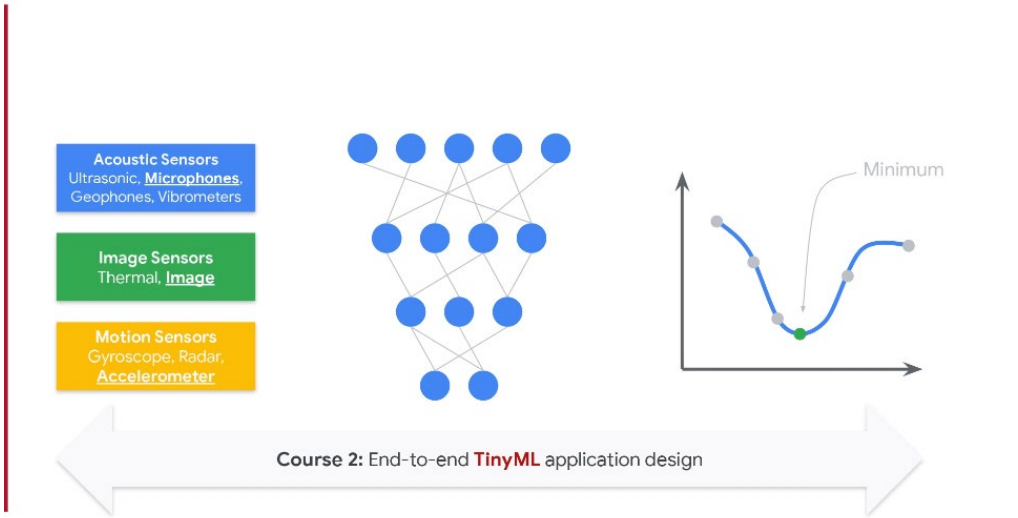
7.1.1 What's the Focus in this Module?

In this section we are going to build on that learning by exploring the TinyML flow and data engineering again in the context of a Visual Wake Words application, focusing on some unique challenges presented by this computer vision application. Visual Wake Words represents a common TinyML visual use case of identifying whether an object (or a person) is present in the image or not.

7.1.2 What's New and Different?

Following what we did for Keyword Spotting (KWS), we will look at several interesting perspectives on the end-to-end VWW TinyML application pipeline. We will understand the characteristics of the camera sensor and how that affects our preprocessing pipeline. We will look into the volume of data generated by the sensor and how that affects our downstream ML workflow. We'll explore a fundamentally different way of doing neural network calculations to make things tiny so they are more efficient on MCUs. Specifically, we will introduce Mobilenets, a new class of models that leverage Depthwise Separable Convolutions to use less memory and reduce the total number of computations. We'll then discover a whole new way to train models, Transfer Learning, which builds off of an existing pre-trained model to drastically reduce training time. Finally we'll explore the different end-to-end systems challenges and metrics present in this novel application. We hope you enjoy this section!

7.2 What are Visual Wake Words (VWW)?



Next **big** thing!

Augmented Reality

- Smart Shopping

Next **big** thing!

Augmented Reality

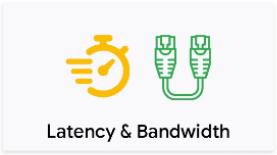
- Smart Shopping
- Navigation

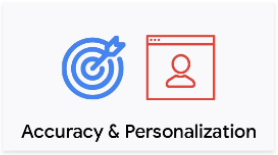
What are we going to learn?

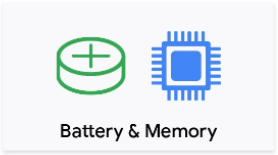
Challenges with
Visual Wake Words

Opportunities with
Visual Wake Words

Challenges



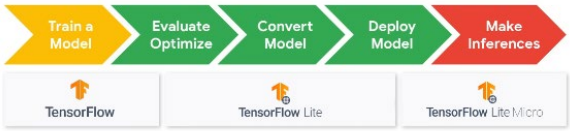




Opportunities

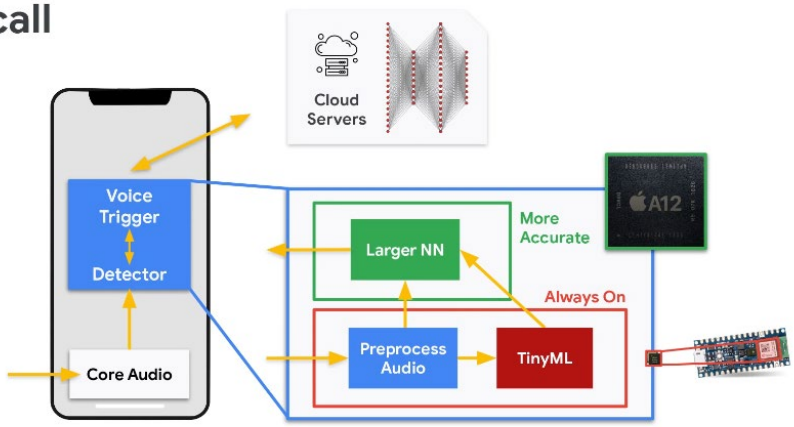
Run visual wake words
on-device

What are we going to learn?

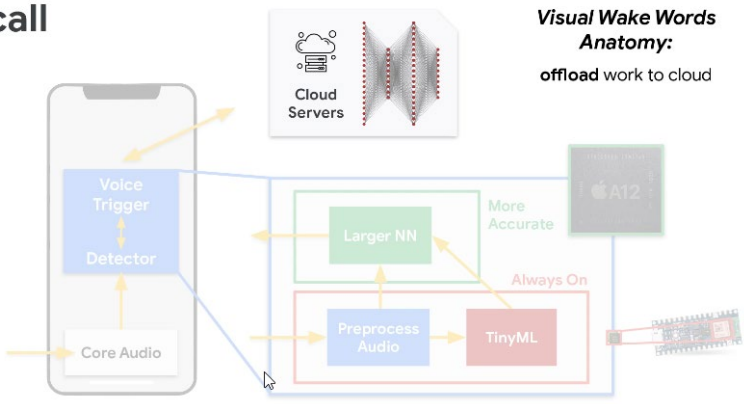


7.3 Visual Wake Words Challenges

Recall



Recall



Simple **Experiment**

$$224 \times 224 \times 3 \times 4 = 602{,}112 \text{ Bytes}$$

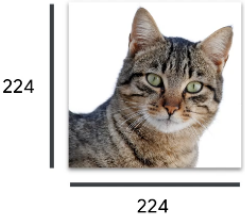
Pixels | RGB (# channels) | Bytes/Pixel

Simple **Experiment**

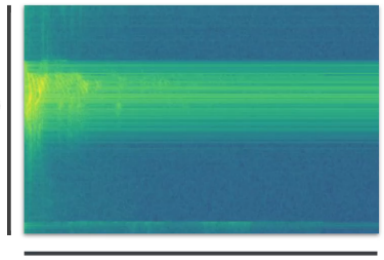
602,112 Bytes

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

~1 second Transfer Time



Simple **Experiment**



$49 \times 40 \times 1 \times 4 = 7{,}840$ Bytes

Pixels | RGB (# channels) | Bytes/Pixel

$224 \times 224 \times 3 \times 4 = 602{,}112$ Bytes

Pixels | RGB (# channels) | Bytes/Pixel

Simple **Experiment**

Always-on (Visual Wake Words)?

→ Much more data (than KWS)
- Higher **latency**
- Higher **power consumption** (drains battery)
  → Lower **user satisfaction**

Anatomy of a Visual Wake Words

Latency




Constraints for Visual Wake Words

Processing Latency

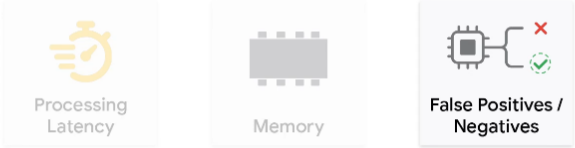
Memory

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

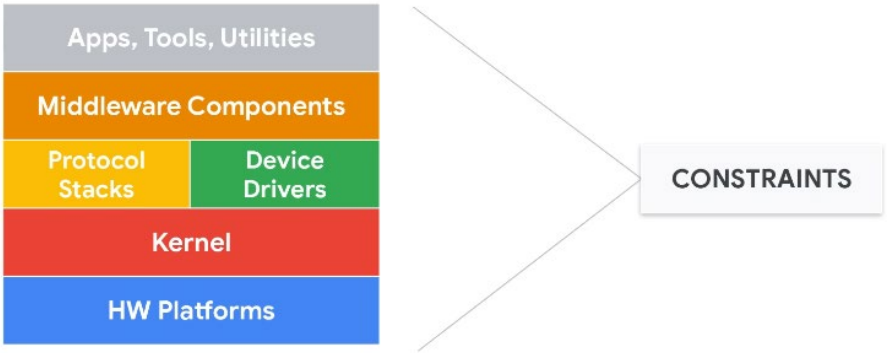
Our board [Course 3 Kit] only has **256 KB** of RAM (memory)

Constraints for Visual Wake Words

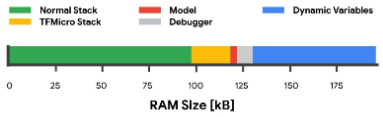


Errors: False positives/negatives

Multiple Layers to Compute Stack



Multiple Layers to Compute Stack

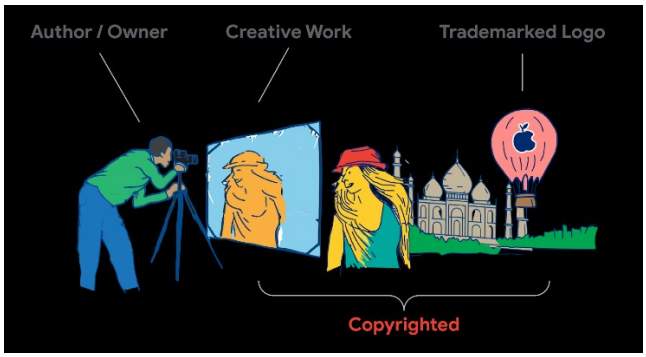


Memory: Model + ***Rest of Stack***

Multiple Layers to Compute Stack

Latency: Model + ***Rest of Stack***

7.4 Data Privacy with Images



The growth of social media has heralded an era of online data sharing, wherein the majority of daily interactions are conducted between technological intermediaries. While this provides huge benefits in terms of productivity, communication, and accessibility, it also presents important privacy challenges.

7.4.1 A Brief History of Data Privacy

Since the human rights movement following the events of World War II, strict ethical requirements were developed which must be adhered to by social scientists and medical practitioners when performing experiments on humans. The prime example of these is the Belmont Report, created in 1978. This is true regardless of their impact, be it physical, psychological, or emotional. Although data science and machine learning practitioners do not directly experiment with human subjects, their impact can be commensurate with those of the social science community. Every time an e-commerce site changes the layout of their site using an A/B test, they are conducting a large-scale social experiment.

One of the key concerns of such requirements is the right to individual privacy. The breadth of data that can be obtained about human subjects can be highly valuable to businesses as well as the research community, but requires informed consent of the individual to be used. Today, companies like Google and Facebook have huge amounts of data based on the way users interact with their services. This information is often then marketed to companies and subsequently used for targeted advertising campaigns. The right of the company to use this information is embedded within their terms of use for their respective services. Whilst these uses might seem unethical, especially to privacy advocates, they are entirely legal and abide by the ethical conventions of Belmont Report, as well as more modern ethical requirements developed during the age of big data, such as the Menlo Report.

However, the principle of informed consent has become increasingly difficult to ensure in the modern world. A good example is genetic information. By uploading genetic information to online platforms such as 23AndMe, an individual is voluntarily waiving their right to privacy. However, genetic information is largely similar between family members, and thus by uploading this information, the privacy of relatives is also violated. A famous example of this was the arrest of the “Golden State Killer”, who went uncaught for decades until DNA information from an online genetic database was linked to a relative of the murderer using DNA from the crime scene. Although this demonstrates a positive use of such data, its potential power is unsettling.

7.4.2 Data Privacy in Images

Image data is one of the most vulnerable mediums of online data. This is troubling as it is also one of the most commonly shared. Datasets may contain images of people curated from online resources wherein the user did not obtain informed consent from the individuals who own or are present in the images. Perhaps the most alarming feature is that of specific types of image which store geographical coordinates of the location a photo was taken, such as GeoTIFF or EXIF.

Most individuals would shrug this off as being harmless, since it has very minor ramifications to their personal life. However, with sufficient knowledge this information can be used to obtain a great deal of personal information. Images uploaded by individuals which are then voluntarily shared can present privacy violations to other individuals present in the image. This information can be used to find personal associations, and combined with image data to determine commonly visited locations or the homes of an individual or their relatives and friends.

This issue is well highlighted by a somewhat amusing and yet concerning study by two Harvard undergraduates looking at images on the Dark Web. By studying images of drugs scraped from websites on the Dark Web (websites only accessible using onion routing on a specialized web browser called Tor), the undergraduates were able to produce a map and isolate the approximate locations of drug dens. Most smartphones today attach this geographical information to pictures automatically, unbeknownst to the average user, and is easily extractable from a scraped image.

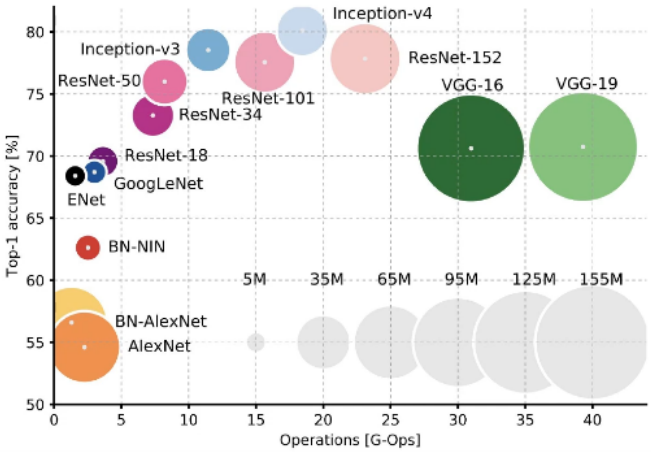
7.4.3 Relevance to TinyML

As data scientists and machine learning practitioners, it is important for us to find ways to uphold the ethical foundations set by our forebearers. One of the best ways for this to be done is to understand the ethical ramifications of our actions, and to try and safeguard against situations where such transgressions may manifest. For example, when curating a publicly available dataset, ensuring that images are within the public domain or by entering into a third party agreement with the original data provider. If the dataset is to be used for computer vision, the geographical data provides no utility, and thus should be removed in order to minimize potential harm to the individuals that produced the original data. Similarly, if images contain personal or revealing information about an individual, they should not be used in a dataset. As a rule of thumb, the minimum amount of data should be utilized to obtain the desired result. If the provenance of a dataset is uncertain or the licensing ambiguous, it is important to err on the side of caution and either confirm the provenance or disregard the data.

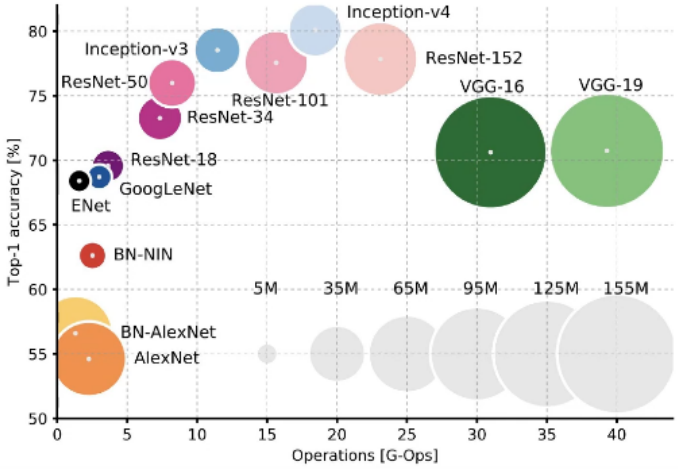
This is true not just for during model training but also inference. In our TinyML visual wake words application, we use a camera which is able to take images in real-time. These images are transferred to a framebuffer which then performs inference on the newly obtained image. In the majority of TinyML applications, it is infeasible to obtain informed consent for each person that may potentially fall inside the image, presenting an ethical dilemma. For example, if an entity would like to save images from a smart doorbell for use at a later time, either for future training data or for diagnostics, it may violate the above-mentioned ethical principles. These images may also be saved with metadata, which may further compromise privacy. Thus, we must be mindful when developing these systems to ensure that the obtained images are only for real-time detection and are not archived, or are covered by additional provisions.

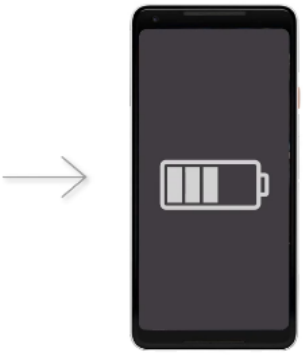
7.5 Neural Network Architectures for Visual Wake Words

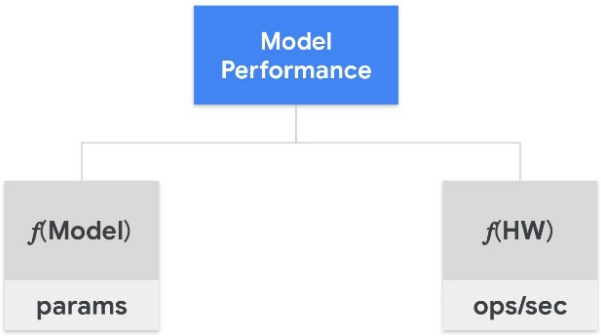
Model **Evolution**



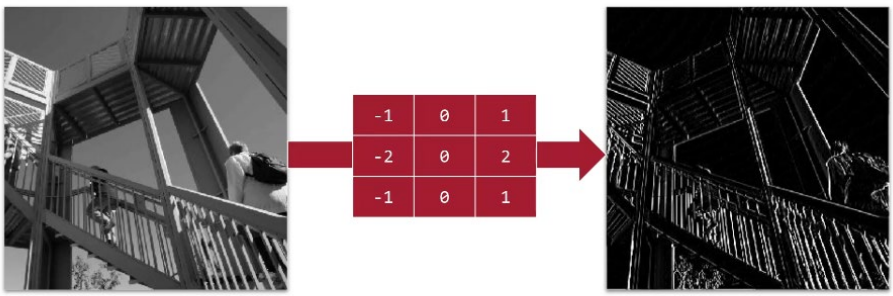
Model **Evolution**



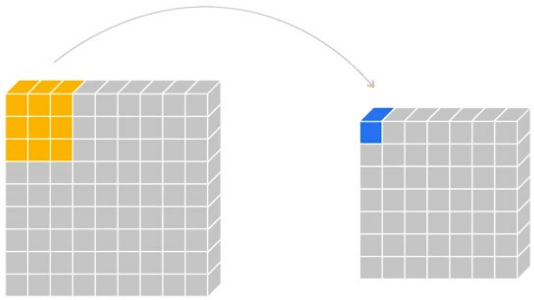




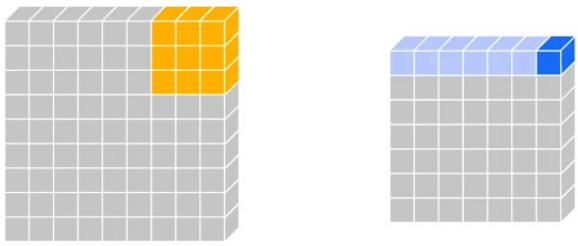
Recall: **Convolutions**



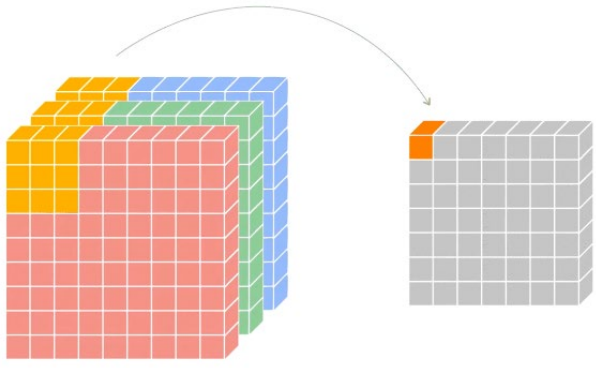
Standard Convolution ***(1 Channel)***



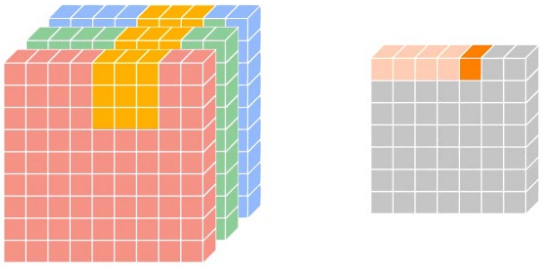
Standard Convolution ***(1 Channel)***



Standard Convolution ***(3 Channel**—e.g., RGB)*

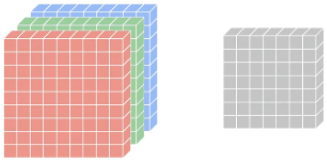


Standard Convolution ***(3 Channel**—e.g., RGB)*

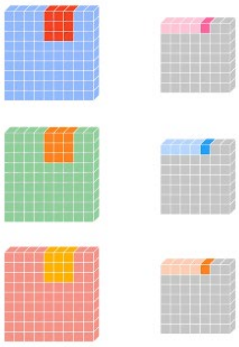


Standard Convolution ***(3 Channel**—e.g., RGB)*

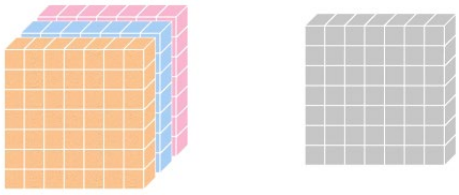
- Input Feature Map
  - 8 × 8 × 3
  - Width × Height × Channels
- Kernel (*1 Filter*)
  - 3 × 3 × 3



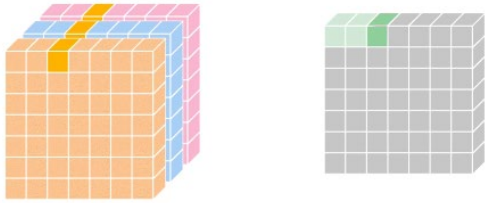
Depthwise Convolution ***(3 Channel**—e.g., RGB)*



Pointwise Convolution



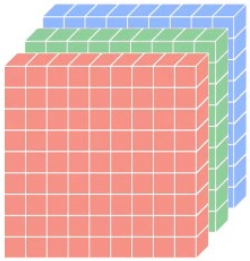
Pointwise Convolution



separable

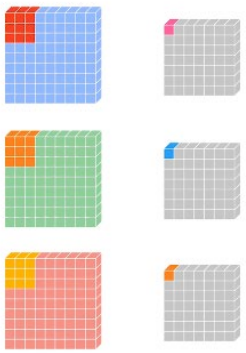
Depthwise Convolution ***(3 Channel**—e.g., RGB)*

includes **pointwise** conv



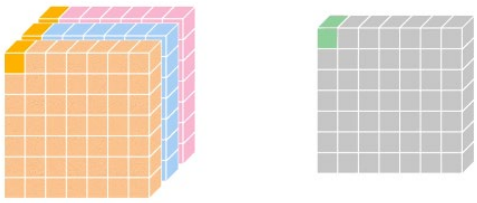
separable Depthwise Convolution ***(3 Channel**—e.g., RGB)*

includes **pointwise** conv



separable Depthwise Convolution ***(3 Channel**—e.g., RGB)*

includes **pointwise** conv

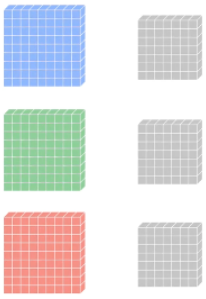


separable Depthwise Convolution ***(3 Channel**—e.g., RGB)*

includes **pointwise** conv

Benefit?

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



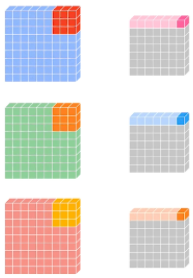
separable

Depthwise Convolution *(3 Channel—e.g., RGB)*

includes **pointwise** conv

Benefit?

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



separable

Depthwise Convolution *(3 Channel—e.g., RGB)*

includes **pointwise** conv

Benefit?

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

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

Filters

Kernel (filter) Dimensions

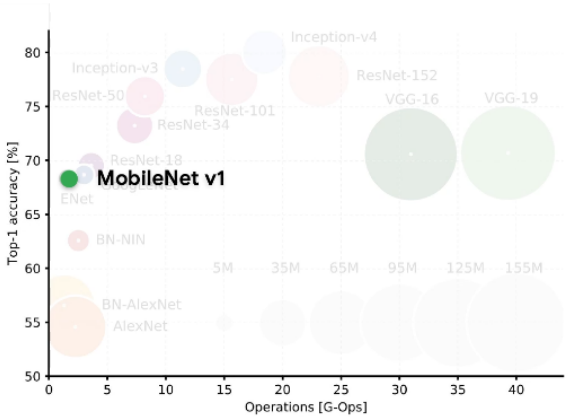
MobileNet v1

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko
Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam

Google Inc.
{howarda,menglong,bochen,dkalenichenko,weijunw,weyand,anm,hadam}@google.com

Model **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

Our board [Course 3 Kit] only has **256KB** of RAM (memory)

Further **Optimizations**

- Effect of **depth multiplier** on model size ⟶ top-1 accuracy
- The size of the model can be reduced further by parameter, ***α***
- ***α*** ⟶ (0, 1]

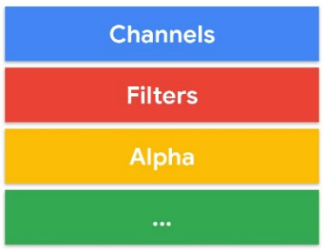
$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$$

Further **Optimizations**

Multiply-Accumulates

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

Neural Architecture Search **(NAS)**



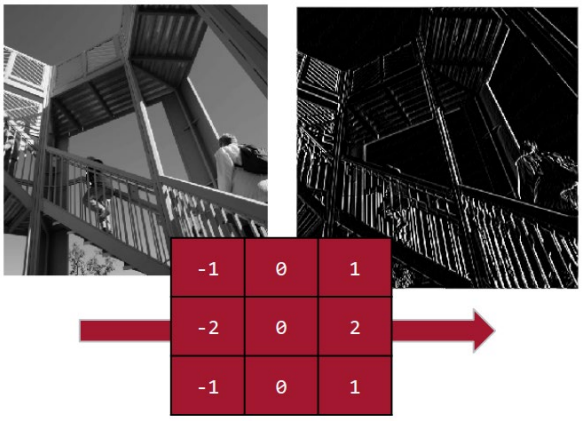
7.6 The Math Behind MobileNets Efficient Computation

Now that you've seen that MobileNets drastically reduce the size of computer vision neural networks let's dive a little deeper into how their key innovation works: Depthwise Separable Convolutions.

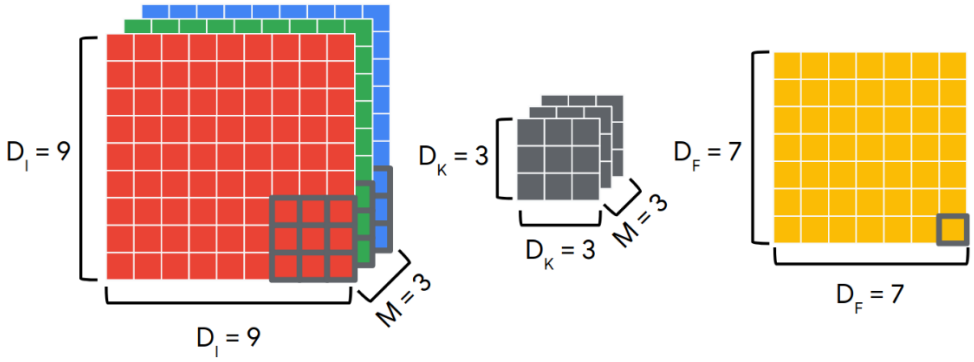
7.6.1 Standard Convolutions

In order to better understand how depthwise separable convolutions are different from standard convolutions let's first re-examine standard convolutions. In particular, we are going to quantify the number of multiplication operations and parameters in a standard convolution.

In the last course, when we considered filters in detail we only looked at filters applied to grayscale images like the vertical line filter shown below:



These images can also be called single-channel images as they only have one set of pixel values. In contrast, most color images are three-channel RGB images. Where there is a value representing how much red, green, and blue is in the color. As such the filters (also often referred to as kernels) used are not simply a matrix but instead a tensor as it needs to multiply all three channels at the same time. This tensor operation is shown below. In this example, we have a 3x3x3 kernel being convolved with a 9x9x3 image to produce a 7x7x1 output. The highlighted squares are the inputs and outputs of the last convolution operation.

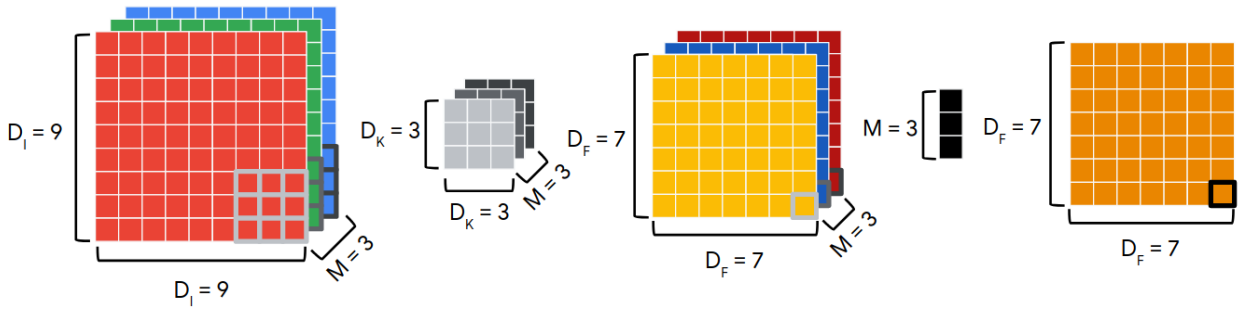


Note that in general, each convolution requires Dk * Dk * M ultiplications due to the size of the kernel and the number of channels in the image. Then to produce a single output we need to do these operations Df* Df to produce the full output. Also in general we don't just use a single kernel ,we use N kernels. Multiple kernels are referred to as a filter. A filter is a concatenation of multiple different kernels where each kernel is assigned to a particular channel of the input. As such the total number of multiplications will be:

$$D_K * D_K * M * D_F * D_F * N = D_K^2 * M * D_F^2 * N$$

7.6.2 Depthwise Separable Convolutions

Depthwise Separable Convolutions instead proceed in a two-step process. First, each channel is treated independently as if they were separate single-channel images and filters are applied to them creating multiple outputs. This is referred to as a Depthwise Convolution. Next, a Pointwise Convolution is applied to those outputs using a 1x1xC filter to compute the final output. This process can be seen below where we again take our 9x9x3 image and apply three separate 3x3 filters to it depthwise to produce a 7x7x3 output. We then take that output and apply a 1x1x3 pointwise filter to it to produce our final 7x7x1 output.



Note that in general now each depthwise convolution requires M filters of Dk*Dk multiplications and to produce a single output we need to do these operations Df*Df times. Therefore we need Dk*Dk*M*M*Df*Df multiplications for that stage.

For the pointwise convolution, we now use a 1x1xM filter Df*Df times. In general, just like regular convolutions, we don't use a single filter we will use multiple filters. During Depthwise Separable Convolutions those multiple filters occur in the pointwise step. Therefore if we had N pointwise filters we will then need M*Df*Df*N total multiplications on this step.

Summing the number of multiplications we will need in both stages we find that in total we need: Dk*Dk*M*Df*Df+M*Df*Df*N

$$D_K * D_K * M * D_F * D_F + M * D_F * D_F * N = M * D_F^2 * (D_K^2 + N)$$

Comparing the two kinds of Convolutions

We can compare the two kinds of convolutions through a ratio of the number of multiplications required for each. Placing standard convolutions on the denominator we get:

$$\frac{\text{Depthwise Separable}}{\text{Standard}} = \frac{M * D_F^2 * (D_K^2 + N)}{D_K^2 * M * D_F^2 * N}$$

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

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

This means that the more filters we use and the larger the kernels are, the more multiplications we can save. If we use our example from above where DK=3 and we use conservatively only N=10 filters we will find that the ratio becomes 0.2111 meaning that by using Depthwise Separable Convolutions we save almost 5x the number of multiplication operations! This is far more efficient and can greatly improve latency.

Also, note that in the case of standard convolution we have

$D_K^2 * M * N$ learnable parameters in our various filters/kernels. In contrast in the Depthwise Separable case, we have

$$D_K^2 * M + M * N$$

Again if we take the ratio of the two we find that:

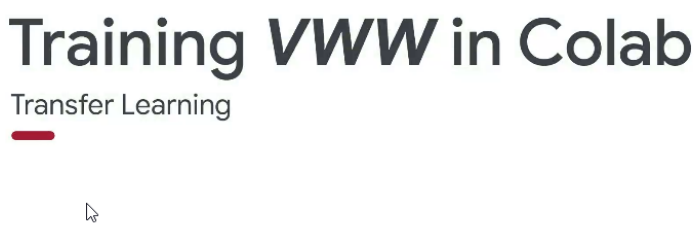
$$\frac{\text{Depthwise Separable}}{\text{Standard}} = \frac{1}{N} + \frac{1}{D_K^2}$$

This means that we also have a much smaller memory requirement as we have far fewer parameters to store!

There is a tradeoff however, in improving our latency and memory needs we have reduced the number of parameters that we can use to learn with. Thus our models are more limited in their expressiveness. This is usually sufficient for TinyML applications but is something to consider when using Depthwise Separable Convolutions in general!

Finally, if you'd like to read more detail about MobileNets you can check out the paper describing them here.

7.7 Transfer Learning for VWW



Training Pipeline: **Need Lots of Data**

1000 Classes

1000 Images / Class

Training Pipeline: **Need Compute Resources**

Memory

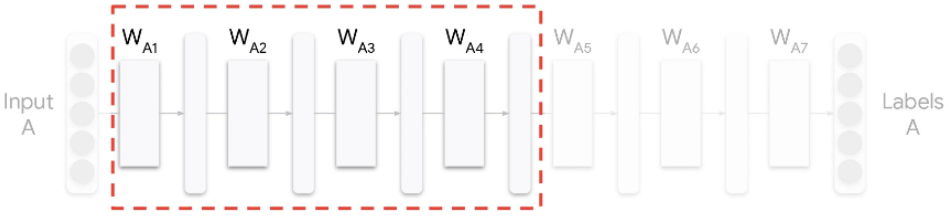
Compute

GPU and Accelerators

End Result of Training

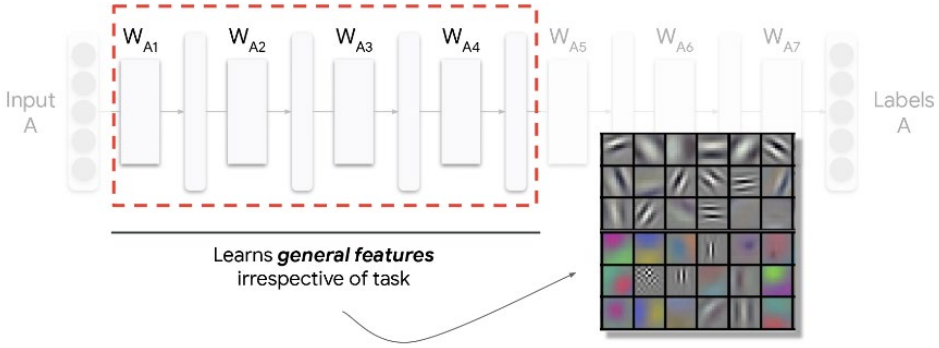
ž

End Result of Training

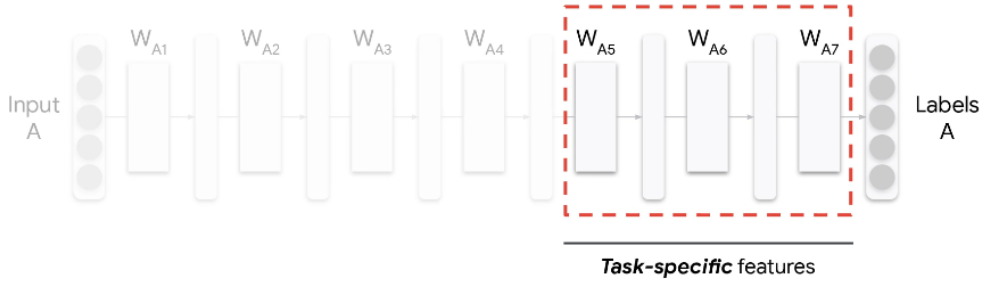


Learns ***general features***
irrespective of task

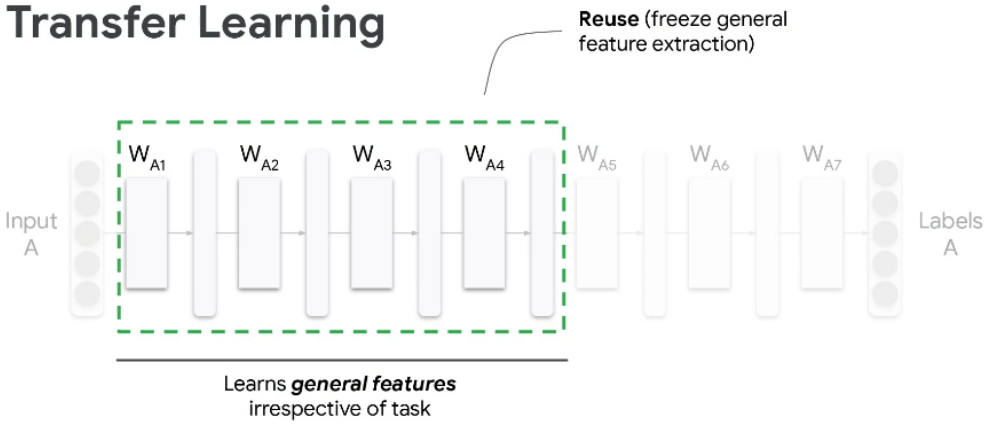
End Result of Training

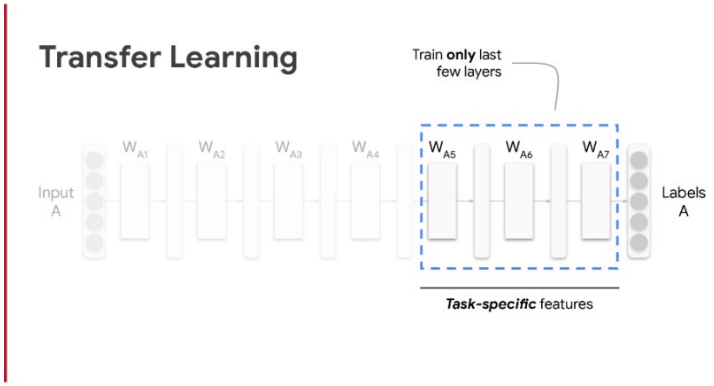


End Result of Training



Transfer Learning





7.8 Assignment: Transfer Learning in Colab

Now that you have explored vision applications for TinyML, the MobileNet model, and transfer learning, we are going to put it all together and train a model to help fight COVID-19. We are going to detect if people are wearing masks or not!

In this Colab you will see through a hands on example how to leverage a pre-trained model as a feature extractor, and how to add new layers and train them to use for classification on a new task. You will also be challenged to find the right number of epochs when using transfer learning. Hint: it's going to be a pretty small number!

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-7-11-Assignment.ipynb

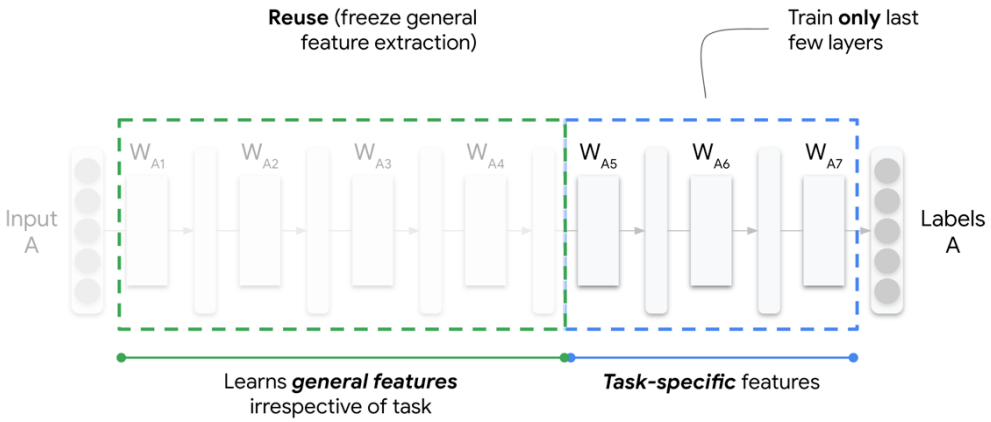
7.8.1 Assignment Solution

You should have found that you only needed VERY FEW epochs to get the model to train very accurately. Depending on how lucky you were with initialization it could have been in the single digits! We found that by setting EPOCHS = 10 our models were always sufficiently trained, and in some cases we were even able to get EPOCHS = 2 to work!

We hope you enjoyed exploring the power of transfer learning! As always if you would like to go back and try the assignment again, it can be found here:

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-7-11-Assignment.ipynb

7.9 Common Myths and Pitfalls about Transfer learning



Hopefully, you have now been convinced of the power of transfer learning. Transfer learning allows us to use a model generated for one task to be fine-tuned for a different but related task. In doing this, we can repurpose models for use in new tasks, thereby saving energy that would be required in computation. Furthermore, models with superior performance can be developed with a relatively small amount of data, which would normally cause overfitting in a network trained from scratch. Consequently, transfer learning provides a trifecta of benefits: reduced energy usage, faster convergence, and higher asymptotic accuracy. However, transfer learning can be difficult to implement and is not always guaranteed to be successful.

7.9.1 Task Similarity

Transfer learning is only viable when the task of interest is similar in scope to the original task. Image recognition is the archetypal example of this, wherein the convolutional filters close to the input can be ported to many imaging tasks due to their generality (e.g., spotting lines, edges, shapes). The further into the convolutional neural network we get, the more specific the filters become and the less transferrable they become.

Tasks that are too dissimilar may not receive any benefit from transfer learning. In fact, the use of transfer learning may even be detrimental to the final model performance. For example, performing transfer learning on a neural network used for language translation may perform well if done between two similar languages, such as two Romanic languages (e.g., French and Spanish), but may perform poorly if done between two dissimilar languages (e.g., Mandarin and French). This situation is commonly referred to as negative transfer.

7.9.2 Fragile Co-Adaptation

Transfer learning involves the use of neuron weights from one trained network to pre-initialize neuron weights in a second network for a related task. Often, the weights of several of the first layers are frozen so that they cannot be changed during learning. The rationale for this is to prevent noise in the new dataset from detrimentally altering the earlier layers. However, freezing weights in itself can be detrimental to model performance as it can lead to fragile co-adaptation of neurons between neighboring frozen and unfrozen layers. This occurs when neurons in the former layers are fixed and the latter layers alone are unable to adapt effectively to the new data. In practice, this is often solved by not freezing any layers and instead using a significantly smaller learning rate used when training the original network. This procedure helps to ensure former layers are not altered significantly while reducing the possibility of fragile co-adaptation.

7.9.3 Fixed Architecture

A key disadvantage of transfer learning is the restriction on neural architecture. Transfer learning is often performed from well-known models developed by commercial entities such as Inceptionv3, ResNet, and NASNet. These models are trained with specific neural architectures which are reflected in the model weights. Model layers cannot be modified, removed, or added without these changes propagating across the network. As such, if the network is altered, we cannot have confidence that the remaining parameters are still able to accurately model the data. This presents important limitations for model tuning and, more importantly for tiny machine learning applications, also for model size. These well-known architectures are large and often cannot be compressed sufficiently to fit within the hardware constraints of embedded systems. Thus, the use of transfer learning for tiny machine learning applications is fundamentally limited by existing model architectures unless the resources are available to train bespoke models for transfer learning using our own pre-defined architecture.

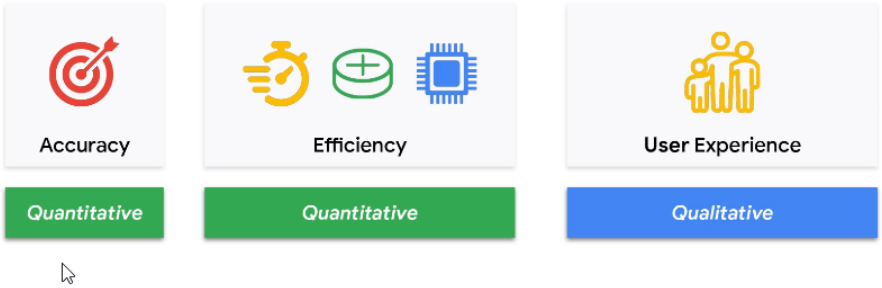
In summary, here is a general rule of thumb of when transfer learning works well when applying transfer learning to your situation where you have a new dataset that you wish to fine-tune your network upon.

	New dataset is similar to original dataset	*New dataset is (less) similar to original dataset*
New dataset is small	Best case scenario for transfer learning	Not the best scenario for transfer learning
New dataset is large	Transfer learning will work	Training from scratch might yield better accuracy

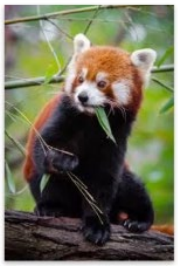
Despite some pitfalls, it is clear that transfer learning is a hugely powerful concept and, although difficult to implement in practice, presents many exciting opportunities for tiny machine learning applications.

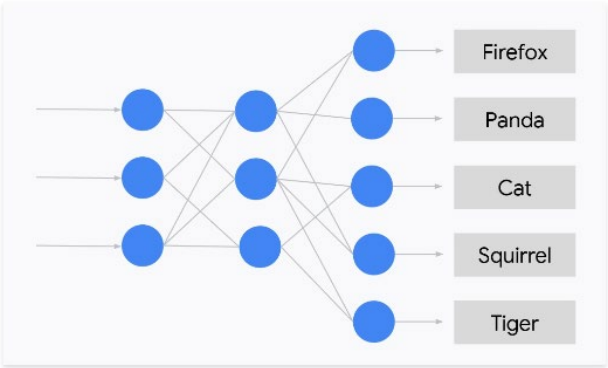
7.10 Metrics for VWW

Common Metrics



Model **Accuracy**

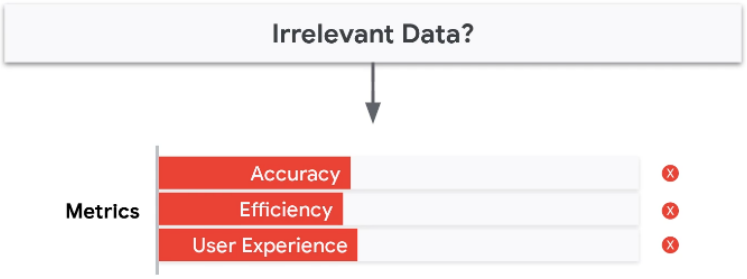




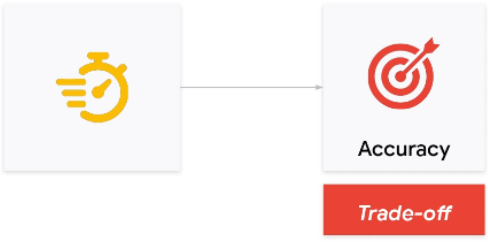
Accuracy

Representative (relevant) data is important because it affects the accuracy of the model

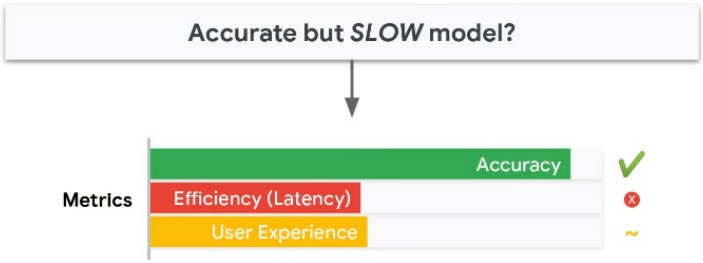
Accuracy



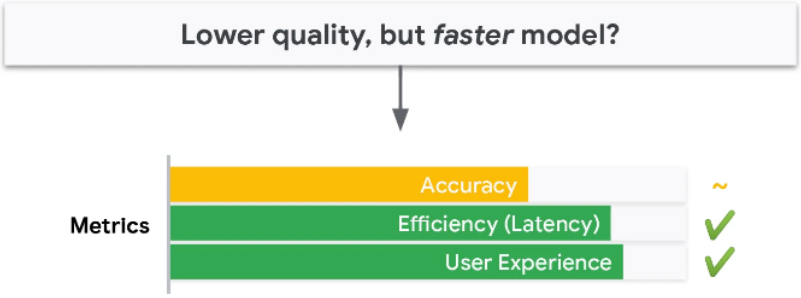
Latency



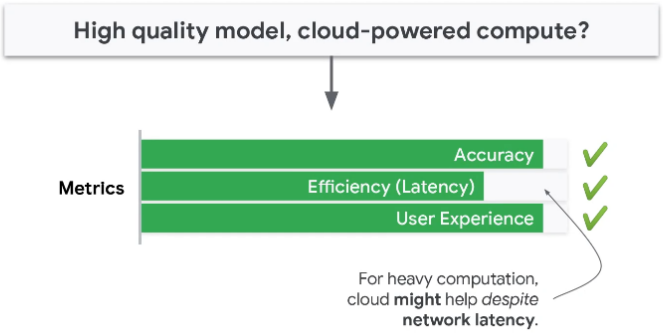
Latency



Latency



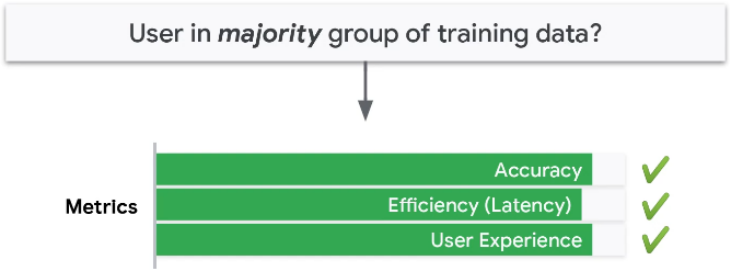
Latency



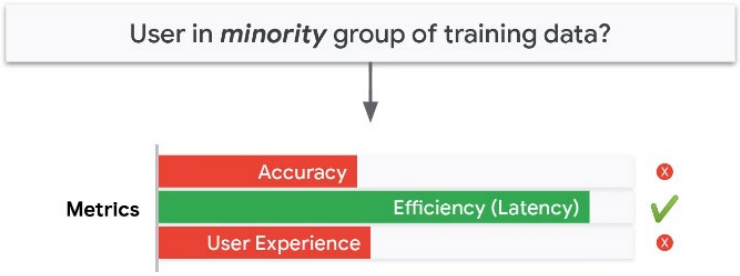
Fairness

***Diverse*, representative data** is important because it enables fair use (equal performance) across populations

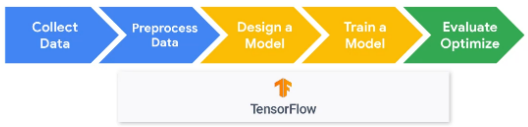
Fairness



Fairness

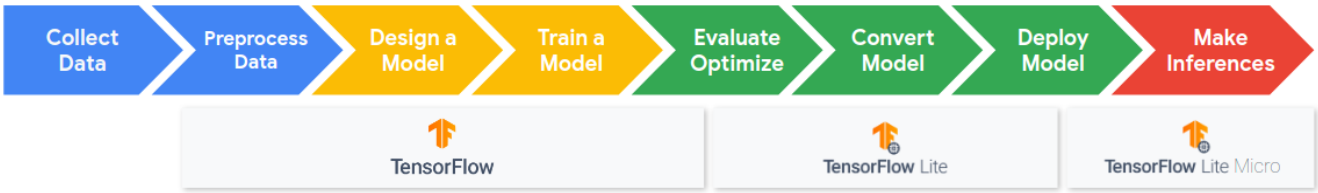


Achieving Ideal Metrics: Revisit pipeline



7.11 Summary Reading

In this section you explored the TinyML flow and data engineering in the context of a Visual Wake Words application, focusing on some unique challenges presented by this computer vision application. Visual Wake Words represents a common TinyML visual use case of identifying whether an object (or a person) is present in the image or not.



7.11.1 Challenges

You explored how latency is a strong constraint for VWW applications as images are large (much larger than spectrograms from KWS) and take a long time to send to the cloud. You also explored how memory is also a strong constraint as images, and their traditional models, take up a lot of memory. Finally, you saw how false positives and false negatives can be particularly challenging in the context of computer vision.

7.11.2 Datasets

In the context of datasets your explored three main topics: licensing, privacy, and reuse. Licensing is a major issue for computer vision applications as while images are readily available, they cannot often be used freely without violating copyright. In terms of privacy, you explored how image metadata can often leak significant information about people. Finally, you explored how the VWW dataset reuses the COCO dataset by sub-sampling images using bounding boxes.

7.11.3 MobileNets

You explored how MobileNets use Depthwise Separable Convolutions to reduce the model size and number of operations, at the cost of a decrease in maximal expressiveness. This reduction is dependent on the size of the kernel and the number of kernels used according to the following formula:

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

You also explored how reducing the depth multiplier and the input image size can also be used to trade off model memory and latency with accuracy.

7.11.4 Transfer Learning

You next explored how transfer learning can be used to drastically reduce the training time and data requirements. Transfer learning works by reusing the first few layers of a pre-trained model for a task using a

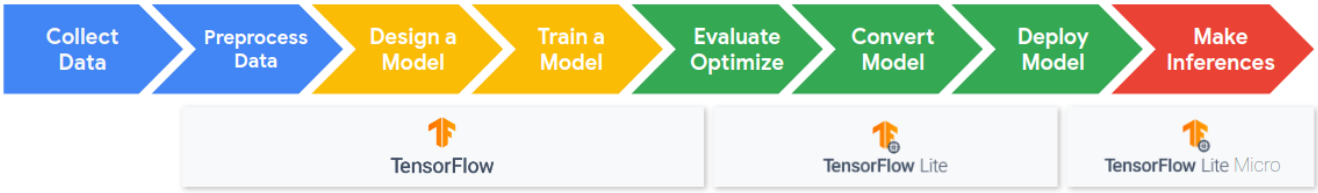
similar dataset and re-training only the later layers. This works if the two datasets are close enough together as the first few layers learn to extract generic features from the data while the latter layers learn to extract task specific features. You also learned, however, that transfer learning is not a panacea and is ultimately also limited by the structure of the pre-trained model. You then got to explore using transfer learning to train your own VWW application in Colab!

7.11.5 Metrics

Finally, you explored both qualitative and quantitative metrics for VWW applications. In particular we discussed the latency accuracy tradeoffs often found in VWW applications and the challenges of getting fair results across data represented by both the majority and minority of training samples.

8 Anomaly Detection

8.1 Introduction to Anomaly Detection



8.1.1 What's the Focus in this Module?

In this section we are going to explore a very different kind of TinyML application that is showing great promise for many different application domains, Anomaly Detection.

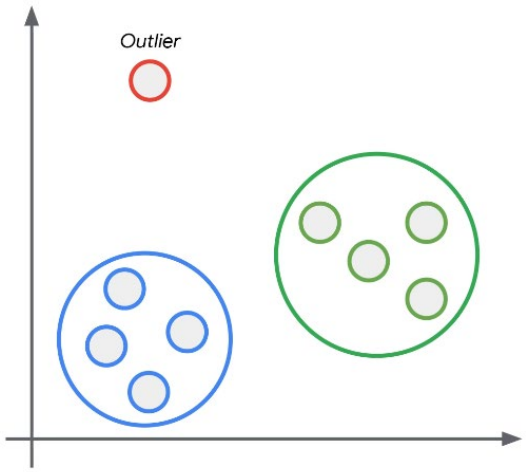
What's New and Different?

Anomaly Detection is different from the previous TinyML applications and datasets that we've explored because we are not looking to match some particular training data, or to use a particular sensor. Instead, we are focused on finding data that we explicitly have not seen before. We'll explore what kinds of data we can use for these challenging and important applications. We'll then explore a new paradigm of machine learning, unsupervised learning, that can be used to solve these problems through the use of both traditional machine learning algorithms, such as K-Means, and neural network based approaches, such as autoencoders. Finally, we'll train one of these approaches, learn how to evaluate performance with new metrics and explore how to select a decision threshold when designing functional applications. We hope you enjoy this section!

8.2 What Is Anomaly Detection

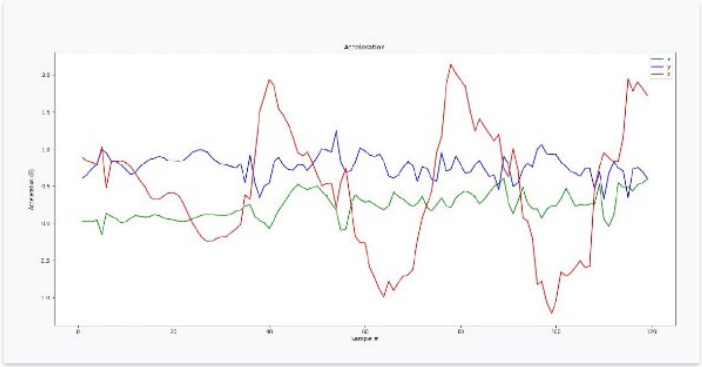
What is **Anomaly Detection**?

In **data analysis**, **anomaly detection** is the **identification of rare** items, events or observations which **raise suspicions** because they **differing significantly** from the **majority of the data**.

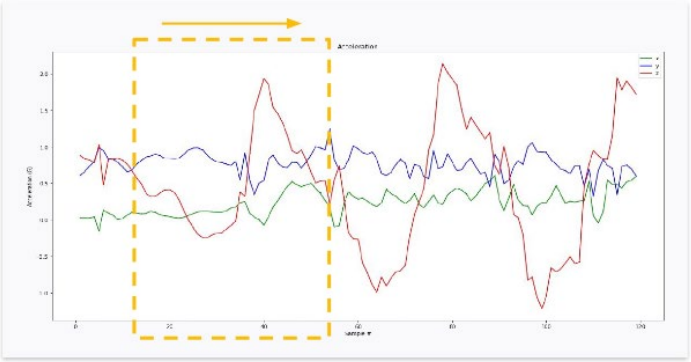


Multiple **Sensors**

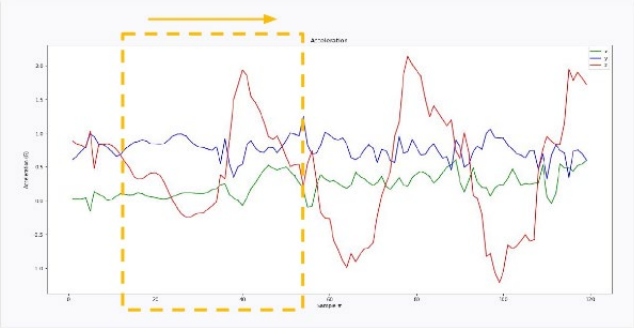
Multiple **Sensors**

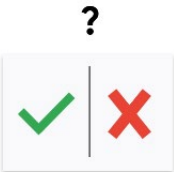


Multiple **Sensors**



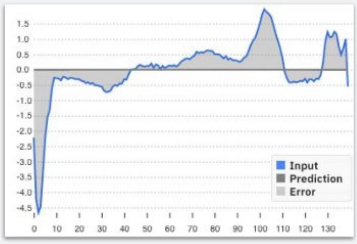
Multiple **Sensors**



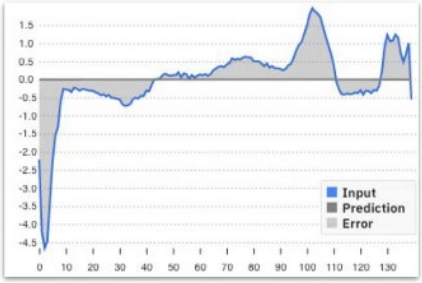


Three Fundamental Aspects of AD

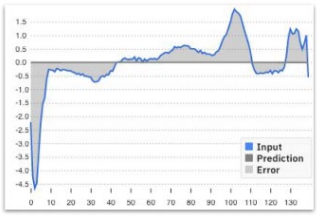
1. Input
2. Prediction
3. **Error**



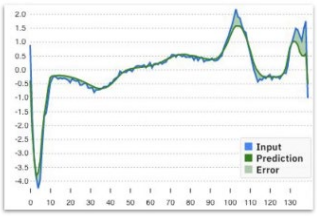
Poor Prediction



Poor Prediction



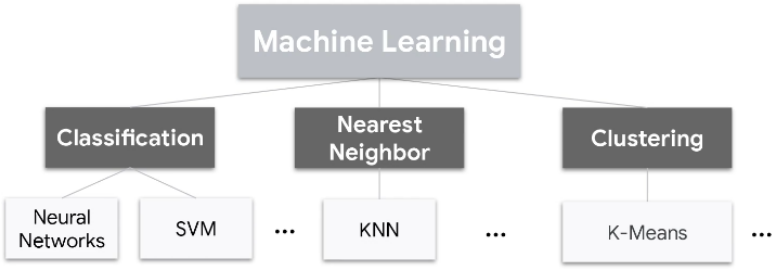
Good Prediction



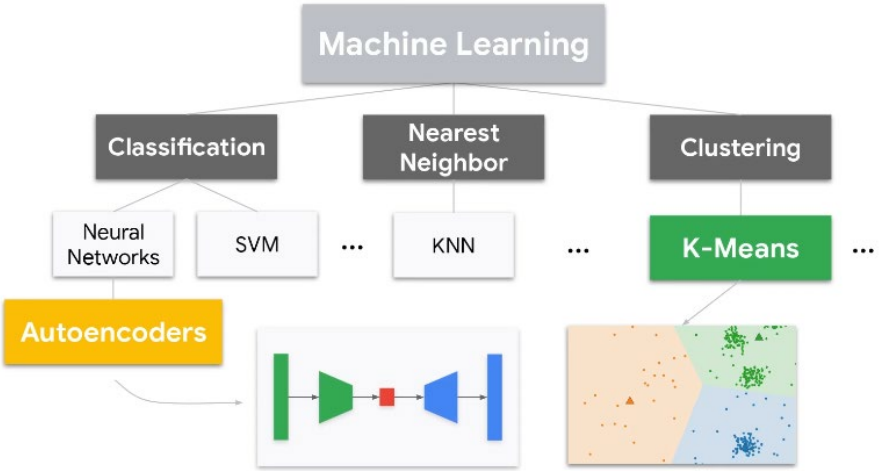
Why **TinyML**?

Machine Learning

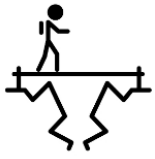
It's **not all** deep learning



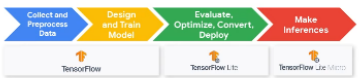
It's **not all** deep learning



What are we going to learn?



Challenges with an Anomaly Detection Application



Anomaly Detection **ML Pipeline**

Training, Testing in Colab

8.3 Anomaly Detection in Industry

What are we going to learn?

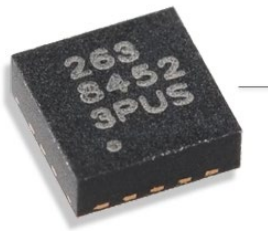
Challenges with an Anomaly Detection Application

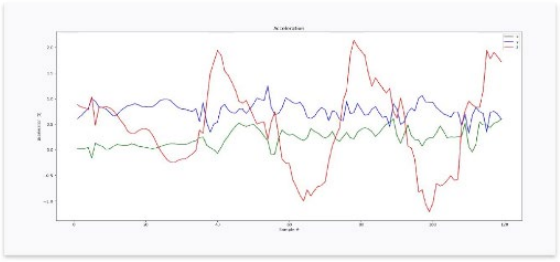
Anomaly Detection **ML Pipeline**

Training, Testing in Colab

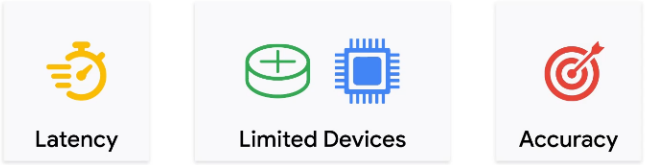
Application: Factory machinery

Sensor: **Accelerometer**

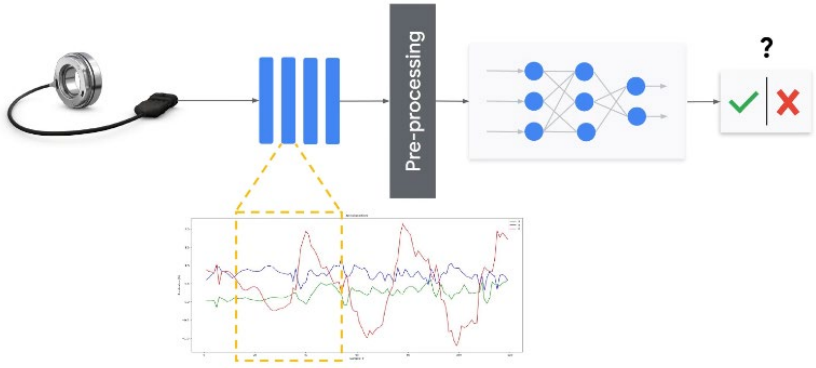




Constraints for **on-device computing**



Real Time Constraint



It's **too expensive** to stream to the cloud

$$2 \text{ bytes} \times 8 \times 20\text{kHz} = \mathbf{320} \text{ KB / sec}$$

Measurement | # Sensors

Sample Rate

It's **too expensive**
to stream to the cloud

2 bytes × 8 × 20kHz = **320** KB / sec

30 KB/s MAX

It's **too expensive**
to stream to the cloud

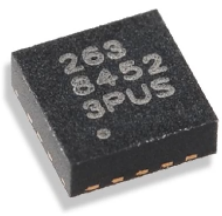
2 bytes × 8 × 20kHz = **320** KB / sec

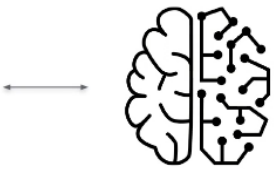
30 KB/s MAX

>

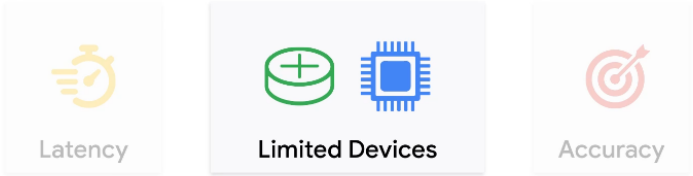
30 KB / sec

Need **"intelligence"**
close **to sensors**

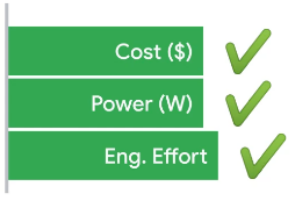




Constraints for **on-device computing**



Very Small (Tiny) Devices

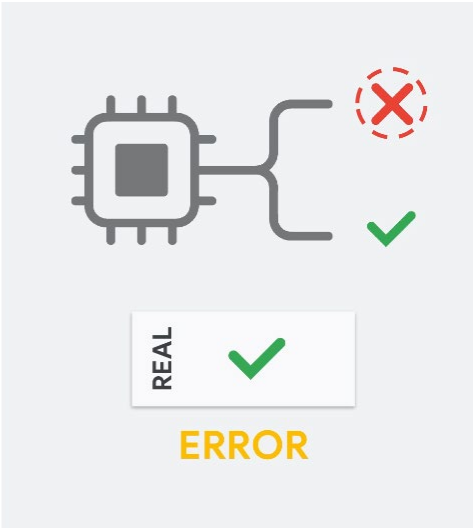


Our board [Course 3 Kit] only has **256KB** of RAM (memory)

Constraints for **on-device computing**

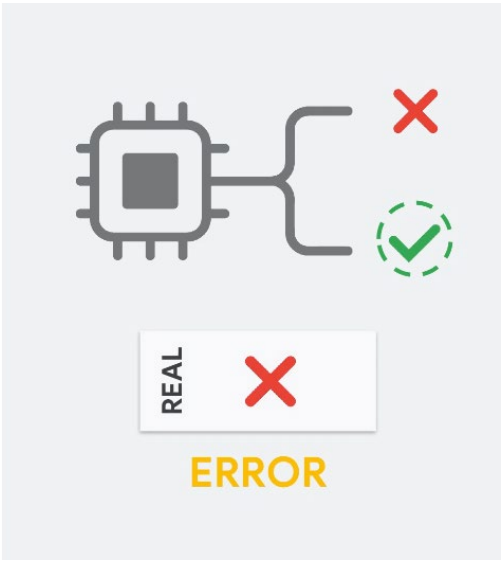
False Negative

Catastrophic impact



False Positive

False alarm, **cost** impact



8.4 Industry 4.0 and TinyML

Possibly the greatest impact of tinyML is likely to occur in industrial sectors. The technology is likely to be both disruptive and enabling, providing large benefits to those who are able to leverage it for effective use in business operations. Some industry professionals prognosticate that the introduction of “intelligent” internet of things (IoT) devices afforded by tinyML is akin to a new industrial revolution, leading some to refer to its future potential as Industry 4.0.

8.4.1 What is Industry 4.0?

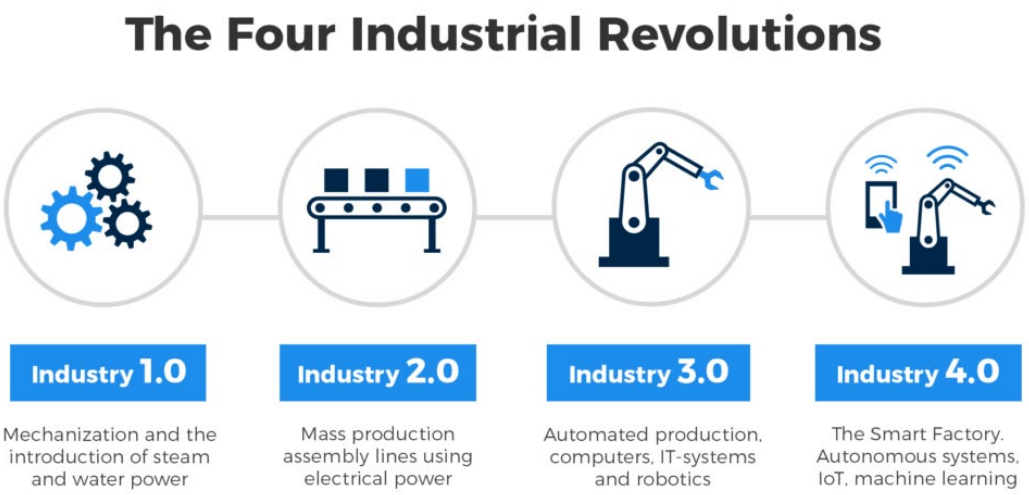
The reason for this name is relatively simple and refers to the fourth industrial revolution. The industrial revolutions refer to specific transformative technologies that revolutionized the industry.

The first industrial revolution started around the 1770’s with the large-scale use of steam and water power, moving from traditional human labor on farms to the first factories and resulting in an exponential increase in productivity.

This continued until the introduction of electricity at the end of the 19th century, when electricity began to be introduced, prompting the second industrial revolution. This revolution kick-started mass production via assembly lines and automation and further increased the productivity of industrial processes, giving us the notion of a factory as we know it today.

This continued until the 1950’s when the digital revolution prompted the third industrial revolution. The introduction of computation allowed industrial processes to become controlled and monitored with greater ease, allowing some of the blue-collar work to be automated.

In recent years, we have started to see a move towards another industrial revolution focused on the interconnectedness of industrial processes. Instead of considering standalone industrial processes, they are considered as part of a network of interconnected processes. Essentially, these processes are in a constant state of communication. This has been enabled by the introduction of intelligent IoT devices tightly coupled with cyber-physical systems, affording capabilities for real-time monitoring and control such as predictive maintenance.



8.4.2 Where does TinyML fit in?

Industry 4.0 encompasses multiple novel technologies, including IoT, blockchains, and tinyML. TinyML, specifically, will come to play a very important role in Industry 4.0.

One reason for this is a reduced network load. In a network of interconnected devices, it is easy for the network to become saturated with information. Such a situation might occur if all data for monitoring purposes has to be continually transmitted to a central repository for monitoring purposes. Instead, TinyML enables individual devices to manage their own data and make intelligent decisions based on this information, thus limiting the communication load on the network.

Another important factor is energy efficiency. As we have discussed previously, transmitting data across devices is energy-intensive. By performing inference on-device and minimizing external communications, we maximize energy efficiency. Given the importance of sustainability in modern industry and the projected exponential growth in IoT usage in the coming decade, this provides an important stimulus for using tinyML.

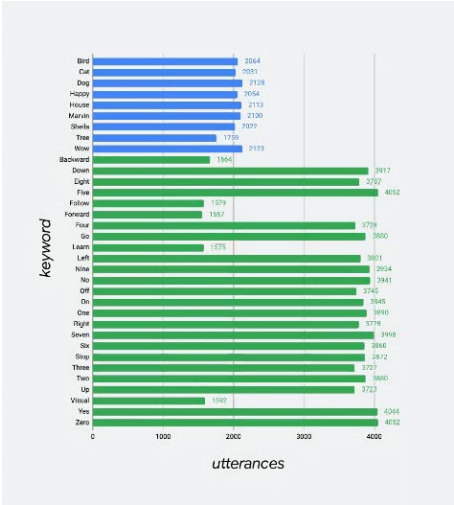
Finally, performing inference on-device minimizes latency. Nowadays, it is common to find systems that upload data from an edge device to a cloud server for processing and then return the processed information to the edge device. This message passing across a network is not only energy-intensive and clogs the network, but is also slow. By performing inference on-device, we are not limited by network speeds and only by the hardware constraints of our devices.

Hopefully, it is now clear that with the introduction of tinyML, the wealth of information procured by edge devices can now be leveraged for intelligent control of industrial processes and business operations. Industry 4.0 essentially provides the capability to perform data-driven optimization of business processes in real-time, allowing inefficiencies to be ironed out almost entirely. TinyML will play a fundamental role for the large-scale use of machine learning in the industry.

8.5 Anomaly Detection Datasets

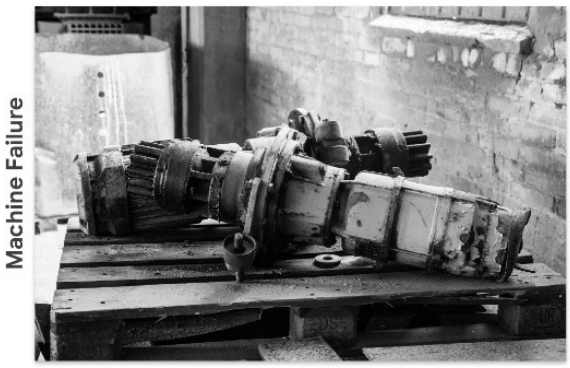
The Speech Commands Dataset

- **25 "IoT keywords"** + **10 "unknown words"**
- The set of words is **known in advance**

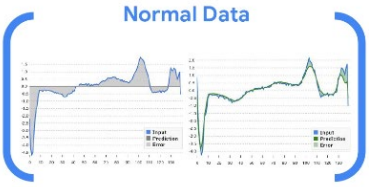


Failed data set is unknown. Failures happen extremely rarely, so data is not easily collectable.

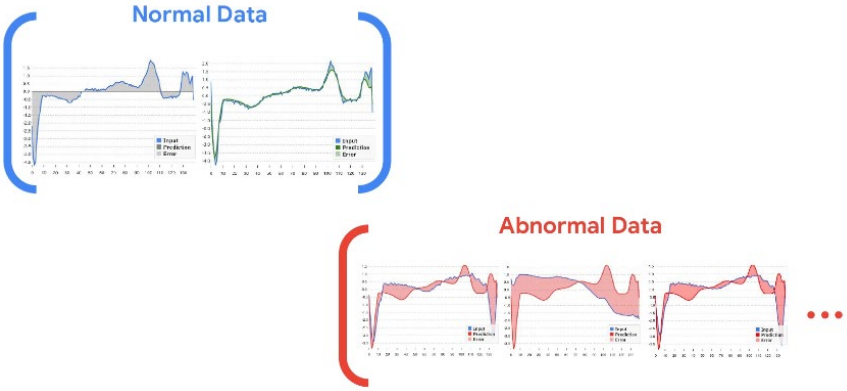
Creating an Anomaly Detection Dataset



How are *Anomaly Detection Datasets* **different**?

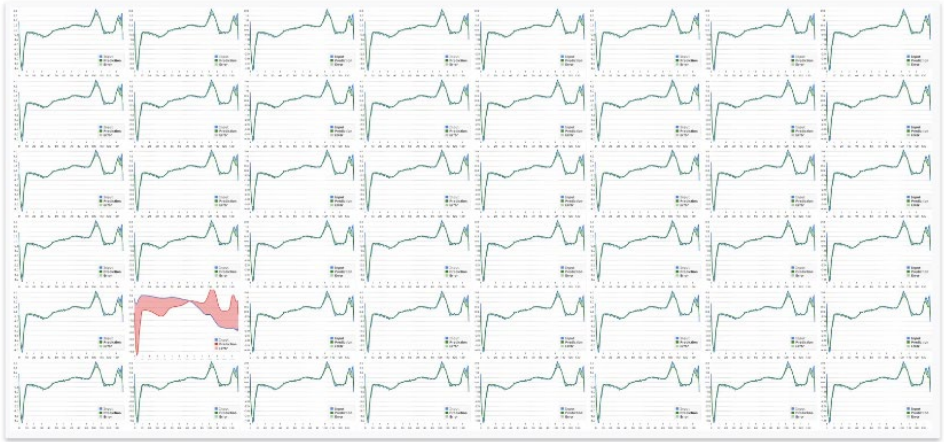


How are *Anomaly Detection Datasets* **different**?



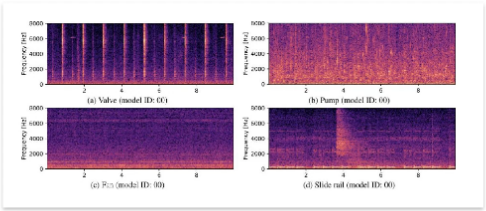
Problem:

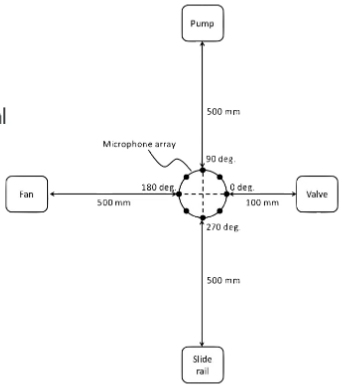
Unbalanced Data



Example: **MIMII Dataset**

- **Sound** Dataset for Malfunctioning Industrial Machine Investigation (MIMII)
- **Anomaly Detection**



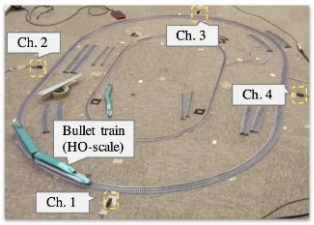


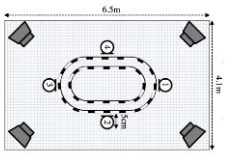
Purohit, Harsh, et al. "MIMII dataset: Sound dataset for malfunctioning industrial machine investigation and inspection." (2019).

Abnormal data set is open dataset. It's UNKNOWN UNKNOWNS!

Example: **ToyADMOS**

- **540 hours** of sound for *anomaly detection*
- Based on **toys**:
  - Car
  - Conveyor
  - Train





ToyADMOS Dataset **Characteristics**

- Designed for **three ADMOS tasks**:
  - Product **inspection** (toy car),
  - **Fault diagnosis** for a **fixed** machine (toy conveyor)
  - **Fault diagnosis** for a **moving** machine (toy train)
- ***Machine-operating sounds*** and ***environmental noise*** are individually recorded for simulating various noise levels
- All sounds are recorded with **four microphones**
  - Testing **noise reduction**
  - **Data-augmentation** techniques such as mix-up

ToyADMOS Dataset **Characteristics**

- Record **multiple machines of same class**
  - Different structure → **Individual variations** despite same toy class
- Record **several times**
  - **Anomalous sound** characteristics from few samples
- **180 hours** of **normal machine-operating sounds** alongside **over 4,000 samples of anomalous sounds**
  - collected at a **48kHz** sampling rate for each task

Low Transferability

- An anomaly detection model will be **very specific** to the training set and therefore **difficult to generalize** to other deployment environments

What are we going to learn?

Challenges with an Anomaly Detection Application

Anomaly Detection **ML Pipeline**

Training, Testing in Colab

8.6 MIMII DATASET

n this reading assignment, we want you to read through the “MIMII Dataset” paper. The paper provides additional information on some of the details that we discussed in the video lectures. It provides a bit more context and detail around the following important topics:

What is the MIMII Dataset?

What was the motivation behind building this dataset?

How was the dataset collected?

Why did the creators also include sample benchmark models?

8.7 Real and Synthetic Data

As we have explored throughout this course, data is at the heart of machine learning. Oftentimes, data is the limiting factor and in turn dictates model performance. A scarcity in the amount of data points or number of features hinders the predictive power of our model. The solution seems obvious: collect more data, but this is not always feasible. So what can an ML engineer do?

Fortunately, with recent advances in deep generative models such as generative adversarial networks (GANs) and variational autoencoders (VAEs), very realistic synthetic data can be generated from a small number of known examples. This is achieved by approximately “learning” the data distribution and drawing random samples from the latent space of the generative model. This can be done on any type of data, allowing us to generate time series information (such as voice or sensor data), as well as photorealistic images. Using these new techniques, ML engineers can now bootstrap machine learning with machine learning -- that is, they can train models to generate more data with which to train other models!

8.7.1 What is Synthetic Data?

Imagine a power plant engineer that wants to use data to prevent a nuclear meltdown. They clearly cannot generate sample nuclear meltdowns to obtain relevant data, but they still need to be able to somehow make predictions. So what can they do? Well, if they have a very small amount of data from previous historical meltdowns or a simulation, then they can possibly use one of these techniques to train a model to generate enough data such that it becomes feasible to train an anomaly detection classifier using supervised learning. In this case, synthetic data is the only viable path forward.

The need to generate anomalous data is a commonplace situation in anomaly detection applications, since often the so-called “anomalies” are undesirable and can be extremely costly or damaging. Perhaps two of the most important applications of anomaly detection are for industrial processes and medical devices, where it can be used for predictive maintenance (fixing something before catastrophic failure or death).

Predictive maintenance will likely become an important feature of Industry 4.0, reducing costs by predicting component failures in advance and resolving the issue before it becomes more severe. An open-source dataset was created for this aiding with this specific application, referred to as MIMII (Malfunctioning Industrial Machine Investigation and Inspection). The MIMII dataset focuses on sounds exhibited by various components and can be used as a starting point to generate synthetic anomalous data for a variety of industrial components, reducing the need for individuals or organizations to collect these data.

Similarly, in the medical device space, anomaly detection allows for moderately malfunctioning devices to be detected on and replaced before they pose life-threatening issues to the device wearer. The ECG5000 dataset was created for this purpose, and provides 20 hours of ECG data - a type of data used to study heart health. Anomalous data is also provided in this dataset which corresponds to an individual with severe congestive heart failure. The dataset is open-source and can be freely used by anyone working on medical devices, and can be used as a starting point to train an anomaly detection model or to create more anomalous data.

This may sound incredible - we can effectively train models by simulating new data based on our current data. However, you have already done this in this course! When you use Data Augmentation, you are effectively expanding your dataset with synthetic data! That said, there are concerns when relying heavily on synthetic data that should be highlighted.

8.7.2 Quality Concerns of Synthetic Data

Real-life datasets are often complex and multifaceted. For example, when recording sounds, the actual sound plays an important role, but so does the microphone, background noise, microphone orientation, as well as myriad other factors. Creating synthetic data that is able to mimic this variability is infeasible, and currently there is no universally accepted method for generating high-quality synthetic data. One important reason for this infeasibility is that data must often go through multiple preprocessing stages before the generation of synthetic data. During this preprocessing stage, assumptions are implicitly made about the data which directly influence the synthetic data, meaning it is not purely based on the properties of the unprocessed data.

Similarly, if only a small amount of real-life data is collected, synthetic data generated using it may exhibit a high degree of bias. This can make it difficult to effectively extrapolate to other environments. Thus, it is still necessary to have access to as large and diverse a dataset as is possible given the contextual constraints to improve the generalizability of our synthetic data. The quality of the generated synthetic data must also be assessed, which can be problematic since the “quality” of a dataset can sometimes be very complex or specific, and not readily described by standard metrics.

8.8 Unsupervised Learning for Anomaly Detection (with K-Means)

Now that you’ve learned the basics of anomaly detection and the data-based challenges it imposes, we will explore anomaly detection through the lens of a traditional ML technique, K-means clustering. In this Colab you will visualize anomaly detection and unsupervised learning as well as learn and evaluate the strengths and weaknesses of K-means clustering. We will be using both the SKLearn KMeans and TSNE libraries.

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-8-9-K-means.ipynb

Additional reading:

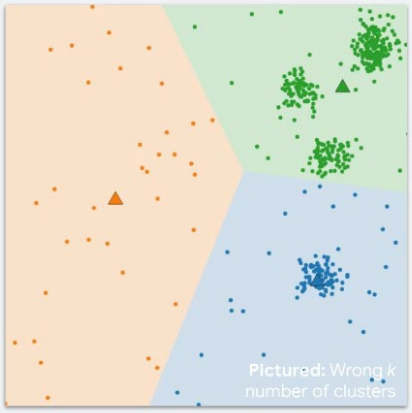
http://amid.fish/anomaly-detection-with-k-means-clustering

8.9 Unsupervised Learning for Anomaly Detection with Autoencoders

Recap of **K-Means**

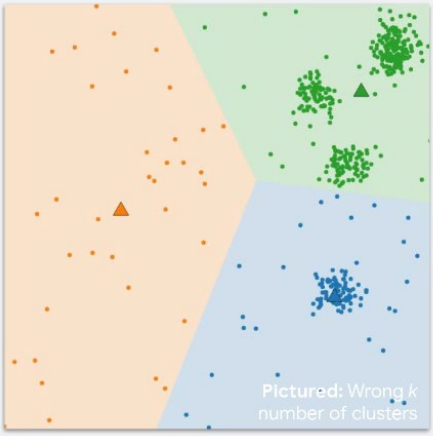
Goal:

Partition *n* observations into *k* clusters

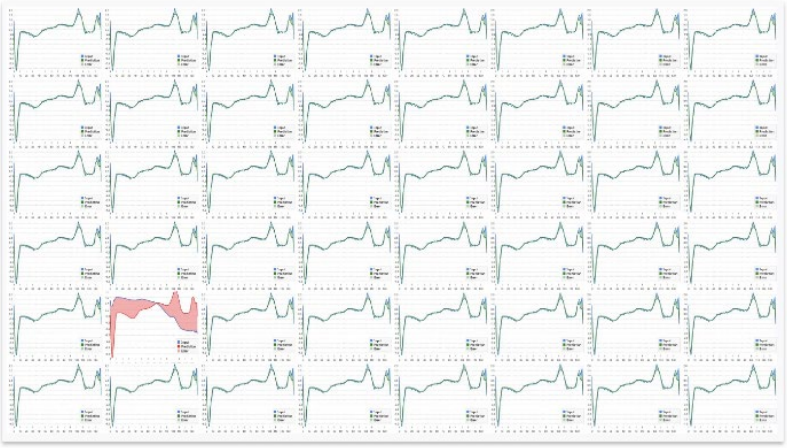


When **K-Means** Fails?

- Scales poorly with **large numbers** of observations
- **"Curse of Dimensionality"**

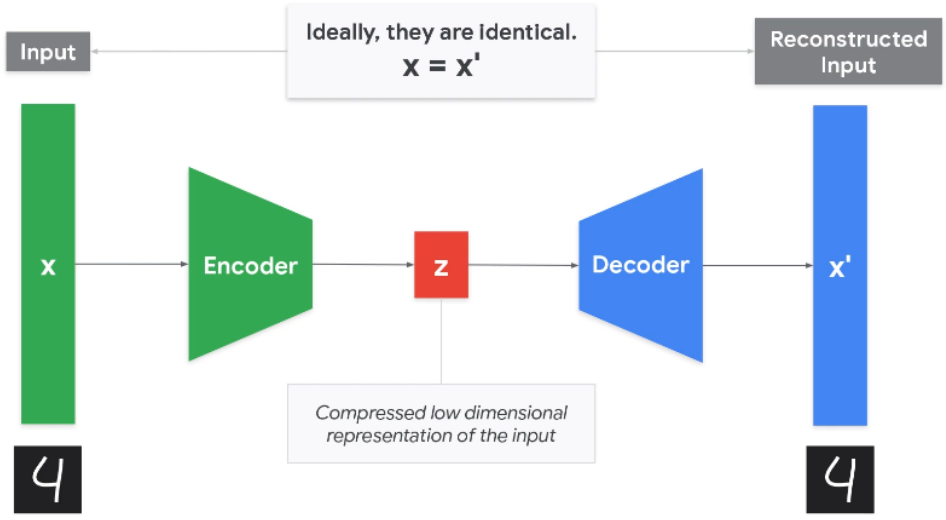


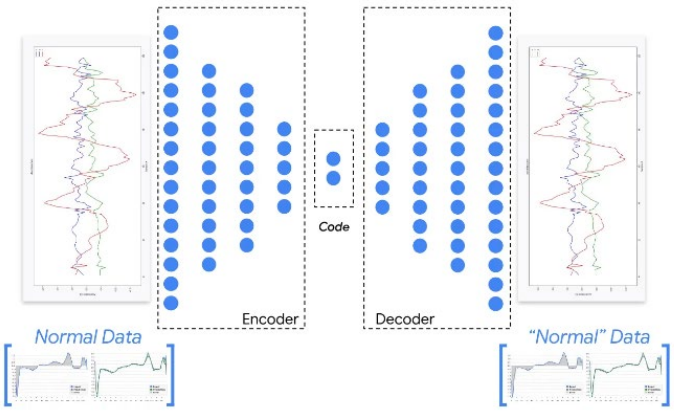
Lot do data for norma operations:

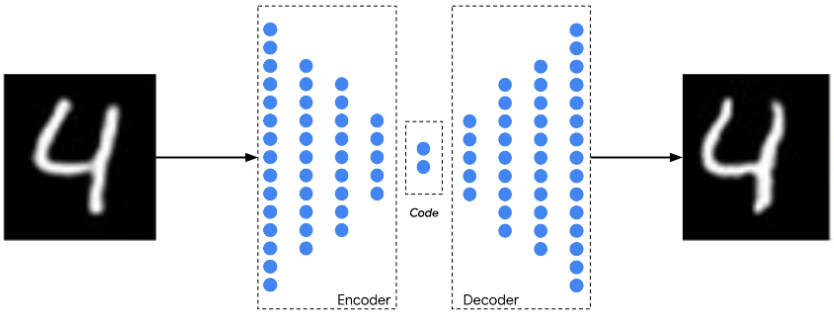


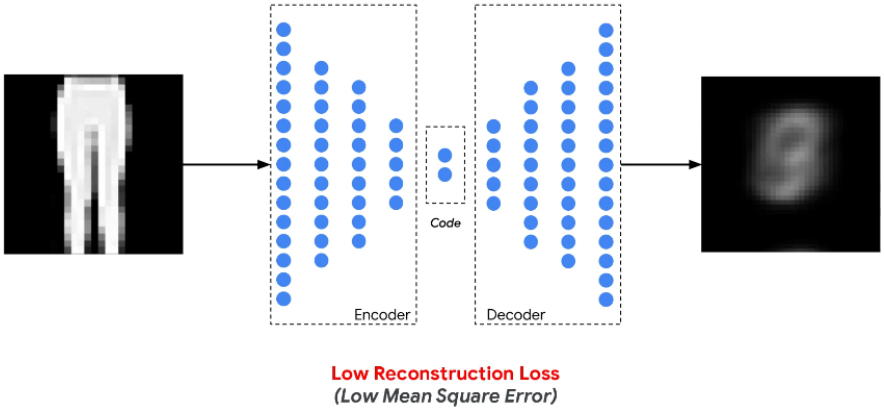
You don't know what you **don't know**.

You don't know what you **don't know**.
But **you do know** what **you know**.



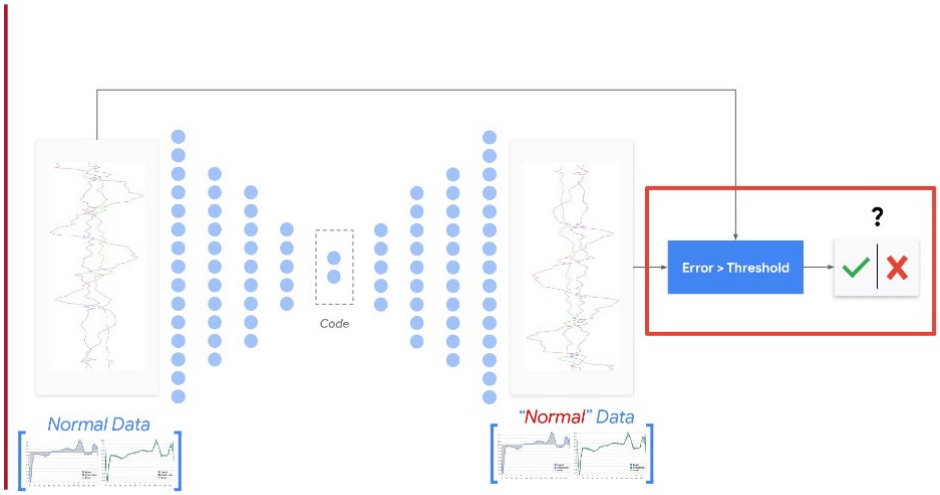




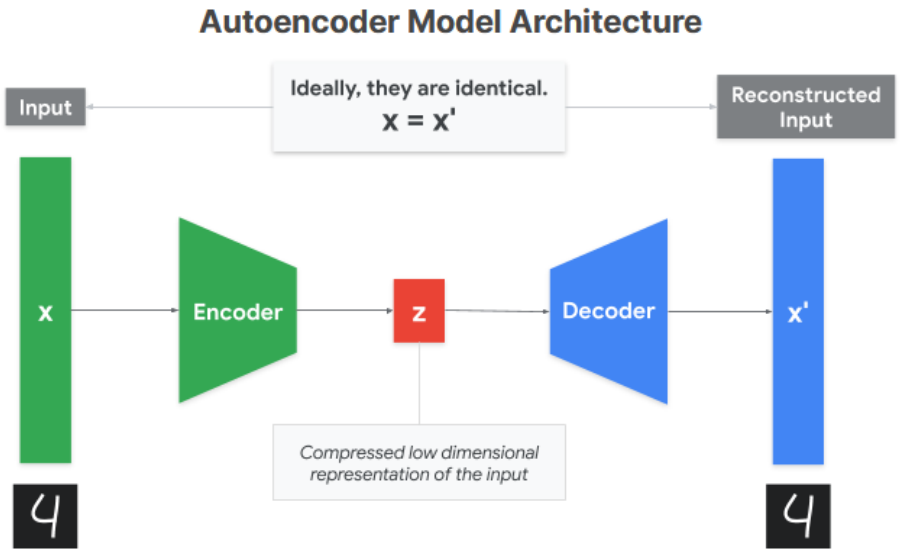


Big Error, FLAG THIS, it's not possible to reconstruct. Abnormal data -> Anomaly detection

In Coolab we will deal with unbalanced data and tune anomaly detector (define treshold):

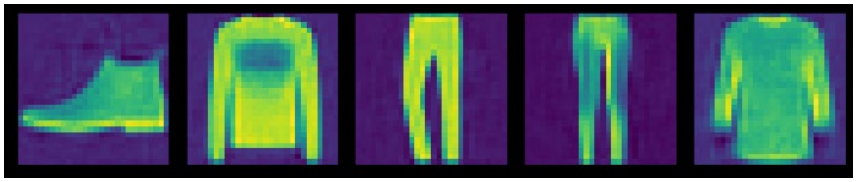


8.10 Autoencoder Model Architecture



You've previously been introduced to the basic concept of an autoencoder model. At its core, an autoencoder is trained to reconstruct data after compressing it into an embedding. You've learned how autoencoders can be applied to anomaly detection by training only on normal data and then using the reconstruction error to determine if a new input is similar to the normal training data. In this reading, we will examine autoencoders in more detail through a high level overview describing the strengths of different types of autoencoders. While autoencoders have many uses, including denoising, compression, and data generation, in this reading we will continue to focus on time-series anomaly detection. If you'd like to examine more applications of autoencoders check out this article.

Image_Reconstruction_from_Autoencoder:



Note :- Autoencoders are more powerful than PCA in dimensionality reduction.

8.10.1 Types of Autoencoders

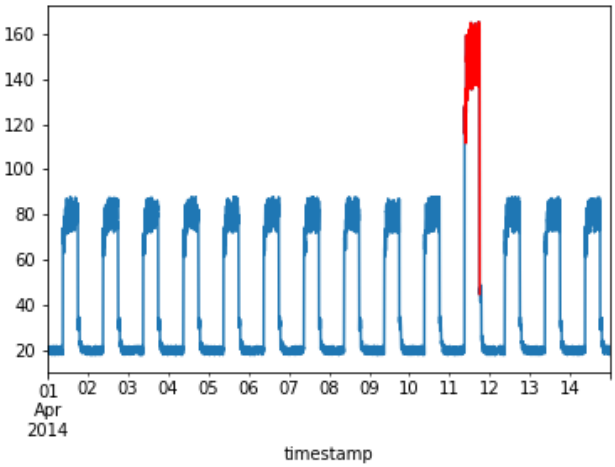
It turns out that there are many kinds of autoencoders that are used in the wild. In this rest of this article we will survey some of the most popular types. To see how to implement all of these different autoencoders in keras check out this article.

8.10.2 Fully Connected Autoencoder

A deep fully-connected autoencoder has multiple fully connected layers with some small embedding in the middle. You've already seen an example of these in the previous video. They are the most basic form of an autoencoder and are simple to implement and deploy.

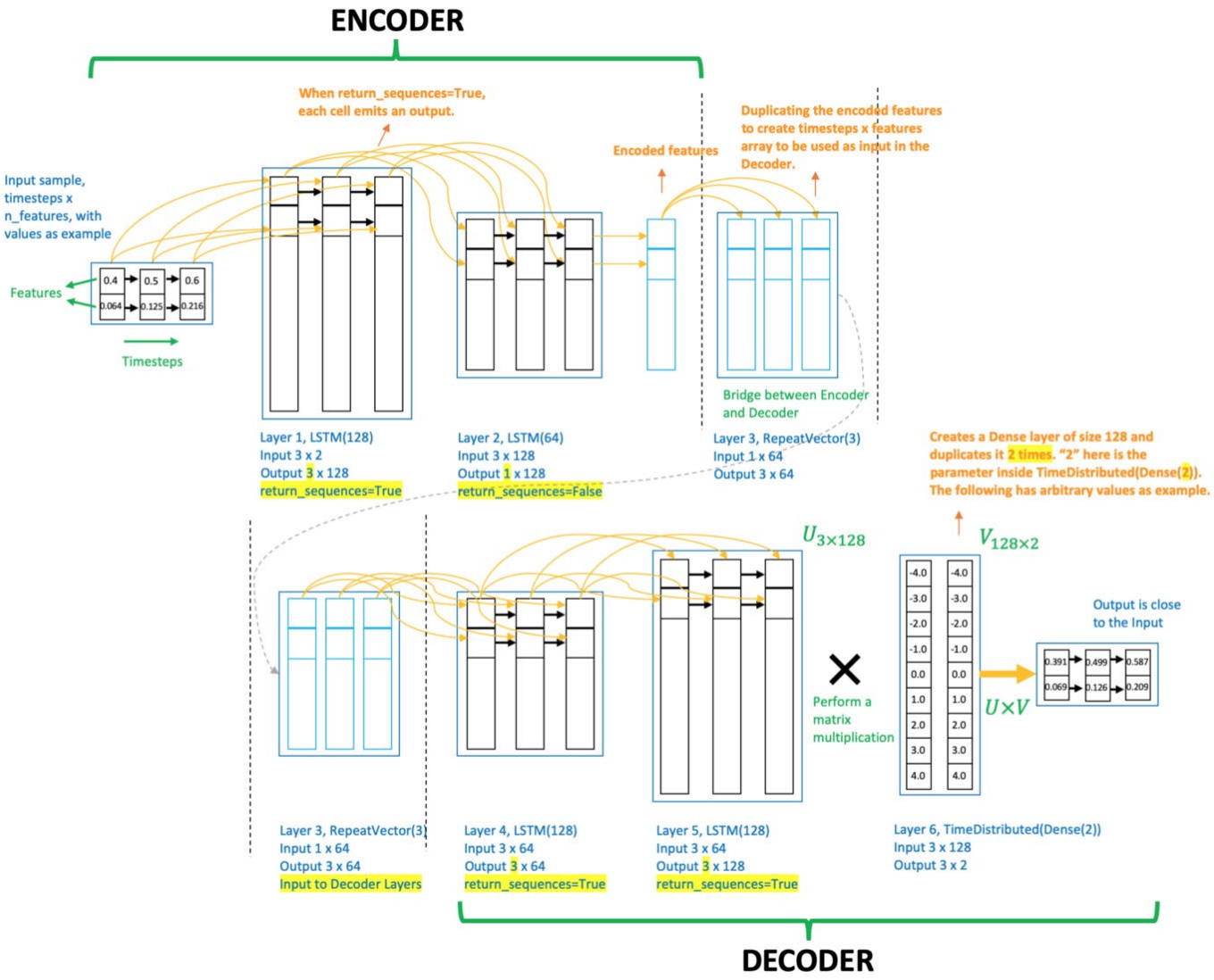
8.10.3 Convolutional Autoencoder

A convolutional autoencoder operates under the same principle as the deep fully-connected autoencoder but uses convolutions for the encoder layers and transpose convolutions for the decoder layers. Due to the spatial nature of convolutions, these autoencoders are particularly useful for images and spectrograms (which are essentially images). Unfortunately, at this time, transpose convolutions are less commonly supported by tinyML frameworks, so deploying convolutional autoencoders to microcontrollers is more challenging. That said, hopefully they will become more accessible soon. In the meantime, if you would like to try using a convolutional autoencoder for anomaly detection in Colab, check out this keras example.



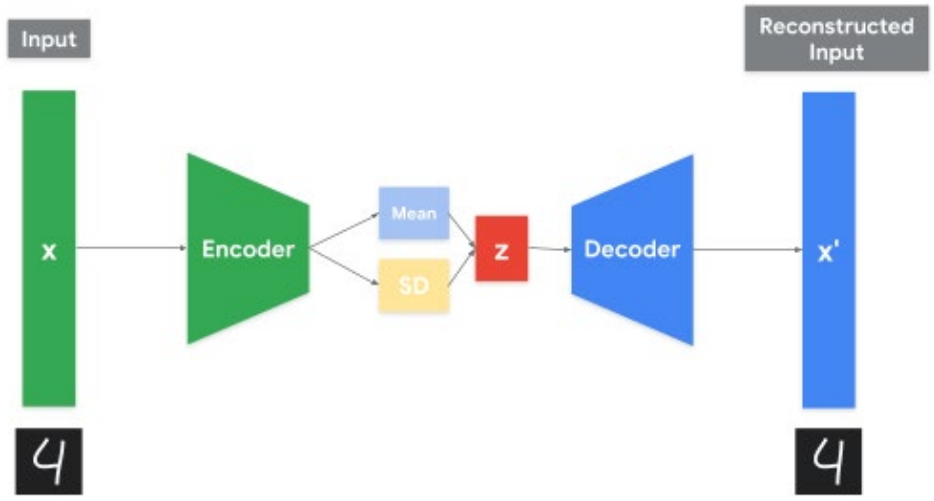
8.10.4 LSTM Autoencoders

Long short term memory networks are a type of recurrent neural network. They are useful for processing longer sequences while capturing the temporal structure of the data like the time-series data we have explored thus far. This can lead to a more detailed analysis of longer sequences which leads to higher accuracy in some cases. However, these RNN layers generally use more working memory as they maintain and modify a state over long sequences. This means they often consume more SRAM, which is a precious resource in tinyML. Additionally, it is less common for tinyML frameworks to support LSTMs or RNNs in general. For more detail on LSTM autoencoders check out this article.



8.10.5 Variational Autoencoders

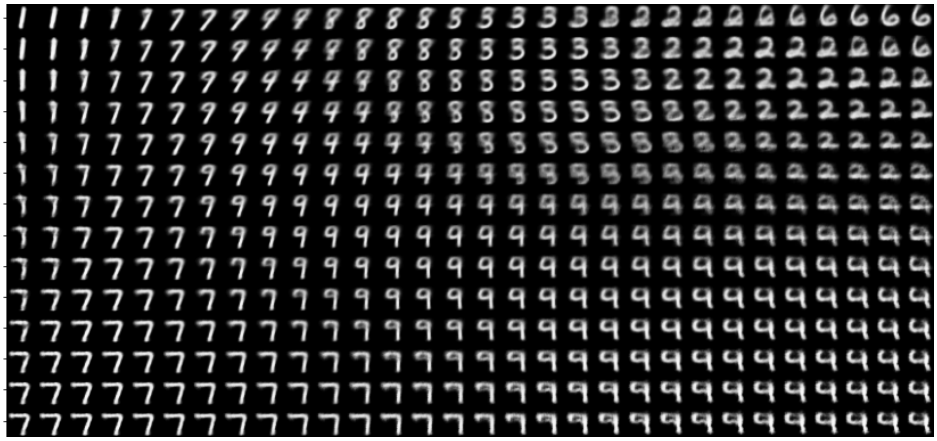
Unlike the previous examples, the unique aspect of variational autoencoders is not that they use a different type of layer. Instead, variational autoencoders impose an additional constraint during training, which forces the model to map the input onto a distribution, which means the normal latent vector of an autoencoder is replaced by a mean vector and a standard deviation vector from which a latent vector is sampled. For anomaly detection, we can use the reconstruction probability that the model produces instead of the reconstruction error used in normal autoencoders, which can at times lead to better predictions.



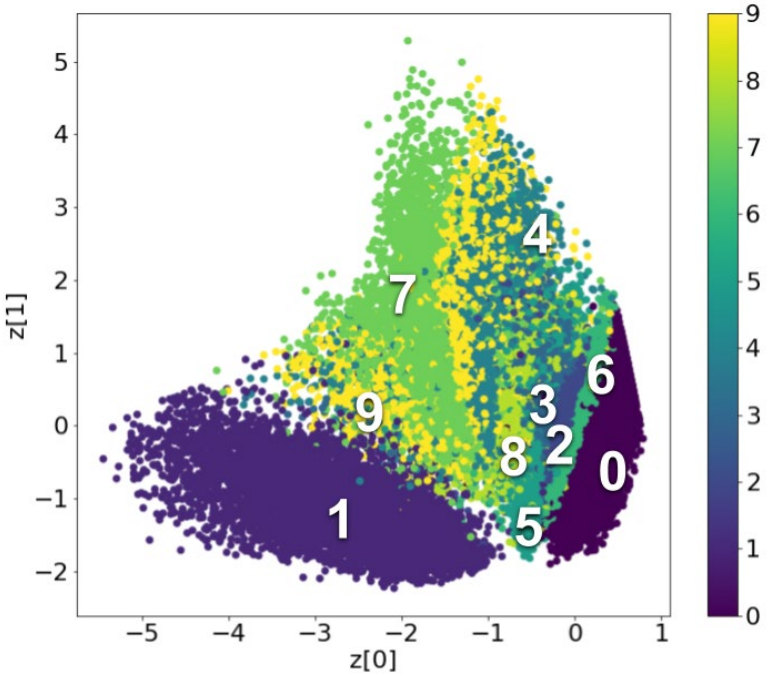
8.10.6 Visualizing the Latent Space of Variational Autoencoders

There are multiple ways of visualizing the learned low-dimensional representation, or latent space, in an autoencoder. Due to the additional constraints imposed on the latent representation in variational autoencoders, visualizing the latent space is more straightforward than with traditional autoencoders.

When variational autoencoders are trained on images, a common technique to visualize the latent space is to randomly draw values from the learned distributions and use these values as inputs to the decoder (ignoring the encoder entirely). This produces a novel image sampled from the latent space. For example, after training on MNIST digits, the resultant images will usually resemble specific digits. Note that these generated images are not present in the training set - they are representations of what the network has learned about encoding digits. In fact, by running linearly-spaced sample values through the decoder, you will notice that the output images appear to interpolate between digits. We are thus visualizing samples along a continuous latent space (i.e., a manifold):



If your training images belong to specific classes, as MNIST digits do, you may also observe potential clustering within the latent space. You might expect each digit to be spatially separated from other digits, and indeed this is the case. The variational autoencoder in this article (which we used for inspiration for this article) encodes MNIST digits in a two-dimensional latent space. In the below figure, each test image from the MNIST dataset is fed through the encoder (the decoder is ignored). The two-dimensional mean vector of the encoder's output is plotted on the x-y plane, and each digit (indicated by the legend on the right using different colors) is fairly well-clustered:



For variational autoencoder networks with larger latent spaces (more than 2 dimensions), common dimensionality reduction algorithms like PCA, t-SNE, or UMAP may help with visualization, though these are outside the scope of our discussion.

Unfortunately, variational autoencoders are more complicated to train and deploy, therefore in the remainder of this course we will focus on standard autoencoders. However, since you may run into these new and powerful techniques in the future we wanted to make sure to introduce you to them now. Next, you will train a fully connected autoencoder to identify abnormal heart rhythms from ECG data and learn how to evaluate an anomaly detection model.

8.11 Training and Metrics

Now that you have been introduced to the basics of autoencoder model architectures, you will explore the process of training one to detect anomalous ECG data. This will get you familiar with the unique process of training only on normal data and allow you to play with the architecture of a fully connected autoencoder. Additionally, you will learn the two important metrics, the receiver operating characteristic (ROC) curve and the area under the ROC curve (AUC).

In this Colab you will be using TensorFlow to define and train the autoencoder and SKLearn to evaluate its performance.

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-8-13-Autoencoders.ipynb

A Gentle Introduction to Anomaly Detection with Autoencoders

How Airbus Detects Anomalies in ISS Telemetry Data Using TFX

8.12 Picking a Threshold

For the application of anomaly detection that is most interesting to you, what kinds of thresholds would you select? What factors influenced the sensitivity you select? How might you test that sensitivity? Do you expect it might be too conservative? Too aggressive? Why?

8.13 Assignment: Training an Anomaly Detection Model

Now that you know how to train an autoencoder for anomaly detection using only normal data, it's time to test your skills. You will be repeating the ECG anomaly detection with a two key differences:

The training set now includes a low percentage of anomalous data. This reflects a real-world scenario where you don't have access to experts to label input data and therefore can not ensure that the training data contains only normal data. Since the anomalous portion of the training data is small, the model can still perform quite well as long as we don't overfit.

You will be tasked with determining the size of the autoencoder's embedding layer and picking the error threshold to obtain the best performance given the application.

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-8-16-Assignment.ipynb

8.13.1 Assignment Solution

The size of the encoding layer should be less than 8 and more than 0 as we need to have a bottleneck from the previous layer (which was size 8). We have found: EMBEDDING_SIZE = 2 seems to work the best! The encoding layer size is a critical factor in how much information our model can encode. Remember, we want the encoding layer (and the model in general) to be large enough that it can encode the normal signals that represent 90% of our training data but we want it to be small enough so that it can't learn to reproduce the 10% of our training data that is anomalous. For this dataset, the model and encoding layer can be very small, but it's important to remember that this is a very controlled and simple example.

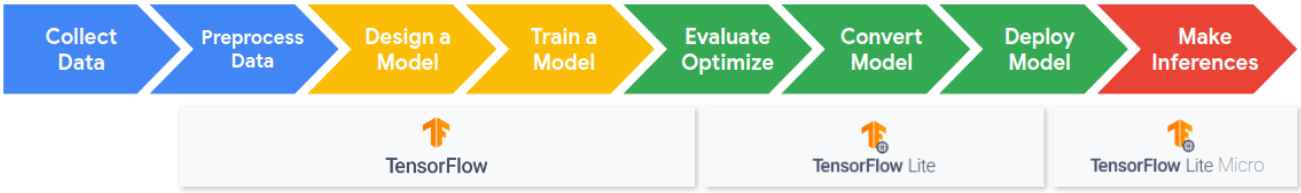
The threshold depends on the training and on the encoding layer size but somewhere in the range of threshold = 0.037 works best. When picking the threshold the goal is to maximize the accuracy, precision, and recall. The 'knee' of the ROC curve is a good place to start as it represents a good balance between precision and recall. For this application, it might be more important to properly classify an abnormal rhythm as such, instead of maximizing the accuracy.

As always if you would like to go back and try the assignment again, it can be found here:

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-8-16-Assignment.ipynb

8.14 Summary Reading

In this section you explored the TinyML flow and data engineering in the context of anomaly detection applications, focusing on some unique challenges presented by unsupervised learning. Anomaly detection represents a common TinyML use case of classifying some time-series data as 'normal' or 'abnormal'. Anomaly detection can be applied to data coming from a wide variety of sensors in a number of different application domains.



8.14.1 Anomaly Detection in Industry

You explored an example of anomaly detection in an industrial setting where you learned about the primary constraints of tinyML anomaly detection. First, anomaly detection systems must be fast enough to keep up with the rate at which sensors produce data. Often the sensors produce too much data to be sent over a wireless network therefore computation must be done locally. To accomplish this, the anomaly detection model must fit within the tight memory constraints of a microcontroller. Finally, an anomaly detection system must mitigate false negative and false positives, which, depending on the application, can have dire consequences.

8.14.2 Data and Datasets

Anomaly detection is unique in that it is impractical to collect real examples of anomalous data. Collecting data on a failure is often too expensive since it usually involves intentionally breaking something. Additionally, anomalies are inherently rare and can take many different forms therefore it is difficult to train anomaly detection models in a normal supervised manner. That all said, you explored the MIMII and ToyADMOS datasets as well as the concept of generating synthetic data to understand some of the processes used to try to collect an anomaly detection dataset.

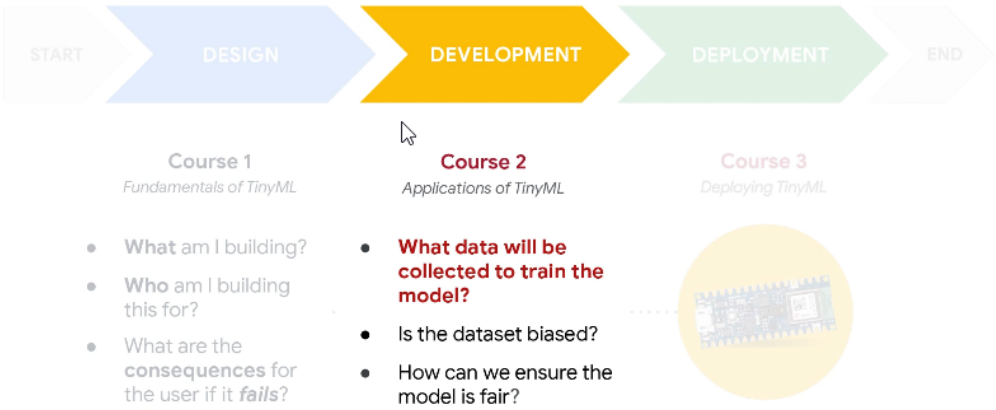
8.14.3 Unsupervised Learning: K-Means and Autoencoders

To avoid some of the issues with supervised learning you then explored a classical and neural network based approach to unsupervised learning. For the classical approach, you explored anomaly detection through the lens of K-means clustering and learned about its advantages (ease of implementation and use) and disadvantages (scalability). You then learned about autoencoders and how they can be applied to many domains, among which is anomaly detection. Autoencoders are trained to reconstruct normal data after compressing it into an embedding. By comparing the error between the input and the output of the autoencoder we are able to determine how similar the input is to our normal training data. We perform anomaly detection by selecting a threshold above which we classify the input as an anomaly. You also learned about how the model architecture impacts the autoencoder and how metics like ROC and AUC help us evaluate our performance. Critically, you learned that anomaly detection models often have low transferability and therefore have to be trained for their specific application.

9 Responsible AI Development

9.1 Data Collection

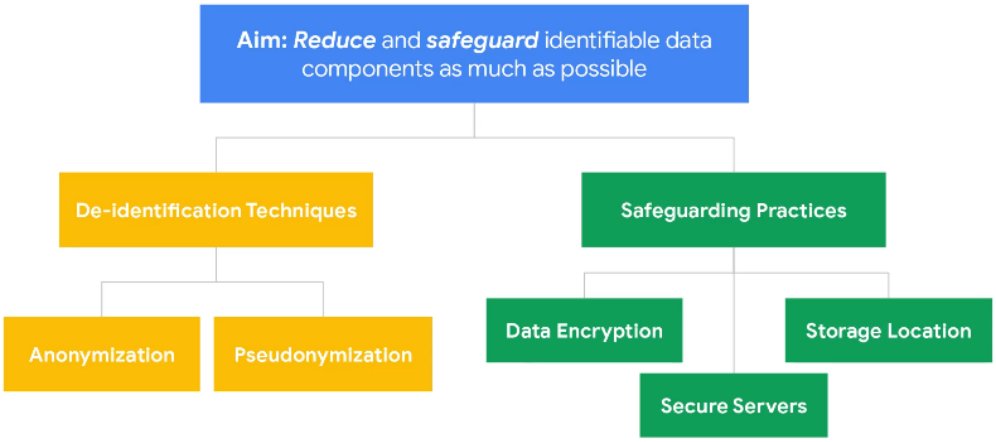
Responsible AI: Human-Centered Design



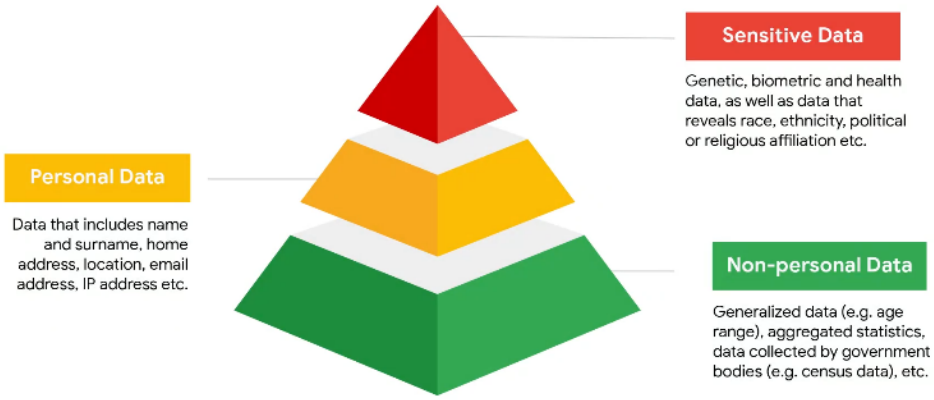
Data Laws and Regulations

3 Key Features of Data Protection

1. Identifiability



Levels of Data Classification



2. Data minimization

Aim: *Limit* data collection and duration of storage to **only** what is **required** to fulfill a specific purpose

- **How long** will I need the data to achieve the purpose?
- Is there **unnecessary data** that can be deleted?
- How often should I periodically **review** data **and delete** what isn't needed?

Right to be Forgotten (**Erasure**)

Subjects have the **right* to request that their data be erased** by the data controller, as soon as possible.

**This right may be overridden in some cases.*

What should data collectors do?

- Provide subjects with **clear information** and **practical ways** to make a request for data erasure

3. Notice and Consent

Aim: Prepare ***clear notice*** and ***consent*** communication to data subjects

- **Notice** ensures that subjects are ***aware*** of the intended data practices
- **Consent** ensures that subjects are only ***implicated*** in those practices ***if they want to be***.

Necessary Conditions of **Informed Consent**

Informed	Subject has sufficient knowledge and comprehension of the matter to enable an enlightened decision	No lies, deceit, or partial disclosure
Voluntary	Subject freely chooses to give consent	No coercion, inappropriate pressure or influence
Competent	Subject has the decisional capacity to offer consent	No children, adults deemed mentally incompetent

Mechanisms for Data Collection

1. Crowdsourcing
2. Product users
3. Paid contributors (mechanical turk)

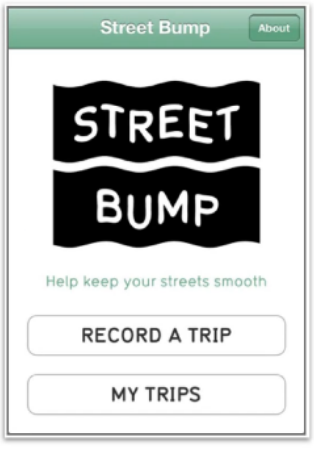
> "...we need to ask which people are excluded. Which places are less visible? What happens if you live in the shadow of big data sets?"

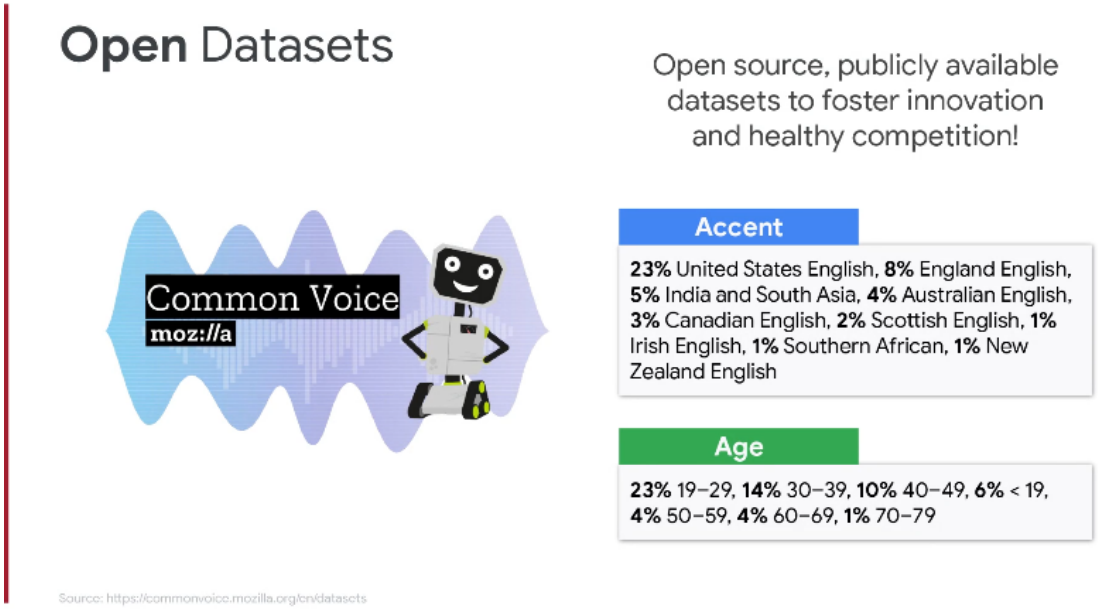
Kate Crawford
Principal Researcher at Microsoft and Professor at NYU Tandon School of Engineering

Source: "The Hidden Biases in Big Data," Harvard Business Review, April 2013.

Data Collection: **Product Users**

Does the demographic of product users accurately represent the population?





9.2 The Many Faces of Bias in ML

One of the promises of machine learning is that it can potentially offer a superior alternative to human decision-making. Algorithms offer objective interpretations of data and can be trained and developed to produce consistent results, unlike human decision-making which can oftentimes be subjective and inconsistent. Thus, an advantage of machine learning is that it isn't susceptible to 'human error' or mistakes. Even more impressive is the potential for machine learning to predict outcomes with a level of accuracy that surpasses that of the world's leading experts.

While these are exciting possibilities, one of the major challenges of machine learning is the problem of bias. The presence of bias can be the downfall of an otherwise perfect model, resulting in skewed outcomes and low accuracy. This can be a frustrating experience for those who dedicate their time and energy into creating a model, only to discover that it doesn't work very well. Even more concerning, however, is the possibility that if bias finds a foothold in machine learning, the model may effectively function to automate bias and inequality and deploy it at scale. When all goes well, machine learning can be consistently accurate. But the presence of bias can cause machine learning to be consistently inaccurate in such a way that leads to discriminatory or unfair outcomes.

Recent discussions of bias have primarily focused on the use of machine learning in high-stakes contexts where inaccurate outputs can have major consequences. For instance, when algorithms are used to inform decisions regarding bail in the criminal justice system, the granting of loans in financial institutions, or the hiring of job applicants, the presence of bias can result in some individuals or groups being systematically disadvantaged in a way that seriously impacts their life prospects. As one might expect, biased algorithms in high stakes contexts

have made national news headlines, as demonstrated by Amazon's hiring algorithm that revealed gender bias and the COMPAS recidivism tool that revealed racial bias.

While the negative impact of bias is most apparent in high stakes contexts, bias remains a serious challenge for any application of machine learning—including tinyML. This may be somewhat surprising when one considers the kinds of tasks with respect to inference that tinyML is meant to tackle with the use of microcontrollers. For instance, if bias leads a keyword spotting or activity detection application to be inaccurate, how problematic could this be? Even though tinyML will not be responsible for complex inference tasks in high-stakes contexts, the unique features of tinyML raise a different kind of concern.

TinyML is at the cutting-edge in terms of low latency, low power, and small size. This is significant because it enables tinyML to be pervasive and find applications across a wide range of contexts. The number of IoT devices has already grown considerably in recent years, and with the emergence of tinyML this number is expected to surge upwards. It's precisely because tinyML is uniquely capable of being everywhere, surrounding us, that the problem of bias remains a major concern. If one were to consider the impact of a single device that is inaccurate due to bias, it may not seem especially problematic. But when one considers the likelihood that tinyML will soon become a feature of a wide variety of our daily interactions with machine learning, it's easy to see how the presence of bias could amount to a more serious concern. For example, if an individual fails to trigger the wake word of a personal voice assistant, this may seem like a minor inconvenience. But if this individual also fails to trigger the wake word for their light switch, thermostat, coffee maker and so forth, then this individual would seem to be systematically disadvantaged insofar as they cannot enjoy the benefits of machine learning applications.

In order for tinyML to make our lives easier and more accessible than ever before, designers will need to remain vigilant in rooting out bias from their applications. So how does bias find its foothold in machine learning algorithms? One reason this can happen is because humans are still responsible for creating machine learning applications, so whatever biases are present in the data scientists may be carried over into the project. The most common reason machine learning becomes biased, however, is owing to the underlying data.

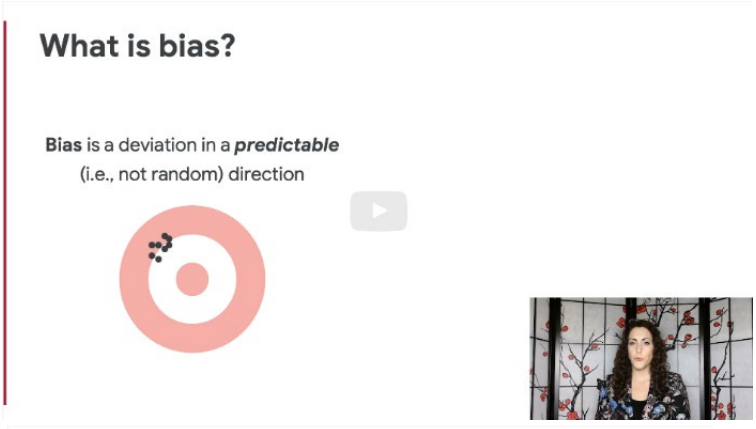
For instance, a dataset may be biased if it is incomplete. If the dataset doesn't fully represent the diversity of end users, then this can lead the model to be less accurate for certain individuals or groups. Researchers have shown that voice recognition tools struggle to identify African American Vernacular English, leading widely popular personal voice assistants to work less well for black individuals. Similarly, research shows that voice recognition struggles to identify non-native English speakers or those with speech impairments. After recognizing the importance of making products accessible to everyone, Google launched "Project Euphonia" in 2019 with the goal of collecting more data from individuals with speech impairments or heavy accents to fill the gaps in current voice datasets.

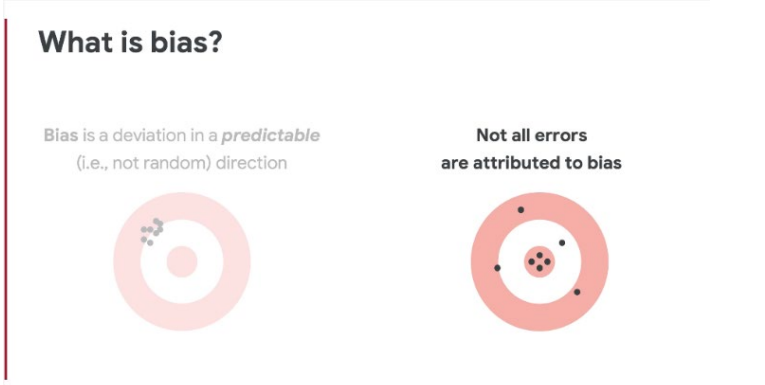
However, tackling the problem of bias isn't always as simple as collecting more data from underrepresented users. Rather, bias has many faces and can present at any stage of the data lifecycle, including creation, collection, sampling and processing. A general piece of advice is to pay close attention to the data that is being used to develop an algorithm. It is worth reflecting on questions like the following: From whom is the data

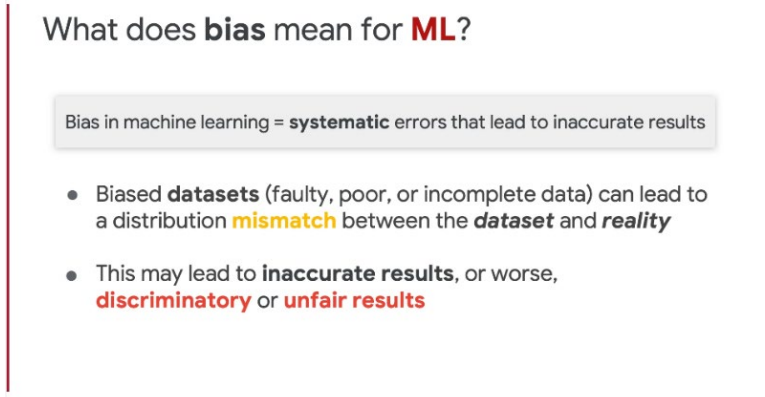
being collected and is it representative of the end-user population? What device is being used to collect the data? What decisions have been made about how to process and label the data?

Carefully analyzing the data is a huge step towards mitigating bias, as identifying the exact form of bias is necessary for making a determination about how to address it. For instance, if the dataset is incomplete, then the solution may be collecting additional data that better represents the end-user population. Alternatively, if the data is faulty due to labeling errors then the solution may be conducting a review of the data to correct these errors. In the next video, you will learn to identify some common forms of bias. In addition, you will learn about some potential solutions that are being developed on the industry level, as well as what you can do as an individual designer to tackle the problem of bias in machine learning.

9.3 Biased Datasets







The **"garbage in, garbage out"** problem

Bias: Defining the **Target Variable**

- **How** should you define a ***"good"*** employee for a hiring algorithm?
- **Subjective** process: ***"good"*** must be defined in ways that correspond to **measurable** outcomes

Number of sick days	**Number of times late to work**
Length of employment	**Number of sales per quarter**

Bias: Defining the **Target Variable**

- **How** should you define a ***"good"*** employee for a hiring algorithm?
- **Subjective** process: ***"good"*** must be defined in ways that correspond to **measurable** outcomes

?

Number of sick days	**Number of times late to work**
Length of employment	**Number of sales per quarter**

Bias: Defining the **Target Variable**

Using **biometric** sensors for a health wearable device, how should you define ***"healthy"***?

- **Heart rate**
- **Blood pressure**
- **Number of steps**

Bias: **Labeling** the Data

Labels applied to the training data must serve as **ground truth**

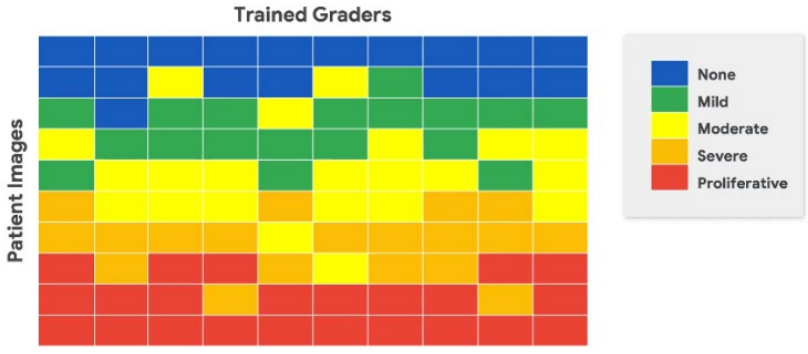
Horse

Human

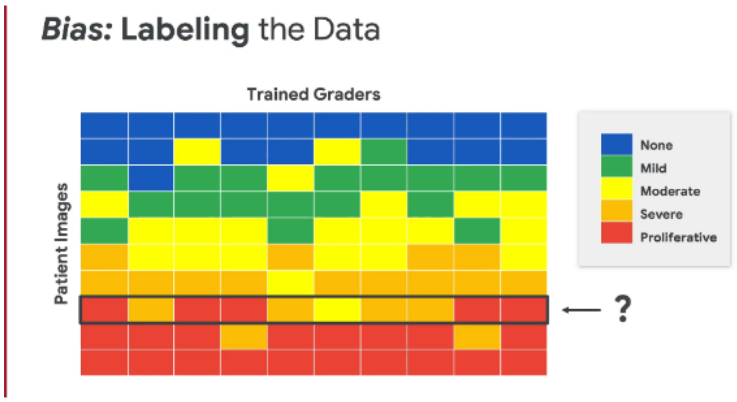
Human

ERROR

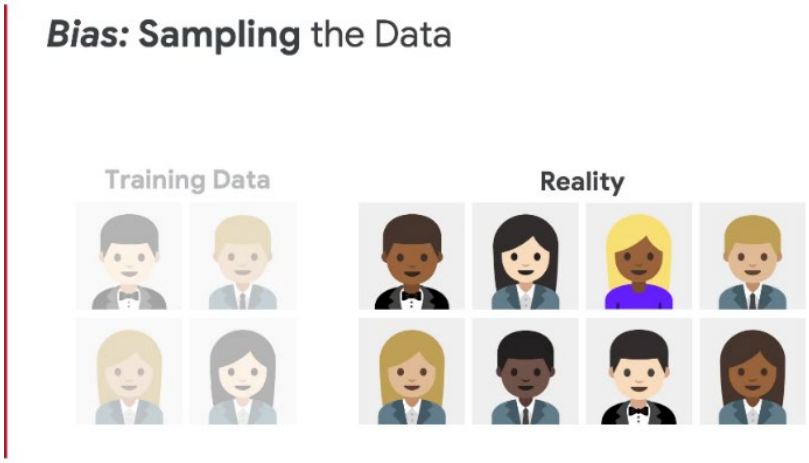
Bias: **Labeling** the Data

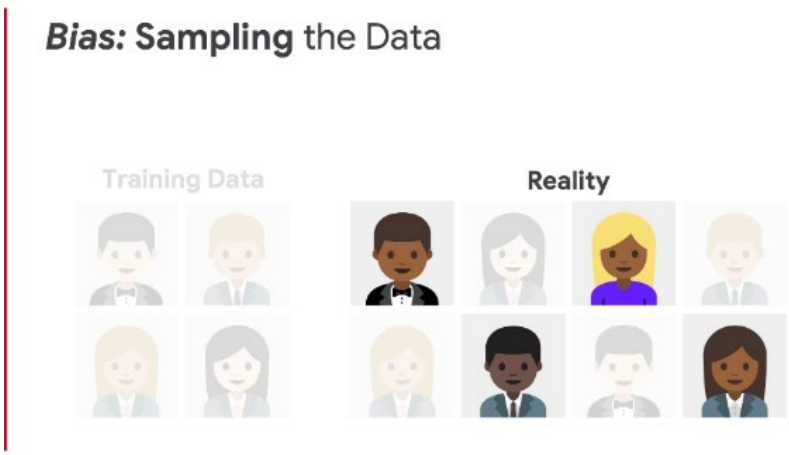


Disagrement:

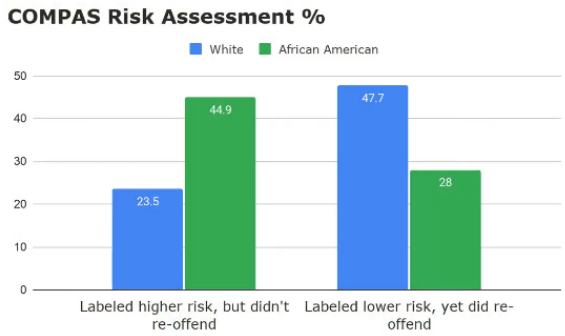


Treaining data does not reflect reality:





Bias: **Prejudice** Reflected in Data



Northpointe's COMPAS Recidivism Prediction Tool

Bias: **Prejudice** Reflected in Data

Dataset: *65%* of people cooking are *women*
Algorithm predicts: *85%* of people cooking are *women*

Bias: Measurement **Distortion**

Hardware matters!

Example: Optic sensors and cameras have problems ***detecting darker skin tones*** due to the way light is reflected

- **Automatic soap dispensers**
- **Activity detection**
- **Facial recognition**

Bias: Measurement **Distortion**

Industry Solutions: **Datasheets for Datasets**

Questions for dataset creators to reflect on during the key stages of the dataset lifecycle:

- ***Motivation***
- ***Composition***
- ***Collection Process***
- ***Preprocessing/ labeling***
- ***Uses***
- ***Distribution***
- ***Maintenance***

paper authored by

JAMIE MORGENSTERN, Georgia Institute of Technology
TIMNIT GEBRU, Google
BRIANA VECCHIONE, Cornell University
JENNIFER WORTMAN VAUGHAN, Microsoft Research
HANNA WALLACH, Microsoft Research
HAL DAUMÉ III, Microsoft Research; University of Maryland
KATE CRAWFORD, Microsoft Research; AI Now Institute

Industry Solutions: **Data Nutrition Labels**

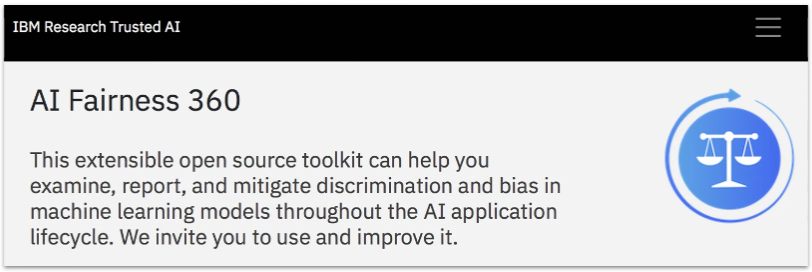
Metadata	
Filename	201612v1-docdollars-product_payments
Format	csv
Url	https://projects.propublica.org/docdollars/
Domain	healthcare
Keywords	Physicians, drugs, medicine, pharmaceutical, transactions
Type	tabular
Rows	500
Columns	18
Missing	5.2%
License	cc
Released	JAN 2017
Range	
From	AUG 2013
To	DEC 2015
Description	This is the data used in ProPublica's Dollars for Docs news application. It is primarily based on CMS's Open Payments data, but we have added a few features. ProPublica has standardized drug, device and manufacturer names, and made a flattened table (product_payments) that allows for easier aggregating payments associated with each drug/device. In [1], one payment record can be attributed to up to five different drugs or medical devices. This table flattens the payments out so that each drug/device related to each payment gets its own line.

DATA NUTRITION PROJECT

A standard label that highlights the **"key ingredients"** of a dataset:

- ***Provenance***
- ***Metadata***
- ***Missing units***
- ***Variables***

Industry Solutions: **Bias Testing Toolkits**



Designer Solutions

- Carefully **research your users in advance**, be aware of potential outliers
- **Ensure your team** of data scientists and data labelers is ***diverse***
- Where possible, **combine inputs from multiple sources** to ensure data diversity
- **Create a gold standard** for data labeling
- Seek out **domain experts** to review your data

9.4 Forum

Now that you've become acquainted with different forms of bias and potential solutions, reflect on the following questions and discuss your response with other students taking this course!

Which form of bias did you find most surprising or interesting? Which form of bias do you think poses the most serious challenge to TinyML? Why?

As a reminder, these are some common forms of bias discussed in the video:

Defining the target variable

Labeling the data

Sampling the data

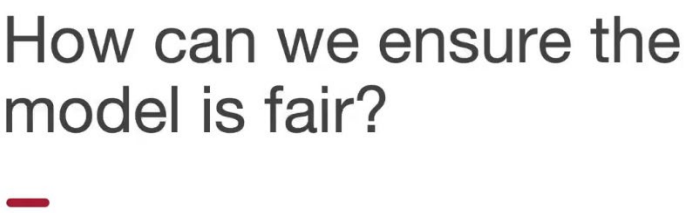
Prejudice reflected in the data

Measurement distortion

Which solutions for bias (either industry or individual designer) do you think are most promising? Are there any obstacles you can foresee for the solutions you think are most promising?

As a reminder, industry solutions included datasheets for datasets, data nutrition labels, and bias testing toolkits. Individual designer solutions included recommendations such as carefully researching users in advance, ensuring your team of data scientists and data labelers are diverse, combining inputs from multiple sources to ensure data diversity, creating a gold standard for data labeling, and seeking out domain experts to review your data.

9.5 Fairness



Unfairness in ML

Model exhibits **discriminatory biases**, perpetuates **inequality** or performs less well for historically **disadvantaged groups**

- ***All ML discriminates*** (it just means to recognize a distinction, differentiate)
- Fairness is concerned with **wrongful** discrimination

Discrimination

Disparate **Treatment**:

Membership in a protected class is used as an input to the model, decisions are differentiated on that basis in a way that disadvantages members of a protected class

Disparate **Impact**:

Outcomes of the model disproportionately disadvantage members of a protected class

1. Group Unawareness

Sensitive attributes are **not** included as features of the data (e.g. race, gender)

Pro: Avoids disparate treatment

Con: Possibility of highly correlated features that are proxies of the sensitive attribute

2. Group Threshold

Counteract historical biases in data by **adjusting** confidence thresholds ***independently*** for each group

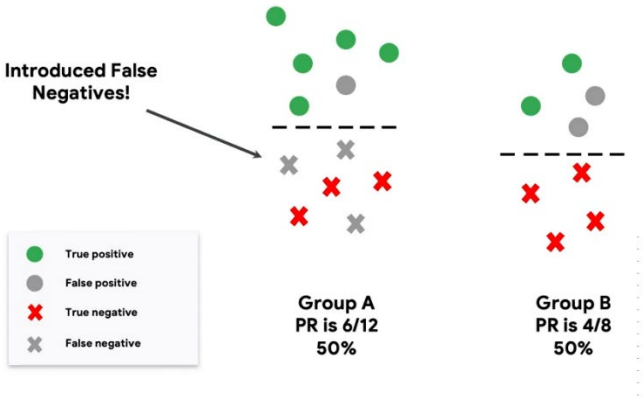
Group A **Group B**

3. Demographic Parity

	Actually Healthy = Yes	Actually Healthy = No
Predicted Healthy = Yes	***True Positive***	***False Positive***
Predicted Healthy = No	False Negative	True Negative

The positive rate is the same across groups

Problem with Demographic Parity

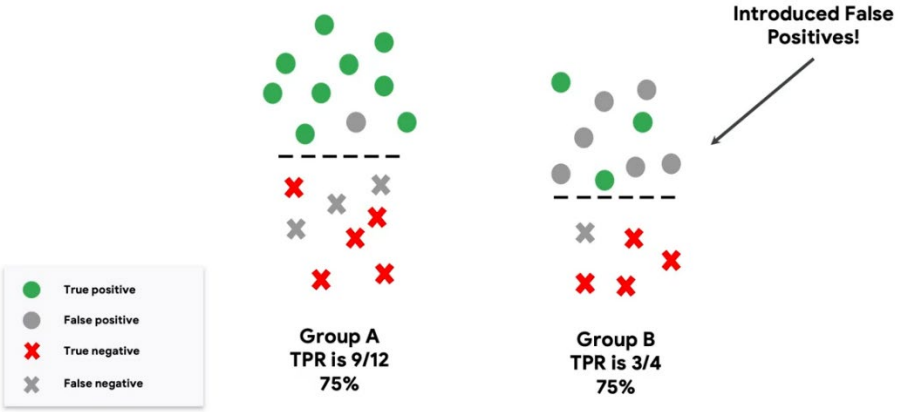


4. Equal Opportunity

	Actually Healthy = Yes	Actually Healthy = No
Predicted Healthy = Yes	***True Positive***	False Positive
Predicted Healthy = No	***False Negative***	True Negative

Qualified individuals should have an equal chance of being correctly classified for a desirable outcome.

Problem with Equality of Opportunity

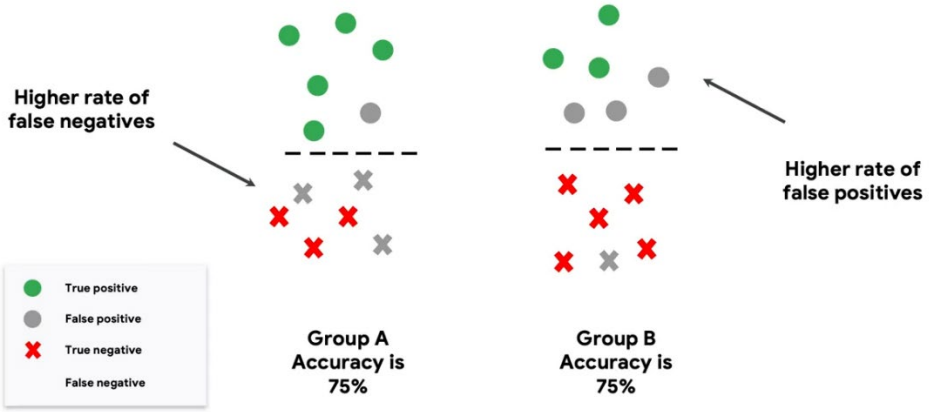


4. Equal Accuracy

	Actually Disease = Yes	Actually Disease = No
Predicted Disease = Yes	***True Positive***	False Positive
Predicted Disease = No	False Negative	***True Negative***

The percentage of correct classifications should be the same for all individuals

Problem with Equal Accuracy



Impossibility Theorem

We cannot satisfy all fairness metrics

at the same time!

For example:

- **Group Unawareness** is incompatible with **Group Threshold**
- **Equal Opportunity** is incompatible with **Equal Accuracy**

How can we mitigate ***unfairness*** in ML?

The Framing Trap

Algorithmic Frame

Do properties of the output match the input? Does the algorithm provide good accuracy on unseen data?

Data Frame

Has bias been removed from the training data? Does the demographic information of the data require optimization of the model?

Sociotechnical Frame

How does the model operate when considered as part of a system of humans and social institutions?

The Framing Trap

Algorithmic Frame

Do properties of the output match the input? Does the algorithm provide good accuracy on unseen data?

Data Frame

Has bias been removed from the training data? Does the demographic information of the data require optimization of the model?

Sociotechnical Frame

How does the model operate when considered as part of a system of humans and social institutions?

The Framing Trap

Algorithmic Frame

Do properties of the output match the input? Does the algorithm provide good accuracy on unseen data?

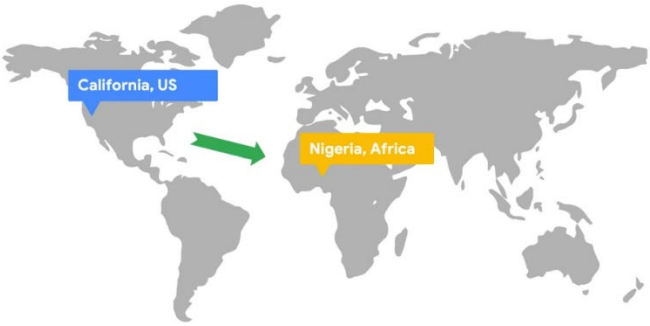
Data Frame

Has bias been removed from the training data? Does the demographic information of the data require optimization of the model?

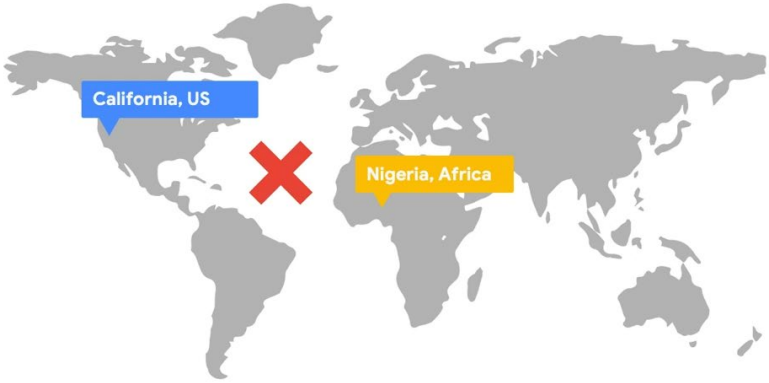
Sociotechnical Frame

How does the model operate when considered as part of a system of humans and social institutions?

The Portability Trap

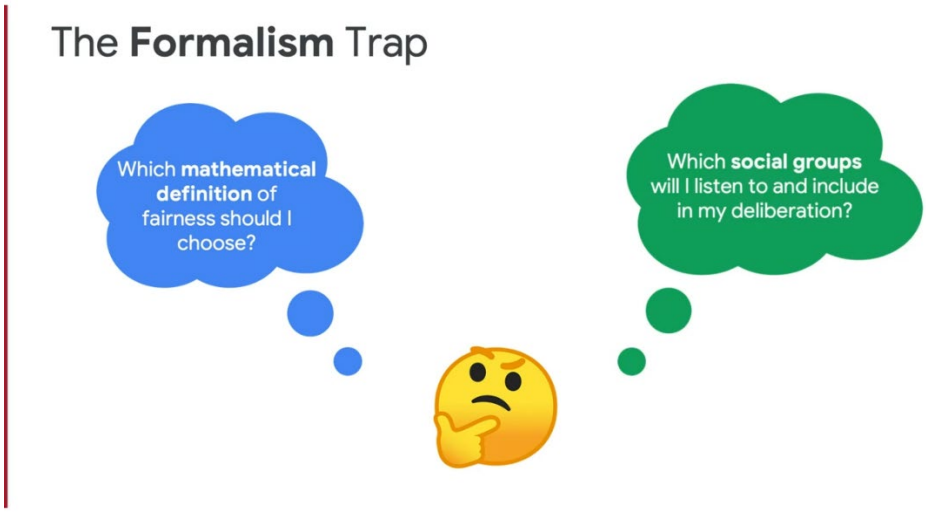


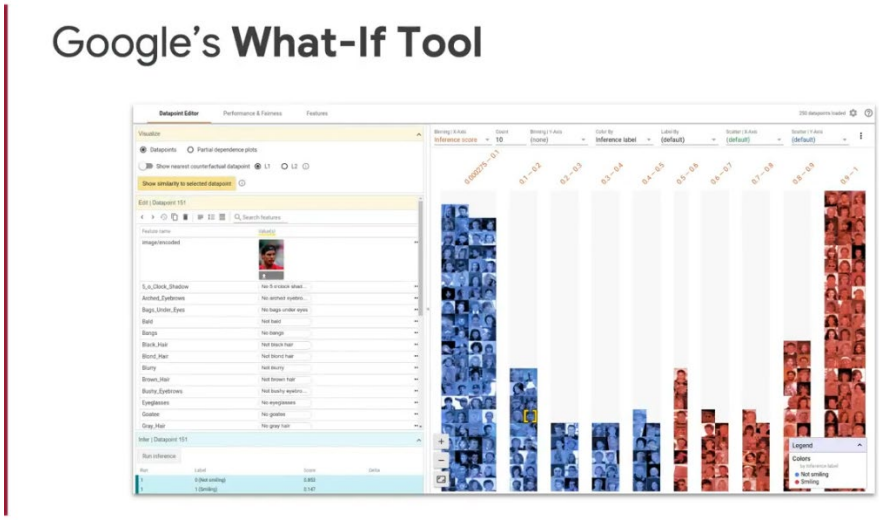
The Portability Trap



Context Matters!

Repurposing algorithmic solutions may not preserve fair outcomes.





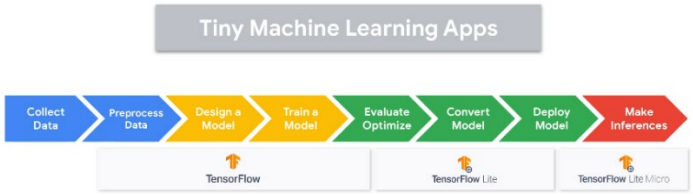
9.6 Google's What-If Tool

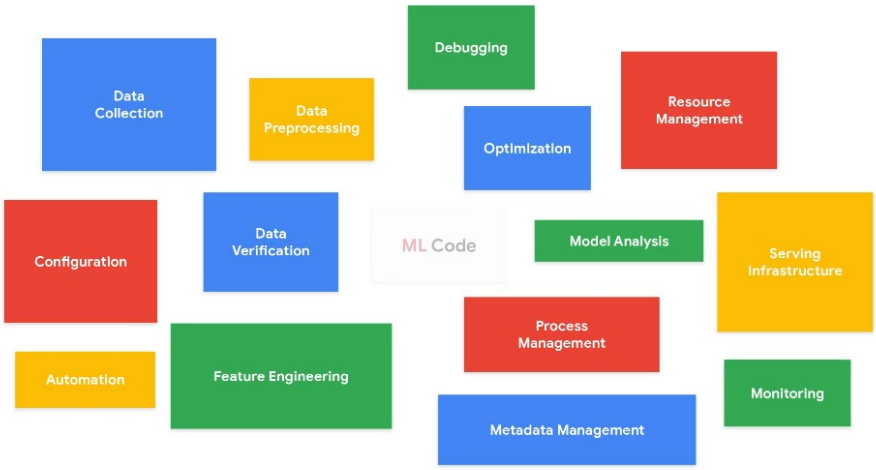
Now that you've gained familiarity with some of the common metrics for fairness in machine learning, you will have the chance to explore Google's What-If Tool (WIT). In this Colab we will walk you through exploring a real-life dataset and identifying sources of unfairness in the data across different social groups. In this case, we will train a DNN on the UCI census problem (predicting whether a person earns more than $50K from their census information), and visualize different ways to optimize the model for fairness using Google's What-If Tool.

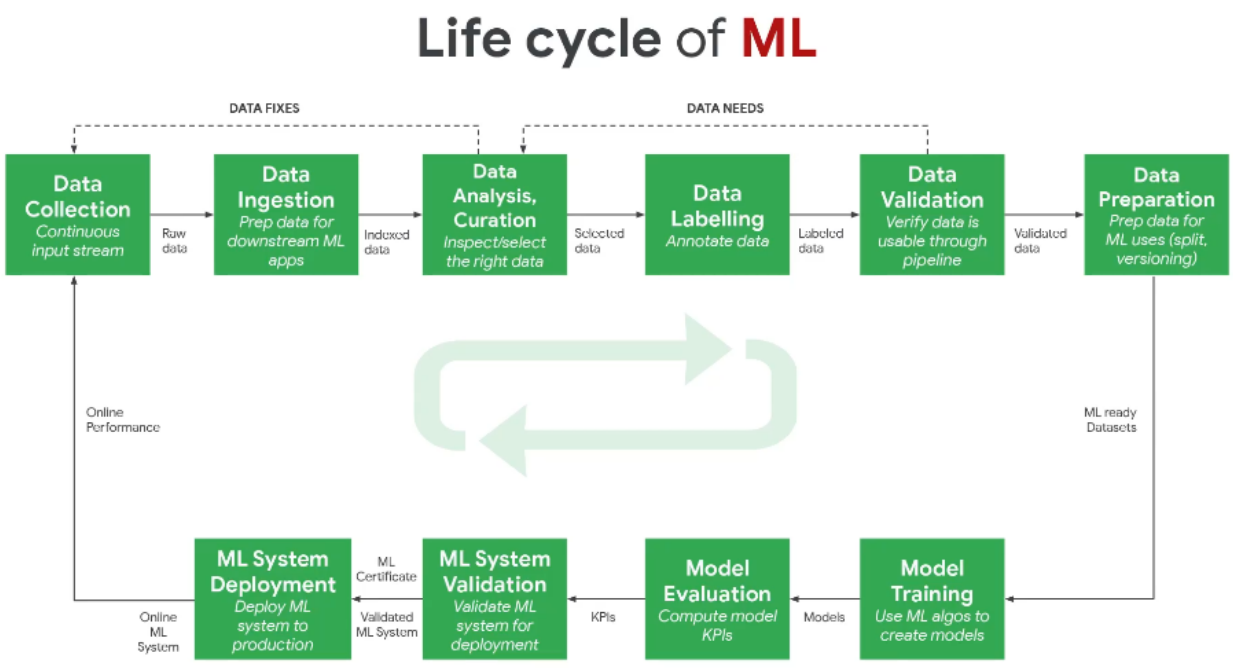
https://colab.research.google.com/github/tinyMLx/colabs/blob/master/3-10-7-WIT.ipynb

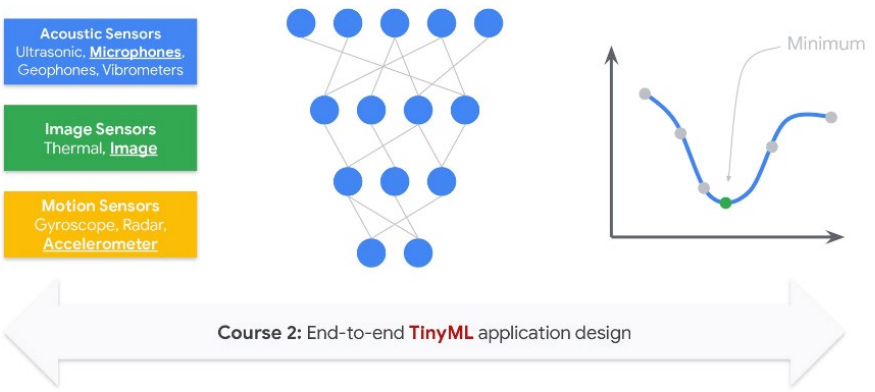
10 Summary

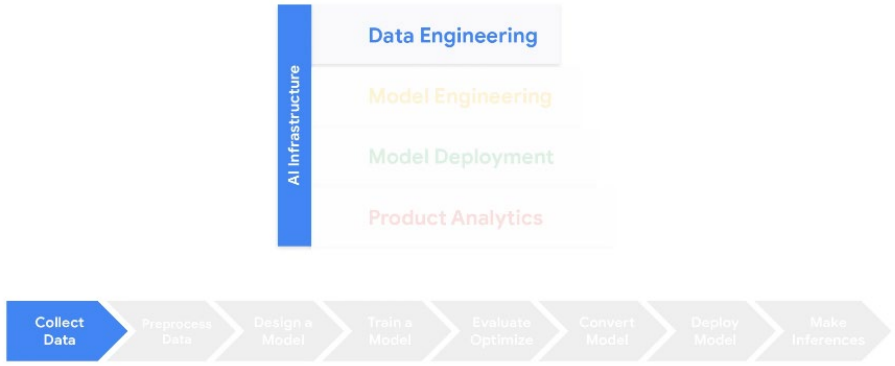
10.1 Summary











We used a dataset, that Pete build:

Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition

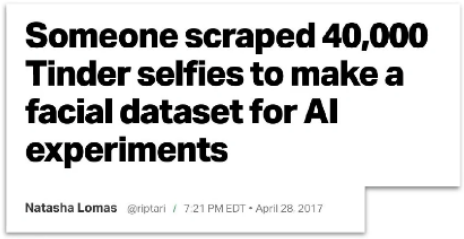
Pete Warden
Google Brain
Mountain View, California
petewarden@google.com

April 2018

Data Engineering

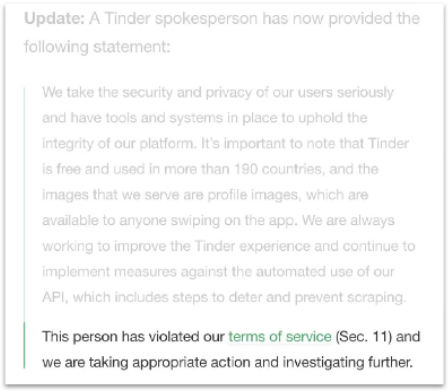
Requirements	Gathering	Refinement	Sustainment
• Problem definition • Permissions & rights • Machine & human usable format	• Data sources • People • Collection • Labeling	• Processing • Validation • Augmentation	• Storage • Security • Errors • Versioning

Data **Collection** and **Processing**



Someone scraped 40,000 Tinder selfies to make a facial dataset for AI experiments

Natasha Lomas @riptari / 7:21 PM EDT • April 28, 2017



Update: A Tinder spokesperson has now provided the following statement:

> We take the security and privacy of our users seriously and have tools and systems in place to uphold the integrity of our platform. It's important to note that Tinder is free and used in more than 190 countries, and the images that we serve are profile images, which are available to anyone swiping on the app. We are always working to improve the Tinder experience and continue to implement measures against the automated use of our API, which includes steps to deter and prevent scraping.
>
> This person has violated our terms of service (Sec. 11) and we are taking appropriate action and investigating further.

Recall: **Don't *collect*** from scratch

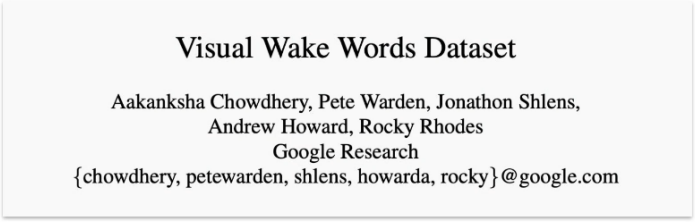
Data collection is **difficult**!

- Can we ***reuse*** existing data?

What's available?

What's missing?

Visual Wake Words **Dataset**



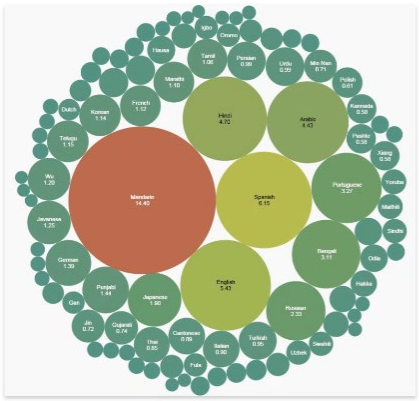
Visual Wake Words Dataset

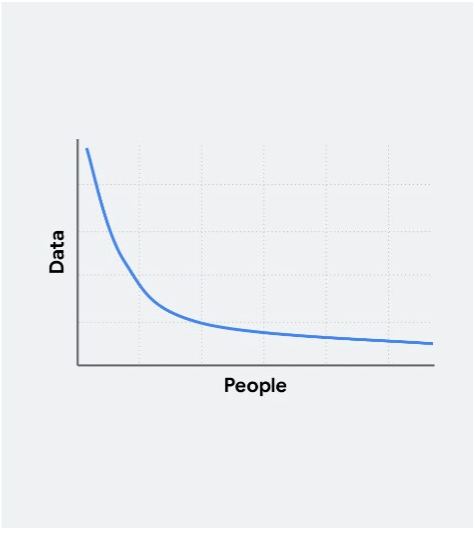
Aakanksha Chowdhery, Pete Warden, Jonathon Shlens,
Andrew Howard, Rocky Rhodes
Google Research
{chowdhery, petewarden, shlens, howarda, rocky}@google.com

Subset the dataset and create more examples:

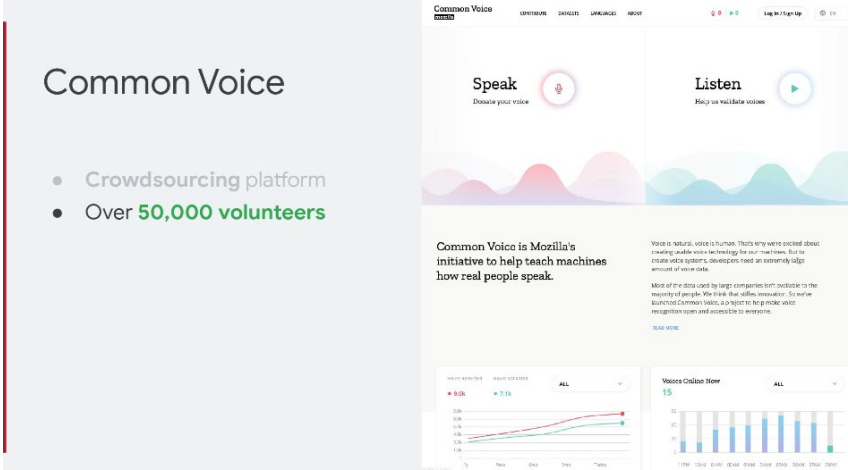
Visual Wake Words **Dataset**

To support diverse people we need a lot of data:

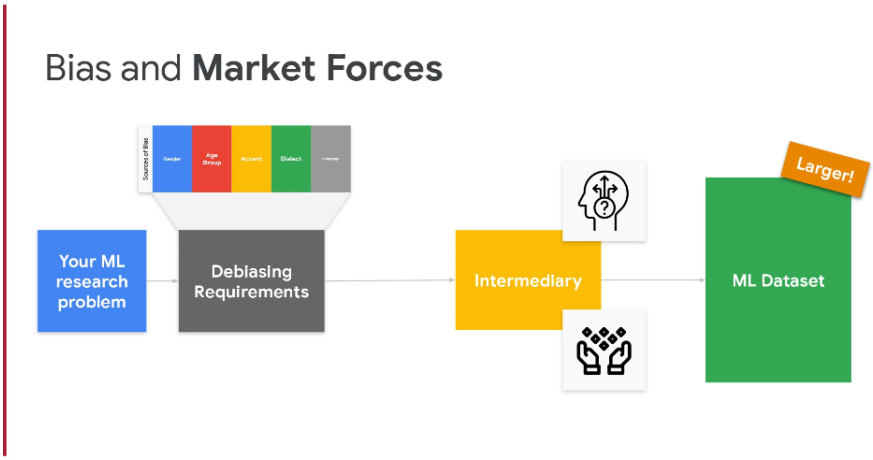


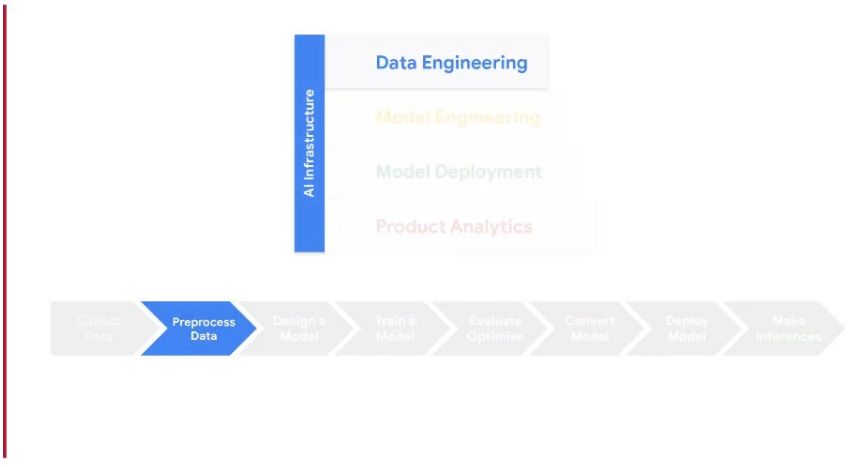


Platform to contribute:



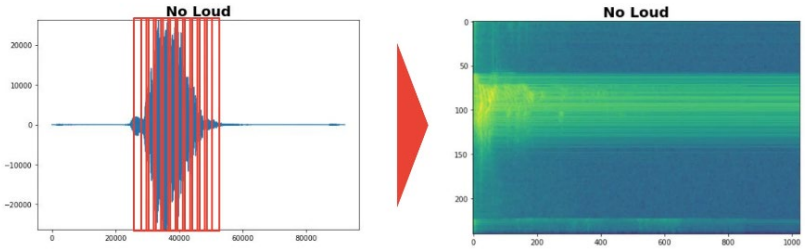
Be aware of biases:

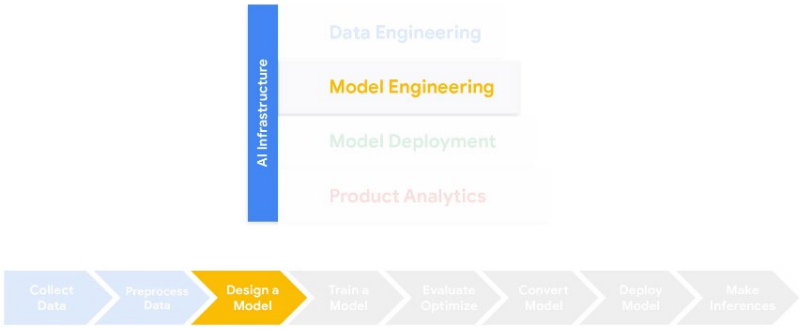




We use a features not raw data:

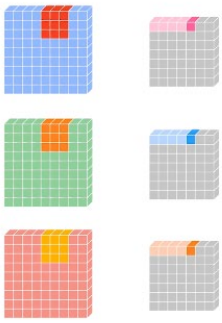
Data Preprocessing: **Spectrograms**

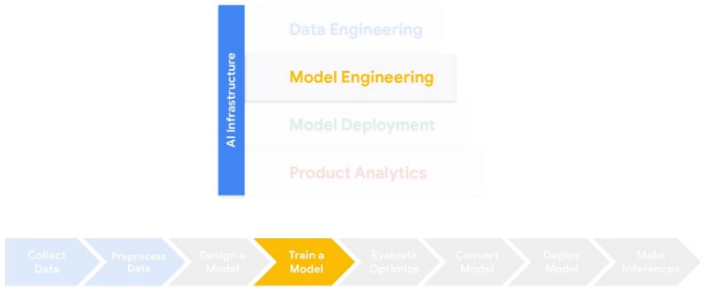


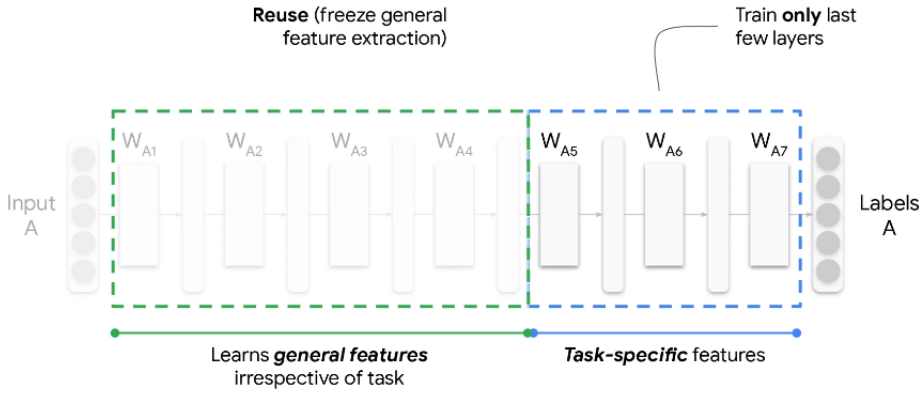


separable Depthwise Convolution *(3 Channel—e.g., RGB)*

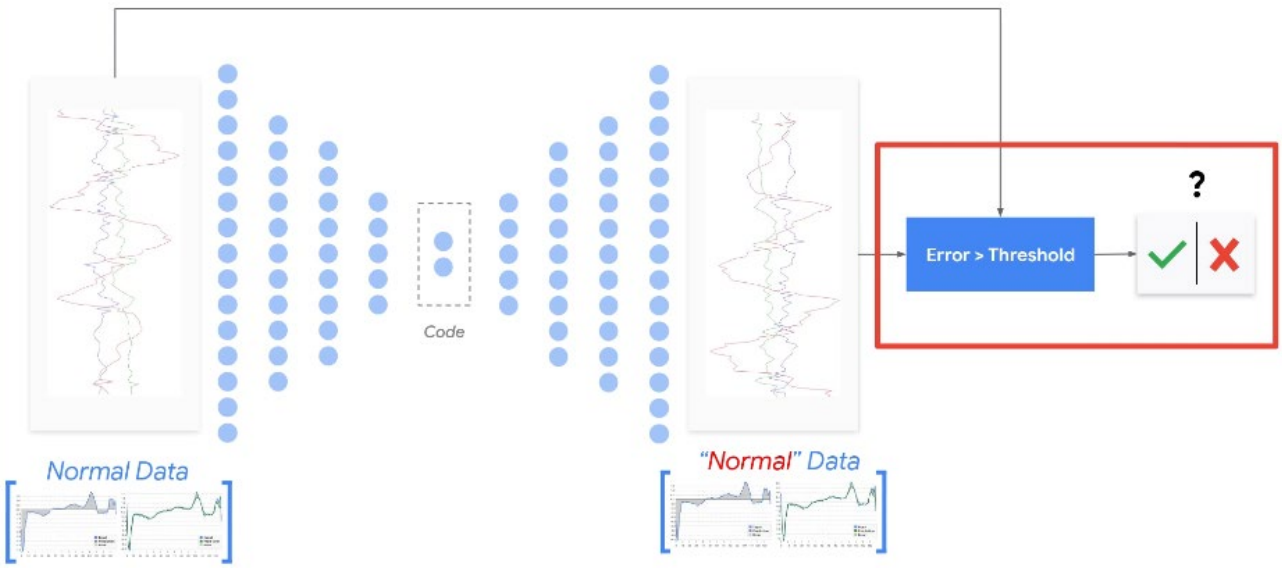
includes **pointwise** conv

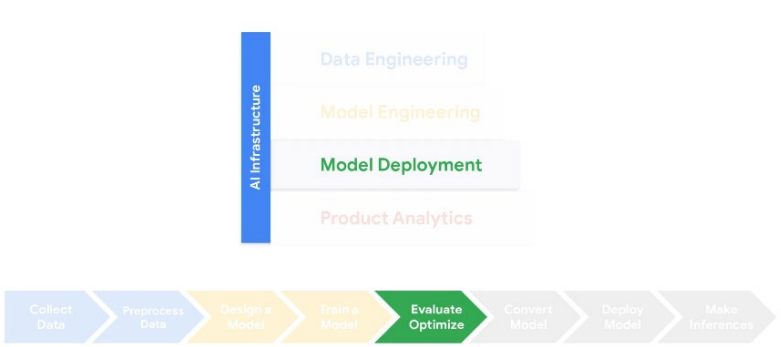


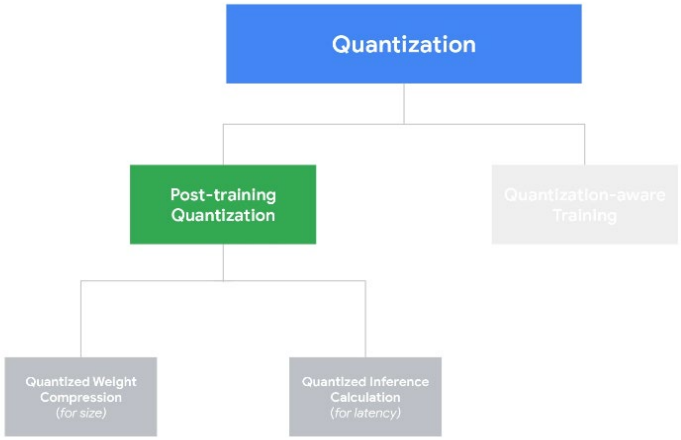


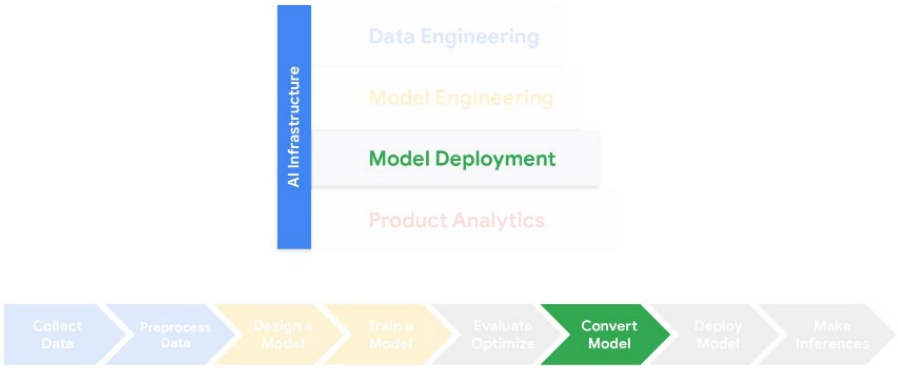


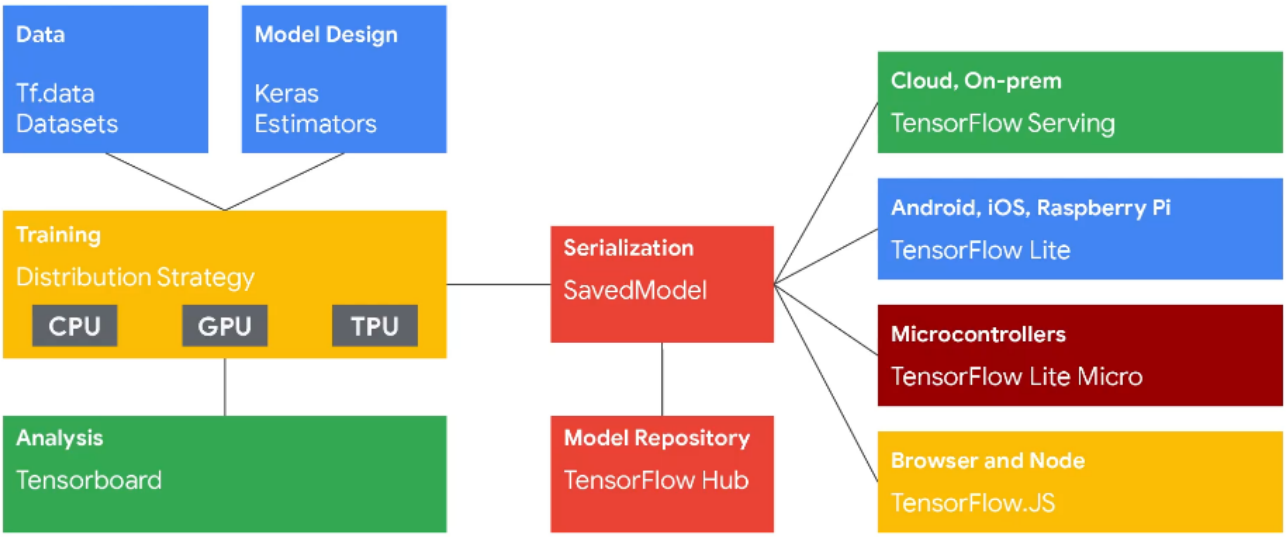
Autoencoder, uinsupervised problems:

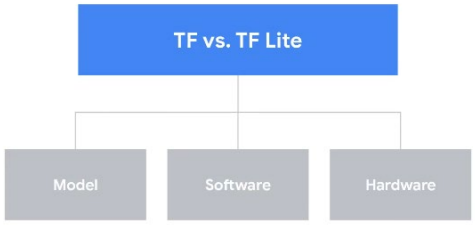


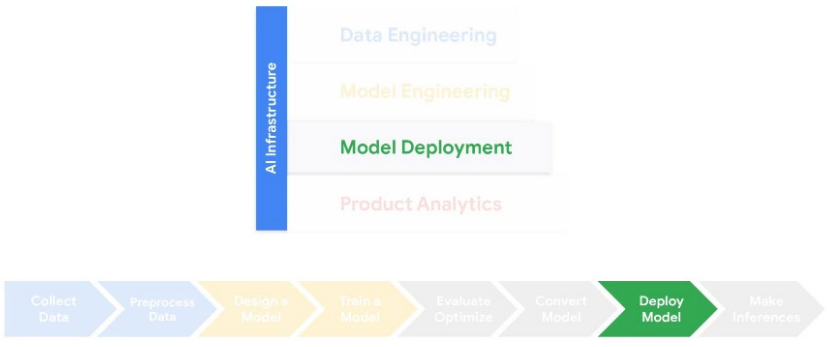

AI Infrastructure
Data Engineering
Model Engineering
Model Deployment
Product Analytics
Collect Data
Preprocess Data
Design a Model
Train a Model
Evaluate Optimize
Convert Model
Deploy Model
Make Inferences


Quantization
Post-training Quantization
Quantization-aware Training
Quantized Weight Compression
(for size)
Quantized Inference Calculation
(for latency)

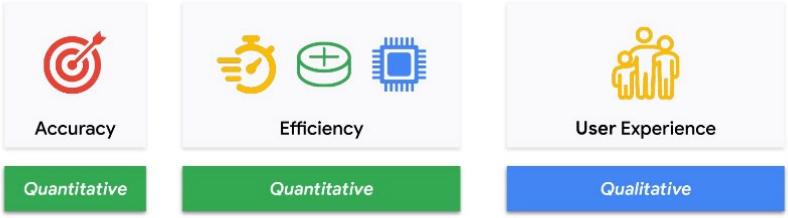

AI Infrastructure
Data Engineering
Model Engineering
Model Deployment
Product Analytics
Collect Data
Preprocess Data
Design a Model
Train a Model
Evaluate Optimize
Convert Model
Deploy Model
Make Inferences


Data
Tf.data
Datasets
Model Design
Keras
Estimators
Training
Distribution Strategy
CPU
GPU
TPU
Analysis
Tensorboard
Serialization
SavedModel
Model Repository
TensorFlow Hub
Cloud, On-prem
TensorFlow Serving
Android, iOS, Raspberry Pi
TensorFlow Lite
Microcontrollers
TensorFlow Lite Micro
Browser and Node
TensorFlow.JS

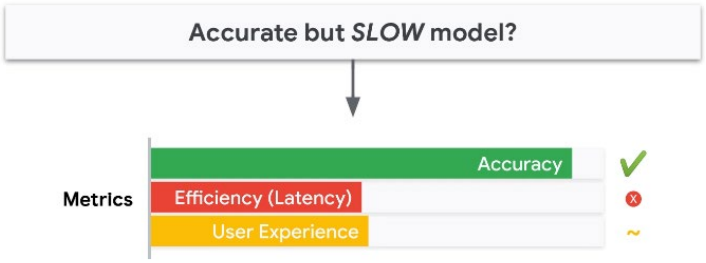

TF vs. TF Lite
Model
Software
Hardware


AI Infrastructure
Data Engineering
Model Engineering
Model Deployment
Product Analytics
Collect Data
Preprocess Data
Design a Model
Train a Model
Evaluate Optimize
Convert Model
Deploy Model
Make Inferences

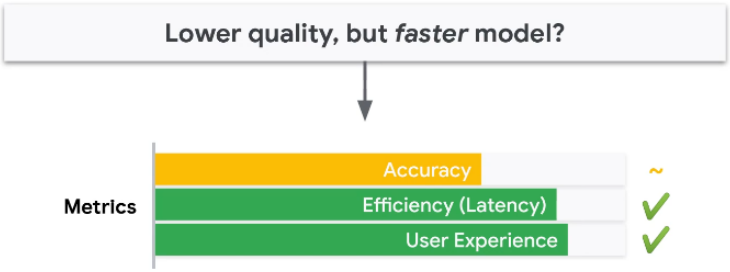
Common Metrics



Latency

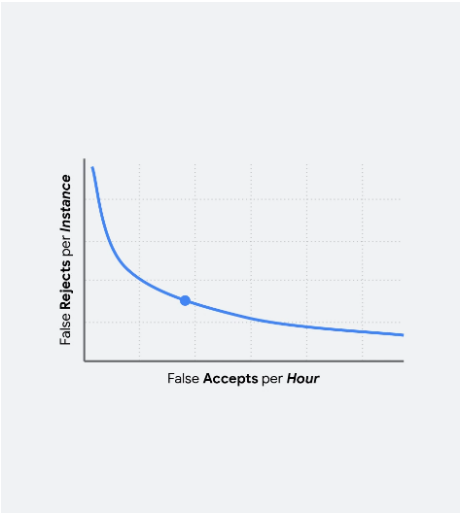


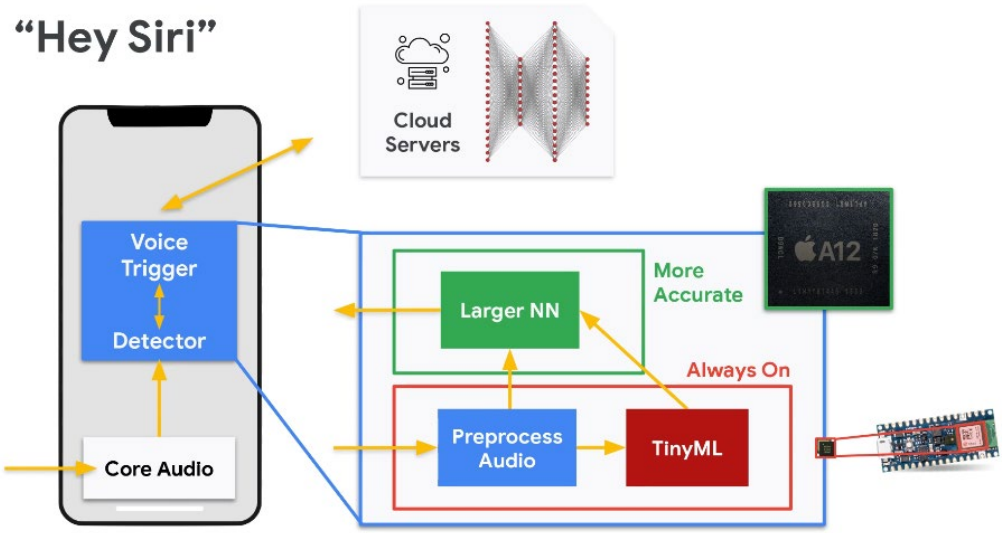
Latency

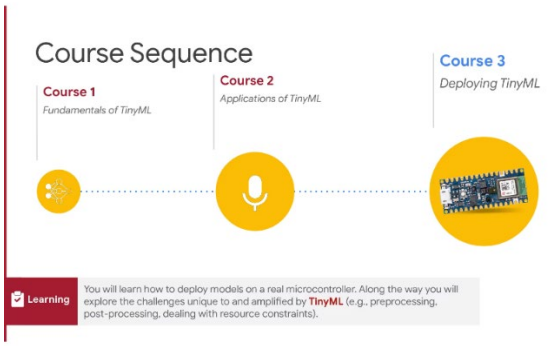


False Positive and **False Negative**

- Accuracy is measured as a tradeoff between **false accept rate** (FAR) and **false reject rate** (FRR)







10.2 Kit for Course 3

We hope you enjoyed Course 2 and we hope you'll join us for Course 3!

Looking forward to course 3 you'll need everything from Course 2, a laptop with a microphone and camera and an internet connection, as well as the course kit as we will be deploying our models onto microcontrollers.

You can buy the course kit directly from Arduino at this link:

https://store.arduino.cc/usa/tiny-machine-learning-kit

October 14, 2021 update: Due to the COVID-19 crisis and the resulting semiconductor shortage, the microcontroller used by the kit (and therefore the kit) is out of stock globally. We've been working with Arduino on this and are expecting kits to be in stock in November. We will update this page and email learners when they are available.

The course kit includes a proprietary shield which will make the camera module much more reliable. However, if you would like to purchase individual components instead of the full kit you will need to purchase the following:

- Arduino Nano BLE 33 Sense WITH HEADERS
- ArduCam OV7675
- Jumper wires and optionally a breadboard for connecting the camera to the Arduino

Keep in mind that if you don't use the Arduino kit, we have found that the camera connection is often unreliable over jumper wires, and it will be hard for your fellow classmates and others to provide guidance if you experience any issues.