TinyML & Edge Impulse

Overview

- Challenges of Centralized Machine Learning
- What is TinyML?
- TinyML Application Design Flow
- TinyML Software Suites
- TinyML Software Suites
- What is Edge Impulse?
- Case Study: Build a Machine Learning Model to Switch On/Off Devices using Voice Commands

Food for Thought!!

Let's watch this video here: https://www.youtube.com/watch?v=ySQTcCOziXs

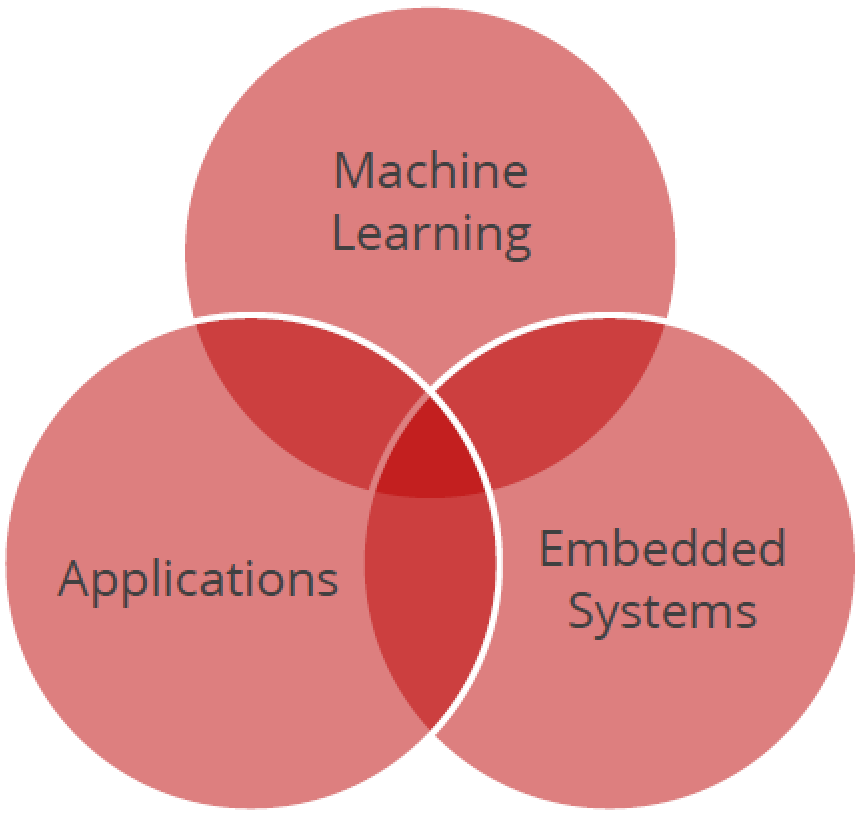
Moving ML from the Cloud to the Edge

- Most of data going to cloud is from IoT devices
- Around 328.77 million terabytes of data are generated each day. Check this reference for more information
- Less than half of structured data is actively used in decision making
- Less than 1% of unstructured data is analyzed or used at all
- ML at the edge can help use that data locally to make decision
- ML + low-power, low-cost devices + low latency = **NEW OPPORTUNITIES**

What is TinyML?

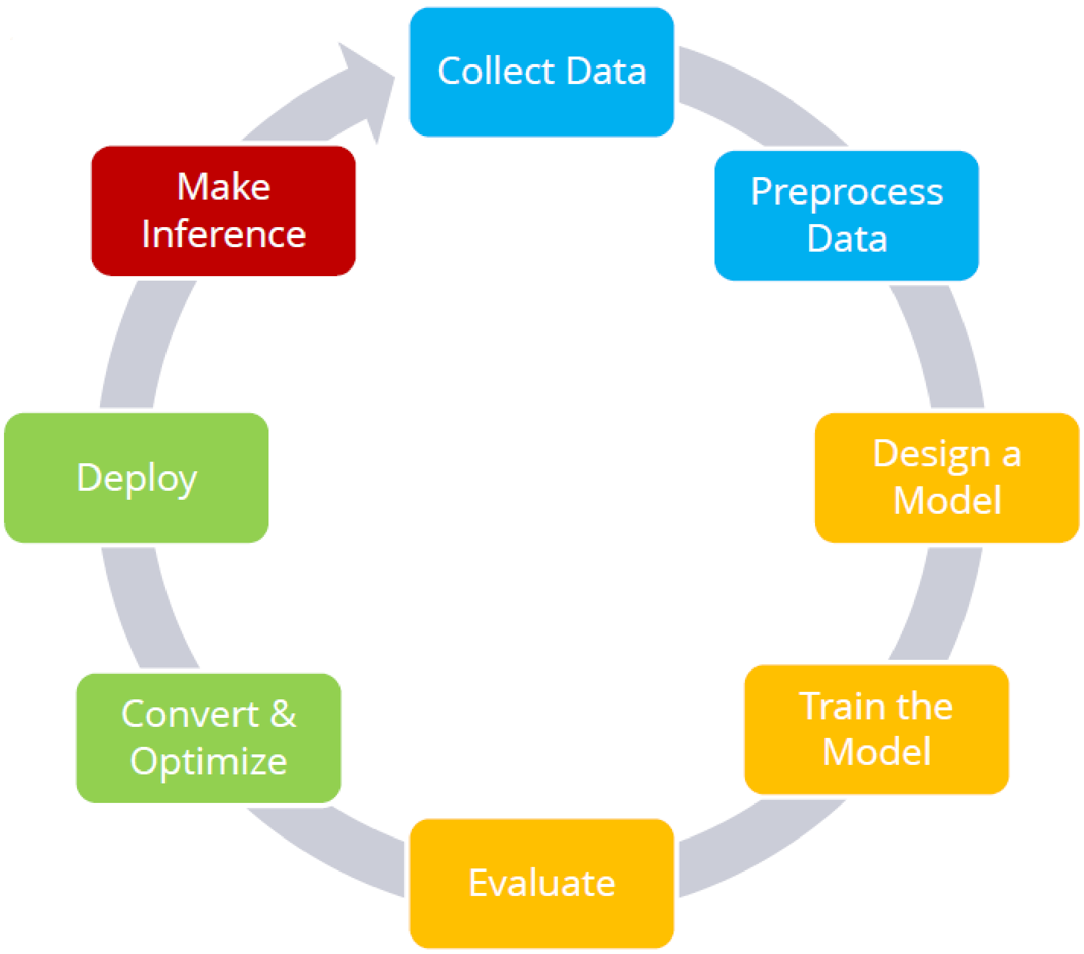
- **Tiny Machine Learning (TinyML)** is a fast-growing field of machine learning technologies and applications including algorithms, hardware, and software capable of performing on-device sensor data analytics at extremely low power consumption, typically in the mW range and below.
- TinyML enables a variety of always-on ML use-cases on battery-operated devices

TinyML Domain Interaction



TinyML lies at the Intersection

TinyML Application Design Flow



TinyML App Design Flow

TinyML Software Suites

- **TensorFlow Lite:** An open-source framework for deploying machine learning models on edge devices.
- **Arm Cortex-M microcontroller-based machine learning:** A suite of tools and libraries for implementing machine learning algorithms on Arm Cortex-M microcontrollers.
- **Zephyr Project:** An open-source real-time operating system for building and deploying machine learning algorithms on IoT devices.

- **Arduino and Arduino Create:** An open-source platform for building and programming electronic devices, including those with machine learning capabilities. Create is an online platform for creating and deploying IoT applications, including those using TinyML algorithms.
- **Edge Impulse:** A platform for developing and deploying machine learning algorithms on edge devices.
- **OpenMV:** An open-source hardware and software platform for developing vision-based applications on microcontrollers.
- **PYNQ:** This is an open-source framework for developing ML algorithms on Xilinx Field-Programmable Gate Arrays (FPGAs).
- **MicroPython:** An open-source Python interpreter that can be used to develop and run TinyML algorithms on microcontrollers and other low-power devices.

In this course, we will use **Edge Impulse** for the following reasons:

- Ease of use with customized models.
- Requires minimal setup on your machine.
- Offers a Browser-based solution that is compatible with every operating system.
- Supports wide number of development boards.
- Scalable and integrates well with other tools.
- Offers low-latency and energy efficient solutions.

Edge Impulse Process Flow

Edge Impulse reduces the number of tasks required to design ML application to 5 UI-led tasks:

- Collect Data
- Clean Data and Generate Features
- Train Your Model and Test It
- Find the Best Model for Your Target Device
- Deploy the Model

Case Study: Build a Machine Learning Model to Turn On/Off Devices using Voice Commands

1. Create Edge Impulse Account

Before we start using Edge Impulse, you need to create a community edition account using your @andrew.cmu.edu email. Use this URL to sign up: https://studio.edgeimpulse.com/signup

2. Create the Dataset

The dataset that we will use contains Speech commands for more than 60,000 instances of one-second-long utterances of 30 different spoken keywords such as "yes", "no", "up", "down", "left", "right", "on", "off", and many more. These commands are stored as digital audio files. The audio files are sampled at 16 kilohertz. You may refer to the paper "*Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition" by Pete Warden to know more about the dataset.*

In [1]:

```
!pip install -U -q tensorflow tensorflow_datasets
#!apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
```

2.1 Import the necessary packages

In [2]:

```
import os
import pathlib

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from numpy import random
import shutil, errno

import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import models
```

```
WARNING:tensorflow:From C:\Users\mfarag\AppData\Roaming\Python\Python39\site-packages\ke
ras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. P
lease use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

2.2 Download the speech commands dataset

A new directory named data will be created, which will contain subdirectories corresponding to all 30 keywords. Each subdirectory will contain the corresponding audio files in WAV format.

In [3]:
```
DATASET_PATH = 'data/'

data_dir = pathlib.Path(DATASET_PATH)
if not data_dir.exists():
    tf.keras.utils.get_file(
      'speech_commands.zip',
      origin='http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz',
      extract=True,
      cache_dir='.', cache_subdir='data')
```

We want to develop a Turn on/Off Switch that includes keyword spotting algorithm so it can detect the target keywords "on" and "off". However, creating a binary classifier may not be sufficient, as the input audio could contain human speech with other words or may have no background audio at all

2.3 Our Multi-class Classifier

We will create a multi-class classifier to detect four different class labels: "on", "off", "others", and "silent". So, we need to form a new dataset from the original Speech Commands dataset containing audio instances corresponding to the four target classes we are going to detect. We will create a new empty directory, mydatset, under the parent directory data and we will populate the labels accordingly.

2.4 Get different class labels

In [4]:
```
commands = np.array(tf.io.gfile.listdir(str('data')))
commands = commands[commands != 'README.md']
print('Commands:', commands)
```

```
Commands: ['.DS_Store' 'backward' 'bed' 'bird' 'cat' 'dog' 'down' 'eight' 'five'
 'follow' 'forward' 'four' 'go' 'happy' 'house' 'learn' 'left' 'LICENSE'
 'marvin' 'mydataset' 'nine' 'no' 'off' 'on' 'one' 'right' 'seven'
 'sheila' 'six' 'speech_commands.zip' 'stop' 'testing_list.txt' 'three'
 'tree' 'two' 'up' 'validation_list.txt' 'visual' 'wow' 'yes' 'zero'
 '_background_noise_']
```

2.5 Form the directory for string on, off, others and silent data

In [6]:
```
try:
    os.mkdir('data/mydataset')
except Exception as e:
    print("Error creating folder. Error details {}".format(e))
```

In [7]:
```
def copy_data(src, dst):
    try:
        shutil.copytree(src, dst)
    except OSError as exc: # python >2.5
        if exc.errno in (errno.ENOTDIR, errno.EINVAL):
            shutil.copy(src, dst)
        else: raise
```

In [8]:
```
copy_data('data/on','data/mydataset/on')
copy_data('data/off','data/mydataset/off')
```

In [9]:
```
# You may use the following code instead of calling the copy_data() function above if yo
#!cp -r 'data/on' 'data/mydataset'
#!cp -r 'data/off' 'data/mydataset'
```

In [10]:
```
os.mkdir('data/mydataset/silent')
os.mkdir('data/mydataset/others')
```

2.5.1 Populate "Others" label data

In [11]:
```
other_labels = ["yes", "no", "up", "down", "left", "right","bed", "bird", "cat", "dog",
sample_per_label = 150


for label in other_labels:
    path = 'data/'+label
    f = os.listdir(path)
    num_files = len(f)
    file_indx = np.arange(num_files)
    random.shuffle(file_indx)
    for i in range(sample_per_label):
        index = file_indx[i]
        file_name = f[index]
        source = path + '/' + file_name
        destination = 'data/mydataset/others/' + file_name
        # copy only files
        if os.path.isfile(source):
            shutil.copy(source, destination)
```

2.5.2 Create "Silent" label data

In [12]:
```
import scipy
from scipy.io.wavfile import write

fs = 16000
num_files = 2400

for i in range(num_files):
    sample = np.zeros(fs)
    filename = str(i*100)+'silent.wav'
    sample = sample + 0.01*i*random.randn(fs)
    scipy.io.wavfile.write('data/mydataset/silent/'+filename, fs, sample.astype(np.int16
```

2.6 Upload the Data to Edge Impulse

Use the data inside **mydataset** folder. Follow the Demo Carefully and watch the Lecture recording if needed. It's OK if some of the silent data were not uploaded due to identical hash values. You may also try to create the silent files yourself and upload them.

3. Add a Processing Block to Generate Features for ML Model

We will use **Mel Frequency Cepstral Coefficient (MFCC)** algorithm for training our model. MFCC is an audio feature extraction algorithm commonly used in speech processing and speaker recognition applications. In MFCC, the frequency bands of the audio are mapped on the Mel scale, which closely approximates the human auditory system.

MFCC is considered as a better feature extraction technique compared to the spectrogram-based approach in speech processing applications. The MFCC components are calculated through the following steps:

1. Break the audio signals into small windows.
2. Calculate the spectrum of the windows via Short-Time Fourier Transform.
3. Map the spectrum power into equivalent Mel scale.
4. Calculate the logarithm of the power components at the Mel frequencies.
5. Calculate the discrete cosine transform of the logarithm powers.
6. The resulting amplitudes represent the MFCC values.

4. Add a Learning Block to Train Your Model and Test It

Keep in mind, the device we will target for this example is: **Arduino Nano 33 BLE Sense (Cortex-M4F 64MHz)**

5. Add a new Device as Your Smartphone using QR Code and Use Your Phone to Conduct Live Classification

Is this the best model for the Target Hardware?

We will answer this question next lecture!