



Electrical and Computer Engineering
Erik Jonsson School of Engineering & Computer Science
The University of Texas at Dallas

CE6302: Microprocessor and Embedded Systems

Dr. Tooraj Nikoubin

TinyML Lab 3- Image Classification

Submitted by:

Muripa Uppaluri (mxu220008)

Chandanam Sai Nived (sxc210186)

Table of Contents

1. Project Objective	1
1.1 Introduction	1
1.2 Espressif ESP-EYE (ESP32)	1
2. Edge Impulse	2
2.1 Sampling New Data	2
2.2 Designing an Impulse.....	2
2.3 Configuring the processing block.....	3
2.4 Configuring the Neural Network.....	4
2.5 Validating our model.....	5
2.6 Deploying back to device.....	5

TinyML - Image Classification

1. Project Objective

- A brief understanding about ML algorithms
- A brief understanding about TI Launchpad (CC1352P) and Booster Sensors
- Use machine learning to build a system that can recognize objects in your house through a camera task known as image classification

1.1 Introduction

Machine learning is a subfield of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behavior. Artificial intelligence systems are used to perform complex tasks in a way that is similar to how humans solve problems.

Tiny machine learning, or TinyML, is an emerging field that is at the intersection of machine learning and embedded systems. An embedded system is a computing device that usually is small, or tiny, that operates with low power, extremely low power. So much so that some of these devices can run for days, weeks, months, sometimes even years on something like a coin cell battery. TinyML is a type of machine learning that shrinks deep learning networks to fit on tiny hardware. It brings together Artificial Intelligence and intelligent devices.

Edge computing brings computation and data storage closer to the origin of data. Majority of the edge devices that are integrated with IoT-based ecosystems are initially designed to collect sensor data and transmission of the data to neighborhood or remote cloud.

1.2 Espressif ESP-EYE (ESP32)

This Espressif ESP-EYE (ESP32) is a compact development board, equipped with a 2-Megapixel camera and a microphone. ESP-EYE also offers plenty of storage, with 8 MB PSRAM and 4 MB SPI flash - and it's fully supported by Edge Impulse. There are plenty of other boards built with ESP32 chips, and of course there are custom designs utilizing ESP32 SoM. Edge Impulse firmware was tested with ESP-EYE and ESP FireBeetle boards, but there is a possibility to modify the firmware to use it with other ESP32 designs.



2. Edge Impulse

It is a cloud service for developing machine learning models in the TinyML targeted edge devices. This supports AutoML processing for edge platforms. It also supports several boards including smart phones to deploy learning models in such devices.

2.1 Sampling New Data

Machine learning works best with lots of data, so a single sample won't cut it. Now is the time to start building your own dataset. The device will capture a various images and transmit it to Edge Impulse:

- 50 images of an Apple
- 50 images of an Avocado
- 50 images of neither an apple nor an avocado

2.2 Designing an Impulse

With the training set in place, an impulse is designed. An impulse takes the raw data, adjusts the image size, uses a preprocessing block to manipulate the image, and then uses a learning block to classify new data. Preprocessing blocks always return the same values for the same input, while learning blocks learn from past experiences. For this, we'll use the Images preprocessing block. This block takes in the color image, optionally makes the image grayscale, and then turns the data into a features array. Then we'll use a Transfer Learning learning block, which takes all the images in, and learns to distinguish between the three classes.

The screenshot displays the Edge Impulse web interface for designing an impulse. It consists of four main colored blocks in a row: 'Image data' (red), 'Image' (light blue), 'Transfer Learning (Images)' (purple), and 'Output features' (teal). Below these blocks are two dashed boxes for adding more blocks: 'Add a processing block' and 'Add a learning block'. A 'Save Impulse' button is located below the 'Output features' block.

- Image data (red block):** Contains 'Input axes' set to 'image', 'Image width' and 'Image height' both set to '96', and 'Resize mode' set to 'Squash'. A note at the bottom states: 'For optimal accuracy with transfer learning blocks, use a 96x96 or 160x160 image size.'
- Image (light blue block):** Contains 'Name' (empty field), 'Input axes (1)' with a checked checkbox for 'image'.
- Transfer Learning (Images) (purple block):** Contains 'Name' (empty field), 'Input features' with a checked checkbox for 'Image', and 'Output features' set to '3 (Apple, Avacado, Random)'. (Note: 'Avacado' is misspelled in the image).
- Output features (teal block):** Contains 'Output features' set to '3 (Apple, Avacado, Random)'.

Buttons and options at the bottom:

- 'Add a processing block' (light blue dashed box with a lightning bolt icon)
- 'Add a learning block' (light blue dashed box with a flask icon)
- 'Save Impulse' (green button)

2.3 Configuring the processing block


After configuring the processing block. This will show the raw data on top of the screen, and the results of the processing step on the right. In the Feature generation screen, we can :


- Resize all the data.
- Apply the processing block on all this data.
- Create a 3D visualization of your complete dataset.

Afterwards the Feature explorer will load. This is a plot of all the data in our dataset. Because images have a lot of dimensions (here: $96 \times 96 \times 3 = 27,648$ features) we run a process called dimensionality reduction on the dataset before visualizing this. Here the 27,648 features are compressed down to just 3, and then clustered based on similarity. Even though we have little data we can already see some clusters forming and can click on the dots to see which image belongs to which dot.

Raw data

Show: Apple Apple.3ucpmb2u (Apple)



Raw features 

0xafbabb, 0xafbabb, 0xadbbb, 0xacb9ba, 0xacb7b8, 0xad7b7, 0xad7b7, 0xad7b8, ...

Parameters


Image


Color depth ⓘ RGB

Save parameters

DSP result


Image




Processed features 

0.6863, 0.7294, 0.7333, 0.6863, 0.7294, 0.7333, 0.6784, 0.7333, 0.7333, 0.6745, ...

On-device performance ⓘ

 PROCESSING TIME
1 ms.

 PEAK RAM USAGE
4 KB

2.4 Configuring the Neural Network

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. The network that we're training here will take the image data as an input and try to map this to one of the three classes.

It's very hard to build a good working computer vision model from scratch, as you need a wide variety of input data to make the model generalize well, and training such models can take days on a GPU. To make this easier and faster we are using transfer learning. By only retraining the upper layers of a neural network, leading to much more reliable models that train in a fraction of the time and work with substantially smaller datasets.

Model

Model version: ?

Quantized (int8) ▾

Last training performance (validation set)



ACCURACY
100.0%



LOSS
0.10

Confusion matrix (validation set)

	APPLE	AVACADO	RANDOM
APPLE	100%	0%	0%
AVACADO	0%	100%	0%
RANDOM	0%	0%	100%
F1 SCORE	1.00	1.00	1.00

Data explorer (full training set) ?

- Apple - correct
- Avacado - correct
- Random - correct



On-device performance ?



INFERRING ...
105 ms.



PEAK RAM USA...
333.8K



FLASH USAGE
580.0K

2.5 Validating our model

The model was trained only on the training data, and thus we can use the data in the testing dataset to validate how well the model will work in the real world. This will help us ensure the model has not learned to overfit the training data, which is a common occurrence

SAMPLE NA...	EXPECTED OUT...	LENG...	ACCURACY	RESULT	
Apple.3uc...	Apple	-	100%	1 Apple	⋮
Apple.3uc...	Apple	-	100%	1 Apple	⋮
Apple.3uc...	Apple	-	100%	1 Apple	⋮
Apple.3uc...	Apple	-	100%	1 Apple	⋮
Avacado.3...	Avacado	-	100%	1 Avacado	⋮
Avacado.3...	Avacado	-	100%	1 Avacado	⋮

2.6 Deploying Back to Device

With the impulse designed, trained, and verified we can deploy this model back to your device. This makes the model run without an internet connection, minimizes latency, and runs with minimum power consumption. Edge Impulse can package up the complete impulse - including the signal processing code, neural network weights, and classification code - up in a single C++ library that you can include in embedded software.

```
Starting inferencing in 2 seconds...
Taking photo...
Predictions (DSP: 8 ms., Classification: 1176 ms., Anomaly: 0 ms.):
  Apple: 0.01172
  Avacado: 0.12500
  Random: 0.86328
```

The output shows the prediction of the sample as the 99% probability of being Random after mangoes are displayed by using already neural networks which are trained by classified data that are labeled earlier.