

Project Report

Counter Detection with a ESP32

Author 1: Nisha Kumari
Matriculation No.: 7023211
Author 2: Dereck Alpizar
Matriculation No.: 7023782
Author 3: Neeranjana Jayamurugan Karthikeyani
Matriculation No.: 7023553
Course of Studies: Business Intelligence and Data Analytics

First examiner: Prof. Dr. Elmar Wings
Submission date: January 31, 2023

Contents

Acronyms	vii
1 Introduction	1
1.1 Problem Description	2
1.2 Challenges	2
2 Domain Knowledge	5
2.1 Tiny ML	5
2.2 TensorFlow/ TensorFlow Lite/ Tensor Flow Micro	5
2.3 Algorithm: Convolutional Neural Network	6
2.3.1 Pooling Layer	8
2.3.2 Fully-Connected Layer	8
2.3.3 CNN Example with code	9
2.4 ESP32-CAM	20
2.4.1 Components in ESP32CAM Development Board	20
2.4.2 ESP32-CAM Microcontroller Features	21
2.4.3 ESP32-S Microprocessor Specification	22
2.5 ESP32-CAM Camera Module	24
2.6 Visual Studio Code	25
2.6.1 Installation and setup	26
2.6.2 ESP-IDF Plug-in	31
2.6.3 Microprocessor Test	33
2.6.4 OV2640 Camera Test	35
3 Knowledge Discovery in Databases Process	39
4 Development	41
4.1 Database Description	41
4.2 Data Description	41
4.3 Data Selection	41
4.4 Data Preparation	41
4.5 Data Transformation	42
4.6 Data Mining	42
4.6.1 Model Description	42
4.6.2 Model Training	42
4.7 Results	42

Contents

5	Deployment	43
5.1	Meter Counter Digitized	43
5.1.1	Software	43
5.1.2	Hardware	44
5.1.3	Configuration	47
5.1.4	Tests	49
5.2	Monitoring	50
6	Conclusion / Open Questions	53
7	Appendix	55
7.1	Material List	55
7.2	Description of the Python Package: Keras	61
7.3	Description of the Python Package: NumPy	65
7.4	Description of the Python Package: Matplotlib	69
	Bibliography	75

List of Figures

2.1	Tensor Flow Lite Workflow	6
2.2	A CNN sequence to classify handwritten digits	7
2.3	Pooling Layer in CNN	8
2.4	Components in ESP32-CAM	20
2.5	OV2640 Camera Module	24
2.6	Visual Studio Code Download Options.	26
2.7	Visual Studio Code License Agreements.	27
2.8	Visual Studio Code Destination Location.	28
2.9	Visual Studio Code Start Menu Folder.	28
2.10	Visual Studio Code Additional Tasks.	29
2.11	Visual Studio Code Install.	29
2.12	Visual Studio Code Finished Installation.	30
2.13	Visual Studio Code Initial Window.	30
2.14	Visual Studio Code "Hello ESP32 Basic Code."	31
2.15	Visual Studio Code Install Extension "esp-df".	31
2.16	Configure ESP-IDF extension: Express.	32
2.17	Download ESP-IDF version from: GitHub.	32
2.18	Successfully installed ESP-IDF.	32
2.19	ESP32 Camera Clone Repository.	33
2.20	Micro processor (ESP32-S) Test in ESP32-CAM.	35
2.21	Camera (OV2640) and SD-Card Test in ESP32-CAM.	37
3.1	KDD process	40
5.1	Important Libraries	43
5.2	Methods in Wrapper Class	44
5.3	ESP32-CAM-MB connection plug to ESP32-CAM.	46
5.4	ESP32-CAM field application.	48
5.5	Software flowchart	50
6.1	Water Meter	53
7.1	ESP32-CAM.	55
7.2	OV2640 Camera.	56
7.3	ESP32-CAM-MB USB Programmer.	57
7.4	Secure Digital Card.	57
7.5	ESP32-CAM Case.	58
7.6	USB cable - USB A / micro USB B.	59

List of Figures

7.7	Python 3.10.	59
7.8	Visual Studio Code v1.73.1.	60
7.9	ESP-IDF Plug-In v1.5.1.	61

List of Tables

Hardware (HW) Deployment Description	44
--	----

Acronyms

AI Artificial Intelligence

CNN Convolutional Neural Network

DIY Do It Yourself

HW Hardware

IoT Internet of Things

ML Machine Learning

NN Neural Networks

OCR Optical Character Recognition

PC Personal Computer

PIL Python Imaging Library

SW Software

tf TensorFlow

TFLite TensorFlow-Lite

VSCode Visual Studio Code

1 Introduction

Over the years, technological advancements have constantly been evolving. Among the rapidly emerging innovations, the concept of smart devices and smart homes has been fancied more and more, by both customers and developers. These advancements have one goal - to make our lives easier. Today in the market we have products such as smart bulbs, smart vacuum cleaners, smart speakers, and many more. What each of these products shares in common is an Edge device [PS20], an embedded system [Rei22], a TinyML framework, and a Machine Learning (ML) algorithm.

As per [OAG+21], ML has played a crucial role regarding innovation in technical and scientific applications. Running and analyzing large amounts of data on a complex ML algorithm requires a significant amount of resources and capabilities, which is a barrier to the mainstreaming of ML in the industry. The recent advances and collaboration of low-power embedded devices and ML could be understood better through TinyML.

TinyML is a paradigm that makes it easy to run machine learning on embedded edge devices with a very little processor and memory. TinyML typically enables IoT-based embedded edge devices to transition to low-power systems by integrating sophisticated power management modules [Ray22]. Moreover, TinyML eliminates the need for cloud server connectivity and improves responsiveness and privacy measures while running using a coin-size battery [OAG+21].

The main advantages of Edge Artificial Intelligence (AI) are data security, latency, energy saving, cost, and no connection dependency. It is required to have a special approach for training the Neural Networks (NN) in Edge to reduce the size of the neural network, accelerate the training process, reduce the memory consumption, and avoid the dependency of needing all data at the same time for this training stage [RS22]. Some examples of Edge devices are micro controllers, Raspberry Pi, Arduino, etc.

Any ML model for an application in speech, text, or image recognition requires features as input. [ZC18] describes the feature as a numeric representation of raw data. It also explains that feature engineering is required to extract features from raw data, for example, an image, and transform those features into formats suitable for machine learning. Therefore, feature engineering will be a part of this project too. However, this will not be necessarily done manually as there are Tensor FLOW libraries that will be helpful in this regard.

1 Introduction

This introduction section explains the problem description of the project and the challenges associated. The domain knowledge section explains the concepts, tools and technologies that are going to be used in this project. Furthermore, the KDD process used can be referred in the section named Knowledge discovery in Database process. The development section mentions the implementation of the steps of the KDD process. Deployment section elaborates the requirements and steps to make the system up and running. At the end the conclusion and open questions about the project are mentioned. Additionally, information about material list and packages can be found in the Appendix section.

1.1 Problem Description

Harnessing the above concepts, [Spa20], [BLO17], [Lis21] built systems as part of automating their homes for meter readings. This implies it is quite cheap and simple to have a smart meter just by using a device with an integrated camera and an appropriate TinyML framework. One of the advantages of a digital meter is that you won't have to manually check the readings to monitor the usage, instead, you will get an automated notification from your smart meter when you are about to reach the limit. The use cases mentioned above include capturing an image of the meter counter and identifying the readings of the meter. This involves image recognition, Optical Character Recognition (OCR), and data augmentation using TensorFlow-Lite (TFLite).

Similar to the aforementioned use cases, the goal of this project includes digitizing an analog meter. Initially, by recognizing one digit using ESP32-CAM and TFLite libraries with the help of a model trained using Convolutional Neural Network (CNN) algorithm. To test this, a printed picture of a digit will be presented in front of the ESP32-CAM system and an output with the corresponding digit is expected. Subsequent improvements will be made depending on the test results.

1.2 Challenges

Concerning the problem description, there might be a few challenges related to OCR. For instance, the inappropriate font size can make character recognition difficult. Also, OCR might detect only partial text, which would lead to erroneous results [Mar21]. Moreover, the hardware capacity and efficiency can pose another challenge. Another challenge we might face is NaN cases where the number from the meter is not clear on a specific value but is stuck on the transition to the next value. Regarding TinyML, there could be certain issues with inconsistent power usage and memory constraints [Sag20]. Some additional challenges we expect to arise are the quality of images (the light conditions and clarity), determining the size of the database for efficient

performance, processing of images, and performance of the algorithm.

2 Domain Knowledge

The main scope of our project is to incorporate the ESP32 Cam device to convert the analog readings in the water and electricity meter into a digital one. The ESP32 cam will detect the last numerical digit from the electricity meter and convert it into a digital one.

2.1 Tiny ML

TinyML is a field of study which lies in the Machine Learning, Deep Learning and Embedded Systems which explores the various types of models that can run on relatively compact and energy efficient devices such as microcontrollers.[Aru20] The TinyML is capable of running both machine learning as well as deep learning models in resource constrained devices which are of small size and consumes low power of few milliwatt or lesser. Due to the limited memory available on these devices, large machine learning models are optimised and reduced in size.

The TinyML provides a number of advantages over the traditional machine learning like low-latency, low bandwidth and low power consumption while still having the functionality to be transformed into a large machine learning model. It also eliminates the privacy limitations of traditional Machine Learning as the model runs on edge devices which does not store your data in any servers.[Aru20]

2.2 TensorFlow/ TensorFlow Lite/ Tensor Flow Micro

TensorFlow is an open-source software library which was developed by Google to build and train machine learning models with deep neural networks. TensorFlow Lite is a lighter version of the TensorFlow. TensorFlow Lite is also an open-source deep learning framework which is specifically designed for on-device computing in edge devices. TensorFlow Lite enables the developers to run their trained models on edge devices, computers and mobiles. TensorFlow Lite provides the ability to perform predictions on an already trained model.[Boe22]

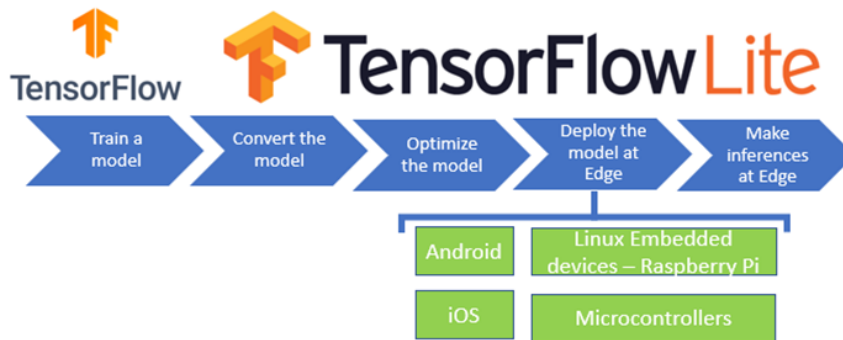


Figure 2.1: Tensor Flow Lite Workflow
Reference: [Kha20]

TensorFlow Lite Microcontrollers (TFLM) is designed to run machine learning models and deep learning models on microcontrollers with only a few kilobytes of memory. TFLM tackles both the efficiency requirements imposed by embedded-system resource constraints as well as the fragmentation challenges and makes cross-platform interoperability possible. The TFLM framework adopts a unique interpreter-based approach that provides flexibility while overcoming the forementioned challenges.[RJA+21]

2.3 Algorithm: Convolutional Neural Network

Convolutional Neural Network (CNN) most common use is as a type of Deep Learning model which takes in an input picture and then assign importance (biases and learnable weights) to various features that is present in the image, and can differentiate one feature from the other features.

The primary task of a CNN is to compress the picture into a format which is comparatively much easier to process than the full-size image. This is done while preserving crucial elements which are required for obtaining a fair prediction. The CNN enables the architecture to be competent of learning features and scalable to bigger datasets.
[Sah18]

The convolution neural network consists of three important layers which are as follows

- Convolution Layer
- Pooling Layer
- Fully-connected Layer

2.3 Algorithm: Convolutional Neural Network

The complexity of the CNN increases with each layer as it identifies larger portions of image. The beginning layers focus on simple features like edges, borders and colors. With the progression of image data through the different layers of CNN, the CNN starts to identify larger shapes and elements present in the object. This stops when the CNN identifies the intended object. [Edu20]

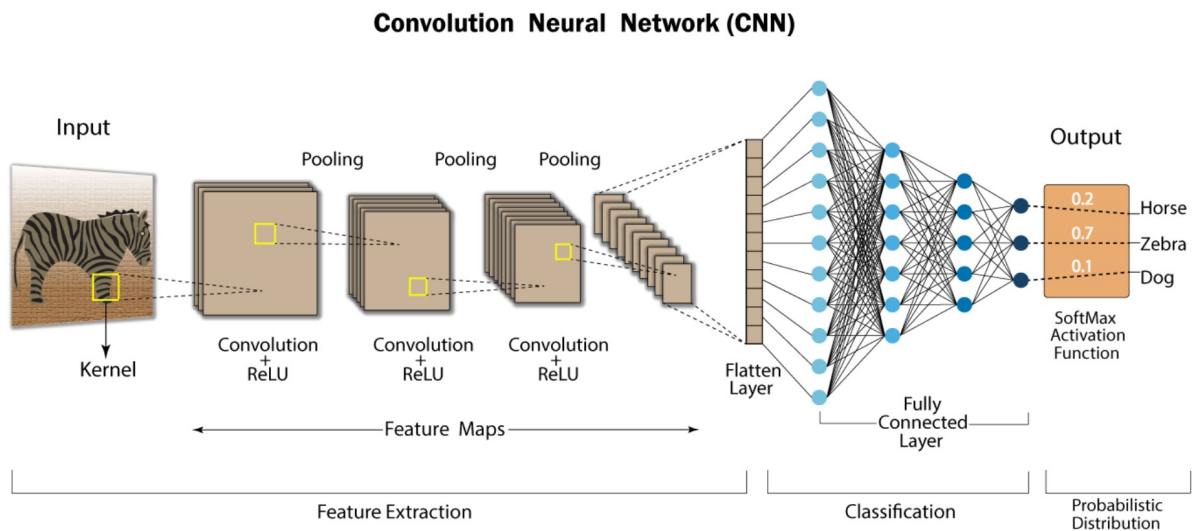


Figure 2.2: A CNN sequence to classify handwritten digits
Reference: [Swa22]

The major part of the computation takes place at this layer. The components of the convolution layer are the input data, filters and feature map. The filter is also known as kernel or a feature detector. The feature detector moves through the recurring fields of the picture, while checking if the feature is present. This entire process is termed as convolution.

The Kernel is a two-dimensional array of biases or weights, which represents a part of the picture. The kernel can be of different sizes. This size of the kernel influences the size of receptive field. A dot product is computed between the kernel and the filter. This value of the dot product is then provided into an output array. Then the kernel moves to the next stride, this process is repeated until the kernel moves across the full picture. The final output is termed as a convolved feature or a feature map. The initial feature map only captures Low-Level information like the colour gradient and edges. The progressing feature maps captures high-level information which results in the network that helps classification of individual features present in the image. [Edu20]

2.3.1 Pooling Layer

Pooling conducts a dimensionality reduction of the feature map. This is done by reducing the number of input parameters. Here the CNN retains the important features of the map which is required for the classification. This process is termed as down sampling.

In the pooling operation a filter without any weights is used to sweep across the whole input. Here an aggregation function is applied to the values of the receptive field by the kernel. The values of the aggregation function are used to populate the output array. [Edu20]

- **Max Pooling:** As the kernel is swept through the input, the kernel identifies the pixel with the highest value and transfers it into the output array. This helps in retaining the most important features present in the feature map.
- **Average Pooling:** As the kernel is swept through the input, it computes the average value of the receptive field and transfers it into the output array.

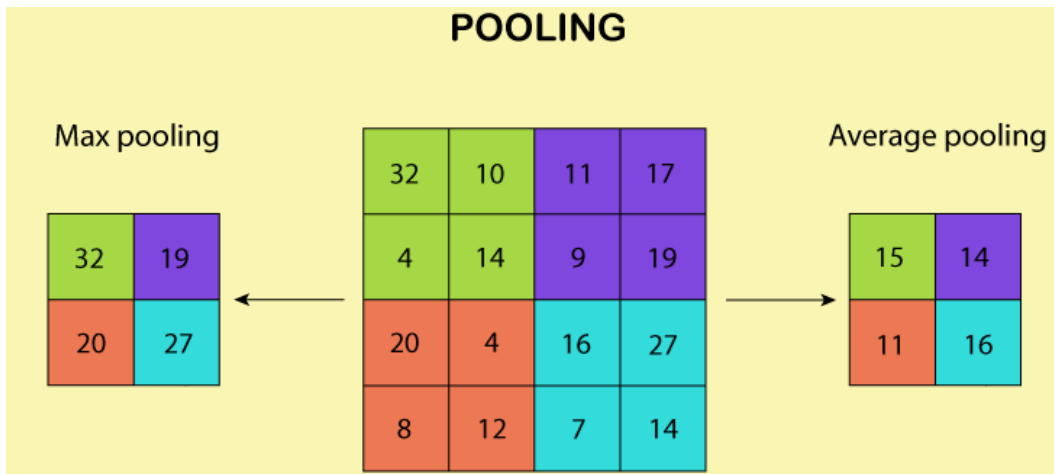


Figure 2.3: Pooling Layer in CNN
Reference: [Swa22]

Pooling has benefits such as, limiting the risk of overfitting, size reduction, noise suppression and reduction in complexity of the feature map. These result in improved efficiency of the feature map.

2.3.2 Fully-Connected Layer

The fully-connected layer is the final layer of the CNN. Here each node of the output layer directly connects to a node of the previous layer. The classification process

carried out in this layer based upon the previous layers and their appropriate filters. A Soft-Max function assigns decimal probabilities from 0 to 1 based on the appropriate classification of the inputs. [Edu20]

2.3.3 CNN Example with code

In the following section, the required steps to train a CNN, will be covered. There are six large sections of the code that need to be taken into account, each one of them with its own characteristics and functions. Also, the inputs and output from each code section needs to be taken into account as a priority to understand the codes functionality.

The main sections to create and train the CNN are the following:

1. Preparation (loading the libraries and settings).
2. Loading the training images.
3. Generate training data and test data.
4. Definition and structure of the network.
5. Train the model.
6. Storing the neural network.

Now with this sections clear, it is time to jump into the coding implementation. For this it is provided the following example, including comments thru out the different code lines to better understand the purpose:

CNN_Training

November 28, 2022

0.1 Preparation

0.1.1 Libraries

It is required to import different libraries for file and image processing, as well as for the definition and construction of neural networks using the Tensorflow library. Therefore the first step is to load all these libraries into the program:

```
[1]: import matplotlib.pyplot as plt
import glob
import os
from PIL import Image
import numpy as np
from sklearn.utils import shuffle

import tensorflow as tf
from tensorflow.keras.layers import BatchNormalization
from tensorflow.python.keras.layers import Dense, InputLayer, Conv2D,
    MaxPool2D, Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

0.1.2 Model name

Next it is important to define the developed model in order to distinguish between different versions of the neural network. It is necessary to assign a unique name for the model before start running its training or testing. This name should be identical at the beginning of each script so that the same data is used consistently.

```
[2]: ModelNameAndVersion = "counter_det"
```

0.2 Loading the training images

For this CNN, as we stated at the beginning of this section it is required to provide an input of images, that will be used to train and test the model. These images are loaded from a local directory; for this matter, it is used in this directory: 'cnnModelImages' - which can be found under the 'CNN' inside 'Code' section of the GitHub Repo.

For classification purposes, this input needs to be standardized for the program to understand them. Therefore it is also necessary for addition to the image (.jpg), that the file name is set properly and

according to the following nomenclature: First Character is the identifier and the rest of the name is not relevant.

Now the output of this section will be the proper classification of the images into two arrays `x_data` (image data) and `y_data` (classification):

```
[7]: #Setting the variables and initializing the arrays
Input_dir = 'cnnModelImages'
x_data = []
y_data = []

#Loading the image files in a loop over as jpeg images
files = glob.glob(Input_dir + '/*.jpg')
for active_file in files:
    img = Image.open(active_file) #Loads the image data
    data = np.array(img)
    x_data.append(data)

    File_Name = os.path.basename(active_file) # File_Name
    Classification = File_Name[0:1] # Fisrt digit will be
    if Classification == "N": # Handling special
    category = 10
    else:
        category = int(Classification)
        category_vector = tf.keras.utils.to_categorical(category, 11) # Conversion
        y_data.append(category_vector)

x_data = np.array(x_data)
y_data = np.array(y_data)

#print(x_data.shape)
#print(y_data.shape)
```

0.3 Generate training data and test data

0.3.1 Merge and slice the training

Once the data set is ready to be transferred to the model it is important to implement some functions that will allow the model to be properly fit. The first thing is to mix the loaded images, this is done with the “shuffle” function. It is important in this step to make sure that both ‘x_data’ and ‘y_data’ allocation in the array is preserved since this will allow the model to distinguish the number of images.

With the data properly mixed it is time to generate the output for this section of the code, one of the key sections, for the model. The image data is going to be divided into output sets of data: -

Training data (x_train and y_train) - Test data (x_test and y_test)

Important to take into account that in this case 80% of the data will be used for training and the remaining 20% is going to be used exclusively for evaluation of the model performance.

```
[9]: x_data, y_data = shuffle(x_data, y_data)

Testing_Percentage = 0.2      # Define the % of the total data to used for
    ↪testing

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data,
    ↪test_size=Testing_Percentage)
#print(x_train.shape)
#print(x_test.shape)

#print(y_train.shape)
#print(y_test.shape)
```

0.3.2 Image augmentation

Some data transformation is required for the input images to be properly handled. For this specific action, there will not be much data, since this will be covered in the coming 'Development' Section of this same report. ImageDataGenerator function takes from the first definition section which random modifications to the input images will be implemented over the various parameters (shift, brightness, zoom, rotation) and then outputs them. A batch size of 4 is defined and used here, this is chosen since it has proven to be very effective when creating a network that will work with geometries. The generator is applied to both the training and test data.

```
[10]: ### Without Image Augmentation - This section is just as reference for the
    ↪original value of the parameters
# Shift_Range = 0
# Brightness_Range = 0
# Rotation_Angle = 0
# ZoomRange = 0
```

```
[11]: ### With Image Augmentation
Shift_Range = 2
Brightness_Range = 0.3
Rotation_Angle = 5
ZoomRange = 0.2

datagen = ImageDataGenerator(width_shift_range = [-Shift_Range, Shift_Range],
                             height_shift_range = [-Shift_Range, Shift_Range],
                             brightness_range = [1-Brightness_Range,
    ↪1+Brightness_Range],
                             zoom_range = [1-ZoomRange, 1+ZoomRange],
                             rotation_range = Rotation_Angle)

Batch_Size = 4
```

```
train_iterator      = datagen.flow(x_train, y_train, batch_size=Batch_Size)
validation_iterator = datagen.flow(x_test,  y_test,  batch_size=Batch_Size)
```

0.4 Definition and structure of the network

It is now the moment to identify the specifications of the implemented network. In this case, the layout consists of a sequence of convolutional and maxpooling layers. 1- The first layer is used to normalize the input data. 2- The second layer is a “flat” layer with 512 neurons.

0.4.1 Inputs

Images of size 32x20 pixels with 3 color channels → input_shape = (32,20,3)

0.4.2 Output

11 neurons: Digits 0, 1, ..., 9 + Not-A-Number (N/A) as a special case.

0.4.3 Compile

Finally, the network is compiled here so that it can be used. The detailed parameters for this can be found in the relevant literature

```
[12]: model = tf.keras.Sequential()

model.add(BatchNormalization(input_shape=(32,20,3)))
model.add(Conv2D(16, (3, 3), padding='same', activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(32, (3, 3), padding='same', activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(16, (3, 3), padding='same', activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(128,activation="relu"))
model.add(Dense(11, activation = "softmax"))

model.summary()

model.compile(loss= tf.keras.losses.categorical_crossentropy,
              optimizer= tf.keras.optimizers.Adadelta(learning_rate=1.0, rho=0.
↪95),
              metrics = ["accuracy"])
```

Model: "sequential"

Layer (type)	Output Shape	Param #
batch_normalization (Batch Normalization)	(None, 32, 20, 3)	12

module_wrapper (ModuleWrapp er)	(None, 32, 20, 16)	448
module_wrapper_1 (ModuleWra pper)	(None, 16, 10, 16)	0
module_wrapper_2 (ModuleWra pper)	(None, 16, 10, 32)	4640
module_wrapper_3 (ModuleWra pper)	(None, 8, 5, 32)	0
module_wrapper_4 (ModuleWra pper)	(None, 8, 5, 16)	4624
module_wrapper_5 (ModuleWra pper)	(None, 4, 2, 16)	0
module_wrapper_6 (ModuleWra pper)	(None, 128)	0
module_wrapper_7 (ModuleWra pper)	(None, 128)	16512
module_wrapper_8 (ModuleWra pper)	(None, 11)	1419

```
=====
Total params: 27,655
Trainable params: 27,649
Non-trainable params: 6
-----
```

0.5 Training the network

For training one of the more important concepts to take into account is the number of training cycles (**Epoch_Number**). And here also comes into play the size of the simultaneously trained images, which was previously defined as 4 (**Batch_Size**)

With these two factor the model will run over the total set of training data and will fit the model accordingly to best match the provided input.

```
[13]: Epoch_Number = 50
      history = model.fit(train_iterator,
                          validation_data = validation_iterator,
                          epochs          = Epoch_Number)
```

Epoch 1/50

98/98 [=====] - 3s 18ms/step - loss: 2.1443 - accuracy:

0.4158 - val_loss: 2.0649 - val_accuracy: 0.3469
 Epoch 2/50
 98/98 [=====] - 1s 13ms/step - loss: 1.9682 - accuracy:
 0.4158 - val_loss: 1.9113 - val_accuracy: 0.3571
 Epoch 3/50
 98/98 [=====] - 1s 13ms/step - loss: 1.7875 - accuracy:
 0.4413 - val_loss: 1.8866 - val_accuracy: 0.3673
 Epoch 4/50
 98/98 [=====] - 1s 13ms/step - loss: 1.5501 - accuracy:
 0.5051 - val_loss: 1.5655 - val_accuracy: 0.5000
 Epoch 5/50
 98/98 [=====] - 1s 13ms/step - loss: 1.5007 - accuracy:
 0.5204 - val_loss: 1.5421 - val_accuracy: 0.4796
 Epoch 6/50
 98/98 [=====] - 1s 13ms/step - loss: 1.2596 - accuracy:
 0.5893 - val_loss: 1.3638 - val_accuracy: 0.5102
 Epoch 7/50
 98/98 [=====] - 1s 13ms/step - loss: 1.1482 - accuracy:
 0.6352 - val_loss: 1.1886 - val_accuracy: 0.6327
 Epoch 8/50
 98/98 [=====] - 1s 12ms/step - loss: 0.9293 - accuracy:
 0.7092 - val_loss: 1.1215 - val_accuracy: 0.6429
 Epoch 9/50
 98/98 [=====] - 1s 12ms/step - loss: 0.8761 - accuracy:
 0.7015 - val_loss: 1.0848 - val_accuracy: 0.6633
 Epoch 10/50
 98/98 [=====] - 1s 13ms/step - loss: 0.7541 - accuracy:
 0.7474 - val_loss: 1.1371 - val_accuracy: 0.6633
 Epoch 11/50
 98/98 [=====] - 1s 13ms/step - loss: 0.7432 - accuracy:
 0.7628 - val_loss: 1.0454 - val_accuracy: 0.6633
 Epoch 12/50
 98/98 [=====] - 1s 12ms/step - loss: 0.6782 - accuracy:
 0.7551 - val_loss: 1.0191 - val_accuracy: 0.7143
 Epoch 13/50
 98/98 [=====] - 1s 12ms/step - loss: 0.6333 - accuracy:
 0.7730 - val_loss: 1.0637 - val_accuracy: 0.7143
 Epoch 14/50
 98/98 [=====] - 1s 13ms/step - loss: 0.6489 - accuracy:
 0.7908 - val_loss: 0.9205 - val_accuracy: 0.7041
 Epoch 15/50
 98/98 [=====] - 1s 13ms/step - loss: 0.5948 - accuracy:
 0.7985 - val_loss: 0.8216 - val_accuracy: 0.7245
 Epoch 16/50
 98/98 [=====] - 1s 12ms/step - loss: 0.5251 - accuracy:
 0.8087 - val_loss: 0.8158 - val_accuracy: 0.7551
 Epoch 17/50
 98/98 [=====] - 1s 12ms/step - loss: 0.6191 - accuracy:

0.7985 - val_loss: 0.7756 - val_accuracy: 0.7347
 Epoch 18/50
 98/98 [=====] - 1s 12ms/step - loss: 0.5469 - accuracy:
 0.8265 - val_loss: 0.8434 - val_accuracy: 0.7143
 Epoch 19/50
 98/98 [=====] - 1s 13ms/step - loss: 0.4843 - accuracy:
 0.8444 - val_loss: 0.8002 - val_accuracy: 0.7755
 Epoch 20/50
 98/98 [=====] - 1s 13ms/step - loss: 0.5197 - accuracy:
 0.8444 - val_loss: 0.8632 - val_accuracy: 0.7041
 Epoch 21/50
 98/98 [=====] - 1s 12ms/step - loss: 0.4779 - accuracy:
 0.8571 - val_loss: 0.8375 - val_accuracy: 0.7449
 Epoch 22/50
 98/98 [=====] - 1s 13ms/step - loss: 0.4117 - accuracy:
 0.8699 - val_loss: 0.8594 - val_accuracy: 0.7653
 Epoch 23/50
 98/98 [=====] - 1s 12ms/step - loss: 0.4668 - accuracy:
 0.8444 - val_loss: 0.9214 - val_accuracy: 0.7143
 Epoch 24/50
 98/98 [=====] - 1s 12ms/step - loss: 0.3587 - accuracy:
 0.8750 - val_loss: 0.7608 - val_accuracy: 0.7857
 Epoch 25/50
 98/98 [=====] - 1s 13ms/step - loss: 0.4094 - accuracy:
 0.8673 - val_loss: 0.7783 - val_accuracy: 0.8061
 Epoch 26/50
 98/98 [=====] - 1s 13ms/step - loss: 0.4493 - accuracy:
 0.8622 - val_loss: 0.6756 - val_accuracy: 0.8163
 Epoch 27/50
 98/98 [=====] - 1s 13ms/step - loss: 0.4481 - accuracy:
 0.8750 - val_loss: 0.7532 - val_accuracy: 0.7857
 Epoch 28/50
 98/98 [=====] - 1s 13ms/step - loss: 0.3530 - accuracy:
 0.8929 - val_loss: 0.8846 - val_accuracy: 0.7143
 Epoch 29/50
 98/98 [=====] - 1s 12ms/step - loss: 0.3040 - accuracy:
 0.9005 - val_loss: 0.7868 - val_accuracy: 0.7653
 Epoch 30/50
 98/98 [=====] - 1s 13ms/step - loss: 0.3558 - accuracy:
 0.8929 - val_loss: 0.8299 - val_accuracy: 0.7755
 Epoch 31/50
 98/98 [=====] - 1s 12ms/step - loss: 0.3190 - accuracy:
 0.8929 - val_loss: 0.6582 - val_accuracy: 0.7959
 Epoch 32/50
 98/98 [=====] - 1s 13ms/step - loss: 0.2992 - accuracy:
 0.9362 - val_loss: 0.9584 - val_accuracy: 0.8163
 Epoch 33/50
 98/98 [=====] - 1s 12ms/step - loss: 0.3456 - accuracy:

0.8903 - val_loss: 0.6098 - val_accuracy: 0.8265
 Epoch 34/50
 98/98 [=====] - 1s 13ms/step - loss: 0.2796 - accuracy:
 0.9107 - val_loss: 0.6716 - val_accuracy: 0.7755
 Epoch 35/50
 98/98 [=====] - 1s 13ms/step - loss: 0.3834 - accuracy:
 0.8699 - val_loss: 0.7031 - val_accuracy: 0.7857
 Epoch 36/50
 98/98 [=====] - 1s 13ms/step - loss: 0.2575 - accuracy:
 0.9209 - val_loss: 0.8095 - val_accuracy: 0.8163
 Epoch 37/50
 98/98 [=====] - 1s 13ms/step - loss: 0.2456 - accuracy:
 0.9209 - val_loss: 0.7661 - val_accuracy: 0.8061
 Epoch 38/50
 98/98 [=====] - 1s 13ms/step - loss: 0.2745 - accuracy:
 0.9133 - val_loss: 0.9254 - val_accuracy: 0.7755
 Epoch 39/50
 98/98 [=====] - 1s 13ms/step - loss: 0.3168 - accuracy:
 0.9158 - val_loss: 0.7265 - val_accuracy: 0.7959
 Epoch 40/50
 98/98 [=====] - 1s 12ms/step - loss: 0.3356 - accuracy:
 0.9005 - val_loss: 0.7074 - val_accuracy: 0.7959
 Epoch 41/50
 98/98 [=====] - 1s 12ms/step - loss: 0.2670 - accuracy:
 0.9107 - val_loss: 0.9057 - val_accuracy: 0.7653
 Epoch 42/50
 98/98 [=====] - 1s 12ms/step - loss: 0.2628 - accuracy:
 0.9082 - val_loss: 0.5808 - val_accuracy: 0.8265
 Epoch 43/50
 98/98 [=====] - 1s 12ms/step - loss: 0.3258 - accuracy:
 0.9158 - val_loss: 0.5568 - val_accuracy: 0.7959
 Epoch 44/50
 98/98 [=====] - 1s 12ms/step - loss: 0.2320 - accuracy:
 0.9311 - val_loss: 0.5323 - val_accuracy: 0.8265
 Epoch 45/50
 98/98 [=====] - 1s 12ms/step - loss: 0.2319 - accuracy:
 0.9235 - val_loss: 0.8088 - val_accuracy: 0.8163
 Epoch 46/50
 98/98 [=====] - 1s 12ms/step - loss: 0.1818 - accuracy:
 0.9388 - val_loss: 0.5199 - val_accuracy: 0.8571
 Epoch 47/50
 98/98 [=====] - 1s 11ms/step - loss: 0.1760 - accuracy:
 0.9413 - val_loss: 0.5991 - val_accuracy: 0.8367
 Epoch 48/50
 98/98 [=====] - 1s 12ms/step - loss: 0.1834 - accuracy:
 0.9464 - val_loss: 0.7558 - val_accuracy: 0.8061
 Epoch 49/50
 98/98 [=====] - 1s 12ms/step - loss: 0.1781 - accuracy:

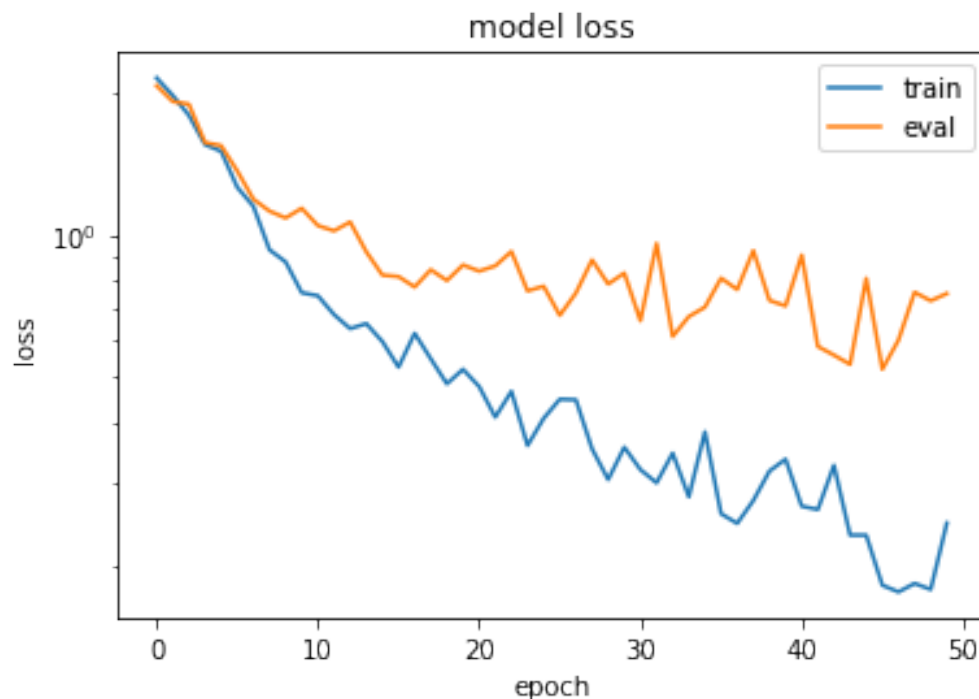
```
0.9388 - val_loss: 0.7252 - val_accuracy: 0.8367
Epoch 50/50
98/98 [=====] - 1s 12ms/step - loss: 0.2461 - accuracy:
0.9286 - val_loss: 0.7511 - val_accuracy: 0.8367
```

0.5.1 Visualization of training

The model is now ready to use. For a better understanding of the output, one can be visualized how the training was performed. For this matter, the error is used, both in the training data as well as in the test data.

```
[14]: plt.semilogy(history.history['loss'])
      plt.semilogy(history.history['val_loss'])

      plt.title('model loss')
      plt.ylabel('loss')
      plt.xlabel('epoch')
      plt.legend(['train', 'eval'], loc='upper right')
      plt.show()
```



0.6 Saving the Neural Network

Last step is to get the outputted neural network saved for further use. The complete network is first saved here in the so-called H5 format.

```
[15]: ## H5-Format  
model.save('saved_model/' + ModelNameAndVersion)
```

```
WARNING:absl:Found untraced functions such as  
conv2d_layer_call_and_return_conditional_losses, conv2d_layer_call_fn,  
conv2d_1_layer_call_and_return_conditional_losses, conv2d_1_layer_call_fn,  
conv2d_2_layer_call_and_return_conditional_losses while saving (showing 5 of  
12). These functions will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: saved_model/counter_det/assets
```

```
INFO:tensorflow:Assets written to: saved_model/counter_det/assets
```

[Mül20]

2.4 ESP32-CAM

The ESP32-CAM is a low-cost full-featured microcontroller with an ESP32-S development board along with an onboard camera module and micro-SD card port. In addition, this board has integrated WiFi and Bluetooth Low Energy (BLE), which allows it to transfer data wirelessly when required.

ESP32-CAM is readily available in different HW Distributors online with costs ranging from around 10 Euros to 20 Euros, depending on other additional gadgets included, such as Development Board, Serial Converter, and others required for larger scale projects.

The small form factor, availability and economic feasibility of the ESP32 make it a perfect solution for various IoT applications such as wireless monitoring, QR code identification, image tracking, image recognition, smart home devices, industrial wireless control and other applications.

2.4.1 Components in ESP32CAM Development Board

The ESP32-CAM development board has the following nine components:

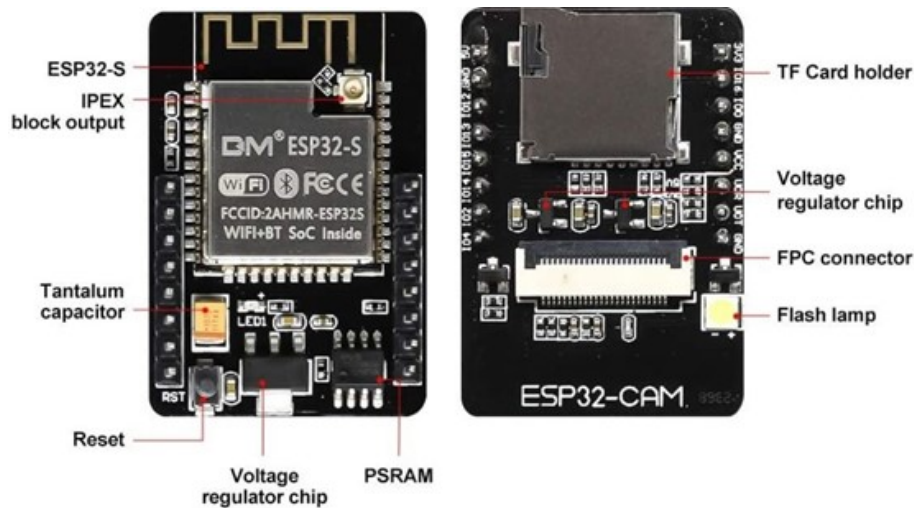


Figure 2.4: Components in ESP32-CAM
Reference: [Lab21]

1. **ESP32-S Chip:** This is the main module which houses dual core high-performance 32-bit LX6 CPUs. It computes all the processing and functioning. This

supports Wireless Fidelity (Wi-Fi) and Bluetooth 4.2

2. **IPEX block output:** The IPEX acts a connector to Global System for Mobile communication (GSM) antennas to transmit signals.
3. **Tantalum capacitor:** Tantalum capacitor is used to provide power supply filtering for fine signal quality.
4. **Reset button:** The reset button restarts the code that is executed on the module when pressed.
5. **Voltage regulator chip:** The voltage regulator chip is used to regulate the voltage to 3.3 volts. This ensures that the module maintains a constant output voltage despite the fluctuations in the input power supply.
6. **PSRAM:** This module is the Pseudo-Random-Access Memory. It consists of about 4MB memory. This enables quick processing of the instruction provided to it and assists the ESP32 camera run smoothly.
7. **TF Card Holder:** TF Card Holder houses the Micro-SD card which stores the data in ESP32. The transmission of data between card and the ESP32-S chip takes place through the Serial Peripheral Interface.
8. **FPC connector:** The Flexible Printed Circuit (FPC) connectors is used to mount the camera. The fine pitch of the FPC ensures the reliability of signal.
9. **Flash Light:** The flash light module produces electric pulses which illuminates the area acts as a flash for the camera in order to take clear pictures.

2.4.2 ESP32-CAM Microcontroller Features

The ESP32-CAM microcontroller has the ESP32-S microprocessor module. The following are the data processing and hardware capabilities.

- Has 802.11b/g/n Wi-Fi SoC Module with a speed of 2.4 GHz
- Has Bluetooth 4.2 with BLE

2 Domain Knowledge

- Has a Low-power dual-core 32-bit CPU for application processing
- Clock speed up to 160 MHz
- Has a summary computing power goes up to 600 DMIPS
- Has a Built-in 520 KB SRAM plus 4 MB PSRAM
- Supports UART, SPI, I2C, and PWM interfaces
- Supports Wi-Fi Image Upload
- Supports multiple sleep modes
- Firmware Over the Air (FOTA) upgrades are possible
- 9 General-Purpose Input/Output (GPIO) ports are available
- Has a Built-in Flash LED
- Support TF card
- Has an onboard PCB antenna
- Has embedded Free real-time operating system (FreeRTOS) and lightweight IP

2.4.3 ESP32-S Microprocessor Specification

The following are the specification of the ESP32-CAM:

- SPI Flash: default 32Mbit
- RAM: built-in 520 KB + external 4MPSRAM
- Dimension: 27*40.5*4.5(± 0.2)mm1.06*1.59*0.18"
- Bluetooth: Bluetooth 4.2 BR/EDR and BLE standards
- Wi-Fi: 802.11b/g/n/e/i
- Support Interface: UART, SPI, I2C, PWM
- Support TF card: maximum support 4G
- IO port: 9
- Serial Port Baud-rate: Default 115200 bps
- Image Output Format: JPEG(OV2640 support only), BMP, GRAYSCALE
- Spectrum Range: 2412 2484MHz

- Antenna: onboard PCB antenna, gain 2dBi
- Transmit Power: 802.11b: 17 ± 2 dBm (@11Mbps); 802.11g: 14 ± 2 dBm (@54Mbps); 802.11n: 13 ± 2 dBm (@MCS7)
- Receiving Sensitivity: CCK, 1 Mbps: -90dBm; CCK, 11 Mbps: -85dBm; 6 Mbps (1/2 BPSK): -88dBm; 54 Mbps (3/4 64-QAM): -70dBm; MCS7 (65 Mbps, 72.2 Mbps): -67dBm
- Power consumption: Flash Off: 180mA@5V Flash On and maximum brightness: 310mA@5V Deep-sleep: 6mA@5V Moderm-sleep: 20mA@5V Light-sleep: 6.7mA@5V
- Security: WPA/WPA2/WPA2-Enterprise/WPS
- Power supply range: 5V
- Operating temperature: $-20\text{ }^{\circ}\text{C}$ $85\text{ }^{\circ}\text{C}$
- Storage environment: $-40\text{ }^{\circ}\text{C}$ $90\text{ }^{\circ}\text{C}$, < 90
- Weight: 10g

2.5 ESP32-CAM Camera Module

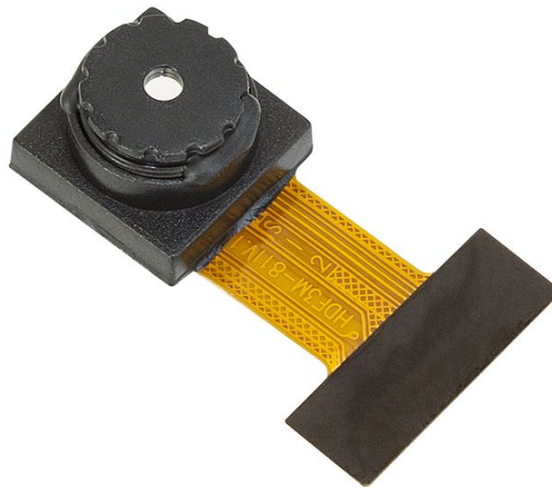


Figure 2.5: OV2640 Camera Module
Reference: [empty citation]

The ESP32 Cam uses the OV2640 camera module, which was the world's first 1/4-inch 2 MP fully-integrated image sensor. The OV2640 image sensor has some notable features such as automatic exposure control (AEC), automatic grain control (AGC), automatic white balance (AWB), automatic band filter (ABF), and automatic black-level calibration (ABLC). One of the most beneficial features of the OV2640 is its embedded compression engine which supports most common compression formats. The image sensor also allows the user to control the image quality controls such as sharpness (edge enhancement), colour saturation, white pixel canceling, noise reduction, variable frame rate and 50/60Hz luminance detection. OV2640 image sensor has high sensitivity for low-light operations and supports formats such as Raw RGB, RGB(RGB565/555), GRB22, YUV (422/420) and YCBCr (4:2:2) formats.

Array Size	UXGA	1600 X 1200
Power Supply	Core	1.3VDC
	Analog	~3.0VDC
	I/O	1.7V to 3.3V
Power Requirements	Active	125 mW (for 15 fps, UXGA YUV mode) 140 mW (for 15 fps, UXGA compressed mode)
	Standby	900 μ A
Temperature Range	Stable Image	0°C to 50°C
Output Formats (8-bit)		• YUV(422/420)/YCbCr422
		• RGB565/555
		• 8-bit compressed data
		• 8-/10-bit Raw RGB data
Lens Size		1/4"
Chief Ray Angle		25° non-linear
Maximum Image Transfer Rate	UXGA/SXGA	15 fps
	SVGA	30 fps
	CIF	60 fps
Sensitivity		0.6 V/Lux-sec
S/N Ratio		40 dB
Dynamic Range		50 dB
Scan Mode		Progressive
Maximum Exposure Interval		1247 x tROW
Gamma Correction		Programmable
Pixel Size		2.2 μ m x 2.2 μ m
Dark Current		15 mV/s at 60°C
Well Capacity		12 Ke
Fixed Pattern Noise		<1% of VPEAK-TO-PEAK
Image Area		3590 μ m x 2684 μ m
Package Dimensions		5725 μ m x 6285 μ m

OV2640 Camera Module Specification

2.6 Visual Studio Code

The acsw tool to be used as editor in the project development is one of the key aspects to consider, in order to set the project's background in place. There are plenty of options to consider for this matter, however taking into account the projects capabilities and constraints, there was one clear option for the implementation. Visual Studio Code (VSCode) is a free editor from Microsoft, which has the enormous advantage of having a plug-in specialized for all products of the manufacturer Espressif [Sys22], to be programmed and flashed.

2 Domain Knowledge

In this section, will be presented the different stages of the implementation, from basic installation, to basic initial test on the Software (SW) and Hardware (HW) communication.

2.6.1 Installation and setup

The first thing to make sure is that everything needed to start the project with VSCode is properly installed and ready to go. Next it is going to be presented and overview to go over the entire setup process, and explain every step to get things working properly.

Even if VSCode is already installed on the machine, to give quick glance over this section is crucial, in order to make sure that the same direction is being followed before starting the project.

Obvious to mention, but also it is clear set that if it is the first time installing VSCode, it is highly recommend completing all of the steps in this section. With no more to add. Let's jump in!

1. Download the executable file.

- You may access the file from the link bellow.

Download Link: <https://code.visualstudio.com/download>

2. Click the option Download.

- Select the option that fits the operating system of the machine where VSCode will be installed. (e.g. Windows)
- VSCode version 1.73 will be used among this report.

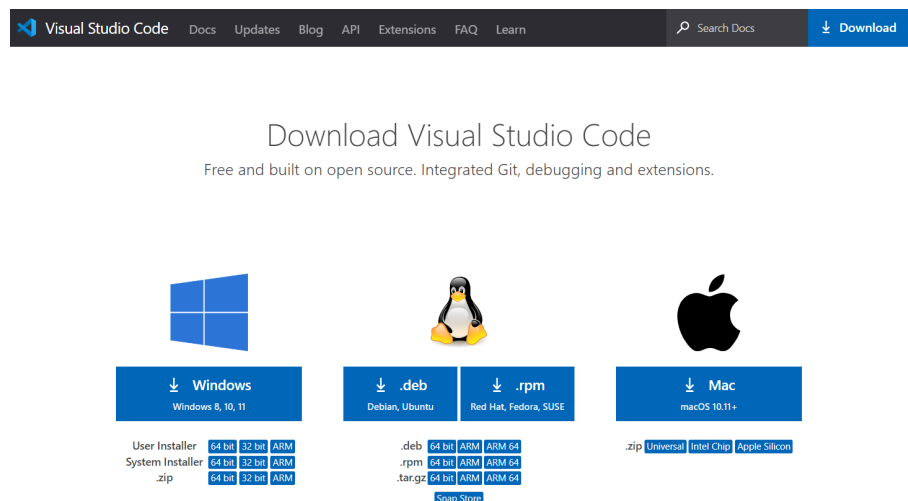


Figure 2.6: Visual Studio Code Download Options.

Reference: [Mic22]

3. Double click the downloaded file.

- File is stored under the name: VSCodeUserSetup-x64-1.73.0.exe
- Now a dialogue box appears.
- Select "I accept the agreement".
- Select "Next >".

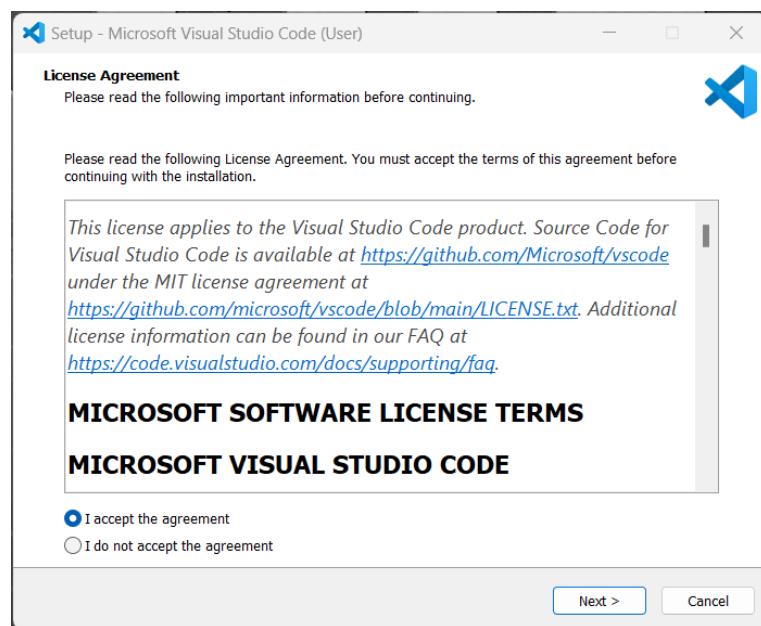


Figure 2.7: Visual Studio Code License Agreements.

4. Select a folder by clicking Browse or just follow the default path.

- Please note that 348.3 MB are required to be free on your device to complete the installation.
- Select "Next >".

Recommendation: Leave Default path.

2 Domain Knowledge

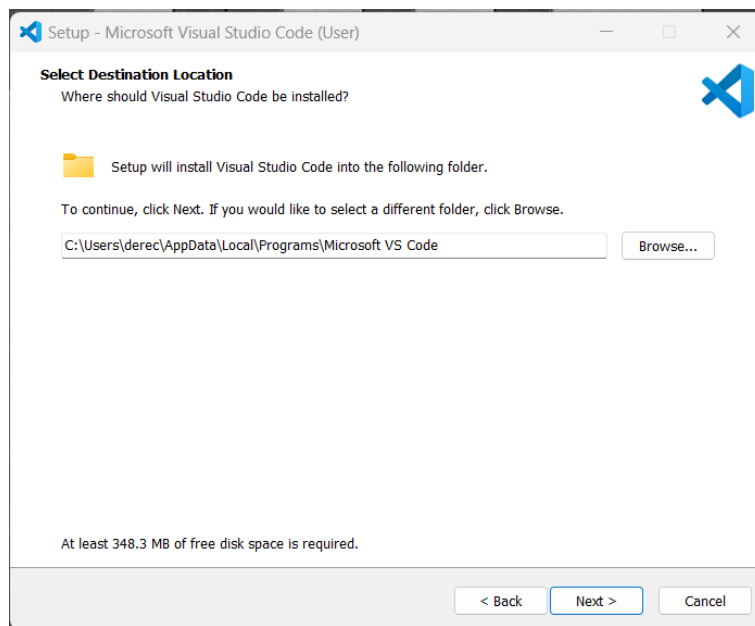


Figure 2.8: Visual Studio Code Destination Location.

5. Select the Start Menu Folder.

Recommendation: Leave Default path.

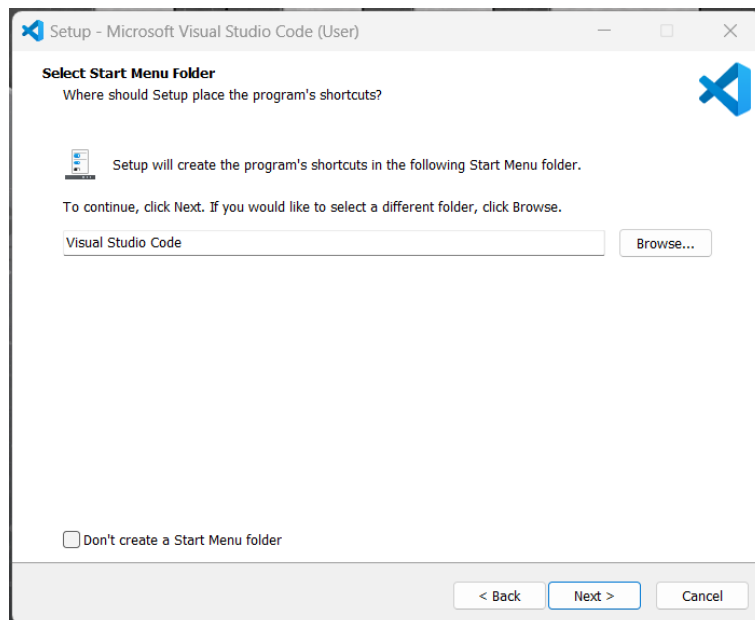


Figure 2.9: Visual Studio Code Start Menu Folder.

6. Select Additional Tasks.

Recommendation: Leave Default path.

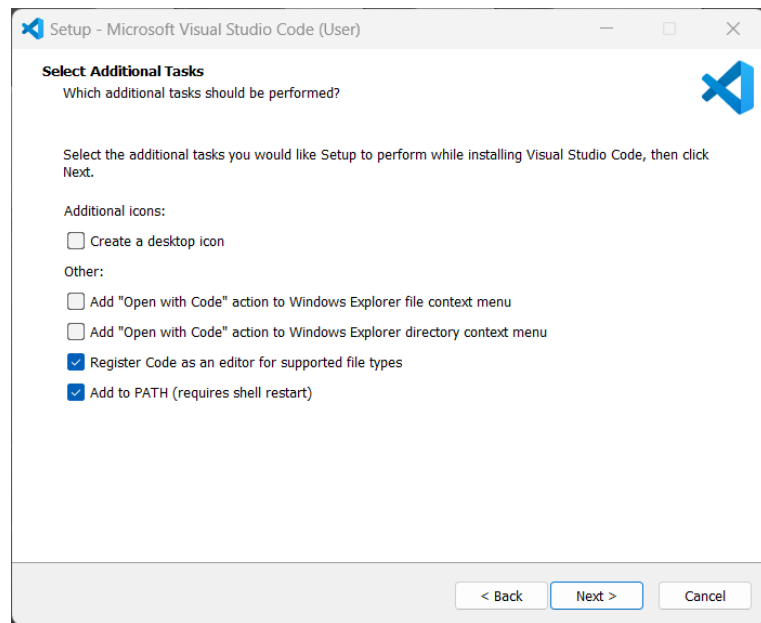


Figure 2.10: Visual Studio Code Additional Tasks.

7. Select Install.

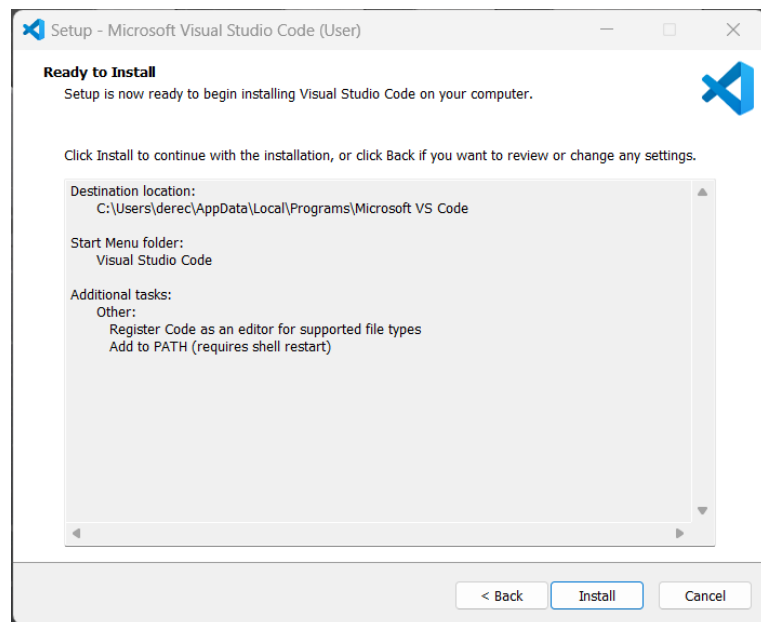


Figure 2.11: Visual Studio Code Install.

2 Domain Knowledge

8. Click Finish to exit Setup. Check in the check box to launch VSCode right now.

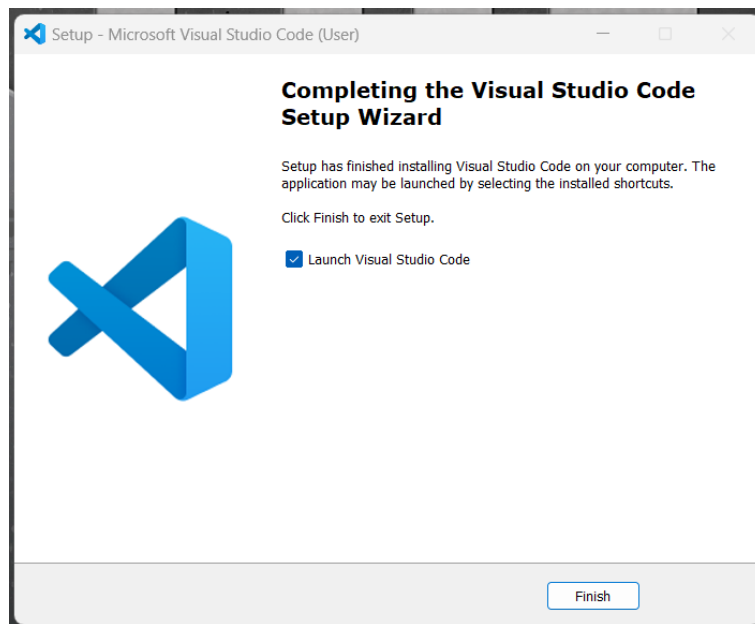


Figure 2.12: Visual Studio Code Finished Installation.

9. Open Visual Studio Code and select "New File" to create a new file as a test.

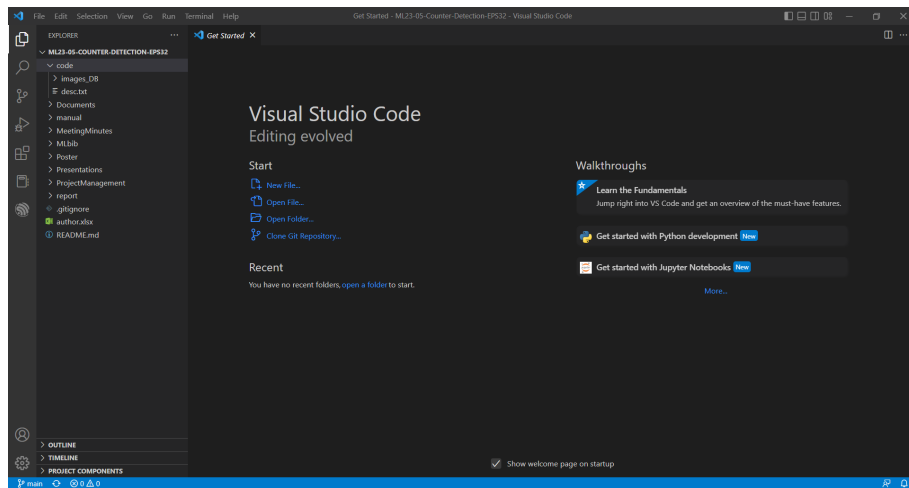


Figure 2.13: Visual Studio Code Initial Window.

10. For this test, select Python as development environment.
 - In the new window write a basic Hello World, print function

- Use this code line: `print("Hello ESP32! :)")`

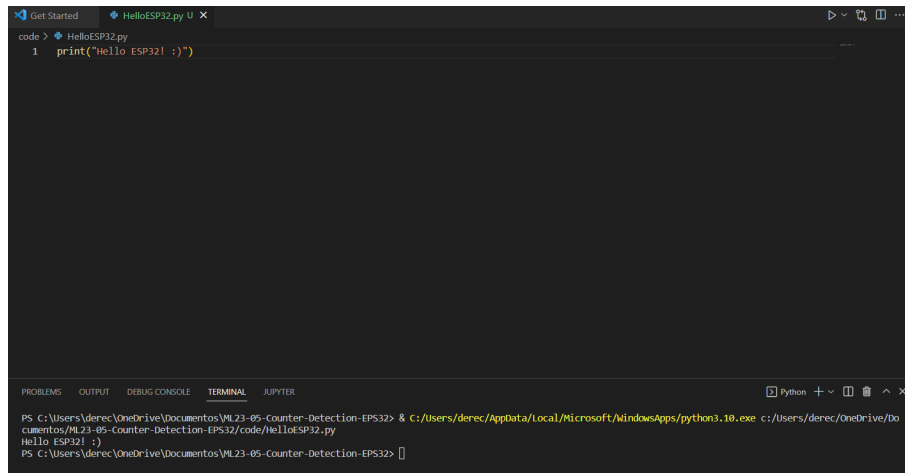


Figure 2.14: Visual Studio Code "Hello ESP32 Basic Code."

Congratulations. VSCode is properly installed and ready to use.

2.6.2 ESP-IDF Plug-in

ESP-IDF is the official development framework for the ESP32 family chips, in VSCode. This tool is recommended over some other options, such as Arduino IDE, because of the capability to offer more powerful applications. With this versatile plug-in it is possible to develop, build, flash, monitor and debug the code loaded in to ESP32 Devices. For this development is recommended to use version v4.4.3 (latest available), next it will be presented the setup process to get everything ready.

1. Open VSCode.
2. Open the Extensions view.
3. Search the extension: "esp-idf"
4. Select Install option on v1.5.1.

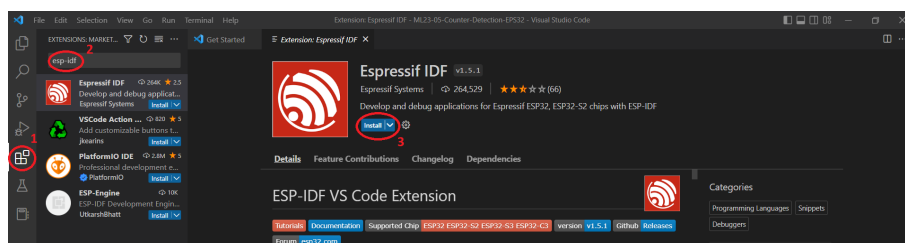


Figure 2.15: Visual Studio Code Install Extension "esp-df".

2 Domain Knowledge

5. Select the ESP-IDF: Configure ESP-IDF extension option.
6. Choose "Express", as the best suited option for the project.
7. Select Install option.

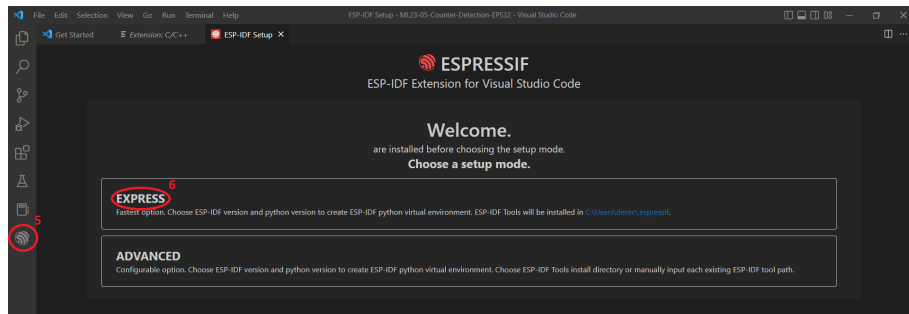


Figure 2.16: Configure ESP-IDF extension: Express.

8. Select Download Server: "Github".
9. Select an ESP-IDF version to download (v4.4.3).

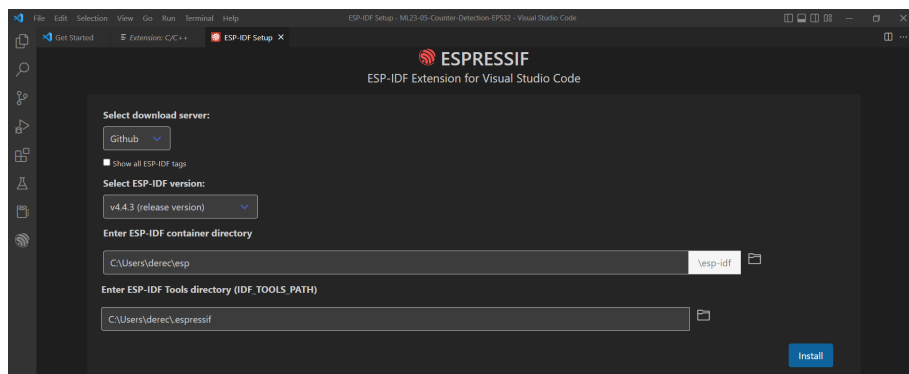


Figure 2.17: Download ESP-IDF version from: GitHub.

Once the Plug-In is properly installed in VSCode, a success message will appear in the screen. [Ign22]:

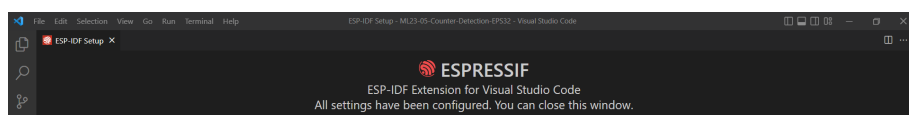
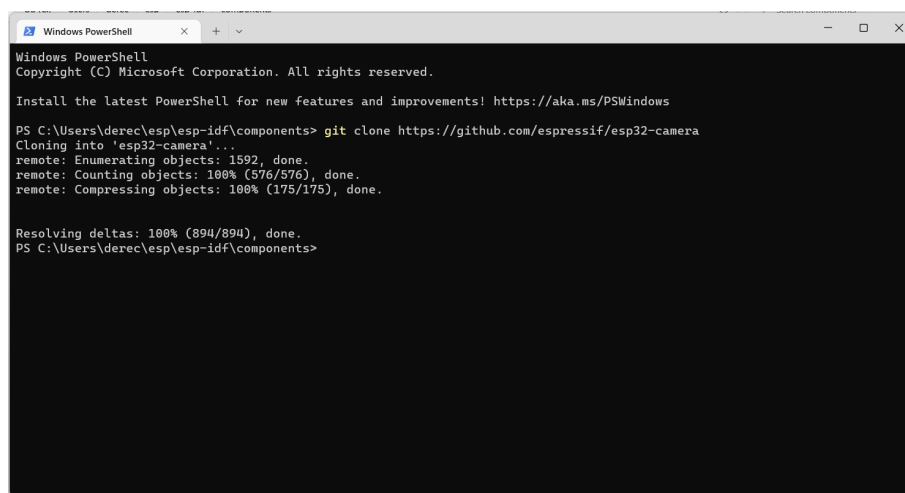


Figure 2.18: Successfully installed ESP-IDF.

It is now time to to install the ESP32-CAM drivers.

The first thing to do is to clone or download and extract the repository [ESP32 Camera GitHub project] to the components folder of your ESP-IDF. To do this, continue with the following steps:

10. In your Directory Navigator, go to the "ESP-IDF" pre-installed folder and look for the "components" folder. (i.e. "C:/Users/dereck/esp/esp-idf/components")
11. Open Windows PowerShell.
12. Run: `git clone https://github.com/espressif/esp32-camera`



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\dereck\esp\esp-idf\components> git clone https://github.com/espressif/esp32-camera
Cloning into 'esp32-camera'...
remote: Enumerating objects: 1592, done.
remote: Counting objects: 100% (576/576), done.
remote: Compressing objects: 100% (175/175), done.

Resolving deltas: 100% (894/894), done.
PS C:\Users\dereck\esp\esp-idf\components>
  
```

Figure 2.19: ESP32 Camera Clone Repository.

Now all the required drivers for the ESP32-CAM are installed and ready to be used in VSCode.

2.6.3 Microprocessor Test

It is now time to test basic functionalities of the microprocessor. This will be done by loading a "Hello World" function to the HW. For this implementation the goal is to program ESP32-S microprocessor to turn on the LED integrated in the ESP32-CAM module. To achieve this, the code will upload a basic algorithm to the ESP32-CAM module. This algorithms are part of the basic test that come along with the ESP-IDF installation. Therefore the code can be found in the installation directory of this plug-in, un de this path: "examples/get-started/blink", which is relative to the installation path (i.e. C:/Users/dereck/esp/esp-idf/examples/get-started/blink).

This is the code's structure:

2 Domain Knowledge

- **Include Section:** This section includes header files that are required to run specific modules on the code. These header's location have to be defined as a path in order to avoid "building errors". This definition is done under this file: ".vscode/c_cpp_properties.json". In particular for the this test we need the predefined path for the ESP-IDF installation (i.e. C:/Users/dereck/esp/esp-idf).
- **Physical Resource Definition Section:** This section is used to indicate the code what GPIO pins must interact during the code execution. In this simple case, the code will run only on the GPIO pin dedicated for the LED on board. This GPIO pin is defined as: "define BLINK_GPIO 33".
- **Functions Section:** This section includes all the sub-functions that are going to be executed during the code. In this case we have only two functions: "configure_led", this is used to initialize the LED condition. "blink_led", this is used to change the LED state accordingly: HIGH or LOW.
- **Main Section:** This is the principal function to be executed and this section of the code will summon accordingly the pre-defined functions.

Once the code is ready, it has to be build, meaning that VSCode will generate a project based on the original code, and "translate" this information into a machine language code to be interpreted by the ESP32-S micro controller. This is done, on the VSCode Terminal with this command: "idf.py build".

Finally, Once this is ready to built code needs to be flashed or loaded to the device (ESP32-CAM). In order to do this, it is used the VSCode Terminal with this command: "idf.py -p <PORT_NUMBER> flash monitor". <PORT_NUMBER> is defined by the USB resource of the computer used to connect the ESP32-CAM module.

This is the result of the test:

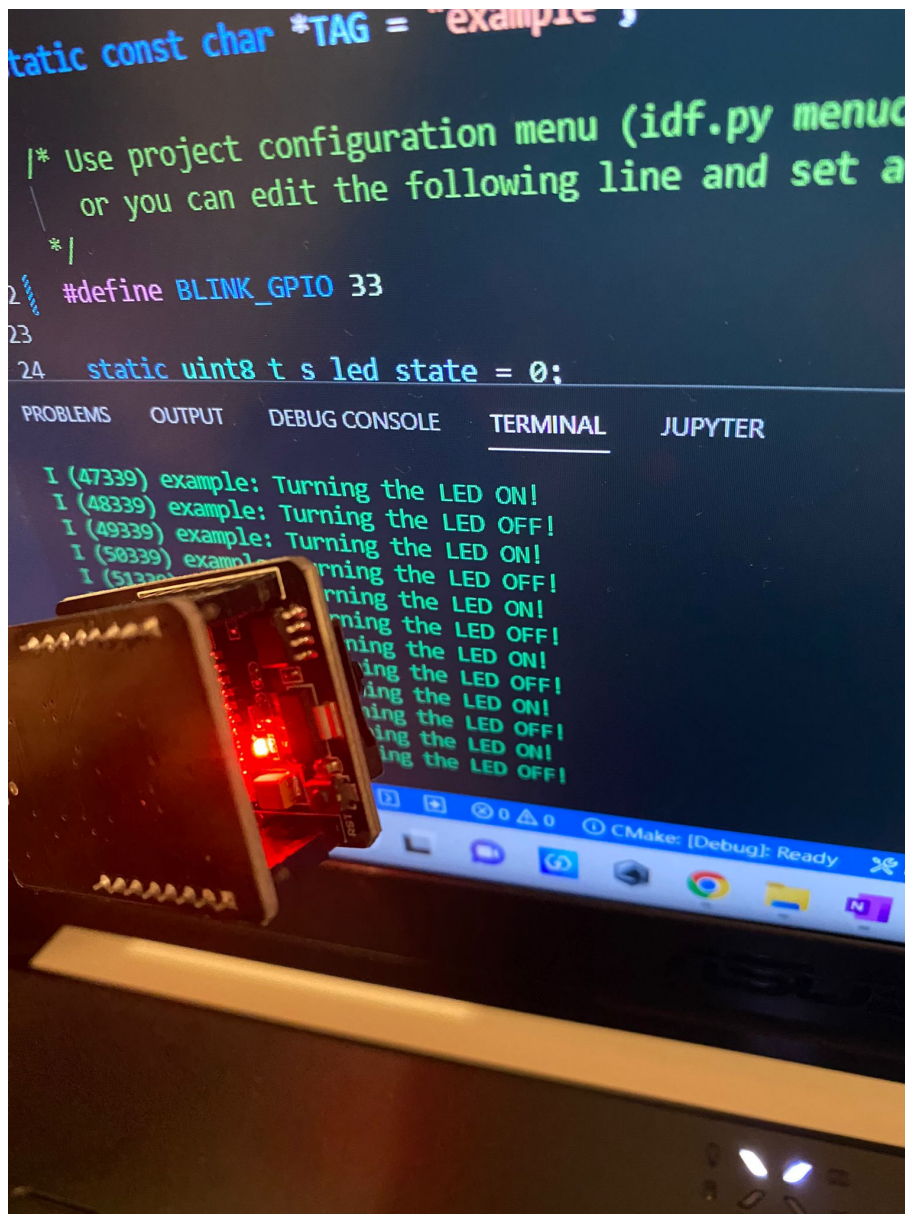


Figure 2.20: Micro processor (ESP32-S) Test in ESP32-CAM.

2.6.4 OV2640 Camera Test

The other key features that are key for the project to test, before the implementation are: OV2640 Camera Interaction with ESP32-CAM, and the SD-Card, proper functionality with the module. For this end, another basic "Hello World" Function is proposed. In this case the idea is to upload a code (following somehow the same strategy as the Microprocessor Test section) in this case with four major features:

1. Initialize the OV2640 Camera to take pictures.

2 Domain Knowledge

2. Enable the camera to take a picture every 5s.
3. Initialize the SD-Card in the ESP32-CAM to store pictures.
4. Store the taken pictures in the SD-Card.

For this example, the code will be a little more complex in order to successfully perform the required functionalities. However it will be a modular code, meaning that each function will have a separate and modular implementation, making the code easy to understand and replicate.

This is the code's structure:

- **Include Section:** Same as previous example.
- **Physical Resource Definition Section:** In this case it will be targeting the "CAM_PIN" to perform the initialization and execute the capability of capturing pictures.
- **Camera Basic Configuration:** This is a really important section to consider, since it was not present on the previous example. It allows the micro controller to specify the characteristic under which the camera will perform, by setting some basic parameters.
 1. `xclk_freq_hz` [20000000], OV2640 camera can run under 20MHz or 10MHz.
 2. `pixel_format` [GRAYSCALE], can select among these options: YUV422, GRAYSCALE, RGB565 and JPEG.
 3. `jpeg_quality` [12], a value between 0 and 63 to determine the picture quality (lower number means higher quality).
- **Functions Section:** For this example the utilized functions are basically to initialize the components, "init_camera", to initialize the camera; and "init_sdcard", to initialize the SD-Card.
- **Main Section:** This is the principal function to be executed, by executing this pre loaded feature: "esp_camera_fb_get" to capture picture. Then assigning a name to the picture as : "picture#". This # is generated by a counter, that increases after each picture.

This is the result of the test:



Figure 2.21: Camera (OV2640) and SD-Card Test in ESP32-CAM.

3 Knowledge Discovery in Databases Process

The basic definition of KDD in [FSS96] describes it as a nontrivial process of "identifying valid, novel, potentially useful, and ultimately understandable patterns in data". [FSS96] mentions that in the context of KDD, description tends to be more important than prediction. The paper also lists the detailed steps of KDD process as mentioned below:

1. Developing an understanding of the application domain and the relevant prior knowledge.
2. Selecting the target data set on which discovery is to be performed
3. Cleaning and pre processing the data to handle noise and missing data fields
4. Find useful features to represent the data depending on the goal of the task. Thenafter, use transformation methods to find more suitable representation for the data.
5. Identify appropriate methods to search for patterns in data as part of data mining.
6. Interpreting and evalutaing the results from data mining. This step can also involve data visualization of the extracted patterns/models.
7. The final step is to consolidate discovered knowledge and share with another system or report to interested parties.
8. Even after the completion of the final step we need to monitor the performance after the addition of new data and recheck the results.

The above steps can be summarised using figure 3.1 :

3 Knowledge Discovery in Databases Process

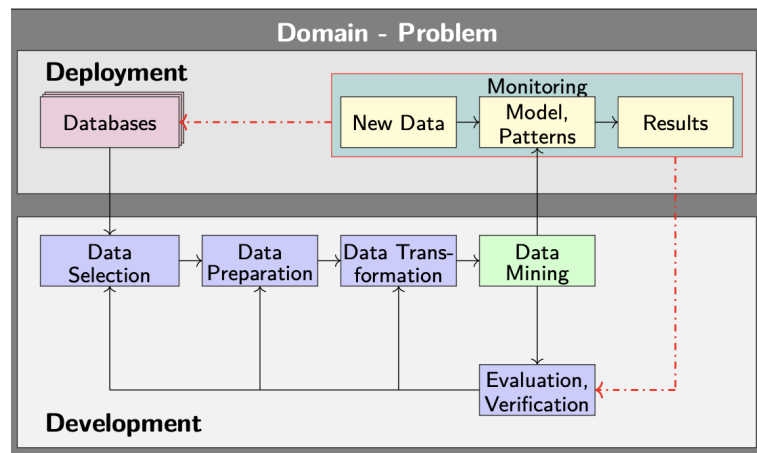


Figure 3.1: KDD process

The detailed description of how we achieved each individual step will be described in next chapter.

4 Development

In order for KDD methodology to be implemented into a feasible project, there are a set of sub process to understand and define before any real implementation, which will be specific for each project purpose. The first section will be covered in the following chapter; its intention is mainly to get deeper comprehension on the data surroundings for the project. Therefor, Database will be the fundamental aspect for discussion.

This chapter intention is to talks about how the KDD process is implemented and the steps followed for each of the KDD Processes starting with the

4.1 Database Description

The database for this project will be two folders consisting of images of different digits. Folder "pictures_original" consists of raw images, whereas folder "pictures_resize" consists of resized images. The images were copied from an existing github project.

4.2 Data Description

- Before data transformation:
Image size : 703 bytes - 8KB
Image dimensions : 32X59 - 37X67
Color space : RGB
- After data transformation:
Image size : 795 bytes - 923 bytes
Image dimensions : 20X32
Color space : RGB

4.3 Data Selection

The data from this particular project is selected because it has images of all digits from 0-9 alongwith different versions of each image. This would be higly useful during the training of the model.

4.4 Data Preparation

The data preparation includes creating various versions of the same image through data augmentation. Data augmentation is the process of generating images with

different brightness, height, width, rotation, and zoom parameters from the same original image. To achieve data augmentation ImageDataGenerator method is loaded from the TensorFlow library [Mak21a]. Concerning this project, the data preparation step simply involves placing the data in the correct folder structure.

4.5 Data Transformation

Data transformation will be required to ensure that correct data, and appropriate quality data are ingested into the model. This is important because it would impact the processing of images and the performance of the overall model. For instance, in this task resizing is required to ensure uniform input size. Ideally, it is believed that the smaller the image size faster the training and recognition process. However, if the image size is too small then the information will be lost [Mak21a]

4.6 Data Mining

4.6.1 Model Description

The model is created based on CNN. Once the network is created it needs to be loaded into variable called model. This could be easily achieved by using Keras from TensorFlow library [Mak21b].

4.6.2 Model Training

Model training can be done by using a simple command with the function `model.fit()`. This function takes the training, evaluation data and the number of epochs to be trained to this function. One epoch is exactly one revolution over all the training data. This step can take a lot of time depending on the computational power and number of images [Mak21b].

4.7 Results

(This section will be updated once the model's development is finalized)

5 Deployment

Following the KDD process, this chapter intends to explain the major components for the implementation in the ESP32-CAM module, to get a digitalized counter measurement. This chapter will mostly be substantiated by previously covered topics in the Development section of this same report. Along with these concepts, it will also approach verification of the behavior of the system.

5.1 Meter Counter Digitized

As already extensively detailed, the main objective of the project is to recognize and digitize the analog number coming from a meter. To fully understand its implication and usage surroundings it is necessary to explain how the project's scope can be translated into multiple and more complex implementations. As of now, the information will target one-digit digitization and get feedback from the system through the elemental communication scheme to the computer terminal, to get a response.

5.1.1 Software

In the following figure, important libraries are shown which will be required in building the software part of the project. The main packages are Image from Python Imaging Library (PIL) and ImageDataGenerator from TensorFlow for image processing, TensorFlow (tf) packages such as Dense, InputLayer, Conv2D, etc for building the model and sklearn library for generating and shuffling the test data.

```
import matplotlib.pyplot as plt
import glob
import os
from PIL import Image
import numpy as np
from sklearn.utils import shuffle

import tensorflow as tf
from tensorflow.python.keras.layers import Dense, InputLayer, Conv2D, MaxPool2D, Flatten, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

Figure 5.1: Important Libraries

For loading, initializing, and using NN, a wrapper class will be required which also encapsulates the TFLite components. The actual code of TFLite that is tflite-lib will be needed to be adapted as per the ESP-IDF environment. Other than these, three files CMakeLists, sdkconfig, and partitions.csv are important. CMakeLists controls the

5 Deployment

details of the compiler for all components. The file `sdkconfig` has all compiler-specific settings for ESP32. The `partitions.csv` file has a user-defined partitioning of flash memory. The methods used in the wrapper class are shown in the following figure 5.2. [Mak21c]

Methods of the Wrapper Class	
Name	Function
<code>LoadModelFromFile</code>	Load network from SD card
<code>LoadModelFromCharArray</code>	Load network from header
<code>LoadInputImage</code>	Load image from SD card
<code>Invoke</code>	Calculate the output (inferencing)
<code>GetOutClassification</code>	Output Neuron
<code>GetOutputValue</code>	Output neuron with the highest Value
<code>GetInputDimension</code>	Output input layer Size
<code>GetOutputDimension</code>	Output Layer Size

Figure 5.2: Methods in Wrapper Class

Reference: [Mak21c]

Overview of the steps to use NN A program needs to be created for using the neural network. This includes five steps namely: create an object of the wrapper class, load the model, load the image, calculate the NN and read the results.

Working with the model Before loading the model, the SD card needs to be integrated with help of functions from the Helper library, the availability of PSRAM needs to be checked and an object from the needs to be created from `CTFLiteClass` to build a NN. Thereafter, the TFLite description of the NN is loaded. The dynamic loading of a model from an SD card is preferred because of the memory management advantage. As a next step, the image is loaded from the SD card, and NN is calculated. This can take some time. In the final step, the results can be read by selecting the ESP IDF Monitor Device option. The result could be saved on an SD card or displayed in a web application. As for now, we will consider the option of saving the results on an SD card.

5.1.2 Hardware

With regards to HW, ESP32-CAM offers a lot of applications and is most commonly used on Do It Yourself (DIY) projects. This is substantiated by the variety of modules and gadgets that can be added to this system. In addition to it, as has already been discussed low power applications, take huge advantage of these devices. [BFS19]

Due to the wide variety of available sensors and applications aligned to this technology, it becomes imperative to describe how the final deliverable of the project will perform in terms of HW. For this, a specific description of the key components of the system will be provided.

HW Deployment Description

SD Card Reader	This is one of the key points for this implementation, therefore it will be detailed in detail below.
PSRAM	The ESP32-CAM provides a processing power of 8 MByte in total, however, out of this value 4 MByte can be used for processing purposes. be used effectively.
4 GByte Flash	This is an external device that will be key to complement both the SW development for the model as well as the HW limitations that come in addition to the device in question.
ESP32-CAM-MB	This is a special shield designed for easily transferring data from computer development to ESP32-CAM device.
OV2640 camera	This includes both the interface to plug the camera in the ESP32-CAM device and also the module itself, which was detailed in the Domain Knowledge of this same document.
LED Flashlight	This white LED incorporated in the board can be utilized as a flash for illuminating the meter, to improve the picture quality.

SD Card implementation and uses

Resuming the employment of both the "SD Card Reader" and the External extended memory device, these pieces of the system are key for complex implementations, such as CNN. In principle, most of the basic projects that can be developed under Internet of Things (IoT) applications and some other fields can be easily mapped on the board's default firmware, and the capability will be enough. However, when it comes to more complex designs, with a large number of processes involved, it is better to have a robust infrastructure to support the analysis performed by the model. This project's specific implementation is designed to have over 100,000 nodes, which produces internal memory to be insufficient. In the SW Deployment section of this same document is widely detailed how this module has to be included on the code to properly be recognized by the device.

The purpose of this additional memory, from the SD Card, is to store sample images, in addition to the capability to dynamically load neural networks to the system, during the program's execution. [KSP+17] This SD-Card has important details to consider, because, nothing can be so perfect. In this particular case, the limitation comes, ironically when using large storage drives (32/64 GB). This type of card can lead to a problem in the system, either when recognizing the SD Card, or when trying to access the images for the corresponding processing. This can be solved by using a 4GB (max) SD Card.

ESP-CAM-MB

To program the ESP32-CAM device it is required to meet certain HW conditions, during the information transfer. The device is to be connected to a USB port in the computer, and through a USB-to-Serial converter, it can translate the information and program the microprocessor with the required functions. To avoid unnecessary wiring of the module, there is an available shield, which is specifically designed for this purpose on the ESP32-CAM. This shield has an integrated Micro-USB port, that will be the medium for the ESP32-CAM to communicate with the computer and vice-versa. To establish communication it is only required to plug the board into the shield since the pin-out is designed to match the exact distribution of the board. Internally the ESP-CAM-MB is designed to power up the device and enable the RX and TX pins of the device accordingly.

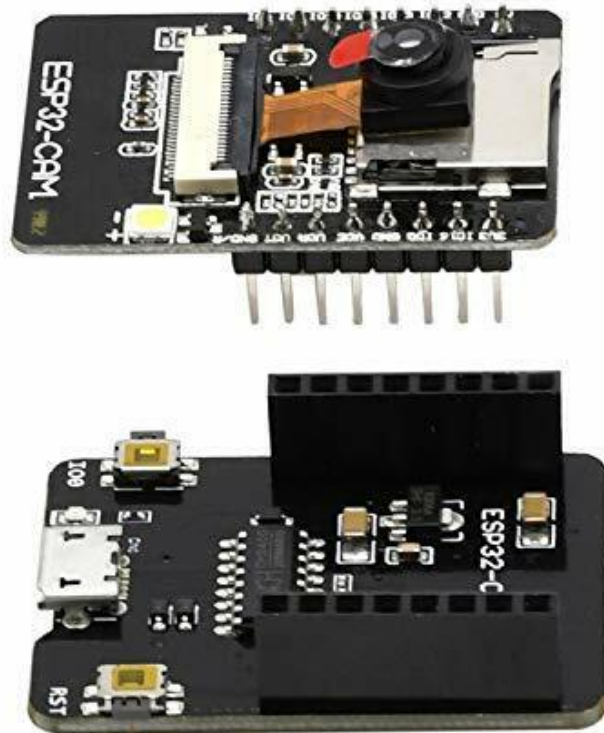


Figure 5.3: ESP32-CAM-MB connection plug to ESP32-CAM.

Once both devices are paired, just as in the above picture is presented, the next required step is to plug a USB-micro-to-USB-A cable to connect to the Personal Computer (PC) and upload the code.

5.1.3 Configuration

Once the model is fitted and properly uploaded in the ESP32-CAM, the device will run under specific environmental conditions. In field application, the environment and conditions for the project to properly perform need to be very clearly set. The first thing is to understand the surroundings where the module is placed. In this application, the ESP32-CAM will be placed in front of a printed number (taken from a real meter). The device will be connected to the Computer directly; this is possible with the already-explained Mother Board shield. This will power the device during operation, and will also link the responses for the user, among other important calibration parameters for image capturing.



Figure 5.4: ESP32-CAM field application.

The alignment is one of the key items to consider for precise image capturing, this means the correct orientation of the image. In the presented application this

feature is not automatically solved by the program. However, adequate conditions are developed to achieve it. With the help of user input and an HTTP server (website), the alignment can be performed for each digit. The website is set so that it updates itself every 2 seconds, this will allow quick response and optimization of the results. Regarding web server details, it is accessed via its IP address. To make it easy for the user, this information will be displayed in the VSCode monitoring interface.

Sample images can be found under the code section in the existing GitHub project. Testing Images will be stored in a Word file (SampleImages.docx) in the folder named: "HWTest" for its simple printout and usage for future testing. After the image is properly set for alignment, by user input, in the same simple HTTP server to display the captured images, the output from the CNN will also be displayed to the user as a resulting number.

5.1.4 Tests

It is clear that after all the previous steps, some parameters need to be defined for the project functionality. This subsection's objective is to present the result from the implementation on two fronts; HW and SW. It is important to specify this feature to understand how the system will be under corroboration in the coming steps.

General Things to Consider during Deployment Testing

The reliability of the neural network to recognize the digit increases with the increase in the training data. The neural network learns to abstract to the general rule and recognizes various aspects in the image such as contrast, brightness and other factors. For instance, if only one image per digit is used in the training of the data. And coincidentally, the image of the digit 1 is taken in a brighter setting (With better background light). Now when we train our neural network with this data, all the other digits might be recognized excellently but the other digits other than 1 can be recognized again and again. When we take a closer analysis, we can notice that this occurs whenever the background light is better. In this case our neural network did not recognize the digit 1 at all, but instead it recognizes the state of "brighter image". To avoid such issues, it is advised to have more training data with different properties of an image, such as varying brightness, position of the digits, different environments, and other factors [Mak21b] .

The raw images captured using the OV2640 camera module are of different sizes, it generally offers more resolution than what is actually needed for the neural computation. For the neural network to operate efficiently, a uniform input data variable must be provided. Generally, the size of the image directly influences the training speed of the neural network. With smaller image size, the training and recognition of the data is faster. On the other hand, there is a loss of information if the size of the image becomes too small. It is advisable to keep the images in a size, which is

5 Deployment

sufficient for the neural network to automatically recognize it [Mak21b].

Software Behavior Results

The following figure gives an overview of the activities which are required for successfully completing the project.

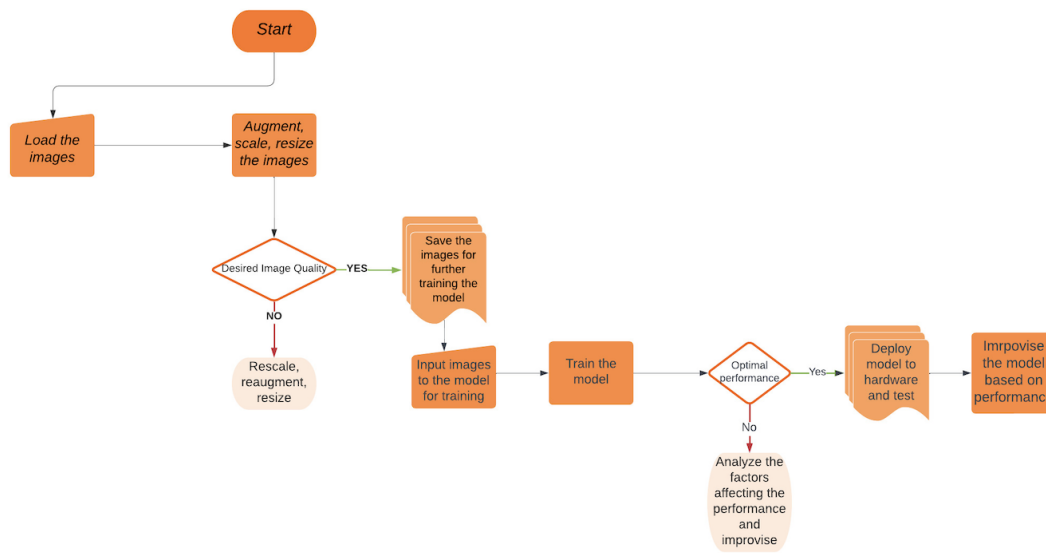


Figure 5.5: Software flowchart

Software Tests After creating Python programs for loading and processing images, it should be tested that the intended results are achieved. Also, after fitting the model, the performance of the CNN needs to be checked that whether it is returning the desired results or not. Moreover, some negative test scenarios should be tested with the help of special characters and alphabets.

Hardware Behavior Results

Once code is implemented in the ESP32-CAM module, the system is capable of identifying and providing a digital response with the corresponding number.

<Here the report will include images of the real project implementation>

5.2 Monitoring

When the developers decide to modify the data set or add fresh training data, we monitor the model's behavior to observe and record it. Or whenever due to environment changes a data base suffers modifications, it is called: Monitoring. The data set for this

project is mostly unchanged because it just includes digits and their many variations. In order to ensure that the model is robust enough since its first development, the TensorFlow library's ImageDataGenerator module is used to create several versions.

The goal of this stage is to ensure that the knowledge that has been extracted from the data is being used to make informed decisions and to improve the overall performance of the system. During this stage, the following activities are performed:

- **Performance monitoring:** This involves keeping a close eye on the effectiveness of the model created to interpret the data from the meters in order to make sure they are producing precise and pertinent findings.
- **Maintenance:** This entails maintaining the models and the database with fresh data and ensuring their functionality. In this particular case, since the data base is "numbers" then this is a very unusual case to happen, since numbers are universal.
- **Evaluation:** Finding any odd or unexpected trends in the meter data, such as sudden spikes or dips in use, is a part of this process.
- **Feedback:** This involves setting up automated alerts or notifications to let consumers know when particular criteria are satisfied, such when the use reaches a specific threshold or when a leak is discovered.
- **Continuous improvement:** To enhance the performance of the models and the whole KDD process, this entails regularly monitoring the system and making the appropriate adjustments.

The KDD process may be enhanced over time by continuously monitoring the models and the data to spot any flaws, inconsistencies, or outliers in the data and giving the system's users useful insights that enable them to make better educated decisions.

6 Conclusion / Open Questions

This report explains in detail the method and the tools that are required to digitise the analog meter readings using ESP32 Cam device. Although the model is only trained to recognise and convert 1 digit, this project can be still further developed to recognise all the digits depending on the need.

Other scope for further developing this project are:

1. Implement a mobile application with user settings and specific account management.
2. A Log-In module that can track users actions in the application, and responses from the counter.
3. A more robust HW implementation where the counter detection can be located easily in front of the meter. 6.1

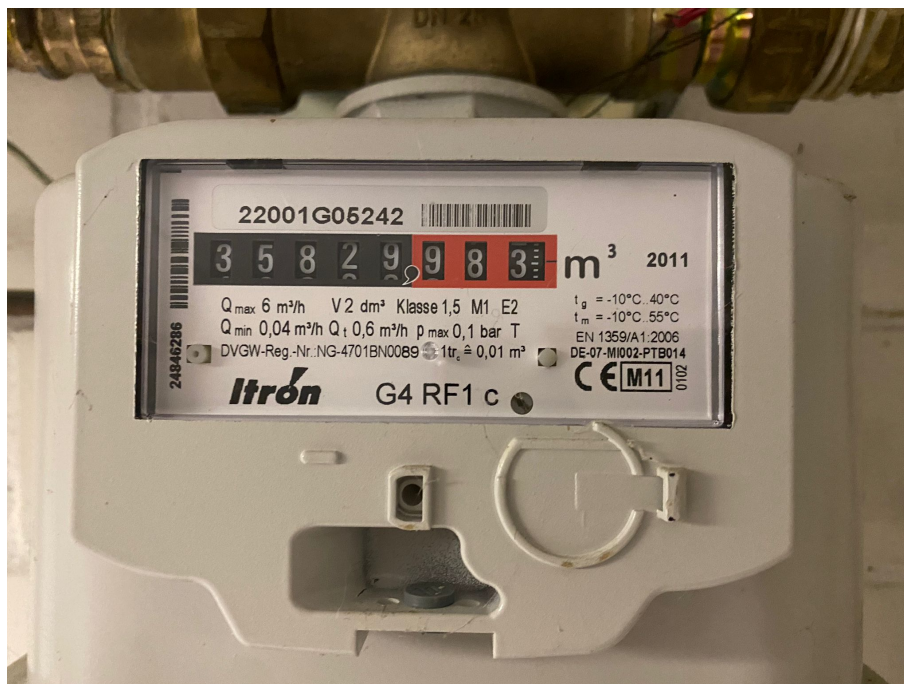


Figure 6.1: Water Meter

7 Appendix

In this section it is included additional information, relevant for the project development among other technical, financial and theoretical explanations related to the Digitization of Meter's Digits, with an ESP32-CAM.

7.1 Material List

HW List of materials required for the project:

- **ESP32-CAM AI-Thinker**
 - Description: ESP32-CAM is a low-cost ESP32-based development board with onboard camera (OV2640), small in size. It is an ideal solution for IoT application, prototypes constructions and DIY projects. The board integrates WiFi, traditional Bluetooth and low power BLE , with 2 high-performance 32-bit LX6 CPUs.
 - Link Datasheet: https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf
 - Cost: Approximately 10 Euro.



Figure 7.1: ESP32-CAM.
Reference: [Sto22]

- **OV2640 Camera**

- Description: OV2640 Camera is a 2 Megapixel OV2640 camera module with an f3.6mm lens. It contains a 24pin FPC interface. A wide-angle lens and 1632 x 1232 high resolution make it a perfect camera for Grove AI HAT for Edge Computing and other Sipeed serial board.
- Link Datasheet: https://www.uctronics.com/download/cam_module/OV2640DS.pdf
- Cost: Approximately 5 Euro.

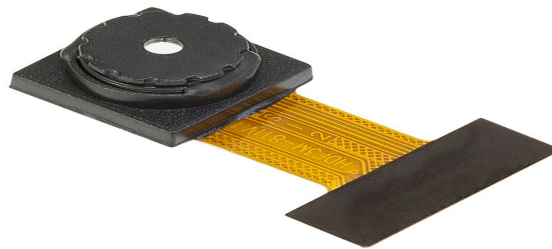


Figure 7.2: OV2640 Camera.
Reference: [Ard22]

- **ESP32-CAM-MB USB Programmer**

- Description: The ESP32-CAM AI-Thinker MB programmer is a shield that can be attached to the ESP32-CAM board GPIOs. The programmer comes with the CH340C USB to serial chip; this allows serial communication to the ESP32-CAM using the USB port on the shield. Additionally, the shield also comes with a RESET and a BOOT (IO0) buttons. This may be useful to easily reset the ESP32-CAM or put it into flashing mode.
- Link Datasheet: N/A
- Cost: Approximately 2 Euro.

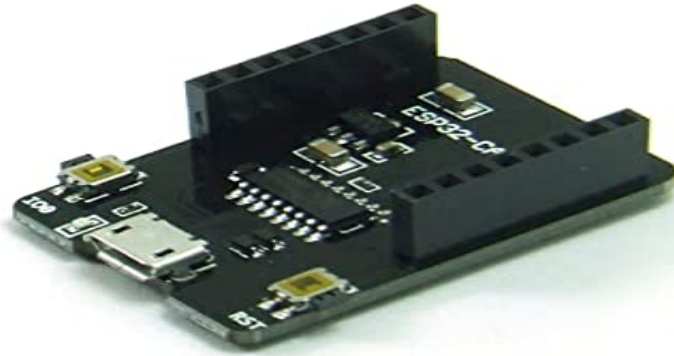


Figure 7.3: ESP32-CAM-MB USB Programmer.
Reference: [Ama22]

- **Secure Digital (SD) Card 16GB**

- Description: A memory Secure Digital card is an electronic data storage device used for storing digital information, typically using flash memory. These are commonly used in digital portable electronic devices. They allow adding memory to such devices using a card in a socket instead of a protruding USB flash drives.
- Link Datasheet: <https://www.intenso.de/en/products/memory-cards/microsd-card-class-10>
- Cost: Approximately 5 Euro.



Figure 7.4: Secure Digital Card.

- **ESP32-CAM Case**

- Description: A case to store the ESP32-CAM module to make it easy for the user transportation, install, and handle.

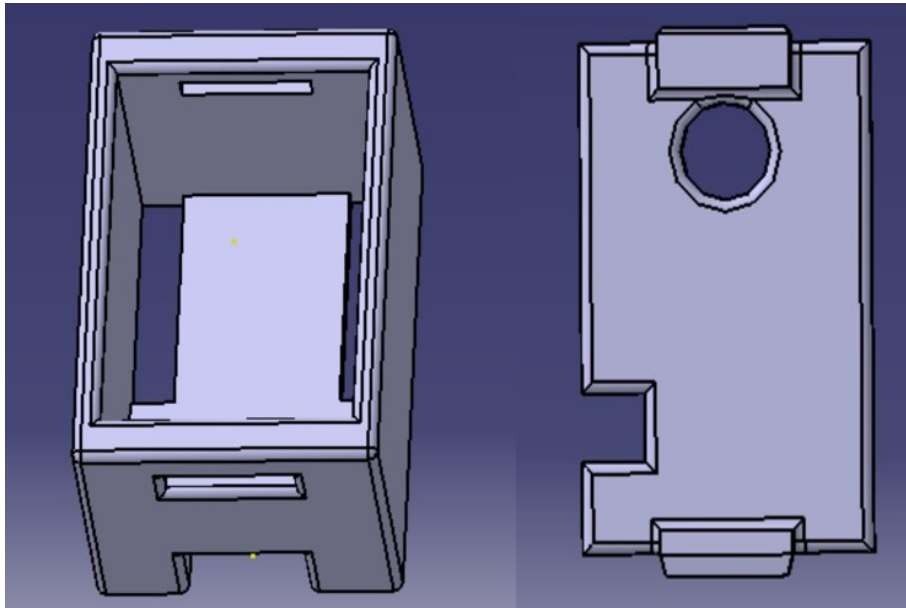


Figure 7.5: ESP32-CAM Case.

Reference: Designed by Author

- **USB cable - USB A / micro USB B**

- Description: USB 2.0 cable with A Male to Micro B connectors is a High-Speed Transmission device that supports up to 480 Mbps. It is mostly used for charging Android phones and tablets or connecting PC peripherals such as hard drives, printers, and more. In this specific case, to enable the communication between PC and ESP32-CAM.
- Link Datasheet: <https://www.tme.eu/Document/700e9581ffc48a630c6d37ae87e788bc/esb22.pdf>
- Cost: Approximately 3 Euro.



Figure 7.6: USB cable - USB A / micro USB B.
Reference: [Tra22]

SW List of materials required for the project:

- **Python v3.10.**
 - Description: Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems.
 - Link Release Notes: <https://docs.python.org/3/whatsnew/3.10.html>
 - Cost: FREE - Open source.



Figure 7.7: Python 3.10.

- **Visual Studio Code v1.73.1.**

- Description: VSCode, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.
- Link Release Notes: https://code.visualstudio.com/updates/v1_73
- Cost: FREE - Open source.



Figure 7.8: Visual Studio Code v1.73.1.

- **ESP-IDF VSCode Plug-In v1.5.1.**

- Description: ESP-IDF is Espressif's official IoT Development Framework for the ESP32, ESP32-S, ESP32-C and ESP32-H series of SoCs. It provides a self-sufficient SDK for any generic application development on these platforms, using programming languages such as C and C++.
- Link Release Notes: <https://github.com/espressif/vscode-esp-idf-extension/releases/tag/v1.5.1>
- Cost: FREE - Open source.

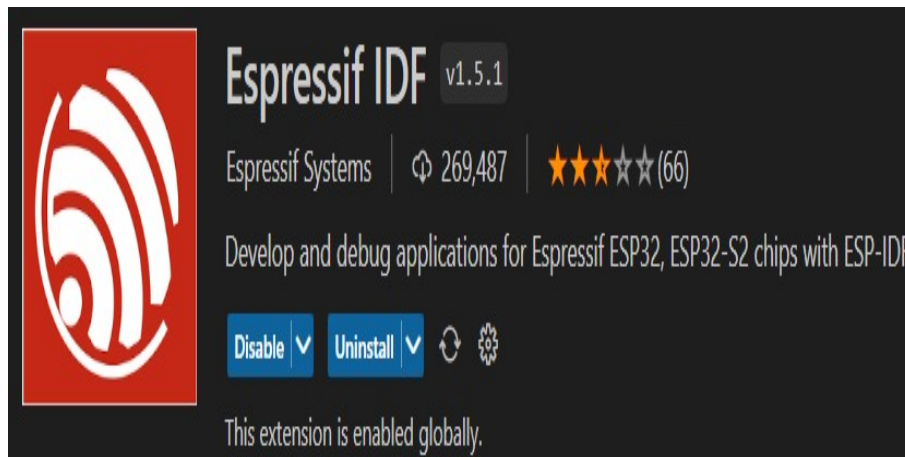


Figure 7.9: ESP-IDF Plug-In v1.5.1.

7.2 Description of the Python Package: Keras

For this project and in general for Machine Learning projects one of the most common packages required to undertake the development in python is Keras. In general, this is used to develop high-level neural networks, it is written in Python and is capable of running on top of TensorFlow. In the specific case of this project, Keras is been used in two stages, with the following tools:

- **ImageDataGenerator**, this tool provides a convenient and powerful way to load, pre-process, and augment image data for use in deep learning models. It allows to easy load image data from a directory structure, the application of pre-processing techniques, and the use of data augmentation to improve model performance.
- **Layers**, this tool is an essential component of a neural network model generation with Keras Package in python. A directed acyclic graph of the neural network is built using layers, where the output of one layer serves as the input for the following layer. Convolutional layers for image processing, recurrent layers for sequence data, and fully connected layers for dense neural networks are just a few of the pre-built layers available in the Keras framework. In the particular instance of this project, the following layers are mostly used for the implementation:
 - **Dense**, is also known as fully connected layer. It links all the neurons from the layer below to the neurons in the layer above. A hyper-parameter that has to be determined is how many neurons are present in the dense layer. In this layer, the activation function can also be specified.
 - **Conv2D**, is used for image processing. The input data is subjected to a convolution procedure, enabling the model to learn spatial hierarchies of

features. The hyper-parameters of this layer may be categorized as the number of filters, kernel size, and strides.

- **MaxPool2D**, is used for down-sampling the input data. It decreases the spatial dimensions of the data by applying a max pooling operation to the input data. The hyper-parameters of this layer may be categorized as the pooling size and the strides.
- **Flatten**, is used for flattening the multi-dimensional input data into a 1D array. It is frequently applied before the last dense layer in a model so that it can process the incoming data.

Building a variety of neural network models for image classification and other image processing tasks, such object recognition and semantic segmentation, may be done by combining the Dense, Conv2D, MaxPool2D, and Flatten layers. To test out several designs and identify the one that best matches the data, the number and mix of these layers may be altered.

Keras' simplicity and ease of use are two of its key benefits. It abstracts away a significant portion of the complexity involved in creating and refining deep learning models, allowing developers to concentrate on the model's architecture and design rather than the specifics of its implementation.

Support for several back-ends, such as TensorFlow, Theano, and Microsoft Cognitive Toolkit, is another benefit of Keras (CNTK). As a result, programmers may create models and train them using any back end of their choosing, switching back-ends without having to alter the model's architecture or training code.

Also it is a simple package to install and use in Python, in order to do this, the following steps must be followed:

1. Verify that Python is installed. This is done by running in the command prompt: `python -V`.
2. Install the required dependencies for Keras; numpy, scipy, and six. Run the following command in the command prompt: `pip install numpy scipy six`.
3. Install TensorFlow or one of the other supported backends. Keras is a high-level library that runs on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). Run the following command: `pip install tensorflow`.
4. Install Keras by running the following command: `pip install keras`.

In order to test that Keras is properly installed, use this code:

```
import keras;
print(keras.__version__)
```

As of now, the version of Keras being used is 2.9.0. Now lets present an example of how to use Keras in a simple python example:

Keras Example

January 26, 2023

1 Keras simple code example

Dereck Alpizar Arce - 7023782

1.1 Imports

```
[1]: import keras
      from keras.datasets import mnist
      from keras.utils import np_utils
      from keras.models import Sequential
      from keras.layers import Dense
```

```
[2]: print(keras.__version__)
```

2.9.0

1.2 Load the MNIST dataset

```
[3]: (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

1.3 Normalize the input data

```
[4]: X_train = X_train.astype('float32') / 255
      X_test = X_test.astype('float32') / 255
```

1.4 Convert the labels to categorical

```
[5]: num_classes = 10
      y_train = np_utils.to_categorical(y_train, num_classes)
      y_test = np_utils.to_categorical(y_test, num_classes)
```

1.5 Create the model

```
[6]: model = Sequential()
      model.add(Dense(512, activation='relu', input_shape=(784,)))
      model.add(Dense(num_classes, activation='softmax'))
```


1.6 Compile the model

```
[7]: model.compile(loss='categorical_crossentropy', optimizer='adam',  
    ↪ metrics=['accuracy'])
```

1.7 Train the model

```
[8]: history = model.fit(X_train.reshape(60000, 784), y_train, batch_size=128,  
    ↪ epochs=5, verbose=1, validation_data=(X_test.reshape(10000, 784), y_test))  
history
```

```
Epoch 1/5  
469/469 [=====] - 1s 3ms/step - loss: 0.2632 -  
accuracy: 0.9261 - val_loss: 0.1381 - val_accuracy: 0.9601  
Epoch 2/5  
469/469 [=====] - 1s 2ms/step - loss: 0.1071 -  
accuracy: 0.9683 - val_loss: 0.0948 - val_accuracy: 0.9713  
Epoch 3/5  
469/469 [=====] - 1s 2ms/step - loss: 0.0699 -  
accuracy: 0.9795 - val_loss: 0.0760 - val_accuracy: 0.9759  
Epoch 4/5  
469/469 [=====] - 1s 3ms/step - loss: 0.0506 -  
accuracy: 0.9847 - val_loss: 0.0737 - val_accuracy: 0.9765  
Epoch 5/5  
469/469 [=====] - 3s 6ms/step - loss: 0.0375 -  
accuracy: 0.9891 - val_loss: 0.0613 - val_accuracy: 0.9799
```

```
[8]: <keras.callbacks.History at 0x19235ed5f10>
```

Lastly it is important to reference the developer source for Keras releases (past, current and next). In this GitHub repository one can access more detailed information regarding this package: Keras: Deep Learning for humans

7.3 Description of the Python Package: NumPy

Numerical Python (NumPy) is one of the most fundamental packages available in python for numerical computation. It is a general-purpose array-processing package which provides high-performance multidimensional arrays and appropriate tools to work with them. NumPy is capable of storing generic multi-dimensional data. In NumPy, dimensions are known as axes and the rank denotes the number of axes present. NumPy's array class is termed as ndarray. [Dug22][Des19]

Functions:

- Computes basic array operations such as addition, multiplication, slicing, flatten, reshape and array indexing.
- Computes advanced array operations such as stacking arrays, splitting into sections and broadcasting arrays
- NumPy works with either Date Time or Linear Algebra

Features:

- Provides pre-compiled functions for numerical routines
- Array-oriented computing for better efficiency
- NumPy supports an object-oriented approach
- It is compact and performs faster computations with vectorization

Applications:

- Predominantly used in data-analysis applications.
- Used for creating powerful N-dimensional array
- Forms the base of other libraries, such as SciPy and scikit-learn
- Used as a replacement of MATLAB when used with SciPy and matplotlib

Installation

To install Numpy, you can use the pip package manager by running the command "pip install numpy" in your terminal.

Further Reading:

7 Appendix

- The Numpy documentation (<https://numpy.org/doc/>) provides a comprehensive overview of the library's features and usage.
- The Scipy website (<https://scipy.org/>) also provides additional resources for scientific computing with Python, including tutorials and a user guide.

Numpy Code Example

January 30, 2023

1 Numpy Code Example

Neeranjan Jayamurugan Karthikeyani - 7023553

1.0.1 Example 1:

Here is a simple example of how to use Numpy to create an array and perform basic operations on it:

```
[1]: import numpy as np

# Create a 1-dimensional array
arr = np.array([1, 2, 3, 4])

# Perform operations on the array
print(arr + 1) # [2, 3, 4, 5]
print(arr * 2) # [2, 4, 6, 8]
```

```
[2 3 4 5]
[2 4 6 8]
```

1.0.2 Example 2:

Here is another example of how to use Numpy to create a 2-dimensional array and perform more advanced operations on it:

```
[2]: import numpy as np

# Create a 2-dimensional array
arr = np.array([[1, 2, 3], [4, 5, 6]])

# Perform operations on the array
print(arr.shape) # (2, 3)
print(arr.mean()) # 3.5
```

```
(2, 3)
3.5
```

1.1 Numpy - Version Check:

You can check the version of Numpy by using the following command:

```
[3]: import numpy
      print(numpy.__version__)
```

1.21.5

1.2 Example - Files:

You can save and load numpy array into a file with numpy functions such as follows: “numpy.save” and “numpy.load”

```
[4]: import numpy as np
      arr = np.array([1, 2, 3, 4])
      np.save('my_array', arr)
      loaded_array = np.load('my_array.npy')
```

7.4 Description of the Python Package: Matplotlib

Matplotlib is an open-source drawing library which has powerful and wide range of visualizations. It extensively provides an object-oriented API which is used for embedding plots into applications. With the visualization capabilities of Matplotlib one can visualise everything that can be drawn from legends and grids to spectrogram. [Dug22][Des19]

Functions:

With few lines of code, Matplotlib can depict a wide range of visualizations such as:

- Line plots
- Scatter plots
- Area plots
- Bar charts and Histograms
- Pie charts
- Stem plots
- Contour plots
- Quiver plots
- Spectrograms

Features:

- With the advantage of being open source, it can be used as a replacement for MATLAB.
- Supports dozens of backends and output types, which enables one to use it regardless of type of the operating system used or the output format used.
- Good runtime behavior and low memory consumption.

Applications:

- Used for correlation analysis of variables.
- Used for visualizing 95 percent confidence intervals of the models.
- Outliers can be detected using a scatter plot.
- Can be used for visualizing the distribution of data to obtain instant insights.

Installation

Matplotlib can be installed using pip, conda, or by downloading the source code and running setup.py. It is recommended to install matplotlib through Anaconda distribution or using the command "pip install matplotlib".

Further Reading:

- For more information on matplotlib, check out the official matplotlib documentation at <https://matplotlib.org/>.

MatplotlibExampleCode

January 30, 2023

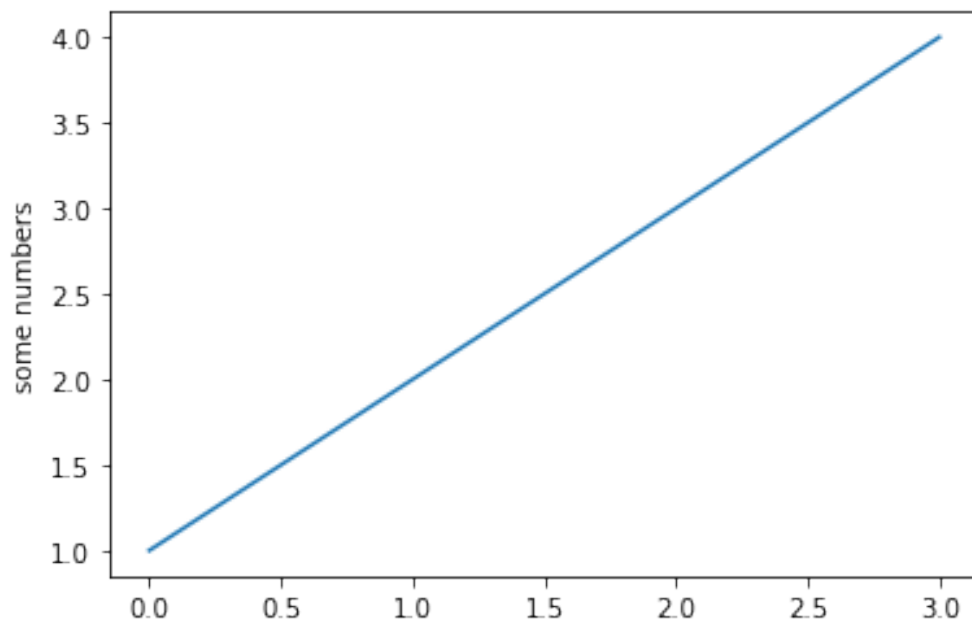
1 Matplotlib Code Example

Nisha Kumari - 7023211

1.0.1 Example 1:

Here is a simple example of how to use Numpy to create an array and perform basic operations on it:

```
[1]: import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```

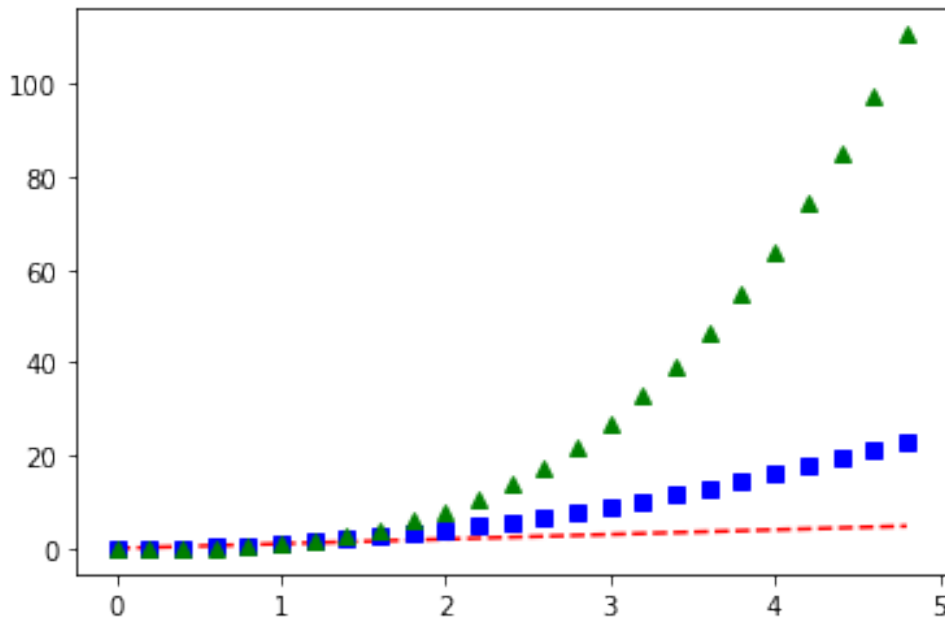


1.0.2 Example 2:

Here's a more complex example of how to plot multiple lines on the same plot:


```
[2]: import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0., 5., 0.2)
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



1.1 Matplotlib - Version Check:

You can check the version of Matplotlib by using the following command:

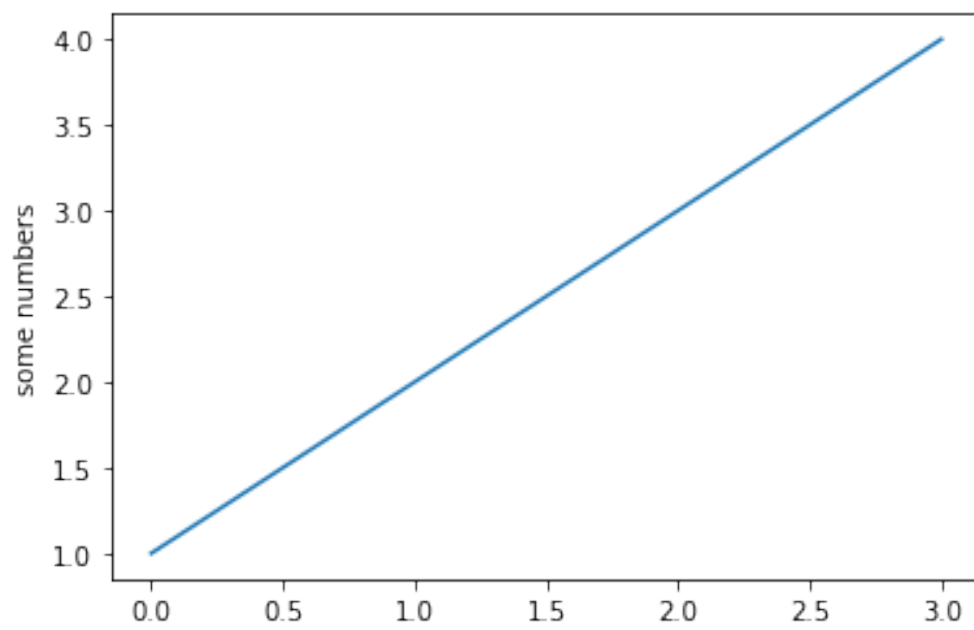
```
[3]: import matplotlib
print(matplotlib.__version__)
```

3.5.1

1.2 Example - Files:

Matplotlib can also save plots to various file formats, such as PNG, PDF, SVG, and more. Here's an example of how to save a plot as a PNG file:

```
[4]: import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.savefig('output.png')
```



Bibliography

- [Ama22] Amazon. “Popesq 1 x esp32-cam-mb programmer.” (2022), [Online]. Available: <https://www.amazon.de/POPESQ%C2%AE-Stk-ESP32-CAM-MB-Programmer-A4444/dp/B09HP56XZ3>.
- [Ard22] ArduCam. “Ov2640 – specs, datasheets, cameras, features, alternatives.” (2022), [Online]. Available: <https://www.arducam.com/ov2640/>.
- [Aru20] Arun, “An introduction to tinymml - machine learning meets embedded systems,” Oct. 2020. [Online]. Available: <https://towardsdatascience.com/an-introduction-to-tinymml-4617f314aa79>.
- [BFS19] M. Babiuch, P. Foltýnek, and P. Smutný, “Using the esp32 microcontroller for data processing,” in *2019 20th International Carpathian Control Conference (ICCC)*, 2019, pp. 1–6. DOI: 10.1109/CarpathianCC.2019.8765944.
- [BLO17] J. T. BLOG, “Wireless water meter,” *JH TECH BLOG*, 2017. [Online]. Available: <http://jheyman.github.io/blog/pages/WirelessWaterMeter/>.
- [Boe22] G. Boesch, “Tensorflow lite - real-time computer vision on edge devices,” 2022. [Online]. Available: <https://viso.ai/edge-ai/tensorflow-lite/>.
- [Des19] R. Desai, “Top 10 python libraries for data science,” Dec. 19, 2019. [Online]. Available: <https://towardsdatascience.com/top-10-python-libraries-for-data-science-cd82294ec266> (visited on 11/29/2022).
- [Dug22] N. Duggal, “Top 10 python libraries for data science for 2023,” Nov. 18, 2022. [Online]. Available: https://www.simplilearn.com/top-python-libraries-for-data-science-article#3_numpy (visited on 11/29/2022).
- [Edu20] I. C. Education, “Convolutional neural networks,” *IBM Cloud Learn Hub*, Oct. 2020. [Online]. Available: <https://www.ibm.com/cloud/learn/convolutional-neural-networks>.
- [FSS96] U. Fayyad, G. Shapiro, and P. Smyth, “Knowledge discovery and data mining:towards a unifying framework,” *ASSOCIATION FOR THE ADVANCEMENT OF ARTIFICIAL INTELLIGENCE*, 1996. [Online]. Available: <https://www.aaai.org/Papers/KDD/1996/KDD96-014.pdf>.
- [Ign22] B. A. Ignacio. “Espressif idf.” (2022), [Online]. Available: <https://github.com/espressif/vscode-esp-idf-extension/blob/v1.5.1/docs/tutorial/install.md>.

Bibliography

- [Kha20] R. Khandelwal, “A basic introduction to tensorflow lite,” 2020. [Online]. Available: <https://towardsdatascience.com/a-basic-introduction-to-tensorflow-lite-59e480c57292>.
- [KSP+17] E. O. Kontis, I. S. Skondrianos, T. A. Papadopoulos, A. I. Chrysochos, and G. K. Papagiannis, “Generic dynamic load models using artificial neural networks,” in *2017 52nd International Universities Power Engineering Conference (UPEC)*, 2017, pp. 1–6. DOI: 10.1109/UPEC.2017.8231937.
- [Lab21] M. Lab, “Esp32-cam ai-thinker board – all about gpio pins,” 2021. [Online]. Available: <https://microcontrollerslab.com/esp32-cam-ai-thinker-pinout-gpio-pins-features-how-to-program/>.
- [Lis21] J. List, “An esp will read your meter for you,” *Personal Blog*, 2021. [Online]. Available: <https://hackaday.com/2021/02/07/an-esp-will-read-your-meter-for-you/>.
- [Mak21a] H. Make Media, “Ai for the esp32 - part 1,” *Make Media*, 2021.
- [Mak21b] H. Make Media, “Ai for the esp32 - part 2,” *Make Media*, 2021.
- [Mak21c] H. Make Media, “Ai for the esp32 - part 3,” *Make Media*, 2021.
- [Mar21] J. Martin, “Limitations of using ocr for file classification,” *IDM Magazine*, 2021. [Online]. Available: <https://idm.net.au/article/0011231-limitations-using-ocr-file-classification>.
- [Mic22] Microsoft. “Download visual studio code.” (2022), [Online]. Available: <https://code.visualstudio.com/download>.
- [Mül20] J. Müller. “Ki für den esp32.” (2020), [Online]. Available: <https://github.com/jomjol/water-meter-measurement-system>.
- [OAG+21] A. Osman, U. Abid, L. Gemma, M. Perotto, and D. Brunelli, “Tinymml platforms benchmarking,” <https://www.researchgate.net>, 2021.
- [PS20] B. Posey and J. Scarpati, “Definition edge device,” <https://www.techtarget.com/>, 2020.
- [Ray22] P. P. Ray, “A review on tinymml: State-of-the-art and prospects,” *www.sciencedirect.com*, 2022.
- [Rei22] T. Reidt, “Types and examples of embedded devices,” <https://emteria.com/learn/embedded-device>, 2022.
- [RJA+21] D. Robert *et al.*, “Tensorflow lite micro: Embedded machine learning for tinymml systems,” *Proceedings of Machine Learning and Systems*, vol. 3, 2021. DOI: <https://doi.org/10.48550/arXiv.2010.08678>.
- [RS22] M. Rüb and A. Sikora, “A practical view on training neural networks in the edge,” *www.sciencedirect.com*, 2022.

- [Sag20] R. Sagar, “What are the challenges of establishing a tinymml ecosystem,” *Analytics India Magazine*, 2020. [Online]. Available: <https://analyticsindiamag.com/tinymml-ecosystem-challenges-machine-learning-iot/>.
- [Sah18] S. Saha, “A comprehensive guide to convolutional neural networks — the eli5 way,” *Towards Data Science*, Dec. 2018. [Online]. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.
- [Spa20] J. Sparber, “Digitizing a analog water meter,” *Personal blog*, 2020. [Online]. Available: <https://blogs.gnome.org/jsparber/2020/01/18/digitizing-a-analog-water-meter/>.
- [Sto22] E. Store. “Esp32-cam-ch340 development board.” (2022), [Online]. Available: <https://www.elektor.de/esp32-cam-ch340-development-board>.
- [Swa22] K. Swapna, “Convolutional nural network | deep learning,” 2022. [Online]. Available: <https://developersbreach.com/convolution-neural-network-deep-learning/>.
- [Sys22] E. Systems. “About espressif.” (2022), [Online]. Available: <https://www.espressif.com/en/company/about-espressif>.
- [Tra22] Tradeinn. “Startech micro usb cable a to micro b 15 cm.” (2022), [Online]. Available: <https://www.tradeinn.com/techinn/de/startech-micro-usb-cable-a-to-micro-b-15-cm/137834550/p>.
- [ZC18] A. Zheng and A. Casari, *Feature Engineeringfor Machine Learning*, R. Roumeliotis and J. Bleiel, Eds. O’Reilly, 2018.

