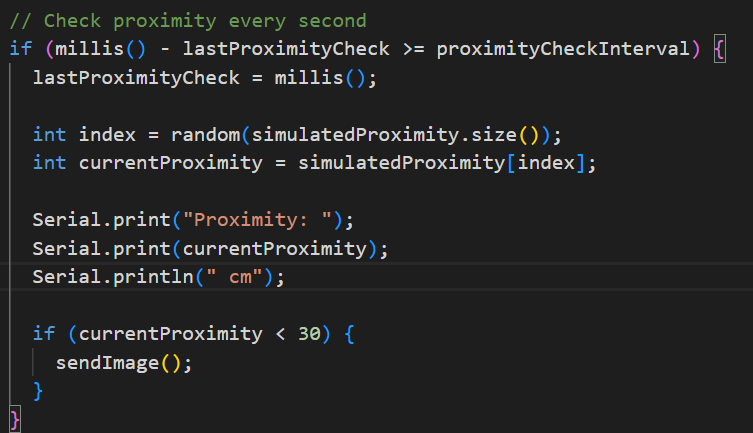
**Lesson 18 assignment will continue the previous assignment to further improve the effectiveness of our esp32 to classify the image of fruits. This time a proximity sensor should be used. However we do not have one, instead we will create a randomize value for proximity, so that every second if the value is less than 30 cms, esp32 will send image for classification**



**Components:**

1. **ESP32 (IoT Device and Data Source):**
   * Equipped with sensors.
   * Collects raw data.
   * Establishes a connection to an MQTT broker (running on your computing device or elsewhere).
   * Publishes sensor data to specific MQTT topics.
2. **Your Computing Device (Desktop/Laptop/Server):**
   * Runs an MQTT broker (e.g., Mosquitto) or connects to an existing one.
   * Executes a Python script that functions as the CNN model runner.
   * This Python script utilizes libraries like TensorFlow/Keras or PyTorch to load and run the trained CNN model.
   * The Python script subscribes to the MQTT topics where the ESP32 is publishing data.
   * Upon receiving data:
     + Processes the data using the CNN model.
     + Generates a result or prediction.
   * Publishes the result back to a different MQTT topic that the ESP32 is subscribed to.

**Data Flow:**

1. ESP32 collects data.
2. ESP32 publishes data to an MQTT topic.
3. Your computing device's Python script subscribes to that topic.
4. The Python script receives the data.
5. The CNN model (running within the Python script) processes the data.
6. The Python script publishes the inference result to another MQTT topic.
7. The ESP32 subscribes to this result topic and receives the prediction.
8. The ESP32 acts on the received result.

**Advantages of this Architecture:**

* **Utilizes Existing Computing Resources:** You can leverage the processing power of your desktop, laptop, or a dedicated server, which likely has more resources than a Raspberry Pi, allowing for more complex CNN models.
* **Easier Development and Debugging:** Developing and debugging Python code with full access to development tools and libraries on your computer is generally easier than on an embedded device.
* **Faster Prototyping:** This setup can be quicker to get up and running for initial testing and proof-of-concept.
* **Potentially More Powerful Processing:** Your computer might have a GPU that can significantly accelerate CNN inference.

**Considerations:**

* **Dependency on Your Computing Device:** The system's functionality is tied to your computing device being powered on and connected to the network.
* **Higher Power Consumption (Usually):** A desktop or laptop typically consumes significantly more power than a Raspberry Pi or an ESP32.
* **Network Dependency:** The ESP32 and your computing device need to be on the same network (or have network connectivity to a remote MQTT broker).
* **Latency:** Network latency between the ESP32 and your computing device might be higher than if the processing happened locally on an edge device.
* **Not Ideal for Truly Embedded Deployments:** This architecture isn't suitable for scenarios where the final product needs to be a standalone, low-power embedded system. It's more appropriate for development, testing, or applications where a nearby computing device is available.

**Use Cases:**

* **Development and Prototyping:** Quickly testing AI models with ESP32 data.
* **Lab or Controlled Environments:** Where a computer is readily available.
* **Data Logging and Analysis:** Collecting data from an ESP32 and performing more intensive analysis on a computer.
* **Human-in-the-Loop Systems:** Where a user might interact with the results on the computer.

**In summary, having your computing device run the CNN model based on ESP32 data sent via MQTT is a practical approach, especially for development and situations where the constraints of a fully embedded edge deployment are not the primary concern. It allows you to utilize the computational power of your computer for AI processing.** When you are ready for a more standalone or lower-power solution, you might then consider deploying the model to a more dedicated edge device like a Raspberry Pi or an even more resource-constrained microcontroller if the model can be optimized sufficiently.