



REALTEK

Ameba 82 ワークショップ

Outline

Chapter 1 Edge AI

1.1 AIoT

1.2 Edge computing

Chapter 2 AMB82-MINI

2.1 AMB82-MINI Introduction

2.2 LoopPostProcessing

2.3 Audio Classification

2.4 Face Recognition

2.5 Image Classification

2.6 MQTT ON AMB82

Outline

Chapter 3 **Object Detection**

3.1 YOLO(You Only Look Once)

3.2 YOLOv7 Gesture Detection



Chapter 1

Edge AI



1.1 AIoT

AIoTとは

- AIoTは、人工知能とモノのインターネットを組み合わせた言葉で、高度な人工知能を使って物事を深く理解し、自動的に判断することで、IoTシステムをより効率的かつ賢くします。

AIoTの主な特徴

- データ収集と分析: IoTデバイスは、センサーを通じてデータを収集し、通信ネットワーク経由で大量のデータを送信し、AIを用いて分析することで、有益な情報やパターンを抽出します。

AIoTの主な特徴

- **意思決定と制御の自動化:** 人工知能は、データ分析の結果に基づいて自動的に意思決定を行い、その決定をIoTデバイスを通じて実行します。

AIoTの主な特徴

- **予測と予防:** AIoTシステムは、機械学習モデルを利用して将来のトレンドやイベントを予測することができます。これにより、**予防保全や資源の最適化が可能になります。**

AIoTの主な特徴

- **学習と最適化:** AIoTシステムは、過去のデータや経験から学習し、その性能と意思決定能力を継続的に向上させることができます。これにより、システムは変化に対応し、より効率的なサービスを提供できます。

Layers of AIoT

- **Device Layer:** 感測器やアクチュエータは、環境からデータを収集し、操作を実行する物理的なデバイスです。
- **Connectivity Layer:** この層は、デバイスと他の層間の通信を処理します。一般的なプロトコルには、Wi-Fi、Bluetooth、MQTTなどがあります。

Layers of AIoT

- **Edge Computing Layer:** エッジコンピューティング層は、デバイスの近くでデータを処理し、分析を行います。
- **Data Management Layer:** データ管理層は、デバイスから収集されたデータを保存し、管理するシステムです。クラウドベースのストレージやローカルストレージが含まれます。

Layers of AIoT

- **AI Analytics Layer:** AI分析層は、機械学習モデルを用いて、収集されたデータを分析し、予測や自動化のための決定を下す層です。
- **Application Layer:** アプリケーション層は、ユーザーがAIoTシステムと相互作用し、システムの状態を監視したり、デバイスを制御したりするための層です。一般的なインターフェースには、アプリやWebインターフェースなどがあります。



1.2 Edge Computing

エッジコンピューティング

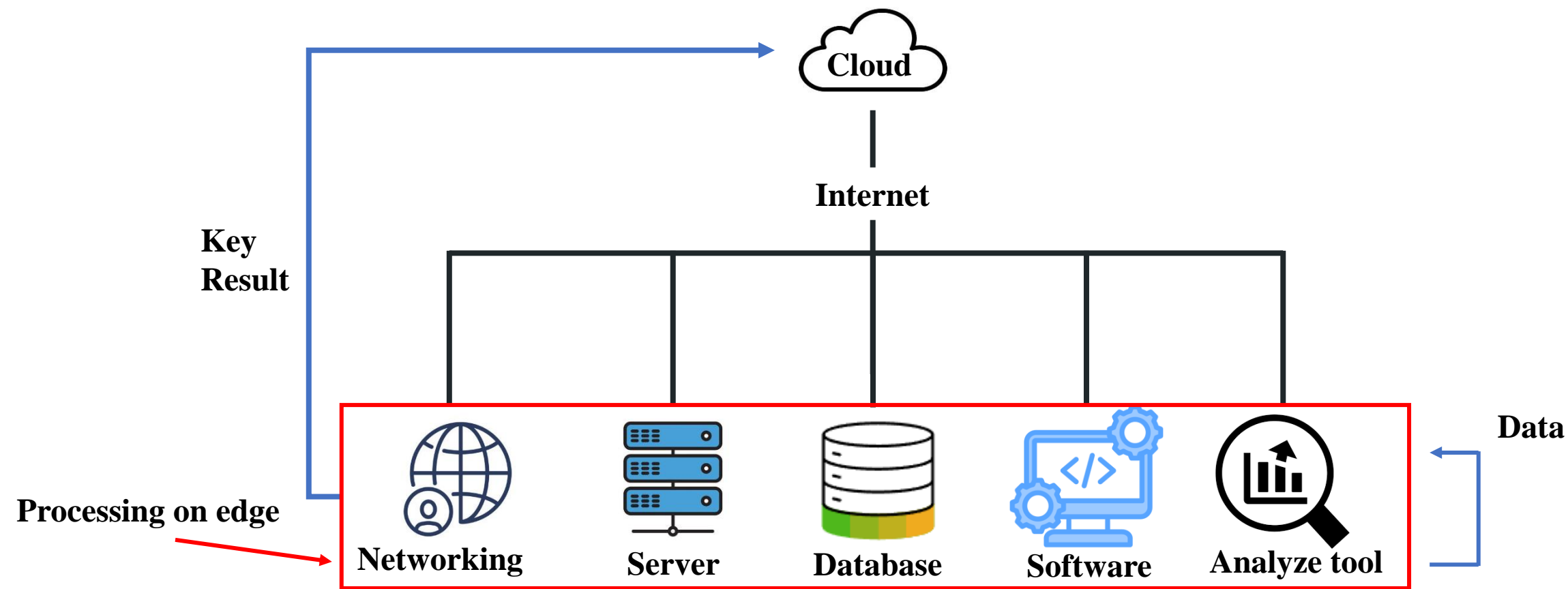
- エッジコンピューティングは、データの処理と分析を集中型のクラウド構造から、データ生成の源に近いエッジデバイスに移行することを目的としています。

エッジコンピューティング

- **目的:** エッジデバイスでリアルタイムに処理と分析を行うことで、遅延を低減し、帯域幅の要求を減らし、システムの信頼性とデータのセキュリティを向上させることができます。

1.2 Edge Computing

Edge Computing



なぜエッジコンピューティングが重要なのか？

- 遅延を減らし、高速化
- データセキュリティの向上
- 生産性の向上
- 容易に統合、運用
- コスト削減

edge computingの応用事例

- 小売
- 自動運転
- 医療
- 教育



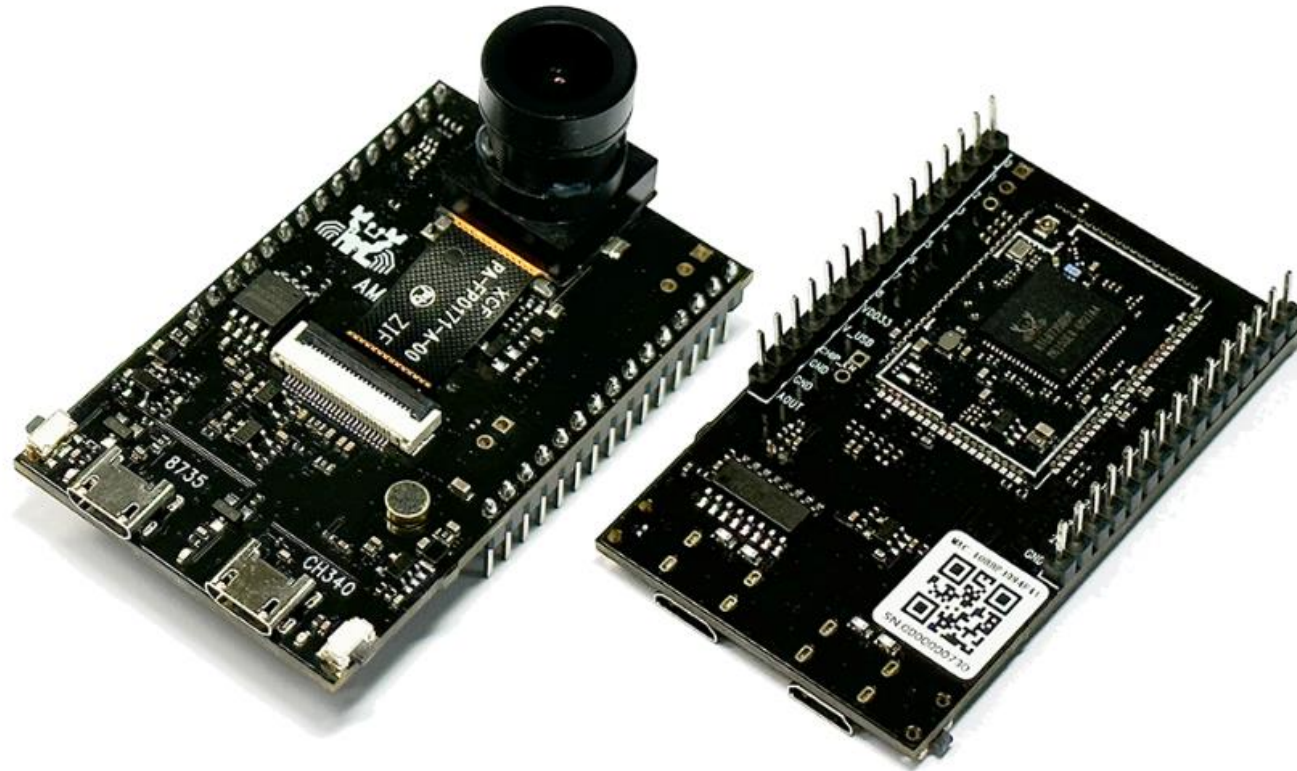
Chapter 2

AMB82-MINI



2.1 AMB82-MINI Introduction

2.1 AMB82-MINI Introduction



<https://www.amebaiot.com/zh/amebapro2/>

2.1 AMB82-MINI Introduction

AMB82 MINIは何ができますか？

- WiFi/BLE
- GPIO/PWM
- 電子ペーパー
- オーディオ/ビデオ
- AI ニューラルネットワーク



2.2 Loop Post Processing

2.2 LoopPostProcessing



2.2 LoopPostProcessing

動体検知とは何ですか？

- **定義：動的にビデオを読み取り、位置の変化を検出します。**

2.2 LoopPostProcessing

動体検知はどのように機能しますか

以下は ChatGPT の回答:

1. 色検出
2. 深度カメラ
3. 機械学習
4. オプティカルフロー

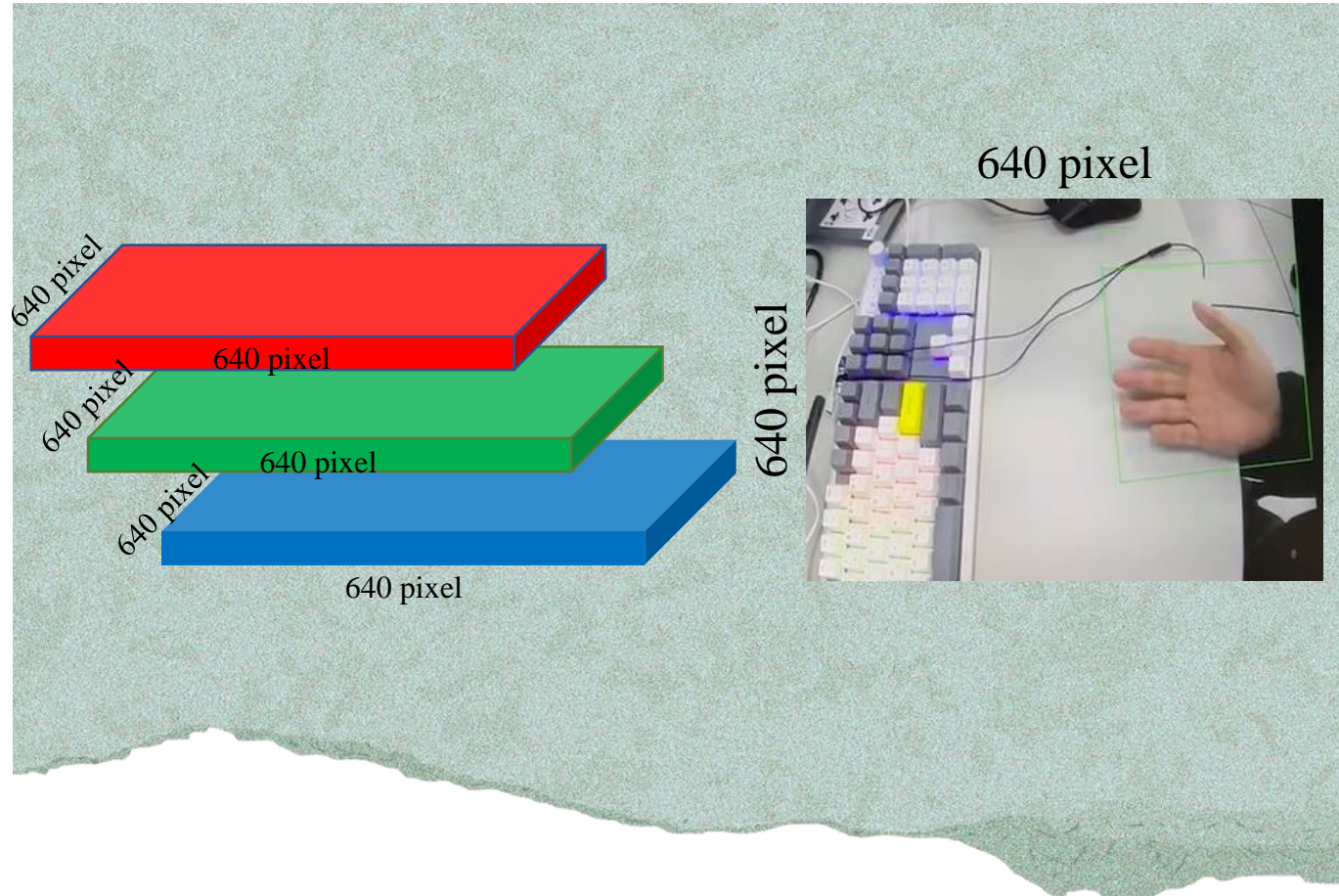
2.2 LoopPostProcessing

AMBの実際の動作はどのようなものですか？

- 隣接する2つのフレーム間の**RGBの差**を計算し、**しきい値**を使用して動きの変化があるかどうかを判断します。

2.2 LoopPostProcessing

RGB channel



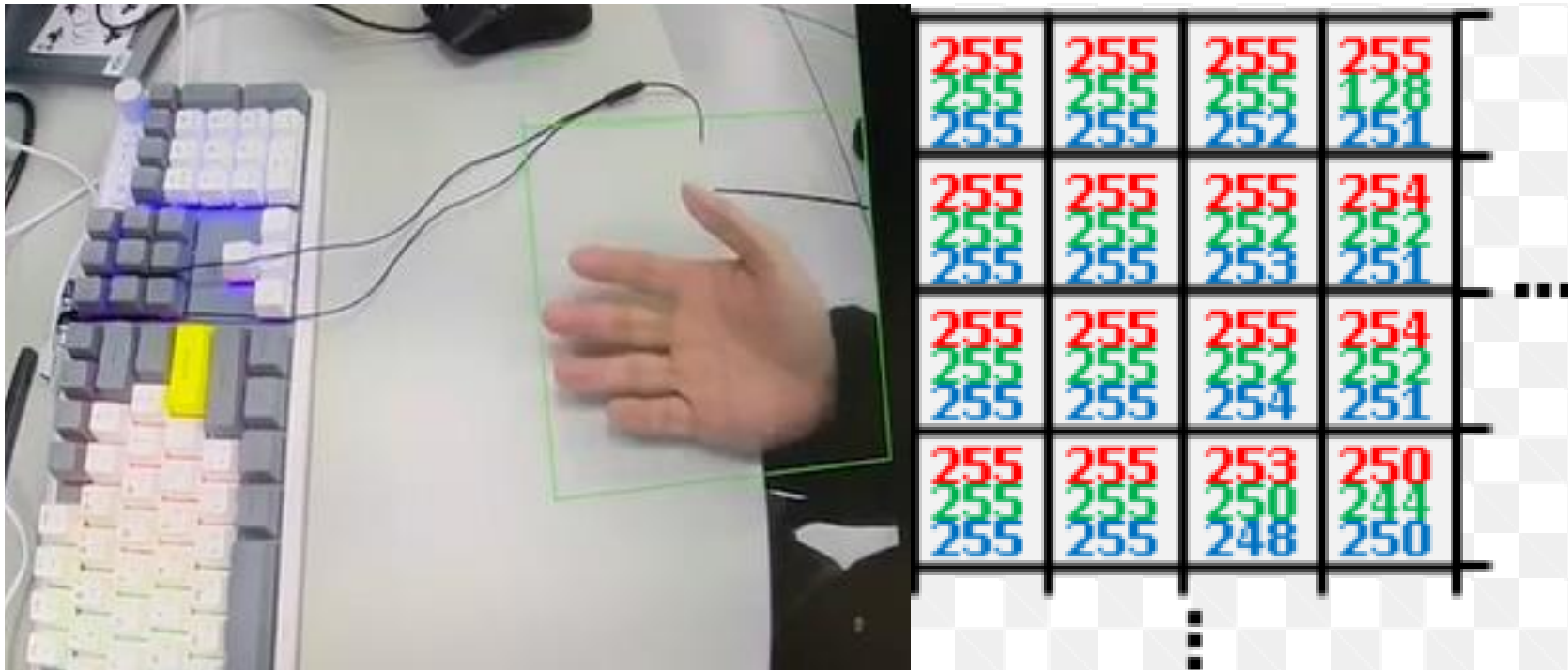
2.2 LoopPostProcessing

RGBの差をどのように定義しますか

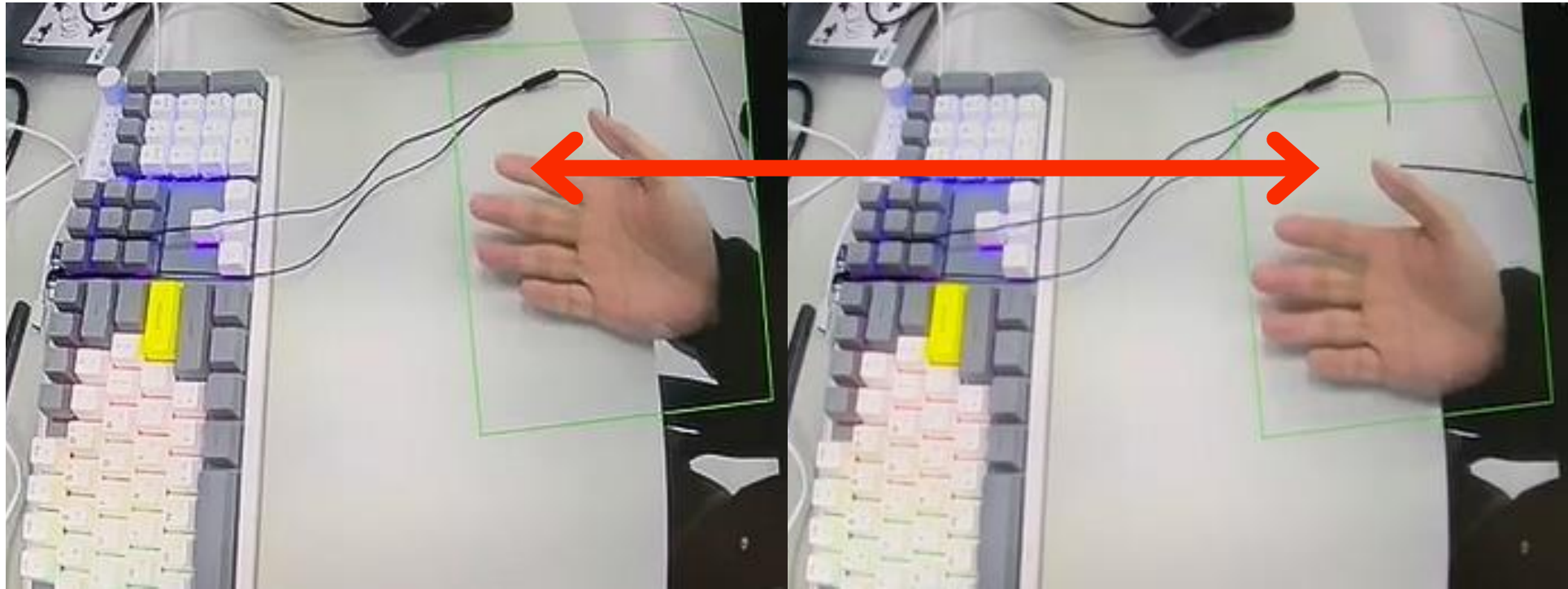
各フレーム内のすべてのピクセルのRGB値の変化を観察
します。2つの異なるフレームのRGB値を(R1, G1, B1)と
(R2, G2, B2)と仮定します。

$$\text{diff} = \sqrt{(R2 - R1)^2 + (G2 - G1)^2 + (B2 - B1)^2}$$

2.2 LoopPostProcessing



2.2 LoopPostProcessing



2.2 LoopPostProcessing

$$\text{diff} = \sqrt{(255 - 185)^2 + (255 - 134)^2 + (255 - 115)^2}$$

Color	Hex	Mode	R	G	B	Red Label	Green Label	Blue Label
Brown	#B98673	RGB	185	134	115	紅色 赤	綠色 緑	藍色 青
White	#FFFFFF	RGB	255	255	255	紅色 赤	綠色 緑	藍色 青

2.2 LoopPostProcessing

$$\text{diff} = \sqrt{(255 - 185)^2 + (255 - 134)^2 + (255 - 115)^2}$$

$$\text{diff} = \sqrt{70^2 + 121^2 + 140^2}$$

$$\text{diff} = \sqrt{4900 + 14641 + 19600}$$

$$\text{diff} = \sqrt{39141}$$

$$\text{diff} \approx 197.84$$

2.2 LoopPostProcessing

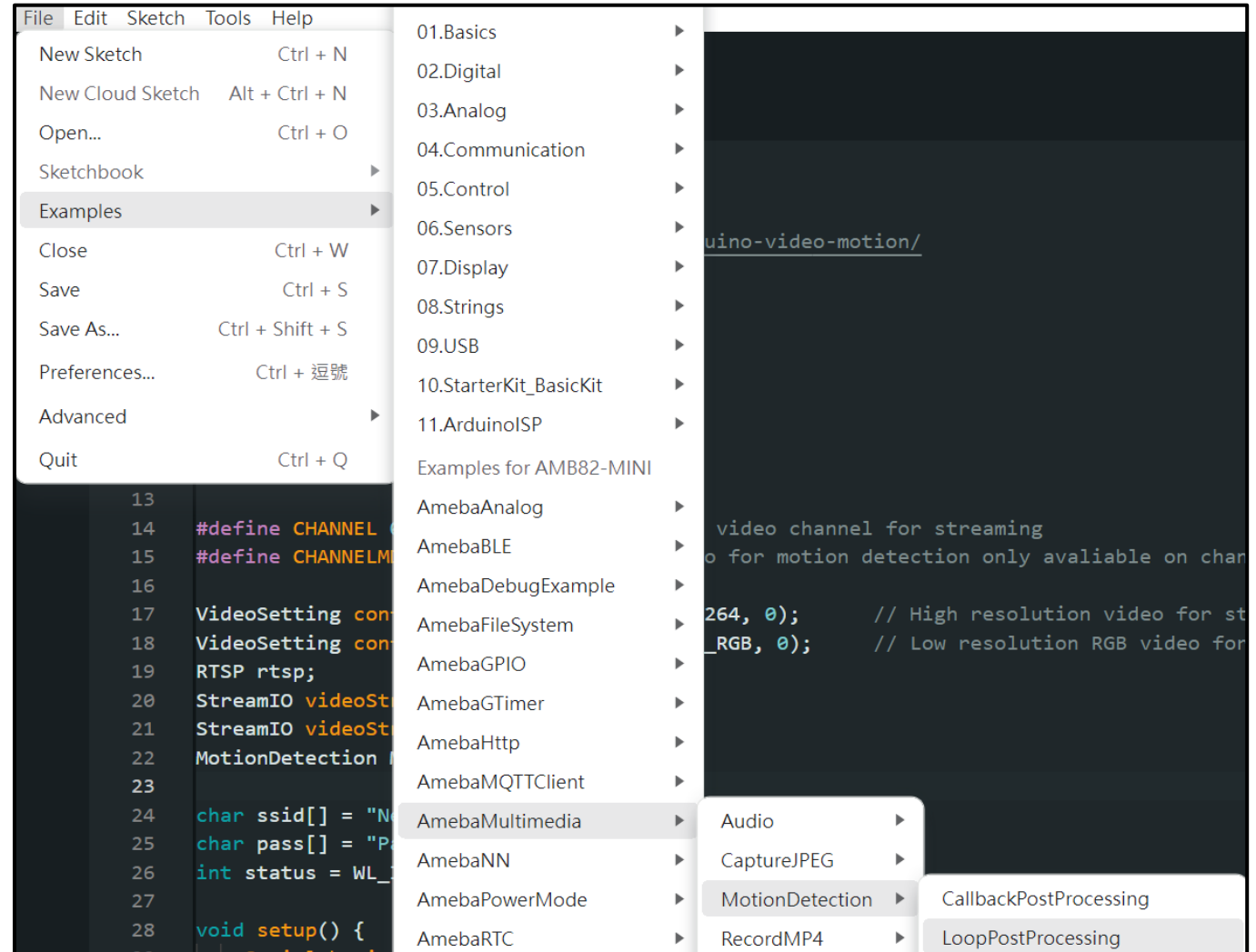
Implementation

2.2 LoopPostProcessing

Step 1.

Arduino IDEで例を開くには、以下のパスに従ってください。

1. File
2. Examples
3. AmebaMultimedia
4. MotionDetection
5. LoopPostProcessing



2.2 LoopPostProcessing

Step 2.

プログラムのWi-Fi接続設定に、
SSIDとパスワードを設定してください。

```
#include <WiFi.h>
#include "StreamIO.h"
#include "VideoStream.h"
#include "RTSP.h"
#include "NNObjectDetection.h"
#include "VideoStreamOverlay.h"
#include "ObjectClassList.h"

#define CHANNEL 0
#define CHANNELLN 3

// Lower resolution for NN processing
#define NNWIDTH 576
#define NNHEIGHT 320

VideoSetting config(VIDEO_FHD, 30, VIDEO_H264, 0);
VideoSetting configNN(NNWIDTH, NNHEIGHT, 10, VIDEO_RGB, 0);
NNObjectDetection objDet;
RTSP rtsp;
StreamIO videoStreamer(1, 1);
StreamIO videoStreamerNN(1, 1);

char ssid[] = "Network_SSID"; // your network SSID (name)
char pass[] = "Password"; // your network password
int status = WL_IDLE_STATUS;

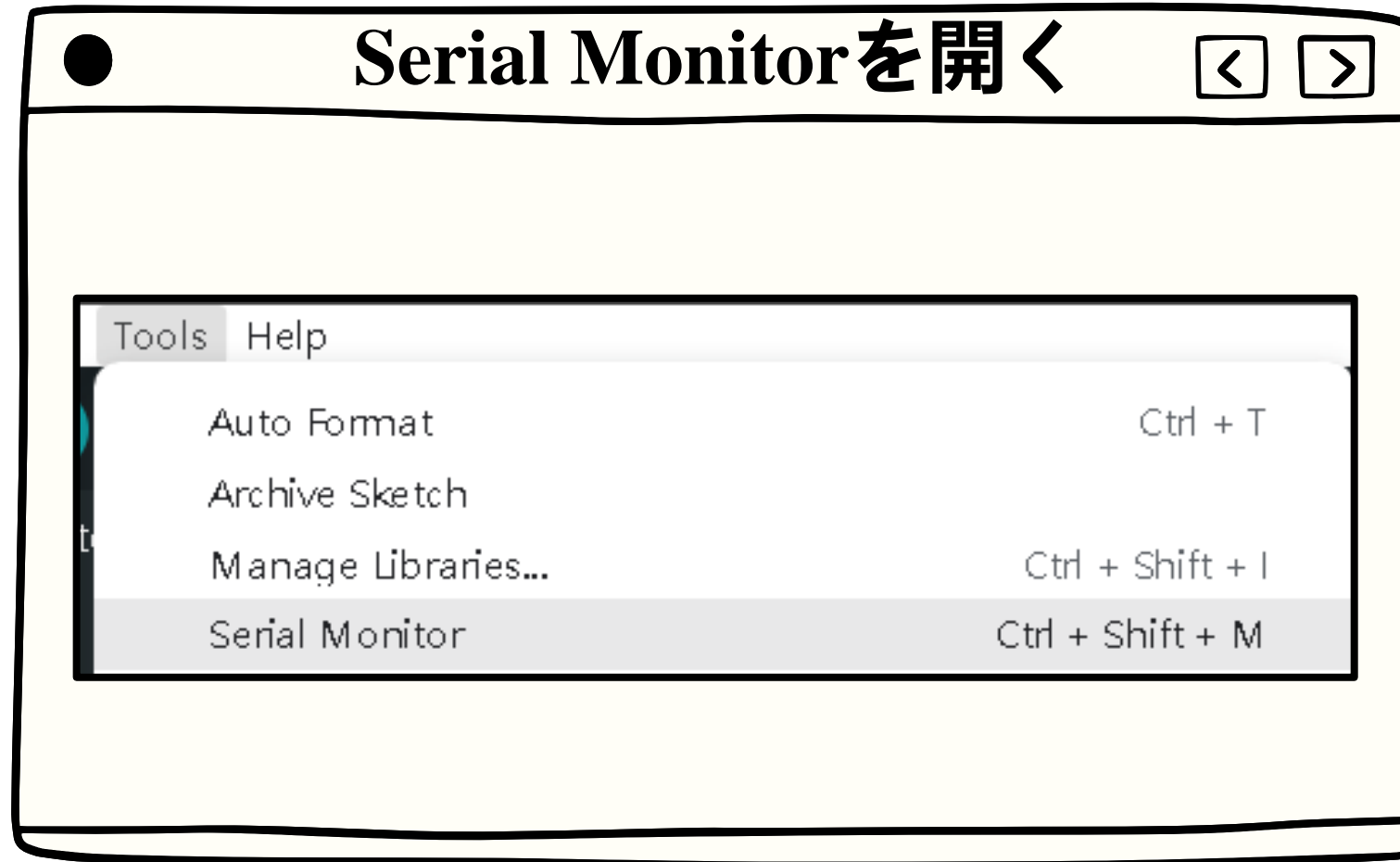
IPAddress ip;
int rtsp_port;

void setup() {
  Serial.begin(115200);

  // attempt to connect to Wifi network:
```

WiFiの名前とパスワード
を入力してください

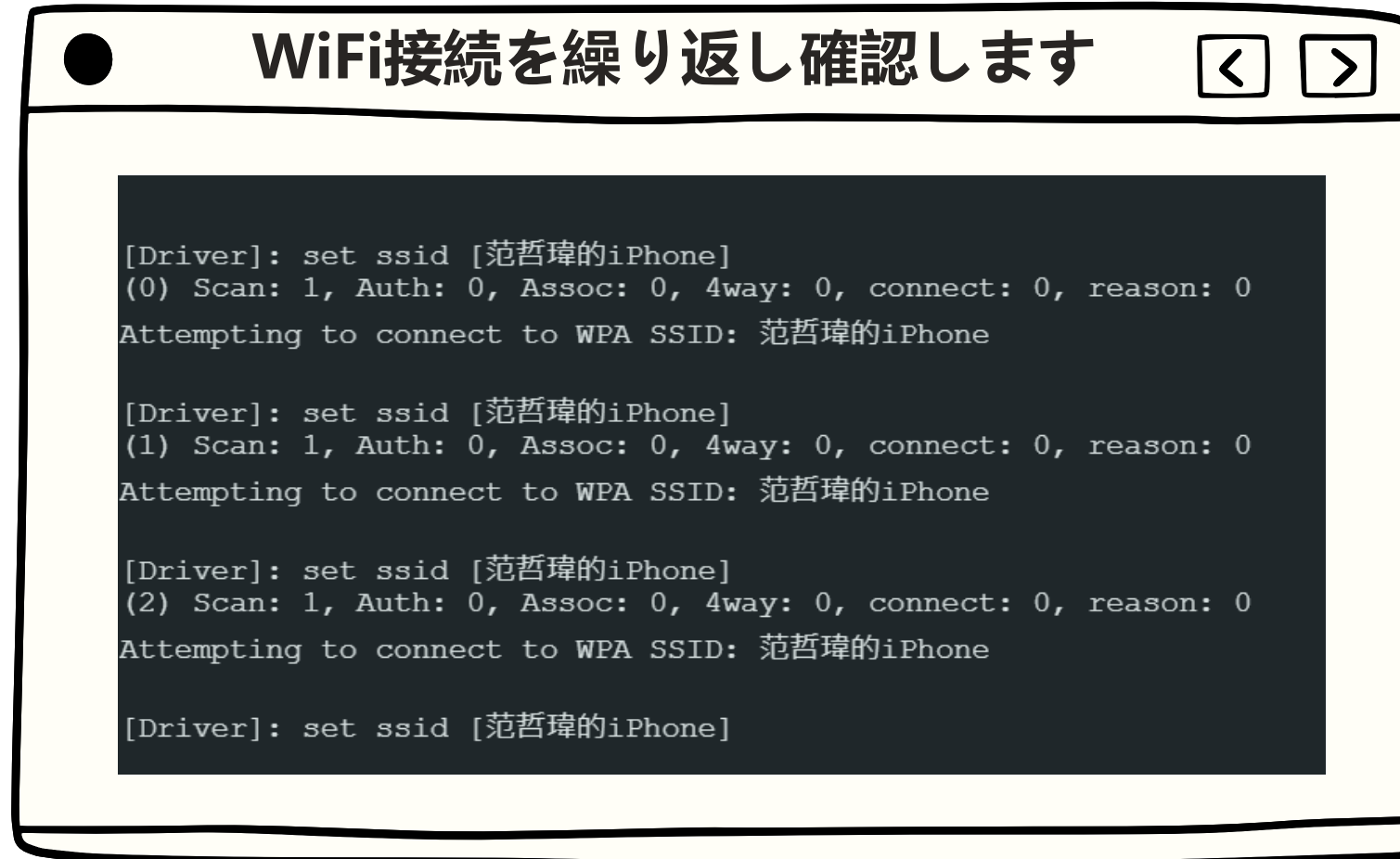
2.2 LoopPostProcessing



2.2 LoopPostProcessing

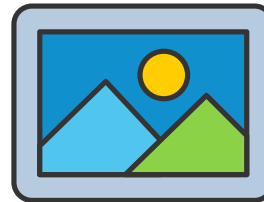
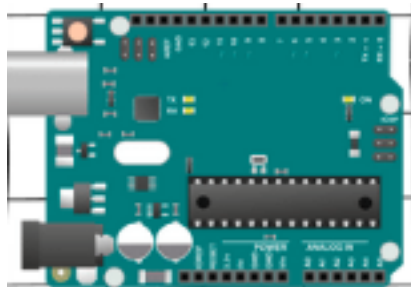
```
while (status != WL_CONNECTED) {  
    Serial.print("Attempting to connect to WPA SSID: ");  
    Serial.println(ssid);  
    status = WiFi.begin(ssid, pass);  
  
    // wait 2 seconds for connection:  
    delay(2000);  
}
```

2.2 LoopPostProcessing



2.2 LoopPostProcessing

RTSP-Real Time Streaming Protocol



2.2 LoopPostProcessing

RTSP-Real Time Streaming Protocol

- **エンターテインメントおよび通信システム**用に特別に設計されたネットワークアプリケーションプロトコルで、ストリーミングメディアサーバーを制御します。

2.2 LoopPostProcessing

Step 1.

パソコンとAMB82が**同じWi-Fi**に接続されていることを確認してください。



2.2 LoopPostProcessing

Step 2.

プログラム内の設定と同じように、
serial monitorの**Baud**を**115200**に設定
してください。

```
char ssid[] = "Network_S  
char pass[] = "Password"  
int status = WL_IDLE_STA
```

```
IPAddress ip;  
int rtsp_portnum;
```

```
void setup() {  
  Serial.begin(115200)
```

```
  // attempt to connect  
  while (status != WL_  
    Serial.println("At
```

115200 baud

4800 baud

9600 baud

19200 baud

31250 baud

38400 baud

57600 baud

74880 baud

115200 baud

2.2 LoopPostProcessing

Step 3.

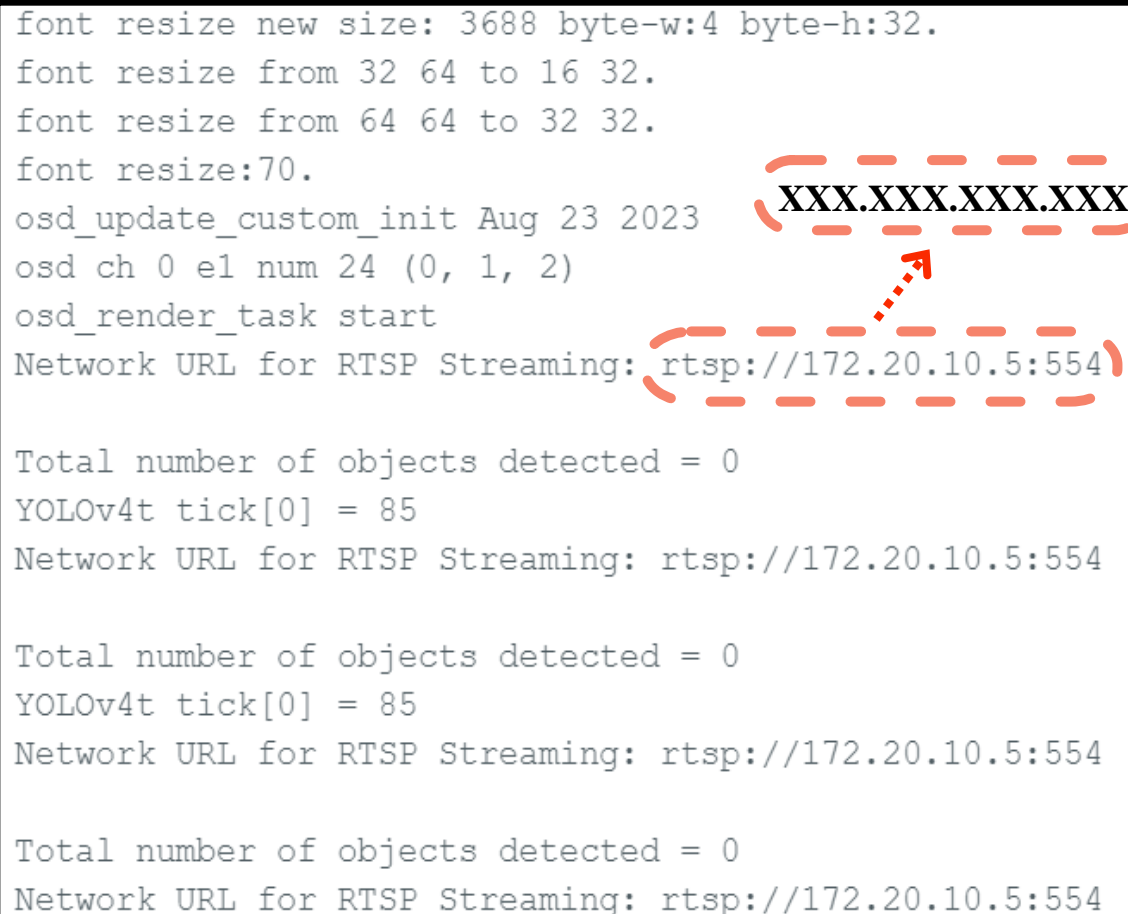
AMB82のresetボタンを押して、
serial monitorでIPアドレスを確認し
ます。次に、それをコピーしてくだ
さい。

```
font resize new size: 3688 byte-w:4 byte-h:32.
font resize from 32 64 to 16 32.
font resize from 64 64 to 32 32.
font resize:70.
osd_update_custom_init Aug 23 2023
osd ch 0 e1 num 24 (0, 1, 2)
osd_render_task start
Network URL for RTSP Streaming: rtsp://172.20.10.5:554

Total number of objects detected = 0
YOLOv4t tick[0] = 85
Network URL for RTSP Streaming: rtsp://172.20.10.5:554

Total number of objects detected = 0
YOLOv4t tick[0] = 85
Network URL for RTSP Streaming: rtsp://172.20.10.5:554

Total number of objects detected = 0
Network URL for RTSP Streaming: rtsp://172.20.10.5:554
```

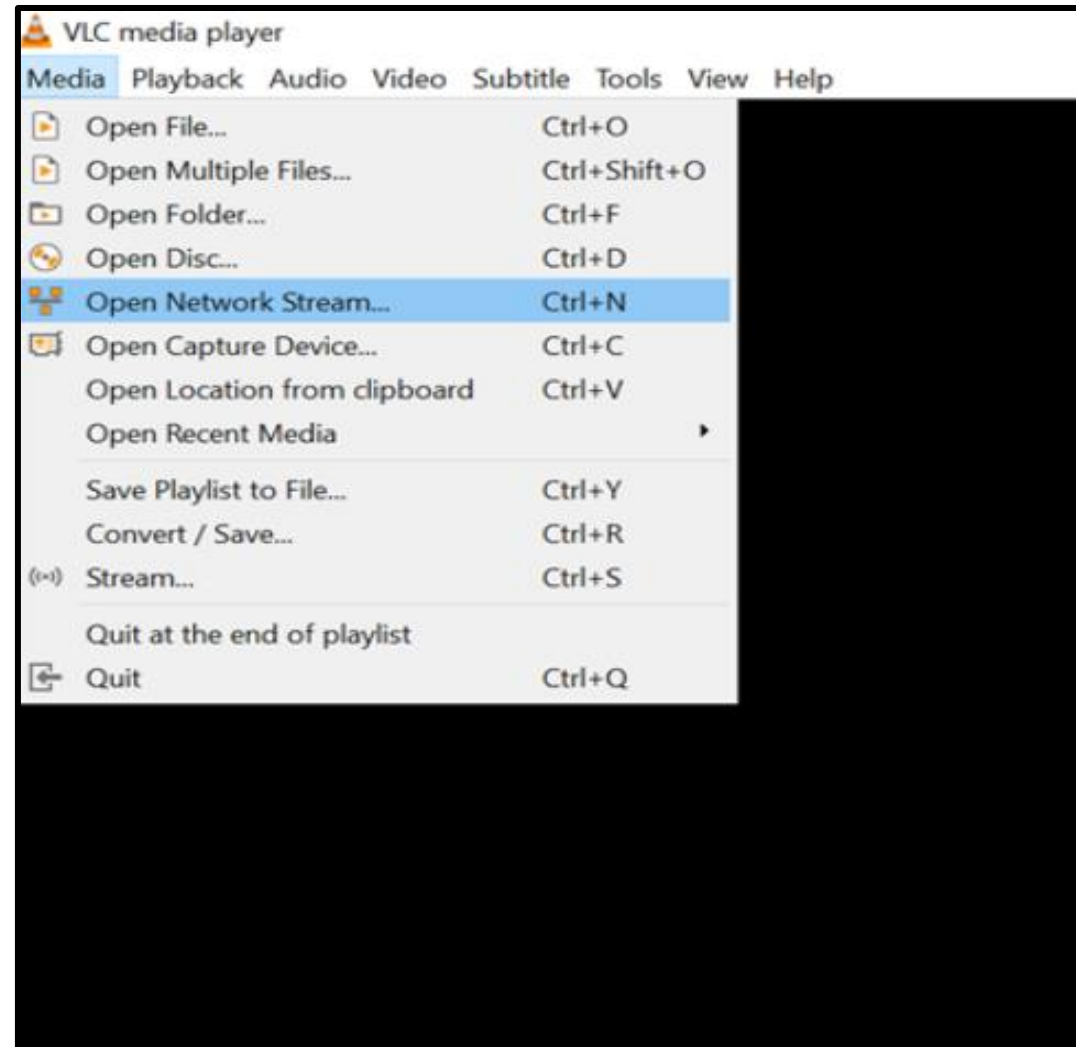


2.2 LoopPostProcessing

Step 4.

VLCメディアプレーヤーで以下のパスに従ってストリーミングを開始してください。

1. Media
2. Open Network Stream



2.2 LoopPostProcessing

Step 5.

コピーしたIPアドレスをVLCに貼り付けてください。それは以下の形式に従う必要があります。

rtsp://XXX.XXX.XXX.XXX:554)



LoopPostProcessing Mask

2.2 LoopPostProcessing

PostProcessingの用途は何ですか？

- 目的 - 検出された結果に対して後処理を行う
- **Mask** を用いて不要な部分を削除する

Motion Detection の応用

- スマートホームシステム：
 - 自動で照明のオンオフを切り替える
- 屋外環境監視：
 - 駐車場、工場などの動態を検知する
- オフィス：
 - 異常な動作を検知し、マークする

Maskの応用

- 特定のエリアでモーション検出を実行し、他のエリアは無視します。たとえば、背景の変化ではなく、ドアや窓の動的な変化のみに関心を持ちます。
- オフィスのプライベートデスクや家庭のプライベートエリアなど。これらのエリアを動体検知範囲から除外するためにMaskを設定します。

2.2 LoopPostProcessing

プログラミング

下記のコードをArduinoコードに追加してください。
アプリケーションでdefault maskを使用するためです。

結果は右のようになる



```
// Configure motion detection for low resolution RGB video stream
MD.configVideo(configMD);
MD.begin();
MD.setDetectionMask(mask);
```

2.2 LoopPostProcessing

プログラミング

Ctrlボタンを押し続け、MotionDetectionをクリックしてください。
すると、default maskの設定が表示されます。以下に示すように。

結果は右のようになる



```
LoopPostProcessing.ino  MotionDetection.h  〇
8  #include "VideoStream.h"
9  #include "StreamIO.h"
10 #include "RTSP.h"
11 #include "MotionDetection.h"
12 #include "VideoStreamOverlay.h"
13
14 #define CHANNEL 0 // High resolution vid
15 #define CHANNELMD 3 // RGB format video fo
16
17 .....
18
19 class MotionDetection
20
21 class MotionDetection : public MMFModule {}
22 MotionDetection MD;
```

2.2 LoopPostProcessing

.hファイルでdefault mask設定を確認
できます。

この設定は変更できません。

```
LoopPostProcessing.inc MotionDetection.h ⏏ ×
50 // Set a mask which would disable the motion detection for the left half of the screen
51 __attribute__((weak)) char mask[] = {
52     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
53     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
54     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
55     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
56     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
57     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
58     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
59     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
60     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
61     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
62     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
63     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
64     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
65     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
66     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
67     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
68     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
69     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
70 };
```

2.2 LoopPostProcessing

.hファイルからdefault maskをコピーし、それを.inoファイルに貼り付けてください。

```
LoopPostProcessing.ino MotionDetection.h
int status = WL_IDLE_STATUS;
__attribute__((weak)) char mask2[] = {
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
};
```

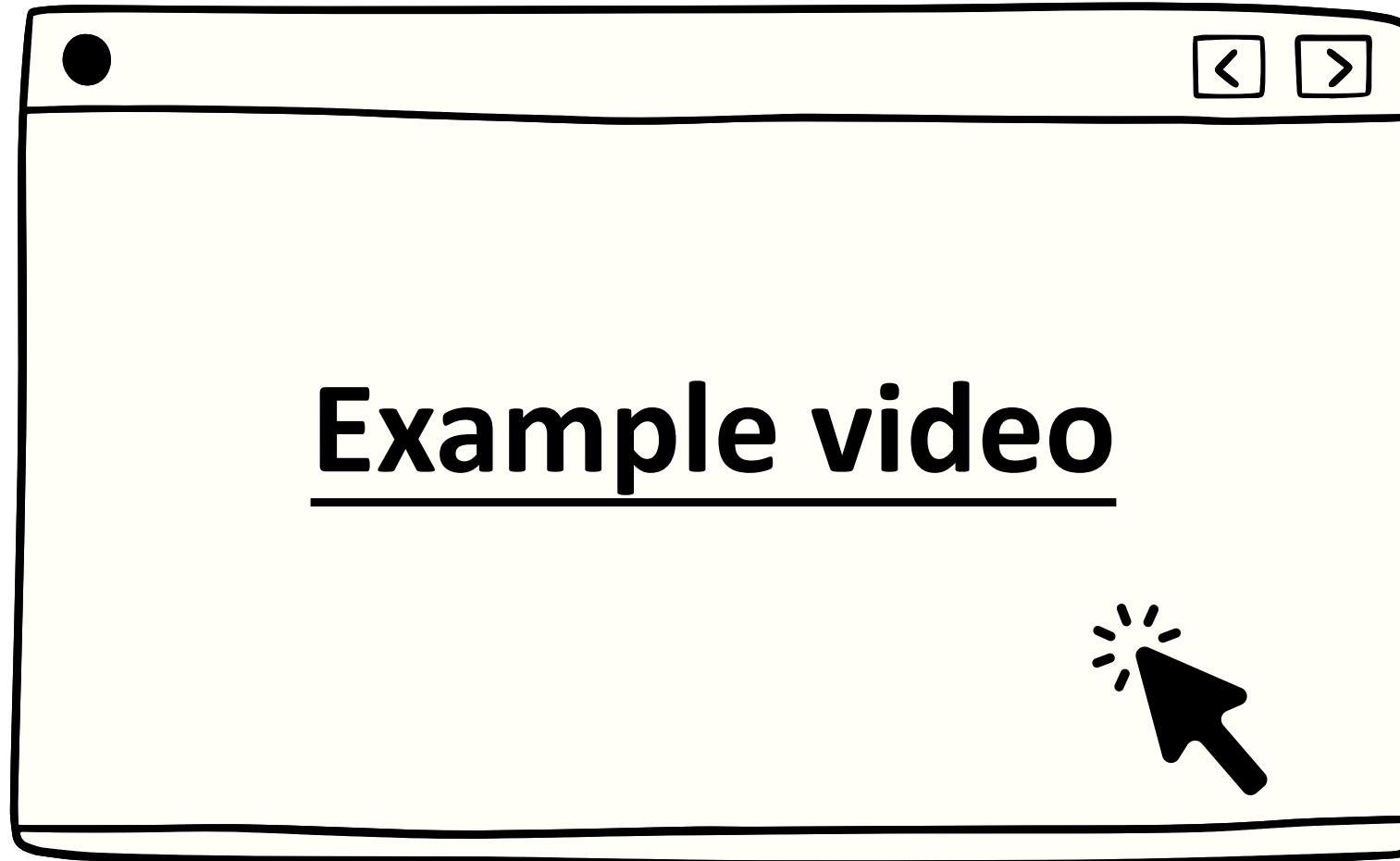



2.3 Audio Classification

Audio Classificationの応用

- スマートホームシステム：
 - 「電気を付けて」「電気を消して」などの音声コマンドを認識。
- 健康監視：
 - 患者の呼吸音、咳の音などを検出。

2.3 Audio Classification

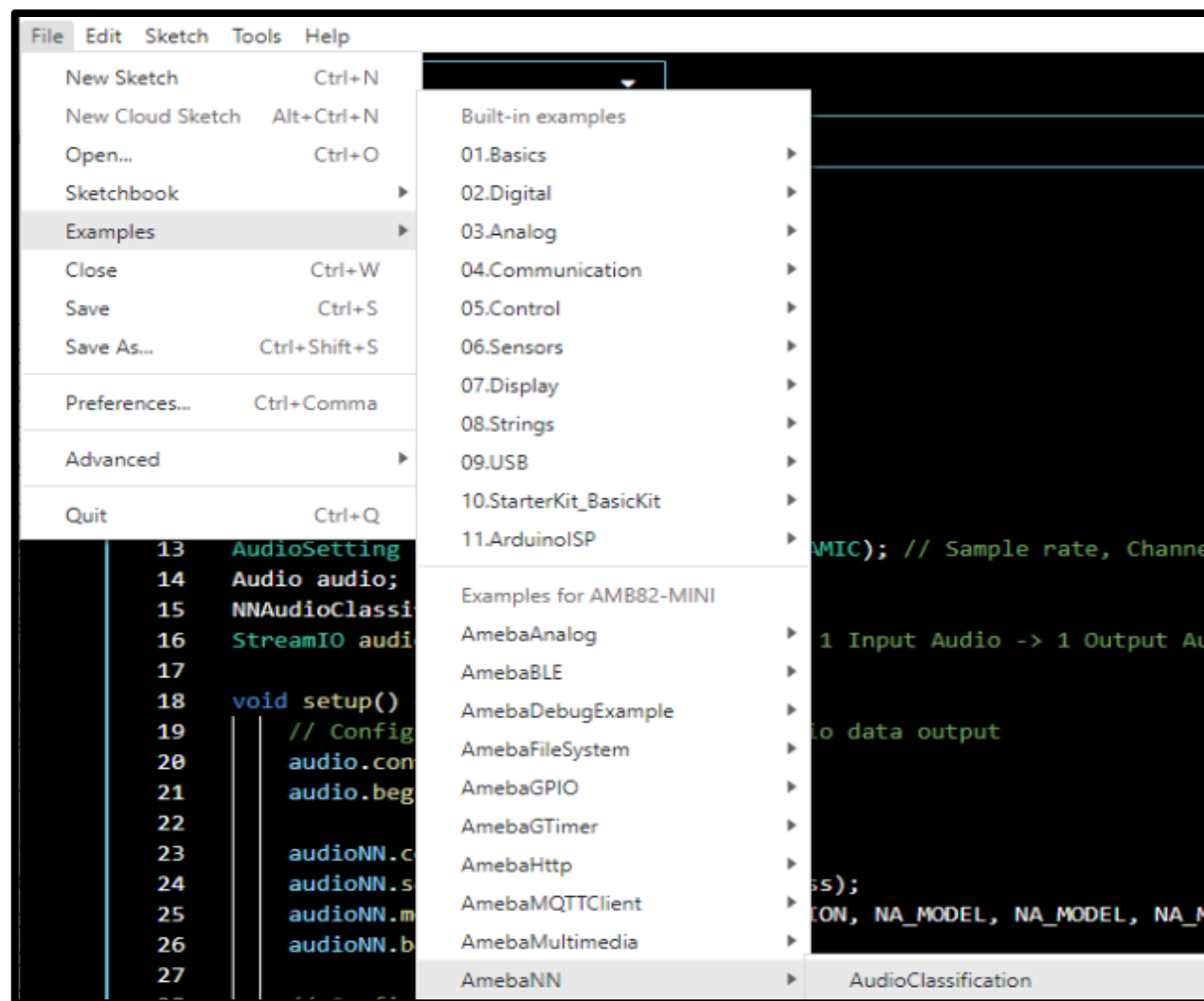


2.3 Audio Classification

Step 1.

Arduino IDEで例を開くには、以下のパスに従ってください。

1. File
2. Examples
3. AmebaNN
4. AudioClassification



2.3 Audio Classification

Step 2. モデルを選択(選択は任意です)

```
audioNN.configAudio(configA);  
audioNN.setResultCallback(ACPostProcess);  
audioNN.modelSelect(AUDIO_CLASSIFICATION, NA_MODEL, NA_MODEL, NA_MODEL, DEFAULT_YAMNET);  
audioNN.begin();
```

各種taskに対応するモデル一覧

```
Models  
=====  
YOLOv3 model      DEFAULT_YOLOV3TINY / CUSTOMIZED_YOLOV3TINY  
YOLOv4 model      DEFAULT_YOLOV4TINY / CUSTOMIZED_YOLOV4TINY  
YOLOv7 model      DEFAULT_YOLOV7TINY / CUSTOMIZED_YOLOV7TINY  
SCRFD model       DEFAULT_SCRFD      / CUSTOMIZED_SCRFD  
MobileFaceNet model DEFAULT_MOBILEFACENET/ CUSTOMIZED_MOBILEFACENET  
No model          NA_MODEL
```

2.3 Audio Classification

結果: Serial Monitorで検出された音を受信できます。

```
Serial Monitor x Output
Message (Enter to send message to 'AMB82-MINI' on 'COM16')
YAMNET tick[0] = 100
No of Audio Detected = 0
YAMNET tick[0] = 100
No of Audio Detected = 2
0 class 393, score: 73, audio name: Smoke detector, smoke alarm
1 class 475, score: 72, audio name: Beep, bleep
YAMNET tick[0] = 100
No of Audio Detected = 1
0 class 475, score: 74, audio name: Beep, bleep
YAMNET tick[0] = 100
No of Audio Detected = 2
0 class 393, score: 76, audio name: Smoke detector, smoke alarm
1 class 475, score: 75, audio name: Beep, bleep
YAMNET tick[0] = 101
No of Audio Detected = 0
YAMNET tick[0] = 101
No of Audio Detected = 1
0 class 494, score: 69, audio name: Silence
YAMNET tick[0] = 101
No of Audio Detected = 1
0 class 494, score: 69, audio name: Silence
```

2.3 Audio Classification

- 事前に訓練されたモデルは、**521種類の異なる音声を識別できます。**
- 特定の音声の識別を無効にするには、**filterを0に設定**してください。

```
AudioClassification.ino  AudioClassList.h
1  #ifndef __AUDIOCLASSLIST_H__
2  #define __AUDIOCLASSLIST_H__
3
4
5  struct AudioDetectionItem {
6      uint32_t index;
7      const char* audioName;
8      uint8_t filter;
9  };
10
11  //// List of audio the pre-trained model is capable of recognizing
12  //// Index number is fixed and hard-coded from training
13  //// Set the filter value to 0 to ignore any recognized audios
14  AudioDetectionItem audioNames[521] = {
15      {0, "Speech", 0},
16      {1, "Child speech, kid speaking", 1},
17      {2, "Conversation", 1},
18      {3, "Narration, monologue", 1},
19      {4, "Babbling", 1},
20      {5, "Speech synthesizer", 1},
21      {6, "Shout", 1},
22      {7, "Bellow", 1},
23      {8, "Whoop", 1},
24      {9, "Yell", 1},
25      {10, "Children shouting", 1},
26      {11, "Screaming", 1},
27      {12, "Whispering", 1},
```

2.3 Audio Classification

結果表示機能を追加する

2.3 Audio Classification

実装には、以下の3点をコードに追加する必要があります。

1.プログラムの最初に追加:

ピンを定義する

```
int output0 = 0 ;  
int output1 = 1 ;  
int output2 = 2 ;  
int output3 = 3 ;  
int output4 = 4 ;
```

2.3 Audio Classification

実装には、以下の3点をコードに追加する必要があります。

**2. 関数void setup()に追加:
出力を定義されたピンに送る**

```
pinMode(output0, OUTPUT);  
pinMode(output1, OUTPUT);  
pinMode(output2, OUTPUT);  
pinMode(output3, OUTPUT);  
pinMode(output4, OUTPUT);
```

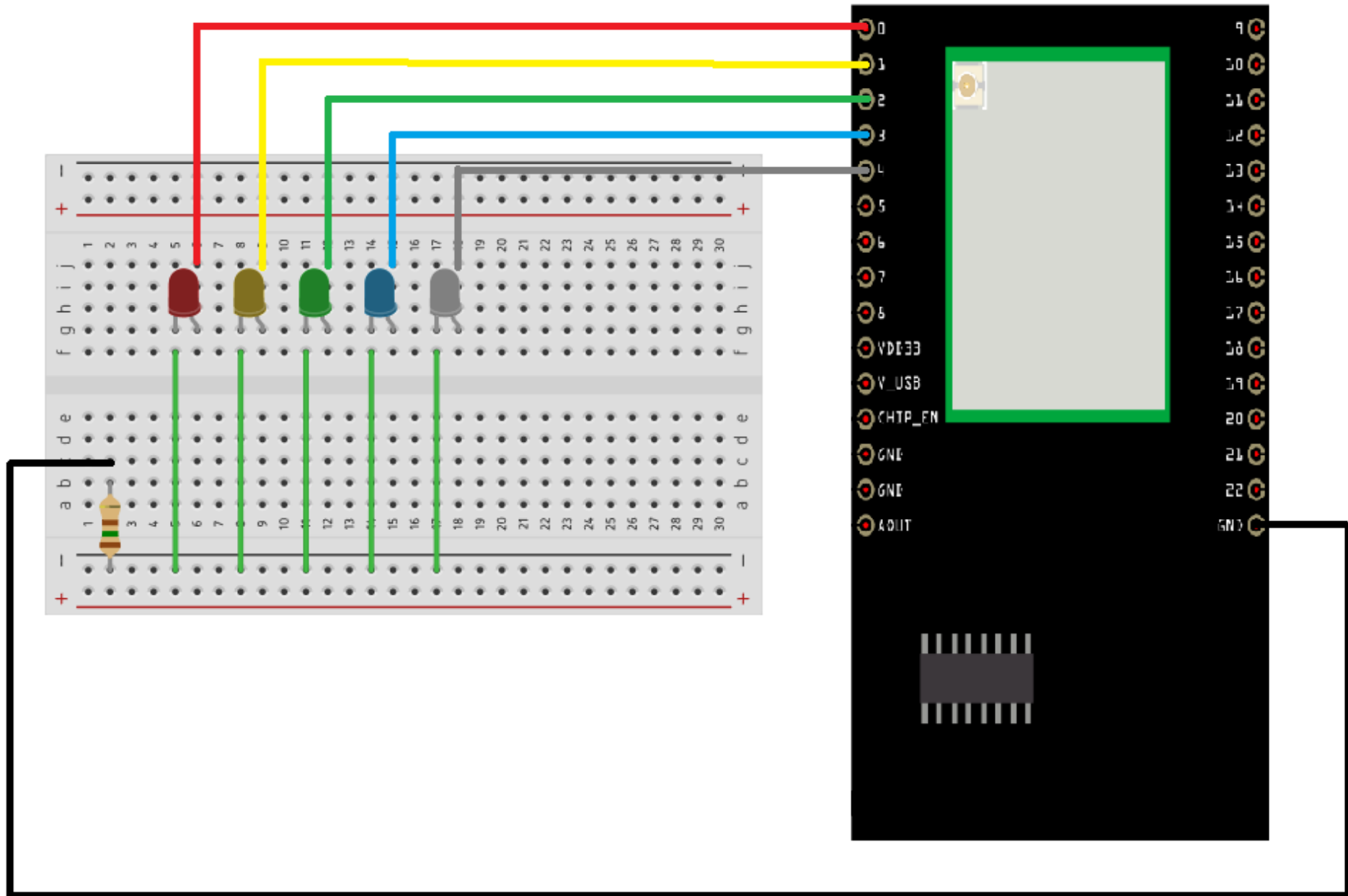
2.3 Audio Classification

3.関数void loop()に追加: 検出された結果がどの指を示しているか判断する

```
if(obj_type==0) //speech
{
    digitalWrite(output0, HIGH);
    delay(1000);
    digitalWrite(output0, LOW);
    delay(1000);
}

else if(obj_type==1) //child speech
{
    digitalWrite(output1, HIGH);
    delay(1000);
    digitalWrite(output1, LOW);
    delay(1000);
}
```

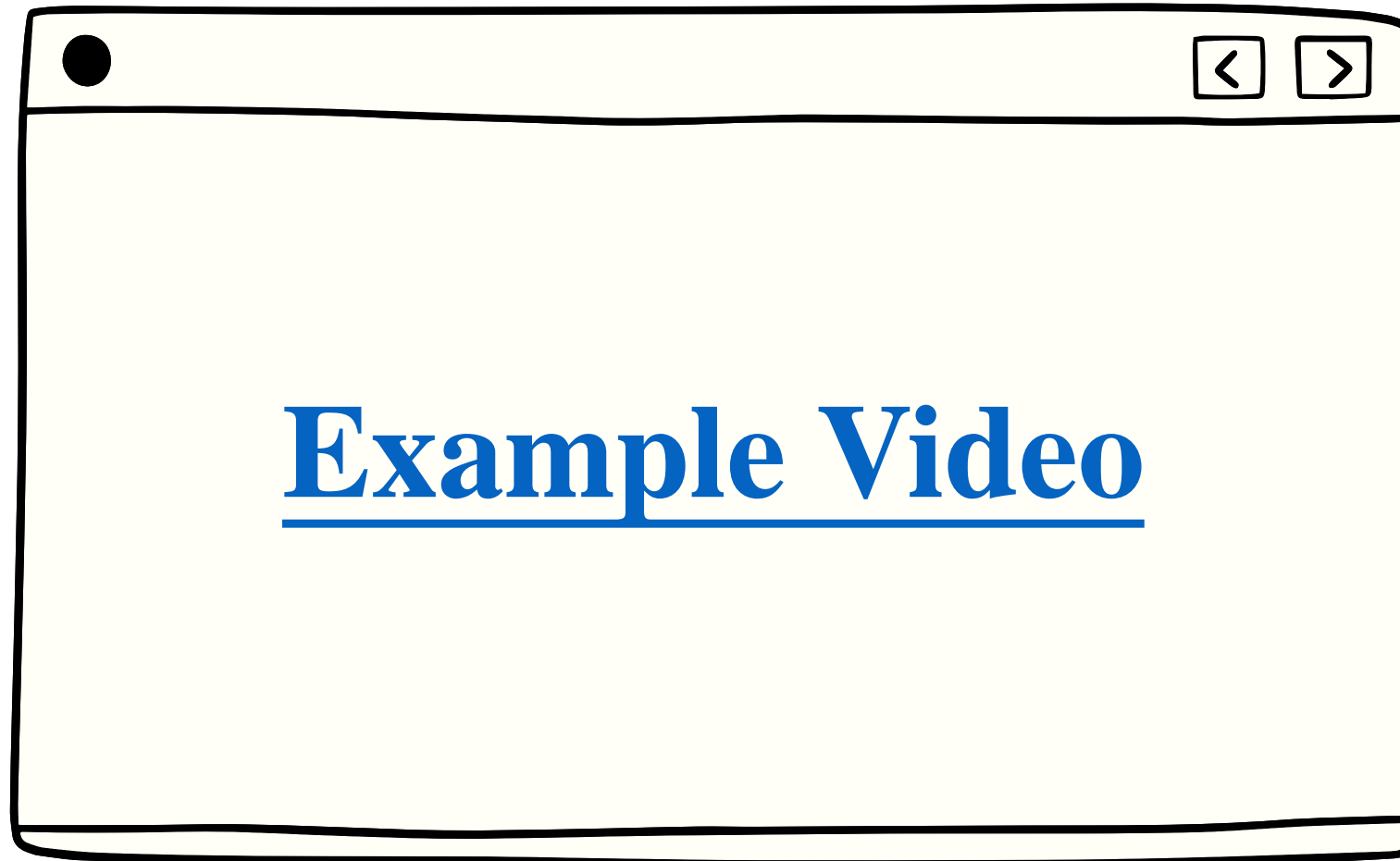
```
else if(obj_type==2)//conversation
{
    digitalWrite(output2, HIGH);
    delay(1000);
    digitalWrite(output2, LOW);
    delay(1000);
}
else if(obj_type==3) //Narration
{
    digitalWrite(output3, HIGH);
    delay(1000);
    digitalWrite(output3, LOW);
    delay(1000);
}
else if(obj_type==4) //Babbling
{
    digitalWrite(output4, HIGH);
    delay(1000);
    digitalWrite(output4, LOW);
    delay(1000);
}
```





2.4 FaceRecognition

2.4 Face Recognition



2.4 Face Recognition

Face Recognition 技術の基礎

1. 顔検出:

- 画像やビデオ内の**顔領域**を検出します。

2. Features extraction:

- これらの特徴には、顔の輪郭、目の位置、鼻の形などが含まれます 特徴マッチング。

3. Features matching:

- **抽出された特徴を既知の顔特徴**と比較し、2つの特徴ベクトル間の類似性を評価します。

2.4 Face Recognition

Face Recognitionの応用

- アクセス制御システム:
 - 家庭用アクセス制御システム
 - 企業や学校での出勤システム
- セキュリティ監視:
 - ブラックリストデータベースとの連携
 - 身元認証

2.4 Face Recognition

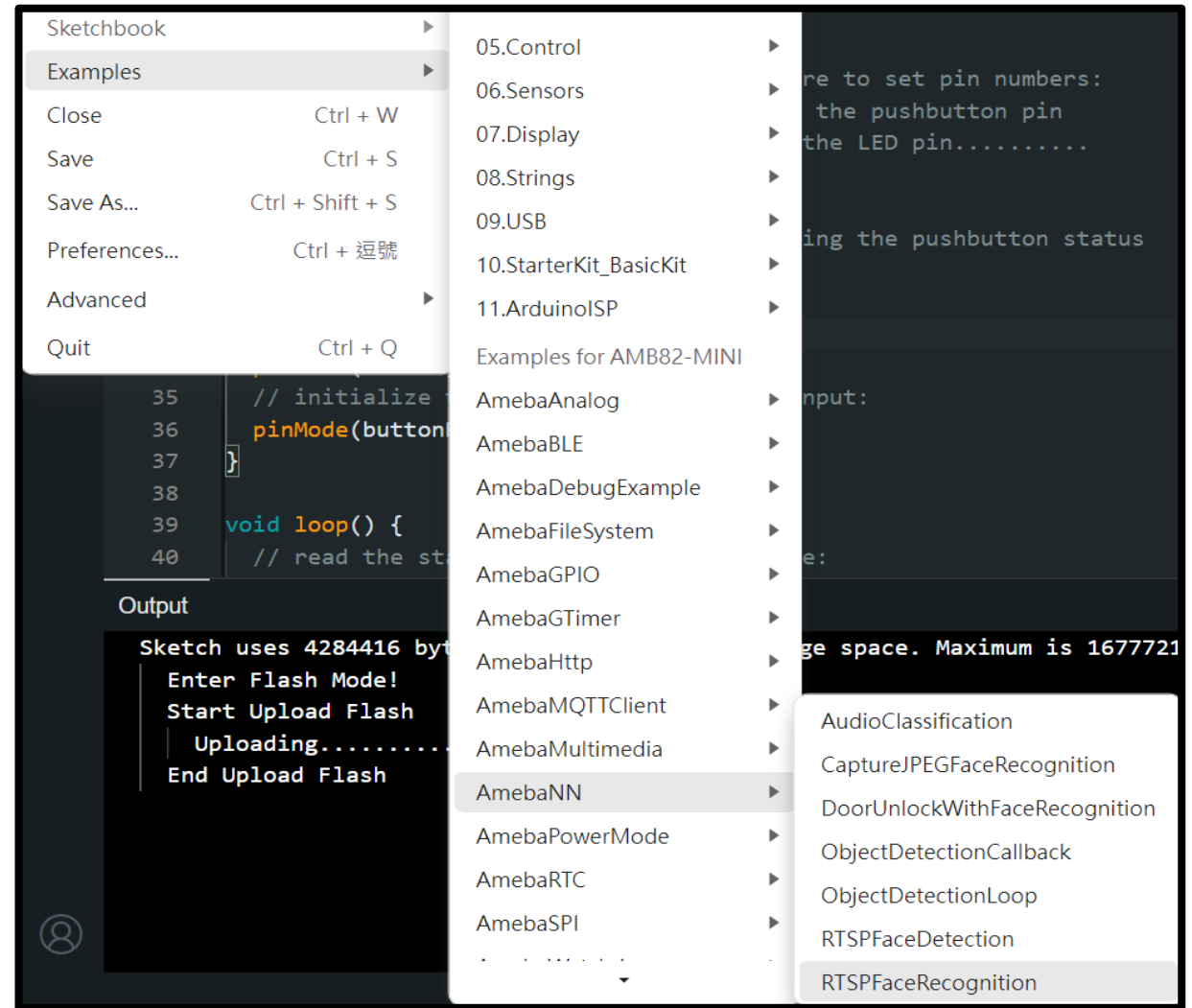
Implementation

2.4 Face Recognition

Step 1.

Arduino IDEで例を開くには、以下のパスに従ってください。

1. File
2. Examples
3. AmebaNN
4. RTSPFaceRecognition



2.4 Face Recognition

Step 2.

プログラムのWi-Fi接続設定に、
SSIDとパスワードを設定してくだ
さい。

```
#include <WiFi.h>
#include "StreamIO.h"
#include "VideoStream.h"
#include "RTSP.h"
#include "NNObjectDetection.h"
#include "VideoStreamOverlay.h"
#include "ObjectClassList.h"

#define CHANNEL 0
#define CHANNELLN 3

// Lower resolution for NN processing
#define NNWIDTH 576
#define NNHEIGHT 320

VideoSetting config(VIDEO_FHD, 30, VIDEO_H264, 0);
VideoSetting configNN(NNWIDTH, NNHEIGHT, 10, VIDEO_RGB, 0);
NNObjectDetection objDet;
RTSP rtsp;
StreamIO videoStreamer(1, 1);
StreamIO videoStreamerNN(1, 1);

char ssid[] = "Network_SSID"; // your network SSID (name)
char pass[] = "Password"; // your network password
int status = WL_IDLE_STATUS;

IPAddress ip;
int rtsp_port;

void setup() {
  Serial.begin(115200);

  // attempt to connect to Wifi network:
```

WiFiの名前とパスワード
を入力してください

2.4 Face Recognition

Step 3.モデルを選択(選択は任意です)

```
// Select neural network task and models
facerecog.configVideo(configNN);
facerecog.modelSelect(FACE_RECOGNITION, NA_MODEL, DEFAULT_SCRFD, DEFAULT_MOBILEFACENET);
facerecog.begin();
facerecog.setResultCallback(FRPostProcess);
```

各種taskに対応するモデル一覧

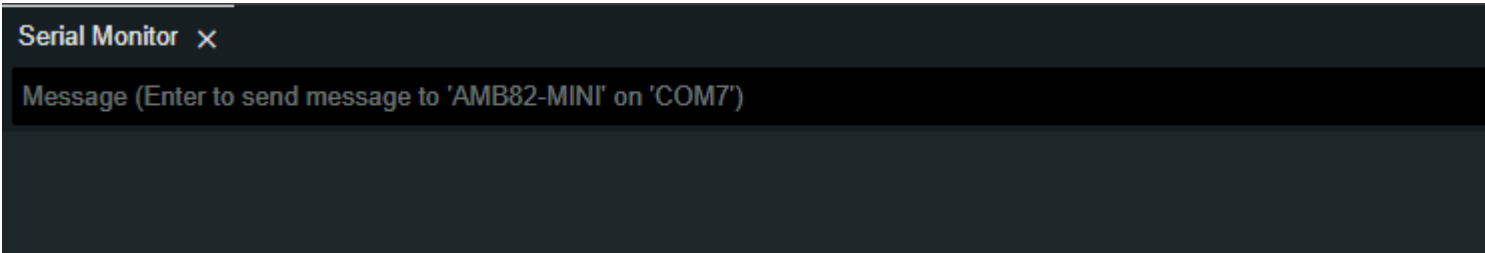
```
Models
=====
YOLOv3 model      DEFAULT_YOLOV3TINY / CUSTOMIZED_YOLOV3TINY
YOLOv4 model      DEFAULT_YOLOV4TINY / CUSTOMIZED_YOLOV4TINY
YOLOv7 model      DEFAULT_YOLOV7TINY / CUSTOMIZED_YOLOV7TINY
SCRFD model        DEFAULT_SCRFD / CUSTOMIZED_SCRFD
MobileFaceNet model DEFAULT_MOBILEFACENET/ CUSTOMIZED_MOBILEFACENET
No model           NA_MODEL
```

2.4 Face Recognition

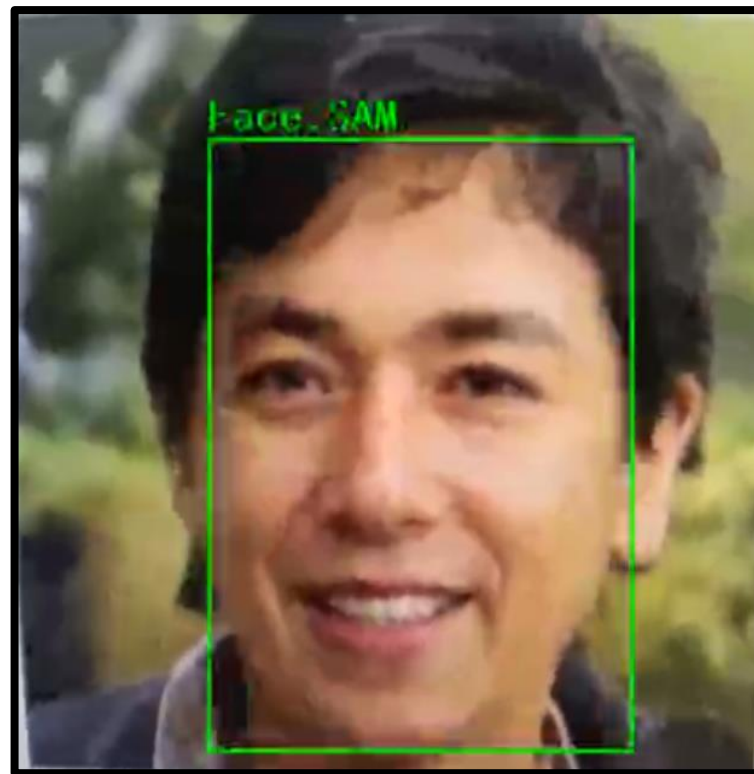
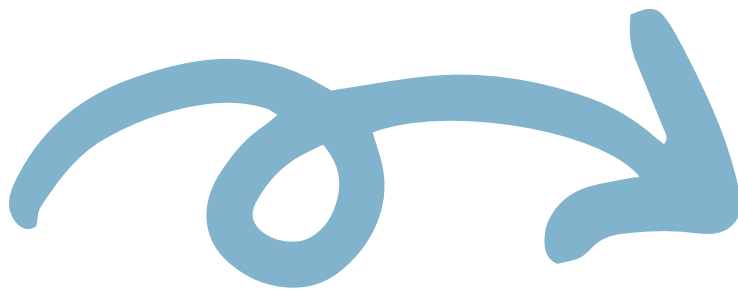
DEMO

2.4 Face Recognition

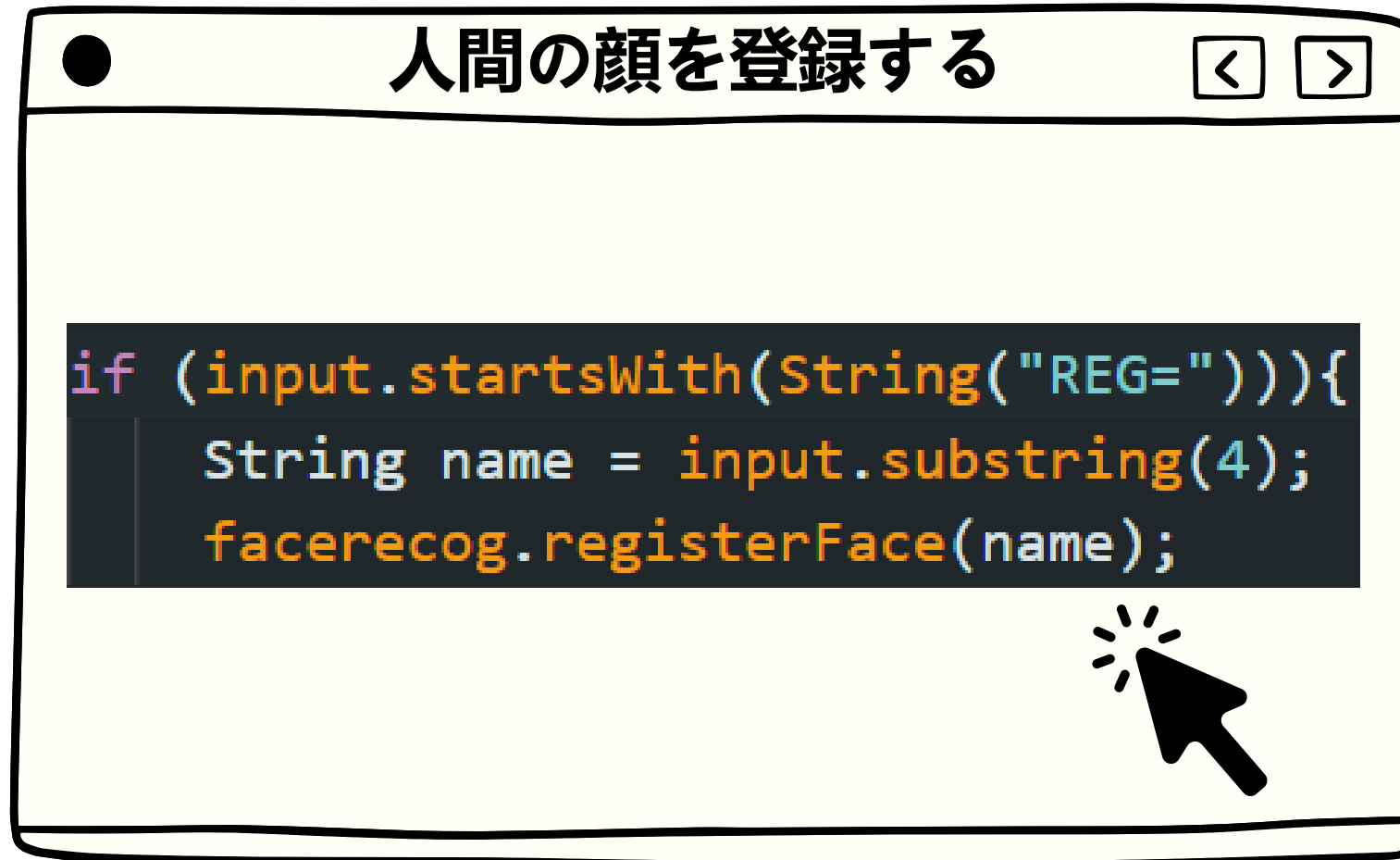
以降のすべての操作は、**Serial monitor message box**に表示されます。



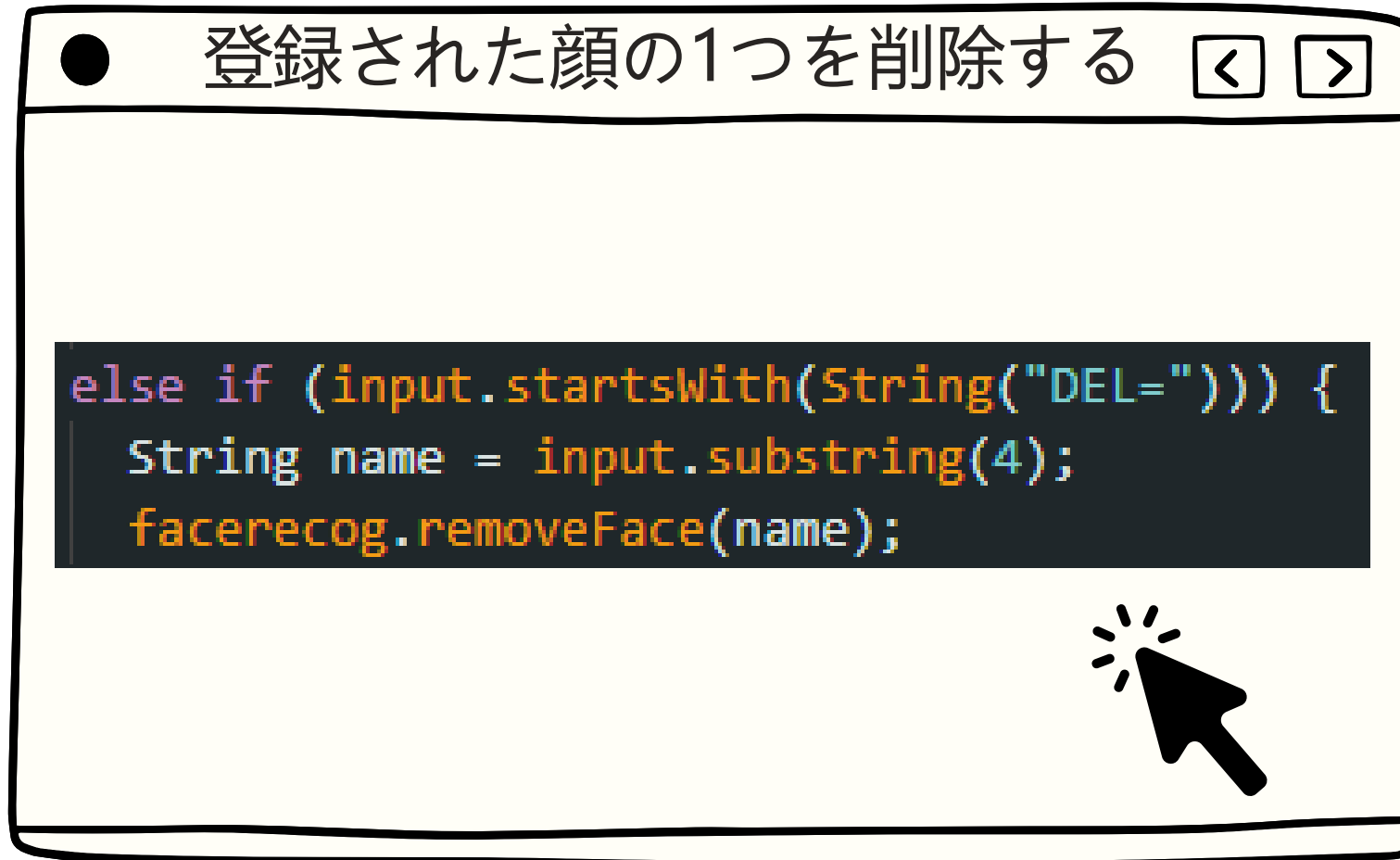
2.4 Face Recognition



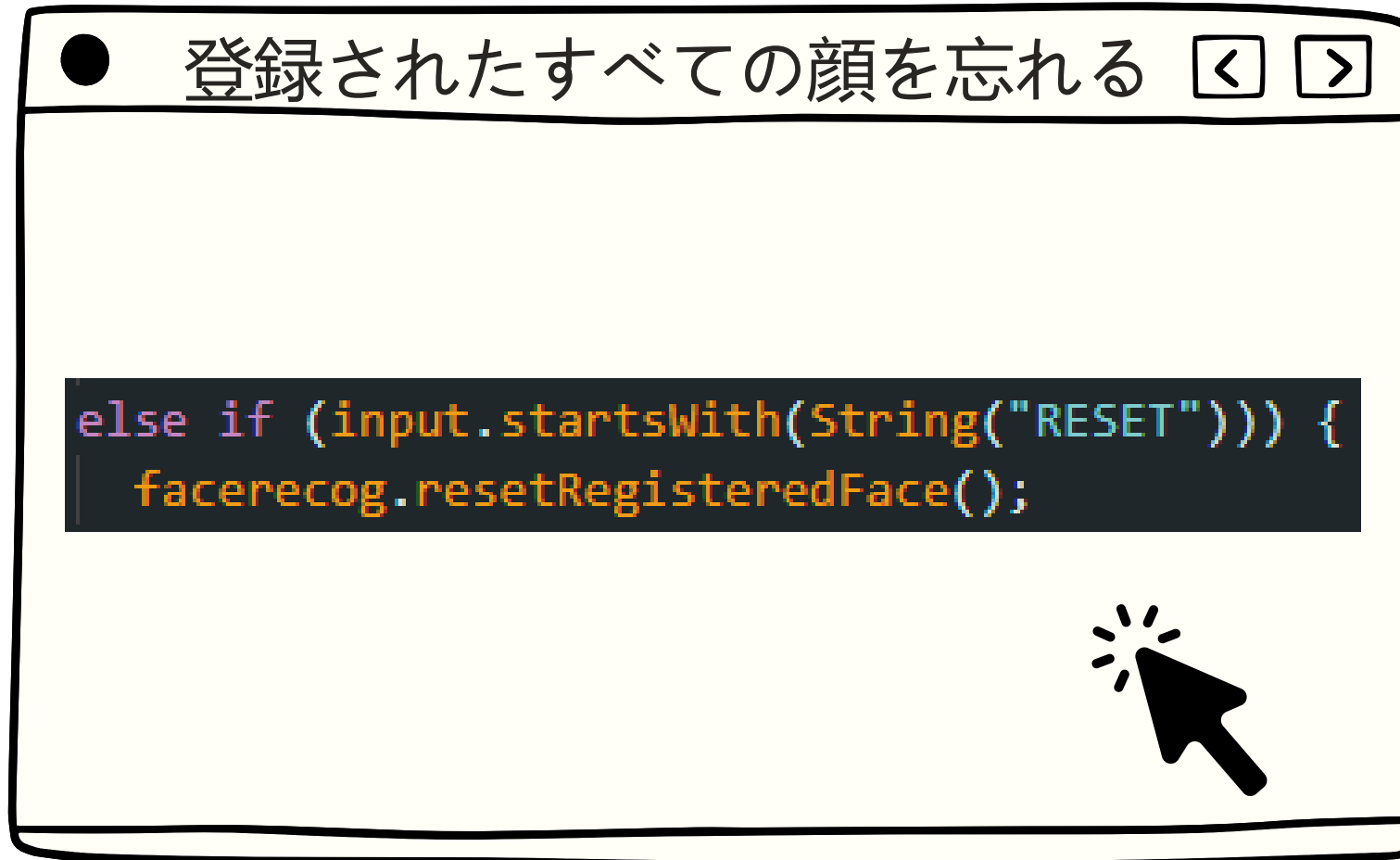
2.4 Face Recognition



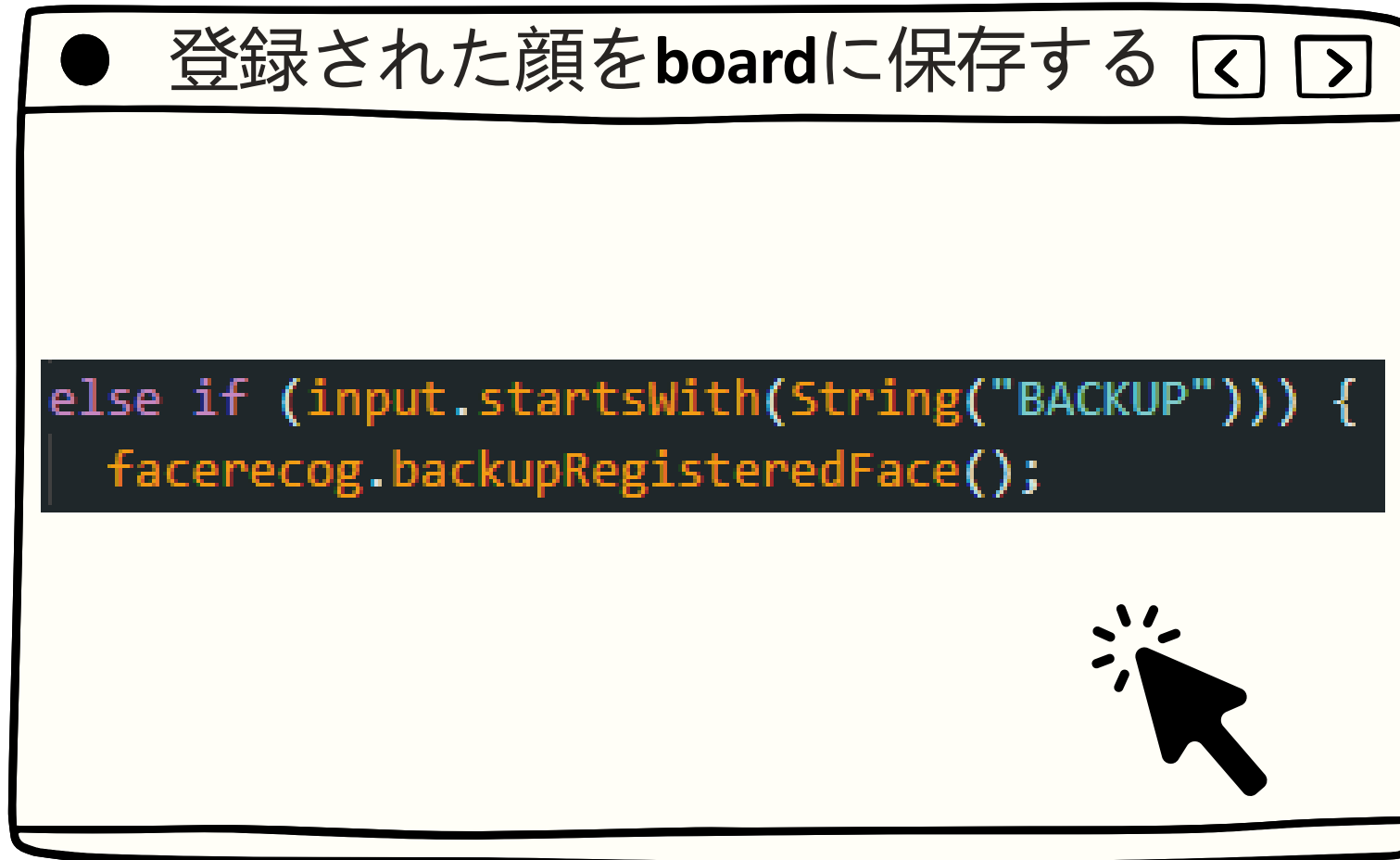
2.4 Face Recognition



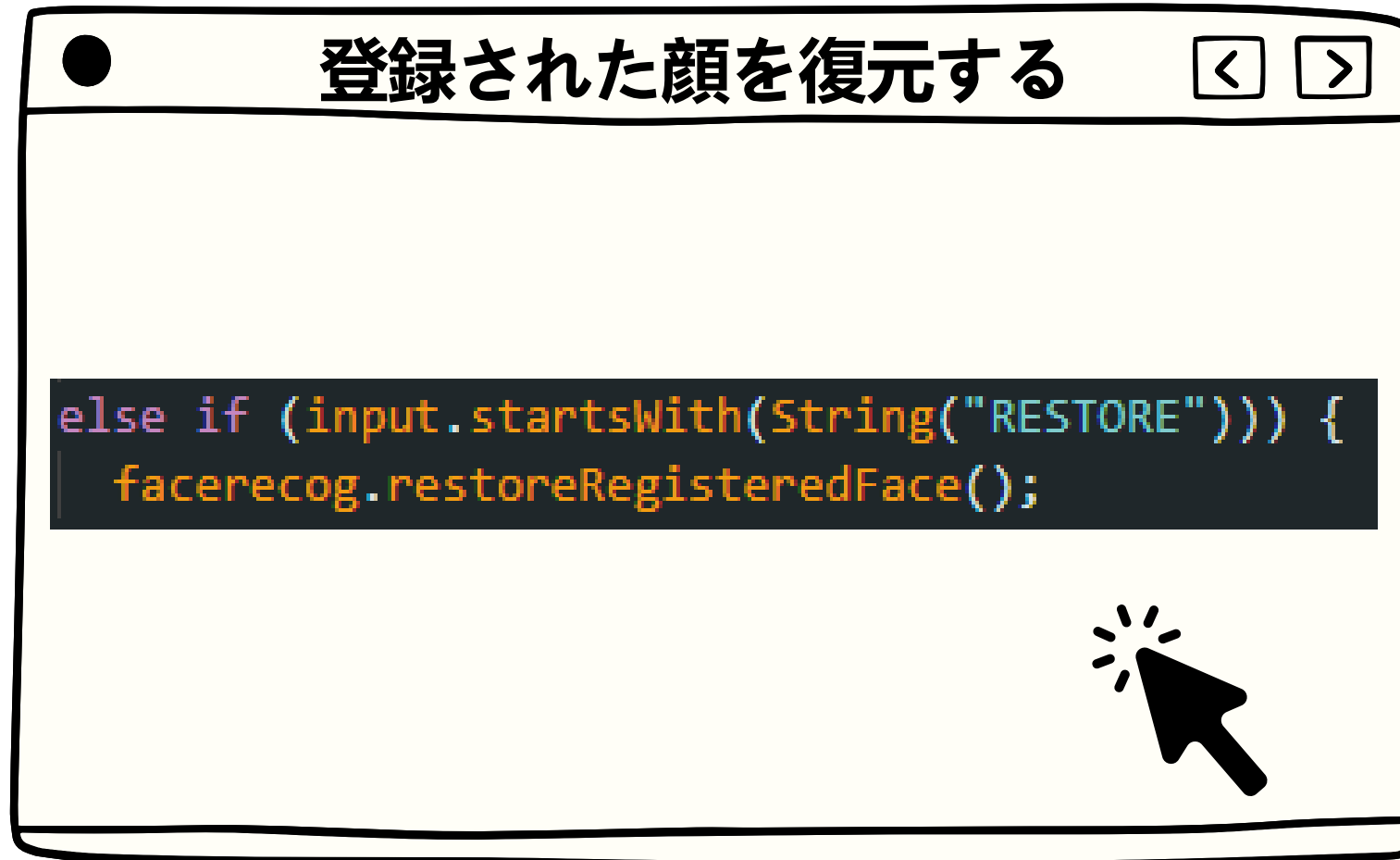
2.4 Face Recognition



2.4 Face Recognition



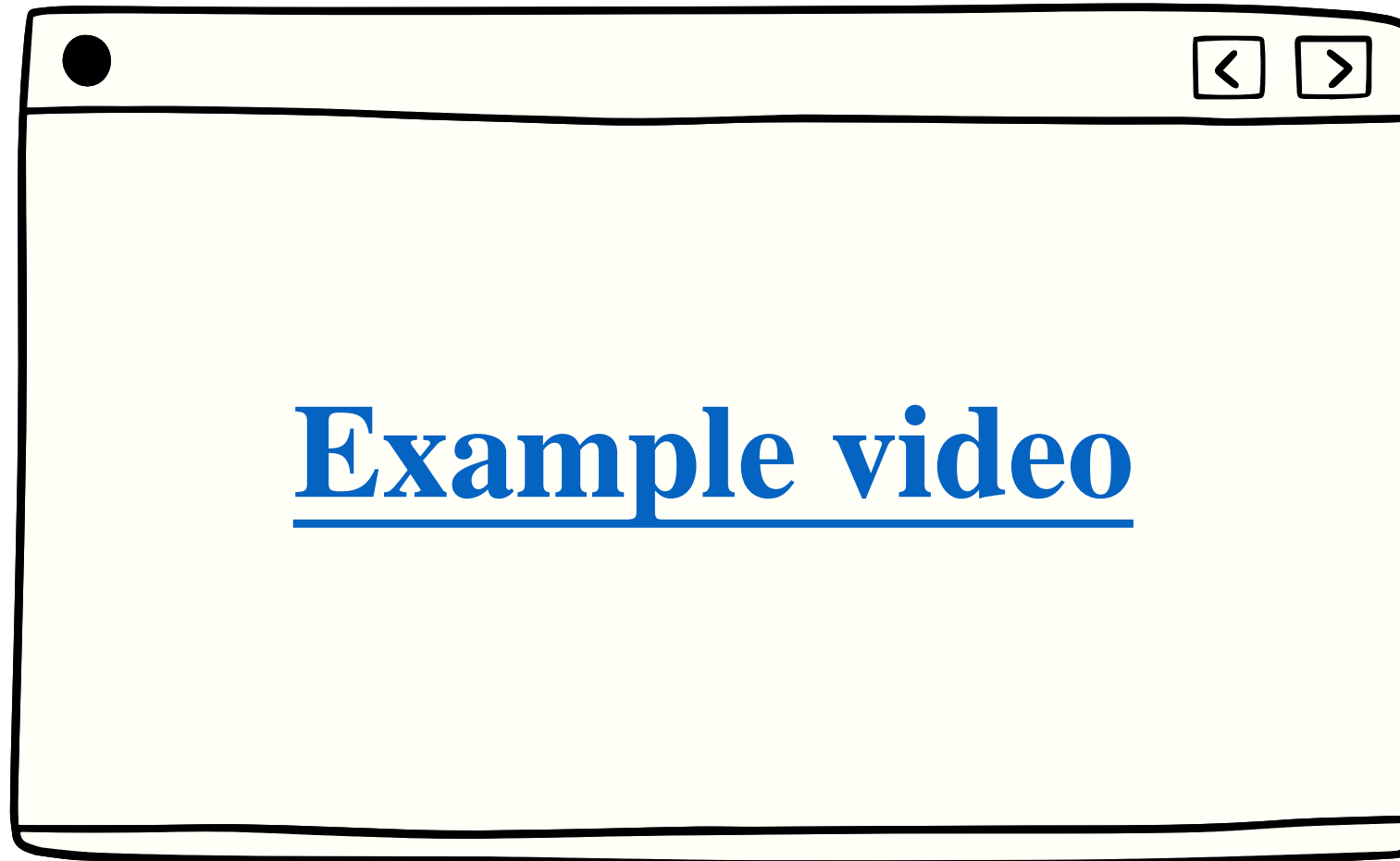
2.4 Face Recognition





2.5 Image Classification

2.4 Face Recognition



2.5 Image Classification

画像分類の基本概念

- 入力された画像から、その画像の主要な被写体が事前に設定されたどのカテゴリに属するかをモデルが判別します。例えば、猫と犬の分類では、モデルは入力された画像が猫なのか犬なのかを判断します。

2.5 Image Classification

Input



猫

Learning

モデルを学習する

Output

Class: 犬



犬



データ前処理

- データ前処理 定義: データ分析、モデリング、または機械学習を実行する前に、生データをクリーニング、変換、および整理するプロセス。
- 目的: データの**品質**と**一貫性**を確保し、不確実性を減らし、データをその後の分析やトレーニングに**適したものに**する。

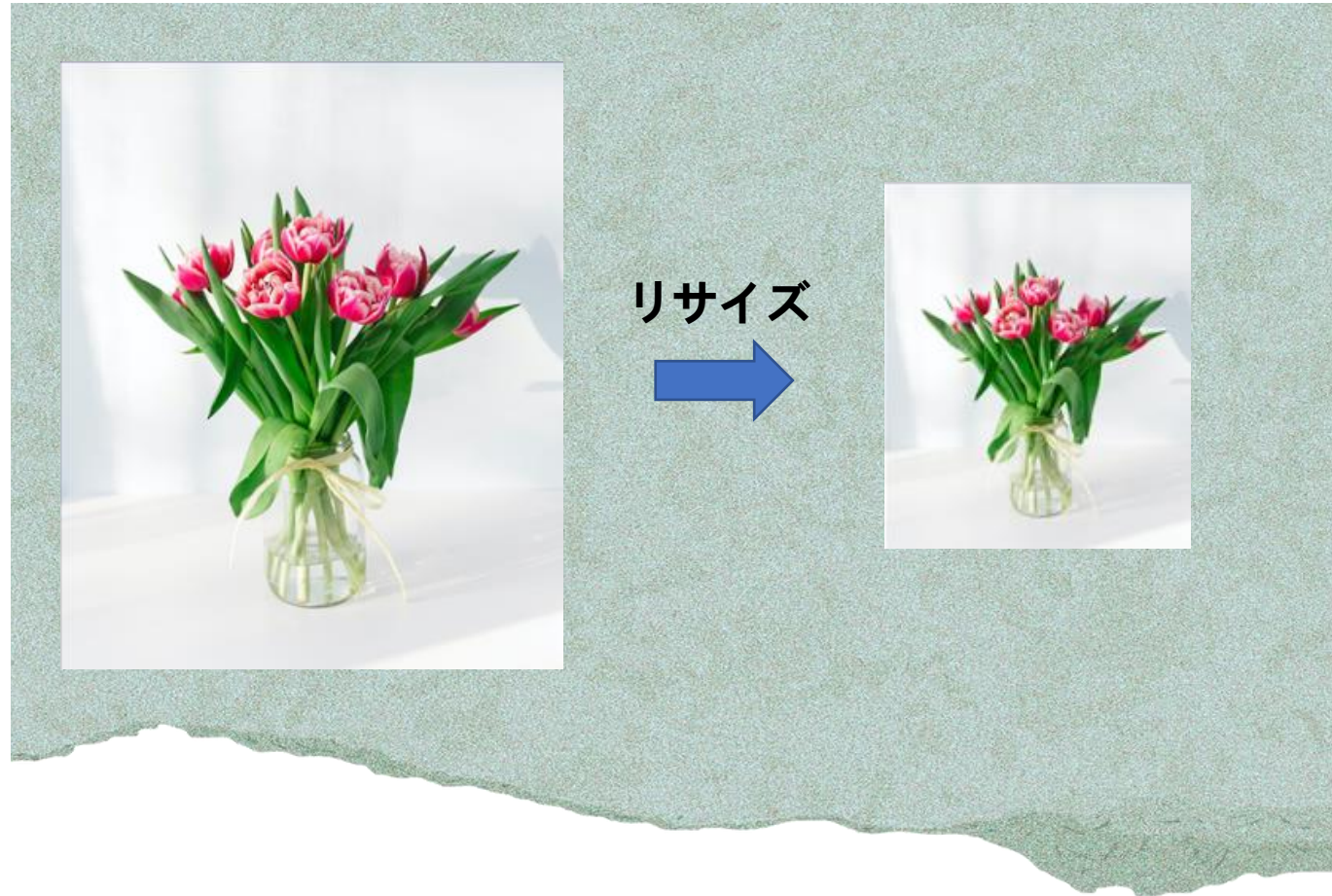
2.5 Image Classification

データ前処理

- リサイズ
- クロッピング
- 正規化
- カラー変換

2.5 Image Classification

データ前処理-リサイズ



2.5 Image Classification

データ前処理-クロッピング



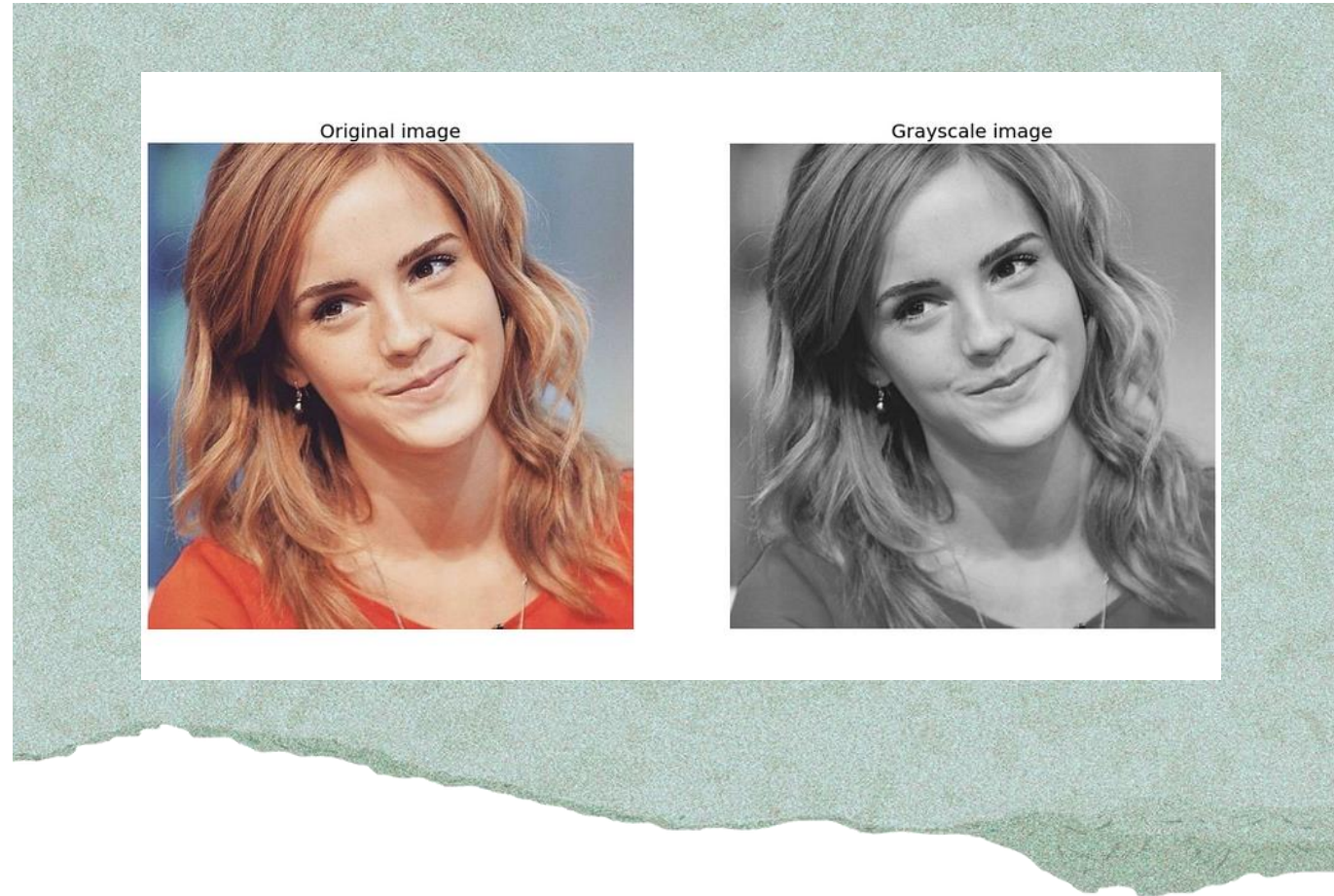
2.5 Image Classification

データ前処理-正規化

- **定義:正規化**とは、画像のピクセル値を標準範囲に変換することです。通常は $[0,1]$ または $[-1,1]$ です。

2.5 Image Classification

データ前処理-カラー変換



データ拡張

- 定義:元のデータにさまざまなランダム変換と処理を行い、より多くの訓練サンプルを作成します。
- 目的:データの多様性を増やし、モデルの汎化能力を向上させ、**過学習**を抑制すること。

2.5 Image Classification

データ拡張

- 回転
- 平行移動
- 反転
- リスケーリング
- クロッピング

2.5 Image Classification

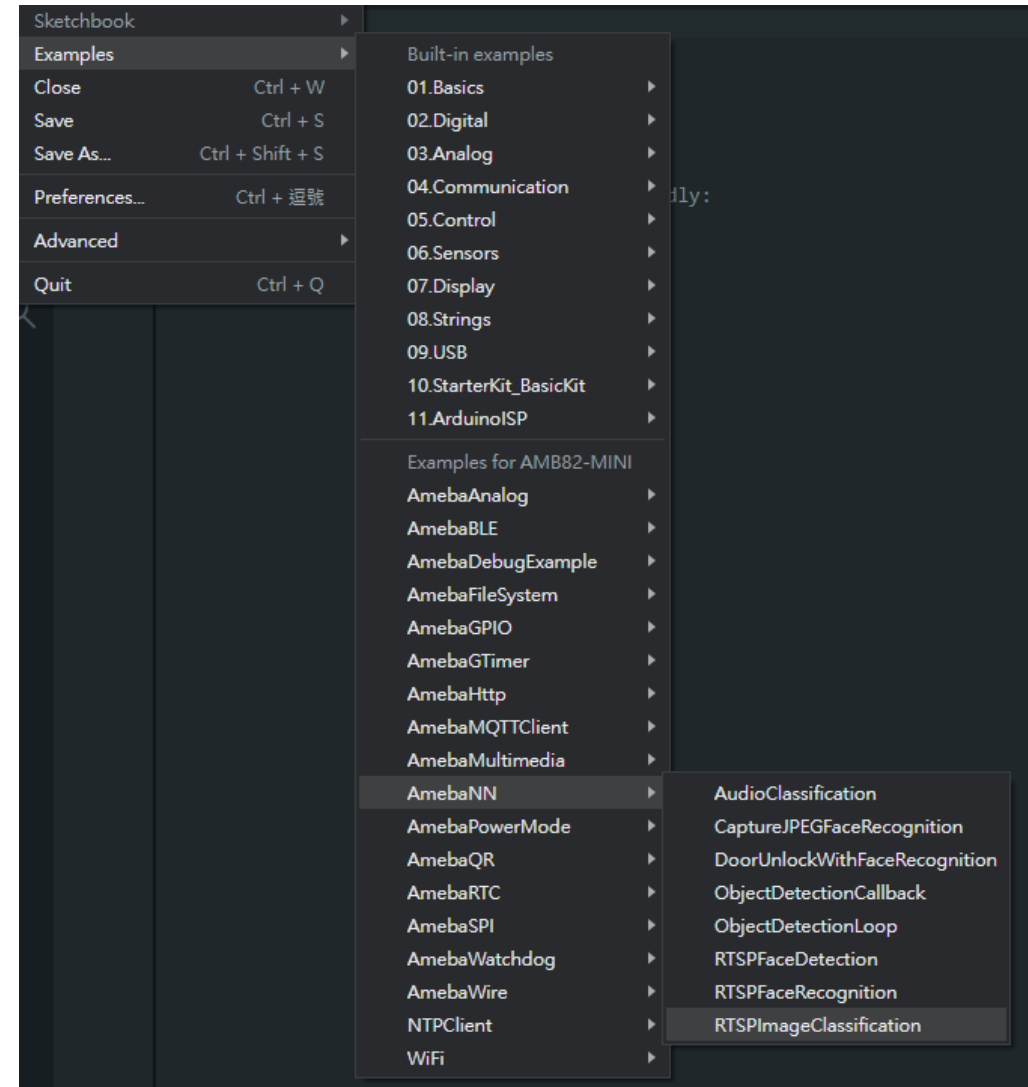
Implementation

2.5 Image Classification

Step 1.

Arduino IDEで例を開くには、以下のパスに従ってください。

1. File
2. Examples
3. AmebaNN
4. RTSPImageClassification



2.5 Image Classification

Step 2.

プログラムのWi-Fi接続設定に、
SSIDとパスワードを設定してくだ
さい。

```
#include <WiFi.h>
#include "StreamIO.h"
#include "VideoStream.h"
#include "RTSP.h"
#include "NNObjectDetection.h"
#include "VideoStreamOverlay.h"
#include "ObjectClassList.h"

#define CHANNEL 0
#define CHANNELLN 3

// Lower resolution for NN processing
#define NNWIDTH 576
#define NNHEIGHT 320

VideoSetting config(VIDEO_FHD, 30, VIDEO_H264, 0);
VideoSetting configNN(NNWIDTH, NNHEIGHT, 10, VIDEO_RGB, 0);
NNObjectDetection objDet;
RTSP rtsp;
StreamIO videoStreamer(1, 1);
StreamIO videoStreamerNN(1, 1);

char ssid[] = "Network_SSID"; // your network SSID (name)
char pass[] = "Password"; // your network password
int status = WL_IDLE_STATUS;

IPAddress ip;
int rtsp_port;

void setup() {
  Serial.begin(115200);

  // attempt to connect to Wifi network:
```

WiFiの名前とパスワード
を入力してください

2.5 Image Classification

Step 3.モデルを選択(選択は任意です)

```
imgclass.configVideo(configNN);  
imgclass.configInputImageColor(IMG_RGB);  
imgclass.setResultCallback(ICPostProcess);  
imgclass.modelSelect(IMAGE_CLASSIFICATION, NA_MODEL, NA_MODEL, NA_MODEL, NA_MODEL, DEFAULT_IMGCLASS);  
imgclass.begin();
```

各種taskに対応するモデル一覧

```
Models  
=====  
YOLOv3 model      DEFAULT_YOLOV3TINY / CUSTOMIZED_YOLOV3TINY  
YOLOv4 model      DEFAULT_YOLOV4TINY / CUSTOMIZED_YOLOV4TINY  
YOLOv7 model      DEFAULT_YOLOV7TINY / CUSTOMIZED_YOLOV7TINY  
SCRFD model       DEFAULT_SCRFD      / CUSTOMIZED_SCRFD  
MobileFaceNet model DEFAULT_MOBILEFACENET / CUSTOMIZED_MOBILEFACENET  
YAMNET model      DEFAULT_YAMNET     / CUSTOMIZED_YAMNET  
CNN model         DEFAULT_IMGCLASS   / CUSTOMIZED_IMGCLASS
```



2.6 MQTT ON AMB82

2.6 MQTT ON AMB82

MQTT

- **定義:** Message Queuing Telemetry Transport (MQTT) は、制約のあるデバイスおよび低帯域幅、高遅延のネットワーク向けに特別に設計された**軽量メッセージングプロトコル**です。

MQTTの主な特徴

- **出版者/購読者**パターンに従う:
 - **出版者:**特定のトピックに対して**情報を発行**します。
 - **購読者:**特定のトピックを購読し、そのトピックに関連する**情報を受信**します。
 - **ブローカー:**出版者と購読者の間の仲介役となり、メッセージの送受信を管理します。

MQTTの主な特徴

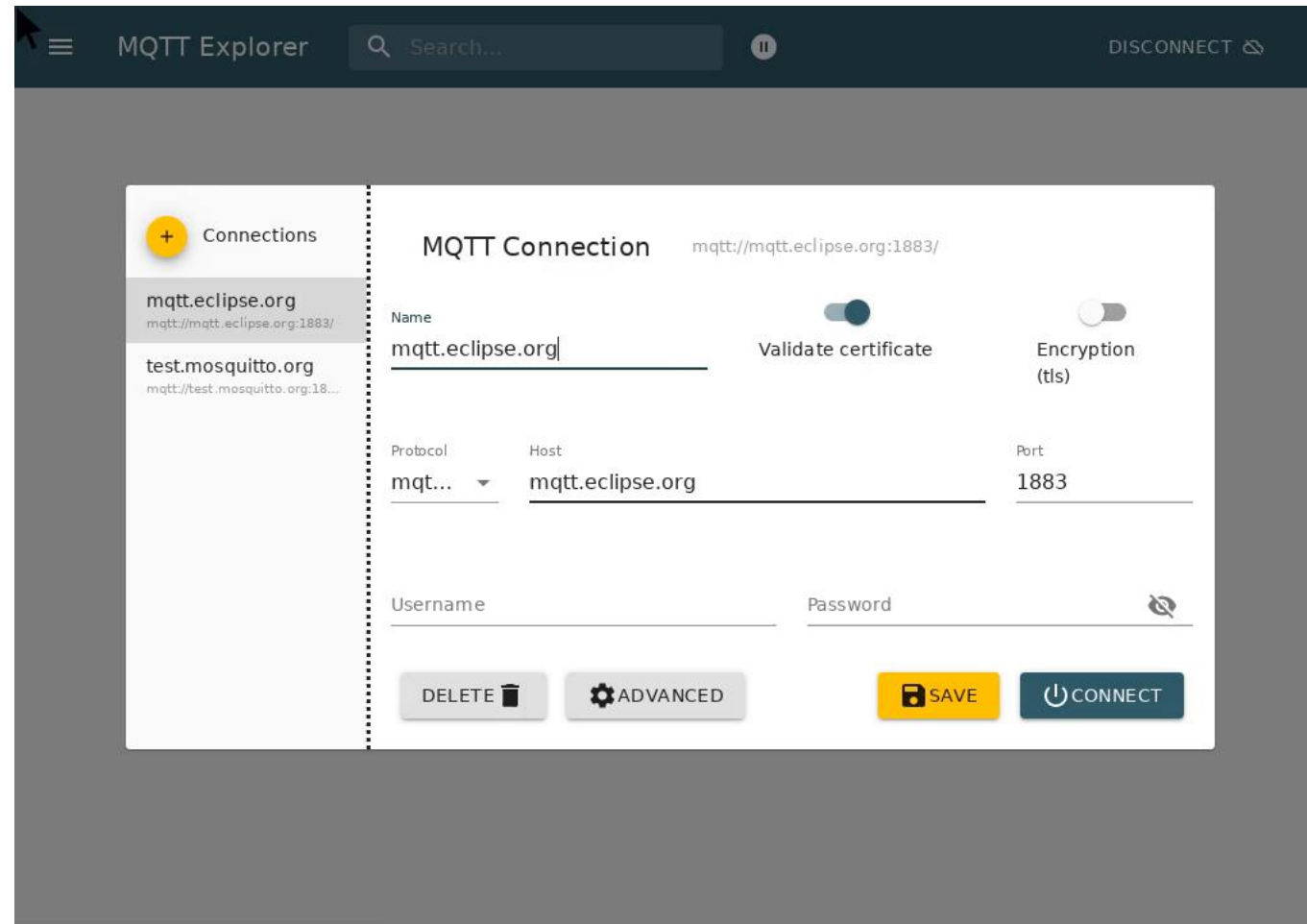
- Quality of Service:
 1. QoS 0: **最大**1回の配信。
 2. QoS 1: **最低**1回の配信。
 3. QoS 2: **正確**に1回の配信。

MQTTの主な特徴

- 遺言 (LWT) :クライアントが切断された場合、ブローカーは**自動的に**メッセージを公開します。
- 永続的セッション:クライアントが**過去のセッション**から重要な情報を取得できるようにします。
- セキュリティ: TSL/SSL暗号化プロトコルと認証をサポートします。

2.6 MQTT ON AMB82

MQTT Explorerの使い方



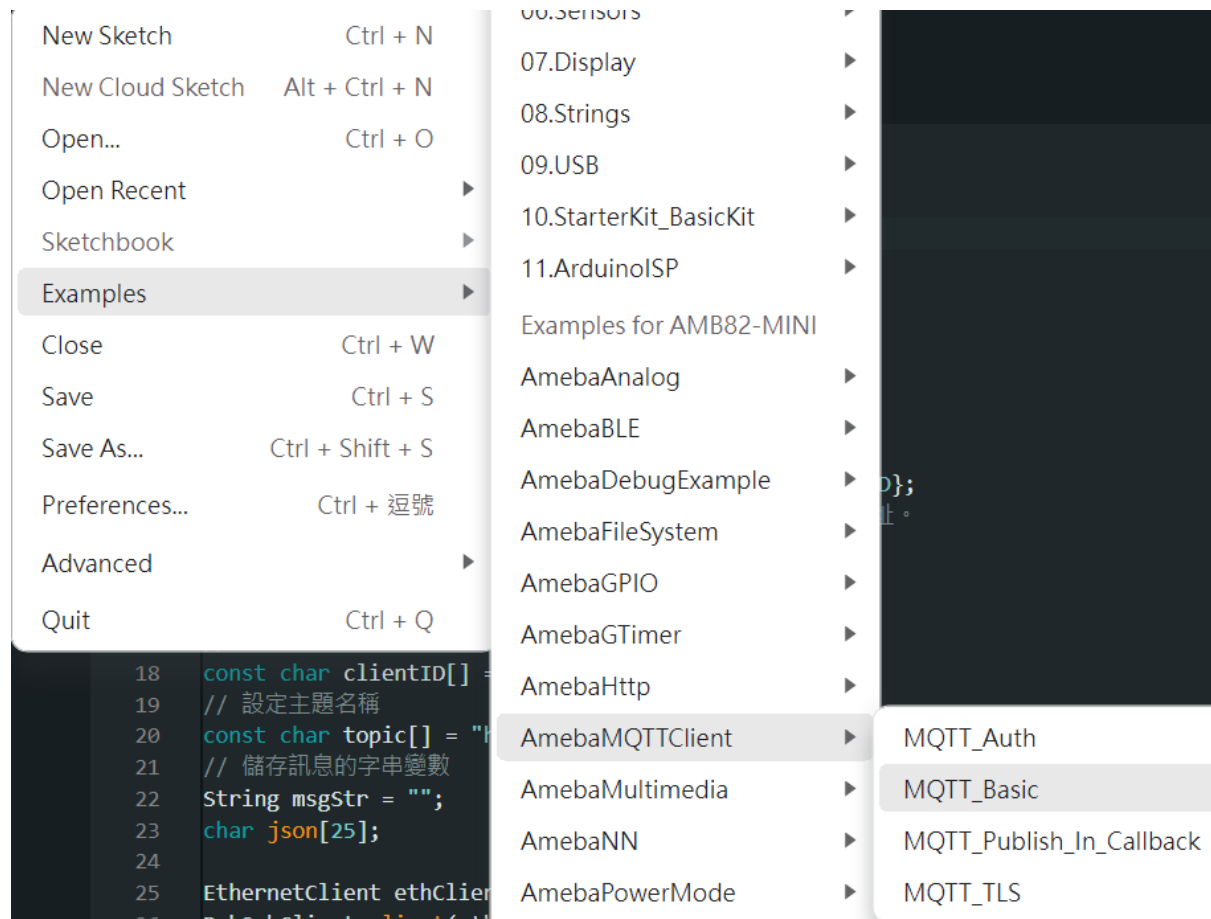
Implementation

2.6 MQTT ON AMB82

Step 1.

Arduino IDEで例を開くには、以下のパスに従ってください。

1. File
2. Examples
3. AmebaMQTTClient
4. MQTT_Basic



2.6 MQTT ON AMB82

Step 2.

WiFiの名前、パスワード、および
publishTopicをプログラムの対応す
る場所に入力してください。

```
18 #include <WiFi.h>
19 #include <PubSubClient.h>
20
21 char ssid[] = "Network_SSID"; // your network SSID (name)
22 char pass[] = "Password"; // your network password
23 int status = WL_IDLE_STATUS; // Indicator of Wifi status
24
25 char mqttServer[] = "test.mosquitto.org";
26 char clientId[] = "amebaClient";
27 char publishTopic[] = "outTopic";
28 char publishPayload[] = "hello world";
29 char subscribeTopic[] = "inTopic";
```

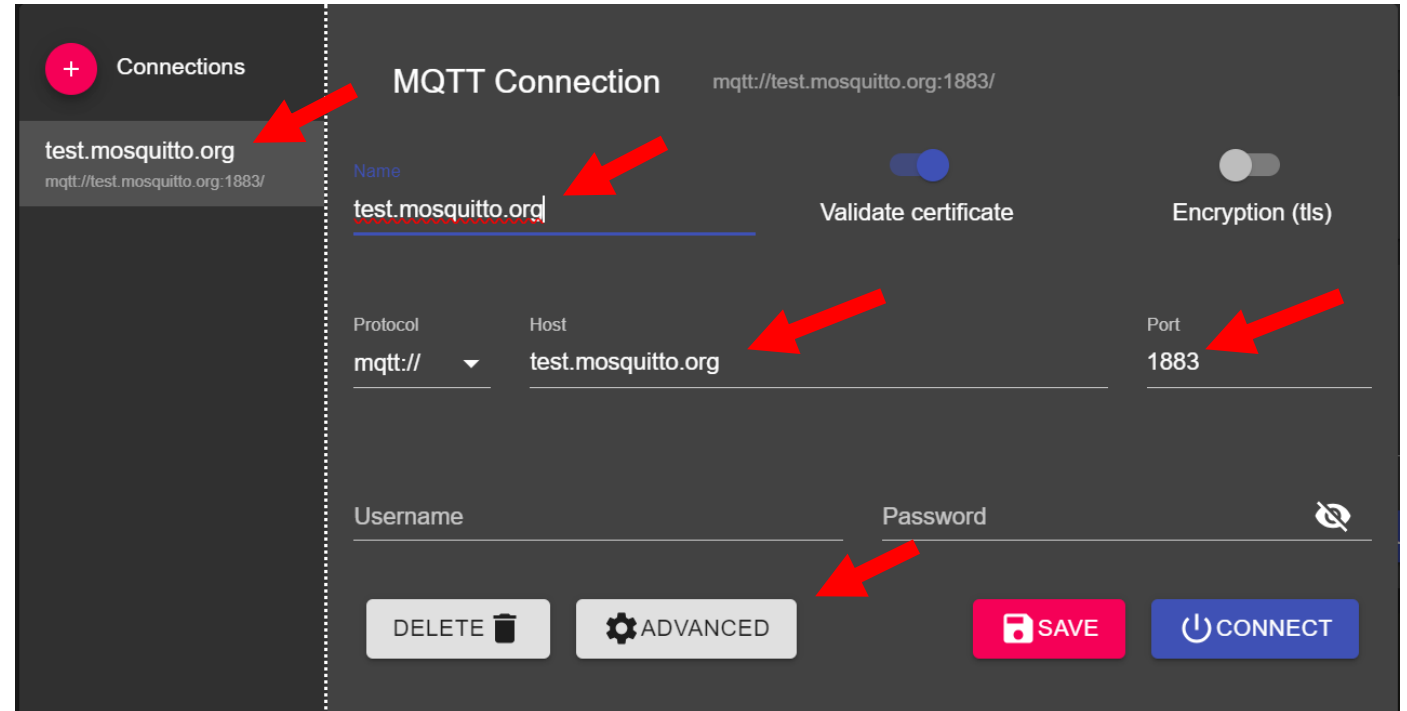
WiFiの名前とパスワードを入力してください。

自分のトピック名を入力してください。

2.6 MQTT ON AMB82

Step 3.

MQTT Explorerを開き、プログラムと同じ接続を開いてください。その後、**ADVANCED**ボタンをクリックしてください。



2.6 MQTT ON AMB82

Step 4.

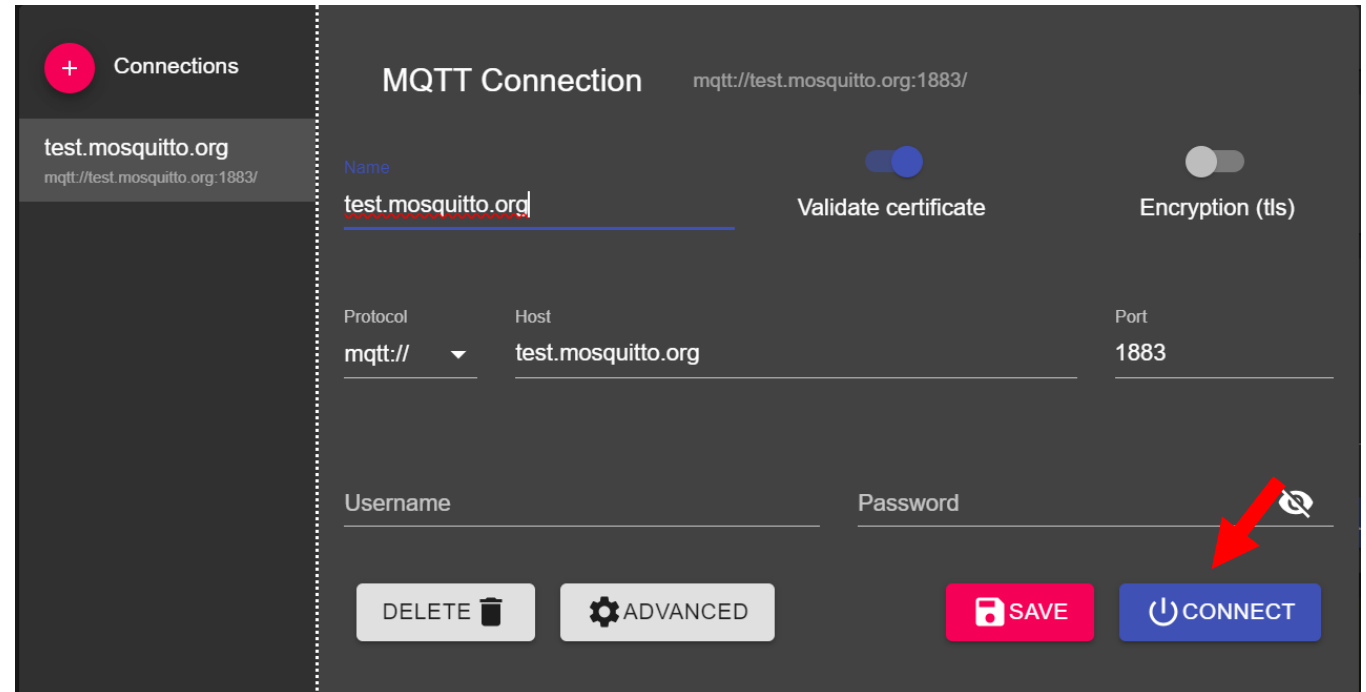
プログラムで指定したpublishTopic
をトピックに入力し、**+ADDボタン**
を押してください。その後、BACK
ボタンを押してください。

The screenshot shows the MQTT Connection interface. At the top, it displays 'MQTT Connection' and the URL 'mqtt://test.mosquitto.org:1883/'. Below this, there is a 'Topic' field containing 'YourTopicName', a 'QoS' dropdown menu set to '0', and a pink '+ ADD' button. A table below the input fields has columns for 'Topic' and 'QoS'. At the bottom, it shows the 'MQTT Client ID' as 'mqtt-explorerer-b43da6d6', a 'CERTIFICATES' button with a lock icon, and a 'BACK' button with a return arrow icon. Red arrows point to the 'Topic' field, the '+ ADD' button, and the 'BACK' button.

2.6 MQTT ON AMB82

Step 5.

前の操作をすべて行った後、
CONNECTボタンを押して接続を
開始してください。



AIoT実装 1

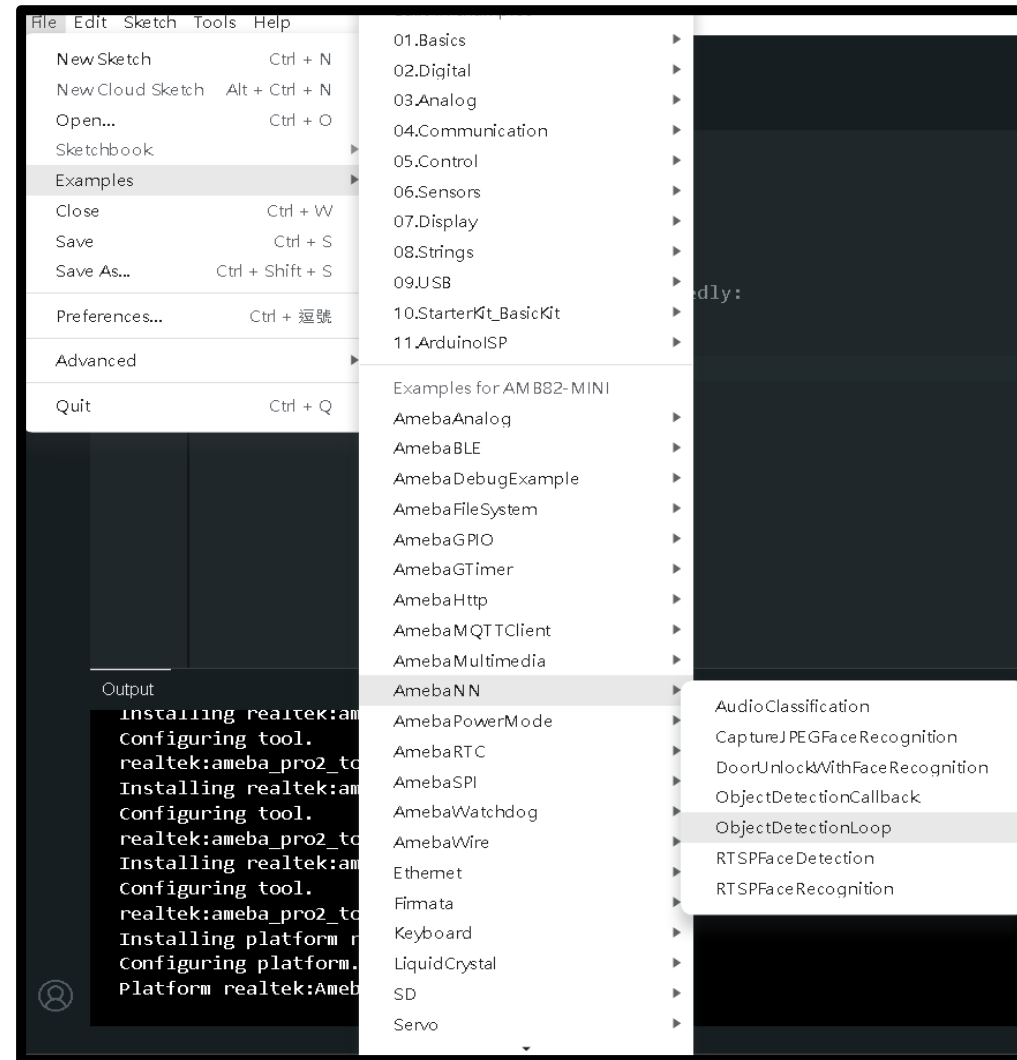
物体検出MQTTを組み合わせて結果をクラウドに送信する

2.6 MQTT ON AMB82

Step 1.

Arduino IDEで例を開くには、以下のパスに従ってください。

1. File
2. Examples
3. AmebaNN
4. ObjectDetectionLoop

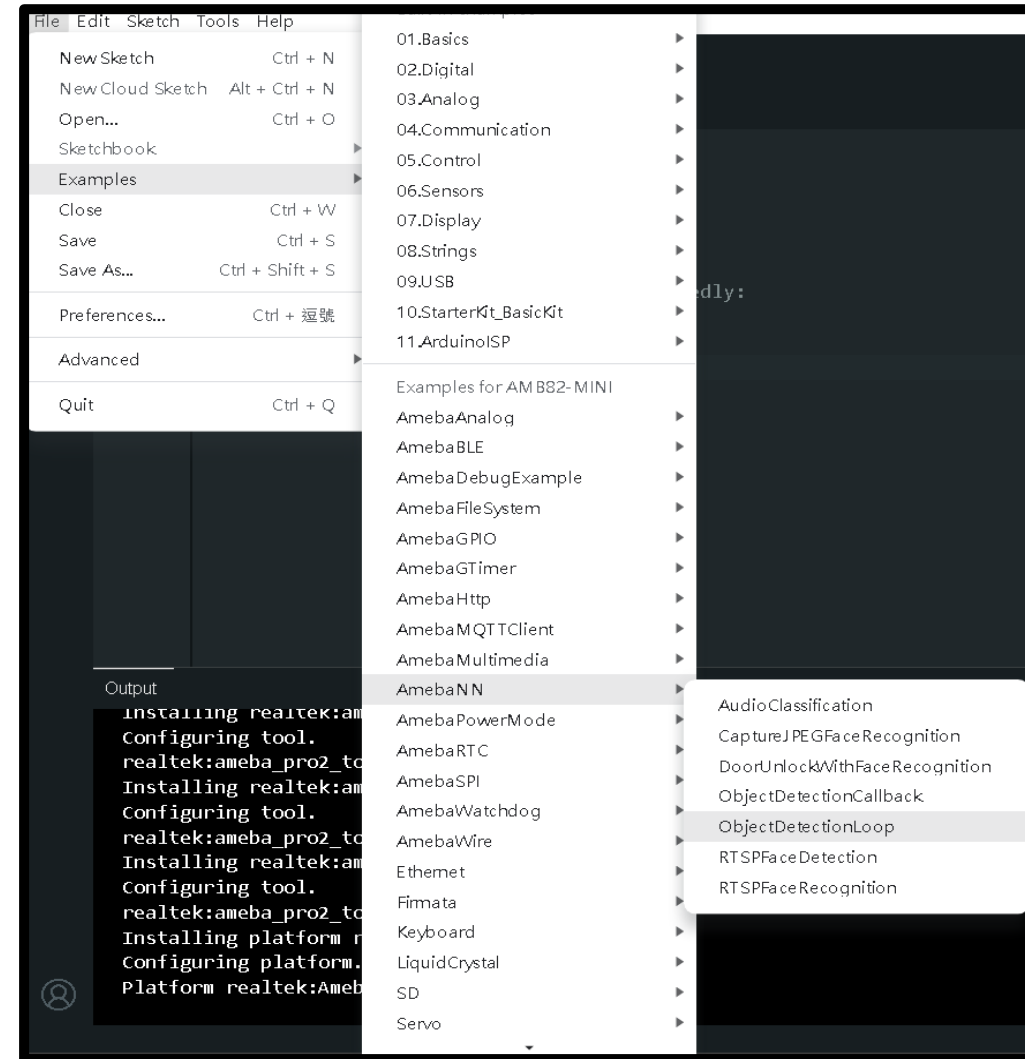


2.6 MQTT ON AMB82

Step 2.

それを開いた後、以下のリンクから
プログラムをコピーし、Arduino
IDEに貼り付けてください。

<https://drive.google.com/file/d/1ABJJTcOY2TODcuO88m8AEMS3zuik4idT/view?usp=sharing>



2.6 MQTT ON AMB82

Step 3.

WiFiの名前、パスワード、および
publishTopicをプログラムの対応す
る場所に入力してください。

```
8 char mqttServer[] = "test.mosquitto.org";
9 char clientId[] = "amebaClient";
10 char publishTopic[] = "TOPIC";
11 char publishPayload[] = "Object Detection with MQTT";
12 char subscribeTopic[] = "inTopic";
13
14 #define NNWIDTH 540
15 #define NNHEIGHT 320
16
17 VideoSetting configNN(NNWIDTH, NNHEIGHT, 10, VIDEO_RGB, 0);
18 NNObjectDetection ObjDet;
19 StreamIO videoStreamerNN(1, 1);
20
21 char ssid[] = "SSID"; // your network SSID (name)
22 char pass[] = "Password"; // your network password
23 int status = WL_IDLE_STATUS;
```

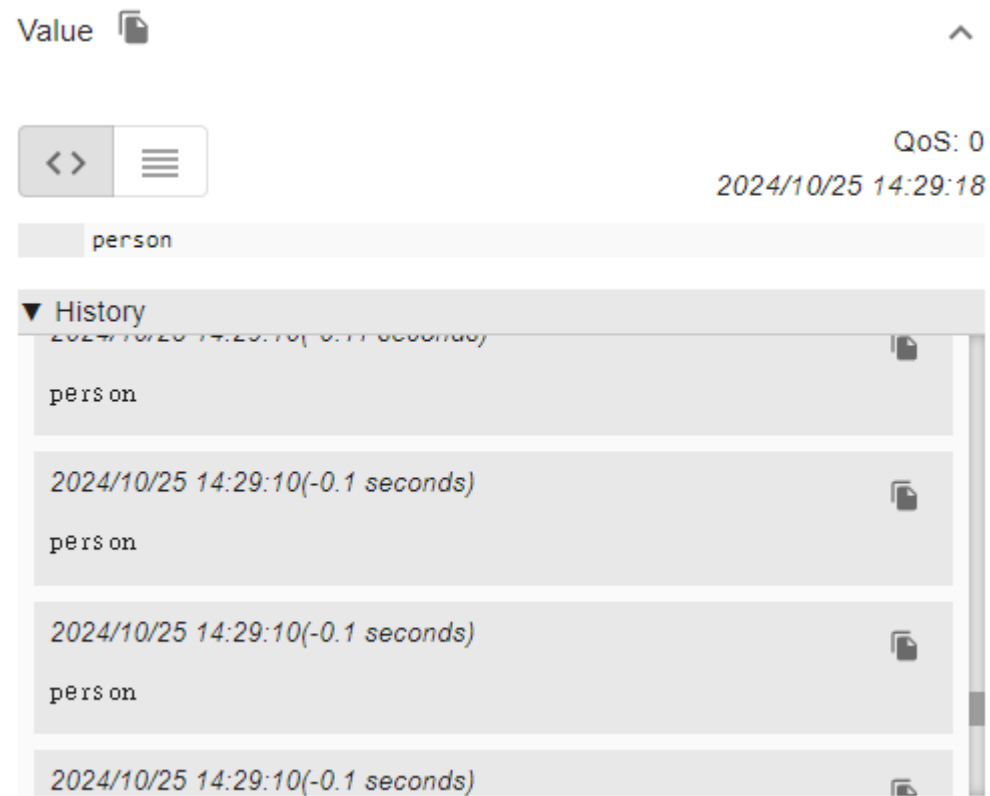
自分のトピック名
を入力してください。

WiFiの名前とパス
ワードを入力して
ください

2.6 MQTT ON AMB82

Step 4.

検出結果はMQTT Explorerに表示されます。



AIoT実装 2

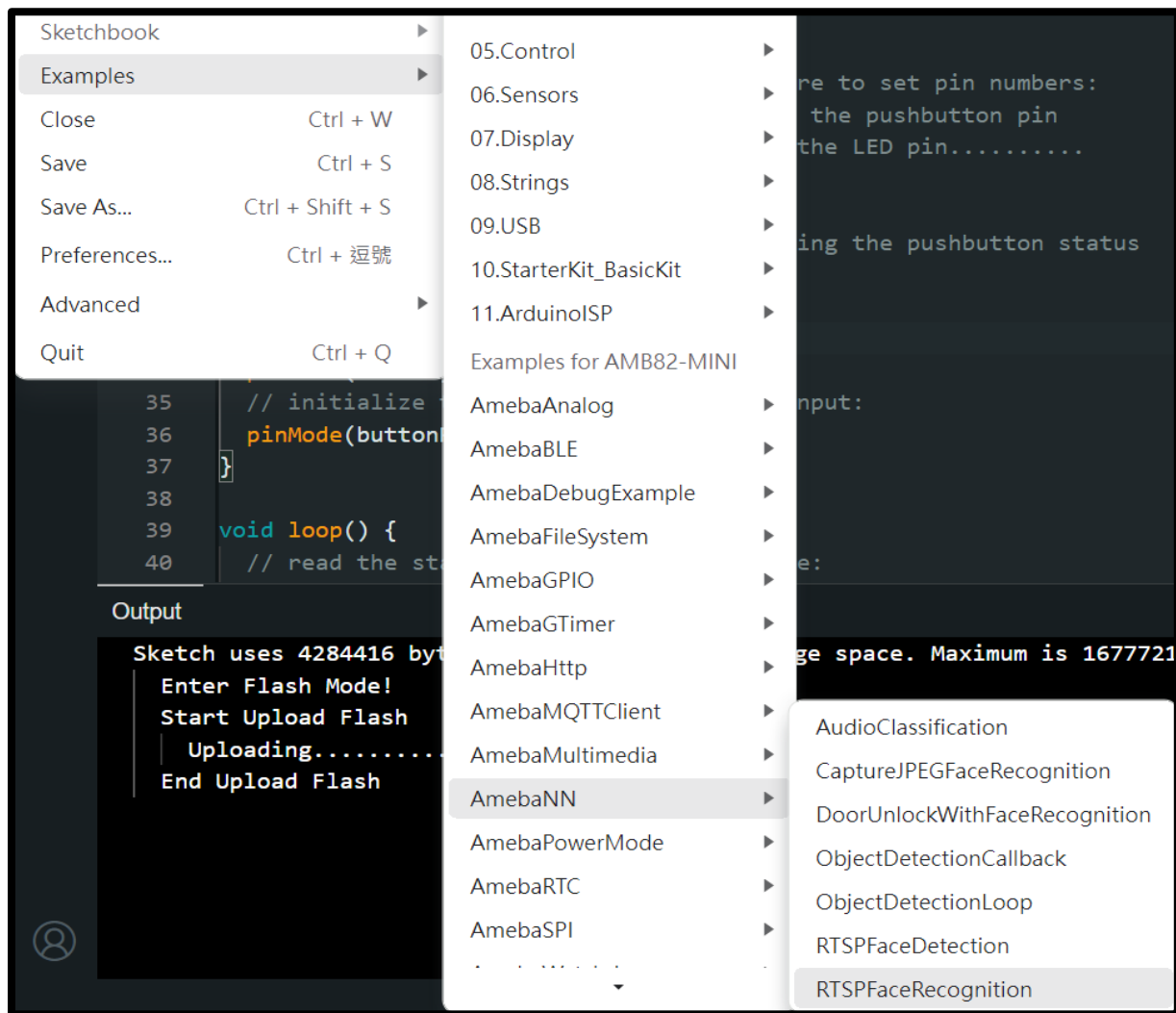
シンプルな顔認識出勤システム

2.6 MQTT ON AMB82

Step 1.

Arduino IDEで例を開くには、以下のパスに従ってください。

1. File
2. Examples
3. AmebaNN
4. RTSPFaceRecognition

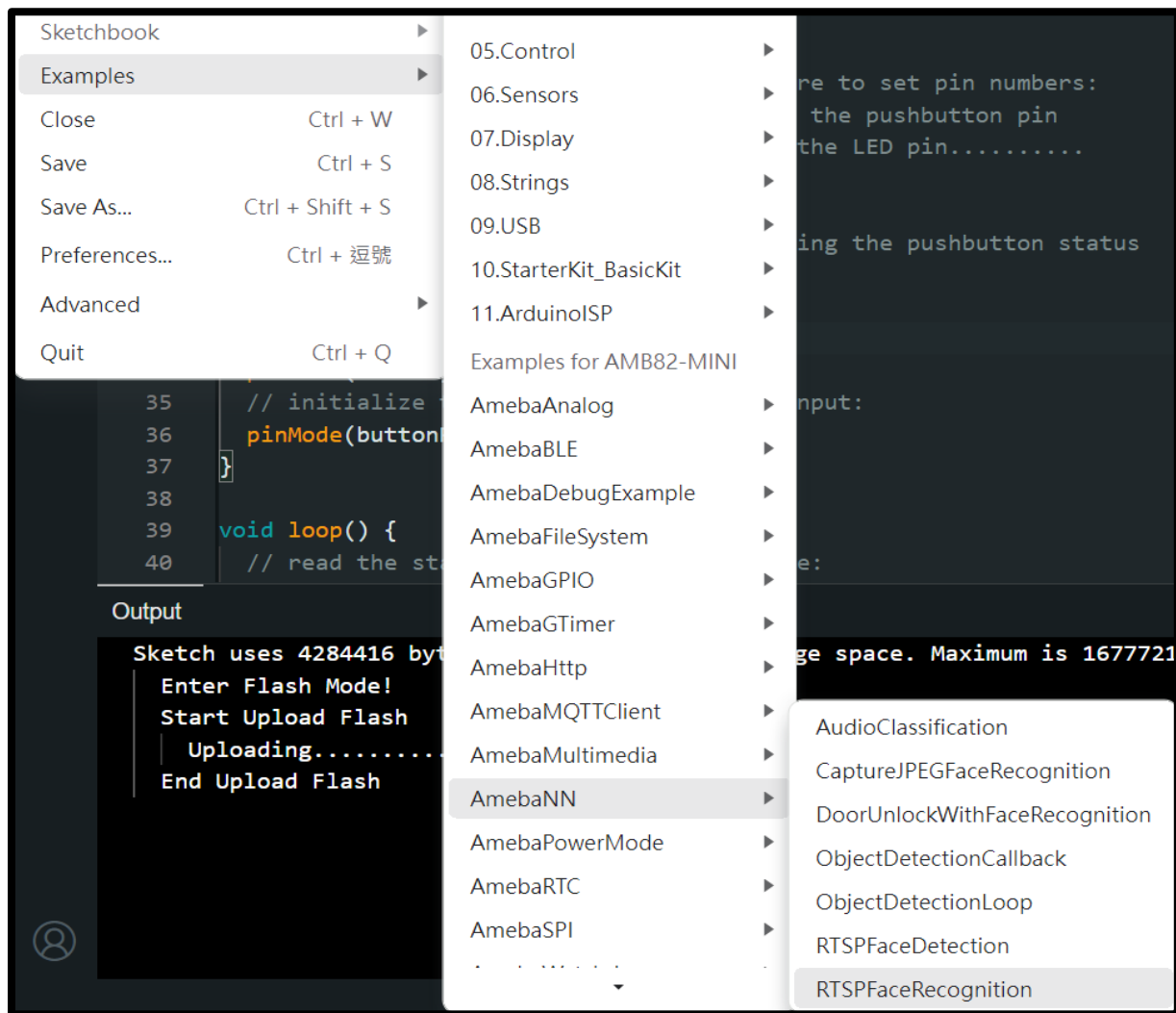


2.6 MQTT ON AMB82

Step 2.

それを開いた後、以下のリンクから
プログラムをコピーし、Arduino
IDEに貼り付けてください。

<https://drive.google.com/file/d/15r02-z5OYz23EaD29MFaOrdinRyMYx8P/view?usp=sharing>



2.6 MQTT ON AMB82

Step 3.

WiFiの名前、パスワード、および
publishTopicをプログラムの対応す
る場所に入力してください。

```
9 // Wi-Fi and MQTT settings
10 char ssid[] = "SSID";
11 char pass[] = "Password";
12 char mqttServer[] = "test.mosquitto.org";
13 char clientId[] = "amebaClient";
14 char publishTopic[] = "TOPIC";
15 WiFiClient wifiClient;
16 PubSubClient client(wifiClient);
17
18 #define CHANNEL 0
19 #define CHANNELNN 3
20
21 // Customised resolution for NN
22 #define NNWIDTH 576
23 #define NNHEIGHT 320
```

WiFiの名前とパスワードを入力してください

自分のトピック名を入力してください。

2.6 MQTT ON AMB82

1. Python (VScode、anaconda) を開いてください。
2. ターミナルに**pip install paho-mqtt**と入力してください。
3. python (.py) ファイルを作成し、以下のリンクからプログラムをコピーして、新しく作成したファイルに貼り付けましてください。

https://drive.google.com/file/d/1_GRLBpuqgWkN8e_25UPkef-dO_pNCIDe/view?usp=sharing

2.6 MQTT ON AMB82

Return Code	Response
0	Connection accepted
1	Connection refused: level of MQTT protocol not supported by server.
2	Connection refused: client identifier not allowed by server.
3	Network connection successful but MQTT service is unavailable.
4	Data in username or password is malformed.
5	Client not authorized to connect.
6-255	Reserved for future use.

2.6 MQTT ON AMB82

```
11 def on_message(client, userdata, msg):
12     message = msg.payload.decode()
13
14     current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
15
16     if message.lower() != "unknown":
17         with open("mqtt_data.txt", "a") as f:
18             f.write(f"Time: {current_time}, Topic: {msg.topic}, Name: {message}\n")
19
20     print(f"{message} was detected at {current_time}")
21 else:
22     print(f"Unknown person detected, ignoring.")
```

on_message関数の説明

2.6 MQTT ON AMB82

```
24  if __name__ == '__main__':
25      client = mqtt.Client()
26      client.on_connect = on_connect
27      client.on_message = on_message
28      client.connect("test.mosquitto.org", 1883, 60)
29      client.loop_forever()
30
```

main関数の説明

2.6 MQTT ON AMB82

DEMO

The screenshot displays two windows on a Windows desktop. The left window is the Arduino IDE 2.3.2, showing the source code for 'RTSPFaceRecognition.ino' on an 'AMB82-MINI' board. The code includes headers for WiFi, MQTT, and face detection, and sets up a client to connect to 'test.mosquitto.org'. The Serial Monitor shows the output of the program, indicating that one face was detected and providing coordinates for the face's bounding box.

```
RTSPFaceRecognition.ino
1 #include <WiFi.h>
2 #include <PubSubClient.h>
3 #include "StreamIO.h"
4 #include "VideoStream.h"
5 #include "RTSP.h"
6 #include "MFaceDetectionRecognition.h"
7 #include "VideoStreamOverlay.h"
8
9 // Wi-Fi and MQTT settings
10 char ssid[] = "Pockyyyz"; // your network SSID (name)
11 char pass[] = "12345678"; // your network password
12 char mqttServer[] = "test.mosquitto.org";
13 char clientId[] = "amebaClient";
14 char publishTopic[] = "Face_AMB_P";
15 WiFiClient wifiClient;
16 PubSubClient client(wifiClient);
17
18 #define CHANNEL 0
19 #define CHANNELLN 3
```

Serial Monitor Output:

```

Total number of faces detected = 1
Face 0 name unknown: 770 1104 332 760
SCRFD tick[28]
MBFACENET tick[16]

Total number of faces detected = 1
Face 0 name unknown: 771 1104 334 760
SCRFD tick[28]
MBFACENET tick[16]

Total number of faces detected = 1
Face 0 name unknown: 768 1104 336 759
```

The right window is a VLC media player showing a video stream from 'rtsp://172.20.10.4:554'. The video shows a person's face with a red bounding box overlaid, indicating successful face detection. The system tray at the bottom shows the date and time as 2024/9/11 at 05:38.



Chapter 3

Object Detection



3.1YOLO

(You Only Look Once)

3.1 YOLO(You Only Look Once)

物体検出とは?

- 物体検出は、**アンカー**を使用して写真やビデオなどの画像コンテンツ内の物体の範囲をマークし、それが**どのような**物体であるか、およびこの物体に対する添付モデルの**信頼度**を分類することです。
- 現在最も人気があり有名なオブジェクト検出モデルは**YOLO**です。

3.1 YOLO(You Only Look Once)

物体検出とは?

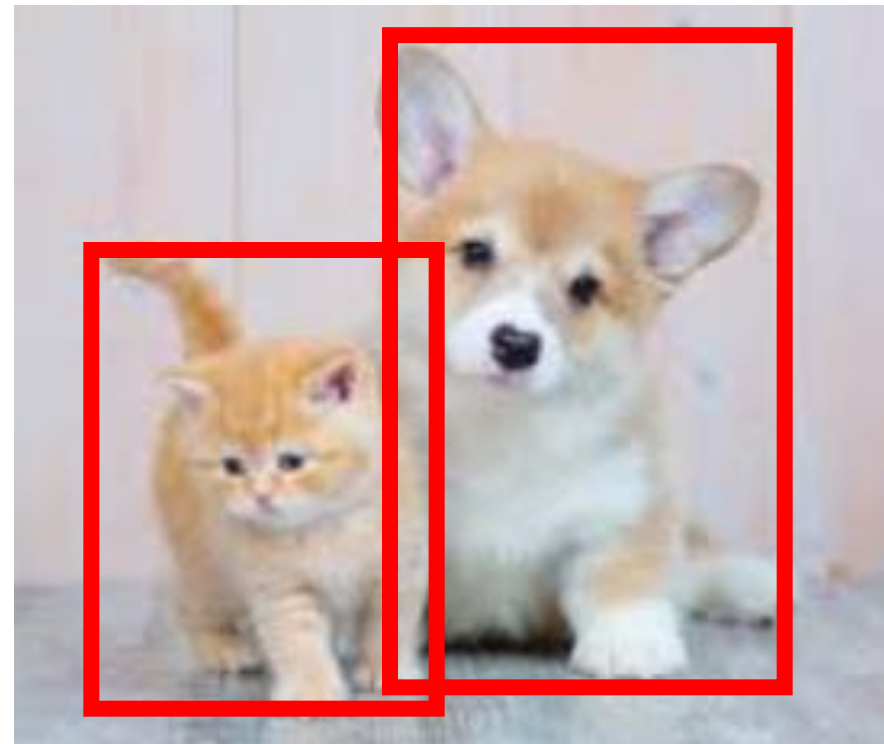
- 物体検出は、**アンカー**を使用して写真やビデオなどの画像コンテンツ内の物体の範囲をマークし、それが**どのような**物体であるか、およびこの物体に対する添付モデルの**信頼度**を分類することです。
- 現在最も人気があり有名なオブジェクト検出モデルは**YOLO**です。



3.1 YOLO(You Only Look Once)

物体検出とは?

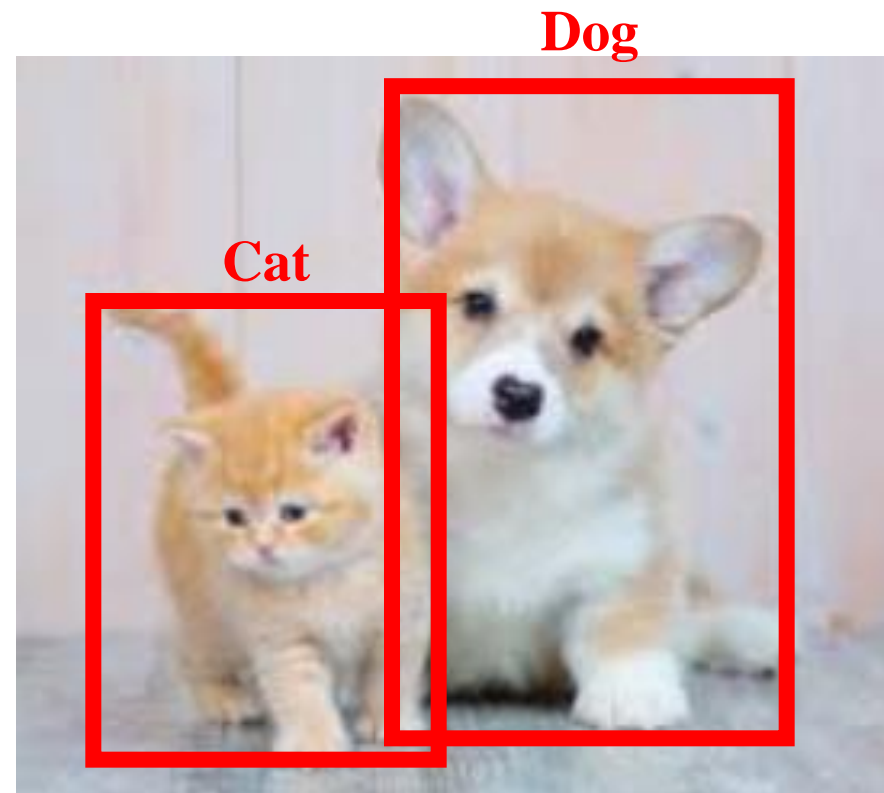
- 物体検出は、**アンカー**を使用して写真やビデオなどの画像コンテンツ内の物体の範囲をマークし、それが**どのような**物体であるか、およびこの物体に対する添付モデルの**信頼度**を分類することです。
- 現在最も人気があり有名なオブジェクト検出モデルは**YOLO**です。



3.1 YOLO(You Only Look Once)

物体検出とは?

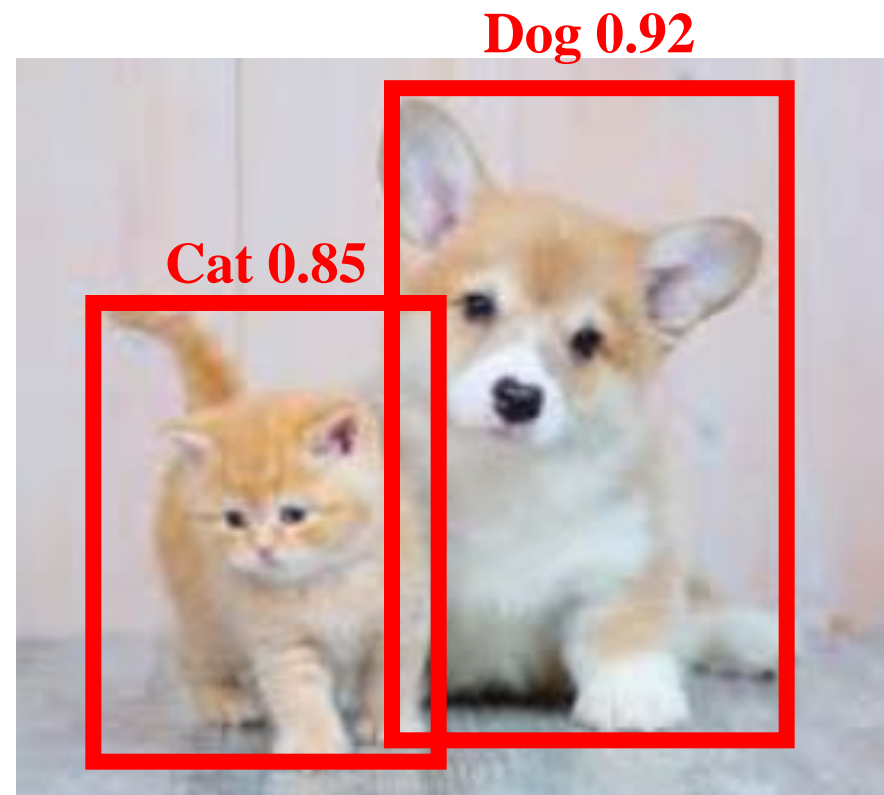
- 物体検出は、**アンカー**を使用して写真やビデオなどの画像コンテンツ内の物体の範囲をマークし、それが**どのような**物体であるか、およびこの物体に対する添付モデルの**信頼度**を分類することです。
- 現在最も人気があり有名なオブジェクト検出モデルは**YOLO**です。



3.1 YOLO(You Only Look Once)

物体検出とは?

- 物体検出は、**アンカー**を使用して写真やビデオなどの画像コンテンツ内の物体の範囲をマークし、それが**どのような**物体であるか、およびこの物体に対する添付モデルの**信頼度**を分類することです。
- 現在最も人気があり有名なオブジェクト検出モデルは**YOLO**です。



3.1 YOLO(You Only Look Once)

物体検出とは?

- 物体検出は、**アンカー**を使用して写真やビデオなどの画像コンテンツ内の物体の範囲をマークし、それが**どのような**物体であるか、およびこの物体に対する添付モデルの**信頼度**を分類することです。
- 現在最も人気があり有名なオブジェクト検出モデルは**YOLO**です。



3.1 YOLO(You Only Look Once)

物体検出とは?

- 物体検出は、**アンカー**を使用して写真やビデオなどの画像コンテンツ内の物体の範囲をマークし、それが**どのような**物体であるか、およびこの物体に対する添付モデルの**信頼度**を分類することです。
- 現在最も人気があり有名なオブジェクト検出モデルは**YOLO**です。



YOLO(You Only Look Once)

INPUT

640 pixel



YOLO(You Only Look Once)

INPUT

640 pixel

Red,Green,Blue

(3,640,640)



640 pixel

YOLO(You Only Look Once)

INPUT

640 pixel



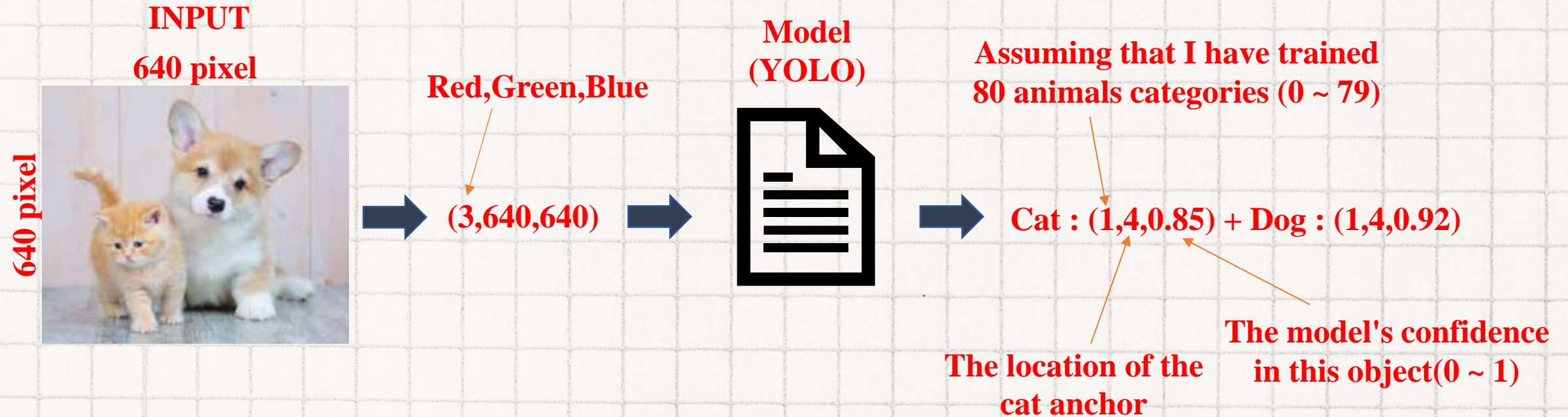
Red,Green,Blue

(3,640,640)

**Model
(YOLO)**



YOLO(You Only Look Once)

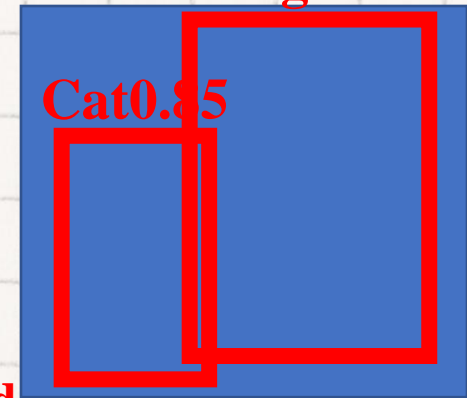


YOLO(You Only Look Once)

OUTPUT

Dog0.92

Cat0.85



INPUT

640 pixel



Red,Green,Blue

(3,640,640)

Model
(YOLO)



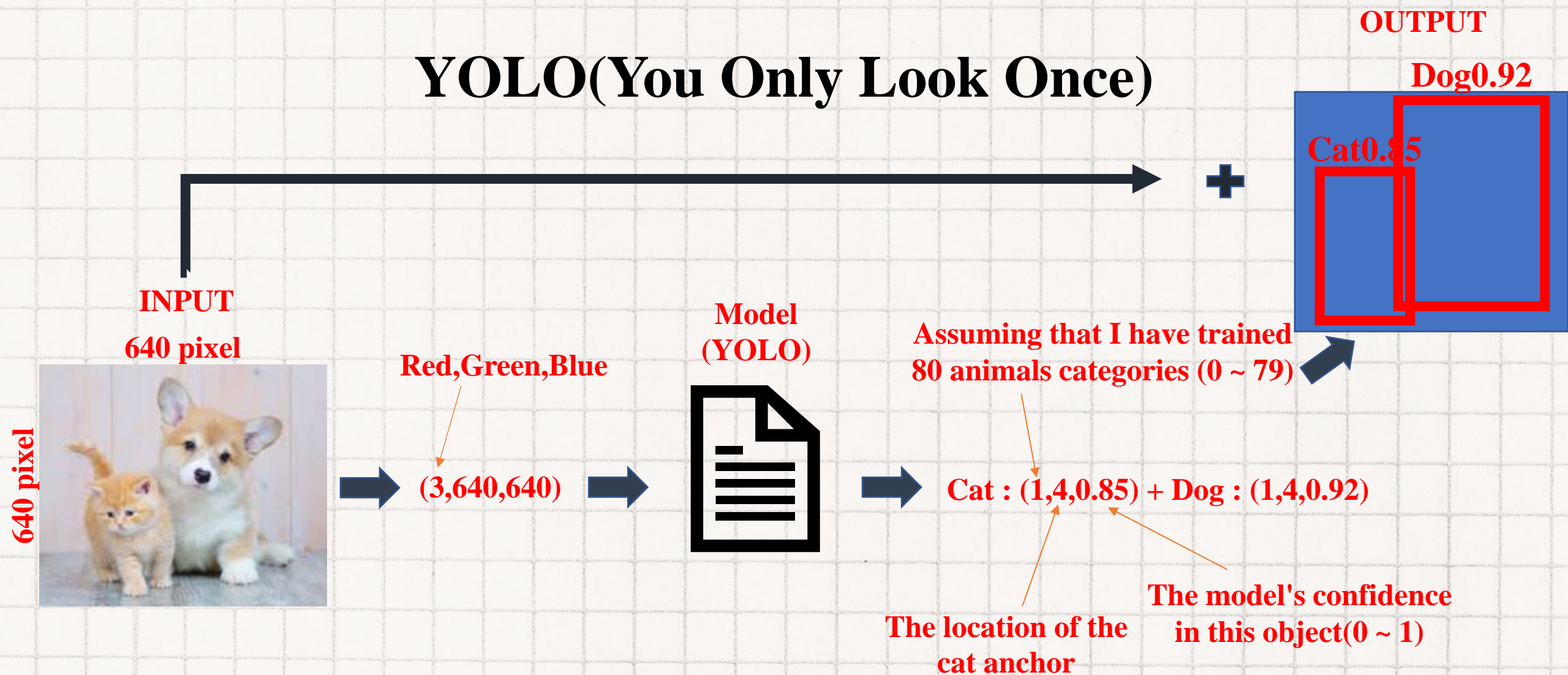
Assuming that I have trained
80 animals categories (0 ~ 79)

Cat : (1,4,0.85) + Dog : (1,4,0.92)

The location of the
cat anchor

The model's confidence
in this object(0 ~ 1)

YOLO(You Only Look Once)



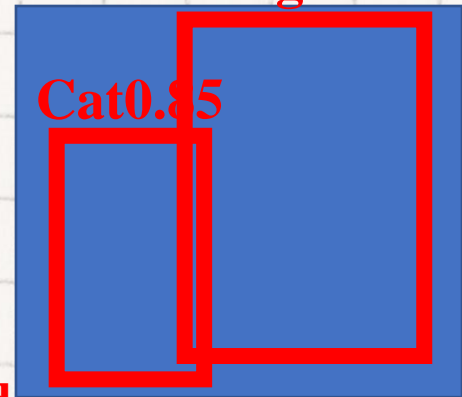
YOLO(You Only Look Once)

OUTPUT

Dog0.92

Cat0.85

+



INPUT

640 pixel



Red,Green,Blue

(3,640,640)

Model
(YOLO)



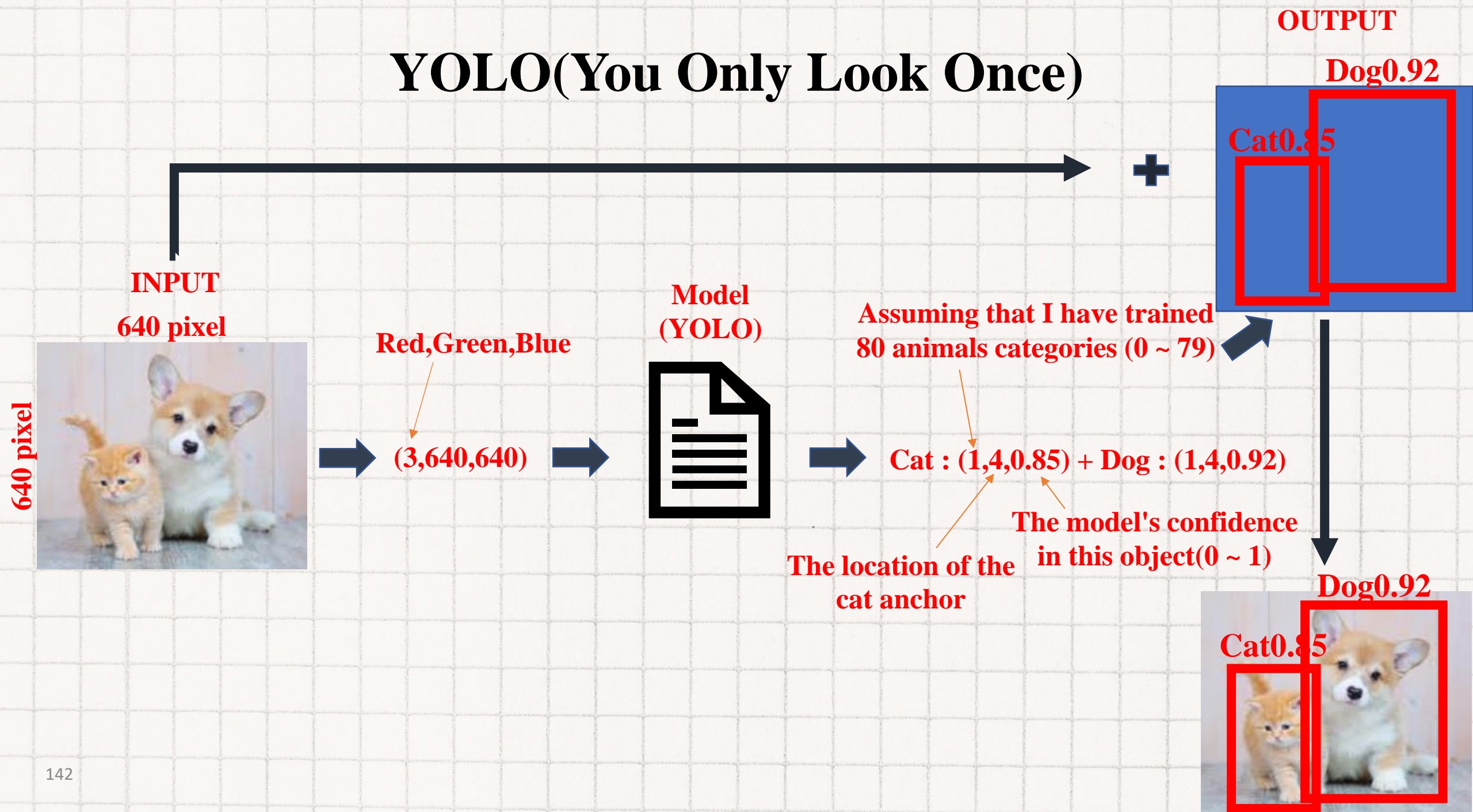
Assuming that I have trained
80 animals categories (0 ~ 79)

Cat : (1,4,0.85) + Dog : (1,4,0.92)

The location of the
cat anchor

The model's confidence
in this object(0 ~ 1)

YOLO(You Only Look Once)



3.1 YOLO(You Only Look Once)

Video of YOLO

3.1 YOLO(You Only Look Once)

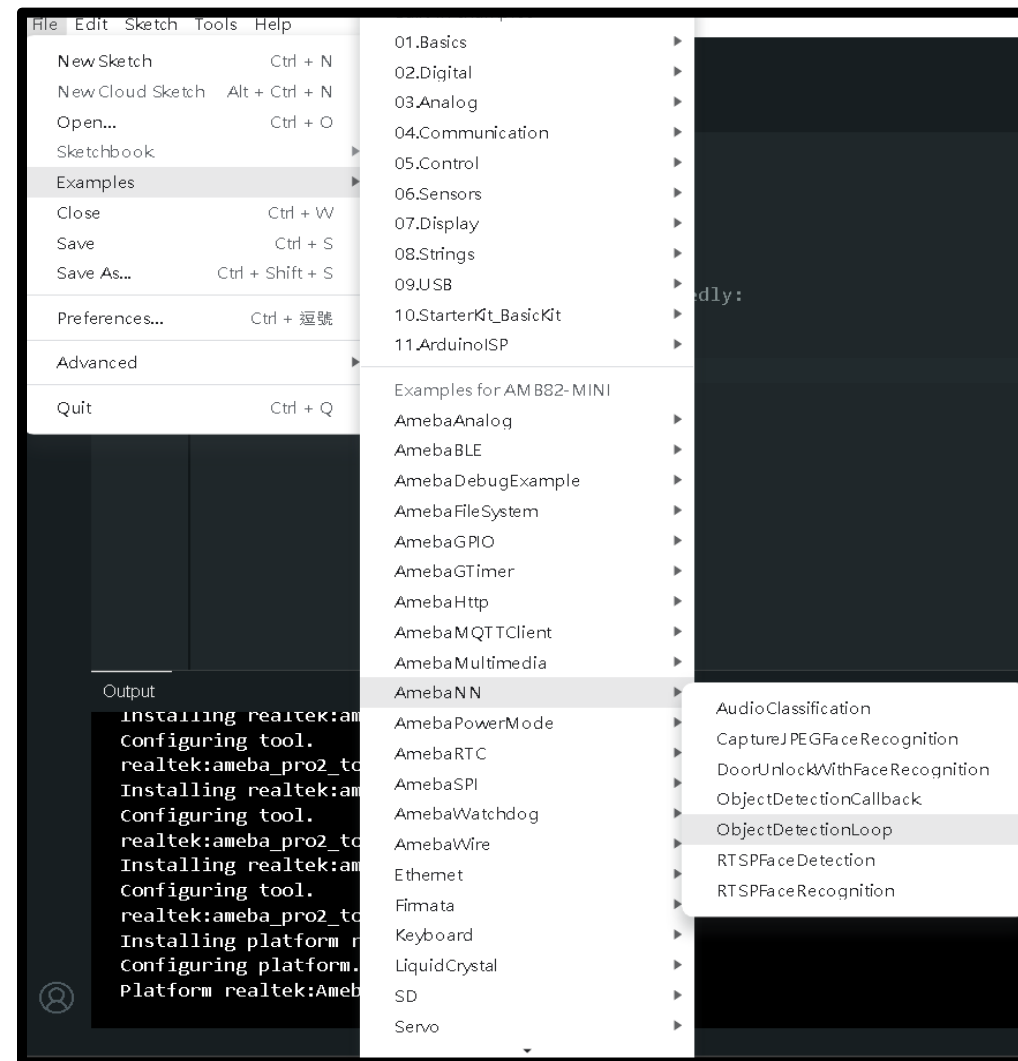
Implementation

3.1 YOLO(You Only Look Once)

Step 1.

Arduino IDEで例を開くには、以下のパスに従ってください。

1. File
2. Examples
3. AmebaNN
4. ObjectDetectionLoop



3.1 YOLO(You Only Look Once)

Step 2.

プログラムのWi-Fi接続設定に、
SSIDとパスワードを設定してください。

```
#include <WiFi.h>
#include "StreamIO.h"
#include "VideoStream.h"
#include "RTSP.h"
#include "NNObjectDetection.h"
#include "VideoStreamOverlay.h"
#include "ObjectClassList.h"

#define CHANNEL 0
#define CHANNELLN 3

// Lower resolution for NN processing
#define NNWIDTH 576
#define NNHEIGHT 320

VideoSetting config(VIDEO_FHD, 30, VIDEO_H264, 0);
VideoSetting configNN(NNWIDTH, NNHEIGHT, 10, VIDEO_RGB, 0);
NNObjectDetection objDet;
RTSP rtsp;
StreamIO videoStreamer(1, 1);
StreamIO videoStreamerNN(1, 1);

char ssid[] = "Network_SSID"; // your network SSID (name)
char pass[] = "Password"; // your network password
int status = WL_IDLE_STATUS;

IPAddress ip;
int rtsp_port;

void setup() {
  Serial.begin(115200);

  // attempt to connect to Wifi network:
```

WiFiの名前とパスワード
を入力してください

3.1 YOLO(You Only Look Once)

Step 3.モデルを選択(選択は任意です)

```
// Configure object detection with corresponding video format information
// Select Neural Network(MN) task and models
ObjDet.configVideo(configMN);
ObjDet.modelSelect(OBJECT_DETECTION, DEFAULT_YOLOV4TINY, NA_MODEL, NA_MODEL);
ObjDet.begin();
```

各種taskに対応するモデル一覧

```
Models
=====
YOLOv3 model      DEFAULT_YOLOV3TINY    / CUSTOMIZED_YOLOV3TINY
YOLOv4 model      DEFAULT_YOLOV4TINY    / CUSTOMIZED_YOLOV4TINY
YOLOv7 model      DEFAULT_YOLOV7TINY    / CUSTOMIZED_YOLOV7TINY
SCRFD model       DEFAULT_SCRFD         / CUSTOMIZED_SCRFD
MobileFaceNet model DEFAULT_MOBILEFACENET / CUSTOMIZED_MOBILEFACENET
No model          NA_MODEL
```

3.1 YOLO(You Only Look Once)

- 事前に訓練されたモデルは合計80種類の物体を認識できます。
- 特定の物体の認識を無効にするには、filterを0に設定してください。

```
ObjectDetectionLoop.ino  ObjectClassList.h
4  struct ObjectDetectionItem {
5      uint8_t index;
6      const char* objectName;
7      uint8_t filter;
8  };
9
10 // List of objects the pre-trained model i
11 // Index number is fixed and hard-coded fr
12 // Set the filter value to 0 to ignore any
13 ObjectDetectionItem itemList[80] = {
14 {0, "person",      1},
15 {1, "bicycle",    1},
16 {2, "car",        1},
17 {3, "motorbike",  1},
18 {4, "aeroplane",  1},
19 {5, "bus",        1},
20 {6, "train",      1},
21 {7, "truck",      1},
22 {8, "boat",       1},
23 {9, "traffic light", 1},
24 {10, "fire hydrant", 1},
25 {11, "stop sign",  1},
```

3.1 YOLO(You Only Look Once)

Program Explanation

3.1 YOLO(You Only Look Once)

include

3.1 YOLO(You Only Look Once)

```
#include "WiFi.h"
#include "StreamIO.h"
#include "VideoStream.h"
#include "RTSP.h"
#include "NNObjectDetection.h"
#include "VideoStreamOverlay.h"
#include "ObjectClassList.h"
// 匯入所需的庫檔案，包括WiFi連線、串流輸入輸出、影音串流、RTSP、神經網路物件偵測等功能

#define CHANNEL 0
#define CHANNELNN 3
// 定義使用的影音通道，CHANNEL 用於一般串流，CHANNELNN 用於神經網路處理

#define NNWIDTH 576
#define NNHEIGHT 320
// 定義神經網路處理的解析度
```


3.1 YOLO(You Only Look Once)

setup()

3.1 YOLO(You Only Look Once)

```
void setup() { // 初始化設置函數
  Serial.begin(115200);
  // 初始化序列通訊，設定傳輸速率
  // 嘗試連接到WiFi網絡
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);

    // 等待2秒鐘以連接
    delay(2000);
  }
  ip = WiFi.localIP();

  // 使用影音格式資訊配置相機影音通道
  // 根據您的WiFi網絡質量調整比特率
  config.setBitrate(2 * 1024 * 1024); // 使用2Mbps以防止網絡擁堵
  Camera.configVideoChannel(CHANNEL, config);
  Camera.configVideoChannel(CHANNELNN, configNN);
  Camera.videoInit();
}
```

3.1 YOLO(You Only Look Once)

```
// 配置RTSP及相應影片格式資訊
rtsp.configVideo(config);
rtsp.begin();
rtsp_portnum = rtsp.getPort();

// 配置物件偵測及相應影片格式資訊
// 選擇神經網絡(NN)任務和模型
ObjDet.configVideo(configNN);
ObjDet.modelSelect(OBJECT_DETECTION, DEFAULT_YOLOV4TINY, NA_MODEL, NA_MODEL);
ObjDet.begin();

// 配置StreamIO物件從影片通道流到RTSP
videoStreamer.registerInput(Camera.getChannel());
videoStreamer.registerOutput(rtsp);
if (videoStreamer.begin() != 0) {
    Serial.println("StreamIO link start failed");
}

// 啟動影片通道
Camera.channelBegin(CHANNEL);
```

3.1 YOLO(You Only Look Once)

```
// 配置StreamIO物件，從RGB影音通道串流數據到物件偵測
videoStreamerNN.registerInput(Camera.getStream(CHANNELNN));
videoStreamerNN.setStackSize();
videoStreamerNN.setTaskPriority();
videoStreamerNN.registerOutput(ObjDet);
if (videoStreamerNN.begin() != 0) {
    Serial.println("StreamIO link start failed");
}

// 開始神經網路的影音通道
Camera.channelBegin(CHANNELNN);

// 在RTSP影音通道上開始OSD繪圖
OSD.configVideo(CHANNEL, config);
OSD.begin();
```

3.1 YOLO(You Only Look Once)

loop()

3.1 YOLO(You Only Look Once)

```
void loop() { // 主循環函數，持續執行物件偵測並更新RTSP串流
  std::vector<ObjectDetectionResult> results = ObjDet.getResult();

  uint16_t im_h = config.height();
  uint16_t im_w = config.width();

  Serial.print("Network URL for RTSP Streaming: ");
  Serial.print("rtsp://");
  Serial.print(ip);
  Serial.print(":");
  Serial.println(rtsp_portnum);
  Serial.println(" ");

  printf("Total number of objects detected = %d\r\n", ObjDet.getResultCount());
  OSD.createBitmap(CHANNEL);
```

3.1 YOLO(You Only Look Once)

```
if (ObjDet.getResultCount() > 0) {
    for (int i = 0; i < ObjDet.getResultCount(); i++) {
        int obj_type = results[i].type();
        if (itemList[obj_type].filter) { // 檢查是否應該忽略該項目

            ObjectDetectionResult item = results[i];
            // 結果坐標是從0.00到1.00的浮點數
            // 與RTSP解析度相乘以獲得像素中的坐標
            int xmin = (int)(item.xMin() * im_w);
            int xmax = (int)(item.xMax() * im_w);
            int ymin = (int)(item.yMin() * im_h);
            int ymax = (int)(item.yMax() * im_h);

            // 繪製邊界框
            printf("Item %d %s:\t%d %d %d %d\n\r", i, itemList[obj_type].objectName, xmin, xmax, ymin, ymax);
            OSD.drawRect(CHANNEL, xmin, ymin, xmax, ymax, 3, OSD_COLOR_WHITE);

            // 打印文字
            char text_str[20];
            snprintf(text_str, sizeof(text_str), "%s %d", itemList[obj_type].objectName, item.score());
            OSD.drawText(CHANNEL, xmin, ymin - OSD.getTextHeight(CHANNEL), text_str, OSD_COLOR_CYAN);
        }
    }
}

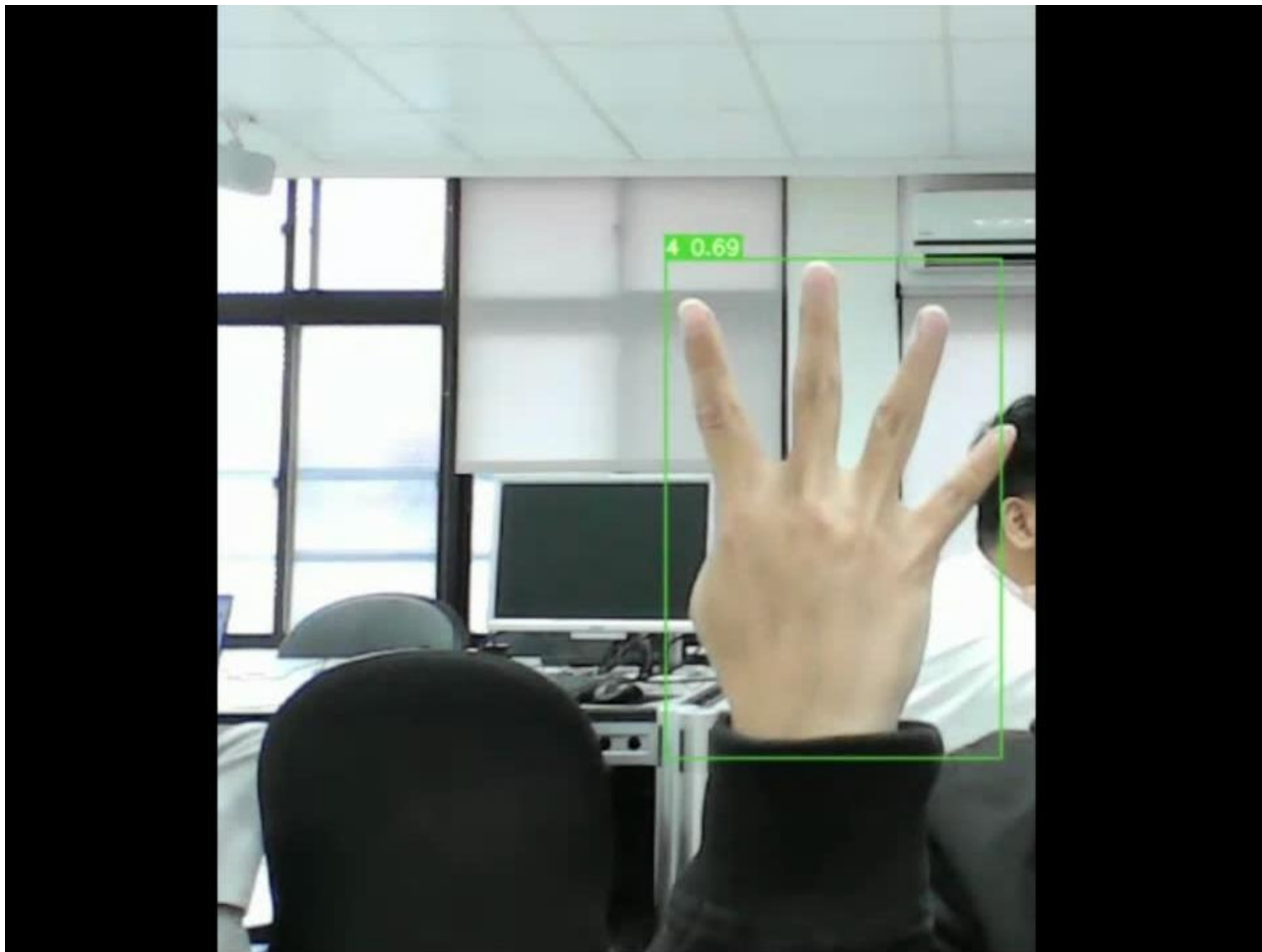
OSD.update(CHANNEL);

// 延遲等待新的結果
delay(100);
```

3.1 YOLO(You Only Look Once)

Advanced implementation

(Using customized model)



3.1 YOLO(You Only Look Once)

比較

以下の表は、AMB82-MINIとRTX 3090の計算能力を比較したものです。

表1.計算能力の比較

	TOPS(Tera Operations Per Second)
RTX 3090	285
AMB82-MINI	0.4

3.1 YOLO(You Only Look Once)

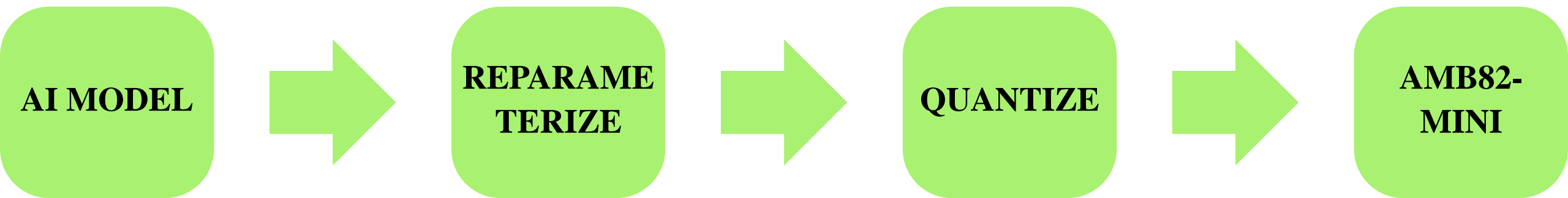
比較

以下の表は、AMB82-MINIとYOLOv7_TINYの容量を比較したものです。

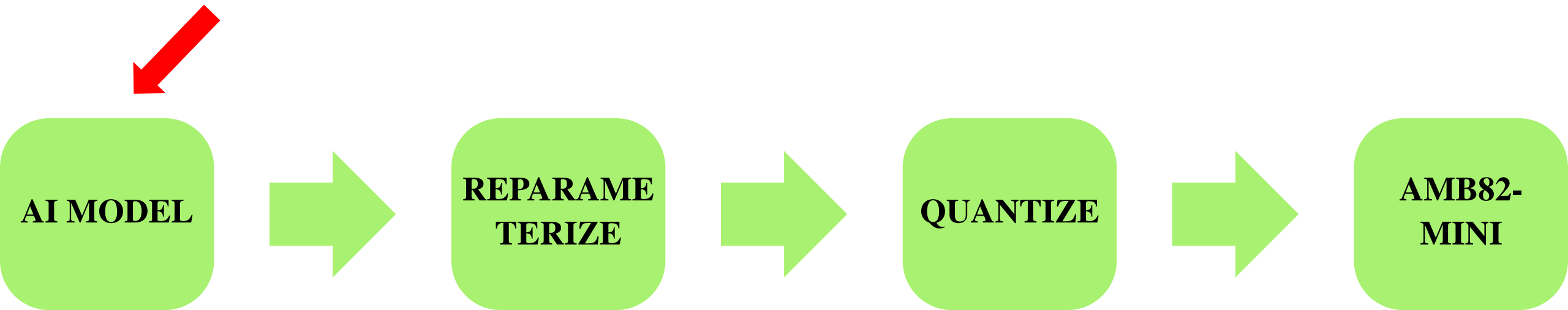
表1. 容量の比較

	MB(Megabyte)
YOLOv7_tiny	23
AMB82-MINI	16

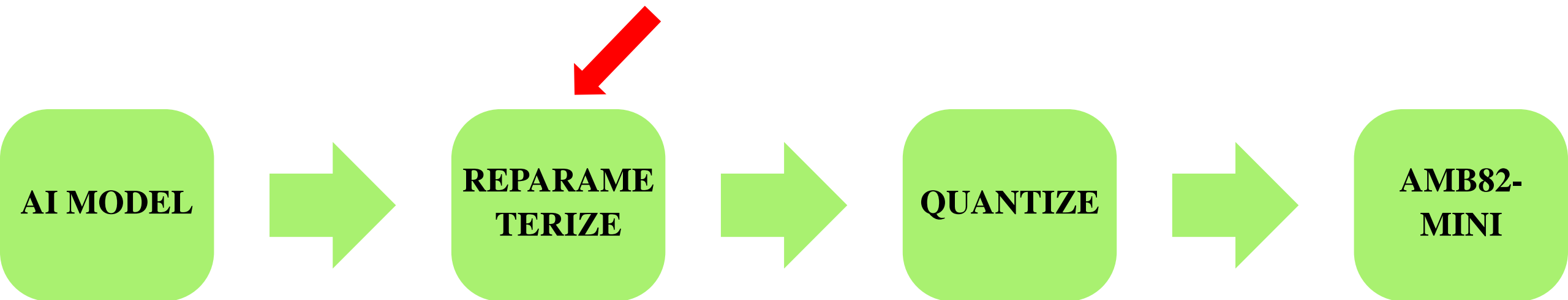
3.1 YOLO(You Only Look Once)



3.1 YOLO(You Only Look Once)



3.1 YOLO(You Only Look Once)



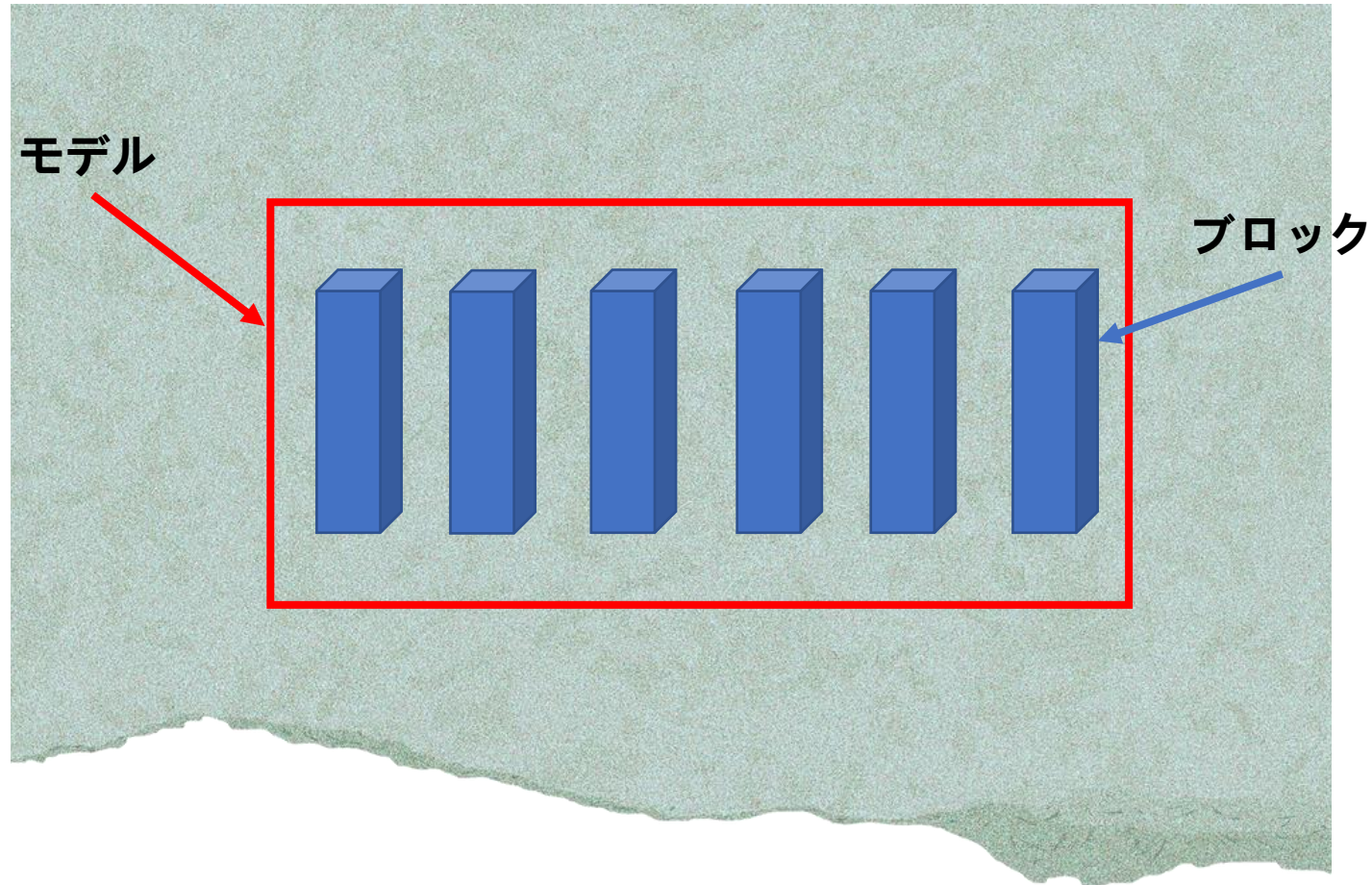
3.1 YOLO(You Only Look Once)

再パラメータ化

定義: 複数の**ブロック**を統合し、分岐を簡素化することで、モデルのパラメータ数を削減し、計算性能を向上させる手法。元のモデルは**訓練**のみに使用され、再パラメータ化されたモデルのみが保存され、**推論**に用いられる。

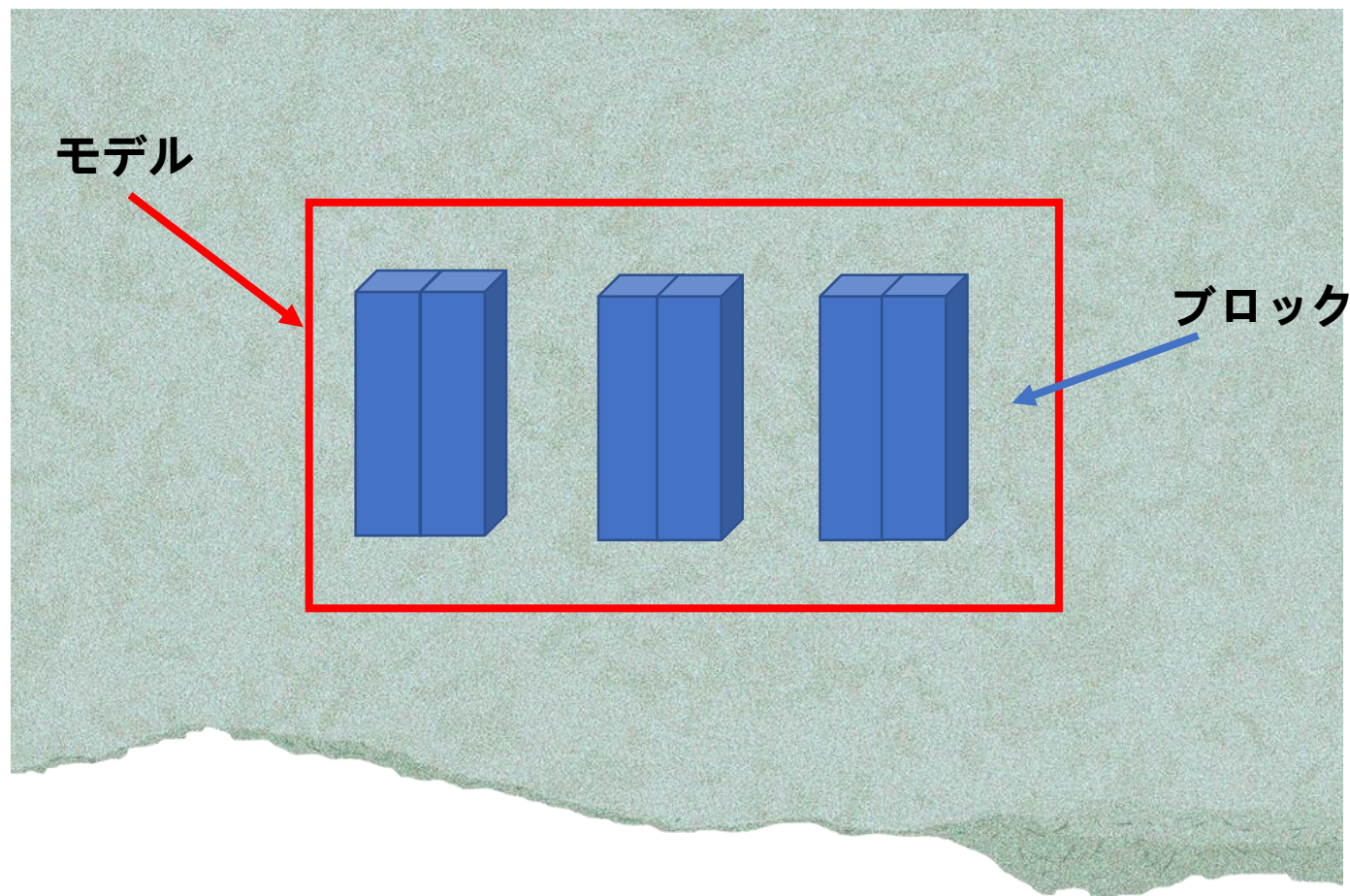
3.1 YOLO(You Only Look Once)

再パラメータ化



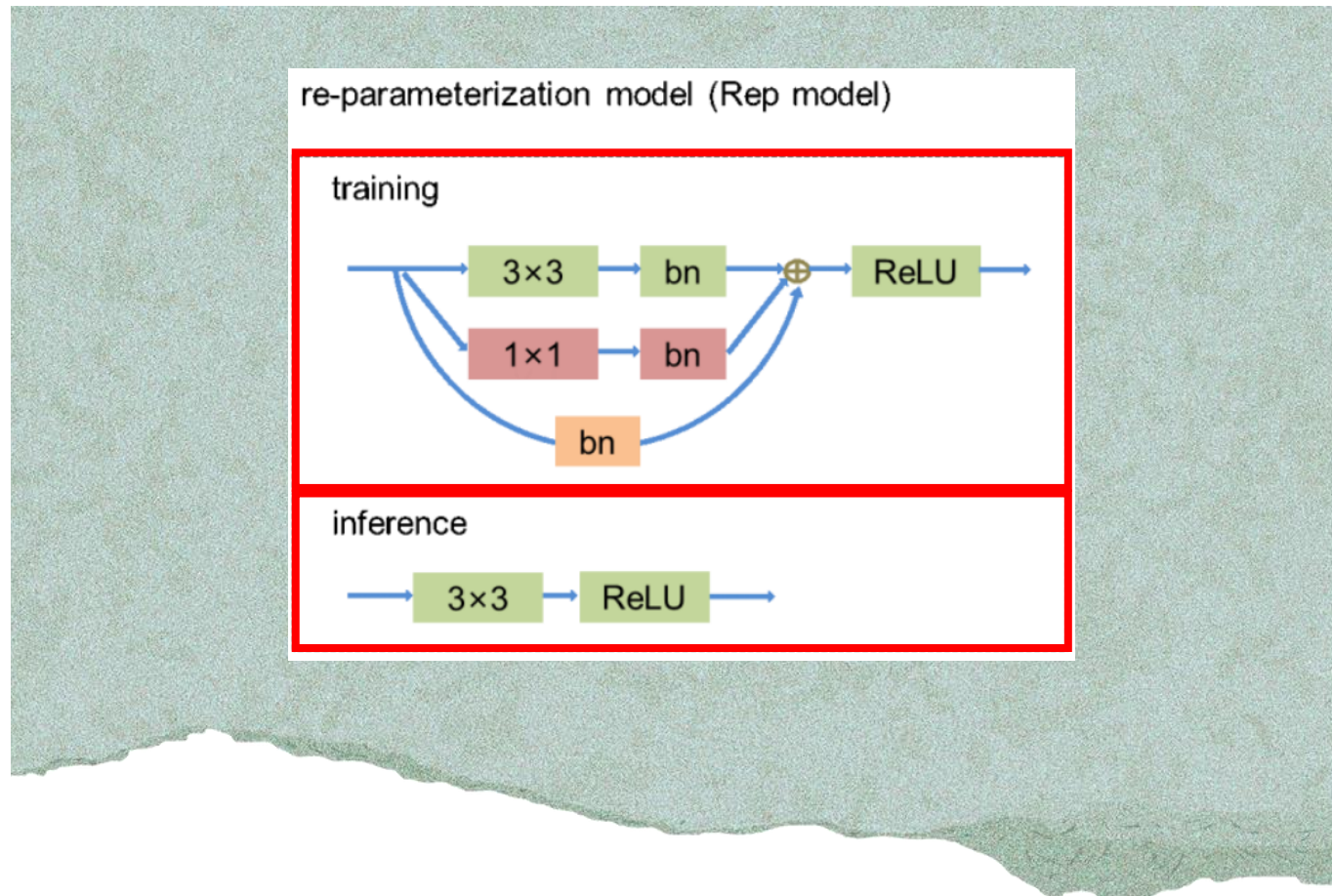
3.1 YOLO(You Only Look Once)

再パラメータ化

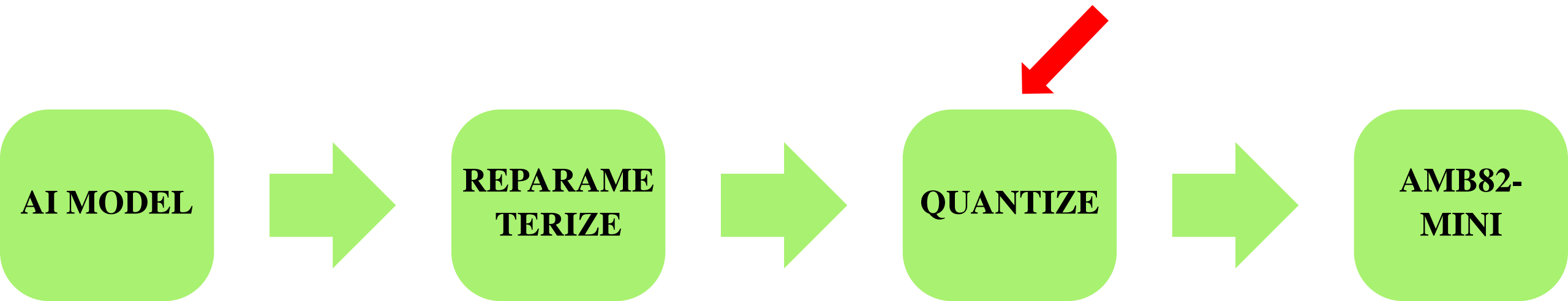


3.1 YOLO(You Only Look Once)

再パラメータ化



3.1 YOLO(You Only Look Once)



3.1 YOLO(You Only Look Once)

量子化

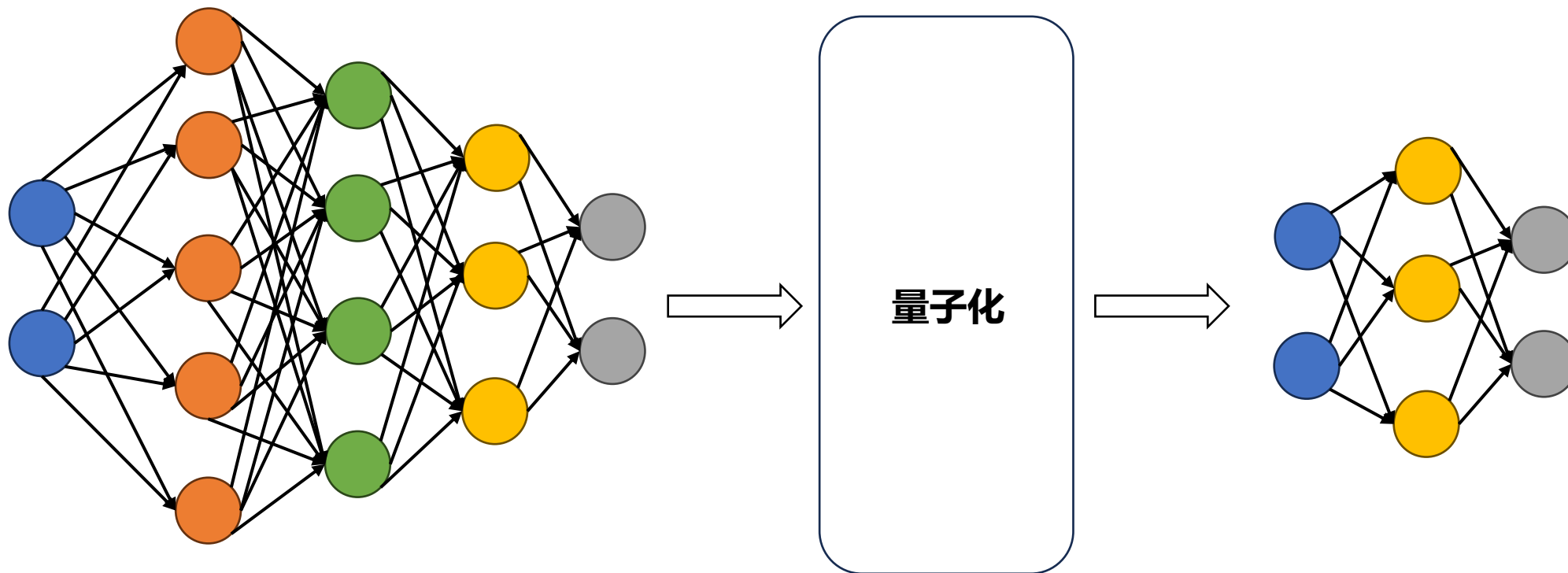
- **定義:** 高精度のパラメータを低精度に変換することで、モデルのサイズと計算量を大幅に削減し、推論速度と効率を向上させ、モバイルデバイスなどのリソース制限のある環境に適したものにします。

3.1 YOLO(You Only Look Once)

原始モデル

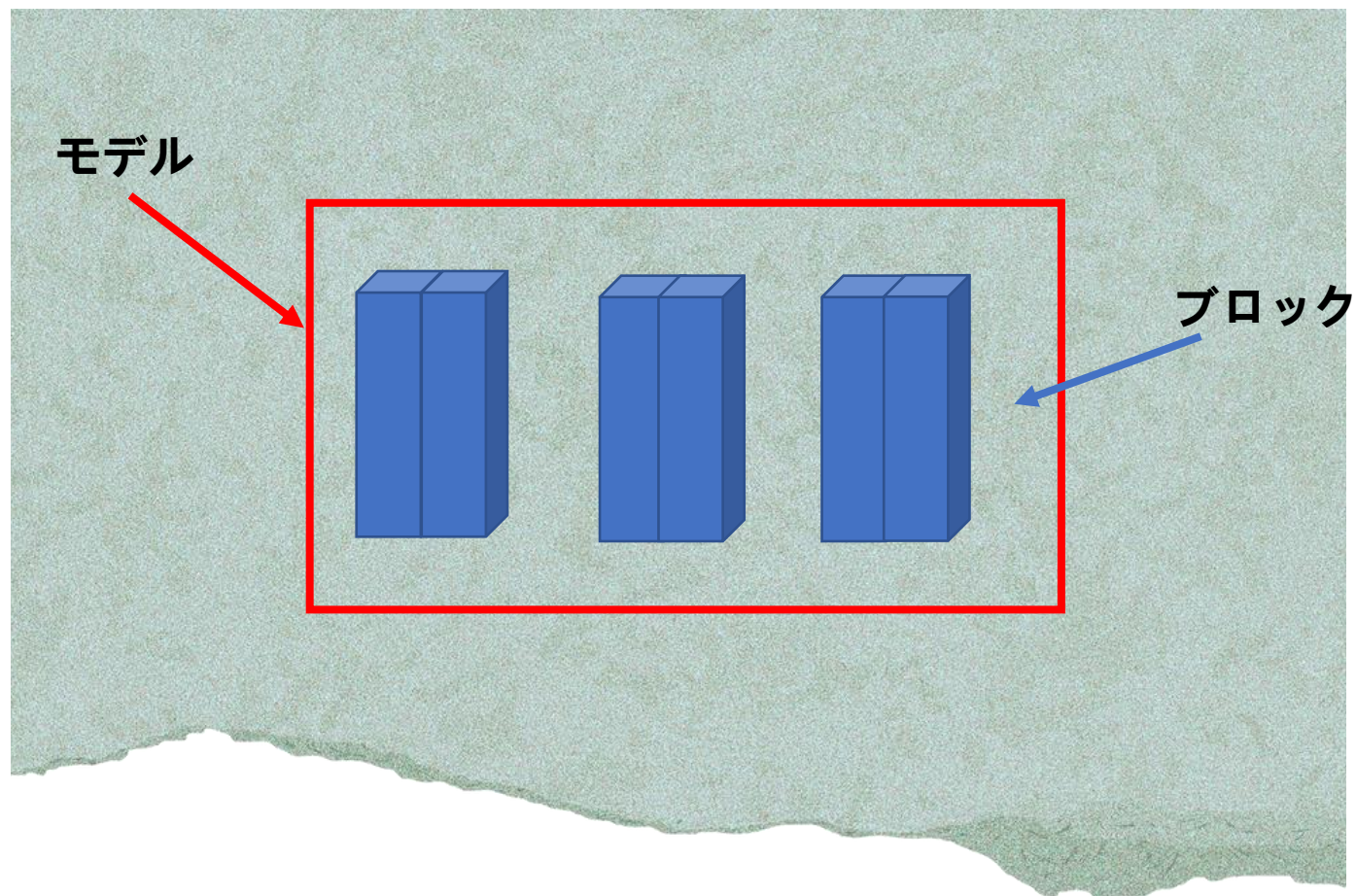
量子化

量子化モデル



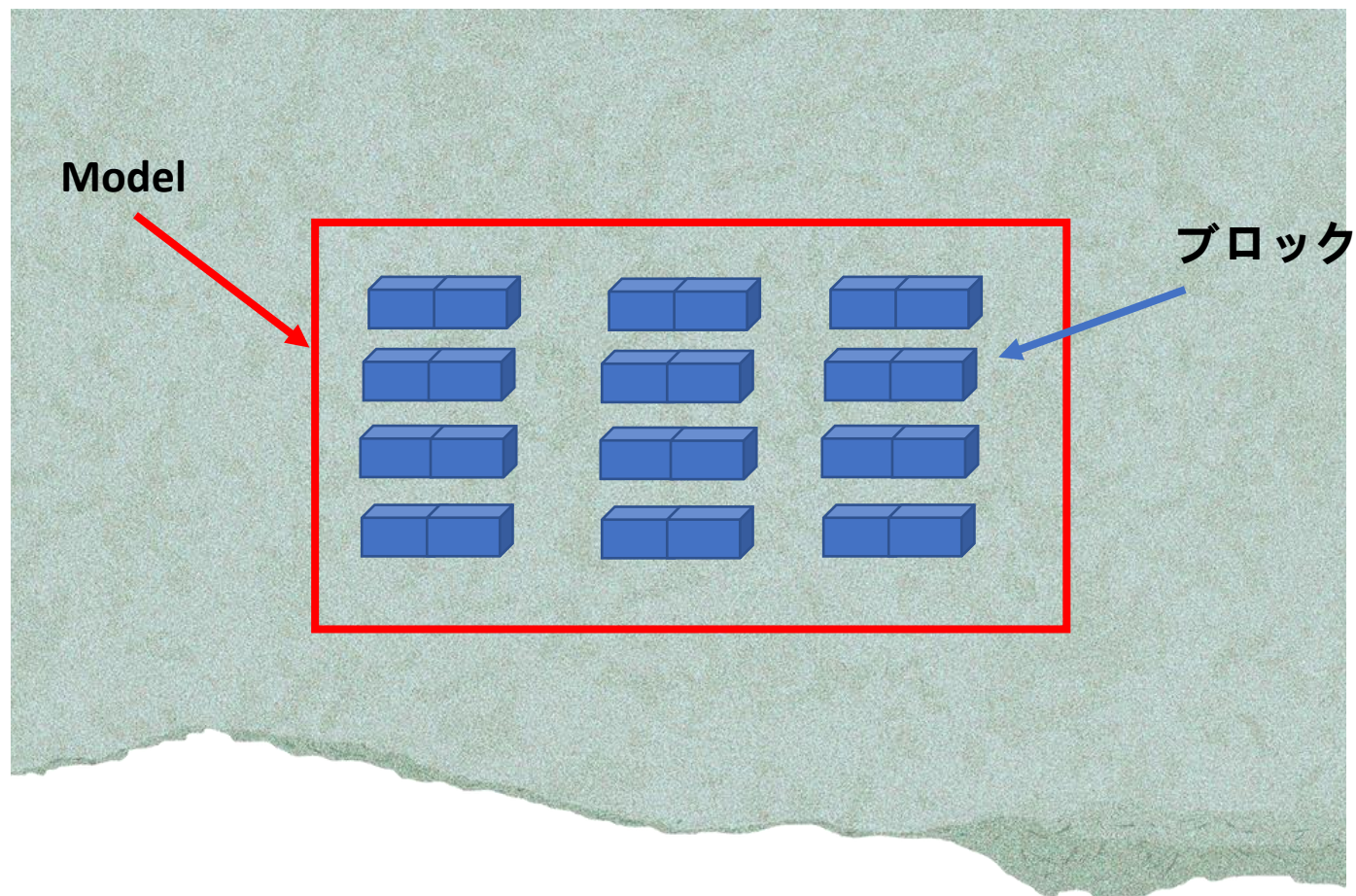
3.1 YOLO(You Only Look Once)

量子化

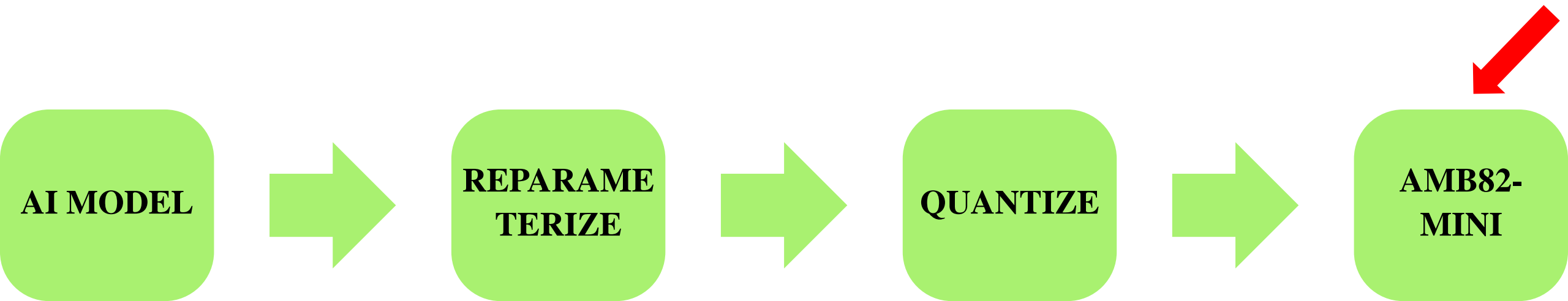


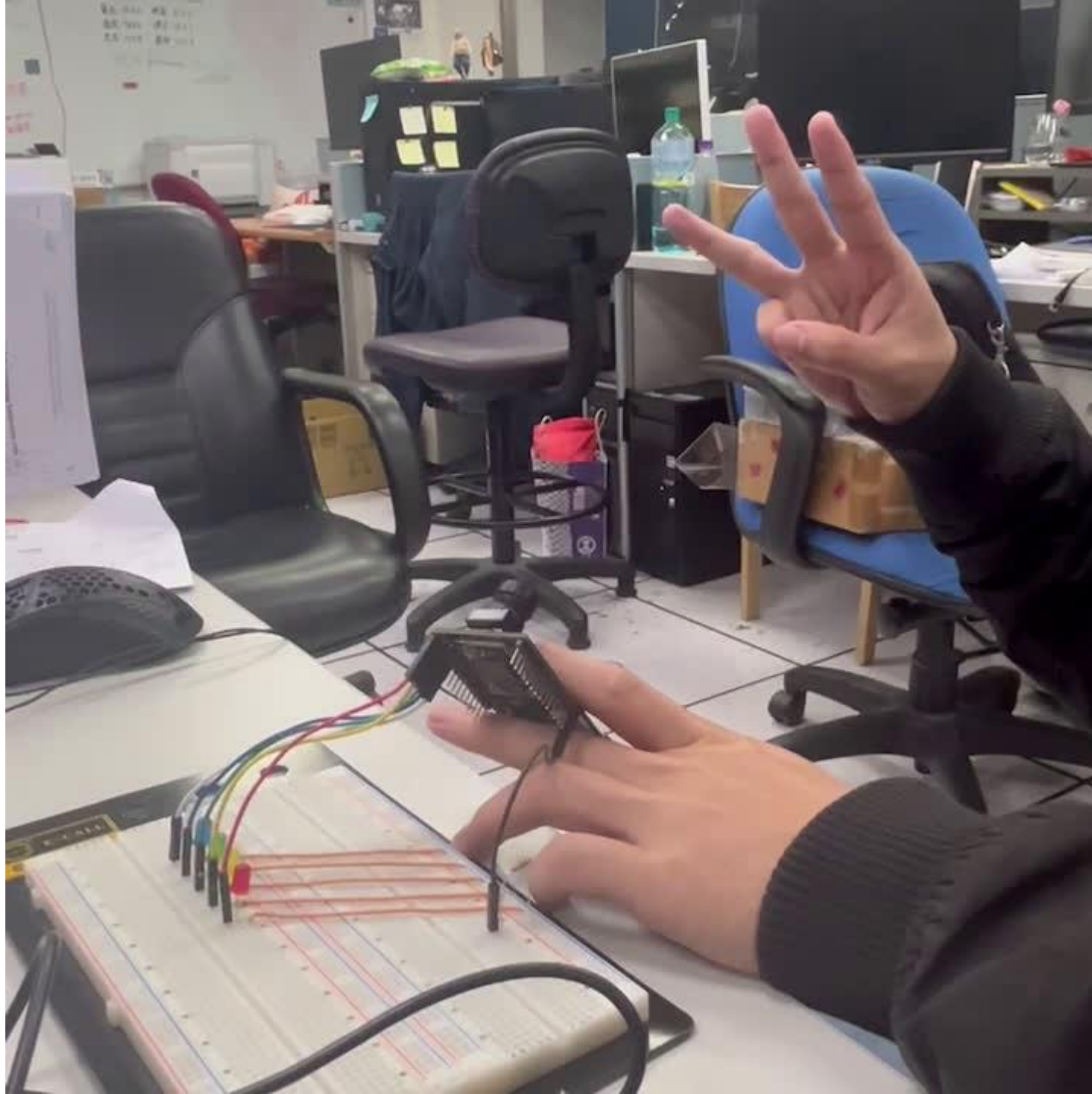
3.1 YOLO(You Only Look Once)

量子化



3.1 YOLO(You Only Look Once)





3.1 YOLO(You Only Look Once)

プログラミング

モデルを**デフォルトモデル**から**カスタマイズモデル**に切り替えます。

結果は次のようになる



```
ObjDet.configVideo(configNN);  
ObjDet.modelSelect(OBJECT_DETECTION, CUSTOMIZED_YOLOV7TINY, NA_MODEL, NA_MODEL);  
ObjDet.begin();
```

3.1 YOLO(You Only Look Once)

プログラミング

ヘッダファイル (.h) は、モデルの出力結果にカテゴリをマッピングする必要があります。

結果は次のようになる



```
#ifndef __OBJECTCLASSLIST_H__
#define __OBJECTCLASSLIST_H__

struct ObjectDetectionItem {
    uint8_t index;
    const char* objectName;
    uint8_t filter;
};

// List of objects the pre-trained model is capable of recognizing
// Index number is fixed and hard-coded from training
// Set the filter value to 0 to ignore any recognized objects
ObjectDetectionItem itemList[5] = {
    {0, "gesture1", 1},
    {1, "gesture2", 1},
    {2, "gesture3", 1},
    {3, "gesture4", 1},
    {4, "gesture5", 1}};

#endif
```

3.1 YOLO(You Only Look Once)

モデルのアップロード

まず、以下のリンクから**変換されたnbファイル**をダウンロードしてください。



https://drive.google.com/file/d/1Wsa2oWUZ4Sd_yjZKzTnHtIUJd38ibtlP/view?usp=sharing

3.1 YOLO(You Only Look Once)

モデルのアップロード

次に、**変換されたnbファイル**の名前を対応するモデルと同じに変更してください。対応するモデルは以下に示されています。この場合、名前を**yolov7_tiny.nb**に変更します。

Model for different tasks

Object Detection: “yolov3_tiny.nb”、”yolov4_tiny.nb” or **yolov7_tiny.nb**’

Face Detection: “scrfd_500m_bnkps_640x640_u8.nb”

Face Recognition: “mobilefacenet_int16.nb”

Audio related: “yamnet_fp16.nb” or “yamnet_s_hybrid.nb”

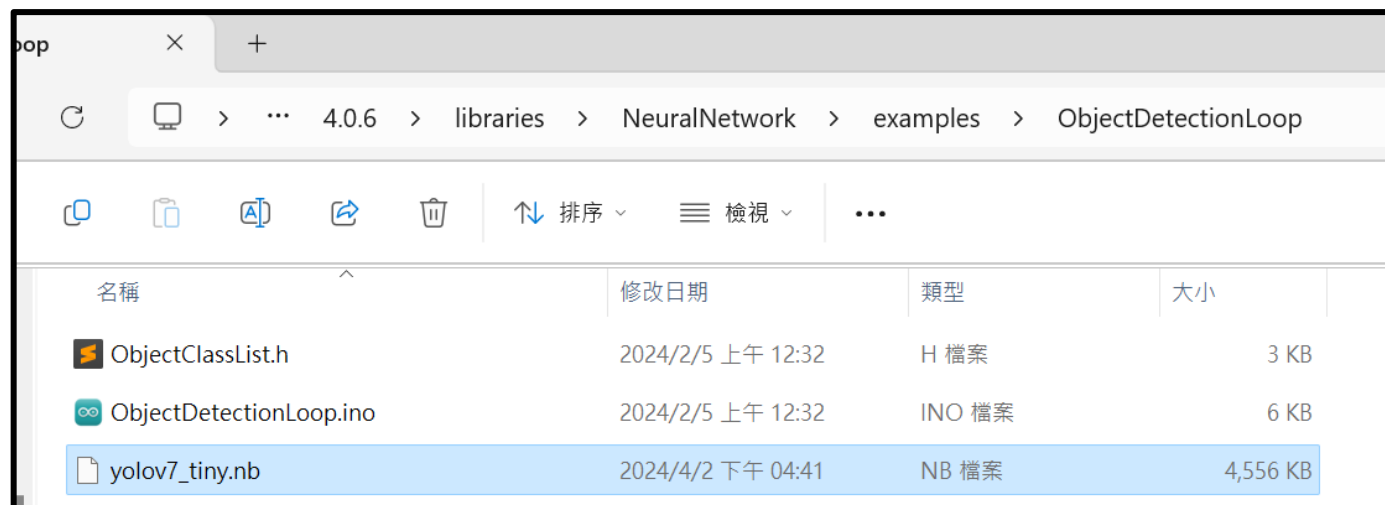
3.1 YOLO(You Only Look Once)

モデルのアップロード

最後に、以下のパスを見つけて、nbファイルを対応するタスクのフォルダに入れてください。

C:\Users**username**\AppData\Local\Arduino15\packages\realtek\hardware\AmebaPro2**version**\libraries\NeuralNetwork\examples**Corresponding task**

結果は次のようになる



3.1 YOLO(You Only Look Once)

実装には、以下の3点をコードに追加する必要があります。

1. プログラムの最初に追加:

(ピンを定義する)

```
int gesture1 = 0 ;  
int gesture2 = 1 ;  
int gesture3 = 2 ;  
int gesture4 = 3 ;  
int gesture5 = 4 ;
```

3.1 YOLO(You Only Look Once)

実装には、以下の3点をコードに追加する必要があります。

2. 関数void setup()に追加:

(出力を定義されたピンに送る)

```
pinMode(gesture1, OUTPUT);  
pinMode(gesture2, OUTPUT);  
pinMode(gesture3, OUTPUT);  
pinMode(gesture4, OUTPUT);  
pinMode(gesture5, OUTPUT);
```

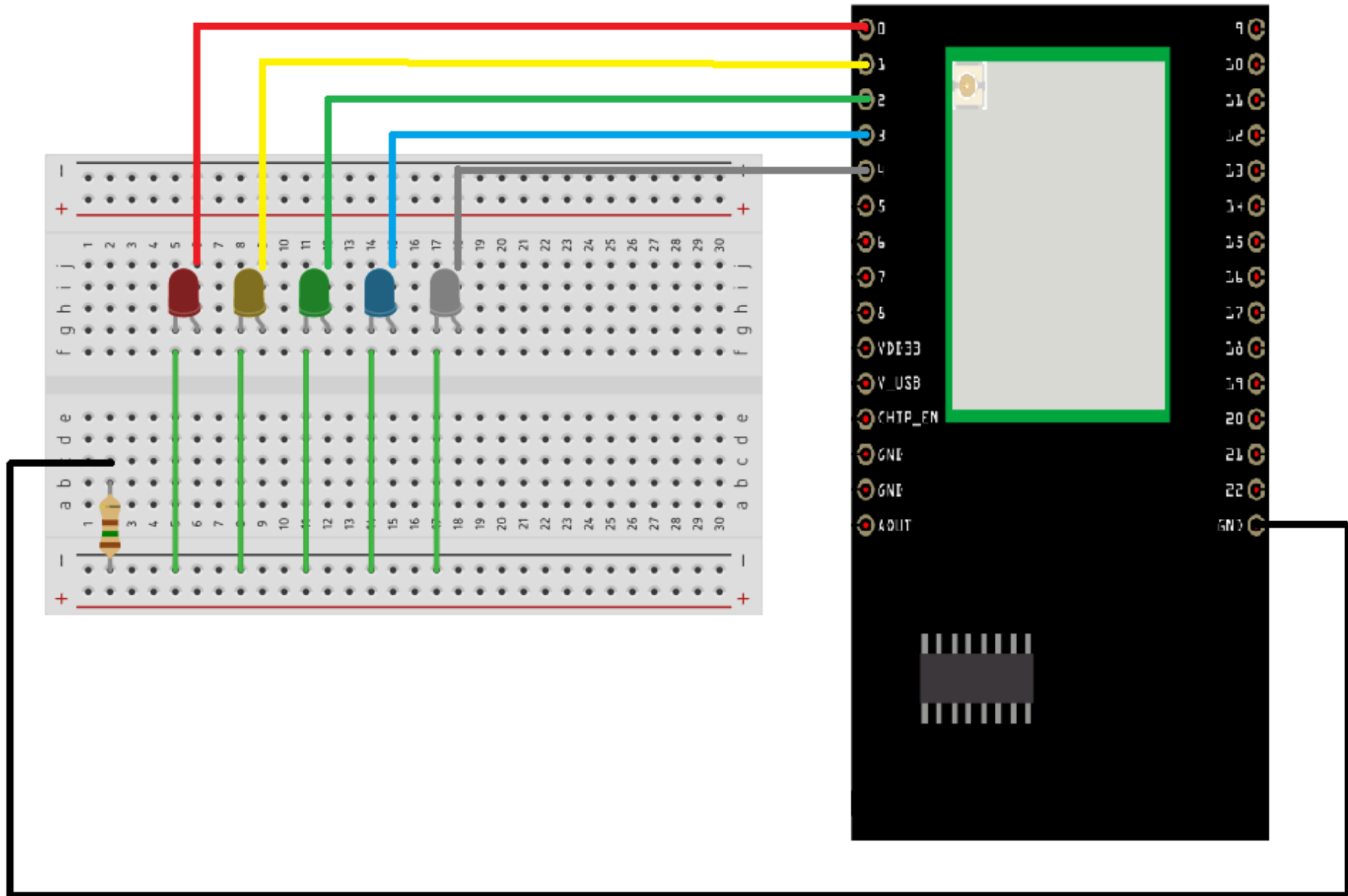

3.1 YOLO(You Only Look Once)

3. 関数void loop()内の **if(itemList[obj_type].filter)**の下に追加: (検出結果がどの指であるかを判定する)

```
if(obj_type==0) //finger1
{
    digitalWrite(gesture1, HIGH);
    delay(1000);
    digitalWrite(gesture1, LOW);
    delay(1000);
}

else if(obj_type==1) //finger2
{
    digitalWrite(gesture2, HIGH);
    delay(1000);
    digitalWrite(gesture2, LOW);
    delay(1000);
}
```

```
else if(obj_type==2)//finger3
{
    digitalWrite(gesture3, HIGH);
    delay(1000);
    digitalWrite(gesture3, LOW);
    delay(1000);
}
else if(obj_type==3) //finger4
{
    digitalWrite(gesture4, HIGH);
    delay(1000);
    digitalWrite(gesture4, LOW);
    delay(1000);
}
else if(obj_type==4) //finger5
{
    digitalWrite(gesture5, HIGH);
    delay(1000);
    digitalWrite(gesture5, LOW);
    delay(1000);
}
```

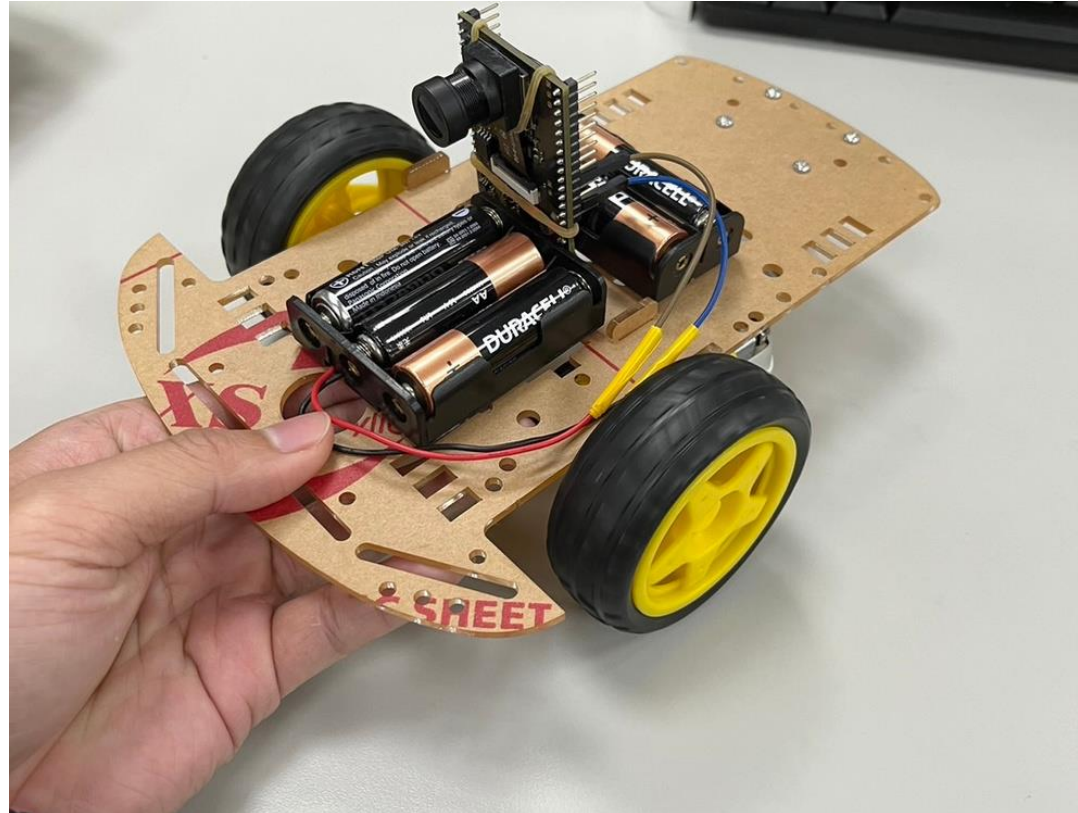




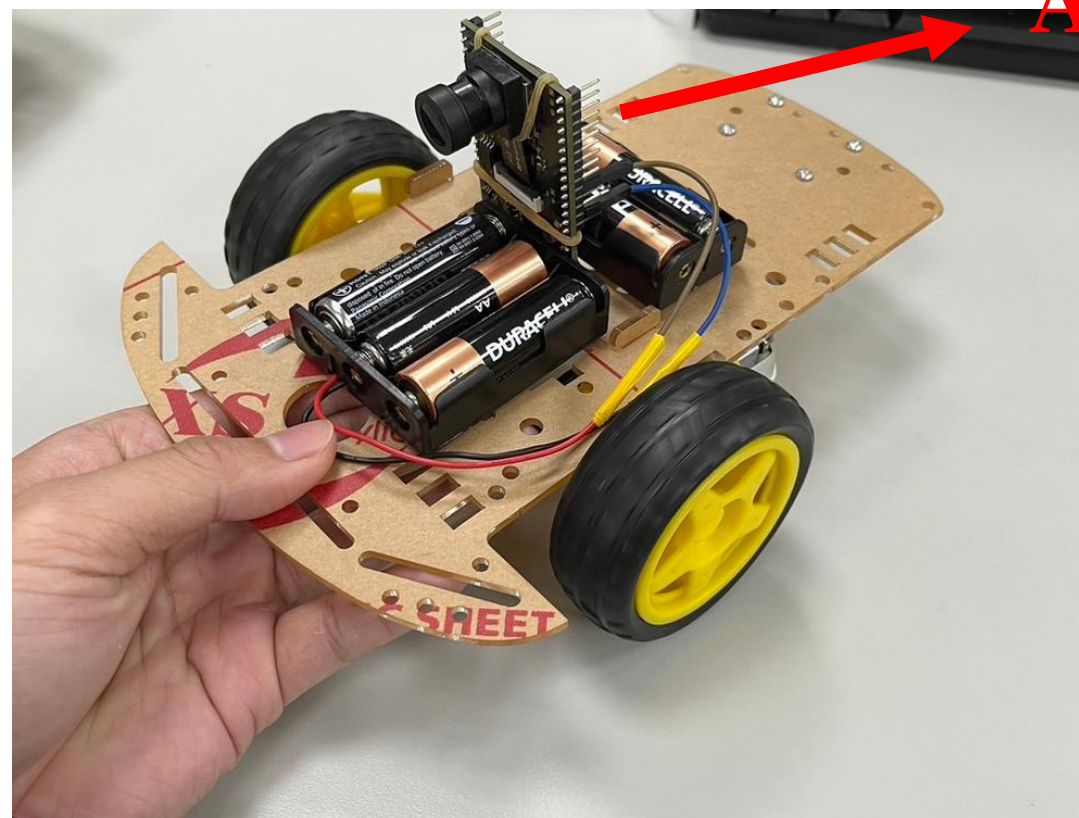
3.2 YOLOv7 Gesture Detection

(Gesture recognition Kart)

3.2 YOLOv7 Gesture Detection

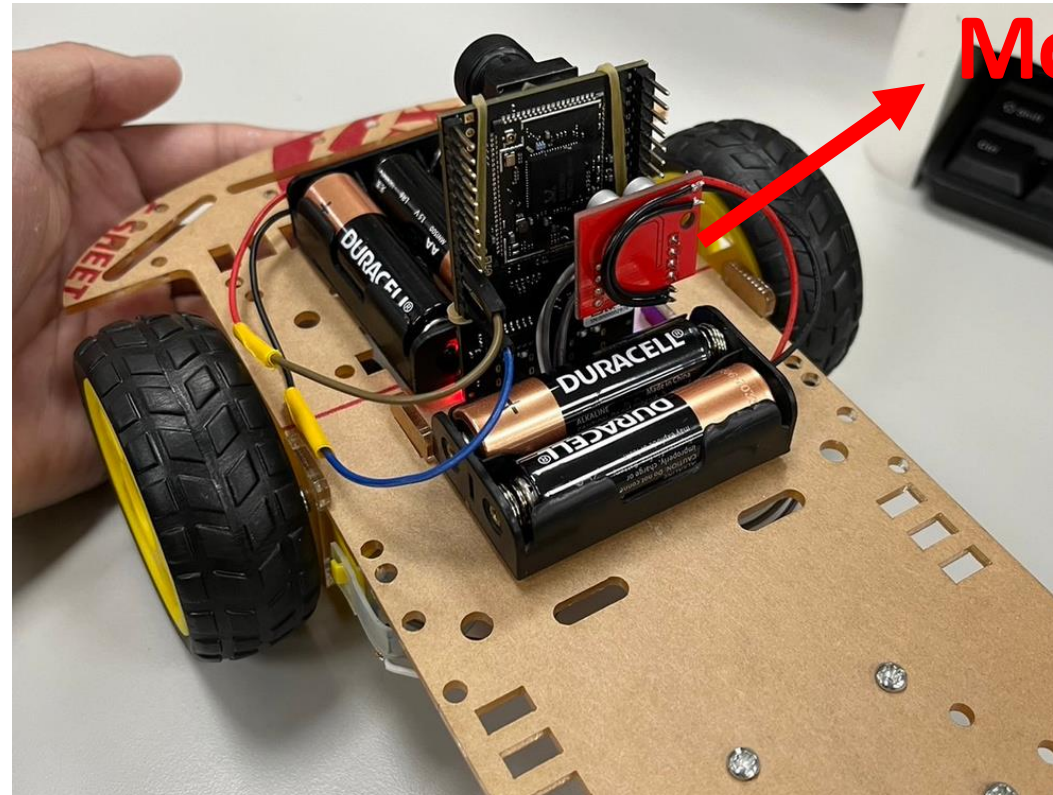


3.2 YOLOv7 Gesture Detection



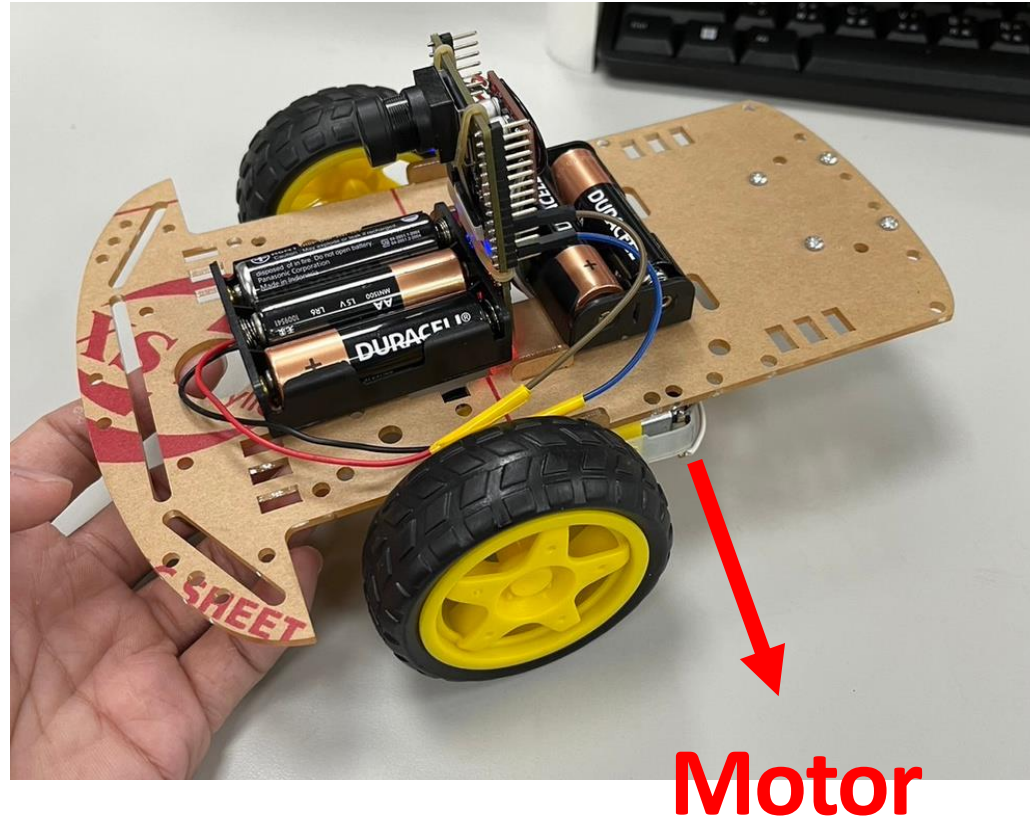
AMB82-MINI

3.2 YOLOv7 Gesture Detection



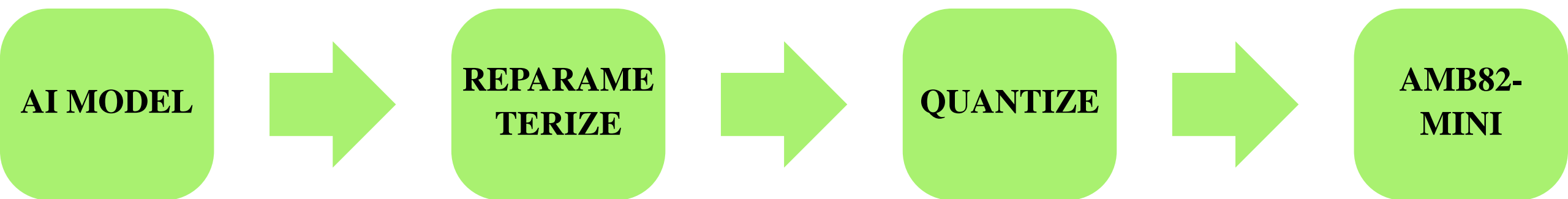
Motor control board

3.2 YOLOv7 Gesture Detection

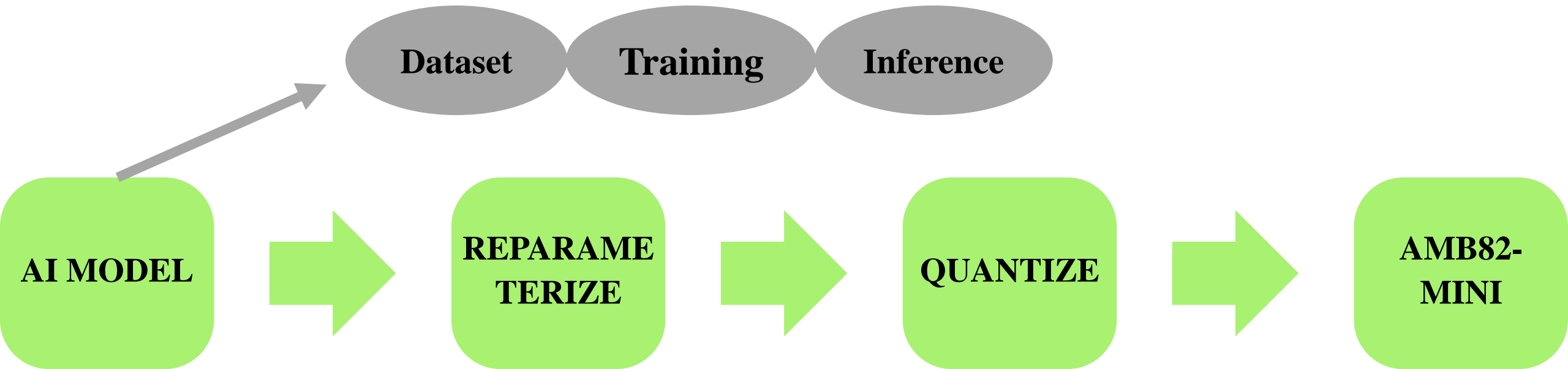


Introduction to AI model training

3.2 YOLOv7 Gesture Detection



3.2 YOLOv7 Gesture Detection



3.2 YOLOv7 Gesture Detection

プログラミング

01

GPIOピンとGPIOの
値を設定する

02

GPIOピンに出力値を
割り当てる

```
20  int a=19;
21  int b=20;
22  int c=21;
23  int d=22;
24
25
26  void setup() {
27      Serial.begin(115200);
28      pinMode(a, OUTPUT);
29      pinMode(b, OUTPUT);
30      pinMode(c, OUTPUT);
31      pinMode(d, OUTPUT);
```

3.2 YOLOv7 Gesture Detection

プログラミング

- 01 遅延時間の問題を軽減するためにcurrentMillisを設定する
- 02 検出結果の信頼度が50を超えるジェスチャーを分類する。
- 03 結果は、最も高い信頼度スコアを持つカテゴリによって決定される。

```
66 unsigned long previousMillis = 0;
67 const long interval = 200;
68
69 void loop() {
70     unsigned long currentMillis = millis();
71
72     if (currentMillis - previousMillis >= interval) {
73
74         previousMillis = currentMillis;
75
76         std::vector<ObjectDetectionResult> results = ObjDet.getResult();
77         int highestScoreIndex = -1;
78         float highestScore = 50;
79         for (int i = 0; i < ObjDet.getResultCount(); i++) {
80             if (results[i].score() > highestScore) {
81                 highestScore = results[i].score();
82                 highestScoreIndex = i;
83             }
84         }
85
86         if (highestScoreIndex != -1) {
87             int obj_type = results[highestScoreIndex].type();
88
```

3.2 YOLOv7 Gesture Detection

プログラミング

予測されたカテゴリーを車の動作
にマッチさせます。

```
86 ▼ if (highestScoreIndex != -1) {
87 ▼   int obj_type = results[highestScoreIndex].type();
88
89     if(obj_type==0) //前進
90     {
91         digitalWrite(a, 1); //右前
92         digitalWrite(b, 0);
93         digitalWrite(c, 1); //左前
94         digitalWrite(d, 0);
95     }
96
97     else if(obj_type==1) //左轉後前進
98     {
99         digitalWrite(a, 0);
100        digitalWrite(b, 0);
101        digitalWrite(c, 1);
102        digitalWrite(d, 0);
103        delay(200); // 維持此狀態0.2秒
104        digitalWrite(a, 1);
105        digitalWrite(b, 0);
106        digitalWrite(c, 1);
107        digitalWrite(d, 0);
108    }
109
```

3.2 YOLOv7 Gesture Detection

プログラミング

予測されたカテゴリーを車の動作
にマッチさせます。

```
111     else if(obj_type==2)//右轉後前進
112     {
113         digitalWrite(a, 1);
114         digitalWrite(b, 0);
115         digitalWrite(c, 0);
116         digitalWrite(d, 0);
117         delay(200);           // 維持此狀態0.2秒
118         digitalWrite(a, 1);
119         digitalWrite(b, 0);
120         digitalWrite(c, 1);
121         digitalWrite(d, 0);
122     }
123
124     else if(obj_type==3)//後退
125     {
126         digitalWrite(a, 0);
127         digitalWrite(b, 1);
128         digitalWrite(c, 0);
129         digitalWrite(d, 1);
130     }
131
132     else if(obj_type==4)//停車
133     {
134         digitalWrite(a, 0);
135         digitalWrite(b, 0);
136         digitalWrite(c, 0);
137         digitalWrite(d, 0);
138     }
139
140
141     }
142     }
143     OSD.update(CHANNEL);
144     delay(100);
145 }
```

3.2 YOLOv7 Gesture Detection

Code

https://drive.google.com/file/d/1AmEl6jfby3BS6mAt2LfuXeCrq86qEFV5/view?usp=drive_link

3.2 YOLOv7 Gesture Detection

プログラミング

ヘッダファイル (.h) は、モデルの出力結果にカテゴリをマッピングする必要があります。

結果は次のようになる



```
#ifndef __OBJECTCLASSLIST_H__
#define __OBJECTCLASSLIST_H__

struct ObjectDetectionItem {
    uint8_t index;
    const char* objectName;
    uint8_t filter;
};

// List of objects the pre-trained model is capable of recognizing
// Index number is fixed and hard-coded from training
// Set the filter value to 0 to ignore any recognized objects
ObjectDetectionItem itemList[5] = {
    {0, "gesture1", 1},
    {1, "gesture2", 1},
    {2, "gesture3", 1},
    {3, "gesture4", 1},
    {4, "gesture5", 1}};

#endif
```


3.2 YOLOv7 Gesture Detection

モデルのアップロード

まず、**変換されたnbファイル**の名前を対応するモデルと同じに変更してください。対応するモデルは以下に示されています。この場合、名前を **yolov7_tiny.nb** に変更します。

Model for different tasks

Object Detection: “yolov3_tiny.nb”、”yolov4_tiny.nb” or **yolov7_tiny.nb** ’

Face Detection: “scrfd_500m_bnkps_640x640_u8.nb”

Face Recognition: “mobilefacenet_int16.nb”

Audio related: “yamnet_fp16.nb” or “yamnet_s_hybrid.nb”

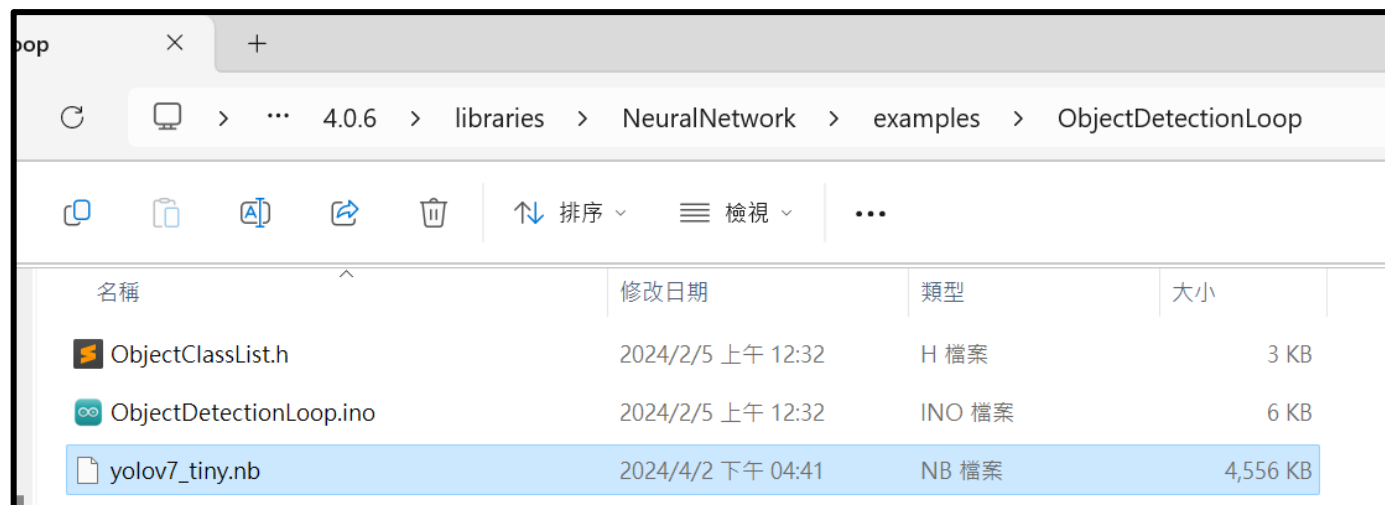
3.2 YOLOv7 Gesture Detection

モデルのアップロード

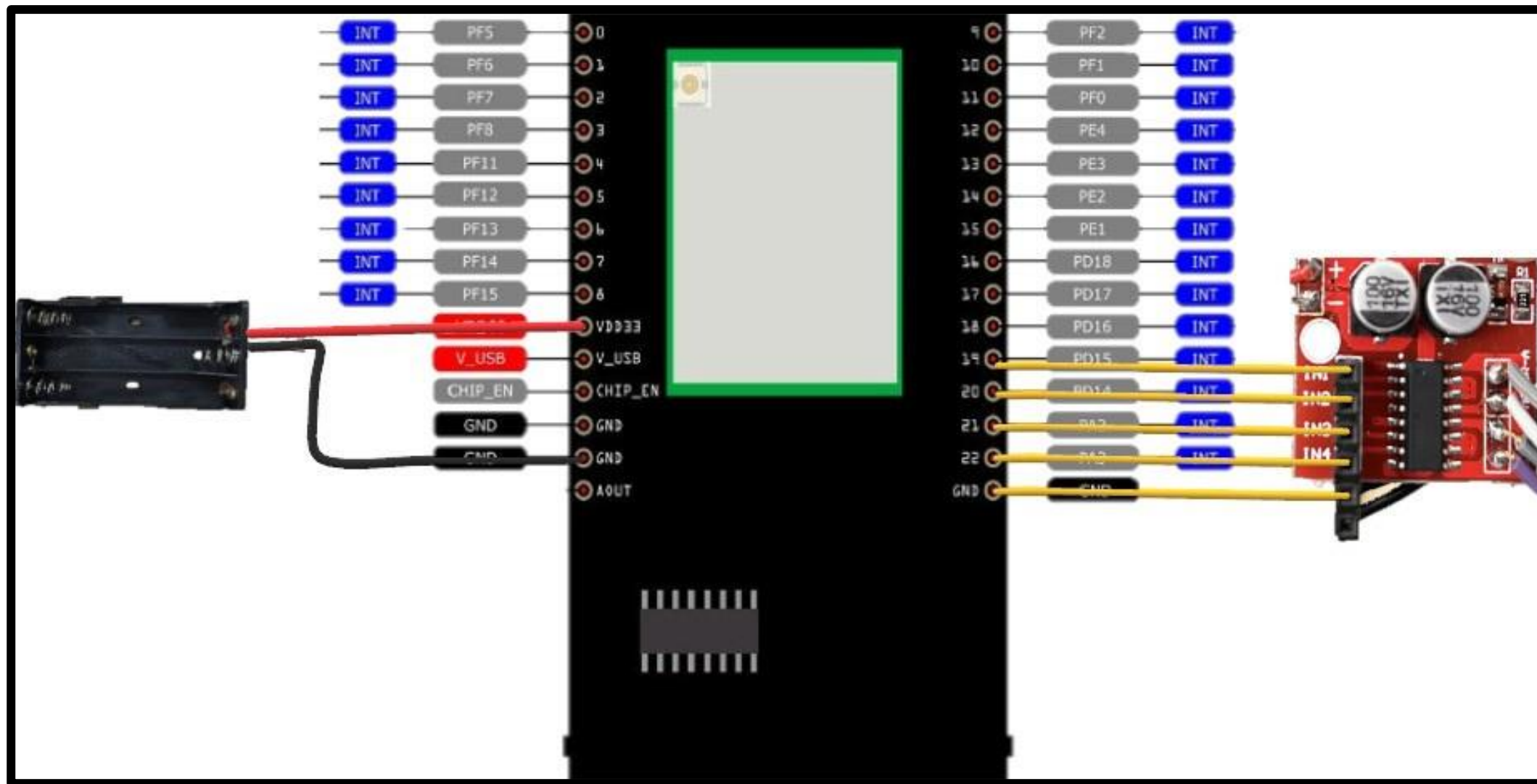
次に、以下のパスを見つけて、nbファイルを対応するタスクのフォルダに入れてください。

C:\Users**username**\AppData\Local\Arduino15\packages\realtek\hardware\AmebaPro2**version**\libraries\NeuralNetwork\examples**Corresponding task**

結果は次のようになる



3.2 YOLOv7 Gesture Detection



3.2 YOLOv7 Gesture Detection

DEMO Video :

