CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE SISTEMÁS ELECTRÓNICOS

INGENIERÍA EN ELECTRÓNICA

APRENDIZAJE AUTOMÁTICO EN SISTEMAS EMBEBIDOS

DNN Detección de Cubrebocas

ALUMNA:

HIDALGO RODRÍGUEZ MELISSA

FUANTOS DÍAS JORGE EDUARDO

INTRODUCCIÓN

El presente reporte aborda el desarrollo e implementación de un sistema de Detección de Cubrebocas utilizando un modelo de Red Neuronal Convolucional. El sistema utiliza el microcontrolador XIAO ESP32-S3 como servidor embebido para realizar la inferencia del modelo en tiempo real.

Los objetivos principales de esta implementación son:

- Convertir y optimizar el modelo para Visión Artificial a formato TensorFlow Lite Micro, aplicando Cuantización para reducir el tamaño y mejorar la velocidad de inferencia.
- Gestionar de manera efectiva la memoria restringida del microcontrolador.
- Desarrollar el firmware en el ESP32-S3 para funcionar como servidor web, manejando *streaming* de video y captura de imágenes

DESARROLLO

Para el entrenamiento del modelo se empleó el dataset de edX TinyML Specialization el cual consiste en imágenes de personas con cubrebocas ficticio y sin cubrebocas.

Se modificaron las imágenes para estar en escala de grises con dimensiones de 96X96 pixeles. Además, se incrementó el dataset aplicando transformaciones de rotación, espejeo y variación de la iluminación.

Inicialmente se contaban con 1000 imágenes en el dataset con etiquetas de con y sin mascarilla, después de aumentar el dataset se generaron 19 bloques con 602 fotos cada uno.

Antes de aumentar el dataset:

```
Found 1000 files belonging to 2 classes.
```

Después de aumentar el datset:

```
(TensorSpec(shape=(None, 96, 96, 1), dtype=tf.float32, name=None)
Total number of batches in the dataset: 19
Total number of elements in the dataset: 602
```

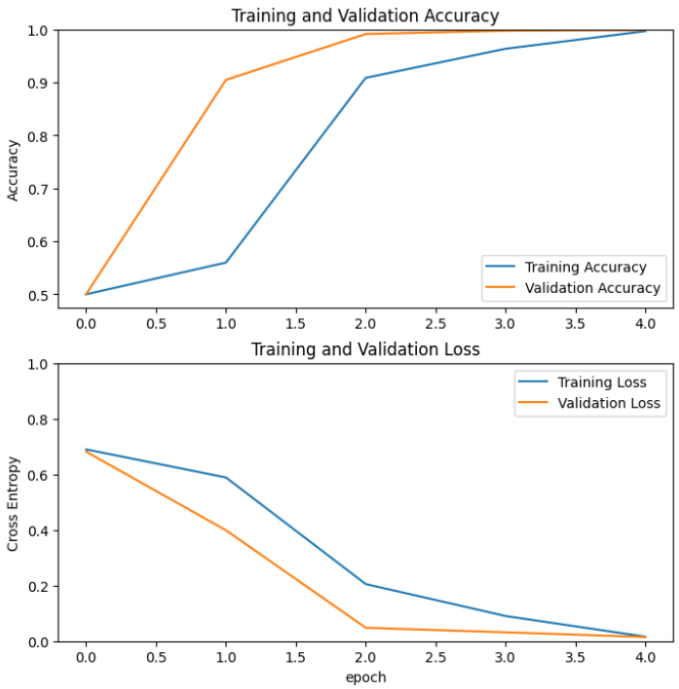
Para la definición de la arquitectura del modelo se empleó una secuencia de una capa convolucional seguida por una capa max poooling, repetida 5 veces, pero variando los tamaños entre cada secuencia, posteriormente una capa convolucional seguida de una capa de global average pooling y finalmente 3 capas densas.

Layer (type)	Output Shape	Param #
conv2d_21 (Conv2D)	(None, 96, 96, 6)	60
max_pooling2d_18 (MaxPooling2D)	(None, 48, 48, 6)	0
conv2d_22 (Conv2D)	(None, 48, 48, 12)	660
max_pooling2d_19 (MaxPooling2D)	(None, 24, 24, 12)	0
conv2d_23 (Conv2D)	(None, 24, 24, 24)	2,616
max_pooling2d_20 (MaxPooling2D)	(None, 12, 12, 24)	0
conv2d_24 (Conv2D)	(None, 12, 12, 48)	10,416
max_pooling2d_21 (MaxPooling2D)	(None, 6, 6, 48)	0
conv2d_25 (Conv2D)	(None, 6, 6, 96)	41,568
max_pooling2d_22 (MaxPooling2D)	(None, 3, 3, 96)	0
conv2d_26 (Conv2D)	(None, 3, 3, 192)	166,080
global_average_pooling2d_4 (GlobalAveragePooling2D)	(None, 192)	0
dense_10 (Dense)	(None, 128)	24,704
dense_11 (Dense)	(None, 32)	4,128
dense_12 (Dense)	(None, 1)	33

Total params: 250,265 (977.60 KB)

El modelo se entrenó por 5 épocas con optimizador Adam y tasa de aprendizaje de 0.001.

```
Epoch 1/5
19/19 ━━━━━━━━━━━━━━━━━━━━ 43s 2s/step - accuracy: 0.5154 - loss: 0.6912 - val_accuracy: 0.5000 - val_loss: 0.6815
Epoch 2/5
19/19 ━━━━━━━━━━━━━━━━━━━━ 95s 3s/step - accuracy: 0.5400 - loss: 0.6399 - val_accuracy: 0.9047 - val_loss: 0.3989
Epoch 3/5
19/19 ━━━━━━━━━━━━━━━━━━━━ 34s 2s/step - accuracy: 0.8791 - loss: 0.2665 - val_accuracy: 0.9913 - val_loss: 0.0482
Epoch 4/5
19/19 ━━━━━━━━━━━━━━━━━━━━ 54s 3s/step - accuracy: 0.9627 - loss: 0.0899 - val_accuracy: 0.9975 - val_loss: 0.0318
Epoch 5/5
19/19 ━━━━━━━━━━━━━━━━━━━━ 35s 2s/step - accuracy: 0.9927 - loss: 0.0235 - val_accuracy: 0.9988 - val_loss: 0.0148
```



Tuvo una exactitud de 1 al evaluarlo con el dataset de validación.

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 640ms/step - accuracy: 1.0000 - loss: 0.0040
Test accuracy : 1.0
```

Se cuantizó el modelo reduciendo su tamaño a casi un cuarto del peso original.

```
Quantized model size = 261KBs.
```

Para el despliegue del modelo se pegaron los valores hexadecimales correspondientes al modelo en un vector, en total fueron 267,880 valores.

```
22335   [267864]= 0x73, [267865]= 0x65, [267866]= 0x72, [267867]= 0x76, [267868]= 0x69,
22336   [267876]= 0x75, [267877]= 0x6c, [267878]= 0x74, [267879]= 0x5f
22337
22338 };
22339 inline const int g_model_len = 267880;
```

Se apartaron 400 KB de memoria para manejar los tensores, donde se almacenan temporalmente todos los datos mientras se calcula la predicción.

```
18  constexpr int kTensorArenaSize = 400 * 1024;
```

Se registraron las operaciones necesarias para que el modelo pudiera implementarse, en este caso fueron 6 funciones.

```
42  static tflite::MicroMutableOpResolver<6> resolver;
43
44    if (resolver.AddConv2D() != kTfLiteOk) { ESP_LOGE("IA","AddConv2D failed"); return; }
45    if (resolver.AddMaxPool2D() != kTfLiteOk) { ESP_LOGE("IA","AddMaxPool2D failed"); return; }
46    if (resolver.AddAveragePool2D() != kTfLiteOk) { ESP_LOGE("IA","AddAveragePool2D failed"); return; }
47    if (resolver.AddFullyConnected() != kTfLiteOk) { ESP_LOGE("IA","AddFullyConnected failed"); return; }
48    if (resolver.AddQuantize() != kTfLiteOk) { ESP_LOGE("IA","AddQuantize failed"); return; }
49    if (resolver.AddMean() != kTfLiteOk) { ESP_LOGE("IA","AddMean failed"); return; }
```

Además, se creó y asignó el intérprete para poder realizar las inferencias, así como obtener los punteros de entrada y salida del interprete

```
52  static tflite::MicroInterpreter static_interpreter(
53        model, resolver, tensor_arena, kTensorArenaSize);
54  interpreter = &static_interpreter;
```

```
70  input = interpreter->input(0);
71  output = interpreter->output(0);
```

Para capturar las imágenes se configuró la cámara con 96x96 pixeles en escala de grises.

```
44          .pixel_format = PIXFORMAT_GRAYSCALE,
45          .frame_size = FRAMESIZE_96X96,
```

Para poder desplegar la imagen se configuró la ESP32 en Station Mode para conectarse al Access Point del teléfono y desplegar un servidor web embebido en el ESP32. En el servidor web se utilizó la ruta /stream para desplegar el video en formato MJPEG (Motion JPEG) enviando una serie rápida y continua de imágenes JPEG individuales. El navegador hace una única solicitud HTTP GET a /stream y el ESP32 responde con un tipo de contenido especial: multipart/x-mixed-replace para mandar la imagen JPEG y un delimitador.

```
192    httpd_uri_t uri_stream = {
193        .uri = "/stream",
194        .method = HTTP_GET,
195        .handler = jpg_stream_httpd_handler,
196        .user_ctx = NULL
197    };
```

```
90      res = httpd_resp_set_type(r: req, type: _STREAM_CONTENT_TYPE);
```

```
109         if(res == ESP_OK){  // Envia delimitador de inicio
110             res = httpd_resp_send_chunk(r: req, buf: _STREAM_BOUNDARY, buf_len: strlen(_STREAM_BOUNDARY));
111         }
112         if(res == ESP_OK){  // Envia los METADATOS del frame (Tamaño y Content-Type del JPEG)
113             size_t hlen = snprintf(part_buf, 64, _STREAM_PART, _jpg_buf_len);
114             res = httpd_resp_send_chunk(r: req, buf: part_buf, buf_len: hlen);
115         }
116         if(res == ESP_OK){  // Envía los DATOS BINARIOS del JPEG
117             res = httpd_resp_send_chunk(r: req, buf: (const char *)_jpg_buf, buf_len: _jpg_buf_len);
118         }
```

```
36      "        const streamUrl = \"/stream\";\n"
```

```
66      "                // 2. REANUDAR STREAM\n"
67      "                videoElement.src = streamUrl;\n"
68      "                document.getElementById('ia_result').innerHTML = \"Modo actual: Streaming\";\n"
69      "                isStreaming = true;\n"
```

Para realizar la inferencia se agregó un botón interactivo, cuando el usuario hace clic en el se cambia a la ruta /single_capture, donde se detiene el streaming mostrando una sola imagen a la que se le realiza la inferencia con el modelo de IA. A través de /single_capture el navegador pide una foto al ESP32. En el ESP32, es donde ocurre la captura de imagen y la ejecución de la IA, para ello se captura un frame de la cámara y se manda a una función de preprocesamiento donde la imagen se cuantiza.

```
102     // 2. Iterar sobre todos los píxeles (96 x 96 = 9216 píxeles)
103     for (int i = 0; i < image_len; i++) {
104
105         // El valor real es el byte de escala de grises (0-255)
106         float real_value = static_cast<float>(image_data[i]);
107
108         // Aplicar la fórmula de cuantificación (simétrica)
109         // La mayoria de los modelos de vision simetricos tienen un zero_point de 0.
110         // Formula: q = round(r / scale) + zero_point
111         int32_t quantized_value = static_cast<int32_t>(roundf(real_value / scale) + zero_point);
112
113         // Limitar el valor al rango de int8_t (-128 a 127)
114         if (quantized_value > 127) quantized_value = 127;
115         if (quantized_value < -128) quantized_value = -128;
116
117         // 3. Escribir el valor cuantizado en el tensor de entrada (int8_t)
118         input->data.int8[i] = static_cast<int8_t>(quantized_value);
119     }
```

Después del procesamiento los datos se encuentran ya en el tensor de entrada por lo que para realizar la inferencia solo se invoca al interprete y se lee el puntero de salida, donde valores positivos son sin cubrebocas y valores negativos con cubrebocas.

```
130     // 1. Ejecutar la inferencia
131     TfLiteStatus invoke_status = interpreter->Invoke();
```

```
139     // 2. Obtiene la salida
140     int8_t output_inf = output->data.int8[0];
```

```
145     // 4. Accion Output
146     if(output_inf > 0){
147         printf("\nSin Cubrebocas   |  Output : %d\n", output_inf);
148         return 1;
149     } else {
150         printf("\nCon Cubrebocas   |  Output : %d\n", output_inf);
151         return 0;
152     }
```

En base al resultado de la inferencia se modifica un string que se enviará como un encabezado junto con la imagen en formato JPEG a la que se le realizó la inferencia.

```
149     if (result == 1) {
150         strcpy(last_result_text, "SIN cubrebocas");
151     } else if (result == 0) {
152         strcpy(last_result_text, "CON cubrebocas");
153     } else {
154         strcpy(last_result_text, "Error en inferencia");
155     }
```

```
167     httpd_resp_set_type(r: req, type: "image/jpeg");
168     httpd_resp_set_hdr(r: req, field: "X-IA-Result", value: last_result_text);
169     esp_err_t res = httpd_resp_send(r: req, buf: (const char *)jpg_buf, buf_len: jpg_buf_len);
```

Del lado del cliente recibe la imagen y la empaqueta como un objeto Blob, crea una URL temporal para mostrar la imagen del Blob sin tener que guardarla en el disco, además actualiza el párrafo de resultado usando el header.

```
37      "        const captureUrl = \"/single_capture\";\n"
```

```
41      "                // 1. MODO FOTO\n"
42      "                document.getElementById('ia_result').innerHTML = \"Procesando foto...\";\n"
43      "                videoElement.src = 'about:blank'; // Pausar visualmente el stream\n"
44      "\n"
45      "                try {\n"
46      "                    // Llamar a la ruta que devuelve la imagen JPEG\n"
47      "                    const response = await fetch(captureUrl);\n"
48      "                    if (!response.ok) throw new Error('Capture failed');\n"
49      "\n"
50      "                    // Crear un Blob de la imagen devuelta y usarlo como fuente\n"
51      "                    const imageBlob = await response.blob();\n"
52      "                    const imageUrl = URL.createObjectURL(imageBlob);\n"
53      "                    \n"
54      "                    const iaResult = response.headers.get(\"X-IA-Result\") || \"Sin resultado\";\n"
55      "                    videoElement.src = imageUrl;\n"
56      "                    document.getElementById('ia_result').innerHTML = \"Resultado IA: <b>\" + iaResult
57      "                    isStreaming = false; // Cambiar estado a estático\n"
```

Por lo que el modo de uso de la interfaz gráfica es el siguiente:

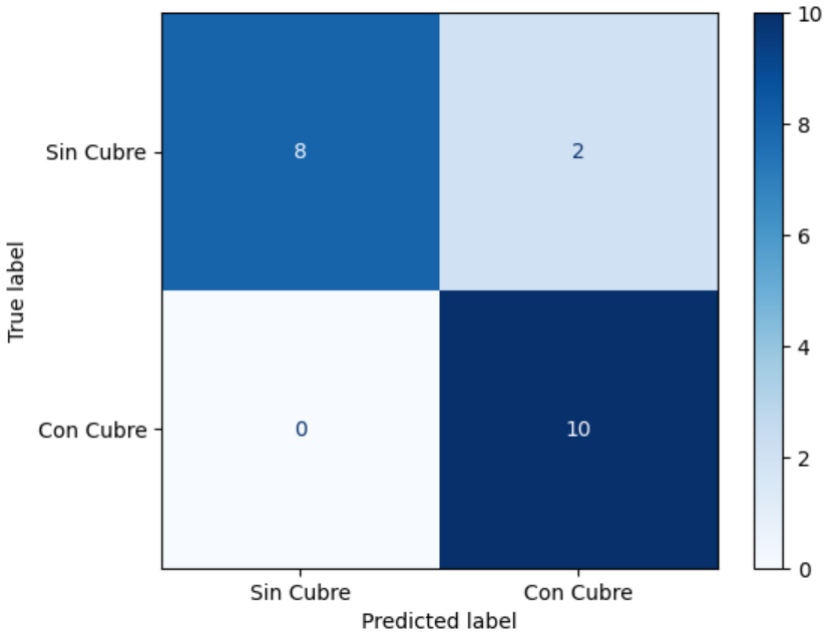
1. Conectar abrir el Access Point y verificar que el ESP32 se haya conectado
2. En un servidor web acceder a la IP del ESP32
3. Se encontrará en modo streaming, para capturar una foto dar clic en el botón
4. Se desplegará la foto y el resultado de la inferencia

5. Para retornar al modo streaming dar clic nuevamente en el botón

Métricas

Las matrices de confusión y las métricas de las inferencias del modelo en el MCU son:

Exactitud	0.9	
	Sin Cubrebocas	Con Cubrebocas
Precisión	1	0.83
Recall	0.8	1
F1 score	0.89	0.91



DISCUSIÓN

La elaboración de esta práctica arrojó resultados muy positivos en cuestión de su desempeño y optimización, pero hubo algunas decisiones que merecen ser analizadas.

El método para el despliegue del modelo, que consistió en pegar los valores hexadecimales en un vector, si bien fue funcional para la prueba de concepto, fue una desventaja en la mantenibilidad. Una alternativa mucho más practica habría sido cargar el modelo en formato .asm, lo que hubiera permitido encajar el modelo sin la necesidad de copiar manualmente una cantidad tan grande de valores, simplificando el proceso de actualización del modelo si fuera necesario, sin embargo, requeriría configurar las particiones de memoria para habilitar el sistema de archivos SPIFFS (SPI Flash File System) .

La implementación como servidor web con streaming de video y captura de imágenes fue un gran acierto, además de que facilitó la obtención de inferencias en tiempo real. Esta idea ofreció una interfaz de usuario intuitiva a través de la página web. La visibilidad de la imagen en tiempo real antes de la captura permite al usuario asegurarse de que el rostro esté bien centrado antes de hacer la inferencia. Su ventaja fue disminuir los errores de predicción causados por distorsión provocada por malas alineaciones.

CONCLUSIÓN

La elaboración de esta práctica hizo que se cumplieran los objetivos planteados en relación con el uso de IA en microcontroladores, como lo son:

- **Optimización de modelos**: La conversión y optimización del modelo a TFLite mediante cuantización redujo el tamaño del archivo, lo que facilitó su despliegue en otro dispositivo.

- **Gestión de memoria:** Se administró de buena manera la memoria RAM restringida del microcontrolador al reservar una cantidad adecuada de “arena” para los tensores y registrar solo las operaciones necesarias para la inferencia.

- **Desarrollo de interfaz**: Se creó un servidor web capaz de gestionar el streaming del video y la captura de imágenes, lo que complementó de buena manera la solución para el despliegue

Después de todo el desarrollo y puesta en práctica de la actividad, se puede decir que capacidad del sistema para realizar inferencias rápidamente en el microcontrolador nos da una buena aceptación del uso de arquitecturas de redes neuronales convolucionales optimizadas para aplicaciones de TinyML en dispositivos de IOT.