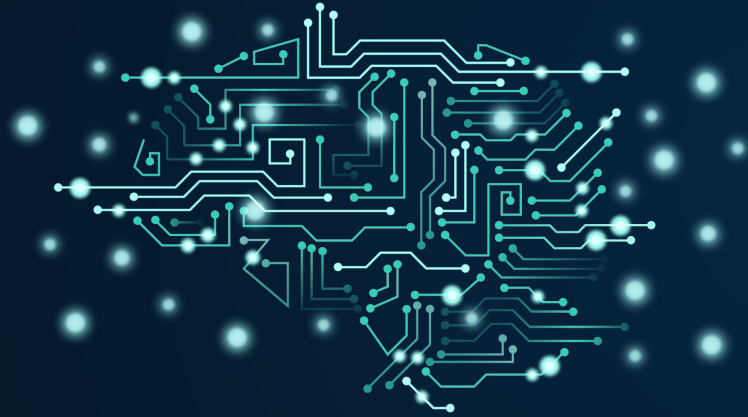




Sistema de seguridad de detección de caras con Arduino Nano 33 BLE Sense

**Luis Javier Herrera
Juan Ignacio Montealegre**



CONTEXTO/JUSTIFICACIÓN

- Importancia de los sistemas de seguridad y tecnología emergente que implementa Inteligencia Artificial.
- La creación de redes neuronales es cada vez más accesible y existen herramientas que lo facilitan.



OBJETIVOS

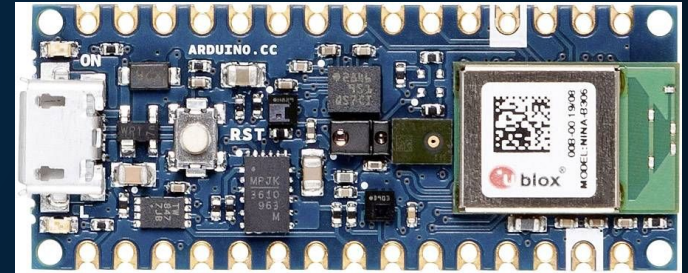
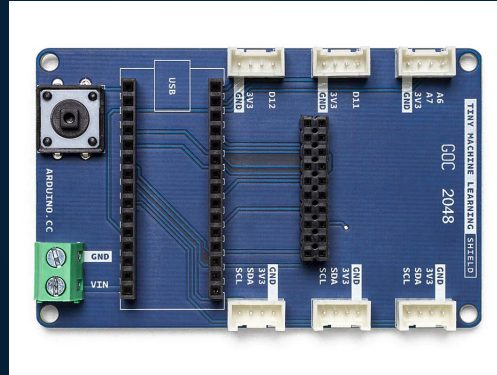
General

- Crear un sistema de seguridad funcional para la detección de rostros humanos y vestimenta utilizando el MCU Arduino NAno 33 BLE Sense

Específicos

- Lograr detectar un rostro humano por medio de la utilización de machine learning haciendo uso de un microcontrolador
- Implementar un sistema de comunicación por medio de IoT para el envío de datos en caso de detectar un rostro humano
- Profundizar conceptos abordados a lo largo del curso por medio de la investigación y práctica

COMPONENTES



PASOS PARA UNA RED NEURONAL



Recopilar Datos

Se obtienen los datos que se desean detectar y en ocasiones se “limpian”



Crear el modelo

Se define la estructura de la red neuronal a implementar



Entrenar el modelo

Se entrena el modelo con una parte de los datos y luego se pone a prueba con el resto de datos.



Exportar el modelo

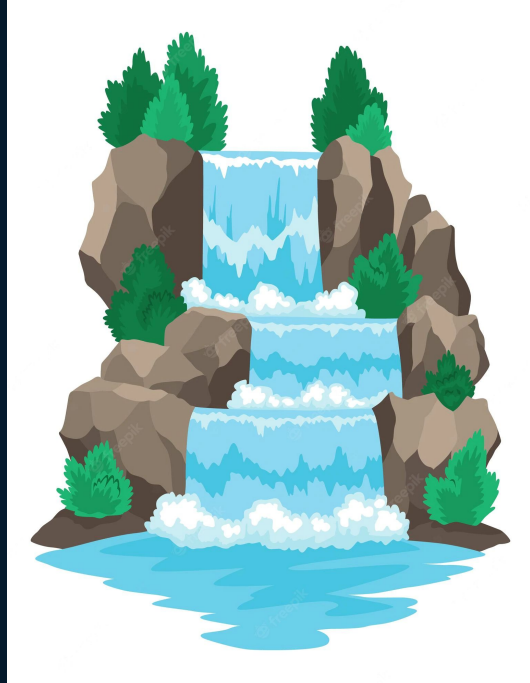
Se prueba el modelo en la aplicación deseada.

Recolección de datos

- Algoritmo de Viola Jones
- Modelo para detectar caras de vista frontal

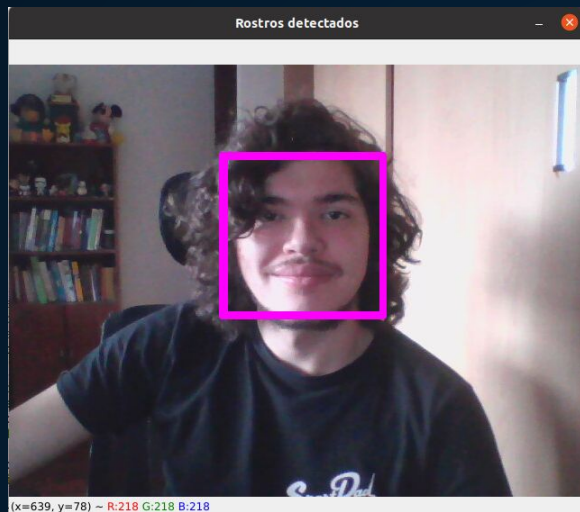
su77ungr Merge pull request #22727 from su77ungr:patch-1 <small>21761c</small> 20 days ago History	
..	
haarcascade_eye.xml	some attempts to tune the performance 9 years ago
haarcascade_eye_tree_eyeglasses.xml	some attempts to tune the performance 9 years ago
haarcascade_frontalcatface.xml	fix files permissions 3 years ago
haarcascade_frontalcatface_extended.xml	fix files permissions 3 years ago
haarcascade_frontalface_alt.xml	some attempts to tune the performance 9 years ago
haarcascade_frontalface_alt2.xml	some attempts to tune the performance 9 years ago
haarcascade_frontalface_alt_tree.xml	some attempts to tune the performance 9 years ago
haarcascade_frontalface_default.xml	some attempts to tune the performance 9 years ago
haarcascade_fullbody.xml	Some mist. typo fixes 5 years ago
haarcascade_lefteye_2splits.xml	some attempts to tune the performance 9 years ago
haarcascade_license_plate_rus_16stages.xml	Merge pull request #22727 from su77ungr:patch-1 20 days ago
haarcascade_lowerbody.xml	Some mist. typo fixes 5 years ago
haarcascade_profileface.xml	some attempts to tune the performance 9 years ago
haarcascade_righteye_2splits.xml	some attempts to tune the performance 9 years ago
haarcascade_russian_plate_number.xml	Create haarcascade_russian_plate_number.xml 9 years ago
haarcascade_smile.xml	fixing models to resolve XML violation issue 6 years ago
haarcascade_upperbody.xml	Some mist. typo fixes 5 years ago

Recolección de datos

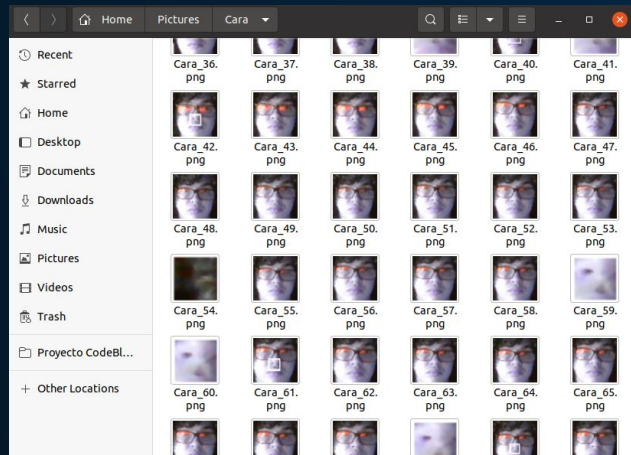


Clasificación en cascada

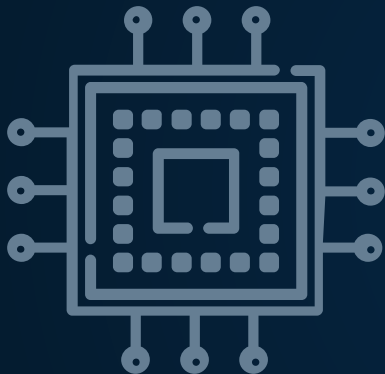
Los clasificadores en cascada basados en características de HAAR son un enfoque basado en el aprendizaje automático. En este se entrena una función en cascada utilizando una muestra que contiene muchas imágenes positivas y negativas. El resultado del clasificador es que los clasificadores fuertes se dividen en etapas para formar clasificadores en cascada. El término "cascada" significa que el clasificador así producido consta de un conjunto de clasificadores más simples que se aplican a la región de interés hasta que el objeto seleccionado se descarta o pasa.



Recolección de datos



- Se utiliza OpenCV para aplicar el clasificador de cascada a imágenes capturadas por la cámara en tiempo real.



CREACIÓN DEL MODELO

Se intenta 3 formas distintas:

1. PCA
2. TensorFlow
3. EdgImpulse

Análisis en Componentes Principales (PCA)



Método Estadístico



Simpleza



Modelos SVM



RESULTADOS PCA




TENSORFLOW

```
# build the model and train it
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(150, activation='relu')) # relu is used for performance
model.add(tf.keras.layers.Dense(50, activation='softmax')) # softmax
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
history = model.fit(inputs_train, outputs_train, epochs=100, batch_size=1, validation_data=(inputs_validate, outputs_validate))
model.summary()
```



EDGE IMPULSE

Image data




Input axes



Image

Image width

Image height


Resize mode



For optimal accuracy with transfer learning blocks, use a 96x96 or 160x160 image size.


Image




Name

Input axes (1)

☒ Image



Transfer Learning (Images)




Name

Input features


☒ Image

Output features

2 (Cara, Mascarilla)




Output features




2 (Cara, Mascarilla)

[Save Impulse](#)



Add a processing block



Add a learning block

EDGE IMPULSE

Neural Network settings

Training settings

Number of training cycles ⓘ

300

Learning rate ⓘ

0.0005

Validation set size ⓘ

20

%

Auto-balance dataset ⓘ


☐

Data augmentation ⓘ

☐

Neural network architecture

Input layer (2,500 features)


MobileNetV1 96x96 0.2 (no final dense layer, 0.1 dropout)

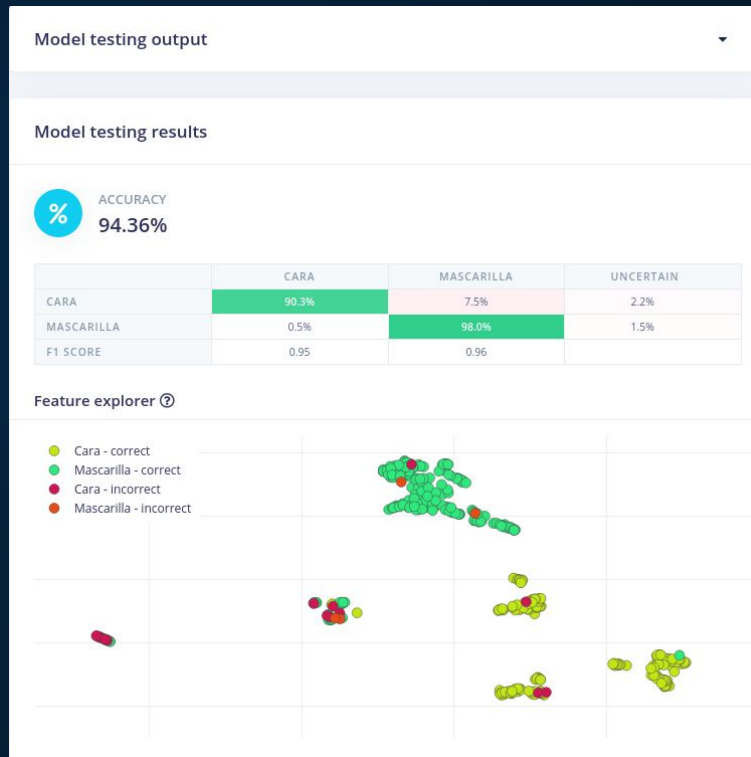
Choose a different model

Output layer (2 classes)



EDGE IMPULSE

Test data				
Set the 'expected outcome' for each sample to the desired outcome to automatically score the impulse.				
SAMPLE NAME	EXPECTED OUTCOME	LENGTH	ACCURACY	RESULT
Mascarilla_0	Mascarilla	-	100%	1 Mascarilla
Mascarilla_1	Mascarilla	-	100%	1 Mascarilla
Mascarilla_2	Mascarilla	-	100%	1 Mascarilla
Mascarilla_10	Mascarilla	-	100%	1 Mascarilla
Mascarilla_6	Mascarilla	-	100%	1 Mascarilla
Mascarilla_11	Mascarilla	-	100%	1 Mascarilla
Mascarilla_14	Mascarilla	-	100%	1 Mascarilla
Mascarilla_15	Mascarilla	-	100%	1 Mascarilla
Mascarilla_26	Mascarilla	-	100%	1 Mascarilla
Mascarilla_36	Mascarilla	-	100%	1 Mascarilla
Mascarilla_25	Mascarilla	-	100%	1 Mascarilla
Mascarilla_24	Mascarilla	-	100%	1 Mascarilla



RESULTADOS

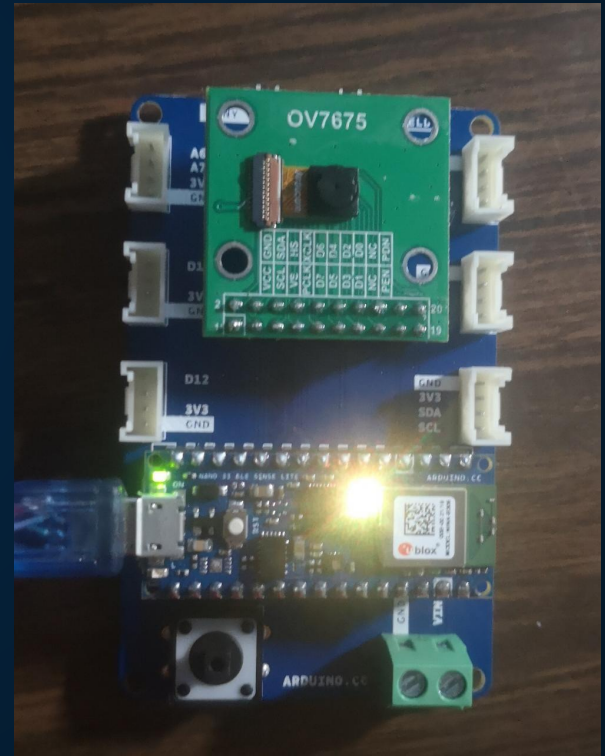
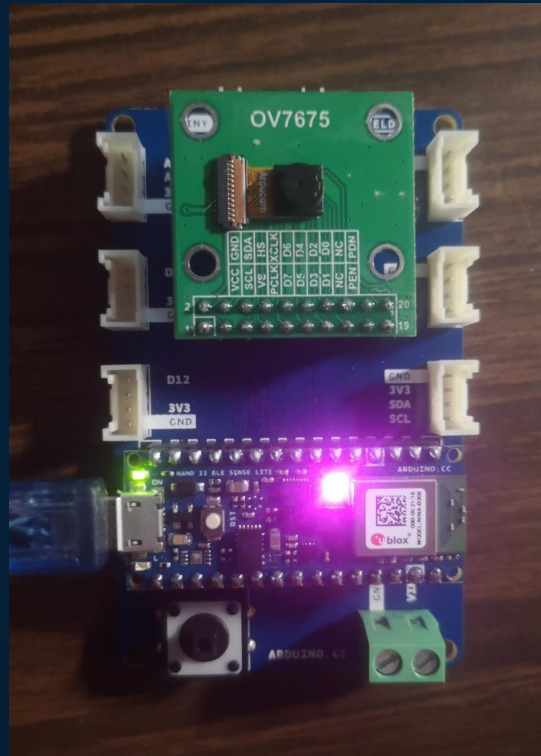
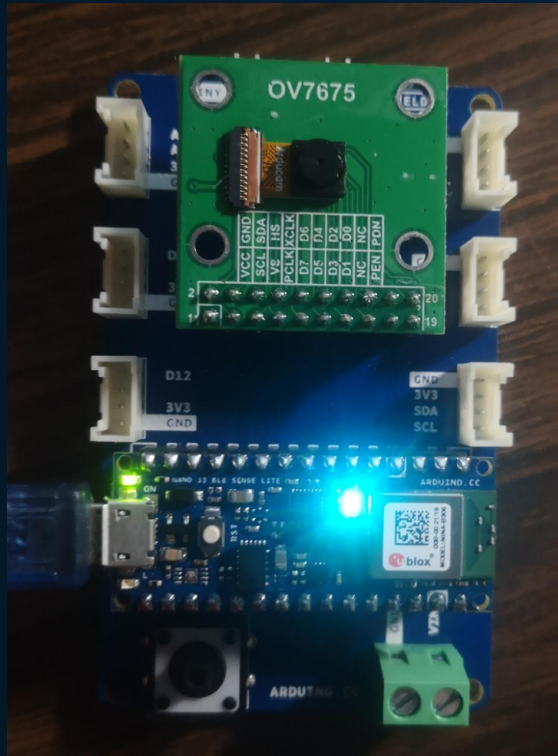
```
Starting inferencing in 2 seconds...
Taking photo...
Predictions (DSP: 3 ms., Classification: 175 ms., Anomaly: 0 ms.):
  Cara: 0.06250
  Mascarilla: 0.93750
```

```
Starting inferencing in 2 seconds...
Taking photo...
Predictions (DSP: 3 ms., Classification: 175 ms., Anomaly: 0 ms.):
  Cara: 0.00781
  Mascarilla: 0.99219
```

```
Starting inferencing in 2 seconds...
Taking photo...
Predictions (DSP: 3 ms., Classification: 175 ms., Anomaly: 0 ms.):
  Cara: 0.97656
  Mascarilla: 0.02344
```

```
Starting inferencing in 2 seconds...
Taking photo...
Predictions (DSP: 3 ms., Classification: 175 ms., Anomaly: 0 ms.):
  Cara: 0.99219
  Mascarilla: 0.00781
```


RESULTADOS



CONCLUSIONES

- A pesar de no cumplir con la totalidad de los objetivos propuestos, se logró desarrollar un programa basado en Machine Learning para el reconocimiento fácil básico y funcional aplicado a microcontroladores.
- Se logró profundizar en conceptos de Machine Learning enfocados en la detección de rostros como lo fueron el de PCA, análisis en cascadas y utilización y manejo de herramienta Edge Impulse.
- Existen múltiples formas de desarrollar una red neuronal, pero es una misma serie de pasos.

RECOMENDACIONES

- El Machine Learning no es un tema de aprendizaje sencillo y se debe realizar una investigación y aprendizaje previo antes de abordar temas y proyectos relacionados con este.
- Se recomienda elegir el modelo que más se ajuste a la aplicación que se está buscando ya que diferentes modelos pueden estar optimizados para ciertas aplicaciones específicas.
- Es vital un buen manejo del tiempo.