

UNIVERSIDAD MAYOR REAL Y PONTIFICIA DE SAN FRANCISCO DE CHUQUISACA
FACULTAD DE TECNOLOGÍA



MODELO DETECTOR DE OBJETOS PUNZOCORTANTES Y
ALARMA MEDIANTE ARDUINO

APELLIDOS: Salinas Vilar

NOMBRE: Maria Luciana

CARRERA: Ing. En ciencias de la computación

MATERIA: Inteligencia Artificial I

CU: 111-562

1. Planteamiento del Problema

En la actualidad, en entornos urbanos, en áreas públicas sea: mercados, metros, parques, calle. Se ha experimentado un creciente índice de violencia y accidentes y situaciones de alto riesgo a consecuencia de objetos punzocortantes, como cuchillos y navajas.

- Transporte público (micros, trufis, metros)
- Mercados y áreas de comercio
- Entidades educativas y espacios recreativos
- Calles y espacios peatonales.

Por lo que se puede apreciar que la sociedad carece de medidas de seguridad, métodos y sistemas que puedan detectar visualmente la presencia de un objeto punzocortantes y que se pueda activar una alarma o una notificación que alerte alguna situación de riesgo.

En otras partes, la mayoría de los sistemas de este estilo usan un hardware costoso con modelos pre-entrenados diseñados para funcionar de forma local sin internet.

2. Problema general

Por lo mencionado anteriormente se plantea la pregunta del problema:

¿Cómo detectar automáticamente la presencia de un objeto punzocortante mediante una cámara en tiempo real, y generar una alerta inmediata usando Arduino?

3. Justificación del proyecto

La detección temprana de un objeto punzocortante sea un cuchillo o navaja puede ayudar a:

- Prevenir incidentes violentos
- Reducir riesgos en escuelas, mercados o transporte
- Alertar a personal de seguridad
- Proteger a familias y niños
- Crear espacios más seguros con tecnología accesible

Ademas, como ventaja, el proyecto no tiene dependencia de internet para su funcionamiento, usa hardware de bajo costo (Arduino + webcam), y demuestra la viabilidad de sistemas de seguridad basados en IA ligera.

4. **Objetivo general**

Desarrollar un sistema de **detección en tiempo real de cuchillos** utilizando redes neuronales convolucionales (CNN), que envíe una señal a un Arduino para activar un buzzer y un LED de alerta cuando se identifique un objeto punzocortante.

5. **Objetivos específicos**

- Capturar y procesar imágenes en tiempo real desde una cámara.
- Entrenar una CNN capaz de distinguir entre *cuchillo* y *no cuchillo*.
- Construir un dataset propio con variedad de imágenes y fondos.
- Activar un buzzer y un LED cuando la probabilidad supere el 80%.

6. **Metodología general**

- Creacion de dataset manual
- Preprocesamiento de las imagenes
- Implementacion de EarlyStopping
- Activación de señales de alerta a través de Arduino.

7. Dataset utilizado

El dataset implementado fue creado manualmente, tomadno imagenes de cuchillos, de objetos que no son cuchillos.

A. Cuchillos (clase positiva)

795 imágenes que incluyen: cuchillos grandes de cocina

- variaciones de fondo
- diferentes distancias a la cámara
- diferentes ángulos

B. No cuchillos (clase negativa)

811 imágenes de:

- objetos similares (destornilladores, cucharas, herramientas)
- objetos comunes del hogar

Total de imágenes: 1606

8. Preprocesamiento

Todas las imágenes fueron:

- convertidas a escala de grises
- redimensionadas a **80 × 80 píxeles**
- normalizadas a valores entre 0 y 1
- Reshape a (80,80,1)

9. Arquitectura del Modelo

```
modelCNN = tf.keras.Sequential([  
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(80, 80, 1)),  
    tf.keras.layers.MaxPooling2D(2,2),  
  
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),  
    tf.keras.layers.MaxPooling2D(2,2),  
  
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),  
    tf.keras.layers.MaxPooling2D(2,2),  
  
    tf.keras.layers.Flatten(),  
    tf.keras.layers.Dense(32, activation='relu'),  
    tf.keras.layers.Dropout(0.5),  
    tf.keras.layers.Dense(1, activation='sigmoid')  
])
```

✓ 0.1s

Es un modelo simple casero, probado con diferentes valores, se determinó estos valores por el tamaño del dataset el cual es pequeño.

Tiene 3 bloques Convolucionales los que ayudan a extraer bordes y formas, Un Dropout medio alto (0.5) para controlar el overfitting, un batch size de 64 para poder tener mejor proceso y que sea más estable.

Y un tamaño de 80 x 80 para mejor manejo y generalización y también por las capacidades del software.

10. Entrenamiento

```
modelCNN.compile(optimizer='adam',
                 loss= 'binary_crossentropy',
                 metrics=['accuracy'])
✓ 0.0s

from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

early = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

# div del dataset
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.15, random_state=42, shuffle=True
)

history=modelCNN.fit(X_train,
                    y_train,
                    batch_size=64,
                    epochs=15,
                    validation_data=(X_val, y_val),
                    callbacks=[early])
✓ 11.0s
```

El modelo fue entrenado con el optimizador **Adam**, por su velocidad y facilidad de ajustar los gradientes.

como funcion de perdida se usó **binary_crossentropy**, como la mejor opción al ser este un problema de clasificación binarios, es decir, **cuchillo = 0** o **no cuchillo = 1**.

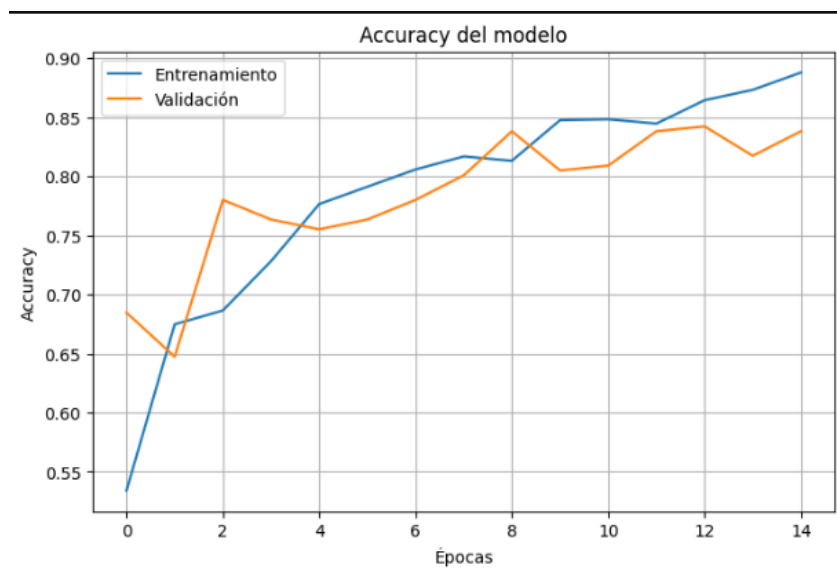
Para métricas se implemento **accuracy**, que ayuda a calcular los porcentajes probabilísticos de la predicción para el **entrenamiento y validación** del modelo.

Para prevenir que exista el **overfitting** se aplicó la técnica **Early Stopping**, que detiene el entrenamiento cuando el loss de la validación deja de mejorar.

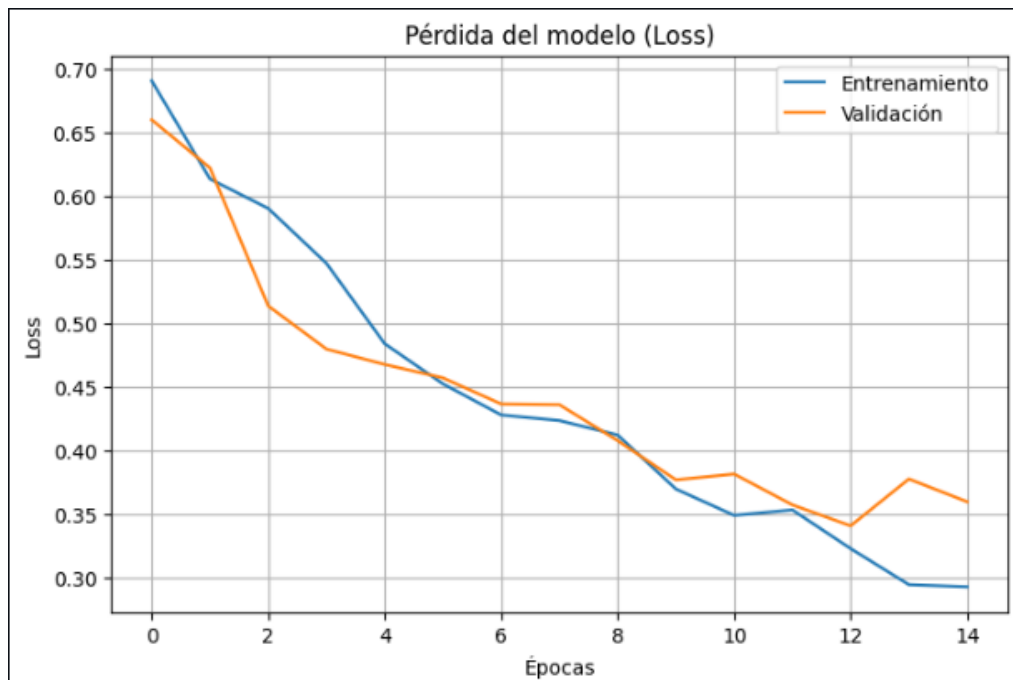
11. Muestra de resultados

```
Epoch 1/15
22/22 ————— 1s 55ms/step - accuracy: 0.9238 - loss: 0.1880 - val_accuracy: 0.8589 - val_loss: 0.4027
Epoch 2/15
22/22 ————— 1s 55ms/step - accuracy: 0.9267 - loss: 0.1680 - val_accuracy: 0.8672 - val_loss: 0.4228
Epoch 3/15
22/22 ————— 2s 74ms/step - accuracy: 0.9209 - loss: 0.1949 - val_accuracy: 0.8589 - val_loss: 0.4100
Epoch 4/15
22/22 ————— 3s 70ms/step - accuracy: 0.9267 - loss: 0.1750 - val_accuracy: 0.8631 - val_loss: 0.3991
Epoch 5/15
22/22 ————— 1s 65ms/step - accuracy: 0.9297 - loss: 0.1705 - val_accuracy: 0.8672 - val_loss: 0.4054
Epoch 6/15
22/22 ————— 1s 62ms/step - accuracy: 0.9260 - loss: 0.1601 - val_accuracy: 0.8714 - val_loss: 0.4295
Epoch 7/15
22/22 ————— 1s 58ms/step - accuracy: 0.9421 - loss: 0.1516 - val_accuracy: 0.8548 - val_loss: 0.4596
```

12. Graficos



En esta grafica se observa una evolución positiva en cuanto a los datos de entrenamiento y , a la vez, en los datos de validación. Las curvas se mantienen casi a la par, con una buena cercanía, sin distorsiones o graficas ruidosas. Esto indica que el modelo, verdaderamente aprende y no genera overfitting, es decir, que el modelo no memoriza la información que existe en el dataset, sino que está aprendiendo patrones que lo ayudaran a comprender los nuevos datos que lleguen.



Se observa que hay un comportamiento aceptable en cuanto a la pérdida de entrenamiento y validación, baja descendiendo de forma continua y sobretodo, muy a la par.

La pérdida de entrenamiento baja desde 70% hasta un 30% y la de validación de un 66% a un 36% aprox.

Puede que no llegue a algo mas bajo y una grafica mejor, por el tema del tamaño del dataset, pero no se ve que haya sobreajuste.

Importante:

Aprendizaje estable.

No se aprecia overfitting.

Se alcanza un 80-85% de precisión, por lo que se puede tomar que el modelo CNN casero, no está tan mal. Se lo puede mejorar y adaptar con mas herramientas.

Pero ya por si, se aprecia que el modelo puede ser capaz de distinguir.

13. Conclusiones

Es posible detectar objetos, aunque en mi caso, fallo a la hora de usar cámara, por las sombras, fondos colores, etc.

Las graficas muestran que el entrenamiento es estable, la precisión de entrenamiento y validación tiene un ritmo paralelo demostrar que no hay sobreajuste.

El proyecto cumple con todos los objetivos planteados: crear un dataset, entrenar una red neuronal, procesar imágenes en tiempo real y comunicar la detección al Arduino.

Aunque hay muchas fallas y limitaciones como el pequeño tamaño del dataset y la simplicidad de la arquitectura, lo que provoca las fallas a la hora de usar la camara y detectar con arduino, por lo ya mencionado, el tipo de imágenes, calidad de imágenes, fondo que sale, detalles, entre otros. Aquí se puede apreciar la importancia de tener un dataset limpio y bien creado.