ESCUELA MILITAR DE INGENIERÍA
“Mcal. Antonio Jose de Sucre”

UNIDAD ACADÉMICA LA PAZ

INTELIGENCIA ARTIFICIAL II

Tienda AI

INTEGRANTES:

- APOLACA MARINO DILAN
- SORIA QUISPE DANIEL JUSTO

DOCENTE: ING. JULIO COLQUE

PARALELO: 6TO “A”

CARRERA: INGENIERIA DE SISTEMAS

GESTION: I / 2024

Resumen del proyecto

Este proyecto se centra en el desarrollo de un entorno de compra automatizada que simula un entorno de tienda utilizando la tecnología de detección de objetos y billetes mediante un modelo de IA. El sistema captura imágenes en tiempo real desde una cámara y las procesa para identificar y contabilizar tanto productos como billetes, generando una lista de compras y un balance total de manera automática.

1. Introducción

La automatización en la experiencia de compra busca reducir la necesidad de interacción manual, ofreciendo un proceso más rápido y eficiente. Este proyecto aborda esta necesidad mediante el uso de modelos de detección de objetos basados en YOLO (You Only Look Once) para identificar productos y billetes en un entorno simulado de tienda.

2. Objetivos

- Desarrollar un sistema de detección y reconocimiento de objetos.

- Implementar la detección y clasificación de billetes para simular transacciones.

- Crear un entorno de simulación de compra que registre automáticamente los productos detectados y calcule el balance de pago.

3.Fundamento teórico

A continuación, se explicará los fundamentos necesarios para poder entender el proyecto.

3.1. Concepto de Modelo Predictivo

Modelos Predictivos son algoritmos que utilizan datos históricos para predecir futuros eventos o valores. Se basan en el reconocimiento de patrones y tendencias en los datos pasados para generar predicciones sobre datos nuevos. Los modelos predictivos son fundamentales en la inteligencia artificial, el aprendizaje automático y diversas aplicaciones prácticas que van desde la predicción del clima hasta la detección de fraudes.

3.2. Aprendizaje Supervisado

Aprendizaje Supervisado es una subcategoría del aprendizaje automático donde el modelo es entrenado utilizando un conjunto de datos etiquetados. Cada

instancia en el conjunto de entrenamiento incluye un par de entrada y salida deseada (etiqueta). El objetivo del modelo es aprender una función que mapee correctamente las entradas a las salidas.

Según (IBM, 2024) este es el proceso de aprendizaje supervisado

1. **Recopilación de Datos**:
    - Se reúne un conjunto de datos que contiene ejemplos representativos de los problemas a resolver.
    - Cada ejemplo incluye una entrada (características, XXX) y una salida conocida (etiqueta, YYY).
2. **División de Datos**:
    - Los datos se dividen en dos subconjuntos: datos de **entrenamiento** y datos de **validación/prueba**.
3. **Entrenamiento del Modelo**:
    - El modelo utiliza el subconjunto de datos de entrenamiento para aprender una función de mapeo f:X→Yf: X \rightarrow Yf:X→Y. Esto se realiza ajustando los parámetros del modelo para minimizar el error en la predicción de YYY a partir de XXX.
    - Ejemplo: En un modelo de regresión, el proceso de ajuste de parámetros minimiza la diferencia entre la salida predicha y la salida real (error).
4. **Evaluación y Ajuste**:
    - El modelo se evalúa utilizando el subconjunto de datos de validación para medir su precisión y generalización.
    - Se pueden realizar ajustes y mejoras basados en los resultados de esta evaluación.
5. **Predicción**:
    - El modelo entrenado se utiliza para predecir la salida Y′Y'Y′ para nuevas entradas X′X'X′ no vistas anteriormente.

3.2.1. Tipos de Aprendizaje Supervisado

Según (CEUPE, 2020)

- **Regresión**: Cuando la variable de salida es continua. Ejemplo: Predicción de precios de casas.
- **Clasificación**: Cuando la variable de salida es discreta o categórica. Ejemplo: Clasificación de correos electrónicos como spam o no spam.

3.3. Detección de Objetos con Modelos Predictivos

A continuación se explicara el algoritmo de aprendizaje YOLO.

*3.3.1. **YOLO***

YOLO es un algoritmo de aprendizaje profundo usado para la **detección de objetos** en imágenes. A diferencia de métodos anteriores que analizan las imágenes en múltiples etapas o regiones, YOLO trata toda la imagen de una sola vez (en un solo paso), prediciendo las ubicaciones y las clases de los objetos simultáneamente. (A., 2018)

3.3.2. **Arquitectura de YOLO**:

1. **Entrada**:
    - La imagen de entrada se divide en una cuadrícula de S×SS \times SS×S.
2. **Predicción**:
    - Para cada celda de la cuadrícula, el modelo predice:
        - **Cajas delimitadoras**: Coordenadas de BBB cajas posibles.
        - **Confianza**: Confianza de que la caja contiene un objeto y la precisión de la caja.
        - **Clases**: Probabilidades de las clases de objetos para cada caja.
3. **Salida**:
    - Genera un conjunto de cajas delimitadoras con puntuaciones de confianza y probabilidades de clase para los objetos detectados.

Entrenamiento de YOLO:

1. **Conjunto de Datos Etiquetados**:
    - Las imágenes están etiquetadas con las coordenadas de las cajas delimitadoras y las clases de los objetos.
2. **Pérdida**:
    - La función de pérdida de YOLO incluye componentes para la precisión de las coordenadas, la confianza de detección y la clasificación de objetos.
3. **Optimización**:
    - Los parámetros del modelo se optimizan usando algoritmos como el descenso de gradiente para minimizar la pérdida y mejorar la precisión de detección

4. **Fundamento práctico**

4.1. Librerías Utilizadas

- cv2: Utilizada para la captura y procesamiento de video.
- ultralytics: Utilizada para la implementación del modelo YOLO para la detección de objetos.

4.2. Modelo de IA

El sistema utiliza dos modelos YOLO entrenados:

1. **Modelo de Detección de Objetos (ObjectModel)**: Capaz de identificar una amplia gama de objetos comunes.
2. **Modelo de Detección de Billetes (billModel)**: Especializado en la identificación de billetes para la simulación de pagos.

Configuración inicial

```
class ShopIA:
    # Init
    def int(self):
        # VideoCapture
        self.cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)
        self.cap.set(3, 1280)
        self.cap.set(4, 720)

        # MODELS:
        # Object model
        ObjectModel = YOLO('Modelos/yolov8l.onnx')
        self.ObjectModel = ObjectModel

        billModel = YOLO('Modelos/billBank2.onnx')
        self.billModel = billModel

        # CLASES:
        # Objects
        #clsObject = ObjectModel.names
        clsObject =
['person','bicycle','car','motorcycle','airplane','bus','train','truck','boat','traffic light',
                    'fire hydrant','stop sign','parking
meter','bench','bird','cat','dog','horse','sheep','cow','elephant',
                    'bear','zebra','giraffe','backpack','umbrella','handbag','tie','suitcase','fris
bee','skis','snowboard',
```

```
                'sports ball','kite','baseball bat','baseball
glove','skateboard','surfboard','tennis racket','bottle',
                'wine
glass','cup','fork','knife','spoon','bowl','banana','apple','sandwich','orange','broccoli','c
arrot',
                'hot dog','pizza','donut','cake','chair','couch','potted plant','bed','dining
table','toilet','tv','laptop',
                'mouse','remote','keyboard','cell
phone','microwave','oven','toaster','sink','refrigerator','book','clock','vase',
                'scissors','teddy bear','hair drier','toothbrush']
        self.clsObject = clsObject

        # Bills Bank
        clsBillBank = ['Billete10', 'Billete20', 'Billete50']
        self.clsBillBank = clsBillBank

        # Total balance
        total_balance = 0
        self.total_balance = total_balance
        self.pay = ''

        return self.cap
```

- **Inicialización de la captura de video:** Configura la cámara para capturar imágenes en resolución 1280x720.
- **Carga de modelos de IA:** Se cargan dos modelos de YOLO entrenados para la detección de objetos y billetes respectivamente.
- **Definición de clases:** Listas que contienen las categorías de objetos y tipos de billetes que el sistema puede reconocer.

4.3. Funciones de Dibujo

Estas funciones se utilizan para añadir anotaciones visuales en las imágenes procesadas.

```
# DRAW FUNCTIONS
    # Area
    def draw_area(self, img, color, xi, yi, xf, yf):
        img = cv2.rectangle(img, (xi, yi), (xf, yf), color, 1, 1)
        return img

    # Text
```

```
    def draw_text(self, img, color, text, xi, yi, size, thickness, back = False):
        sizetext = cv2.getTextSize(text, cv2.FONT_HERSHEY_DUPLEX, size,
thickness)
        dim = sizetext[0]
        baseline = sizetext[1]
        if back == True:
            img = cv2.rectangle(img, (xi, yi - dim[1] - baseline), (xi + dim[0], yi +
baseline - 7),(0, 0, 0), cv2.FILLED)
        img = cv2.putText(img, text, (xi, yi - 5), cv2.FONT_HERSHEY_DUPLEX, size,
color, thickness)
        return img

    # Line
    def draw_line(self, img, color, xi, yi, xf, yf):
        img = cv2.line(img, (xi, yi), (xf, yf), color, 1, 1)
        return img
```

draw_**area**: Dibuja un rectángulo para demarcar áreas de interés.
draw_text: Añade texto en la imagen.
draw_line: Dibuja líneas.

```
def marketplace_list(self, frame, object):
        list_products = {'handbag':30000, 'sports ball':10000, 'bottle':50000,
'cup':30000, 'fork':5000, 'knife':5000, 'spoon':5000,
                         'banana':1000, 'apple':1000, 'orange':1000, 'broccoli':500,
'carrot':1000, 'mouse':60000, 'keyboard':100000,
                         'book':40000, 'clock':50000, 'scissors':15000, 'toothbrush':8000}

        # Text Config
        list_area_xi, list_area_yi, list_area_xf, list_area_yf = self.area(frame, 0.7739,
0.6250, 0.9649, 0.9444)
        size_obj, thickness_obj = 0.60, 1
# Add shopping list with price
        # Bolso
        if object == 'handbag' not in [item[0] for item in self.shopping_list]:
            price = list_products['handbag']
            self.shopping_list.append([object, price])
            # Show
            text = f'{object} --> ${price}'
            frame = self.draw_text(frame, (0, 255, 0), text, list_area_xi + 10,
                                   list_area_yi + (40 + (self.posicion_products * 20)),
                                   size_obj, thickness_obj, back=False)
            self.posicion_products += 1
           return frame
```

Base de datos de productos: list_products contiene los precios de los productos.
Actualización de la lista de compras: Añade el producto detectado a la lista de compras si no está ya presente.

Proceso de Balance y Pago

```
    # Balance process
    def balance_process(self, bill_type):
        if bill_type == 'Billete10':
            self.balance = 10000
        elif bill_type == 'Billete20':
            self.balance = 20000
        elif bill_type == 'Billete50':
            self.balance = 50000
# Payment process
    def payment_process(self, accumulative_price, accumulative_balance):
        payment = accumulative_balance - accumulative_price
        print(payment)
        if payment < 0:
            text = f'Falta cancelar {abs(payment)}$'

        elif payment > 0:
            text = f'Su cambio es de: {abs(payment)}$'
            self.accumulative_price = 0
            self.total_balance = 0

        elif payment == 0:
            text = f'Gracias por su compra!'
            self.accumulative_price = 0
            self.total_balance = 0

        return text
```

- **Procesamiento de billetes:** Ajusta el balance total basado en el tipo de billete detectado.
- **Proceso de pago:** Calcula el monto a pagar o el cambio y resetea los valores para una nueva compra.

4.4. Predicción e Inferencia

La función prediction_model realiza la detección en el flujo de video y actualiza la imagen con anotaciones.

```
# INFERENCE
    def prediction_model(self, clean_frame, frame, model, clase):
        bbox = []
        cls = 0
        conf = 0
        # Yolo | AntiSpoof
        results = model(clean_frame, stream=True, verbose=False)
        for res in results:
            # Box
            boxes = res.boxes
            for box in boxes:
                # Bounding box
                x1, y1, x2, y2 = box.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)

                # Error < 0
                if x1 < 0: x1 = 0
                if y1 < 0: y1 = 0
                if x2 < 0: x2 = 0
                if y2 < 0: y2 = 0

                bbox = [x1,y1,x2,y2]

                # Class
                cls = int(box.cls[0])

                # Confidence
                conf = math.ceil(box.conf[0])

                if clase == 0:
                    # Draw
                    objeto = self.clsObject[cls]
                    text_obj = f'{self.clsObject[cls]} {int(conf * 100)}%'
                    # Marketplace list
                    frame = self.marketplace_list(frame, objeto)

                    # Draw
                    size_obj, thickness_obj = 0.75, 1
                    frame = self.draw_text(frame, (0, 255, 0), text_obj, x1, y1, size_obj,
thickness_obj, back=True)
                    frame = self.draw_area(frame, (0, 255, 0), x1, y1, x2, y2)

                if clase == 1:
                    # Draw
```

```
                bill_type = self.clsBillBank[cls]
                text_obj = f'{self.clsBillBank[cls]} {int(conf * 100)}%'
                self.balance_process(bill_type)

                # Draw
                size_obj, thickness_obj = 0.75, 1
                frame = self.draw_text(frame, (0, 255, 0), text_obj, x1, y1, size_obj,
thickness_obj, back=True)
                frame = self.draw_area(frame, (0, 255, 0), x1, y1, x2, y2)

                #break
        return frame
```

- **Resultados de detección:** Obtiene los resultados de detección del modelo.
- **Actualización visual:** Anota el objeto detectado o billete en la imagen y actualiza la lista de compras o el balance.

4.5. Función Principal

La función tiendaIA ejecuta el flujo principal del sistema, capturando frames y realizando la detección de objetos y billetes.

```
# Main
    def tiendaIA(self, cap):
        while True:
            # Frames
            ret, frame = cap.read()
            # Read keyboard
            t = cv2.waitKey(5)

            # Frame Object Detect
            clean_frame = frame.copy()

            # Info Marketplace list
            shopping_list = []
            self.shopping_list = shopping_list
            posicion_products = 1
            self.posicion_products = posicion_products
            accumulative_price = 0
            self.accumulative_price = accumulative_price
            # Info Payment process
            balance = 0
            self.balance = balance
```

```
            # Areas
            # Shopping area
            shop_area_xi, shop_area_yi, shop_area_xf, shop_area_yf =
self.area(frame, 0.0351, 0.0486, 0.7539, 0.9444)
            # Draw
            color = (0,255,0)
            text_shop = f'Shopping area'
            size_shop, thickness_shop = 0.75, 1
            frame = self.draw_area(frame, color, shop_area_xi, shop_area_yi,
shop_area_xf, shop_area_yf)
            frame = self.draw_text(frame, color, text_shop, shop_area_xi,
shop_area_yf + 30, size_shop, thickness_shop)
# Payment area
            pay_area_xi, pay_area_yi, pay_area_xf, pay_area_yf =
self.area(frame, 0.7739, 0.0486, 0.9649, 0.6050)
            # Draw
            text_pay = f'Payment area'
            size_pay, thickness_pay = 0.50, 1
            frame = self.draw_line(frame, color, pay_area_xi, pay_area_yi,
pay_area_xi, int((pay_area_yi+pay_area_yf)/2))
            frame = self.draw_line(frame, color, pay_area_xi, pay_area_yi,
int((pay_area_xi+pay_area_xf)/2), pay_area_yi)
            frame = self.draw_line(frame, color, pay_area_xf,
int((pay_area_yi+pay_area_yf)/2), pay_area_xf, pay_area_yf)
            frame = self.draw_line(frame, color,
int((pay_area_xi+pay_area_xf)/2), pay_area_yf, pay_area_xf, pay_area_yf)
            frame = self.draw_text(frame, color, text_pay, pay_area_xf-100,
shop_area_yi+10, size_pay, thickness_pay)

            # List area
            list_area_xi, list_area_yi, list_area_xf, list_area_yf =
self.area(frame, 0.7739, 0.6250, 0.9649, 0.9444)
            # Draw
            text_list = f'Shopping List'
            size_list, thickness_list = 0.65, 1
            frame = self.draw_line(frame, color, list_area_xi, list_area_yi,
list_area_xi, list_area_yf)
            frame = self.draw_line(frame, color, list_area_xi, list_area_yi,
list_area_xf, list_area_yi)
            frame = self.draw_line(frame, color, list_area_xi+30,
list_area_yi+30, list_area_xf-30, list_area_yi+30)
            frame = self.draw_text(frame, color, text_list, list_area_xi+55,
list_area_yi+30, size_list, thickness_list)
```

```
            # Predict Object
            frame = self.prediction_model(clean_frame, frame,
self.ObjectModel, clase=0)
            # Predict Bills Bank
            frame = self.prediction_model(clean_frame, frame,
self.billModel, clase=1)

            # Accumulative Price Show
            text_price = f'Compra total: {self.accumulative_price} $'
            frame = self.draw_text(frame, (0, 255, 0), text_price,
list_area_xi + 10, list_area_yf, 0.60, 1, back=False)
            # Total Balance Show
            text_balance = f'Saldo total: {self.total_balance} $'
            frame = self.draw_text(frame, (0, 255, 0), text_balance,
list_area_xi + 10, list_area_yf + 30, 0.60, 1, back=False)
            # Payment
            frame = self.draw_text(frame, (0, 255, 0), self.pay,
list_area_xi + - 300, list_area_yf + 30, 0.60, 1, back=False)


            # Show
            cv2.imshow("Tienda IA", frame)

            # Balance
            if t == 83 or t == 115:
                self.total_balance = self.total_balance + self.balance
                self.balance = 0
            # Payment
            if t == 80 or t == 112:
                self.pay = self.payment_process(self.accumulative_price,
self.total_balance)
            # Exit
            if t == 27:
                break

        # Release
        self.cap.release()
        cv2.destroyAllWindows()
```

5. Conclusiones

1. **Cumplimiento de Objetivos de Detección**:
    - El modelo predictivo implementado cumple eficazmente con los objetivos planteados de identificar productos y billetes en imágenes en tiempo real. Utilizando la técnica YOLO, se ha logrado una

detección precisa de múltiples clases de objetos, proporcionando una base sólida para aplicaciones en el comercio y la banca.

2. **Desafíos de Latencia**:
    - A pesar de la precisión en la detección, se ha observado una latencia significativa en la captura y procesamiento de imágenes en tiempo real. Esta lentitud puede ser atribuida a limitaciones en el procesamiento del hardware, la complejidad del modelo, o la eficiencia del código. La latencia afecta la experiencia del usuario y puede limitar la aplicabilidad del sistema en entornos que requieren respuestas inmediatas.
3. **Optimización del Rendimiento**:
    - Para mejorar la velocidad de respuesta del sistema, se sugiere la optimización de la implementación del modelo. Esto incluye la reducción de la complejidad del modelo sin comprometer significativamente la precisión, la mejora de la eficiencia del código, y la utilización de hardware más avanzado. Técnicas como la compresión de modelos, la cuantización, y el uso de frameworks optimizados pueden ayudar a reducir la latencia.
4. **Adaptación del Modelo a las Condiciones del Entorno**:
    - La capacidad del modelo para adaptarse a diferentes condiciones de iluminación y perspectivas de la cámara es crucial para su éxito en aplicaciones prácticas. La recolección y el etiquetado de datos adicionales que representen una variedad de escenarios pueden ayudar a entrenar un modelo más robusto y reducir los errores de detección en entornos variados.
5. **Implementación Práctica y Mejora Continua**:
    - A pesar de las limitaciones actuales, la implementación del sistema proporciona una base sólida para futuras mejoras. Es fundamental mantener un ciclo continuo de evaluación y ajuste, basado en los comentarios de los usuarios y pruebas en condiciones reales, para mejorar la eficiencia y la usabilidad del sistema.
6. **Perspectivas Futuras para la Detección en Tiempo Real**:
    - La integración de algoritmos de detección de objetos con técnicas de optimización en hardware especializado (como GPU o TPU) y la adopción de métodos de aprendizaje más avanzados, como el aprendizaje semi-supervisado o auto-supervisado, podrían ofrecer mejoras significativas en el rendimiento y la capacidad de detección en tiempo real.

6. Bibliografía

A., E. (12 de mayo de 2018). *Detección de objetos con YOLO: implementaciones y como usarlas*. Obtenido de https://medium.com/@enriqueav/detecci%C3%B3n-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246.

CEUPE. (2020). *Tipos de aprendizaje supervisado*. Obtenido de https://www.ceupe.com/blog/aprendizaje-supervisado.html.

IBM. (2024). *Procesos de aprendizaje supervisado*. Obtenido de https://www.ibm.com/es-es/topics/supervised-learning.