# **Plan de Proyecto: Detector de Billetes Argentinos con IA**

Este documento describe una posible hoja de ruta para desarrollar una aplicación web y móvil capaz de identificar billetes argentinos y realizar operaciones matemáticas con sus valores.

## **Visión General de la Arquitectura**

La arquitectura que propones es muy acertada:

1. **Frontend (React + TSX):** Es la interfaz con la que el usuario interactúa. Se encargará de acceder a la cámara, mostrar el video en tiempo real, capturar imágenes y comunicarse con el backend.
2. **Backend (Python):** Será el cerebro de la operación. Recibirá una imagen desde el frontend, la procesará usando el modelo de IA y devolverá el resultado de la identificación.
3. **Modelo de IA (CNN):** Es el componente especializado que aprenderá a diferenciar los billetes. Se "alojará" en el backend.

## **Fase 1: El Corazón del Proyecto - El Modelo de IA (CNN)**

Esta es la fase más crítica y la que definirá el éxito de la aplicación. **Te recomiendo empezar por aquí.** Si no tienes un modelo que funcione, el resto de la aplicación no tendrá propósito.

#### **1.1. Recopilación Masiva de Datos (El paso más importante)**

La red neuronal necesita "ver" muchísimos ejemplos para aprender.

* **Cantidad:** Necesitarás **cientos o miles de imágenes por cada billete** que quieras identificar (10, 20, 50, 100, 200, 500, 1000, 2000 pesos).
* **Variedad:** Las fotos deben ser muy variadas para que el modelo pueda generalizar y no falle en el mundo real. Incluye:
  + Billetes nuevos y billetes gastados/arrugados.
  + Diferentes condiciones de luz (luz de día, luz artificial, sombras).
  + Diferentes ángulos y perspectivas.
  + Fotos con el billete parcialmente tapado (por un dedo, por ejemplo).
  + Diferentes fondos.
* **Organización:** Crea una estructura de carpetas clara. Por ejemplo:  
  /dataset/  
  ├── 10\_pesos/  
  │ ├── img\_0001.jpg  
  │ ├── img\_0002.jpg  
  │ └── ...  
  ├── 20\_pesos/  
  │ └── ...  
  ├── 500\_pesos/  
  │ └── ...  
  └── (etc...)

#### **1.2. Elección de Herramientas de IA**

* **Frameworks:** Te recomiendo usar **TensorFlow con Keras** o **PyTorch**. Keras es conocido por ser más amigable para principiantes.
* **Entorno de Entrenamiento:** No necesitas una supercomputadora al principio. Puedes usar **Google Colab**, que te da acceso gratuito a GPUs para acelerar el entrenamiento.

#### **1.3. Entrenamiento del Modelo (CNN)**

1. **Pre-procesamiento:** Ajustarás el tamaño de todas las imágenes a uno estándar (ej. 224x224 píxeles) y las normalizarás.
2. **Arquitectura de la Red:** Definirás las capas de tu Red Neuronal Convolucional. Puedes empezar con una arquitectura conocida como MobileNetV2 o EfficientNet, que son potentes y están optimizadas para ser rápidas, ideal para un futuro en móviles.
3. **Entrenamiento:** Alimentarás a la red con tu set de datos. El modelo ajustará sus "pesos" internos hasta que pueda predecir con alta precisión a qué carpeta (clase) pertenece cada imagen.
4. **Validación y Pruebas:** Usarás un subconjunto de imágenes que el modelo nunca ha visto para evaluar su rendimiento real.

#### **1.4. Exportación del Modelo**

Una vez que estés satisfecho con la precisión del modelo, lo guardarás en un archivo (ej. modelo\_billetes.h5 o formato SavedModel de TensorFlow). Este archivo es el que usarás en tu backend.

## **Fase 2: El Backend con Python**

Su única misión será exponer el modelo de IA a través de una API.

* **Frameworks:** **FastAPI** o **Flask** son perfectos para esto. FastAPI es más moderno y extremadamente rápido, ideal para este caso de uso.
* **Endpoint de la API:** Necesitarás crear un "endpoint" (una URL) que acepte peticiones. Por ejemplo: POST /api/identificar.
  + **Entrada (Request):** Recibirá un archivo de imagen.
  + **Proceso:**
    1. Cargará el modelo de IA (el archivo de la Fase 1).
    2. Tomará la imagen recibida y la pre-procesará de la misma forma que en el entrenamiento.
    3. Le pasará la imagen al modelo para que haga una predicción.
  + **Salida (Response):** Devolverá un resultado en formato JSON. Por ejemplo:  
    {  
     "denominacion": "500 Pesos",  
     "valor": 500,  
     "precision": 0.985  
    }

## **Fase 3: El Frontend con React y TypeScript**

Aquí construirás la interfaz de usuario.

* **Acceso a la Cámara:** Usa una librería como react-webcam para acceder fácilmente a la cámara del dispositivo y mostrar el video en la pantalla.
* **Captura de Imagen:** Implementa un botón de "Analizar" que tome un fotograma del video de la cámara.
* **Comunicación con el Backend:** Al pulsar el botón, la imagen capturada se envía al endpoint /api/identificar del backend (usando fetch o axios).
* **Visualización de Resultados:** Cuando el backend responde, muestras la denominación y el valor en pantalla.
* **Lógica de Cálculo:**
  + Usa el estado de React (useState) para mantener un total.
  + Crea botones de "Sumar" y "Restar" que, al ser presionados, tomen el valor recién identificado y lo sumen o resten al total.

## **Fase 4: Integración y Despliegue Web (Versión 1)**

1. **Conexión:** Asegúrate de que tu app de React pueda comunicarse correctamente con tu API de Python (atención a los problemas de CORS que puedan surgir).
2. **Despliegue:**
   * **Frontend:** Puedes desplegar tu app de React de forma gratuita y muy sencilla en servicios como **Vercel** o **Netlify**.
   * **Backend:** Puedes "dockerizar" tu aplicación de Python y desplegarla en servicios como **Heroku**, **Google Cloud Run** o **AWS Elastic Beanstalk**.

## **Fase 5: Futuro - Migración a Apps Nativas**

Esto es un gran paso y usualmente implica reescribir una parte de la aplicación.

* **Opción 1: React Native:** Te permite reutilizar gran parte de tu lógica de React y TypeScript. Tendrás que cambiar la parte de la cámara por una librería nativa (react-native-vision-camera) pero la estructura general se mantiene.
* **Opción 2: Nativo Puro (Swift para iOS, Kotlin para Android):** Ofrece el mejor rendimiento posible.
* **El Gran Cambio - IA en el dispositivo:** Para una app móvil, no es eficiente enviar cada imagen a un servidor. Lo ideal es que el modelo de IA se ejecute **directamente en el teléfono**. Para esto, necesitarás convertir tu modelo de TensorFlow a un formato optimizado:
  + **TensorFlow Lite (.tflite):** Funciona tanto en Android como en iOS.
  + **CoreML:** Específico de Apple, máximo rendimiento en iOS.

## **Preguntas Clave para Ti**

Para ayudarte a refinar la idea, te dejo estas preguntas:

1. **Sobre los Datos:** ¿Cómo y dónde planeas conseguir los miles de fotos de billetes? ¿Tomarás las fotos tú mismo? ¿Buscarás datasets existentes? (Este es el mayor cuello de botella).
2. **Sobre la Experiencia de Usuario (UX):**
   * ¿La detección debe ser en tiempo real (apuntas y te dice qué es instantáneamente) o el usuario debe presionar un botón para analizar? El tiempo real es mucho más complejo.
   * ¿Cómo manejará la app los errores de identificación? (ej. si identifica mal un billete o si apuntas a algo que no es un billete).
   * ¿Cómo se visualizará la suma? ¿Simplemente un número grande en la pantalla?
3. **Sobre el Rendimiento:** ¿Qué tan rápido debe ser el reconocimiento? ¿Menos de 1 segundo? Esto influirá en la arquitectura del modelo y del backend.
4. **Funcionalidad Offline:** ¿La aplicación debe funcionar sin conexión a internet? Si la respuesta es sí, la **Fase 5 (IA en el dispositivo)** se vuelve obligatoria desde el principio.

Espero que este plan detallado te sea de gran ayuda. ¡Es un proyecto increíblemente formativo! No dudes en preguntar si quieres profundizar en alguna de las fases.