

# **DETECCIÓN DE MANOS**

# **USANDO CNN**

**GONZALO BERLATI**



# TABLA DE CONTENIDOS

01

OBJETIVO

02

INVESTIGACIÓN Y  
DEFINICIÓN MODELO

03

PRIMER VERSIÓN  
DEL MODELO

04

SEGUNDA VERSIÓN  
DEL MODELO

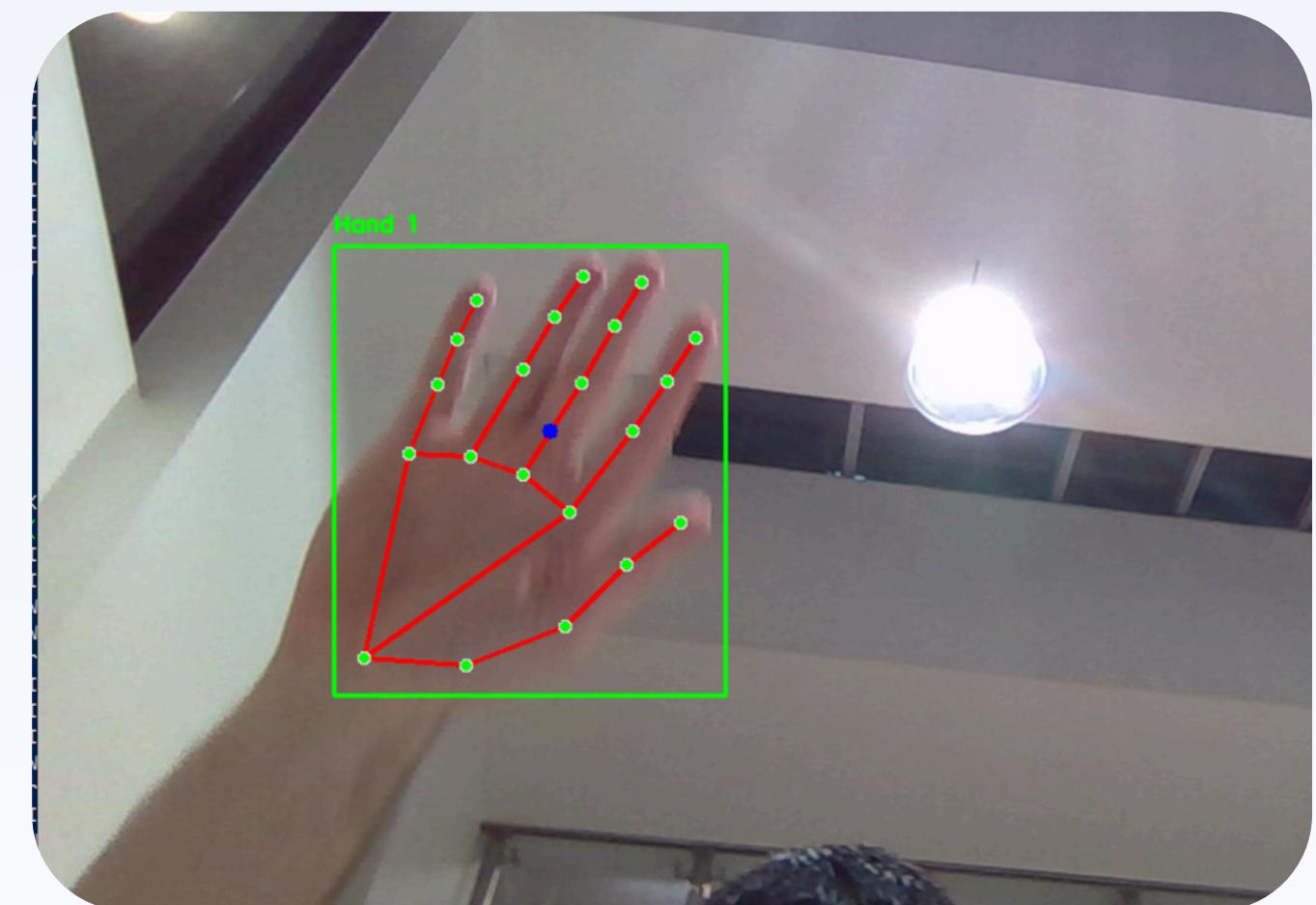
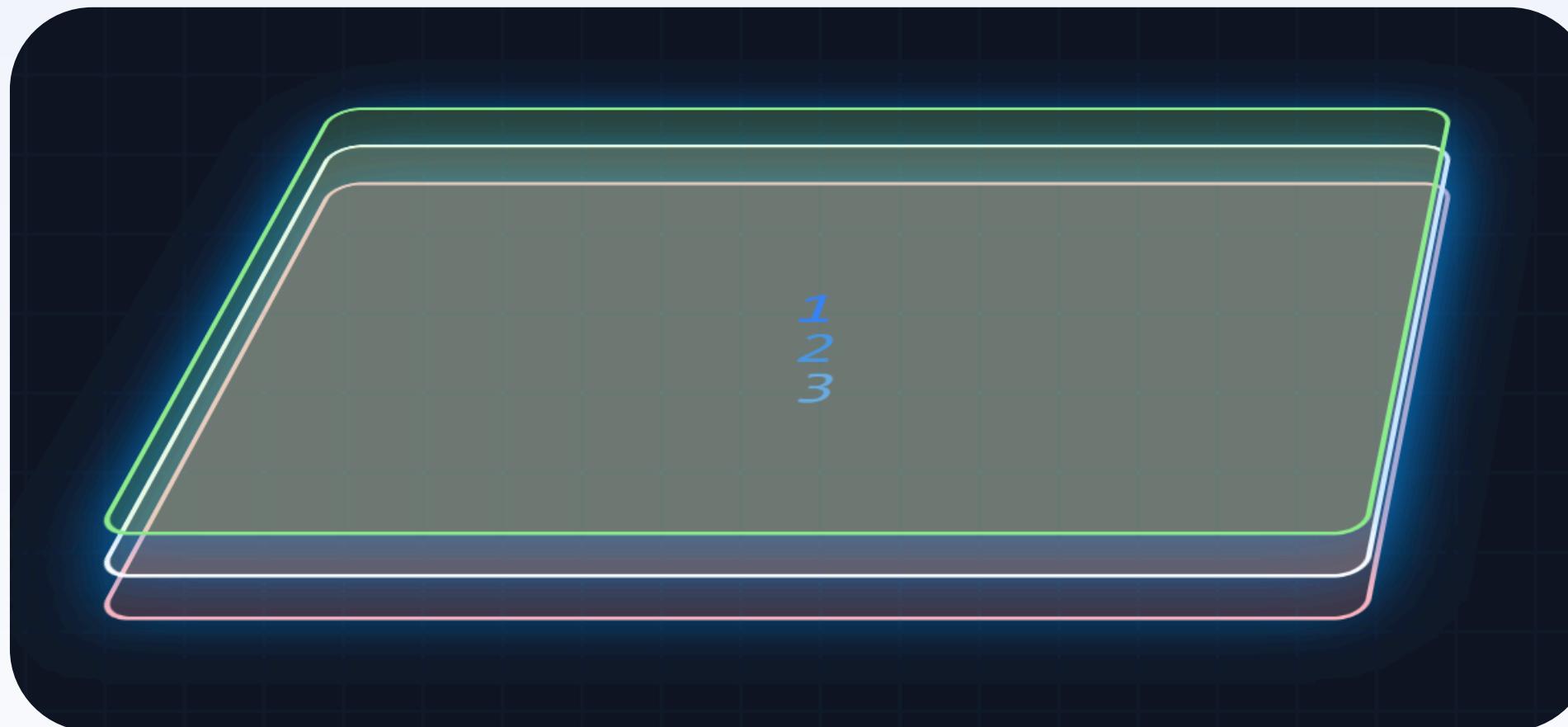
05

CONCLUSIONES

# OBJETIVO

El objetivo del proyecto es:

- Poder identificar manos en tiempo real.
- Si hay dos manos, detectar la distancia.
- En base a la distancia, poder interactuar con un gráfico a tiempo real.



# ANÁLISIS DEL MODELO

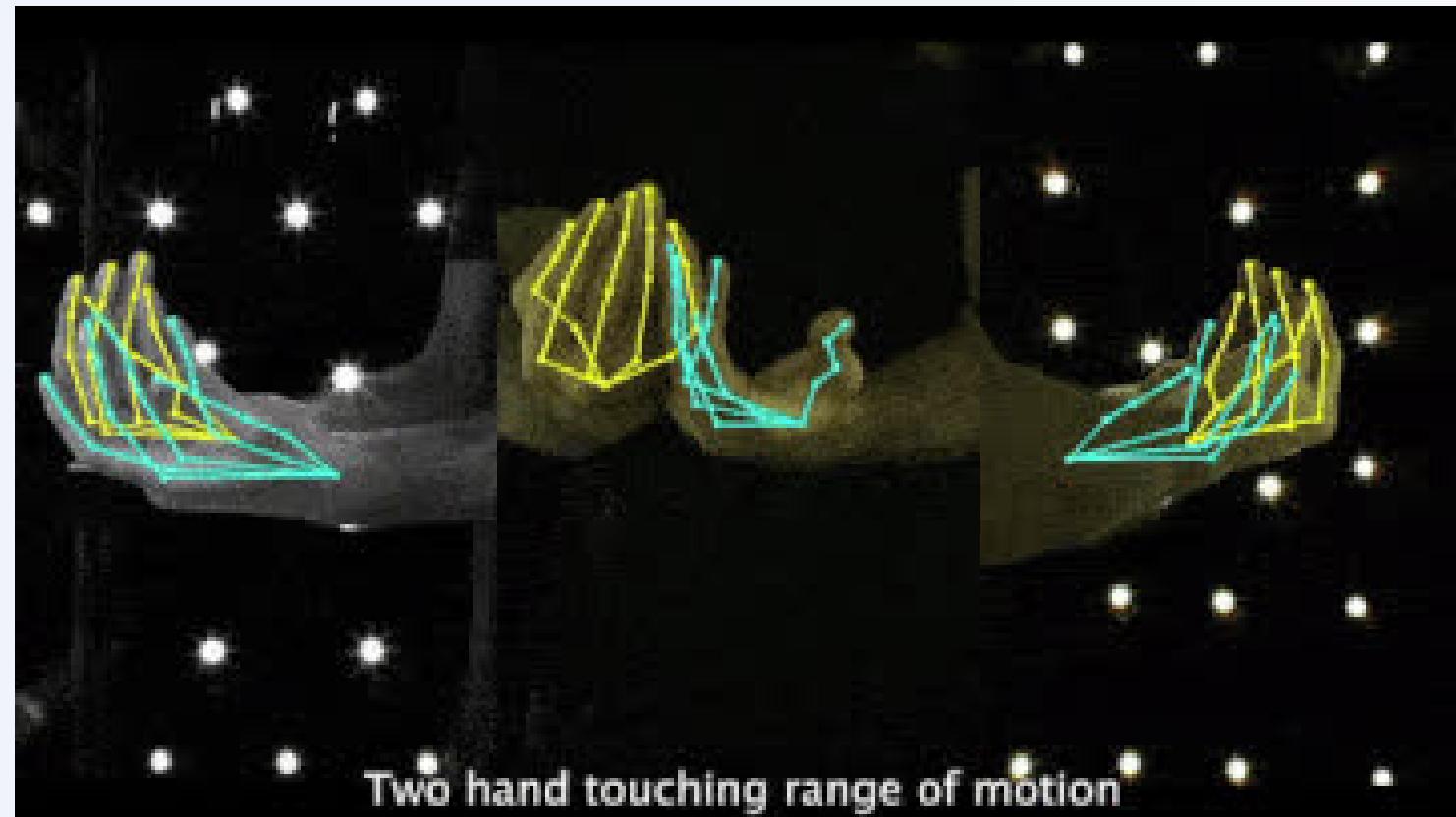
Para lograr esto, es necesario construir una CNN para identificar la palma de las manos.

Algunos puntos a definir son:

- ¿Que datasets a utilizar?
- ¿Qué arquitectura de CNN utilizar?
- ¿Cómo vamos a procesar los datos?

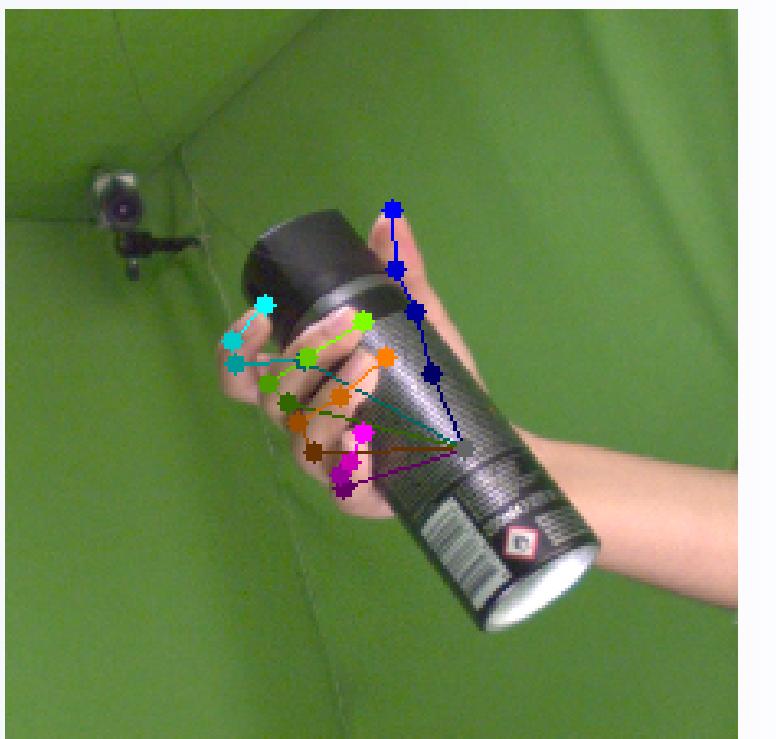


# DATASETS DISPONIBLES



InterHand 2.6m

Un conjunto de 2.6 millones de imágenes con anotaciones de posiciones de manos. Utilizado para predicción de posición de manos.



FreiHAND

Dataset con +130k imágenes con anotaciones para estimación de posición de manos.



EgoHands

48 videos grabados con Google Glass, con anotaciones por cada frame de posición de las manos.

# DATASET UTILIZADOS

El principal dataset utilizado fue **EgoHands**, debido a:

- Sus anotaciones sencillas
- Buena calidad de frames para entrenar el modelo.
- Distingue entre sus anotaciones entre mano izquierda y derecha.
- Soporta más de dos personas (4 manos) a la vez.

Algunos puntos interesantes de la organización del dataset:

- La carpeta `_LABELLED_SAMPLES` contiene 48 directorios con los frames de cada video.
- Las anotaciones están en archivos `.mat` (archivos MatLab).
- Las segmentaciones están definidas como polígonos en cada frame.



# ARQUITECTURAS EXISTENTES

## MobileNet V2

Arquitectura enfocada en lograr un performance en dispositivos móviles.

### Beneficios

- Velocidad de Inferencia rápida.
- Liviana y Eficiente
- Estructura Residual Invertida

### Contras

- Accuracy un poco menor a otras arquitecturas.
- Puede tener dificultades con objetos pequeños.

## ResNet

Arquitectura que logra reducir los problemas de degradación y mejora la extracción de features.

### Beneficios

- Extracción de Features muy bueno.
- Buen soporte contra problemas de gradiente desvanecido.

### Contras

- Computacionalmente Caro.
- Modelos de tamaño grande.

## EfficientNet

Arquitectura que utiliza el concepto de escalado compuesto. Trabaja con el ancho, largo y resolución

### Beneficios

- Muy alta Accuracy.
- Escalado eficiente del modelo.

### Contras

- Arquitectura más compleja
- Más nueva, no esta tan probada en producción.



# ARQUITECTURA ELEGIDA

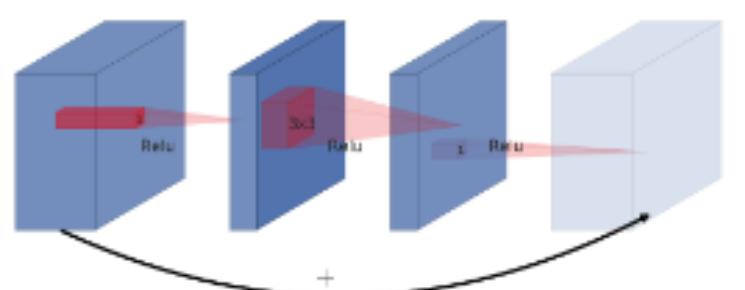
## MobileNet V2

Se decidió utilizar MobileNet debido a la rápida velocidad de inferencia de la Arquitectura, al necesitar rendimiento a tiempo real por la cámara.

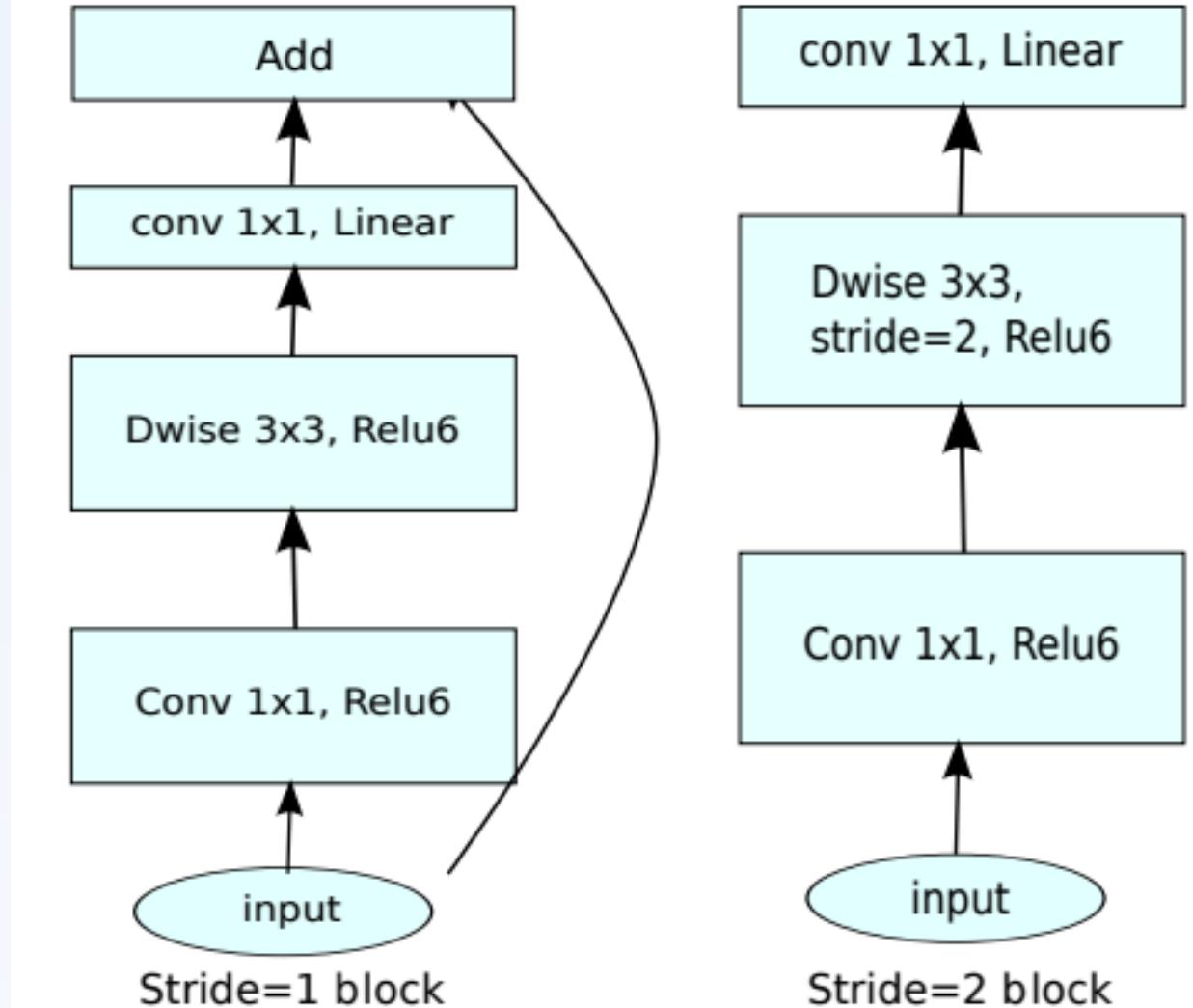
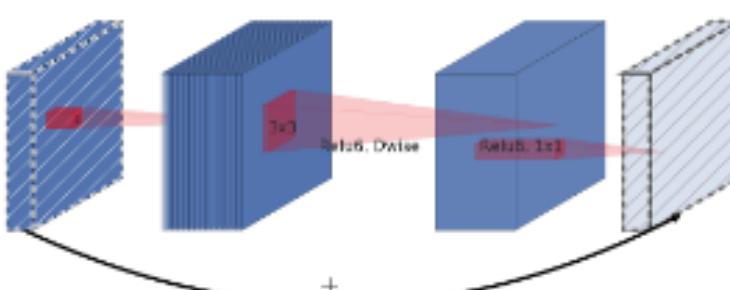
El corazón de la Arquitectura son los bloques de residuo invertido, lo cual utiliza un patrón donde primero expande la información, y después se reduce (como un embudo invertido).

En el contexto de la detección de manos, esta arquitectura resulta particularmente efectiva. Las primeras capas de la red aprenden a detectar características básicas como bordes y texturas, las capas intermedias identifican formas y partes más complejas, y las capas finales combinan toda esta información para reconocer y localizar manos completas en la imagen.

(a) Residual block



(b) Inverted residual block



(d) Mobilenet V2

# DEFINICIÓN DEL MODELO

El modelo utiliza la Arquitectura de MobileNet v2, y funciona de la siguiente manera:

- Recibe de input una imagen RGB de 224x224x3
- Hay un primer bloque de convolución, con tres bloques residuales luego con 128, 256 y 512 canales.
- Cada bloque incluye una capa de activación ReLU y BatchNorm.
- Luego, hay una capa de AdaptiveAvgPool2D de 1x1, para el feature pooling.
- Por último hay un detection head para identificar las bounding boxes.
- El output final se divide en puntajes de predicción y las coordenadas de las bounding boxes.

```
def __init__(self, max_boxes=4): # Reduced max_boxes to match typical hand count
    super(PalmDetectionModel, self).__init__()
    self.max_boxes = max_boxes

    # Feature extraction backbone (using ResNet-like architecture)
    self.features = nn.Sequential(
        # Initial conv block
        nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=3, stride=2, padding=1),

        # Residual blocks
        self._make_layer(64, 128, 2),
        self._make_layer(128, 256, 2),
        self._make_layer(256, 512, 2),

        # Final layers
        nn.AdaptiveAvgPool2d((1, 1))
    )

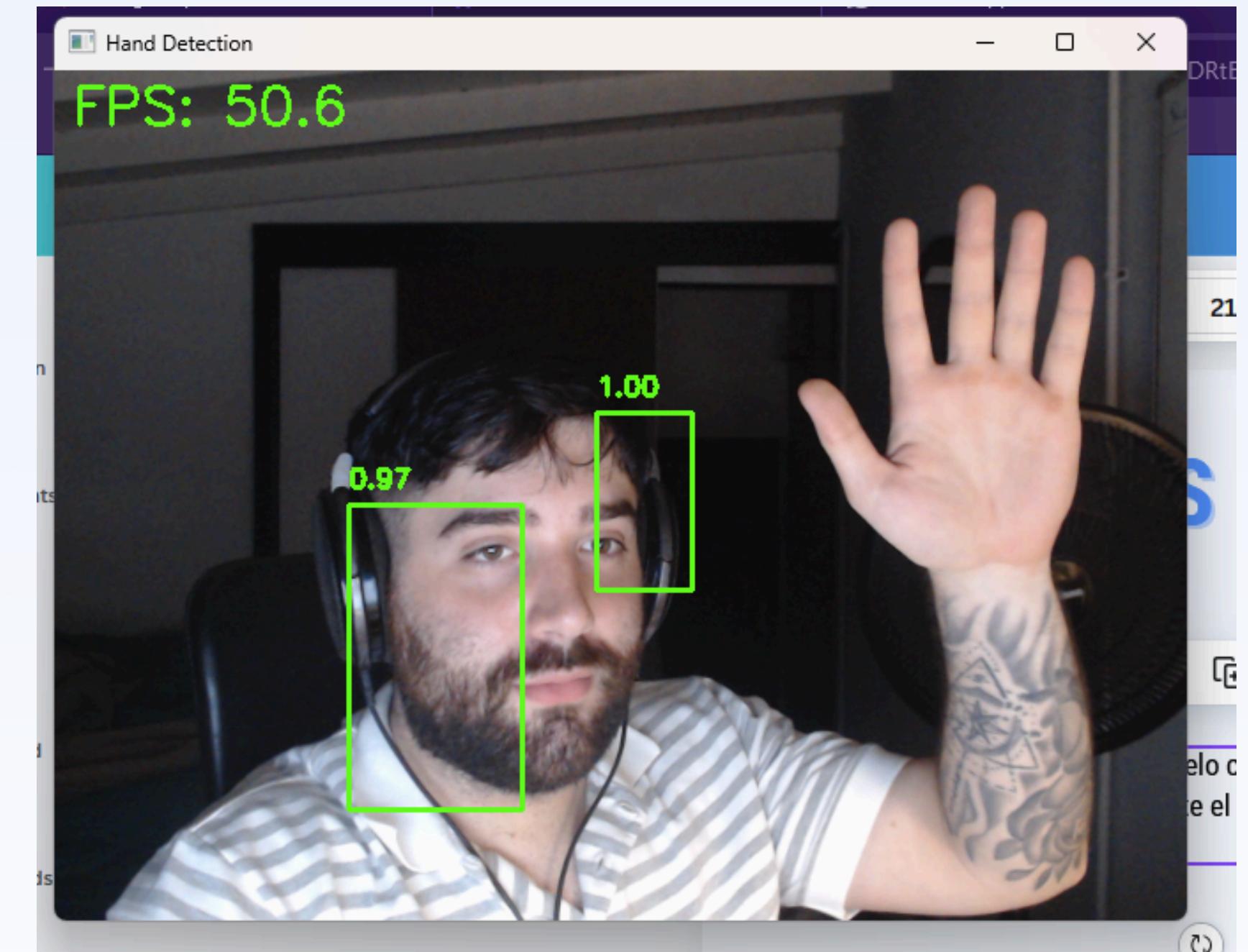
    # Regression head for bounding boxes
    self.bbox_head = nn.Sequential(
        nn.Flatten(),
        nn.Linear(512, 256),
        nn.ReLU(inplace=True),
        nn.Dropout(0.5),
        nn.Linear(256, 128),
        nn.ReLU(inplace=True),
        nn.Linear(128, max_boxes * 5) # 5 values per box (4 coords + 1 conf)
    )

def _make_layer(self, in_channels, out_channels, stride):
    return nn.Sequential(
        # Conv block with skip connection
        nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True),
        nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
        nn.BatchNorm2d(out_channels)
    )
```

# PROBLEMAS DEL MODELO 1

Se entrenó durante 20 epochs el modelo con esta arquitectura, y devolvió un Loss Score de 0.1532, pero igualmente el modelo no funciona al intentar identificar las manos:

- Las bounding boxes no coinciden con las manos, y siempre aparece un bounding box con 100% de accuracy. Esto puede ser debido al split final del resultado.
- No deberíamos obtener las coordenadas, sino únicamente los anchors de la mano para no introducir mas outputs al modelo.
- Actualmente el input es de 224x224, pudiendo perder muchos detalles de la imagen.



# IMPLEMENTACIÓN CON MEDIPIPE

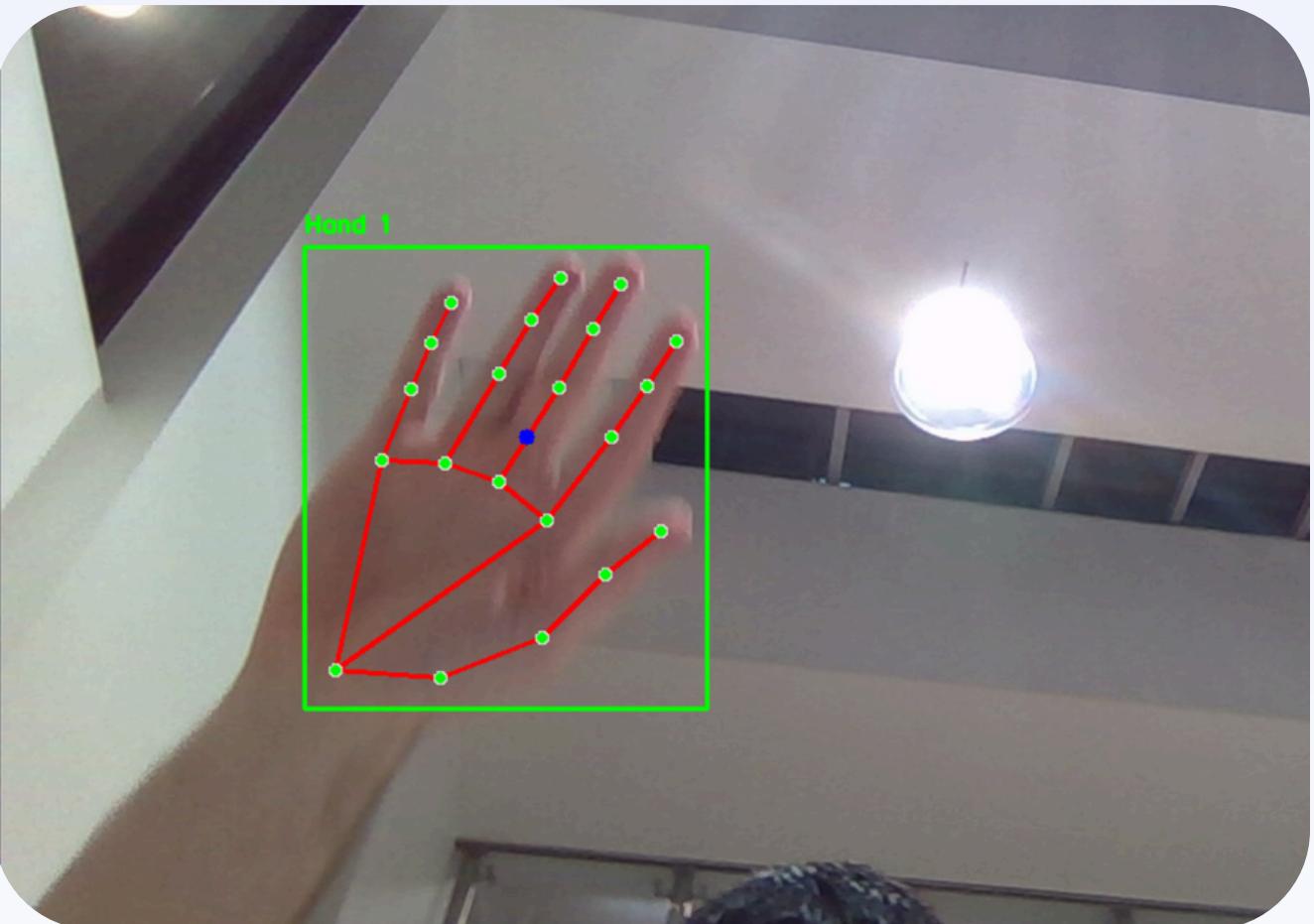
Por problemas de rendimiento, no pude seguir entrenando el modelo e iterandolo, pero encontré otra solución, utilizar **MediaPipe**!

MediaPipe es una librería de Google, cuya meta es crear modelos pre-entrenados para resolver problemas como:

- Detección de posiciones de manos/cuerpo.
- Detección de objetos.
- Detección de caras.

La librería ofrece tanto tareas ya resueltas como modelos pre-entrenados.

En este caso utilizamos el modelo pre-entrenado de **hands**, para la detección de manos



# COMO FUNCIONA LA SOLUTION HANDS?

La solución de MediaPipe para el tracking de manos es un Pipeline de ML con dos modelos:

- una CNN para detectar la presencia de la palma de una mano.
- otra CNN para detectar los puntos en base a la mano detectada.

Esto se resolvió de esta manera, ya que descubrieron que era mucho más eficiente detectar el área de la palma de la mano, y sobre esa información aplicar la detección de la posición de la mano.

No hay detalles sobre la arquitectura específica, pero dice en el Paper que el modelo para detectar las manos esta basado en la implementación de BlazeFace, la cual utiliza una modificación de MobileNet v1.

El modelo para detectar los puntos de la mano es un desarrollo customizado por Google.

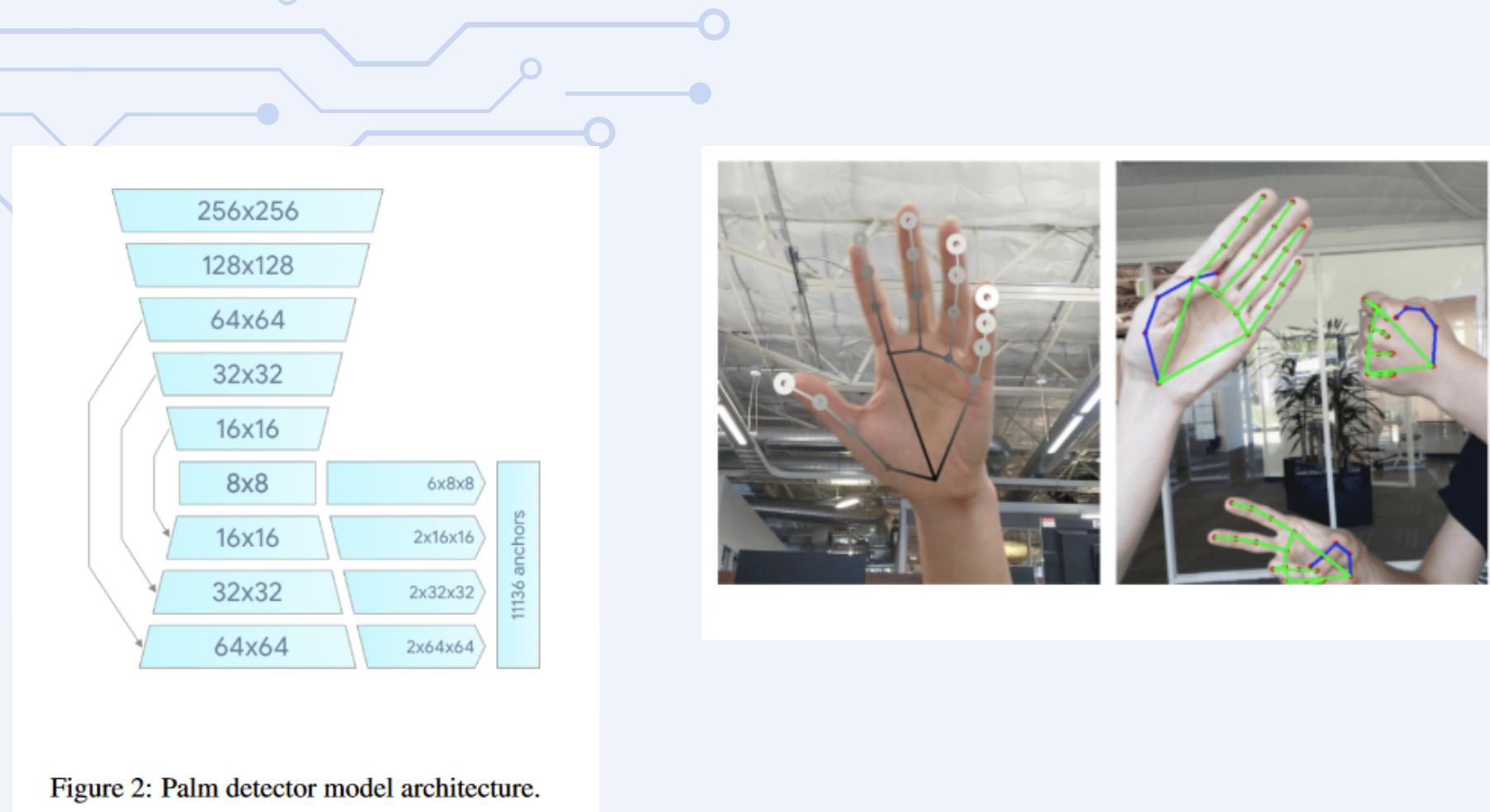


Figure 2: Palm detector model architecture.

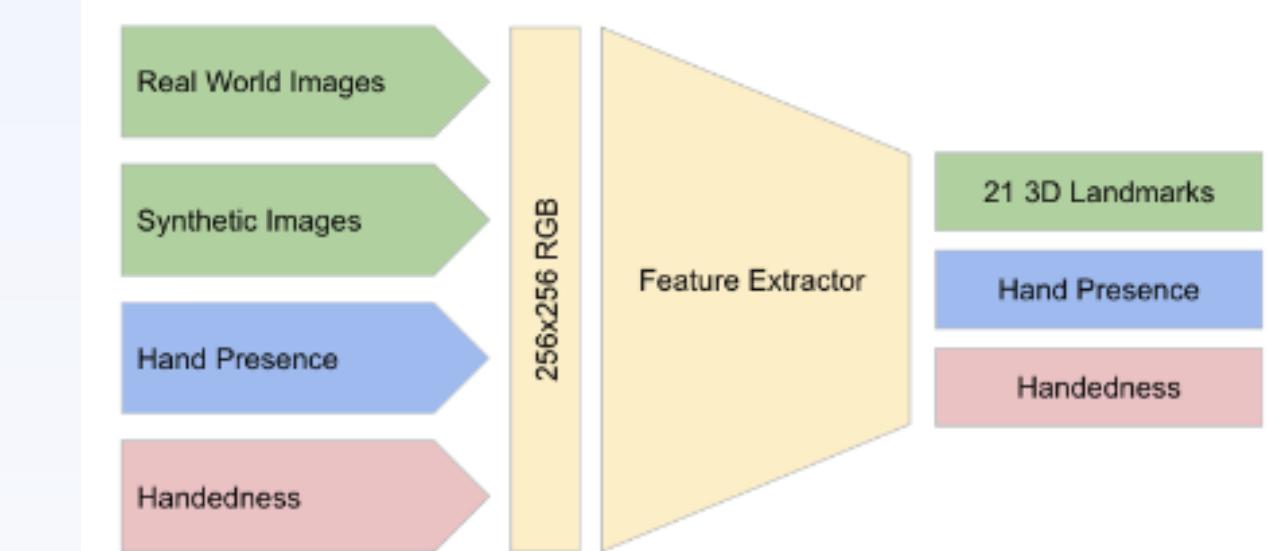
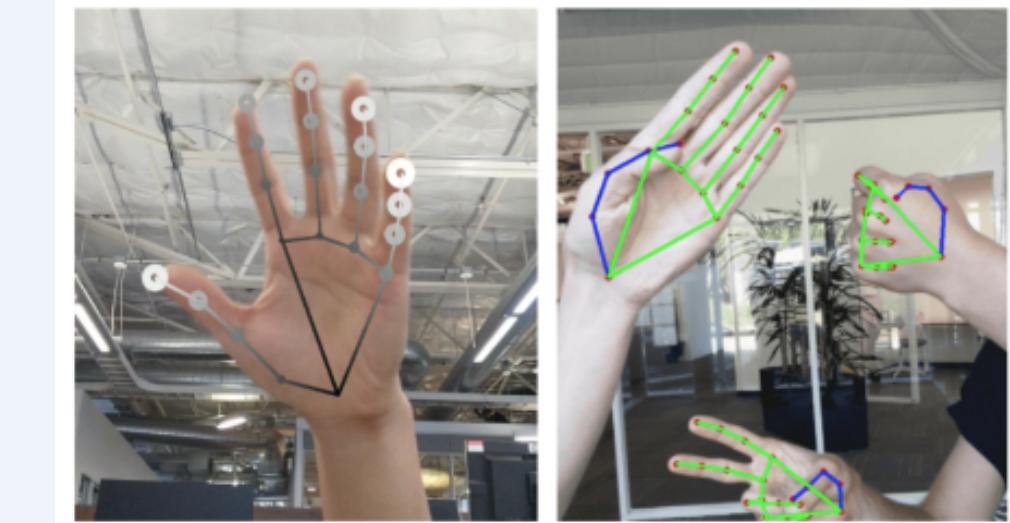


Figure 3: Architecture of our hand landmark model. The model has three outputs sharing a feature extractor. Each head is trained by correspondent datasets marked in the same color. See Section 2.2 for more detail.

# IMPLEMENTACIÓN CON MEDIPIPE

```
while True:  
    ret, frame = cap.read()  
    if not ret:  
        logger.error("Failed to grab frame")  
        break  
  
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  
    results = hands.process(rgb_frame)  
  
    hand_centers = []  
  
    if results.multi_hand_landmarks:  
        for idx, hand_landmarks in enumerate(results.multi_hand_landmarks):  
            mp_draw.draw_landmarks(  
                frame,  
                hand_landmarks,  
                mp_hands.HAND_CONNECTIONS,  
                mp_draw.DrawingSpec(color=(0,255,0), thickness=2, circle_radius=2),  
                mp_draw.DrawingSpec(color=(0,0,255), thickness=2)  
            )  
  
            h, w, _ = frame.shape  
            x_coords = [lm.x * w for lm in hand_landmarks.landmark]  
            y_coords = [lm.y * h for lm in hand_landmarks.landmark]
```

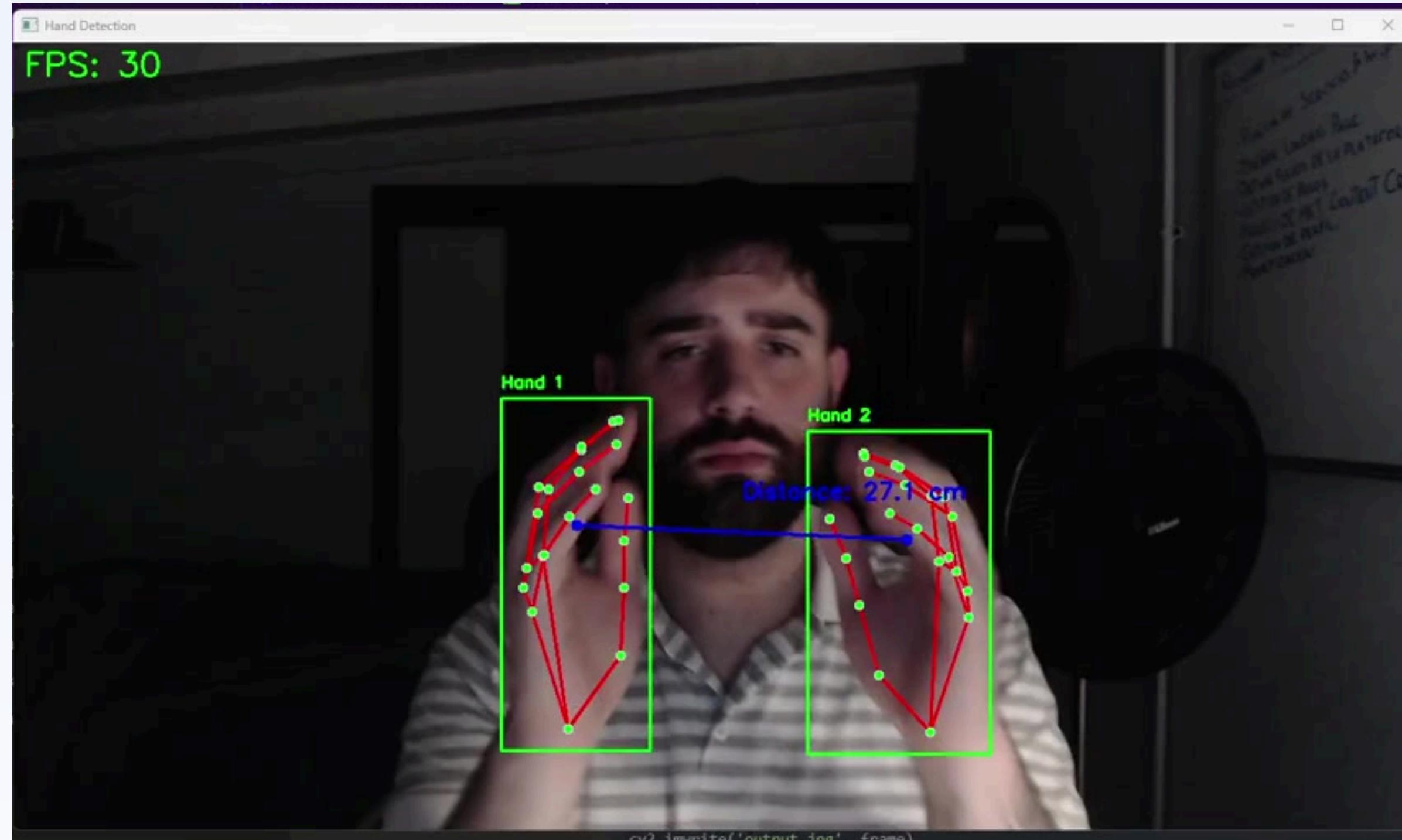
PROCESAR FRAMES

```
# Initialize MediaPipe  
mp_hands = mp.solutions.hands  
mp_draw = mp.solutions.drawing_utils  
hands = mp_hands.Hands(  
    static_image_mode=False,  
    max_num_hands=2,  
    min_detection_confidence=0.7,  
    min_tracking_confidence=0.5  
)
```

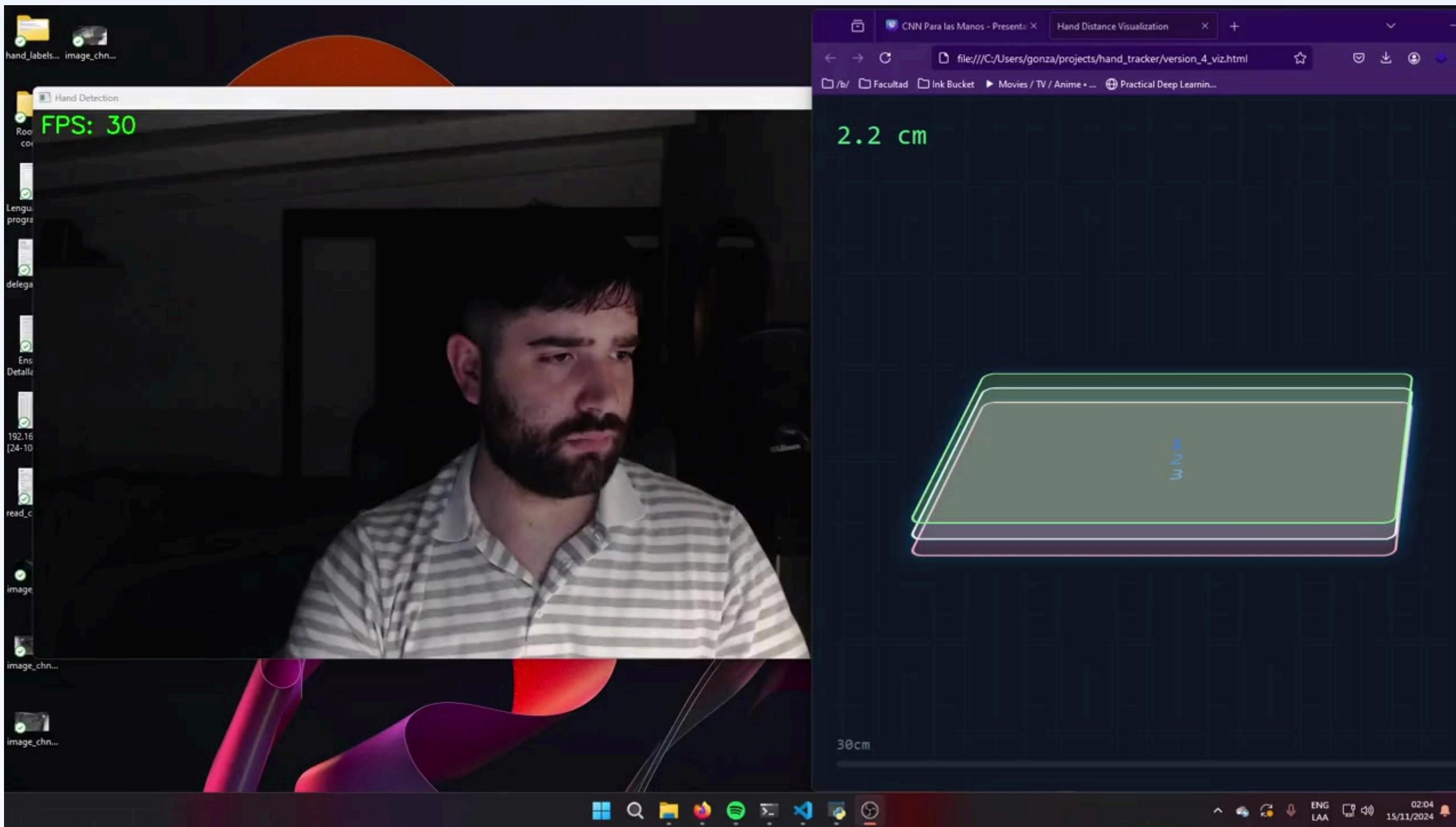
```
if len(hand_centers) == 2:  
    cv2.line(frame, hand_centers[0][0], hand_centers[1][0], (255, 0, 0), 2)  
    pixel_distance = calculate_distance(hand_centers[0][0], hand_centers[1][0])  
    avg_hand_width_pixels = (hand_centers[0][1] + hand_centers[1][1]) / 2  
    pixels_per_cm = avg_hand_width_pixels / AVERAGE_HAND_WIDTH_CM  
    real_distance_cm = pixel_distance / pixels_per_cm  
  
    current_distance = real_distance_cm  
  
    mid_point = (  
        (hand_centers[0][0][0] + hand_centers[1][0][0]) // 2,  
        (hand_centers[0][0][1] + hand_centers[1][0][1]) // 2 - 30  
    )  
    cv2.putText(  
        frame,  
        f"Distance: {real_distance_cm:.1f} cm",  
        mid_point,  
        cv2.FONT_HERSHEY_SIMPLEX,  
        0.7,  
        (255, 0, 0),  
        2  
    )
```

SI HAY DOS MANOS, CALCULAR DISTANCIA

# USO DEL MODELO



# USO DEL MODELO



# ¡MUCHAS GRACIAS!

