TECNOLÓGICO
NACIONAL DE MÉXICO


# Instituto tecnológico de culiacán

Ingeniería en sistemas computacionales

Módulo 4

# Modelo de visión artificial para detección de imagenes

**Nombre:**
- De La Rocha Linarez Diego Alejandro
- Roriguez Cazarez Joaquín

**No. De control:**
- 21170302
- 21170458

**Materia:**
Inteligencia artificial
08-09 hrs

**Maestro:**
Dr. Zuriel Dathan Mora Felix

**Semestre:** 8

29 de Mayo del 2025

## Elección de la arquitectura de red neuronal

La arquitectura de red neuronal elegida para este entrenamiento fue una red neuronal convolucional, esto debido a que es la mejor y ampliamente usada para la clasificación y detección de imágenes que es justamente lo que el problema requiere.

```
# Configuración de parámetros mejorados
img_height = 80
img_width = 80
batch_size = 8  # Reducido para CPU
epochs = 50  # Reducido para pruebas más rápidas
learning_rate = 0.001  # Aumentado ligeramente para CPU
```

Nuestras imágenes tienen un tamaño de 80 x 80 esto debido a las limitaciones de memoria ya que las computadoras disponibles no tienen más de 8 GB de ram y el procesamiento será en CPU, por eso también se decidieron los demás parámetros para no obtener un modelo decente sin comprometer la computadora durante muchas horas.

```
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_height,
img_width, 3)),
        layers.BatchNormalization(),
        layers.Conv2D(32, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),
```

Primera Sección Convolucional

Primera Capa Conv2D(32, (3,3)): Se aplica una capa convolucional con 32 filtros de tamaño 3x3 con función de activación ReLU. Esta capa está configurada para recibir imágenes de tamaño 80x80x3 píxeles que es el tamaño estándar definido en los parámetros del modelo (img_height = 80, img_width = 80). Los 32 filtros detectan características básicas como bordes horizontales, verticales y diagonales.

BatchNormalization: Después se aplica normalización por lotes que estabiliza el entrenamiento al normalizar las activaciones de cada mini-batch, acelerando la convergencia y reduciendo la sensibilidad a la inicialización de pesos.

Segunda Capa Conv2D(32, (3,3)): Se aplica otra capa convolucional con 32 filtros de 3x3 y activación ReLU. Esta segunda capa permite detectar patrones más complejos combinando las características extraídas por la primera capa.

MaxPooling2D(2,2): Se aplica max pooling con ventana de 2x2 que reduce la dimensión espacial de 76x76 a 38x38 (aproximadamente), extrayendo las características más importantes y reduciendo el costo computacional.

Dropout(0.25): Se aplica regularización dropout que desactiva aleatoriamente el 25% de las neuronas durante el entrenamiento, previniendo el sobreajuste y mejorando la generalización.

```
layers.Conv2D(64, (3, 3), activation='relu'),
      layers.BatchNormalization(),
      layers.Conv2D(64, (3, 3), activation='relu'),
      layers.MaxPooling2D((2, 2)),
      layers.Dropout(0.25),
```

Segunda Sección Convolucional
Tercera Capa Conv2D(64, (3,3)): Se duplica el número de filtros a 64 con tamaño 3x3 y activación ReLU. Al tener más filtros, esta capa puede detectar características de nivel medio como formas geométricas simples y texturas más complejas.

BatchNormalization: Nueva normalización por lotes para estabilizar el aprendizaje en esta capa más profunda.

Cuarta Capa Conv2D(64, (3,3)): Otra capa con 64 filtros que refina la detección de patrones de nivel medio, combinando las características extraídas anteriormente.

MaxPooling2D(2,2): Segunda reducción espacial que reduce las dimensiones aproximadamente a 17x17, concentrando la información más relevante.

Dropout(0.25): Segunda aplicación de dropout del 25% para mantener la regularización.

```
layers.Conv2D(128, (3, 3), activation='relu'),
      layers.BatchNormalization(),
      layers.Conv2D(128, (3, 3), activation='relu'),
      layers.MaxPooling2D((2, 2)),
      layers.Dropout(0.25),
```

Tercera Sección Convolucional
Quinta Capa Conv2D(128, (3,3)): Se incrementa a 128 filtros de 3x3 con activación ReLU. Esta capa profunda detecta características de alto nivel como formas

complejas, patrones faciales específicos y combinaciones de rasgos que son cruciales para clasificar emociones.

BatchNormalization: Normalización por lotes para esta capa de alta complejidad.

Sexta Capa Conv2D(128, (3,3)): Última capa convolucional con 128 filtros que refina y combina todas las características de alto nivel extraídas hasta este punto.

MaxPooling2D(2,2): Tercera y última reducción espacial que concentra la información en un mapa de características de aproximadamente 7x7 píxeles.

Dropout(0.25): Última regularización convolucional antes de las capas densas.

```
layers.Flatten(),
       layers.Dense(512, activation='relu'),
       layers.BatchNormalization(),
       layers.Dropout(0.5),
       layers.Dense(256, activation='relu'),
       layers.BatchNormalization(),
       layers.Dropout(0.5),
       layers.Dense(num_classes, activation='softmax')
```

Sección de Clasificación
Flatten(): Convierte los mapas de características 2D (7x7x128) en un vector unidimensional de 6,272 elementos para alimentar las capas densas.

Primera Dense(512): Capa totalmente conectada con 512 neuronas y activación ReLU que aprende combinaciones complejas de todas las características extraídas por las capas convolucionales.

BatchNormalization: Normalización para estabilizar el aprendizaje en las capas densas.

Dropout(0.5): Regularización agresiva del 50% ya que las capas densas son más propensas al sobreajuste.

Segunda Dense(256): Capa de 256 neuronas que refina las representaciones aprendidas y reduce gradualmente la dimensionalidad hacia la salida final.

BatchNormalization: Segunda normalización en capas densas.

Dropout(0.5): Última regularización antes de la clasificación final.

Capa de Salida Dense(num_classes): Capa final con tantas neuronas como clases de emociones hay en el dataset, usando activación softmax que convierte las salidas en probabilidades que suman 1, permitiendo la clasificación multiclase de emociones.

Esta arquitectura está optimizada para imágenes de 80x80 píxeles y utiliza un enfoque jerárquico que va desde características simples hasta representaciones complejas de emociones faciales.

## Obtención de las características de los archivos coco

Inicialización del Dataset COCO

Carga de Anotaciones: Se carga el archivo de anotaciones COCO (_annotations.coco.json) que contiene toda la información sobre las imágenes, categorías y etiquetas. Este archivo JSON estructura la información de manera que cada imagen tiene asociadas sus respectivas anotaciones de emociones.

```
coco = COCO(annotations_path)
        categories = coco.loadCats(coco.getCatIds())
```

Mapeo de Categorías: Se extraen todas las categorías (emociones) disponibles en el dataset y se crea un mapeo category_id_to_label que convierte los IDs originales de COCO a índices secuenciales (0, 1, 2, ...) que el modelo puede usar. Por ejemplo, si COCO tiene IDs [5, 12, 23] para tres emociones, se mapean a [0, 1, 2].

```
category_id_to_label = {cat['id']: idx for idx, cat in
enumerate(categories)}
        global num_classes
        num_classes = len(categories)
```

Filtrado de Categorías: Si existe una categoría padre llamada 'emotions', se filtra automáticamente ya que no representa una emoción específica sino una categoría contenedora.

```
if 'emotions' in [cat['name'] for cat in categories]:
            print("Detectada categoría 'emotions' - será ignorada")
                filtered_categories = [cat for cat in categories if
cat['name'] != 'emotions']
                category_id_to_label = {cat['id']: idx for idx, cat in
enumerate(filtered_categories)}
            num_classes = len(filtered_categories)
            print(f"Clases filtradas: {num_classes}")
                print(f"Categorías finales: {[cat['name'] for cat in
filtered_categories]}")
```

Generador de Datos por Imagen

Obtención de IDs de Imágenes: Se extraen todos los IDs de imágenes disponibles en el dataset usando coco.getImgIds(), que devuelve una lista con todos los identificadores únicos de las imágenes.

```
        image_ids = coco.getImgIds()
        total_images = len(image_ids)
        processed = 0
        successful = 0
```

Carga Individual de Imágenes: Para cada ID de imagen, se obtiene la información completa usando coco.loadImgs(image_id)[0] que incluye el nombre del archivo, dimensiones originales y metadatos.

```
for image_id in image_ids:
            try:
                # Obtener información de la imagen
                image_info = coco.loadImgs(image_id)[0]
                                    image_path  =  os.path.join(images_dir,
image_info['file_name'])
```

Verificación de Archivos: Se verifica que el archivo de imagen existe físicamente en el disco y que no está vacío (tamaño > 0 bytes) antes de intentar procesarlo, evitando errores durante el entrenamiento.

```
                # Verificar que el archivo existe y no está vacío
                if not os.path.exists(image_path):
                    continue

                if os.path.getsize(image_path) == 0:
                    continue
```

Procesamiento de Imágenes

Lectura y Decodificación: Se lee el archivo de imagen desde disco usando tf.io.read_file() y se intenta decodificar en múltiples formatos (JPEG, PNG, formato genérico) para manejar diferentes tipos de imagen con robustez.

```
# Decodificar imagen con mejor manejo de errores
                try:
                    image = tf.image.decode_jpeg(image, channels=3)
                except:
                    try:
                        image = tf.image.decode_png(image, channels=3)
```

```
                    except:
                        try:
                                    image = tf.image.decode_image(image,
channels=3)
                        except:
                            continue
```

Redimensionamiento con Padding: Se usa tf.image.resize_with_pad() para redimensionar las imágenes a 80x80 píxeles manteniendo la relación de aspecto original. Si la imagen no es cuadrada, se agrega padding negro para evitar distorsión.

```
image.set_shape([None, None, 3])
                     image = tf.image.resize_with_pad(image, img_height,
img_width)
                image = tf.cast(image, tf.float32)
```

Normalización Zero-Centered: Se aplica la fórmula (image / 127.5) - 1.0 que convierte los valores de píxeles del rango [0, 255] al rango [-1, 1]. Esta normalización centrada en cero mejora la convergencia del entrenamiento al tener valores equilibrados alrededor del cero.

```
# Normalización mejorada (zero-centered)
                image = (image / 127.5) - 1.0  # Normalización [-1, 1]
```

Extracción de Etiquetas y Características

Obtención de Anotaciones: Para cada imagen, se extraen todas las anotaciones asociadas usando coco.getAnnIds(imgIds=image_id) y luego se cargan los detalles completos con coco.loadAnns().

```
annotation_ids = coco.getAnnIds(imgIds=image_id)
                annotations = coco.loadAnns(annotation_ids)
```

Filtrado de Etiquetas Válidas: Se filtran solo las anotaciones que corresponden a categorías válidas (excluyendo la categoría padre 'emotions') y se convierten a los índices mapeados previamente.

```
                valid_labels = []
                for ann in annotations:
                    if ann['category_id'] in category_id_to_label:

valid_labels.append(category_id_to_label[ann['category_id']])
```

Selección de Etiqueta Principal: Como una imagen puede tener múltiples anotaciones, se toma solo la primera etiqueta válida (valid_labels[0]) para evitar conflictos en la clasificación. Esto asume que la primera anotación es la más relevante.

Omisión de Imágenes Sin Etiquetas: Si una imagen no tiene ninguna etiqueta válida, se omite completamente del dataset para evitar que afecte negativamente el entrenamiento.

```
                if valid_labels:
                    label = valid_labels[0]
                else:
                    continue  # Saltar imágenes sin etiquetas válidas
```

Optimización del Dataset
Creación del Dataset de TensorFlow: Se usa tf.data.Dataset.from_generator() con especificaciones precisas de forma y tipo de datos: imágenes de 80x80x3 píxeles como float32 y etiquetas como enteros int32.

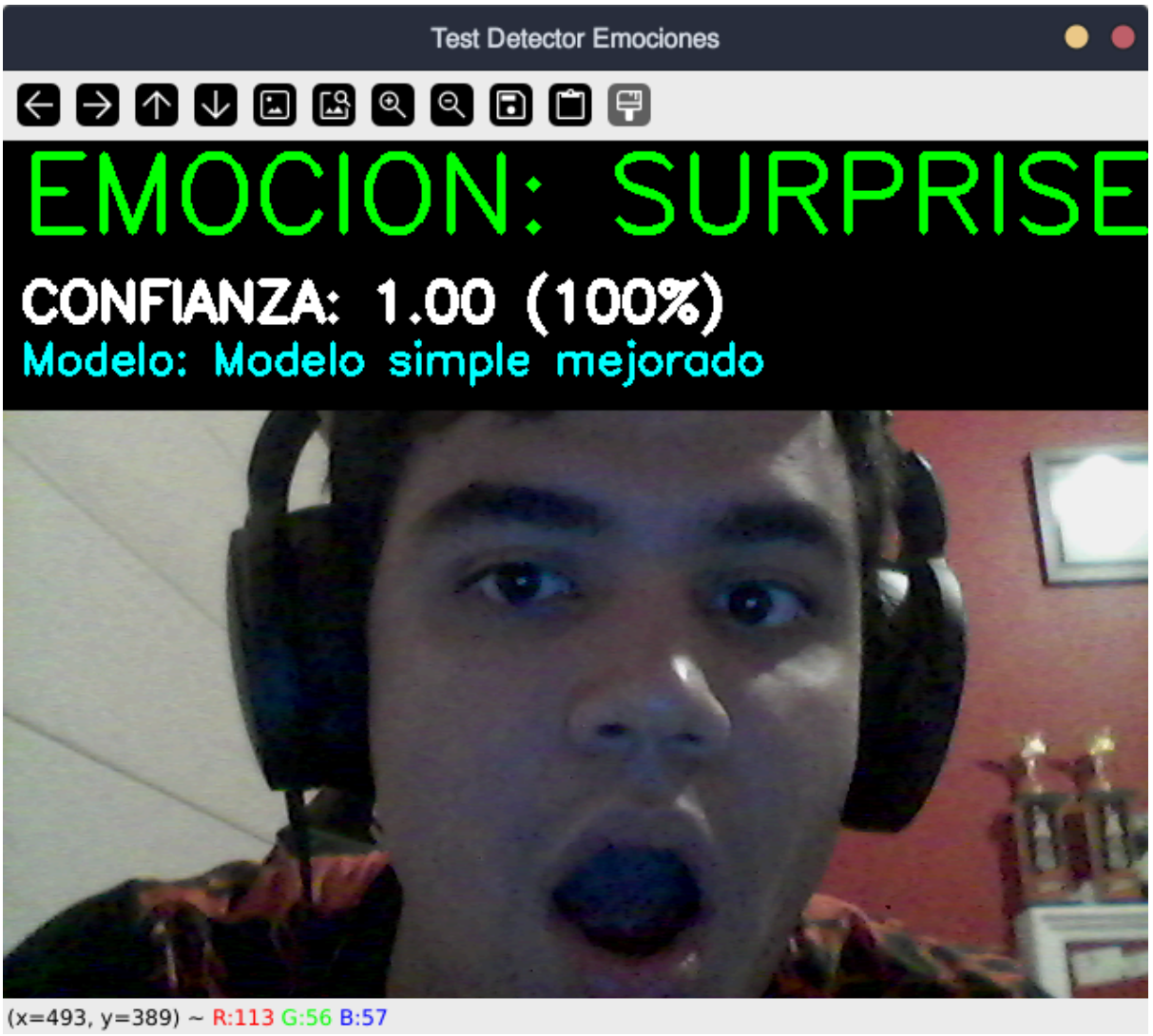
Data Augmentation Condicional: Solo se aplica augmentación de datos al conjunto de entrenamiento (is_training=True) para aumentar la variabilidad y mejorar la generalización sin afectar los conjuntos de validación y prueba.

Pipeline de Optimización: Se aplican optimizaciones secuenciales:

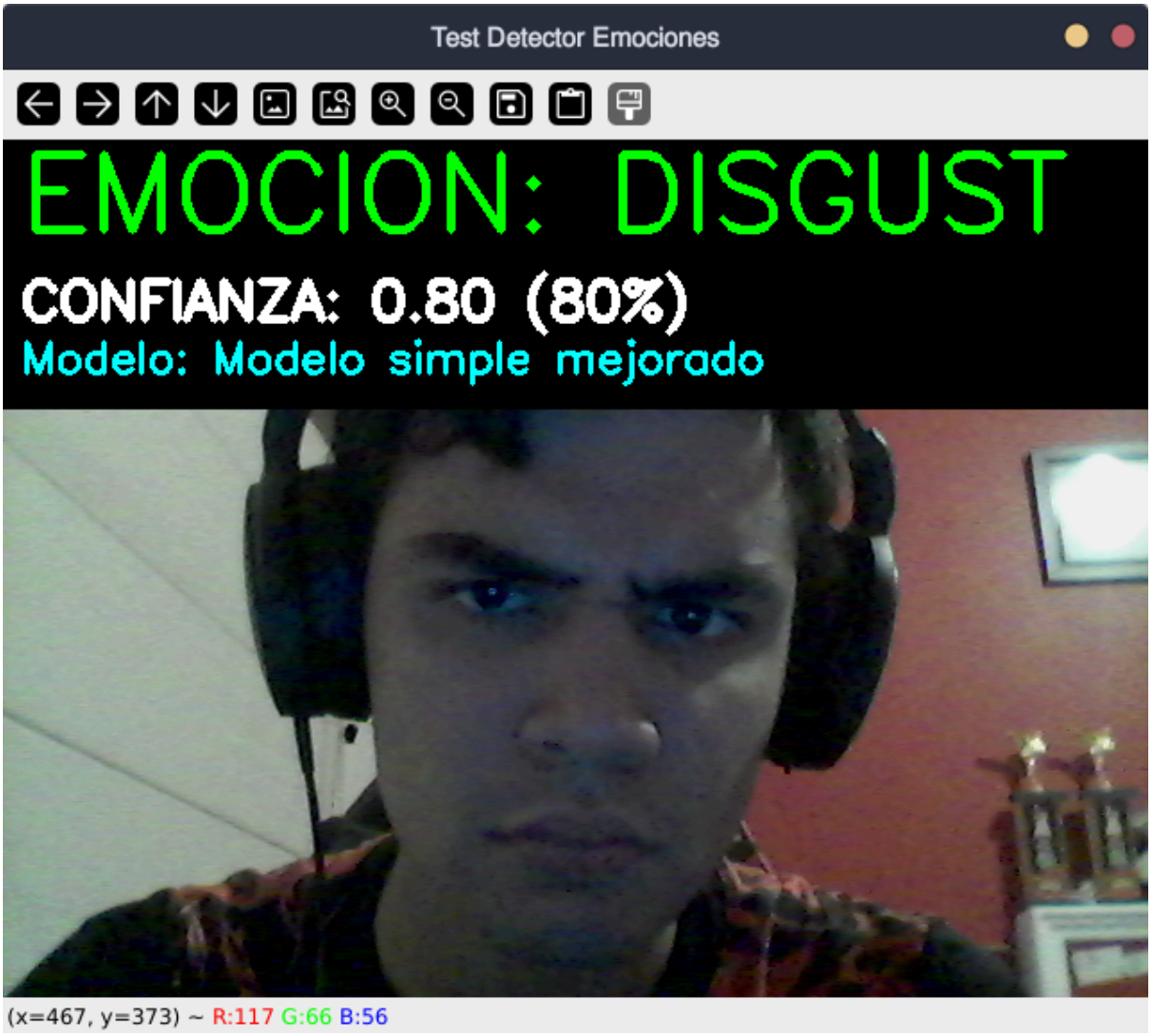
Cache(): Almacena el dataset procesado en memoria para evitar reprocesamiento
Shuffle(): Mezcla aleatoriamente las muestras solo en entrenamiento con buffer de 500 imágenes
Batch(): Agrupa las imágenes en lotes de 8 (optimizado para CPU)
Prefetch(): Prepara el siguiente lote mientras se procesa el actual, maximizando la utilización de recursos

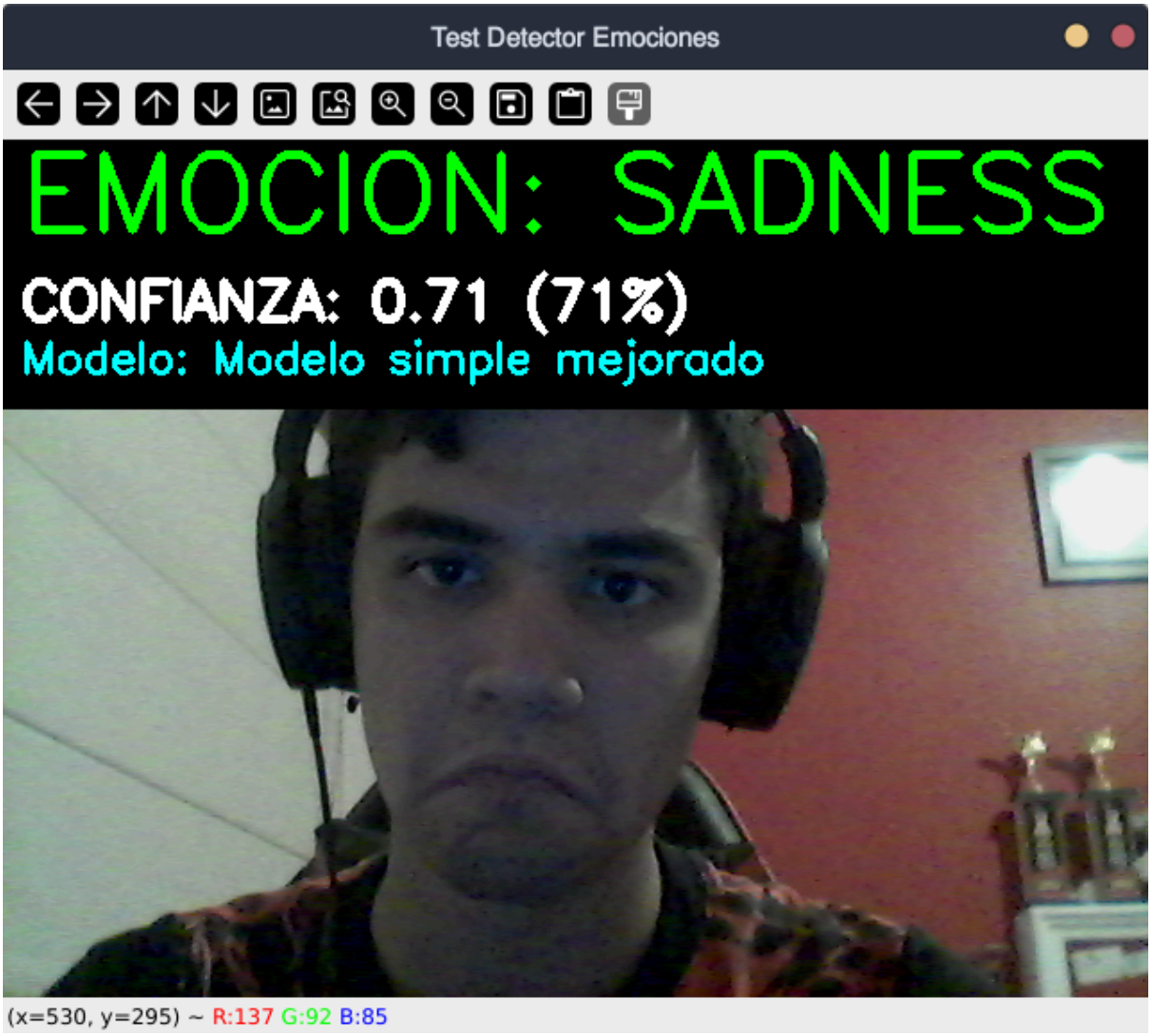
```
# Optimizaciones del dataset
        dataset = dataset.cache()
        if is_training:
             dataset = dataset.shuffle(buffer_size=500)  # Reducido para
CPU
        dataset = dataset.batch(batch_size)
        dataset = dataset.prefetch(tf.data.AUTOTUNE)
```

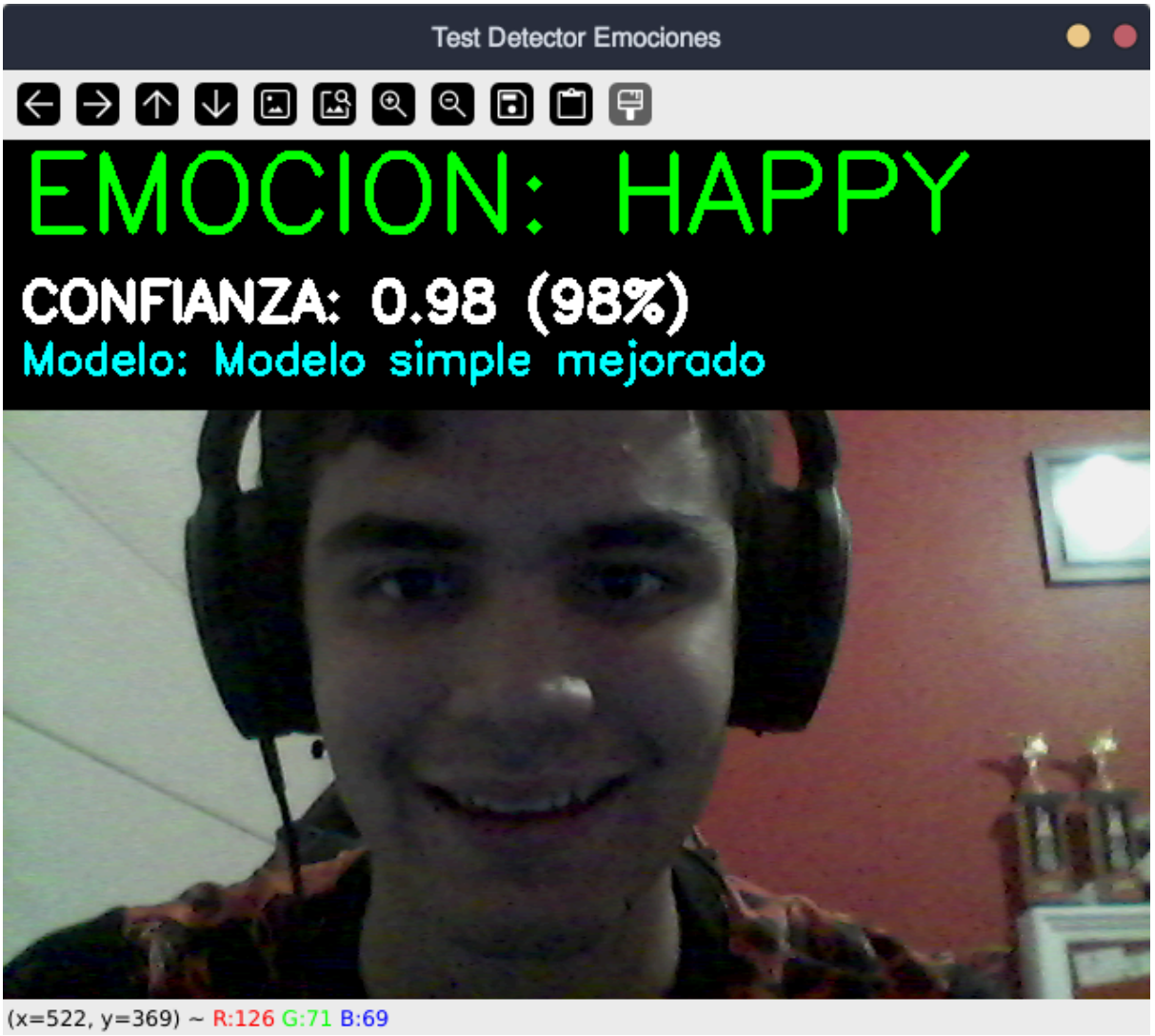
**Pruebas del modelo**




Test Detector Emociones
EMOCION: ANGER
CONFIANZA: 0.99 (99%)
Modelo: Modelo simple mejorado
(x=467, y=373) ~ R:122 G:60 B:67


Test Detector Emociones
EMOCION: DISGUST
CONFIANZA: 0.80 (80%)
Modelo: Modelo simple mejorado
(x=467, y=373) ~ R:117 G:66 B:56

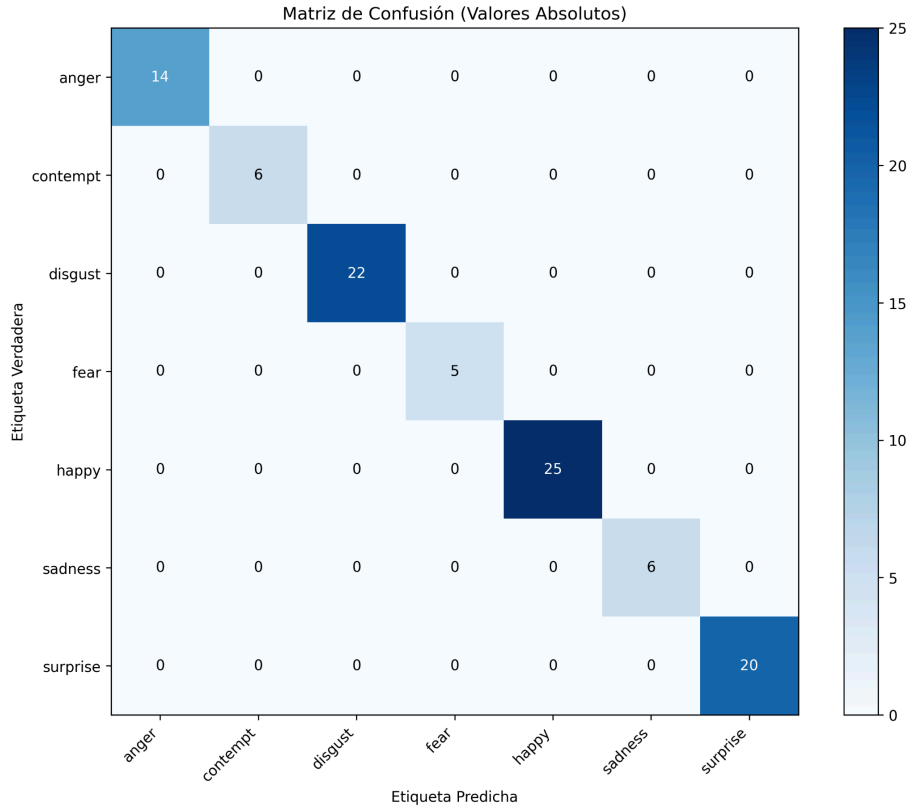

Test Detector Emociones
EMOCION: SADNESS
CONFIANZA: 0.71 (71%)
Modelo: Modelo simple mejorado
(x=530, y=295) ~ R:137 G:92 B:85



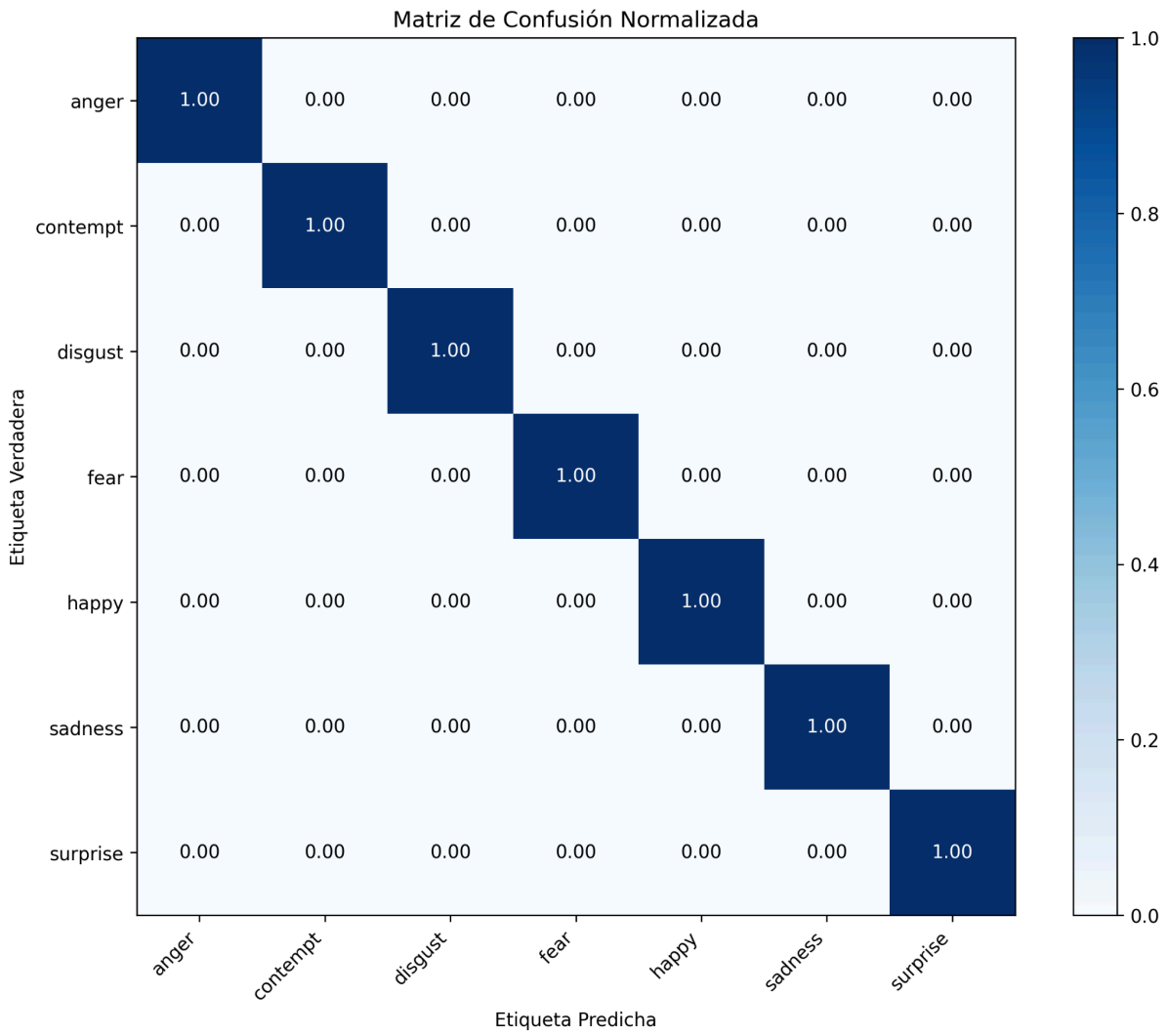

Video de youtube: https://youtu.be/i5HZ_8WjFOY

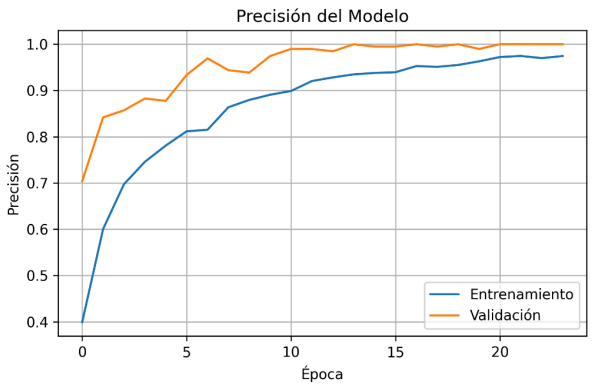
# Validación

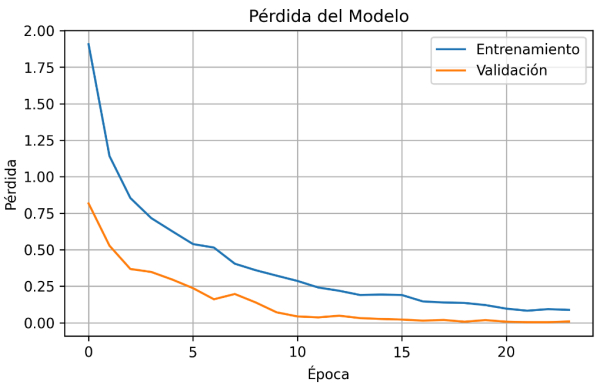
Aquí se muestran las validaciones para comprobar que el modelo está funcionando correctamente




.


Matriz de Confusión Normalizada
anger
contempt
disgust
fear
happy
sadness
surprise
1.00
0.00
Etiqueta Verdadera
Etiqueta Predicha
1.0
0.8
0.6
0.4
0.2
0.0


Precisión del Modelo
Precisión
Época
Entrenamiento
Validación


Pérdida del Modelo
Pérdida
Época
Entrenamiento
Validación

```
================================================
REPORTE DE CLASIFICACIÓN
================================================
              precision    recall  f1-score   support

       anger       1.00      1.00      1.00        14
    contempt       1.00      1.00      1.00         6
     disgust       1.00      1.00      1.00        22
        fear       1.00      1.00      1.00         5
       happy       1.00      1.00      1.00        25
     sadness       1.00      1.00      1.00         6
    surprise       1.00      1.00      1.00        20

    accuracy                           1.00        98
   macro avg       1.00      1.00      1.00        98
weighted avg       1.00      1.00      1.00        98
```

## Conclusiones

Este proyecto de visión artificial nos ayudó a entender cómo funcionan internamente las redes neuronales, como configurar capas de redes neuronales usando las distintas herramientas que nos provee actualmente las librerías del lenguaje python democratizando así la posibilidad de que todos podamos generar y aprender cosas tan importantes en la actualidad como lo es la IA, también nos permitió conocer un nuevo tipo de notación para la notación de imágenes como lo es COCO.
Además de la importancia de escoger un buen dataset y preprocesar.