Katherine Michelle Ortiz Burgos
Código: 506221041
Juan Esteban Acosta Rodriguez
Código: 506221068
Profesor: Oscar Alexander Mendez Aguirre
Fundación Universitaria Konrad Lorenz

# Prototipo Detector De Emociones

## I. Resumen

Este prototipo consiste en la detección de emociones en imágenes mediante la creación de un microservicio API con FastAPI y una aplicación web. El microservicio, implementado en FastAPI, recibe imágenes y utiliza un modelo de inteligencia artificial entrenado para predecir emociones, devolviendo los resultados de manera eficiente. La aplicación web, diseñada con HTML, CSS y JavaScript para la interfaz y jQuery para la interacción, permite a los usuarios cargar imágenes y visualizar las emociones detectadas.

## II. Introducción

En este prototipo, la finalidad es proporcionar a los usuarios una herramienta fácil de usar que permita analizar las emociones presentes en imágenes cargadas, utilizando para ello un modelo de inteligencia artificial previamente entrenado. Este prototipo se divide en dos componentes principales: el backend, que gestiona la lógica del procesamiento de imágenes y predicción de emociones, y el frontend, que ofrece una interfaz intuitiva y atractiva para la interacción del usuario. La combinación de estos dos componentes proporciona una experiencia de usuario fluida y eficiente, facilitando la carga de imágenes y la visualización de resultados en tiempo real.

En el lado del servidor, se emplea FastAPI, un marco web rápido para Python, para manejar las solicitudes HTTP. Utilizando TensorFlow y TensorFlow Hub, se carga un modelo de aprendizaje profundo previamente entrenado, capaz de reconocer emociones en imágenes. Cuando se recibe una imagen a través de una solicitud POST, el backend procesa la imagen y devuelve la emoción detectada como respuesta.

Por otro lado, en el lado del cliente, el frontend proporciona una interfaz intuitiva para que los usuarios carguen imágenes. Esto se logra mediante HTML y CSS para estructurar y diseñar la página web, y JavaScript con jQuery para manejar la interacción del usuario. Los usuarios pueden seleccionar una imagen, enviarla al servidor y recibir la emoción detectada en respuesta.

## III. Colab Modelo Entrenado

El prototipo desarrollado en Google Colab se centra en la implementación y entrenamiento de un modelo de red neuronal para el reconocimiento de emociones en imágenes. Mediante el uso de datos etiquetados que representan expresiones faciales de felicidad, tristeza y estrés, se ha creado un prototipo capaz de analizar imágenes y clasificarlas en una de estas categorías emocionales. Este trabajo fusiona la potencia computacional de Google Colab con técnicas de aprendizaje profundo para ofrecer una herramienta que pueda identificar y categorizar estados emocionales a partir de imágenes visuales.

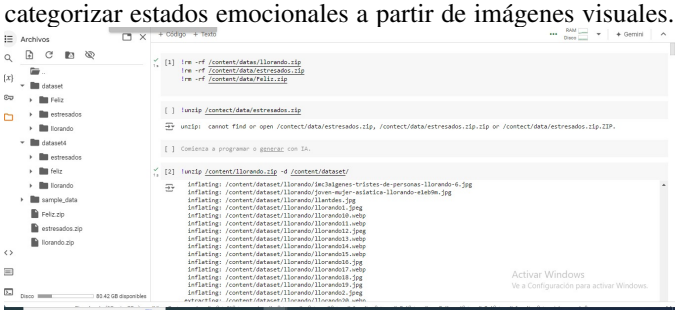


Figura 1. Colab Modelo Entrenado.

## IV. Código De Elaboración Del Modelo Entrenado

A continuación se presenta el código utilizado para entrenar el modelo de red neuronal destinado al reconocimiento de emociones en imágenes. Se detallará paso a paso el proceso de preprocesamiento de datos, la construcción y configuración de la arquitectura de la red, la compilación del modelo y el entrenamiento con el conjunto de datos proporcionado. Además, se explicarán las métricas de evaluación utilizadas para medir el rendimiento del modelo y se ofrecerán recomendaciones para ajustes y mejoras futuras. Este código se ha desarrollado en Python, haciendo uso de bibliotecas como TensorFlow y Keras, y se ha ejecutado en el entorno colaborativo de Google Colab para aprovechar su potencia computacional y facilidad de acceso.

1. En una instrucción de terminal que eliminará de manera recursiva y forzada los archivos y directorios especificados. En este caso, los directorios ”/content/datas/llorando.zip”, ”/content/data/estresados.zip” y ”/content/data/Feliz.zip” serán eliminados de forma permanente del sistema de archivos. Es importante tener cuidado al ejecutar comandos de este tipo, ya que pueden eliminar datos importantes de manera irreversible.
2. Se usan comandos de terminal descomprimen archivos ZIP en directorios específicos. El comando ”!unzip” se utiliza para descomprimir archivos ZIP, seguido de la ruta del archivo ZIP que se va a descomprimir y la opción ”-d”

que indica el directorio de destino donde se van a extraer los archivos.

3. Se listan el contenido de los directorios ”/content/dataset/llorando”, ”/content/dataset/estresados” y ”/content/dataset/Feliz” y luego cuentan el número de archivos y directorios presentes en cada uno de ellos. La salida ””, indica el número de elementos listados en cada uno de los directorios.

```
!ls /content/dataset/llorando | wc -l #475
!ls /content/dataset/estresados | wc -l #515
!ls /content/dataset/Feliz | wc -l #515
```

```
200
200
193
```

Figura 2. Contenido De Directorios.

4. Como objetivo de visualizar las primeras 25 imágenes contenidas en tres carpetas distintas (’llorando’, ’estresados’ y ’Feliz’) dentro del directorio ’/content/dataset/’. Utiliza la biblioteca Matplotlib para crear una figura de 5x5 subtramas, donde cada subtrama representa una imagen. El proceso implica iterar sobre los archivos en cada carpeta, cargando y mostrando las imágenes utilizando la función mpimg.imread() y plt.imshow(). Este enfoque ofrece una forma eficiente de inspeccionar el contenido de cada carpeta.

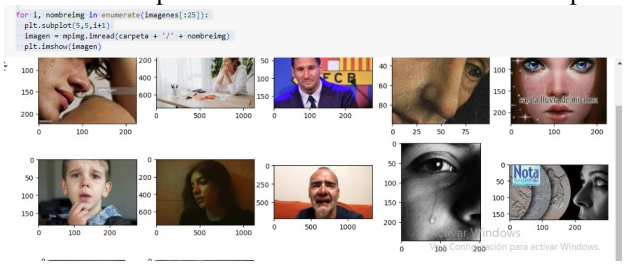

Figura 3. Imágenes Llorando.

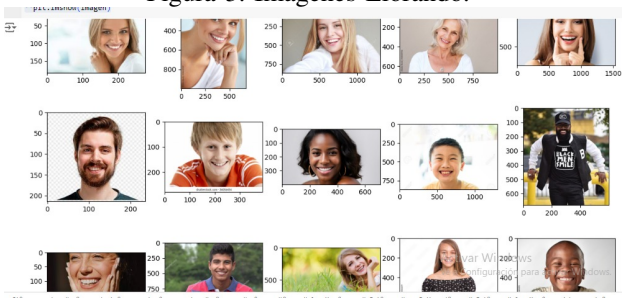

Figura 4. Imágenes Estresado.

Figura 5. Imágenes Feliz.

5. Se usa !mkdir para crear una estructura de directorios organizada para un nuevo conjunto de datos. El directorio principal ’dataset4’ se crea primero, seguido de tres subdirectorios adicionales: ’llorando’, ’estresados’ y ’feliz’. Esta organización jerárquica sugiere una clasificación basada en estados emocionales, donde cada subdirectorio contendría imágenes relacionadas con una emoción específica, como tristeza, estrés o felicidad.
6. Se importa el módulo shutil, que es una herramienta de alto nivel para operaciones de archivos,como copiar, mover y eliminar.
7. Se utiliza el módulo shutil para realizar la tarea de copiar archivos desde carpetas de origen a carpetas de destino. Se han definido rutas específicas para las imágenes de cada categoría emocional (’llorando’, ’estresados’ y ’Feliz’). Luego, mediante un bucle for, se iteran sobre los nombres de archivo en cada carpeta de origen, copiando los primeros 200 archivos de ’llorando’ y ’estresados’, y los primeros 193 archivos de ’Feliz’ a las carpetas de destino correspondientes dentro de ’dataset4’.
8. Se vuelve a istar el contenido de los directorios ’/content/dataset4/llorando’, ’/content/dataset4/estresados’ y ’/content/dataset4/feliz’, respectivamente, y luego cuentan el número de archivos y directorios presentes en cada uno de ellos.

```
!ls /content/dataset4/llorando | wc -l
!ls /content/dataset4/estresados | wc -l
!ls /content/dataset4/feliz | wc -l
```

```
200
200
193
```

Figura 6. Directorios.

9. Con la clase `ImageDataGenerator` de TensorFlow Keras para generar y aumentar lotes de imágenes en tiempo real, un proceso fundamental en el entrenamiento de modelos de redes. Al configurar el generador con una serie de transformaciones, como rotación, cambio de tamaño, cizallamiento y zoom, junto con la normalización de los valores de píxeles, se busca aumentar la diversidad y cantidad de datos disponibles para el entrenamiento, lo que puede mejorar la capacidad del modelo para generalizar a datos nuevos y no vistos. Además, se dividen los datos en conjuntos de entrenamiento y validación utilizando la opción `validation_split`, lo que facilita la evaluación del rendimiento del modelo durante el entrenamiento. Mediante el uso de `flow_from_directory`, se carga y procesa el conjunto de datos de manera eficiente directamente desde los directorios de imágenes, simplificando así el flujo de trabajo de preparación de datos.
10. TensorFlow y TensorFlow Hub para cargar el modelo preentrenado MobileNetV2, específicamente la versión diseñada para la extracción de características. MobileNetV2 es una arquitectura de red neuronal bien conocida y eficiente, que se ha entrenado en conjuntos de datos masivos para aprender características útiles de

las imágenes. Al cargar este modelo a través de TensorFlow Hub, se accede a su capacidad para extraer características relevantes de las imágenes de entrada. La URL proporcionada dirige el código a la ubicación del modelo preentrenado en TensorFlow Hub. La capa Keras se configura con las dimensiones de entrada esperadas para las imágenes (224x224x3), lo que permite que el modelo procese imágenes con esta resolución y número de canales.

11. Posteriormente se establece un modelo de red neuronal utilizando MobileNetV2 como base preentrenada y añade una capa densa al final para realizar la clasificación en tres clases diferentes. Al establecer mobilenetv2.trainable = False, se congela el modelo preentrenado, lo que impide que sus pesos se modifiquen durante el entrenamiento. Esto es útil para evitar la pérdida de los conocimientos previos aprendidos por MobileNetV2. El modelo final se muestra utilizando modelo.summary(), proporcionando una visión general de su arquitectura y la cantidad de parámetros entrenables y no entrenables.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              2257984

 dense (Dense)               (None, 3)                 3843

=================================================================
Total params: 2261827 (8.63 MB)
Trainable params: 3843 (15.01 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

Figura 7. Resumen De Entrenamiento.

12. Se compila el modelo de red neuronal, estableciendo los parámetros esenciales para su entrenamiento. El optimizador seleccionado es Adam, una opción popular conocida por su eficacia en la optimización de redes neuronales. La función de pérdida utilizada es la entropía cruzada categórica, adecuada para problemas de clasificación con múltiples clases. Además, se incluye la métrica de precisión para evaluar el rendimiento del modelo durante el entrenamiento y la validación. Con esta configuración, el modelo está listo para ser entrenado utilizando datos de entrenamiento, lo que permitirá ajustar sus pesos para minimizar la pérdida y maximizar la precisión en la clasificación de las muestras.
13. Se entrena el modelo de red neuronal utilizando un número específico de épocas, en este caso, 200. El método `fit()` se utiliza para ajustar los pesos del modelo utilizando los datos generados por los objetos `data_gen_entrenamiento` y `data_gen_pruebas`. Durante cada época, el modelo se entrena en lotes de tamaño 32, lo que significa que se actualizan los pesos del modelo después de procesar 32 imágenes. El proceso de entrenamiento incluye la validación del modelo en los datos generados por `data_gen_pruebas`. El historial del entrenamiento se guarda en la variable `historial`, lo que permite analizar y visualizar las métricas de rendimiento del modelo, como la pérdida y la precisión, a lo largo del entrenamiento.

```
...  Epoch 134/200
     14/14 [==============================] - 23s 2s/step - loss: 0.2154 - accuracy: 0.9029 - val_loss: 0.6453 - val_accuracy: 0.7248
     Epoch 135/200
     14/14 [==============================] - 21s 1s/step - loss: 0.2070 - accuracy: 0.9142 - val_loss: 0.5627 - val_accuracy: 0.7890
     Epoch 136/200
     14/14 [==============================] - 27s 2s/step - loss: 0.1972 - accuracy: 0.9233 - val_loss: 0.5509 - val_accuracy: 0.8073
     Epoch 137/200
     14/14 [==============================] - 23s 2s/step - loss: 0.2147 - accuracy: 0.9165 - val_loss: 0.4577 - val_accuracy: 0.8257
     Epoch 138/200
     14/14 [==============================] - 22s 2s/step - loss: 0.2498 - accuracy: 0.9120 - val_loss: 0.6339 - val_accuracy: 0.7706
     Epoch 139/200
     14/14 [==============================] - 23s 2s/step - loss: 0.2522 - accuracy: 0.9142 - val_loss: 0.5153 - val_accuracy: 0.8624
     Epoch 140/200
     14/14 [==============================] - 22s 2s/step - loss: 0.1742 - accuracy: 0.9368 - val_loss: 0.4924 - val_accuracy: 0.8073
     Epoch 141/200
     14/14 [==============================] - 23s 2s/step - loss: 0.2155 - accuracy: 0.9165 - val_loss: 0.5724 - val_accuracy: 0.7615
     Epoch 142/200
     14/14 [==============================] - 23s 2s/step - loss: 0.1973 - accuracy: 0.9255 - val_loss: 0.5602 - val_accuracy: 0.7890
     Epoch 143/200
     14/14 [==============================] - 22s 2s/step - loss: 0.2142 - accuracy: 0.9233 - val_loss: 0.6382 - val_accuracy: 0.7615
     Epoch 144/200
     14/14 [==============================] - 22s 2s/step - loss: 0.2208 - accuracy: 0.9210 - val_loss: 0.5613 - val_accuracy: 0.8073
     Epoch 145/200
     14/14 [==============================] - 22s 2s/step - loss: 0.2157 - accuracy: 0.9142 - val_loss: 0.5606 - val_accuracy: 0.8532
     Epoch 146/200
```

Figura 8. Epocas.

14. Se usa el historial de entrenamiento almacenado en la variable historial para visualizar la precisión y la pérdida del modelo a lo largo de las épocas de entrenamiento. Se extraen las métricas de precisión y pérdida tanto para el conjunto de entrenamiento como para el conjunto de pruebas del historial de entrenamiento. Luego, se trazan dos gráficos lado a lado para comparar la evolución de la precisión y la pérdida durante el entrenamiento.
    - En el primer gráfico, se muestran las curvas de precisión para el conjunto de entrenamiento y el conjunto de pruebas a lo largo de las épocas. La precisión de entrenamiento se representa con una línea continua, mientras que la precisión de pruebas se representa con una línea punteada. Esto permite visualizar cómo la precisión del modelo evoluciona a medida que se entrena y se valida en diferentes épocas.
    - En el segundo gráfico, se muestran las curvas de pérdida para el conjunto de entrenamiento y el conjunto de pruebas a lo largo de las épocas. Al igual que con la precisión, la pérdida de entrenamiento se representa con una línea continua, mientras que la pérdida de pruebas se representa con una línea punteada. Este gráfico proporciona información sobre cómo la pérdida del modelo cambia durante el entrenamiento y la validación en diferentes épocas.

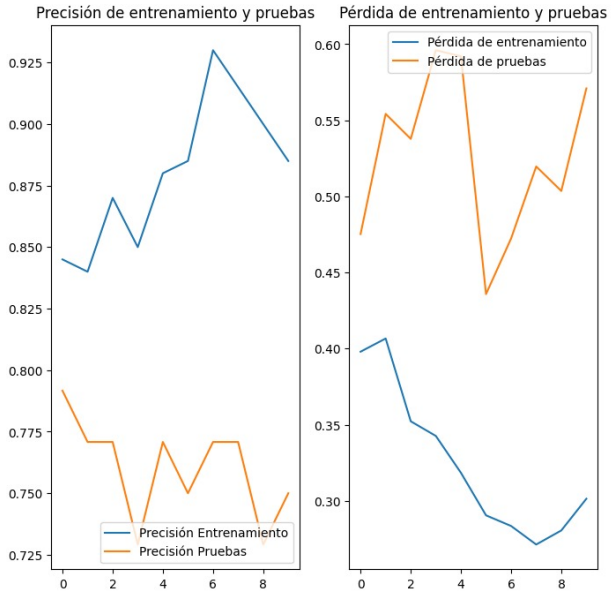


Figura 9. Entrenamiento 1.

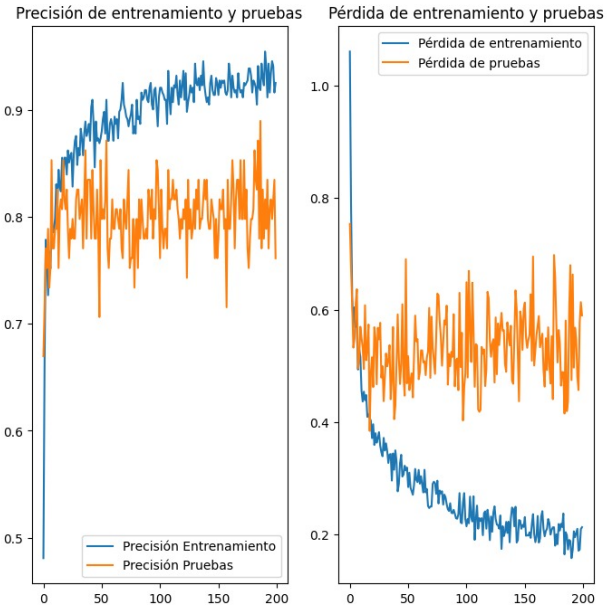


Figura 10. Entrenamiento 2.

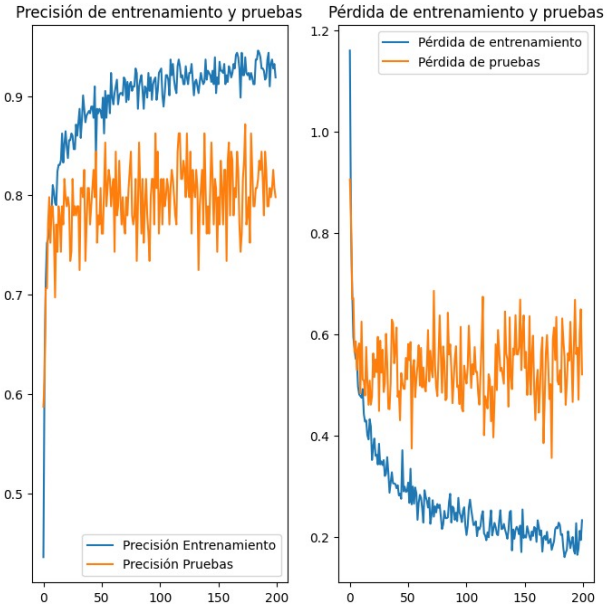


Figura 11. Entrenamiento 3.

15. Por último, se define una función categorizar que descarga una imagen desde una URL, la procesa y luego utiliza el modelo previamente entrenado para predecir la categoría de la imagen.Esta función toma como entrada una URL de una imagen, descarga la imagen y la convierte a un formato compatible con el modelo de red neuronal entrenado. Utiliza requests.get para obtener la imagen desde la URL y BytesIO para manejar el contenido de la imagen en memoria. La imagen se abre con PIL.Image y se convierte a un arreglo numpy normalizado dividiendo por 255. Luego, la imagen se redimensiona a 224x224 píxeles usando OpenCV (cv2.resize) para que coincida con el tamaño de entrada esperado por el modelo. La imagen procesada se pasa al modelo para obtener una predicción de su categoría. Finalmente, la función devuelve la clase con la mayor probabilidad (np.argmax) y el vector completo de predicciones. Esto permite categorizar imágenes nuevas basadas en el modelo entrenado.

## V. Diseño y desarrollo de la API backend

El desarrollo de una API Backend son fundamentales para la creación de aplicaciones escalables. La API Backend sirve como un puente entre el frontend de la aplicación, la parte con la que interactúan los usuarios, y el backend, la parte encargada de procesar los datos y la lógica de la aplicación. En este contexto, el backend desempeña un papel crucial al proporcionar los servicios y la funcionalidad necesarios para que la aplicación funcione de manera eficiente y segura.

**Estructura main.py**

1. Se importan los módulos y clases necesarios de FastAPI y otras bibliotecas como TensorFlow y PIL.
2. ISe configuran las rutas de los archivos estáticos y las plantillas para servir archivos HTML y recursos estáticos como imágenes y estilos.
3. Se configura el middleware CORS para permitir el acceso a la API desde un origen específico (en este caso, la aplicación frontend que se ejecuta en http://127.0.0.1:5501).
4. Se define una ruta para manejar las solicitudes GET en la raíz de la aplicación.
5. Cuando se recibe una solicitud GET en esta ruta, se devuelve la plantilla HTML index.html.
6. Se define una ruta para manejar las solicitudes GET en la raíz de la aplicación.
7. Cuando se recibe una solicitud GET en esta ruta, se devuelve la plantilla HTML index.html.

**Estructura script.js**

1. Se ejecuta cuando el DOM (Document Object Model) está completamente cargado y listo para ser manipulado. Es el punto de entrada principal para el código JavaScript en la página.
2. Se agrega un evento de cambio al elemento de entrada de tipo archivo (<input type=”file” >) con el ID image. Cuando el usuario selecciona una imagen, este evento se activa y ejecuta la función asociada.
3. Se hace uso de FileReader para leer los datos de la imagen seleccionada por el usuario. Cuando se completa la lectura de la imagen, la función onload se activa y ejecuta el código dentro de ella. En este caso, se muestra la imagen seleccionada al usuario.
4. Hay que agregar un evento de envío al formulario con el ID uploadForm. Cuando el usuario envía el formulario (por ejemplo, al hacer clic en el botón de enviar), este evento se activa y ejecuta la función asociada.
5. La imagen se envía como datos del formulario (FormData). Se configura contentType y processData como false para asegurar que la imagen se envíe correctamente como un archivo binario.

6. La función success maneja la respuesta exitosa del servidor, y la función error maneja cualquier error que pueda ocurrir durante la solicitud AJAX.

## VI. Creación de la interfaz gráfica y conexión con el backend

El código del frontend es la parte de una aplicación web que los usuarios interactúan directamente en sus navegadores. Está compuesto por HTML, CSS y JavaScript, que trabajan juntos para crear la interfaz de usuario y brindar una experiencia interactiva y atractiva.

**Estructura index.html**

El archivo index.html sirve como el punto de entrada de la aplicación web, presentando la estructura y el contenido que los usuarios interactúan en sus navegadores. Desde el título de la página hasta los elementos de formulario y áreas de visualización de resultados, cada componente juega un papel crucial en la experiencia del usuario. En él, se enlazan hojas de estilo CSS para definir el aspecto visual, incluyendo la popular biblioteca Bootstrap para un diseño responsivo. Los elementos de formulario permiten a los usuarios seleccionar y enviar imágenes al backend para su procesamiento. El uso de scripts JavaScript, incluyendo la biblioteca jQuery y código personalizado en script.js, facilita la interactividad y la comunicación con el servidor, completando la experiencia del usuario en la aplicación web..

**Estilos CSS**

El archivo styles.css contiene reglas de estilo que definen la apariencia visual de la página web. En él, se establecen propiedades como el tamaño, el color y el espaciado de los elementos HTML para crear un diseño coherente y atractivo. Por ejemplo, las reglas aplicadas al contenedor principal container limitan su ancho y lo centran en la página, mientras que las reglas para el botón de envío btn-primary definen su apariencia y comportamiento al interactuar con él. Además, se proporcionan estilos específicos para el formulario, como el espaciado entre los elementos del formulario, y para el contenedor de resultados, asegurando una presentación visualmente agradable y funcional para los usuarios. Estos estilos son esenciales para mejorar la usabilidad y la estética de la página web, proporcionando una experiencia de usuario mejorada y coherente.

En el prototipo proporcionado, la conexión entre el backend y el frontend se establece a través de solicitudes HTTP. Cuando un usuario interactúa con la interfaz de usuario en el frontend, como seleccionar una imagen y enviarla al servidor, se activa un evento en JavaScript que desencadena una solicitud AJAX al backend. Esta solicitud se procesa en el servidor, donde se lleva a cabo la lógica de negocio, como la detección de emociones en la imagen. Una vez completado el procesamiento, el backend devuelve una respuesta al frontend, que puede contener resultados o mensajes de estado. Esta respuesta se maneja en el frontend mediante una función JavaScript, que actualiza dinámicamente la interfaz de usuario para mostrar los resultados al usuario.

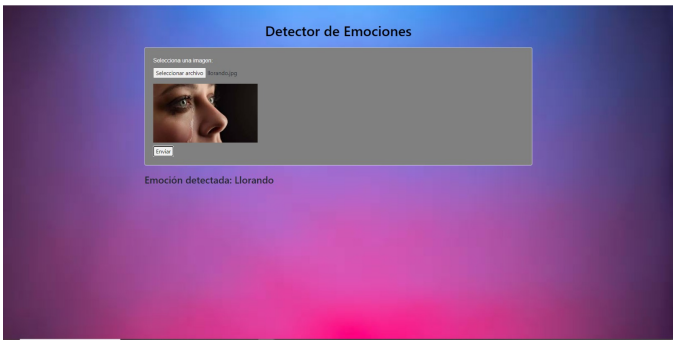


Figura 2. Prototipo Análisis De Imagénes.

## VII. Conclusiones

1. El uso de AJAX en JavaScript permite una experiencia de usuario más interactiva al permitir que el frontend envíe y reciba datos del backend sin necesidad de recargar la página completa. Esto mejora la experiencia del usuario al proporcionar respuestas en tiempo real y una sensación de fluidez en la aplicación.
2. El prototipo demuestra un potencial significativo en la detección de emociones a través del análisis de imágenes. Al integrar tecnologías de inteligencia artificial o machine learning en el backend, la aplicación puede analizar imágenes enviadas por los usuarios y detectar emociones como tristeza, enojo. Esta capacidad puede ser útil en una variedad de aplicaciones, desde análisis de sentimientos en redes sociales hasta sistemas de atención al cliente basados en emociones.
3. La conexión entre el frontend y el backend permite una experiencia del usuario centrada en los resultados. Después de enviar una imagen al backend para su procesamiento, los usuarios reciben una respuesta en tiempo real que muestra los resultados del análisis de emociones. Esta experiencia directa y receptiva puede aumentar la satisfacción del usuario al proporcionar una retroalimentación inmediata sobre la imagen enviada.

## VIII. Referencias

1. Google colab. (s/f). Google.com. Recuperado el 30 de mayo de 2024, dehttps://colab.research.google.com/drive/1N_6xHfa-GsFxMOCpgZD9ViChTFNidO5C#scrollTo=AyLhD03ZB_jf