

Proyecto M7

Bootcamp Ciencia de Datos e Inteligencia Artificial

Pedro Guzmán

Descripción General

- Para este proyecto, se utilizó el dataset de kaggle llamado *Chest X-Ray Images (Pneumonia)*, el cual incluye imágenes de pacientes con y sin Neumonía.
- El propósito de este proyecto es entrenar un modelo, utilizando técnicas de Tuning, para poder predecir si un paciente tiene o no Neumonía.
- Para poder hacer uso de esta modelo, se creó una API REST y una SPA para poder consumir el servicio.
- A continuación se mostrarán segmentos de código de cómo se hizo cada parte.
- Repositorio en Github: <https://github.com/pgzm29/M7Project>

Análisis Exploratorio

- Lo primero que se hizo, fue hacer un análisis exploratorio para poder tener una idea de lo que se tiene que hacer. Al ser un modelo basado en imágenes, no se tuvo que hacer limpieza de datos, pero si se tuvo que hacer una normalización de las imágenes para poder trabajar de una manera correcta.
- El modelo cuenta con 3 folders con imágenes (train, test, val). Dichos folders servirán para entrenar y validar el modelo.
 - En total se tienen 5,216 imágenes para entrenar, 624 de prueba y 16 imágenes para validar.

Preprocesamiento



```
1 train_datagen = ImageDataGenerator(  
2     rescale=1.0/255.0,  
3     rotation_range=20,  
4     width_shift_range=0.2,  
5     height_shift_range=0.2,  
6     shear_range=0.2,  
7     zoom_range=0.2,  
8     horizontal_flip=True,  
9     fill_mode='nearest'  
10 )
```

Parámetros para el generador de datos de las Imágenes



```
1 validation_datagen = ImageDataGenerator(rescale=1.0/255.0)
```



```
1 train_generator = train_datagen.flow_from_directory(  
2     './dataset/train',  
3     target_size=(128, 128),  
4     batch_size=32,  
5     class_mode='binary' # For binary classification  
6 )
```



```
1 test_generator = train_datagen.flow_from_directory(  
2     './dataset/test',  
3     target_size=(128, 128),  
4     batch_size=32,  
5     class_mode='binary' # For binary classification  
6 )
```



```
1 val_generator = validation_datagen.flow_from_directory(  
2     './dataset/val',  
3     target_size=(128, 128),  
4     batch_size=32,  
5     class_mode='binary' # For binary classification  
6 )
```

Lectura de las imágenes, pasando por el generador de datos

Entrenamiento del Modelo

Creación del Modelo (Sin Tuning)

- El modelo se entrenó con Keras de Tensorflow, lo primero fue crear un modelo de manera empírica, seleccionando parámetros que podrían funcionar. La idea de hacer esto es tener una idea de cómo se comporta el modelo para después poder aplicar técnicas de Tuning y tener un modelo con mayor precisión.
- Las capas que se eligieron fueron Conv2D , MaxPooling2D , Flatten , Dense y Dropout.



```
1 from tensorflow.keras.models import Sequential  
2 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```



```
1 model = Sequential([  
2     Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),  
3     MaxPooling2D(2, 2),  
4     Conv2D(64, (3, 3), activation='relu'),  
5     MaxPooling2D(2, 2),  
6     Conv2D(128, (3, 3), activation='relu'),  
7     MaxPooling2D(2, 2),  
8     Flatten(),  
9     Dense(512, activation='relu'),  
10    Dropout(0.5),  
11    Dense(1, activation='sigmoid') # Binary classification  
12 ])
```

- **Conv2D:** Capa con filtros convolucionales que se utilizan para capturar patrones y características locales en la imagen.

- **MaxPooling2D:** Capa que reduce las dimensiones espaciales de los mapas de características.

- **Flatten:** Capa para convertir los mapas de características 2D en un vector 1D para preparar la entrada para la capa Dense.

- **Dense:** Capa para realizar la clasificación final. La función de activación 'relu' introduce no linealidad y captura relaciones más complejas en los datos y la función 'sigmoid' es adecuada para clasificación binaria.

- **Dropout:** Capa que nos ayuda a evitar el overfitting, ya que elimina aleatoriamente una fracción de las neuronas, lo que permite un aprendizaje más específico en relación a las características más robustas.



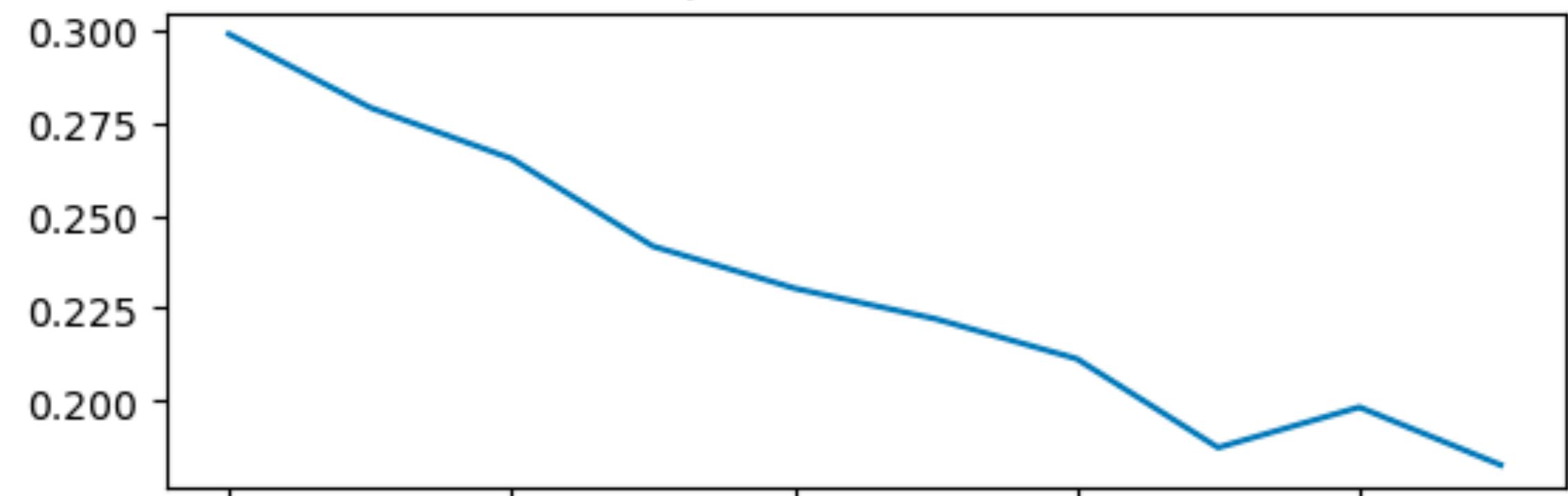
```
1 model.compile(optimizer=Adam(learning_rate=0.001), loss="binary_crossentropy", metrics=["accuracy"])
```



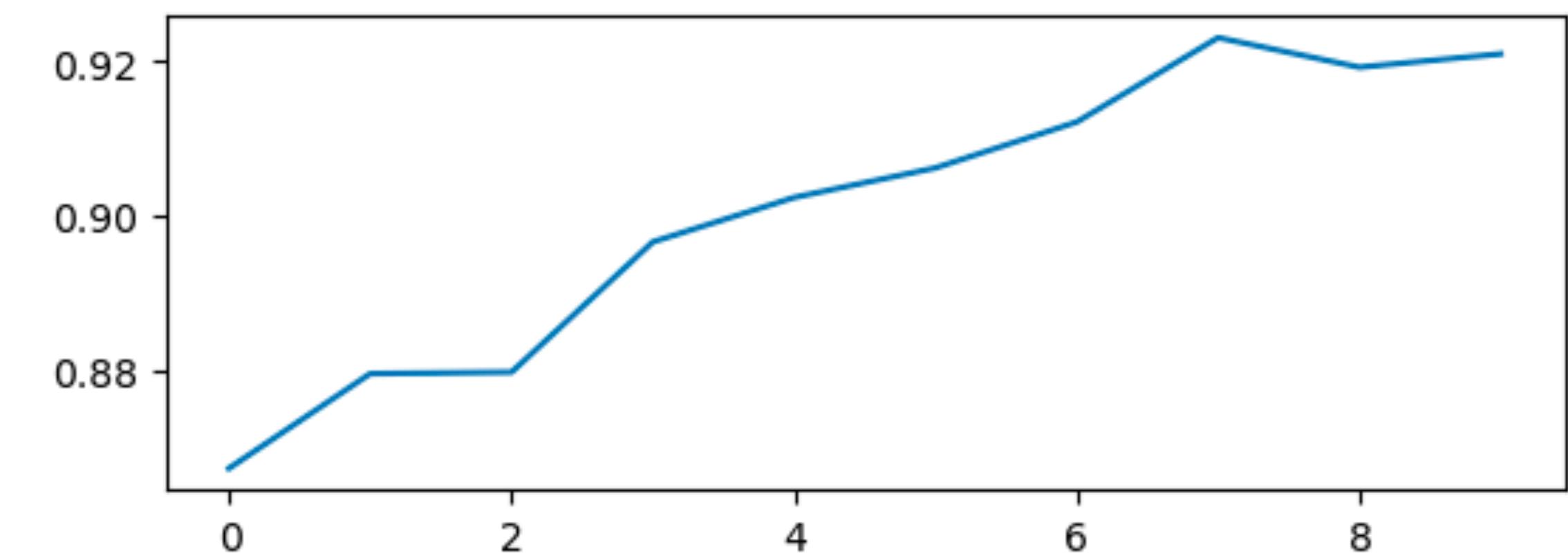
```
1 history = model.fit(  
2     train_generator,  
3     steps_per_epoch=len(train_generator),  
4     epochs=10,  
5     validation_data=val_generator,  
6     validation_steps=len(val_generator)  
7 )
```

Precisión: 0.8285256624221802

Función de pérdida del entrenamiento



Precisión del entrenamiento



Entrenamiento del Modelo

Creación del Modelo (Con Tuning)

- Una vez que se tuvo un modelo con una precisión, ahora pasamos a la creación del modelo pero usando técnicas de Tuning.
- Para lograr esto, se utilizó RandomSearch de Keras y se pusieron rangos de parámetros para que, por medio de diferentes pruebas que se hacen internamente, nos pudiera dar los mejores parámetros para poder entrenar nuestro modelo.

Implementación de RandomSearch



```
1 def build_model(hp: HyperModel):
2     model = tf.keras.models.Sequential()
3
4     # Hyperparameters
5     filters = hp.Int('filters', min_value=32, max_value=128, step=32)
6     kernel_height = hp.Int('kernel_height', min_value=3, max_value=5, step=2)
7     kernel_width = hp.Int('kernel_width', min_value=3, max_value=5, step=2)
8     pool_size = (2, 2)
9     dense_units = hp.Int('dense_units', min_value=128, max_value=256, step=64)
10    dropout_rate = hp.Float('dropout_rate', min_value=0.2, max_value=0.5, step=0.1)
11    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
12
13    model.add(Conv2D(filters, (kernel_height, kernel_width), activation='relu', input_shape=(128, 128, 3)))
14    model.add(MaxPooling2D(pool_size))
15    model.add(Conv2D(filters * 2, (kernel_height, kernel_width), activation='relu'))
16    model.add(MaxPooling2D(pool_size))
17    model.add(Flatten())
18    model.add(Dense(dense_units, activation='relu'))
19    model.add(Dropout(dropout_rate))
20    model.add(Dense(1, activation='sigmoid'))
21
22    model.compile(optimizer=Adam(learning_rate=hp_learning_rate), loss='binary_crossentropy', metrics=['accuracy'])
23
24    return model
```



```
1 tuner = RandomSearch(
2     build_model,
3     objective='val_accuracy',
4     max_trials=10,
5     directory='tuning',
6     project_name='pneumonia_detection'
7 )
```

Ejecución de RandomSearch



```
1 tuner.search(train_generator, validation_data=val_generator, epochs=6)
```



```
1 best_model = tuner.get_best_models(num_models=1)[0]
```

Precisión: 0.8285256624221802

Trial 10 Complete [00h 08m 21s]

val_accuracy: 0.75

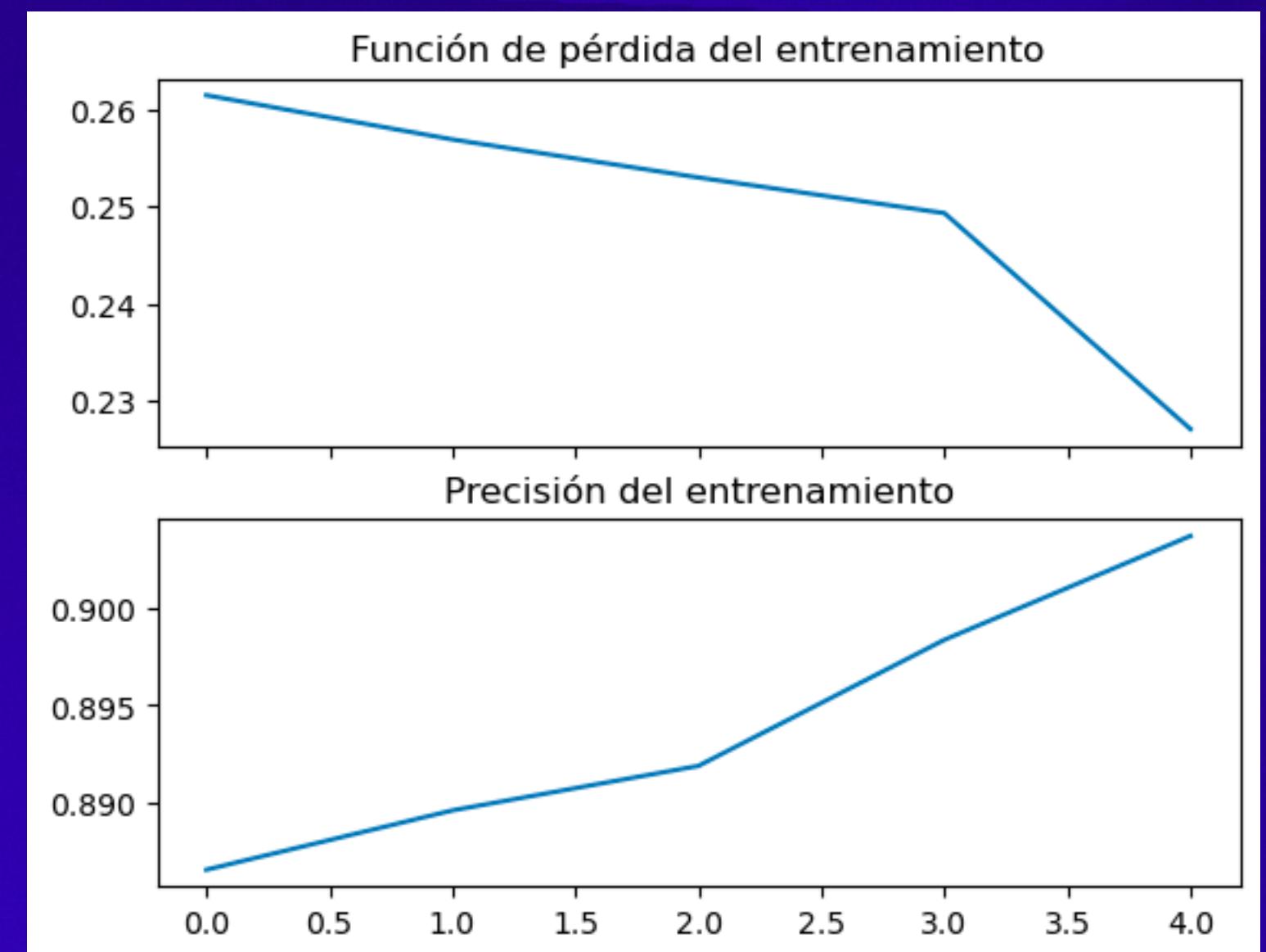
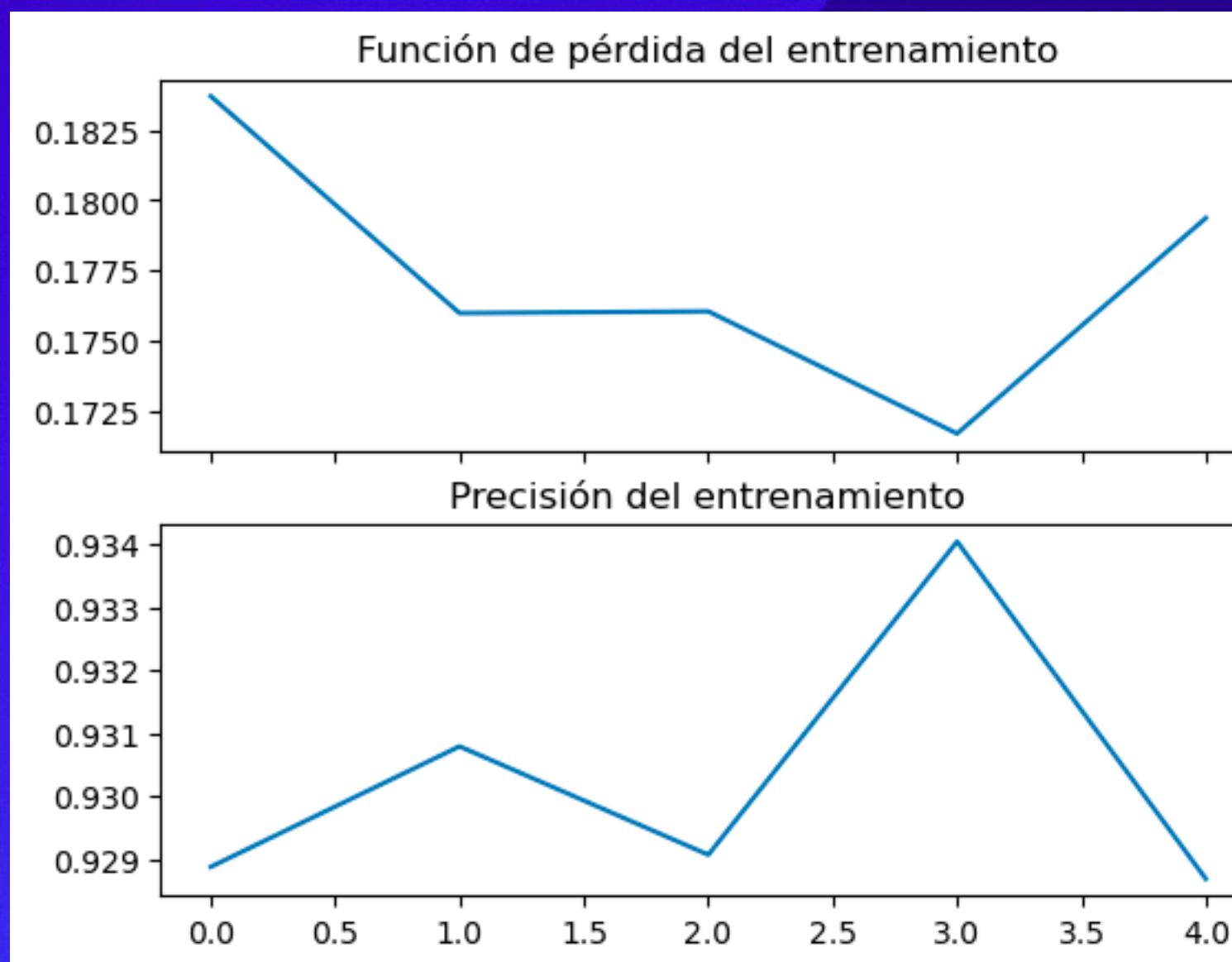
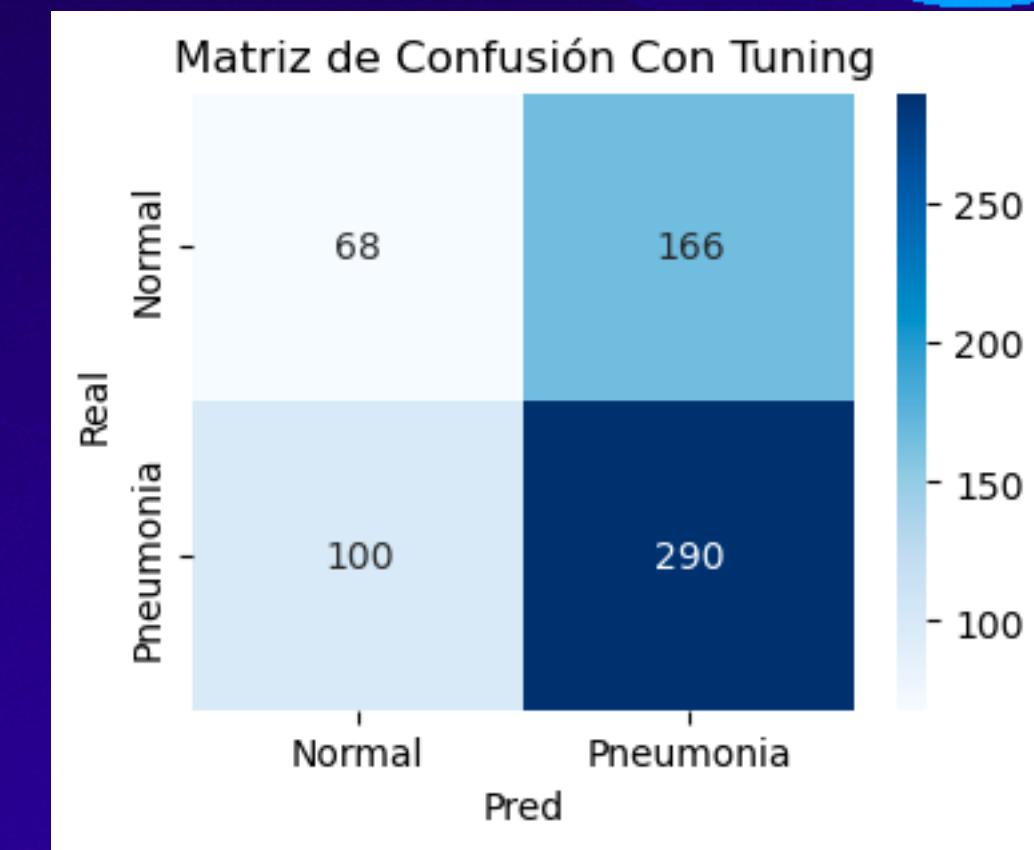
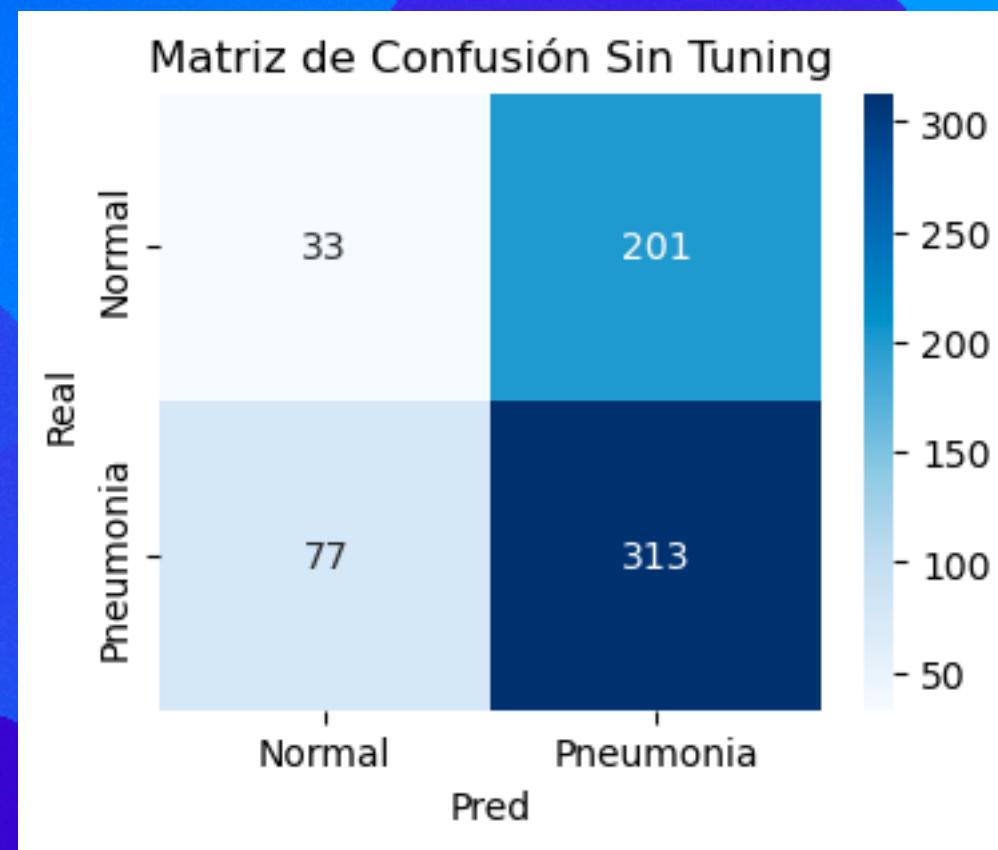
Best val_accuracy So Far: 0.875

Total elapsed time: 03h 53m 52s



```
1 best_model.save('pneumonia_detection_model_tuning.keras')
```

Comparación de Modelos



Predicciones con el Modelo



```
1 def load_random_images(path, n = 5):
2     image_files = [f for f in os.listdir(path) if f.lower().endswith('.jpg', '.jpeg', '.png')]}
3
4 return random.sample(image_files, n)
```

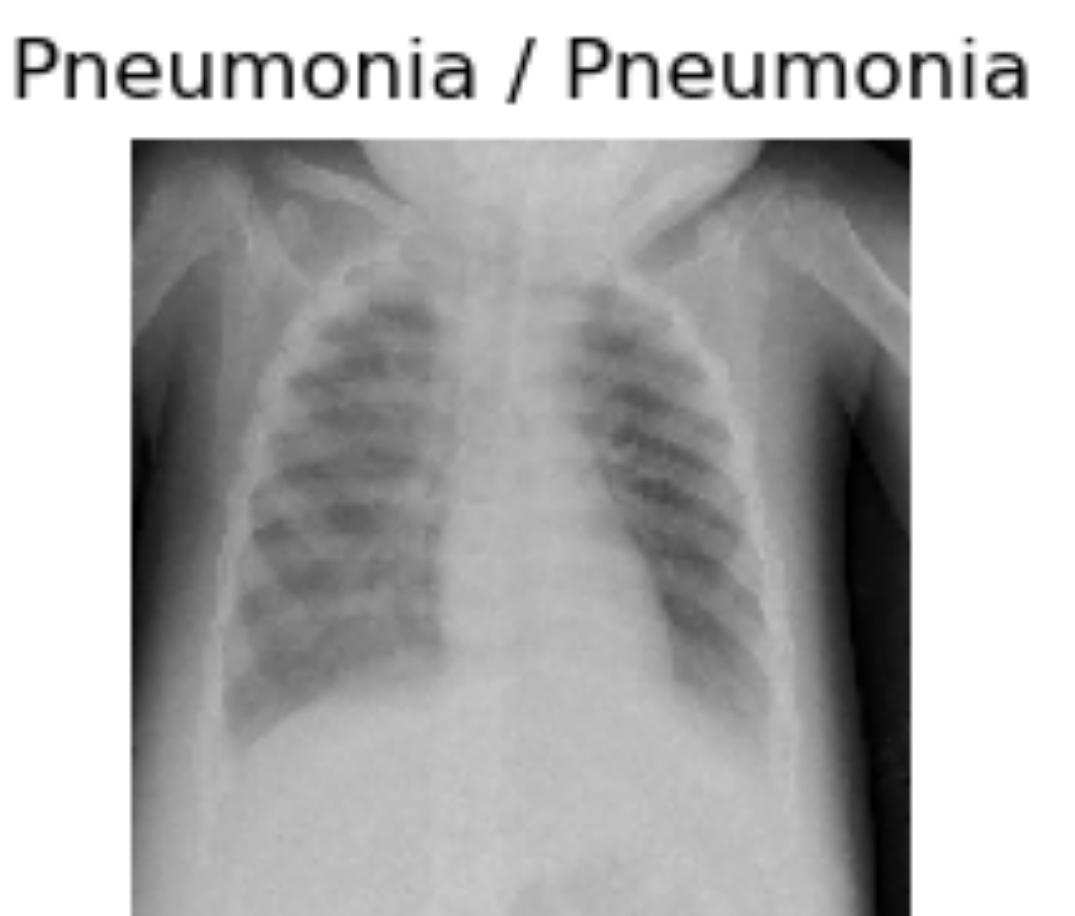
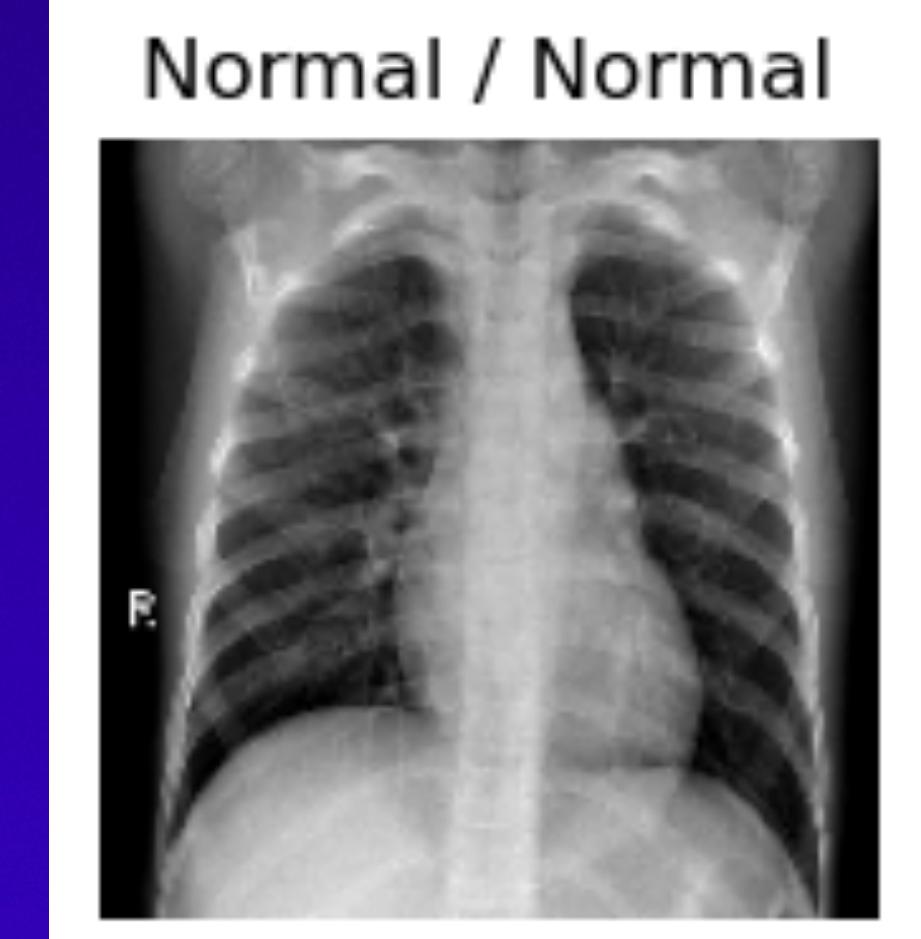


```
1 def predict_on_random_images(model, path, images, target):
2     for image_file in images:
3         # Leemos las imágenes
4         image_path = os.path.join(path, image_file)
5         img = image.load_img(image_path, target_size=(128, 128))
6         img_array = image.img_to_array(img)
7         img_array = np.expand_dims(img_array, axis=0)
8         img_array /= 255.0
9
10        # Realizamos una predicción
11        prediction = model.predict(img_array)
12
13        # Obtenemos el label de la predicción
14        real = 'Pneumonia' if prediction[0][0] >= 0.5 else 'Normal'
15        show_image(img, target, real)
```



```
1 def show_image(image, target, real):
2     plt.figure(figsize=(2,2))
3     plt.imshow(image)
4     plt.title(f'{target} / {real}')
5     plt.axis('off')
6     plt.show()
```

Resultado: Real VS Predicción



API REST & Aplicación Web

- Para poder consumir de una manera más sencilla y amigable para el usuario, se creó una API REST hecha en Python con Flask y para la parte visual, se hizo una SPA (Single Page Application) con Svelte.
- El funcionamiento de esta app consiste en subir cualquier fotografía, y la API, gracias al modelo entrenado y creado, podrá hacer una predicción y enviarla al usuario.
- URL de la App: <https://m7api-dxkxrwg5lq-uc.a.run.app>

Código de la API

- La API tiene un endpoint llamado `predict`, que lo que hace es validar que venga un archivo en la petición. Se hace esto para evitar errores.
- Una vez que se tiene la imagen, se hace una transformación similar a lo que se tuvo que hacer para el entrenamiento del modelo.
- Una vez que la imagen paso por el preprocessamiento, se procede a invocar el modelo que exportamos durante la sección anterior, y hacemos una predicción.
- El resultado de dicha predicción se envía de regreso al usuario.

```
1  @app.route('/api/predict', methods=['POST'])
2  def predict():
3      # Load the trained model
4      model_path = './model.keras'
5      model = tf.keras.models.load_model(model_path, compile=False)
6
7      if 'file' not in request.files:
8          return jsonify({'error': 'No file part'})
9
10     file = request.files['file']
11
12     if file.filename == '':
13         return jsonify({'error': 'No selected file'})
14
15     # Load and preprocess the image
16     img = Image.open(file) # type: ignore
17     img = img.convert('RGB') # Convert to RGB color mode
18     # Resize to match the input size of your model
19     img = img.resize((128, 128))
20     img_array = np.array(img) / 255.0 # Normalize
21
22     # Make a prediction
23     pred = model.predict(np.expand_dims(img_array, axis=0)) # type: ignore
24
25     # Interpret prediction
26     prediction = 'Pneumonia' if pred[0][0] >= 0.5 else 'Normal'
27
28     return jsonify({'prediction': prediction})
```

Código del FrontEnd

● ● ●

```
1 <div class="flex flex-col items-center justify-center md:w-1/2 lg:w-1/3">
2   <div class="flex w-full flex-col items-center justify-center gap-3">
3     <h1 class="text-xl">Modelo de Predicción</h1>
4
5     <label
6       class="mb-2 hidden text-sm font-medium text-stone-900 dark:text-white"
7       for="file_input">Subir Archivo</label>
8   >
9   <input
10    class="block w-full cursor-pointer rounded-lg border border-stone-300 bg-stone-50 text-sm text-stone-900"
11    id="file_input"
12    type="file"
13    accept="image/png, image/jpeg"
14    bind:files
15  />
16  {#if file}
17    <img bind:this={image} src="" alt="Preview" class="rounded-md" />
18  {/if}
19  {#await promise}
20    <p class="text-gray-100">Cargando...</p>
21  {:then result}
22    {#if result.prediction != ''}
23      <div class="text-lg text-green-500">
24        Predicción: {result.prediction}
25      </div>
26    {/if}
27  {:catch}
28    <p class="text-red-600">Algo salió mal</p>
29  {/await}
30  </div>
31 </div>
```

● ● ●

```
1 <script lang="ts">
2   import { axiosClient } from '../api';
3
4   interface PredictionResponse {
5     prediction: string;
6   }
7
8   async function makePrediction(file: File) {
9     const form = new FormData();
10    form.append('file', file);
11    const { data } = await axiosClient.post<PredictionResponse>(
12      '/predict',
13      form,
14    );
15    return data;
16  }
17
18  let promise = Promise.resolve<PredictionResponse>({ prediction: '' });
19  let files: FileList;
20  let file: File;
21  let image: HTMLImageElement;
22
23  $: if (files) {
24    file = files[0];
25    const reader = new FileReader();
26    reader.addEventListener('load', function () {
27      image.setAttribute('src', reader.result as string);
28    });
29    reader.readAsDataURL(file);
30    promise = makePrediction(file);
31  }
32 </script>
```

Despliegue de la App

- Para poder subir la aplicación a la nube, la primera opción era Python Anywhere, pero debido al tamaño del archivo del modelo, no podía subirlo.
- Se optó por subirla a Google Cloud como una aplicación Docker.
- Uno de los errores que ocurrían al correr la aplicación en un ambiente de Cloud, era que por el tamaño del modelo, GIT hacía algo raro con el archivo y la API lo tomaba como corrupto, así que se tuvo que hacer un cambio en cómo se obtenía dicho archivo en el código de Flask.

```
● ● ●  
1 FROM python:3.11-slim  
2  
3 RUN apt-get update  
4 RUN apt-get install -y gcc  
5 RUN apt-get install -y curl  
6  
7 ENV PYTHONUNBUFFERED True  
8  
9 ENV APP_HOME /app  
10 WORKDIR $APP_HOME  
11 COPY . ./  
12  
13 RUN pip install --no-cache-dir -r requirements.txt  
14  
15 CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 --timeout 0 app:app
```

```
● ● ●  
1 if not os.path.isfile('model.keras'):  
2     subprocess.run(  
3         ['curl --output model.keras --location "https://.../model.keras"', shell=True])
```

Aplicación Web

