Detección de cáncer de pulmón con un enfoque basado en aprendizaje automático y aprendizaje profundo

Proyecto final del Samsung Innovation Campus 2024

Abarca Cruz Zdenko Emilio

Aburto Lopéz Francisco Javier

Fuentes Herrera Carlos

Ramírez Rodríguez Enrique

Supervisor: José Indalecio Ríos

Supervisor: Marco Antonio Macías Cedeño

Proyecto

México, Abril 2025

Índice

1. Introducción

El presente proyecto se desarrolla como parte del curso de **Samsung Innovation Campus** sobre **Inteligencia Artificial y Liderazgo**. El objetivo es aplicar los conocimientos adquiridos en el diseño e implementación de modelos de IA en una problemática específica, generando información de utilidad y explorando la capacidad de la inteligencia artificial para resolver problemas reales o crear soluciones innovadoras.

La **temática elegida** fue el cáncer de pulmón, una de las principales causas de muerte en el mundo (**WHO2025**). Su diagnóstico temprano es fundamental, ya que permite un tratamiento más efectivo y reduce los costos médicos al evitar intervenciones complejas en etapas avanzadas. Sin embargo, la detección temprana presenta desafíos significativos, como la falta de acceso a especialistas en algunas regiones y la complejidad de interpretar imágenes médicas (**OMS2017diagnosis**).

Según **MNT2025<empty citation>**, el cáncer de pulmón puede causar varios síntomas que pueden indicar que existe un problema en los pulmones. Entre los síntomas más comunes se encuentran los siguientes:

- Tos persistente
- Dolor torácico
- Disnea
- Tos con sangre (hemoptisis)
- Cansancio
- Pérdida de peso inexplicada
- Infecciones pulmonares recurrentes

El pronóstico del cáncer de pulmón depende de múltiples factores como el tipo de cáncer, el estadio en el momento del diagnóstico, la edad y salud general del paciente. Según datos de la Sociedad Americana Contra el Cáncer(**ACS2025**), la tasa de supervivencia a 5 años varía significativamente dependiendo del tipo de cáncer y de la etapa en la que se diagnostique, siendo mayor cuando se detecta en etapas tempranas.

Etapa	CPCNP	CPCP
Localizado(aún en la ubicación original)	63 por ciento	27 por ciento
Regional(Se ha extendido a zonas cercanas)	35 por ciento	16 por ciento
Distante o metastásica(Se ha diseminado por todo el cuerpo)	7 por ciento	3 por ciento
General	25 por ciento	7 por ciento

Supervivencia según etapa y tipo de cáncer de pulmón.

El diagnóstico del cáncer de pulmón se basa en una combinación de estudios que incluyen exploraciones físicas, imágenes médicas (radiografías, tomografías computarizadas y resonancias magnéticas), broncoscopias, biopsias y pruebas moleculares para identificar mutaciones genéticas específicas. Estos procedimientos permiten determinar la etapa del cáncer y guiar el tratamiento.

1.1. Problematica

El cáncer de pulmón es la principal causa de muertes relacionadas con el cáncer a nivel mundial, siendo responsable de aproximadamente el 85 % de los casos debido al tabaquismo(**WHO2025**). Este cáncer se diagnostica frecuentemente en etapas avanzadas, lo que limita las opciones de tratamiento y reduce las probabilidades de supervivencia.

Además, la detección oportuna se ve obstaculizada por la falta de acceso a tecnologías avanzadas de diagnóstico, la carencia de especialistas en áreas rurales y la complejidad de interpretar imágenes médicas de manera precisa. Esto resalta la necesidad de desarrollar herramientas que faciliten el diagnóstico temprano y preciso, disminuyendo la mortalidad asociada al cáncer de pulmón(**mayoclinic˙lung˙cancer**).

Los tipos más comunes de cáncer de pulmón son el carcinoma no microcrítico (NSCLC) y el carcinoma microcrítico (SCLC). El NSCLC es más común y crece lentamente, mientras que el SCLC es menos común pero de crecimiento rápido(**WHO2025**).

1.2. Datos a Nivel Mundial

Según la Sociedad Americana Contra el Cáncer(**cancer˙american**), para el año 2025 se estiman los siguientes datos en Estados Unidos:

- 226,650 nuevos casos de cáncer de pulmón.
- 124,730 muertes por cáncer de pulmón.

A nivel mundial, el cáncer de pulmón representa el 12.4 % de todos los diagnósticos de cáncer, con aproximadamente 2.21 millones de casos nuevos diagnosticados en 2022. Además, es la principal causa de muerte por cáncer, responsable del 18.7 % de las muertes por cáncer, con alrededor de 1.8 millones de muertes anuales. Este cáncer afecta principalmente a personas mayores de 65 años y se asocia principalmente con factores de riesgo como el tabaquismo, que es responsable del 85 % de los casos, así como la exposición a sustancias tóxicas y la contaminación ambiental (**anticancerlifestyle**; **cancerorg**; **who**)

1.3. Datos en México

En México, el cáncer de pulmón es la causa más letal entre los tipos de cáncer, con más de 8,000 muertes al año (**INCan2025**). En 2018 se reportaron más de 9,000 nuevos casos, de los cuales el 85 % estaban relacionados con el tabaquismo. A pesar de los esfuerzos por mejorar el diagnóstico, la detección en etapas avanzadas sigue siendo un problema.

1.4. Justificación

Este proyecto se desarrolla como parte del curso de Samsung Innovation Campus sobre Inteligencia Artificial y Liderazgo. Se busca aplicar los conocimientos adquiridos en el diseño e implementación de modelos de IA en una problemática específica que sea capaz de generar información de utilidad.

A través de este trabajo, se busca aplicar conceptos teóricos y prácticos aprendidos durante el curso, explorando la capacidad de los modelos de IA para resolver problemas reales o crear soluciones innovadoras.

La selección de la temática fue completamente libre, el tema que decidimos a tratar fue cáncer de pulmón, debido a que es una de las principales causas de muerte en el mundo, con una alta tasa de mortalidad y su detección temprana podría salvar miles de vidas al permitir tratamientos más efectivos y a su vez reducir los costos de tratamiento, esto contribuiría a reducir la mortalidad asociada a la enfermedad y a optimizar los recursos médicos, ofreciendo una herramienta accesible y de gran impacto social.

Este proyecto no solo permitirá aplicar conocimientos técnicos, sino que también representa una oportunidad para explorar el impacto positivo de la IA en el campo de la salud, alineándose con los objetivos del Samsung Innovation Campus de fomentar el desarrollo de habilidades tecnológicas con un enfoque práctico y transformador.

1.5. Objetivo General

Desarrollar un modelo de inteligencia artificial capaz de diagnosticar el cáncer de pulmón de manera temprana y precisa, contribuyendo a reducir la tasa de mortalidad y optimizando el tratamiento médico.

1.6. Objetivos Específicos

- Desarrollar un modelo de IA capaz de detectar el cáncer de pulmón en etapas tempranas, facilitando el diagnóstico oportuno y mejorando las probabilidades de tratamiento efectivo.
- Búsqueda de bases de datos open source.
- Análisis sistemático de literatura
- Desarrollar un proceso estandarizado para la extracción y procesamiento de datos relevantes de las bases de datos seleccionadas.
- Aplicar conceptos teóricos y prácticos aprendidos durante el curso, explorando la capacidad de los modelos de IA.
- Identificar patrones complejos y sutiles en imágenes médicas mediante técnicas avanzadas de procesamiento de imágenes.

1.7. Revisión de Literatura: Aplicaciones de la Inteligencia Artificial en la Predicción y Diagnóstico del Cáncer de Pulmón

La inteligencia artificial (IA) ha emergido como una herramienta revolucionaria en el campo de la salud, permitiendo diagnósticos más precisos, pronósticos personalizados y tratamientos optimizados para diversas enfermedades, incluido el cáncer de pulmón. A continuación, se presentan algunas de las aplicaciones más destacadas de la IA en el manejo del cáncer de pulmón:

- **Redes Neuronales Convolucionales (CNNs):**En el estudio de Hatuwal y Thapa (2020) titulado "Lung Cancer Detection Using Convolutional Neural Network on Histopathological Images", publicado en el International Journal of Computer Trends and Technology(**CNN1**), se utilizó una red neuronal convolucional (CNN) para detectar el cáncer de pulmón a partir de imágenes histopatológicas. Este tipo de imágenes son esenciales en el diagnóstico del cáncer, ya que proporcionan detalles a nivel celular sobre los tejidos afectados, se pueden ver los resultados en la sección conclusión.

  El articulo "Deep Learning for Computer Vision: A Brief Review"proporciona una revisión general sobre cómo el aprendizaje profundo (Deep Learning) ha revolucionado el campo de la visión por computadora (Computer Vision). En particular, cubre los avances en redes neuronales convolucionales (CNNs) y otras arquitecturas de redes profundas que se han aplicado exitosamente en tareas como clasificación de imágenes, detección de objetos, y segmentación de imágenes, entre otras, los resultados se pueden observar en la conclusión (**CNN2**).

- **XGBClassifier**El artículo "Lung and colon cancer classification using medical imaging: a feature engineering approach",(2022) presenta un sistema asistido por computadora para clasificar de manera precisa tejidos de cáncer de colon y pulmón utilizando imágenes histopatológicas. El estudio aplica técnicas de aprendizaje automático y procesamiento de imágenes, utilizando modelos como XGBoost (**modelo˙XGBClassifier**), se pueden ver los resultados del estudio en la sección conclusión.

Una de las aplicaciones más innovadoras de la IA es el análisis histopatológico de imágenes de biopsias teñidas con hematoxilina y eosina (H&E). Esta técnica ha permitido a los sistemas de IA identificar y cuantificar patrones complejos que serían difíciles de detectar manualmente. Un estudio publicado en *Nature Communications* presentó **SEQUOIA**, una herramienta de IA capaz de predecir la actividad de miles de genes en células tumorales utilizando solo imágenes de biopsias teñidas con H&E. Los resultados mostraron una correlación superior al 80 % entre las predicciones de la IA y los datos reales de actividad genética en tipos específicos de cáncer (**labmedica2025**).

El análisis histopatológico automatizado no solo permite detectar células tumorales, sino que también cuantifica la densidad celular y calcula índices de proliferación, lo que facilita la estratificación de pacientes según el riesgo y la personalización de tratamientos oncológicos. Esta tecnología puede clasificar imágenes con una precisión sorprendente, lo que agiliza la toma de decisiones clínicas y reduce el tiempo diagnóstico (**roche2025**).

Además, el enfoque interdisciplinario de la IA y las biopsias digitales ha abierto nuevas oportunidades para la investigación clínica, permitiendo explorar correlaciones entre patrones histológicos y resultados clínicos. Esto no solo mejora el diagnóstico y el pronóstico del cáncer de pulmón, sino que también proporciona un marco más robusto para desarrollar nuevas terapias dirigidas y medicina personalizada (**labmedica2025**).

En resumen, la integración de la inteligencia artificial en el diagnóstico y tratamiento del cáncer de pulmón ha permitido avances significativos en la detección temprana, el pronóstico preciso y la planificación terapéutica personalizada. Estos enfoques innovadores continúan evolucionando, prometiendo un futuro en el que la IA desempeñe un papel cada vez más crucial en la lucha contra el cáncer de pulmón.

1.8. Entrevista con el Dr. Alberto Daniel Saucedo Campos - ISSTE

Se realizó una entrevista con el Dr. Alberto Daniel Saucedo Campos, especialista en oncología del ISSSTE, para conocer su perspectiva sobre el diagnóstico y seguimiento de cancer de pulmón mediante herramientas de inteligencia artificial con el objetivo de conocer su perspectiva sobre la detección temprana de neoplasias mediante el uso de modelos basados en inteligencia artificial.

Preguntas y respuestas:

- **¿Cuáles son las principales dificultades para diagnosticar cancer de pulmón?** *El problema del cáncer de pulmón es que en sus etapas iniciales no da síntomas, no duele, no provoca sangrados a veces puede provocar una tos muy leve que se puede confundir con una alergia, una gripa, infortunadamente el cáncer de pulmón hasta que llega a niveles de invasión del bronquio más intenso es cuando se comienza con la disnea y mínimo puede que ya este en la etapa 3. Entonces ese es el problema principalmente, que las neoplasias son muy silenciosas*

- **¿Qué tipo de errores diagnósticos son más comunes en la práctica clínica?** *El cáncer es una enfermedad muy catastrófica, si vemos la cantidad de diabéticos, de hipertensos que tenemos vs los que tienen cáncer, pues la proporción que tenemos es de 30 a 1, por cada 30 diabéticos va a haber uno con cáncer, entonces infortunadamente el clínico no piensa de primera intención en cáncer, hasta que les hacen la placa, y típicamente cuando hacen la placa encuentran una cosa que se llama nódulo solitario, es cuando comienzan a sospechar de que puede ser una neoplasia. Actualmente lo que se está haciendo es buscar marcadores moleculares en sangre cuando haya la sospecha, y pues ahí en lo que se enfoca principalmente es en la historia clínica, si el paciente es fumador, si está expuesto a biomasas, es decir, que tenga contacto con anafres, con polvos intensos, humos, ya que es importante en la historia clinica.*
- **¿Qué papel juega el médico de primer contacto en el diagnóstico inicial?** *El médico de primer contacto juega un papel importante ya que debe realizar un diagnóstico preliminar basado en los síntomas del paciente y muchas veces no tienen la preparación para sospecharlo y cuando lo sospechan no saben qué buscar.*
- **¿Cómo podría un sistema predictivo mejorar el diagnóstico?** *Un modelo de predicción con inteligencia artificial puede identificar factores de riesgo y calcular la probabilidad de enfermedad antes de que los síntomas se presenten, permitiendo un seguimiento médico más eficiente.*
- **¿Cómo ve el uso de modelos predictivos para seguimiento de pacientes?** *Podrían reducir costos, optimizar recursos y permitir un monitoreo constante, especialmente en pacientes de alto riesgo.*
- **¿Qué aspectos clave deben considerarse al implementar un sistema de predicción de enfermedades?** *La precisión del modelo, la accesibilidad y el acompañamiento médico para interpretar los resultados.*

En la entrevista, se discutieron las principales dificultades para diagnosticar el cáncer de pulmón, enfatizando que la enfermedad suele ser asintomática en etapas tempranas, lo que dificulta una detección oportuna. Frecuentemente, los síntomas iniciales son leves y se confunden con afecciones menores, lo que retrasa el diagnóstico hasta etapas avanzadas.

El doctor destacó que en la práctica clínica, el diagnóstico temprano del cáncer de pulmón es complejo debido a que los médicos de primer contacto no suelen considerarlo inicialmente. Este error diagnóstico es común, ya que se priorizan enfermedades más frecuentes como la diabetes o la hipertensión. Por lo general, el cáncer de pulmón solo se sospecha tras realizar una radiografía que revela un nódulo solitario. En respuesta a esta problemática, se mencionó que actualmente se buscan marcadores moleculares en sangre para una detección más precisa cuando hay sospechas fundadas, además de realizar una historia clínica exhaustiva enfocada en factores de riesgo como el tabaquismo y la exposición a biomasas y polvos intensos.

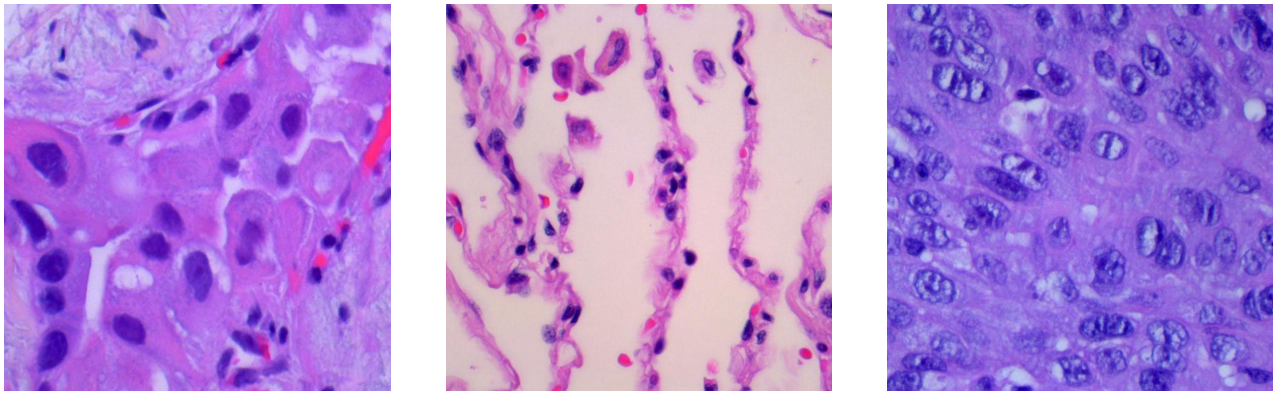
Respecto al papel del médico de primer contacto, el doctor señaló que su intervención es fundamental para un diagnóstico temprano, aunque la falta de preparación específica limita su capacidad de identificar la enfermedad.

Finalmente, el doctor expresó que un modelo de predicción basado en inteligencia artificial (IA) podría transformar el diagnóstico y seguimiento de pacientes con cáncer de pulmón al calcular el riesgo antes de la aparición de síntomas. Este enfoque permitiría optimizar recursos, reducir costos y ofrecer un monitoreo constante a pacientes de alto riesgo. No obstante, enfatizó que la precisión del modelo, la accesibilidad y el acompañamiento médico adecuado son aspectos críticos para la implementación exitosa de esta tecnología

2. Metodología

2.1. Metodología de clasificación

Para el apartado de modelos de clasificación se utilizó un "*dataset*" público (**500images**), llamado *"Lung and Colon Cancer Histopathological Images"*, el cual consta de 25,000 imágenes histopatológicas de cáncer de pulmón y de colón. Siendo nuestro interés el cáncer de pulmón, se trabajó únicamente con las imágenes de este tipo de cáncer, las cuales se encuentran divididas en 3 carpetas *"lung_aca"*, *"lung_n"* y *"lung_scc"*, correspondientes a adenocarcinoma, tejido benigno y carcinoma escamocelular respectivamente, cada una de ellas con 5,000 imágenes.



Adenocarcinoma | Benigno | Carcinoma

Ejemplo de imagen de cada clase

Las versiones de *Python*, *pip*, *wheel* y librerías utilizadas, fueron las siguientes:

- TensorFlow versión: 2.10.0
- NumPy versión: 1.24.4
- OpenCV versión: 4.11.0
- Matplotlib versión: 3.7.5
- Seaborn versión: 0.13.2
- Keras versión: 2.10.0
- Sklearn versión: 1.3.2
- Pip versión: 25.0.1
- Python versión: 3.8.0
- Wheel versión: 0.45.1

Adicionalmente, para los modelos que lo permitían, se utilizó CUDA y cuDNN durante el entrenamiento para acelerar el proceso. Las versiones utilizadas fueron las siguientes:

- CUDA versión: 11.8
- cuDNN versión: 8.6.0

El objetivo en este apartado fue crear modelos que dada una imagen histopatológica de un cáncer de pulmón, pudieran determinar el tipo de tumor entre los 3 mencionados previamente.

Se trabajaron con 2 *"datasets"* diferentes, uno para los modelos existentes en la librería *sklearn*, y otro para los modelos que se crearon con el apoyo de la librería *tensorflow* y *keras*. Esto se debe a que para los modelos creados con *tensorflow* se aprovecharon los métodos y funciones que se encuentran en esta librería, resultando en objetos que podrían causar problemas de compatibilidad o simplemente no se podrían usar con los modelos pertenecientes a la librería *sklearn*. Para el *dataset* utilizado en los modelos de la librería *sklearn* se utilizaron las librerías *cv2* ,*os* y de la librería *sklearn.model_selection* se importo la clase *train_test_split* para la creación

de las variables usuales *X_train*, *X_test*, *Y_train* y *Y_test*, a continuación se muestran los códigos de la definición de estos 2 *datasets* comentados, así como una breve descripción de ambos:

```python
def creacion_dataset(folder):

    imagenes = [] # Creacion array de imagenes
    etiquetas = [] # Creacion array de etiquetas
    etiquetas_clases = os.listdir(folder) # Definicion de etiquetas a partir de nombres de carpetas

    for etiqueta in etiquetas_clases:

        class_path = os.path.join(folder, etiqueta) # Define la ruta de la clase iterada
        if not os.path.isdir(class_path): # Verifica que el folder existe
            continue

        for img_name in os.listdir(class_path):
            img_path = os.path.join(class_path, img_name) # Define la ruta de la imagen iterada
            img = cv2.imread(img_path) # Lee la imagen iterada
            if img is not None: # Comprueba que la imagen no es nula
                img = cv2.resize(img, (64, 64)) # Redimensiona la imagen
                img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convierte a escala de grises
                img = img.astype("float32") / 255.0 # "Normaliza" la imagen
                imagenes.append(img.flatten()) # Convierte el arreglo a una dimension y lo agrega al arrelgo imagenes
                etiquetas.append(etiqueta) # Agrega la etiqueta al arreglo de etiquetas

    return np.array(imagenes), np.array(etiquetas) # Retorna los arreglos de imagenes y etiquetas

X, Y = creacion_dataset('Direccion carpetas de imagenes') # Llamado de la funcion

le = LabelEncoder() # Se crea el objeto
Y = le.fit_transform(Y) # Se utiliza el metodo fit_transform para reetiquetar el conjunto de etiquetas

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42) # Creacion conjuntos de entrenamiento y de test

print(f'Tamano de train: {len(X_train)}, Tamano de test: {len(X_test)}') # Imprime el tamano de los conjuntos de entrenamiento y de test de las imagenes
print(f'Clases codificadas: {le.classes_}')  # Verifica las clases transformadas
```

Código creación *dataset* para los modelos de *sklearn*

```
Tamano de train: 12000, Tamano de test: 3000
Clases codificadas: ['lung_adenocarcinoma' 'lung_benigno' 'lung_carcinoma']
```

Salida del código

Como se dijo previamente esta celda utiliza las librerías *cv2* y *os* para la creación o definición de los conjuntos de entrenamiento y de test. Se utilizó una función que se le da como argumento una cadena con la dirección de la carpeta donde se encuentran las 15,000 imágenes que ya se encuentran separadas en 3 carpetas cada una con 5,000 imágenes con los nombres de las clases. Adicionalmente se importó de la librería *sklearn.preprocessing* la clase *LabelEncoder*, que se utilizó para convertir las etiquetas de tipo *string* a tipo entero, pues algunos modelos no aceptan etiquetas en formato de cadena.
Para el preprocesamiento de las imágenes, estas se redimensionaron a un arreglo de 64 x 64, se transformaron a escala de grises pues se buscó evitar un sesgo en el tono de las tinciones entre las diferentes clases. Es decir, se buscó evitar que los modelos simplemente aprendieran a separar entre clases por el tono y no por las características de las imágenes. También se "normalizaron" las imágenes dividiendo el valor de cada *pixel* entre 255 para tener únicamente valores entre 0 y 1, y por último se "aplano" el arreglo de 2 dimensiones a un arreglo de 1 dimensión.

Para el dataset utilizado en los modelos de *tensorflow* se utilizó el siguiente código para crear los conjuntos de validación y de entrenamiento:

```python
# Definicion de parametros
dataset_dir = "Direccion carpetas de imagenes" # Define la direccion de la carpeta de imagenes
batch_size = 32 # Define el tamano de batch
img_size = (224, 224)  # Ajusta a las dimensiones deseadas
validation_split = 0.2  # Define el tamano del conjunto de validacion
seed = 123  # Fijamos una semilla de aleatoriedad

# Carga del dataset para entrenamiento
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_dir,
    validation_split = validation_split,
    subset = "training",
    seed = seed,
    image_size = img_size,
    batch_size = batch_size,
    label_mode = "categorical", # Definimos las etiquetas como categoricas
    shuffle = True # Barajea los registros
)

# Carga del dataset para validacion
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_dir,
    validation_split = validation_split,
    subset = "validation",
    seed = seed,
    image_size = img_size,
    batch_size = batch_size,
    label_mode = 'categorical', # Definimos las etiquetas como categoricas
    shuffle = True # Barajea los registros
)

# Verificar etiquetas asignadas
class_names = train_ds.class_names  # Deberia mostrar ['lung_adenocarcinoma', 'lung_benigno',
    'lung_carcinoma']
print("Clases asignadas:", class_names)

# Verificar tamanos de los datasets
train_size = tf.data.experimental.cardinality(train_ds).numpy()
val_size = tf.data.experimental.cardinality(val_ds).numpy()

print(f"Entrenamiento: {train_size} batches")
print(f"Validacion: {val_size} batches")

# Funcion para preprocesar las imagenes
def normalize_img(image, label):
    image = tf.image.rgb_to_grayscale(image) # Convierte a escala de grises
    # image = tf.image.grayscale_to_rgb(image) # Esta linea se comenta o no segun la red
    image = tf.cast(image, tf.float32) / 255.0 # "Normaliza" las imagenes
    return image, label

# Aplicar el preprocesamiento a los datasets
train_ds = train_ds.map(normalize_img)
val_ds = val_ds.map(normalize_img)
```

Código creación *dataset* para los modelos de *tensorflow*

```
Found 15000 files belonging to 3 classes.
Using 12000 files for training.
Found 15000 files belonging to 3 classes.
Using 3000 files for validation.
TClases asignadas: ['lung_adenocarcinoma', 'lung_benigno', 'lung_carcinoma']
Entrenamiento: 375 batches
Validacion: 94 batches
```

Salida del código

Para la creación de los datasets de entrenamiento y de validación para los modelos creados con *tensorflow*, se utilizó únicamente la clase *image* de la librería *tensorflow.keras.preprocessing*. Al inicio del código se declarán los parámetros a utilizar para evitar reescribirlos, se declara la dirección de la carpeta que contiene las 3 carpetas con las 15,000 imágenes.

Se declara el tamaño de batch que se utilizará en el entrenamiento, esto es importante pues ya se podra redefinir el tamaño de batch al momento de entrenar los modelos pues esto podría crear incongruencias.
Se declara el tamaño de imagen con el que deseamos trabajar y también la semilla de aleatoriedad para poder replicar los resultados. Cabe destacar que el tamaño de la imagen se conservo tan grande pues estos modelos fueron entrenados con el apoyo de *CUDA* por lo que se pudo trabajar con una mayor cantidad de datos.
Posteriormente se crean los *datasets* de entrenamiento y de validación utilizando el método *tf.keras.preprocessing.image_dataset_ from_directory* al cual se le dió los parámetros anteriormente definidos y en adición se definió el parametro *label_mode* como "*categorical*" pues se trata de una clasificación multiclase y el parámetro *suffle* se declaró como *True* para "barajear" las imágenes.
Como se puede ver en el código lo que distingue ambos *datasets* no es únicamente el nombre si no también el valor del parámetro *subset*, para el *dataset* de entrenamiento se declara como *"training"* y para el *dataset* de validación se declara como *"validation"*.
Posteriormente se hace una breve comprobación de las clases asignadas así como de la cantidad de *batches* en el conjunto de entrenamiento y en el de validación.
Por último se declara una función de preprocesamiento de las imágenes la cual convierte las imágenes a tono de grises y las "normaliza" dividiendo cada pixel entre 255 para tener únicamente valores de 0 a 1. Existe una linea en la función que se comenta o no dependiendo del modelo, esto es porque se utilizaron técnicas de *Transfer Learning* y de *Fine Tuning*, por lo que algunos modelos que se usaron de base están diseñados para trabajar con imágenes RGB o de 3 canales. Al transformar las imágenes a escala de grises, reducimos esos 3 canales a 1, provocando una incompatibilidad entre los datos y las dimensiones de entrada de los modelos. Por lo que a veces es necesario "descomentar" esta linea para convertir la imagen de vuelta a 3 canales, aunque realmente sigue estando en blanco y negro.
Luego de declarada la función se le aplica a los conjuntos por medio del metodo *map* el cual toma como argumento la función a aplicar al objeto que lo llama.

2.1.1. Support Vector Machine

Gracias a la disponibilidad de tiempo y de poder computacional disponible del equipo, se tuvo la oportunidad de usar la clase *GridSearchCV* importada de *sklearn.model_selection* en lugar de la clase *RandomizedSearchCV* para buscar los mejores parámetros de un *Support Vector Classifier* que se encuentra en la librería *sklearn.svm*. A continuación la celda de código comentado, así como su descripción:

```
svm_model = SVC() # Creacion del objeto Support Vector Classifier

# Definicion de parametros a probar
param_grid = {'C': [0.1, 1, 10, 100],
              "kernel":["linear", "rbf", "poly"],
              'gamma': ['scale', 'auto', 0.01, 0.1, 1, 10, 100]}

# Creacion objeto GridSearchCV con parametros definidos y con 5 CV
grid_search = GridSearchCV(svm_model, param_grid, cv=5, verbose=2, n_jobs=4)
# Entrenamiento de posibles modelos
grid_search.fit(X_train, Y_train)
```

Código *GridSearchCV* para *SVC*

Como se explicó, en este código se utilizó la clase *GridSearchCV* por lo que al comienzo se crea un objeto tipo *Support Vector Classifier* para usarlo como parámetro del *GridSearchCV*. Como parámetros adicionales a los ya definidos se declaró *verbose* igual a 2, para poder monitorear constantemente el progreso del modelo y *n_jobs* igual a 4 para aprovechar los núcleos del procesador sin sobrecalentarlo. Para este modelo se consideraron diferentes valores para *C*, para el *kernel* y para *gamma*, así como 5 *cross-validation's*. Lo que generaba 84 posibles modelos cada uno con 5 *fits* por la validación cruzada, dándonos un total de 420 modelos a entrenar. Esta es una considerable cantidad de modelos de *SVC* y teniéndo en cuenta que suelen tardar en entrenarse por su complejidad, el tiempo estimado de entrenamiento fue alto, pero permisible dados lo recursos del equipo. Por último se aplicó el método *fit* al modelo y se dieron como parámetros los conjuntos creados para los modelos de la librería *sklearn*.

2.1.2. Regresión Logística

Al ser un modelo de menor complejidad que un *SVM*, se utilizó nuevamente la clase *GridSearchCV* de la librería *sklearn.model_selection* y se importó la clase *LogisticRegression* de la librería *sklearn.linear_model*. A continuación la celda de código comentado, así como su descripción:

```
log_reg = LogisticRegression(max_iter=3000) # Creacion del objeto Logistic Regression

# Definicion de parametros a probar
param_grid = {"C": [0.01, 0.1, 1, 10, 100],
              "penalty": ["l2", "none"],
              "solver":["liblinear", "lbfgs", "saga"]}

# Creacion objeto GridSearchCV con parametros definidos y con 5 CV
grid_search_lrm = GridSearchCV(log_reg, param_grid, cv=5)
# Entrenamiento de posibles modelos
grid_search_lrm.fit(X_train, Y_train)
```

Código *GridSearchCV* para *LogisticRegression*

Se creó al comienzo del código un objeto del tipo *LogisticRegression* al cual se le dió como parámetro *max_iter* el entero 3000, este parámetro determina el número máximo de iteraciones para que el modelo converja, se eligió un número alto de iteraciones ya que los recursos del equipo lo permitían. Adicionalmente a los parámetros a iterar, se declararon 5 *cross-validation's*, para este modelo no se creyó necesario definir los parámetros *n_jobs* y *verbose* al ser un modelo con poca complejidad. Para este modelo se consideraron diferentes valores de *C*, de *penalty* y de *solver*. Esto generó 30 posibles modelos cada uno con 5 *fits* por la validación cruzada, resultando en un total de 150 modelos a entrenar. Por último se le aplicó el método *fit* al modelo y se dieron como parámetros los conjuntos creados para los modelos de la librería *sklearn*.

2.1.3. XGBClassifier

Como último modelo ajeno a *tensorflow* se utilizó la clase *GridSearchCV* para encontrar los mejores parámetros del modelo *XGBClassifier*, el cual se importó de la librería xgboost. Se aprovechó la plataforma de desarrollo CUDA para entrenar este modelo en una GPU, mejorando considerablemente el tiempo de entrenamiento. A continuación la celda de código comentado, así como su descripción:

```
modelo_XGB = XGBClassifier(device="cuda", random_state=42)

param_grid = {
    'n_estimators': [100, 200],  # Numero de arboles
    'learning_rate': [0.01, 0.1],  # Tasa de aprendizaje
    'max_depth': [3, 6, 10],  # Profundidad del arbol
    'subsample': [0.7, 1],  # Proporcion de muestras usadas en cada arbol
    'colsample_bytree': [0.7, 1],  # Proporcion de caracteristicas usadas en cada arbol
    'gamma': [0, 0.1],  # Poda del arbol (reduccion de sobreajuste)
    'reg_alpha': [0, 0.1, 1],  # Regularizacion L1
    'reg_lambda': [1, 10, 100],  # Regularizacion L2
    "device": ["cuda"]  # Habilitar GPU
}

# Creacion objeto GridSearchCV con parametros definidos y con 5 CV
grid_search_XGB = GridSearchCV(modelo_XGB, param_grid, cv=5, verbose=2)
# Entrenamiento de posibles modelos
grid_search_XGB.fit(X_train, Y_train)
```

Código *GridSearchCV* para *XGBClassifier*

Como en los anteriores modelos, se crea primero un objeto del tipo del modelo que deseamos entrenar, en este caso *XGBClassifier*, para usarlo como parámetro de *GridSearchCV*. Se definió como parámetro de *device*, *cuda*, para aprovechar la GPU disponible del equipo y como *random_seed* se eligió el entero 42 por si se deseará replicar el modelo. Los parámetros a iterar fueron *n_estimators* el número de arboles, *learning_rate* tasa de aprendizaje, *max_depth* máxima profundidad del árbol, *subsample* cantidad de datos usados, *colsample_bytree* cantidad de datos usados por árbol, *gamma* poda del árbol, *reg_alpha* regularización L1, *reg_lambda* regularización L2 y de manera redundante se volvió a definir el parámetro *device* como *cuda*.

Se creó el objeto GridSearchCV, y se le dió como parámetros el modelo XGBClassifier, los parámetros que se definieron, el parámetro *cross-validation* se definió con el entero 5 y el parámetro *verbose* se definió con 2 para poder monitorear el modelo. Dado la cantidad de parámetros a probar se tienen 864 candidatos, a cada uno se le aplican 5 *fits* por la validación cruzada, siendo el total, 4320 *fits*.

2.1.4. Transfer Learning con MobileNetV2, VGG16 y ResNetRS101

Se trabajarón 3 modelos usando la técnica de *Transfer Learning*, se usarón como modelo base para cada uno, *MobileNetV2*, *ResNetRS101*, *VGG16*. Se eligió *MobileNetV2* por ser un modelo pequeño y rápido, a pesar de que su uso es en sistemas de visión por computadora en tiempo real. El modelo *VGG16* es un modelo grande, lento y pesado computacionalmente usado para la clasificación de imágenes, por lo que se consideró una buena opción para la problemática presente. Por último se utilizo el modelo *MobileNetV2* el cual es un modelo muy grande y lento pero con una muy alta precisión. Este modelo suele ser utilizado para algunas aplicaciones médicas.

Estos 3 modelos de *Transfer Learning* usaron el *dataset* que se preparo con la librería *tensorflow*. Sus códigos de definición o creación del modelo son prácticamente idénticos excepto por una sola linea de código, a continuación el código comentado de la definición de estas redes y su descripción:

```
# Se define la variable del modelo base dependiendo cual se usara, sin incluir las capas de salida y con las dimensiones de entrada que se utilizaran

# Modelo base con MobileNetV2
base_model = MobileNetV2(weights="imagenet", include_top = False, input_shape = (224, 224, 3))

# Modelo base con VGG16
base_model = VGG16(weights="imagenet", include_top = False, input_shape = (224, 224, 3))

# Modelo base con ResNetRS101
base_model = ResNetRS101(weights="imagenet", include_top = False, input_shape = (224, 224, 3))


# Se congelan los pesos preentrenados del modelo base
for layer in base_model.layers:
    layer.trainable = False

# Se agregan capas de salida

# Se agrega una capa de GlobalAveragePooling2D para reducir el numero de parametros
x = layers.GlobalAveragePooling2D()(base_model.output)

# Se agregan dos capas densas de 1024 neuronas con funcion de activacion ReLu
# Se agregan dos capas Dropout para reducir el sobre ajuste
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.5)(x)

# Se agrega una capa densa de 3 neuronas con funcion de activacion softmax como capa de salida
x = layers.Dense(3, activation="softmax")(x)

# Se crea y se compila el modelo usando el modelo base elegido y las capas extras
model = Model(inputs=base_model.input, outputs=x)
model.compile(optimizer=Adam(learning_rate=1e-4), metrics=['accuracy'], loss="
    categorical_crossentropy")
```

Código creación CNN con *Transfer Learning* con diferentes modelos base

Al comienzo del código se definió el modelo base dependiendo cual de los 3 candidatos se usaría, esto comentando y descomentando las 3 diferentes definiciones de la variable *base_model*. Estos modelos se encuentran alojados en la librería *tensorflow.keras.applications*. Después se cargaron los pesos de *ImageNet* para cualquiera de los 3, no se utilizaron las capas de salida, pues se usaron capas de salidas propias y se definió las dimensiones de los datos de entrada que fueron definidos en la creación de los conjuntos de entrenamiento y de validación. Posteriormente se "congelaron" los pesos del modelo base pues se buscó evitar un sobre ajuste y aprovechar los pesos originales del modelo base seleccionado.

Adicionalmente al modelo base elegido, se agregaron capas extras de salida: una capa de *GlobalAverage-Pooling2D* conectada a la salida del modelo base, en seguida una capa densa, es decir, una capa totalmente conectada, de 1024 neuronas con función de activación "relu" para disminuir la complejidad computacional del modelo. Después, una capa de *Dropout* con una probabilidad del 50 % para evitar un sobreajuste y dos capas idénticas a las anteriores para hacer un total de 5 capas. Como capa final se agrego una capa densa de 3 neuronas con función de activación *softmax*, esto debido a que se trata de una clasificación multiclase de 3 clases, de ser una clasificación binaria, se hubiera usado la función de activación sigmoide.

Al final del código se definió la variable *modelo* usando la función *Model* a la cual se le dio como parámetros *inputs* y *outputs*, el modelo base seleccionado y las capas de salidas definidas anteriormente.
Y por último se utilizó el método *compile* para compilar el modelo usando como parámetros de *optimizer*, *metrics* y *loss*, el optimizador *Adam* con una tasa de aprendizaje de 0.001, la métrica *accuracy* para evaluar el desempeño del modelo en cada época y la función de pérdida *categorical_crossentropy* pues se trata de un modelo de clasificación multiclase.

Se utilizó el método *summary* para tener una idea del contenido, dimensiones y parámetros de los modelos. Debido a la cantidad de capas y la relevancia para el presente proyecto, se exhibirán únicamente la cantidad de parámetros entrenables, no entrenables y cantidad total de cada uno de estos modelos:

```
Total params: 4,622,403
Trainable params: 2,364,419
Non-trainable params: 2,257,984
```

Salida del método summary aplicado a la red de *Transfer Learning* con *MobileNetV2*

```
Total params: 16,292,675
Trainable params: 1,577,987
Non-trainable params: 14,714,688
```

Salida del método summary aplicado a la red de *Transfer Learning* con *VGG16*

```
Total params: 64,826,147
Trainable params: 3,150,851
Non-trainable params: 61,675,296
```

Salida del método summary aplicado a la red de *Transfer Learning* con *ResNetRS101*

Posteriormente se procedió al entrenamiento del modelo, este proceso fue exactamente idéntico para los 3 modelos propuestos, a continuación el código comentado y su descripción:

```
# Declaracion parametros EarlyStopping
early_stopping = EarlyStopping(
    monitor='val_loss',       # Monitorea la perdida de validacion
    patience=5,               # Si no mejora en 5 epocas consecutivas, se detiene
    restore_best_weights=True # Restaura los mejores pesos durante el entrenamiento
)

# Se aprovecha CUDA y se entrena con la GPU de la computadora
with tf.device('/GPU:0'):
    history = model.fit(train_ds,
                        # Se usa el 20% del conjunto de validacion
                        validation_data=val_ds.take(int(tf.data.experimental.cardinality(
                            val_ds).numpy() * 0.2)),
                        epochs=50, # Se entrena el modelo 50 epocas
                        callbacks=[early_stopping]) # Se define el early stopping
```

Código entrenamiento de la CNN con *Transfer Learning* sin importar el modelo base

Esta celda de código tiene como finalidad el entrenamiento del modelo bautizado *model*, el cual puede tener como modelo base cualquiera de los 3 modelos candidatos. Al comienzo se definieron los parámetros del *EarlyStopping*, el cual monitorea la pedida de validación, tiene una paciencia de 5 épocas y se le solicita resturar los pesos con mejor *accuracy* del entrenamiento. Como con modelos anteriores se aprovechó la plataforma de desarrollo *CUDA* como lo refleja el código. Se creó una variable *history* para el entrenamiento con el propósito de poder visualizar el desempeño del modelo a través de las épocas. Utilizando el método *fit* se entrenó el modelo y se le dio como parámetros, el conjunto de entrenamiento, el 20 % del conjunto de validación para evitar un sobreajuste, 50 épocas de entrenamiento y los parámetros del *EarlyStopping*.

2.1.5. Fine Tuning con ResNetRS101

Dados los resultados de los modelos de *Transfer Learning*, los cuales se discutirán más adelante, se decidió usar la técnica de Fine Tuning usando como modelo base únicamente el modelo *ResNetRS101*. A continuación, el código comentado y su descripción:

```
# Se define la variable base_model usando ResNetRs101, sin incluir las capas de salida y con las dimensiones de entrada que se utilizaran
base_model = ResNetRS101(weights="imagenet", include_top = False, input_shape = (224, 224, 3))

# Se congelan los pesos preentrenados del modelo base
for layer in base_model.layers:
    layer.trainable = False

# Se descongelan las ultimas 20 capas del modelo base
for layer in base_model.layers[-20:]:
    layer.trainable = True

# Se agregan capas de salida

# Se agrega una capa de GlobalAveragePooling2D para reducir el numero de parametros
x = layers.GlobalAveragePooling2D()(base_model.output)

# Se agregan dos capas densas de 1024 neuronas con funcion de activacion ReLu
# Se agregan dos capas Dropout para reducir el sobre ajuste
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.5)(x)

# Se agrega una capa densa de 3 neuronas con funcion de activacion softmax como capa de salida
x = layers.Dense(3, activation="softmax")(x)

# Se crea y se compila el modelo usando el modelo base y las capas extras de salida
model = Model(inputs=base_model.input, outputs=x)
model.compile(optimizer=Adam(learning_rate=1e-4), metrics=['accuracy'], loss="categorical_crossentropy")
```

Código creación CNN con *Fine Tuning* con *ResNetRS101*

Se definió al comienzo del código la variable *base_model* usando el modelo base *ResNetRS101* alojado en *tensorflow.keras.applications*. Se cargaron los pesos de ImageNet, no se cargaron las capas de salida y se definieron las dimensiones de los datos de entrada. Eso es idéntico a cuando se utilizó la técnica de *Transfer Learning*.

Como en la técnica de *Transfer Learning* se "congelaron" todos los pesos del modelo base para evitar un sobreajuste. Después, para realizar un *Fine Tuning* se "descongelaron" las últimas 20 capas del modelo; esto tuvo como objetivo un mayor aprendizaje del modelo.

Posteriormente, exactamente como con los modelos de *Transfer Learning*, se agregaron capas extras de salida: una capa de *GlobalAveragePooling2D* conectada a la salida del modelo base, seguida de una capa densa de 1024 neuronas con función de activación "relu", seguida de una capa de *Dropout* con una probabilidad del 50 %, seguidas de una capa densa y una capa *Dropout* idénticas a las primeras. Como capa final del modelo se conectó una capa densa de 3 neuronas con función de activación *softmax* al tratarse de una clasificación multiclase.

Para finalizar, nuevamente como en los modelos de *Transfer Learning*, se creó la variable *model* con la función *Model* a la cual se le dio como parámetros *inputs* y *outputs*, el modelo base y las capas extra respectivamente. Por último, se compiló el modelo con el optimizador *Adam* con una tasa de aprendizaje de 0.0001, como medida de desempeño del modelo la medida *accuracy* y como función de perdida se le dio *categorical_crossentropy* al tratarse de una clasificación multiclase de 3 clases.

Como con los modelos de *Transfer Learning* se utilizó el método *summary* para tener una idea del tamaño, dimensiones y cantidad de parámetros. Nuevamente, por motivos de cantidad de capas y relevancia para el presente proyecto solo se mostrará la información de los parámetros entrenables, no entrenables y el total de parámetros:

```
Total params: 64,826,147
Trainable params: 11,812,867
Non-trainable params: 53,013,280
```

Salida del método summary aplicado a la red de *Fine Tuning* con *ResNetRS101*

El entrenamiento del modelo se hizo de manera idéntica a los anteriores con la siguiente celda de código:

```
# Declaracion parametros EarlyStopping
early_stopping = EarlyStopping(
    monitor='val_loss',       # Monitorea la perdida de validacion
    patience=5,               # Si no mejora en 5 epocas consecutivas, se detiene
    restore_best_weights=True # Restaura los mejores pesos durante el entrenamiento
)

# Se aprovecha CUDA y se entrena con la GPU de la computadora
with tf.device('/GPU:0'):
    history = model.fit(train_ds,
                        # Se usa el 20% del conjunto de validacion
                        validation_data=val_ds.take(int(tf.data.experimental.cardinality(
                            val_ds).numpy() * 0.2)),
                        epochs=50, # Se entrena el modelo 50 epocas
                        callbacks=[early_stopping]) # Se define el early stopping
```

Código entrenamiento de la CNN con *Fine Tuning*

2.1.6. Propuesta de CNN

Como último modelo propuesto en el apartado de clasificación y adicionalmente a los modelos con técnicas de *Transfer Learning* y *Fine Tuning*, se propuso una Red Neuronal Convolucional de diseño propio. A continuación el código de su diseño comentado y su descripción:

```
# Propuesta de CNN

# Se define un modelo secuencial, las capas se agregan una tras la otra
model = models.Sequential([

    # Primer conjunto de capas
    # Capa convolucional con 64 filtros de 3x3 con funcion de activacion relu, esta capa es
        particular porque es la primera, por lo que tambien se definen las dimensiones de
        entrada
    layers.Conv2D(filters=64, kernel_size=(3,3), activation="relu", input_shape=(224, 224, 1))
        ,
    # Capa MaxPooling2D reduce las dimensiones de la imagen a la mitad
    layers.MaxPooling2D((2,2)),

    # Segundo conjunto de capas
    # Capa convolucional de 128 filtros de 3x3 con funcion de activacion relu
    layers.Conv2D(filters=128, kernel_size=(3,3), activation="relu"),
    # Capa MaxPooling2D reduce las dimensiones de la imagen a la mitad
    layers.MaxPooling2D((2,2)),

    # Tercer conjunto de capas
    # Capa convolucional de 256 filtros de 3x3 con funcion de activacion relu
    layers.Conv2D(filters=256, kernel_size=(3,3), activation="relu"),
    # Capa MaxPooling2D reduce las dimensiones de la imagen a la mitad
    layers.MaxPooling2D((2,2)),

    # Capas de salida
    # Aplana la salida de 2D a un vector de 1D
    layers.Flatten(),
    # Capa densa de 128 neuronas con funcion de activacion relu y una capa dropout con 0.4 de
        probabilidad
    layers.Dense(128, activation="relu"),
    layers.Dropout(0.4),
    # Capa densa de 64 neuronas con funcion de activacion relu y una capa dropout con 0.4 de
        probabilidad
    layers.Dense(64, activation="relu"),
    layers.Dropout(0.4),
    # Capa de salida densa con 3 neuronas con funcion de activacion softmax
    layers.Dense(3, activation="softmax")
])

# Compilacion del modelo
model.compile(optimizer=Adam(learning_rate=1e-4), metrics=["accuracy"], loss="
    categorical_crossentropy")
```

Código diseño propio de CNN

El diseño propuesto se trata de una CNN al usar capas convolucionales. Se declaró al inicio del código la variable *model* con ayuda de la función *models.Sequential*. Se le dio como parámetros las diferentes capas que conformarían este modelo, se comenzó con una capa convolucional de 64 filtros de tamaño 3x3 con función de activación *relu*, esta capa tenía la particularidad de ser la capa de entrada por lo que tambien se le dio las dimensiones de entrada de este modelo. Al ser un modelo de diseño propio no fue necesario "convertir" los conjuntos de validación y entrenamiento a 3 canales como con los modelos de *Transfer Learning* y *Fine Tuning*, por lo que se conservaron las dimensiones originales de las imágenes, es decir, escala de grises o un solo canal. Seguida a esta capa convolucional se agregó una capa *MaxPooling2D* para reducir las dimensiones de la imagen a la mitad.

El segundo conjunto de capas siguió un diseño similar, teniendo en seguida otra capa convolucional de 128 filtros de dimensiones 3x3 con función de activación *relu* y una capa *MaxPooling2D* para reducir nuevamente las dimensiones de la imagen a la mitad. El tercer conjunto de capas es idéntico al anterior con la única diferencia de que en lugar de 128 filtros, la capa convolucional cuenta con 256 filtros.

El conjunto de capas de salida comienza con una capa *Flatten* que convierte la salida 2D de la última capa convolucional a un vector de 1D para conectar con las capas densas. En seguida, se conectó una capa densa de 128 neuronas con función de activación *relu* y una capa *Dropout* con una probabilidad del 40 % de apagar neuronas para evitar un sobreajuste. Siguiendo a estas dos se conectaron dos capas idénticas con la diferencia de que la capa densa en lugar de tener 128 neuronas tiene 64. Para la última capa, la capa de salida, se conectó una capa densa de 3 neuronas con función de activación *softmax* pues se trata de una clasificación multiclase de 3 clases.

Al final del código, se compila el modelo con el método *compile*. Se le dio como parámetros, *oprtimizer*, *metrics* y *loss*, el optimizador *Adam* con una tasa de aprendizaje de 0.0001, *accuracy* como métrica de desempeño del modelo y la función de pérdida *categorical_crossentropy* al tratarse de una clasificación multiclase.

Como con todos los modelos anteriores, se volvió a usar el método *summary* en el modelo propuesto. A continuación, la cantidad de parámetros que muestra el método:

```
Total params: 22,529,411
Trainable params: 22,529,411
Non-trainable params: 0
```

Salida del método summary aplicado a la CNN propuesta

El entrenamiento fue idéntico al de los otros modelos de redes neuronales, utilizándose exactamente el mismo código que con los anteriores modelos:

```
# Declaracion parametros EarlyStopping
early_stopping = EarlyStopping(
    monitor='val_loss',       # Monitorea la perdida de validacion
    patience=5,               # Si no mejora en 5 epocas consecutivas, se detiene
    restore_best_weights=True # Restaura los mejores pesos durante el entrenamiento
)

# Se aprovecha CUDA y se entrena con la GPU de la computadora
with tf.device('/GPU:0'):
    history = model.fit(train_ds,
                        # Se usa el 20% del conjunto de validacion
                        validation_data=val_ds.take(int(tf.data.experimental.cardinality(
                            val_ds).numpy() * 0.2)),
                        epochs=50, # Se entrena el modelo 50 epocas
                        callbacks=[early_stopping]) # Se define el early stopping
```

Código entrenamiento de la CNN propuesta

2.2. Metodología de predicción

La presente sección tiene como objetivo predecir la cantidad de muertes por cáncer de pulmón en México utilizando dos enfoques distintos: **Regresión Lineal** y una **Red Neuronal Recurrente (LSTM)**. A través de esta comparación, se busca evaluar la precisión de ambos modelos y determinar cuál ofrece mejores resultados para la proyección de datos futuros.

Las versiones de *Python* fueron las siguientes:

- TensorFlow versión: 2.10.0
- NumPy versión: 1.26.4
- Matplotlib versión: 3.7.1
- Sklearn versión: 1.3.2
- Python versión: 3.8.0

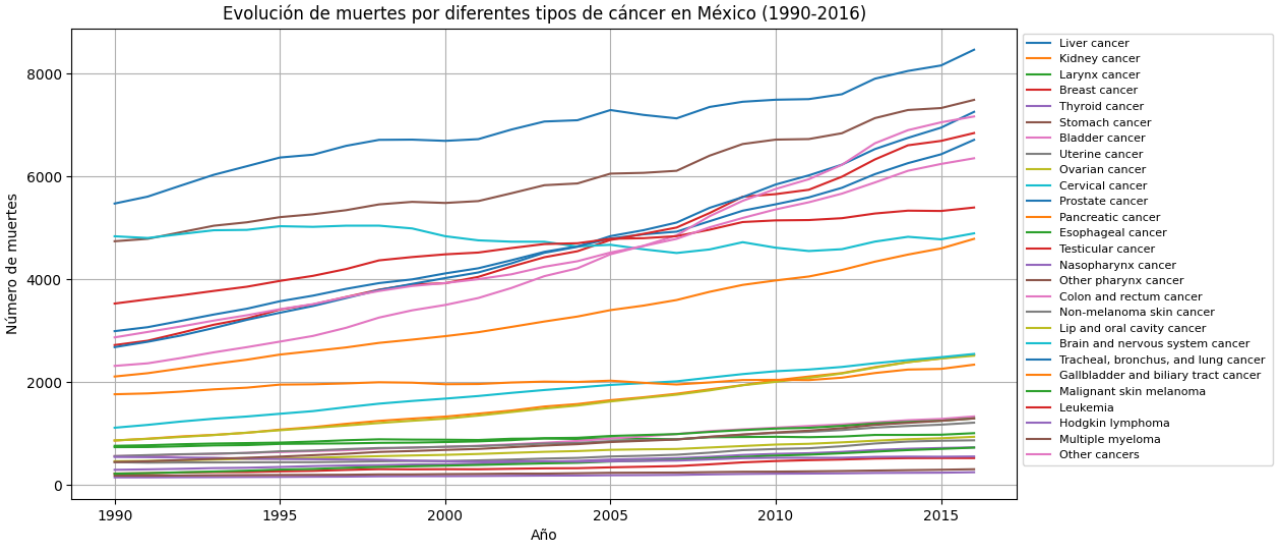
Para el análisis, se utilizó un conjunto de datos que contiene el registro anual de muertes por una gran cantidad de tipos de cáncer en todo el mundo desde 1990 hasta 2016. Este dataset fue obtenido de la plataforma Kaggle, específicamente del conjunto de datos: *"Cancer Deaths by Country and Type (1990-2016)"*.

Dicho dataset recopila la cantidad de muertes por distintos tipos de cáncer en múltiples países. Sin embargo, dado que este estudio se enfoca exclusivamente en México, se realizó una preprocesamiento de los datos para filtrar únicamente la información relevante al país. Esta versión depurada del dataset nos permite analizar la cantidad de muertes causadas por algún tipo de cáncer que ha habido en el país de 1990 al 2016.

Year	Liver cancer	Kidney cancer	Larynx cancer	...
1990	2995.515782	867.859970	765.007794	...
1991	3072.586220	903.779729	776.570274	...
1992	3194.398432	947.317928	796.109315	...
1993	3318.471538	977.590436	809.788569	...
1994	3431.584846	1018.196197	818.597311	...
...	...	...	...	...

Ejemplo del dataset Cancer deaths mexico

Como punto de partida en el análisis exploratorio, se presenta un gráfico que muestra la evolución de las muertes por diferentes tipos de cáncer en México. Esto permite visualizar las diferencias en la mortalidad a lo largo del tiempo y comprender la relevancia del cáncer de pulmón en el contexto general.



Comparación de muertes entre diferentes tipos de cáncer en México

Dado que el enfoque de este estudio es el cáncer de pulmón, se llevó a cabo una segunda fase de preprocesamiento para extraer exclusivamente los registros relacionados con esta enfermedad, modificando el dataset y dejando únicamente los anos y las cantidades de muertes.

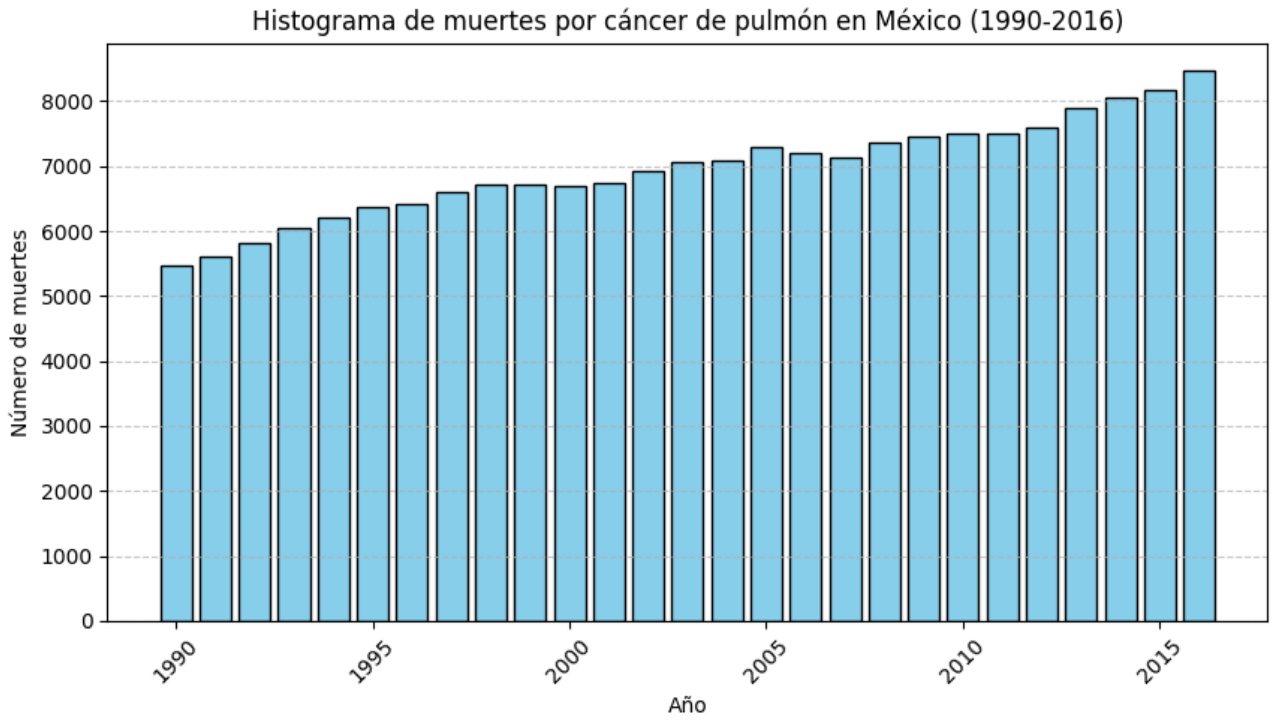
Year	Tracheal, bronchus, and lung cancer
1990	5477.606751
1991	5612.037783
1992	5826.280440
1993	6036.566757
1994	6203.391357
1995	6372.230116
1996	6427.327966
1997	6599.432588
...	...

Evolución de casos de cáncer traqueal, bronquial y pulmonar

Este análisis exploratorio inicial permite observar de manera detallada la evolución de la mortalidad por cáncer de pulmón en México, identificando posibles tendencias y patrones de crecimiento. Al examinar la variación en el número de muertes anuales, se pueden detectar periodos de mayor incidencia, así como evaluar la estabilidad o fluctuaciones en la progresión de la enfermedad.

Estos hallazgos resultan fundamentales para comprender el comportamiento histórico de los datos y proporcionan una base sólida para la implementación de los modelos predictivos en las siguientes secciones. Con esta información, se busca desarrollar algoritmos capaces de estimar con mayor precisión la evolución futura de las muertes por cáncer de pulmón, permitiendo una comparación efectiva entre los enfoques de Regresion lineal y redes neuronales recurrentes (LSTM).

Para ilustrar la evolución de las muertes por cáncer de pulmón en México a lo largo del tiempo, se presenta el siguiente histograma:



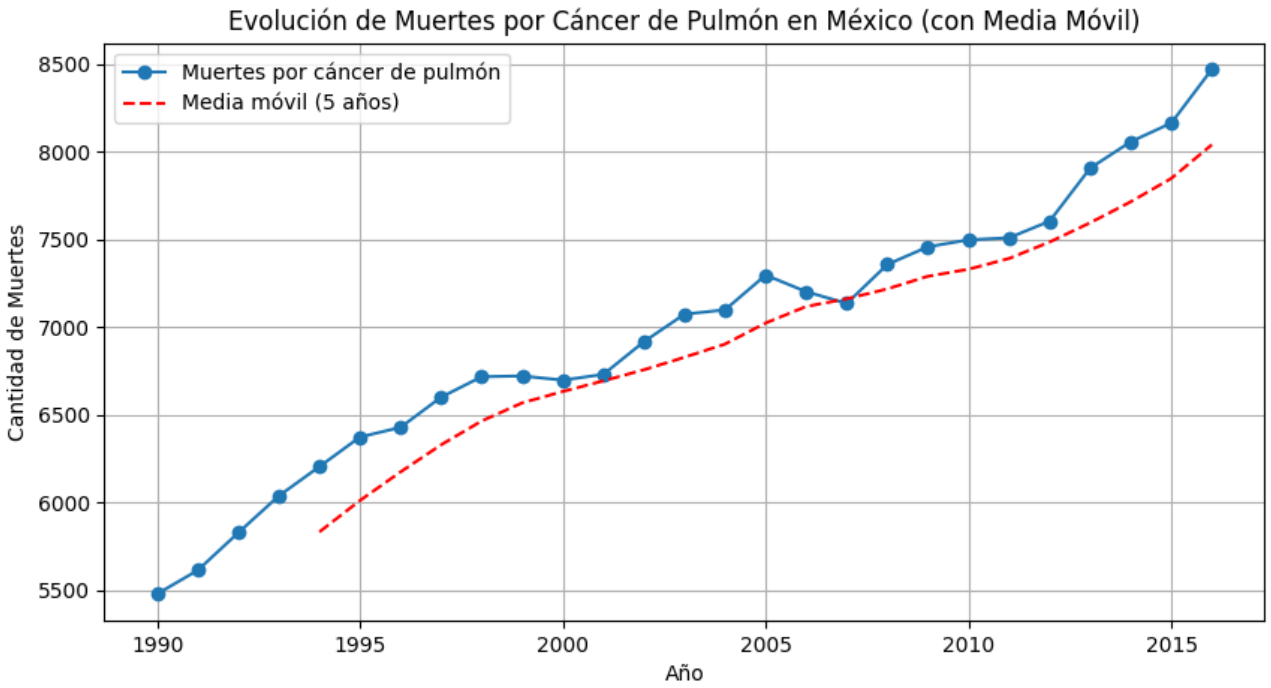
Evolución de muertes por cáncer de pulmón en México

Se realizó el cálculo de la media móvil, generando el siguiente gráfico. Este método es clave para suavizar fluctuaciones y resaltar tendencias subyacentes en los datos. Al eliminar el ruido, facilita una interpretación más clara y apoya el desarrollo de estrategias predictivas más precisas.

El cálculo de la media móvil de 5 anos se realiza con el siguiente código Python:

```
df_mexico_lung["Moving_Avg_5"] = df_mexico_lung["Tracheal, bronchus, and lung cancer"].rolling
    (window=5).mean()
```

Este código aplica una media móvil utilizando una ventana de 5 anos para los valores en la columna específica.



Media móvil en la evolución de muertes por cáncer de pulmón en México

Complementando el histograma y la media móvil, se realizaron los cálculos de la tasa de crecimiento anual:

```
# Calcular la tasa de crecimiento anual
df['Tasa de Crecimiento (%)'] = df['Tracheal, bronchus, and lung cancer'].pct_change() * 100

# Calcular el Crecimiento Anual Compuesto (CAGR)
valor_inicial = df['Tracheal, bronchus, and lung cancer'].iloc[0]
valor_final = df['Tracheal, bronchus, and lung cancer'].iloc[-1]
anos = df['Year'].iloc[-1] - df['Year'].iloc[0]

CAGR = ((valor_final / valor_inicial) ** (1/anos) - 1) * 100

print(f"Tasa de Crecimiento Anual Compuesto (CAGR): {CAGR:.2f}%")
print(df[['Year', 'Tracheal, bronchus, and lung cancer', 'Tasa de Crecimiento (%)']])
```

Este código realiza cálculos clave para analizar la evolución de las tasas de crecimiento anual.

EL cual, la Tasa de Crecimiento Anual Compuesto es del **1.69**Tal Crecimiento Anual Compuesto se desglosa de la siguiente manera:

Teniendo presentes tales dimensiones de la enfermedad, daremos inicio al modelo de Regresion Lineal.

Year	Tracheal, bronchus, and lung cancer	Tasa de Crecimiento (%)
1990	5477.606751	NaN
1991	5612.037783	2.45
1992	5826.280440	3.82
1993	6036.566757	3.61
1994	6203.391357	2.76
1995	6372.230116	2.72
1996	6427.327966	0.86
1997	6599.432588	2.68
1998	6716.166865	1.77
1999	6720.453596	0.06
2000	6696.652587	-0.35
2001	6729.512760	0.49
2002	6915.995188	2.77
2003	7072.815396	2.27
2004	7097.696918	0.35
2005	7295.835659	2.79
2006	7201.171839	-1.30
2007	7134.160110	-0.93
2008	7356.270666	3.11
2009	7457.153413	1.37
2010	7496.590945	0.53
2011	7508.335423	0.16
2012	7603.165121	1.26
2013	7905.750272	3.98
2014	8056.332637	1.90
2015	8162.953063	1.32
2016	8469.498356	3.76

Casos de cáncer traqueal, bronquial y pulmonar, y su tasa de crecimiento anual (1990-2016)

2.2.1. Modelo de Regresión Lineal

Se implementó un modelo de **Regresión Lineal** para predecir las muertes futuras. Se ajustó una línea de tendencia sobre los datos de entrenamiento y se proyectaron valores hasta el año 2040.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

Cargar datos

df = pd.read_csv("Muertes_Mexico.csv")
X = df[["Year"]].values
y = df["Tracheal, bronchus, and lung cancer "]

Entrenamiento del modelo

modelo = LinearRegression()
modelo.fit(X, y)

Prediccion para el 2040

anio_futuro = np.array([[2040]])
muertes_2040 = modelo.predict(anio_futuro)
print(f"Prediccion de muertes para el ano 2040: {muertes_2040[0]:.2f}")
```

Código de Regresion Lineal

2.2.2. Red Neuronal Recurrente (LSTM)

Se implementó una red neuronal recurrente del tipo LSTM para comparar la Predicción de los datos temporales. En contraste con el Modelo de Regresion Lineal, se normalizaron los datos antes del entrenamiento.

```python
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error

X = df[["Year"]].values
y = df["Tracheal, bronchus, and lung cancer"].values

# Normalizar los datos
scaler_x = MinMaxScaler()
X_scaled = scaler_x.fit_transform(X)

scaler_y = MinMaxScaler()
y_scaled = scaler_y.fit_transform(y.reshape(-1, 1))

# Crear secuencias para la LSTM
def create_sequences(X_data, y_data, seq_length):
    sequences, labels = [], []
    for i in range(len(X_data) - seq_length):
        sequences.append(np.hstack((X_data[i:i + seq_length], y_data[i:i + seq_length])))
        labels.append(y_data[i + seq_length])
    return np.array(sequences), np.array(labels)

seq_length = 5
X_seq, y_seq = create_sequences(X_scaled, y_scaled, seq_length)

# Division de datos en entrenamiento y prueba
split = int(0.8 * len(X_seq))
X_train, X_test = X_seq[:split], X_seq[split:]
y_train, y_test = y_seq[:split], y_seq[split:]

# Construccion del modelo LSTM optimizado
model = Sequential([
    LSTM(32, activation='tanh', return_sequences=True, input_shape=(seq_length, 2)),
    Dropout(0.2),
    LSTM(16, activation='tanh'),
    Dropout(0.2),
    Dense(10, activation='relu'),
    Dense(1)])

# Compilar el modelo
model.compile(optimizer='adam', loss='mse')

# Usar Early Stopping para evitar sobreentrenamiento
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Entrenar el modelo
model.fit(X_train, y_train, epochs=200, batch_size=16, validation_data=(X_test, y_test),
    callbacks=[early_stop])

# Hacer predicciones futuras hasta 2040
anos_futuros = np.arange(X.min(), 2041).reshape(-1, 1)
anos_futuros_scaled = scaler_x.transform(anos_futuros)

future_preds = []
input_seq = X_seq[-1].copy()

for _ in range(2040 - X.max()):
    pred = model.predict(input_seq.reshape(1, seq_length, 2))[0]
    future_preds.append(pred)
    input_seq = np.roll(input_seq, -1, axis=0)
    input_seq[-1] = np.hstack((anos_futuros_scaled[len(future_preds) - 1], pred))

# Desnormalizar predicciones
future_preds = scaler_y.inverse_transform(np.array(future_preds).reshape(-1, 1))

mse_regresion = mean_squared_error(y_test, y_pred)
mse_lstm = mean_squared_error(y_test, y_pred_lstm)

print(f"MSE Regresion Lineal: {mse_regresion}")
print(f"MSE LSTM: {mse_lstm}")
```

Modelo LSTM

2.3. Metodología de agrupamiento

Para trabajar con el problema de agrupamiento se utilizó el *datset* "Lung Cancer Prediction", que puede consultarse en la página de Kaggle. Dicho *dataset* tiene mil registros de pacientes con cáncer de pulmón, cuenta con 25 características, entre las que se encuentran; edad, género, contaminación del aire, riesgo genético, dolor de pecho y el nivel de su condición (bajo, medio y alto). Si bien esta última característica podría permitirnos considerar a los datos como etiquetados, no la tomaremos en cuenta para poder trabajarlos con métodos de aprendizaje no supervisado.

Las versiones de *Python* y las librerías utilizadas fueron las siguientes:

- Pandas versión: 2.2.2
- NumPy versión: 2.0.2
- Seaborn versión: 0.13.2
- Matplotlib versión: 3.10.0
- Sklearn versión: 1.6.1
- SciPy versión: 1.14.1

Como ya se mencionó, los registros tienen la característica *'Level'*, por lo que el objetivo de esta sección es implementar métodos de agrupamiento para generar 3 *clústeres* que nos permitan agrupar nuestros de manera similar a la característica *'Level'*, debido a esto utilizaremos métodos de no jerárquicos, posteriormente utilizaremos el método *PCA* para reducir la dimensión de nuestra *dataset* para poder visualizar los *clústeres* generados en gráficos 2D y 3D, finalmente compararemos los resultados que obtuvimos del agrupamiento con los datos originales para conocer la precisión que obtuvimos.

Primero hacemos una copia del *dataset* con el que estamos trabajando y eliminamos las características *'index'*, *'Patient ID'* y *'Level'*, pues las primeras dos no son relevantes para el agrupamiento y ya planteamos que la última la usaremos para conocer la precisión que obtuvimos.

2.3.1. Agrupamiento con k-means

```
cancer_data = df.drop(columns=["Patient Id", "Level", "index"])

# Estandarizamos los datos
scaler = StandardScaler()
scaled_data = scaler.fit_transform(cancer_data)

# Los convertimos en una data frame
scaled_data_df = pd.DataFrame(scaled_data, columns=cancer_data.columns)
```

```
# posibles grupos: Low, Medium y High
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
kmeans.fit(scaled_data)

scaled_data_df['cluster'] = kmeans.labels_
```

Código de k-means

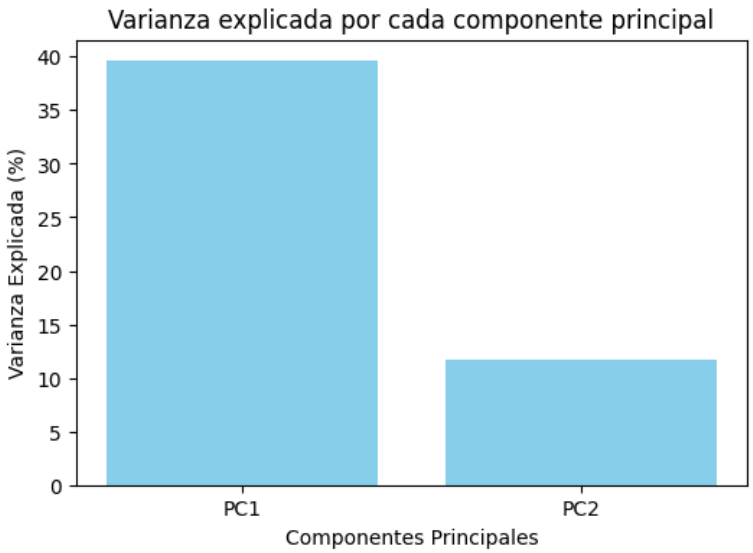
Hacemos una reducción de dimensiones para poder visualizar los datos en 2D y 3D.

```
pca = PCA(n_components=2)
cancer_pca = pca.fit_transform(scaled_data)
print(f"Varianza total explicada: {sum(pca.explained_variance_ratio_)*100:.2f}%")
```

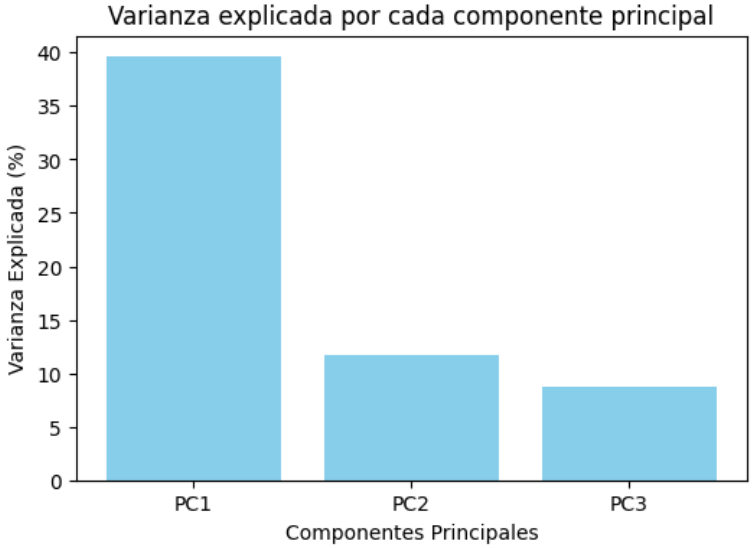
PCA para 2D

```
pca3 = PCA(n_components=3)
cancer_pca3 = pca.fit_transform(scaled_data)
print(f"Varianza total explicada: {sum(pca3.explained_variance_ratio_)*100:.2f}%")
```

PCA para 3D



Varianza de 51.32 %



Varianza de 60.09 %

2.3.2. Agrupamiento con DBSCAN

También implementaremos el agrupamiento con el método *DBSCAN*. Los primeros valores que proponemos para ε y el número mínimo de muestras son 0.5 y 20 respectivamente, de lo cual obtenemos.

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=.5, min_samples=20)
dbscan.fit(scaled_data)

print(dbscan.labels_.min(), dbscan.labels_.max())
```

DSCAN con ε=0.5

```
-1  9
```

Salida del código

El -1 nos indica que el algoritmo clasificó algunos datos como *atípicos*, que la etiqueta más grande sea 9 quiere decir que el algoritmo formó 10 grupos, como solo queremos formar 3 grupos, correremos el algoritmo para distintos valores de ε hasta que solo se formen 3 *clústeres*.

```
    i = 0.5

    while dbscan.labels_.max() > 2:
        i += 0.1
        dbscan = DBSCAN(eps=i, min_samples=20)
        dbscan.fit(scaled_data)
        print(i, dbscan.labels_.min(), dbscan.labels_.max())
```

Con esto encontramos que la ε que nos permite tener 3 grupos es 4.7, pero el algoritmo sigue clasificando algunos datos como atípicos, por lo que De manera similar encontramos el número de puntos mínimos

```
    j = 20

    while dbscan.labels_.min() < 0 :
        j -= 1
        dbscan = DBSCAN(eps=4.9, min_samples=j)
        dbscan.fit(scaled_data)
        print(j, dbscan.labels_.min(), dbscan.labels_.max())
```

Finalmente obtenemos que $\varepsilon = 4{,}9$ y *min_samples = 19*, por lo que utilizaremos estos parámetros para el método *DBSCAN*. Las etiquetas que obtengamos las guardaremos en una columna del *dataframe scaled_data_df*

```
    dbscan = DBSCAN(eps=4.9, min_samples=19)
    dbscan.fit(scaled_data)

    scaled_data_df['cluster DBSCAN'] = dbscan.labels_
```

3. Resultados

3.1. Resultados de clasificación

3.1.1. Support Vector Machine

Posterior al tiempo de búsqueda de parámetros del *GridSearchCV* para el modelo de *Support Vector Classifier*, se ejecutó la siguiente celda de código para saber los parámetros con mejor desempeño entre los propuestos:

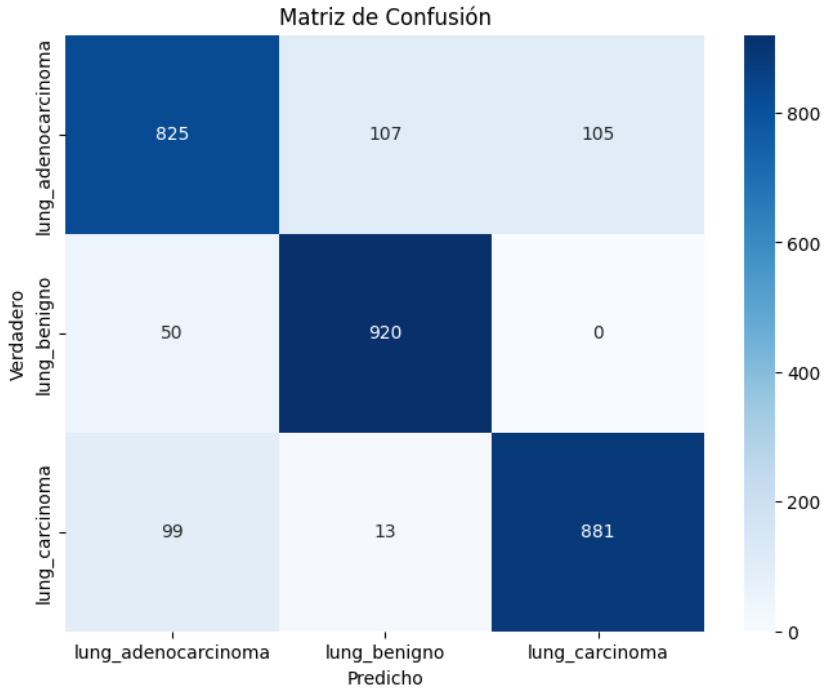
```
# Imprimir parametros con mejores metricas
print(f'Best C: {grid_search.best_params_["C"]}')
print(f'Best kernel: {grid_search.best_params_["kernel"]}')
print(f'Best gamma: {grid_search.best_params_["gamma"]}')

# Evaluar el modelo con los mejores parametros
best_svm_model = grid_search.best_estimator_
y_pred = best_svm_model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
print(f'Precision con el mejor C, kernel y gamma: {accuracy:.2f}')
```

Código para impresión de parámetros con mejor desempeño y evaluación del modelo

```
Best C: 10
Best kernel: rbf
Best gamma: 0.01
Precision con el mejor C, kernel y gamma: 0.87
```

Impresión mejores parámetros y evaluación del modelo

Se graficó la matriz de confusión con ayuda de la función *confusion_matrix* alojada en la librería *sklearn.metrics* y con apoyo de la librería *seaborn*, obteniendo el resultado siguiente:



Matriz de confusión del modelo de *SVC*

Como última métrica para este modelo se generó el reporte de clasificación con ayuda de la función *classification_report* alojada igualmente en la librería *sklearn.metrics*:

	precision	recall	f1-score	support
lung_adenocarcinoma	0.85	0.80	0.82	1037
lung_benigno	0.88	0.95	0.92	970
lung_carcinoma	0.89	0.89	0.89	993
accuracy			0.88	3000
macro avg	0.88	0.88	0.88	3000
weighted avg	0.87	0.88	0.87	3000

Reporte de clasificación del modelo de *SVC*

3.1.2. Regresión Logística

Tras el tiempo de búsqueda de los mejores parámetros del *GridSearchCV* para el modelo de Regresión Logística, se ejecutó la siguiente celda de código para saber los parámetros con mejor desempeño, de los dados al *GridSearchCV*:

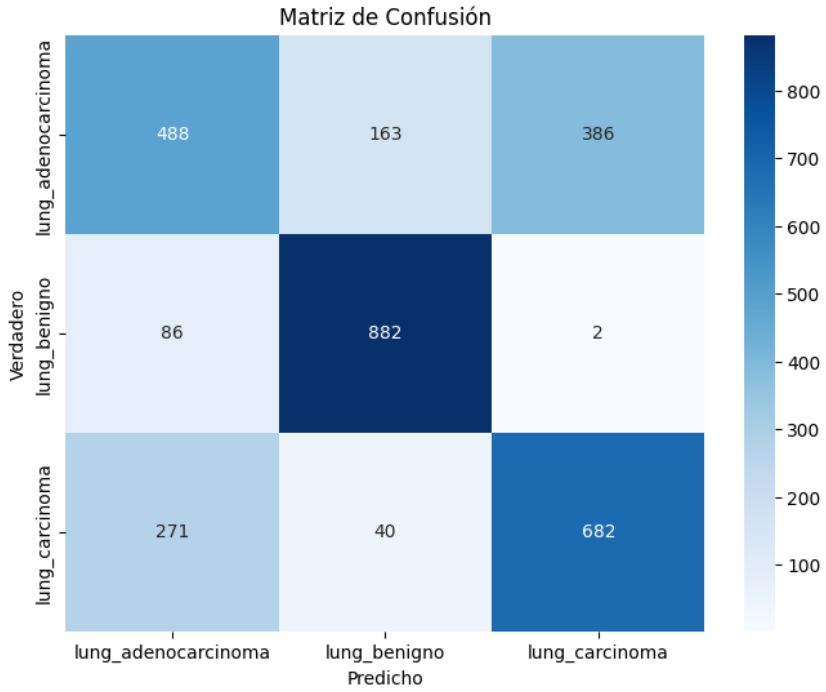
```
# Imprimir lista de mejores parametros
print("Mejores parametros encontrados: ", grid_search_lrm.best_params_)
# Imprimir precision del modelo con mejores parametros
print("Mejor precision obtenida: ", grid_search_lrm.best_score_)
```

Código para impresión de parámetros con mejor desempeño y evaluación del modelo

```
Mejores parametros encontrados:  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
Mejor precision obtenida:  0.6869166666666666
```

Impresión mejores parámetros y evaluación del modelo

Se graficó la matriz de confusión del modelo de regresión logística usando las mismas librerías, obteniendo la gráfica siguiente:



Matriz de confusión del modelo de Regresión Logística

Como última métrica del modelo de regresión logística se generó el reporte de clasificación, obteniendo el resultado siguiente:

	precision	recall	f1-score	support
lung_adenocarcinoma	0.58	0.47	0.52	1037
lung_benigno	0.81	0.91	0.86	970
lung_carcinoma	0.64	0.69	0.66	993
accuracy			0.68	3000
macro avg	0.68	0.69	0.68	3000
weighted avg	0.67	0.68	0.68	3000

Reporte de clasificación del modelo de Regresión Logística

3.1.3. XGBClassifier

Luego de una extensa búsqueda del *GridSearchCV* con el modelo *XGBClassifier*, se ejecutó la siguiente celda de código para saber los parámetros del modelo con mejor desempeño:

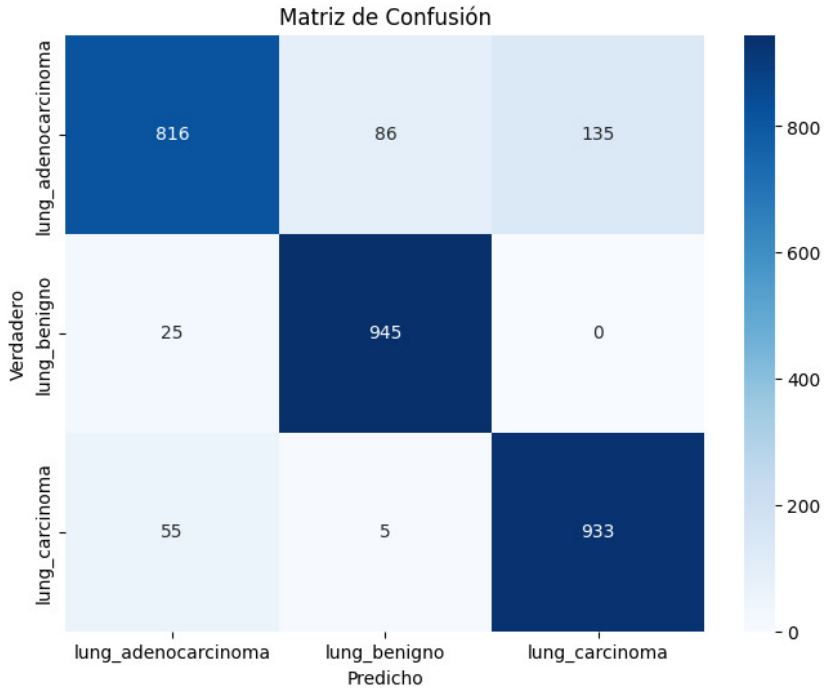
```
# Imprimir parametros con mejores metricas
print(f'Best n_estimators: {grid_search_XGB.best_params_["n_estimators"]}')
print(f'Best learning_rate: {grid_search_XGB.best_params_["learning_rate"]}')
print(f'Best max_depth: {grid_search_XGB.best_params_["max_depth"]}')
print(f'Best subsample: {grid_search_XGB.best_params_["subsample"]}')
print(f'Best colsample_bytree: {grid_search_XGB.best_params_["colsample_bytree"]}')
print(f'Best gamma: {grid_search_XGB.best_params_["gamma"]}')
print(f'Best reg_alpha: {grid_search_XGB.best_params_["reg_alpha"]}')
print(f'Best reg_lambda: {grid_search_XGB.best_params_["reg_lambda"]}')

# Evaluar el modelo con los mejores parametros
best_XGB_model = grid_search_XGB.best_estimator_
y_pred = best_XGB_model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
print(f'Precision con los mejores parametros: {accuracy:.2f}')
```

Código para impresión de parámetros con mejor desempeño y evaluación del modelo

```
Best n_estimators: 200
Best learning_rate: 0.1
Best max_depth: 10
Best subsample: 0.7
Best colsample_bytree: 0.7
Best gamma: 0.1
Best reg_alpha: 0.1
Best reg_lambda: 10
Precision con los mejores parametros: 0.90
```

Impresión mejores parámetros y evaluación del modelo

Se graficó la matriz de confusión del modelo *XGBClassifier*, obteniendo la gráfica siguiente:



Matriz de confusión del modelo de *XGBCLassifier*

Y nuevamente como última métrica del modelo, se generó el reporte de clasificación generando lo siguiente:

	precision	recall	f1-score	support
lung_adenocarcinoma	0.91	0.79	0.84	1037
lung_benigno	0.91	0.97	0.94	970
lung_carcinoma	0.87	0.94	0.91	993
accuracy			0.90	3000
macro avg	0.90	0.90	0.90	3000
weighted avg	0.90	0.90	0.90	3000

Reporte de clasificación del modelo *XGBClassifier*

3.1.4. *Transfer Learning* con *MobileNetV2*

Este modelo tardó 22 épocas en entrenarse. Se ejecutó la siguiente celda de código para medir el *accuracy* del modelo:

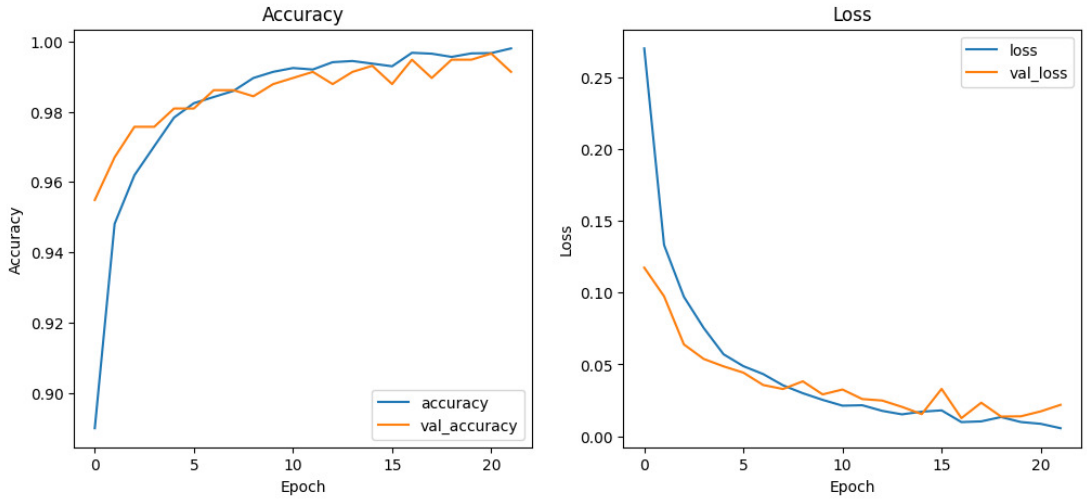
```
# Evaluamos el modelo
loss, accuracy = model.evaluate(val_ds)
print(f"Accuracy: {accuracy}")
```

Código para la impresión del *accuracy* del modelo

```
Accuracy: 0.9940000176429749
```

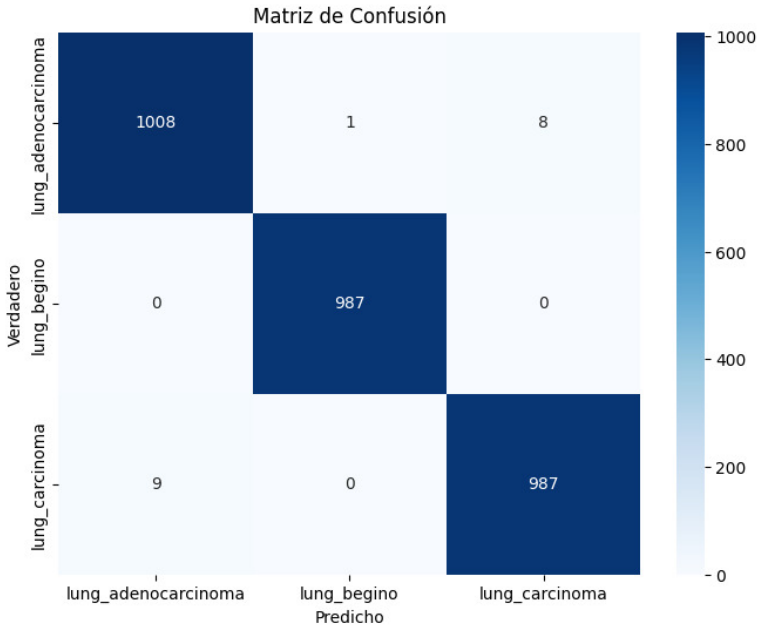
Impresión *accuracy* del modelo

Para los modelos CNN, se generaron adicionalmente las gráficas de *Training vs Validation* para las medidas *accuracy* y *loss*. Esto con apoyo del método *history* aplicado al modelo entrenado, estos fueron los resultados:



Gráfica *Training vs Validation* de modelo *Transfer Learning* con *MobileNetV2*

Al ser un modelo de clasificación, se generó igualmente una matriz de confusión con las predicciones del modelo y se obtuvo la gráfica siguiente:



Matriz de confusión de modelo *Transfer Learning* con *MobileNetV2*

Una vez más se generó el reporte de clasificación, el resultado fue el siguiente:

	precision	recall	f1-score	support
lung_adenocarcinoma	0.99	0.99	0.99	1017
lung_benigno	1.00	1.00	1.00	987
lung_carcinoma	0.99	0.99	0.99	996
accuracy			0.99	3000
macro avg	0.99	0.99	0.99	3000
weighted avg	0.99	0.99	0.99	3000

Reporte de clasificación de modelo *Transfer Learning* con *MobileNetV2*

3.1.5. *Transfer Learning* con *VGG16*

Este modelo tardó 31 épocas en entrenarse. Se ejecutó la siguiente celda de código para medir el *accuracy* del modelo:

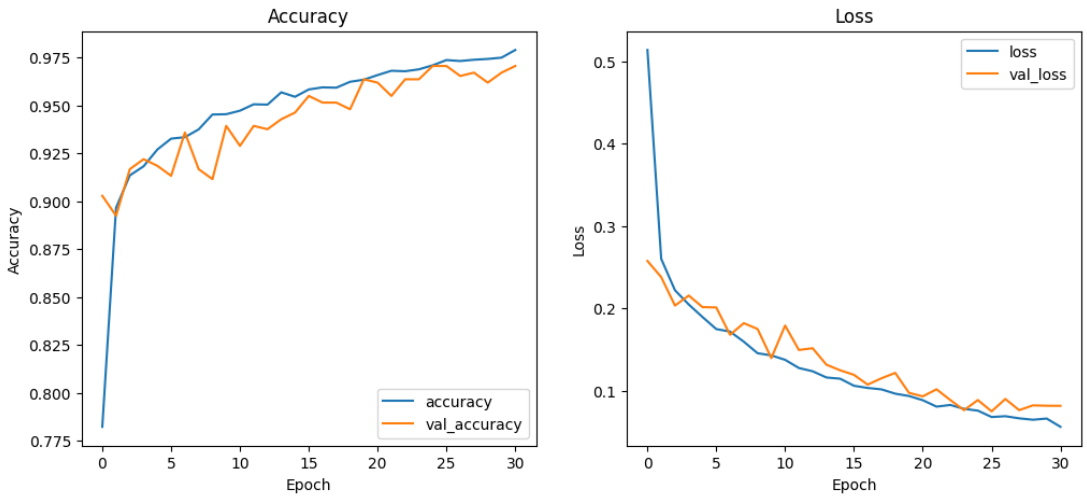
```
# Evaluamos el modelo
loss, accuracy = model.evaluate(val_ds)
print(f"Accuracy: {accuracy}")
```

Código para la impresión del *accuracy* del modelo

```
Accuracy: 0.9660000205039978
```

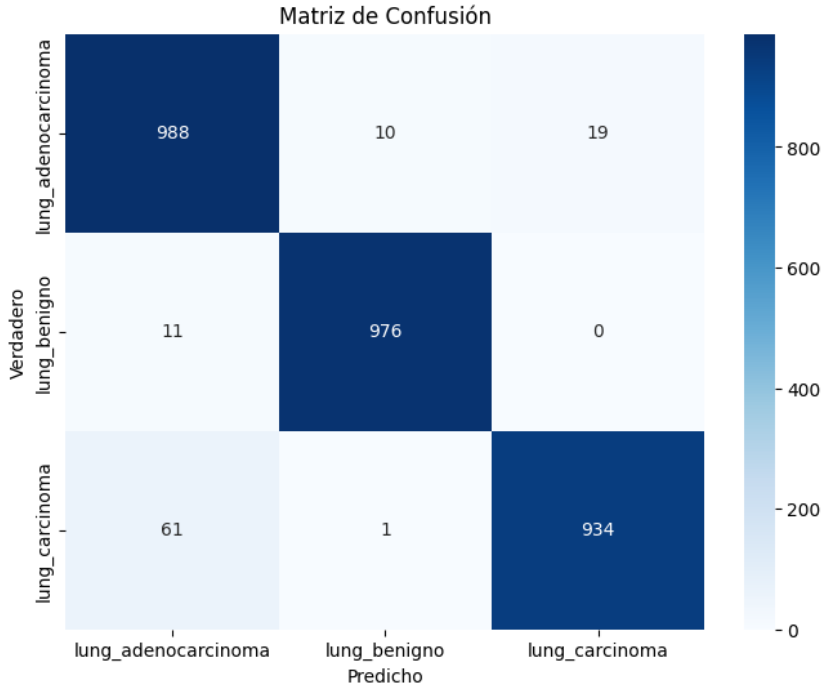
Impresión *accuracy* del modelo

Las gráficas de *Training vs Validation* para las medidas de *accuracy* y *loss*, fueron las siguientres para este modelo:



Gráfica *Training vs Validation* de modelo *Transfer Learning* con *VGG16*

La matriz de confusión para esta CNN con modelo base *VGG16* dio el resultado siguiente:



Matriz de confusión de modelo *Transfer Learning* con *VGG16*

Se generó el reporte de clasificación, el resultado fue el siguiente:

	precision	recall	f1-score	support
lung_adenocarcinoma	0.93	0.97	0.95	1017
lung_benigno	0.99	0.99	0.99	987
lung_carcinoma	0.98	0.94	0.96	996
accuracy			0.97	3000
macro avg	0.97	0.97	0.97	3000
weighted avg	0.97	0.97	0.97	3000

Reporte de clasificación de modelo *Transfer Learning* con *VGG16*

3.1.6. *Transfer Learning* con *ResNetRS101*

Este modelo tardó también 31 épocas en entrenarse. Como con los modelos anteriores de CNN se ejecutó la siguiente celda de código para medir el *accuracy* del modelo:

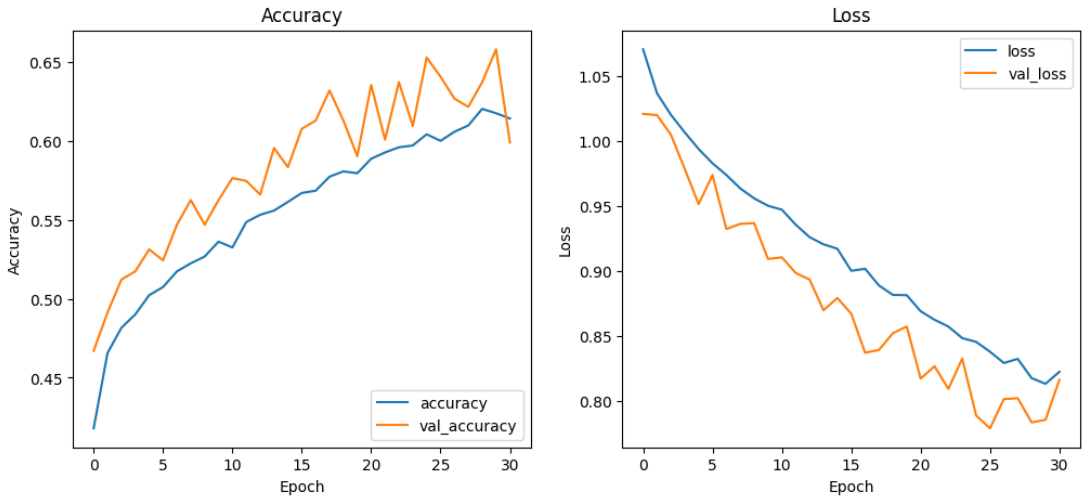
```
# Evaluamos el modelo
loss, accuracy = model.evaluate(val_ds)
print(f"Accuracy: {accuracy}")
```

Código para la impresión del *accuracy* del modelo

```
Accuracy: 0.6293333172798157
```

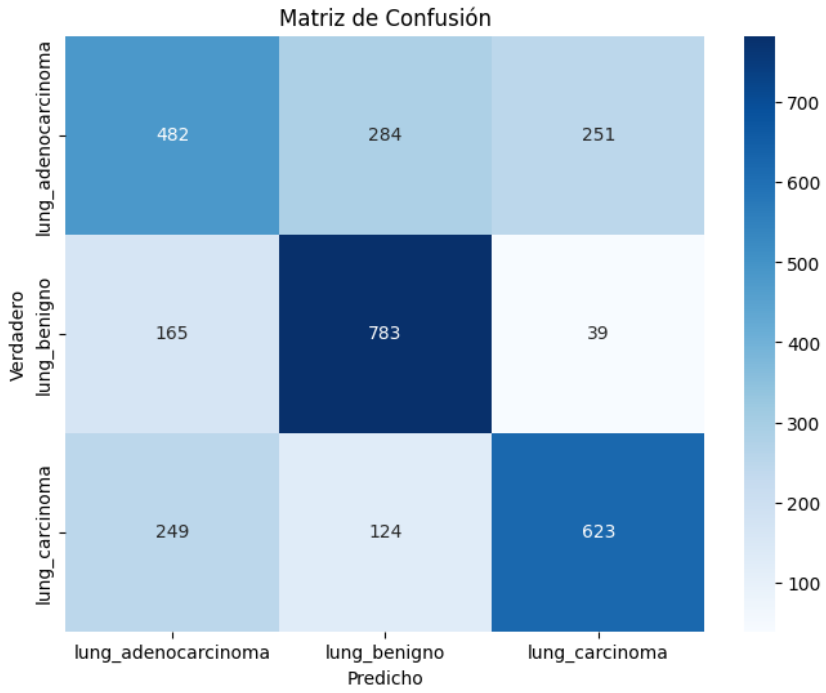
Impresión *accuracy* del modelo

Para este modelo de CNN, se graficó igualmente *Training vs Validation* para las medidas de *accuracy* y *loss*, resultando lo siguiente:



Gráfica *Training vs Validation* de modelo *Transfer Learning* con *ResNetRS101*

La matriz de confusión para este modelo de *Transfer Learning* con *ResNetRs101* fue la siguiente:



Matriz de confusión de modelo *Transfer Learning* con *ResNetRs101*

Y por último el reporte de clasificación de este modelo fue el siguiente:

	precision	recall	f1-score	support
lung_adenocarcinoma	0.54	0.47	0.50	1017
lung_benigno	0.66	0.79	0.72	987
lung_carcinoma	0.68	0.63	0.65	996
accuracy			0.63	3000
macro avg	0.63	0.63	0.63	3000
weighted avg	0.63	0.63	0.62	3000

Reporte de clasificación de modelo *Transfer Learning* con *ResNetRs101*

3.1.7. *Fine Tuning* con *ResNetRS101*

Este modelo tardó 9 épocas en entrenarse y demoró ligeramente más al ser un modelo con un mayor número de parámetros. Se ejecutó la siguiente celda de código para medir el accuracy del modelo:

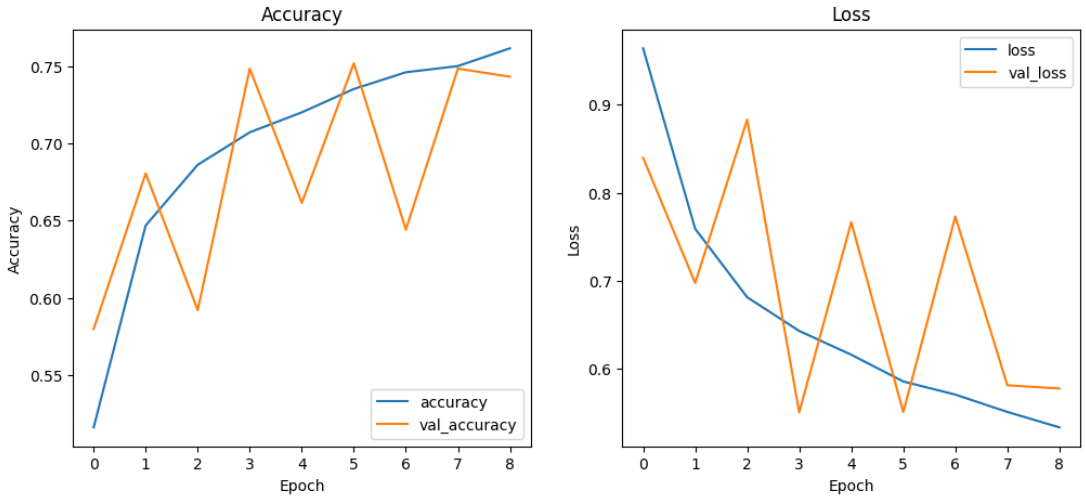
```
# Evaluamos el modelo
loss, accuracy = model.evaluate(val_ds)
print(f"Accuracy: {accuracy}")
```

Código para la impresión del *accuracy* del modelo

```
Accuracy: 0.7236666679382324
```

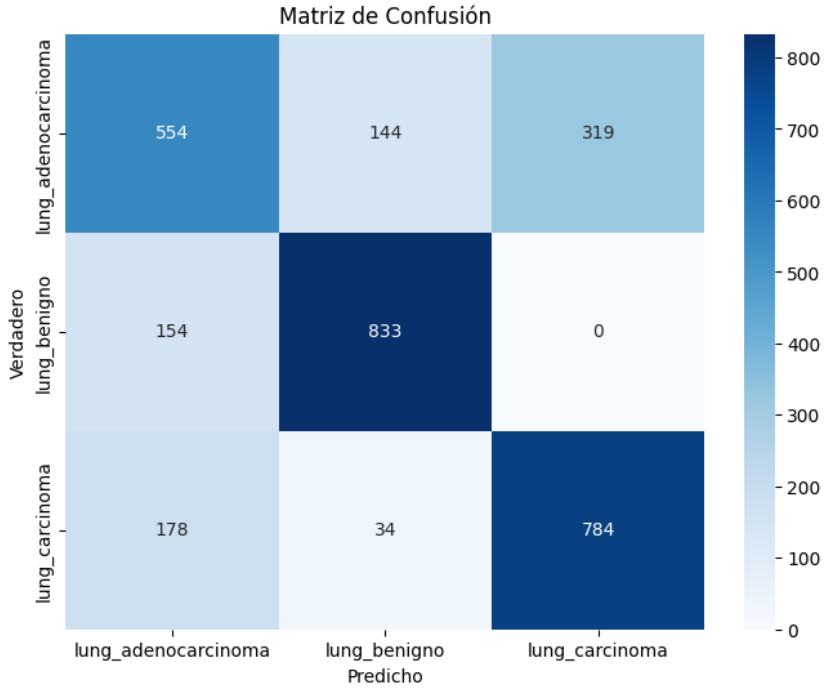
Impresión *accuracy* del modelo

Las gráficas de *Training vs Validation* para las medidas *accuracy* y *loss* de este modelo fueron las siguientes:



Gráfica *Training vs Validation* de modelo *Fine Tuning* con *ResNetRS101*

La matriz de confusión de este modelo de *Fine Tuning* fue la siguiente:



Matriz de confusión de modelo *Fine Tuning* con *ResNetRs101*

El reporte de clasificación de este modelo fue el siguiente:

	precision	recall	f1-score	support
lung_adenocarcinoma	0.63	0.54	0.58	1017
lung_benigno	0.82	0.84	0.83	987
lung_carcinoma	0.71	0.79	0.75	996
accuracy			0.72	3000
macro avg	0.72	0.73	0.72	3000
weighted avg	0.72	0.72	0.72	3000

Reporte de clasificación de modelo *Fine Tuning* con *ResNetRs101*

3.1.8. Propuesta de CNN

Esta última CNN tardó 15 épocas en entrenarse. Como con todos los modelos anteriores, se ejecutó la siguiente celda de código para calcular el *accuracy* del modelo:

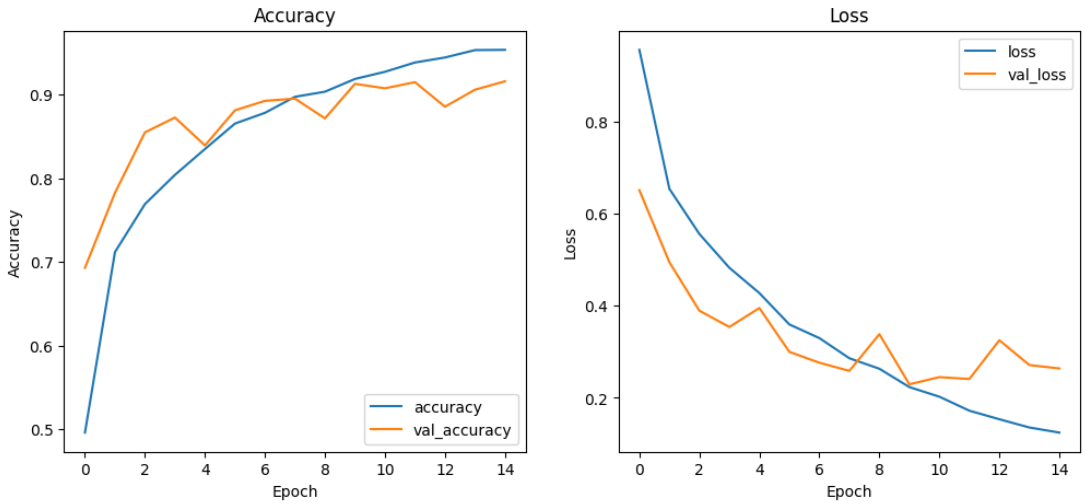
```
# Evaluamos el modelo
loss, accuracy = model.evaluate(val_ds)
print(f"Accuracy: {accuracy}")
```

Código para la impresión del *accuracy* del modelo

```
Accuracy: 0.9129999876022339
```

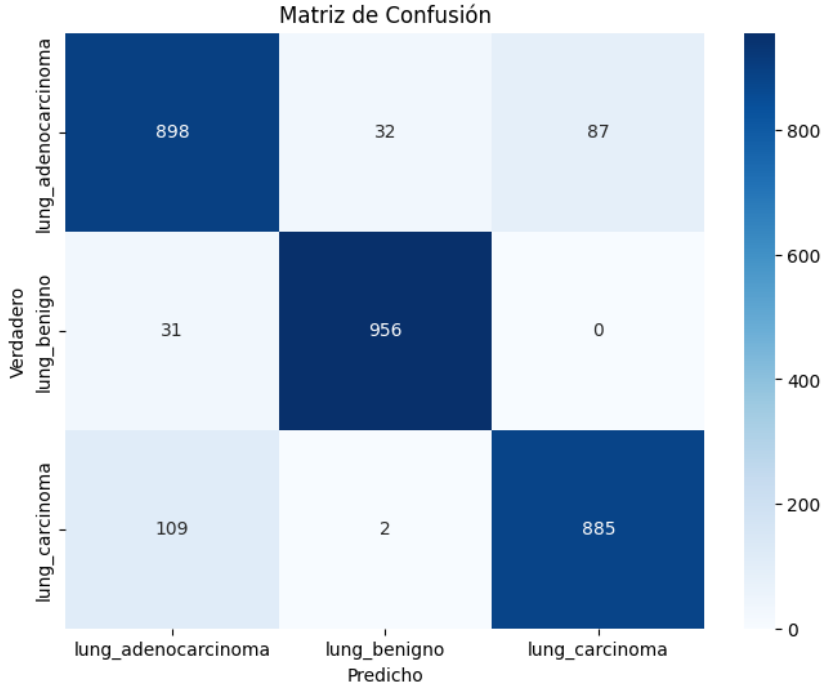
Impresión *accuracy* del modelo

Igualmente se generaron las gráficas *Training vs Validation* para este modelo, generando lo siguiente:



Gráfica *Training vs Validation* del modelo de CNN propuesto

La matriz de confusión del modelo propuesto fue la siguiente:



Matriz de confusión deL modelo de CNN propuesto

Y por último el reporte de clasificación fue el siguiente:

	precision	recall	f1-score	support
lung_adenocarcinoma	0.87	0.88	0.87	1017
lung_benigno	0.97	0.97	0.97	987
lung_carcinoma	0.91	0.89	0.90	996
accuracy			0.91	3000
macro avg	0.91	0.91	0.91	3000
weighted avg	0.91	0.91	0.91	3000

Reporte de clasificación del modelo de CNN propuesto

Para finalizar el apartado de resultados de la clasificación en este proyecto, se propone una tabla comparativa de los promedios de las diferentes métricas en los reportes de clasificación de los modelos utilizados a lo largo de esta sección.

Modelo	Accuracy	Precision	Recall	F1-score
SVC	0.88	0.88	0.88	0.88
Regresión Logística	0.68	0.68	0.69	0.68
XGBClassifier	0.90	0.90	0.90	0.90
TL MobileNetV2	**0.99**	**0.99**	**0.99**	**0.99**
TL VGG16	**0.97**	**0.97**	**0.97**	**0.97**
TL ResNetRS101	0.63	0.63	0.63	0.63
FT ResNetRS101	0.72	0.72	0.73	0.72
CNN propuesta	0.91	0.91	0.91	0.91

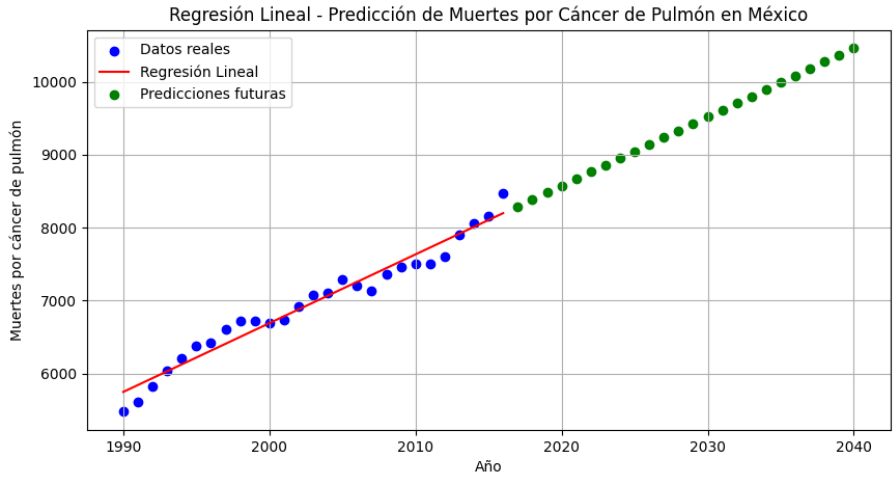
Tabla comparativa de métricas de los modelos

3.2. Resultados de predicción

El fin de trabajar con dos modelos, el de regresión lineal y la red neuronal, es para comparar su desempeño y así mismo, sus respectivos resultados. Cabe destacar que el año "objetivo" es el 2040.

3.2.1. Modelo de Regresión Lineal

De manera concisa, el número total de muertes estimadas para el año 2040 por el Modelo de Regresión Lineal son: **10461.07**, proporcionándonos el siguiente gráfico:

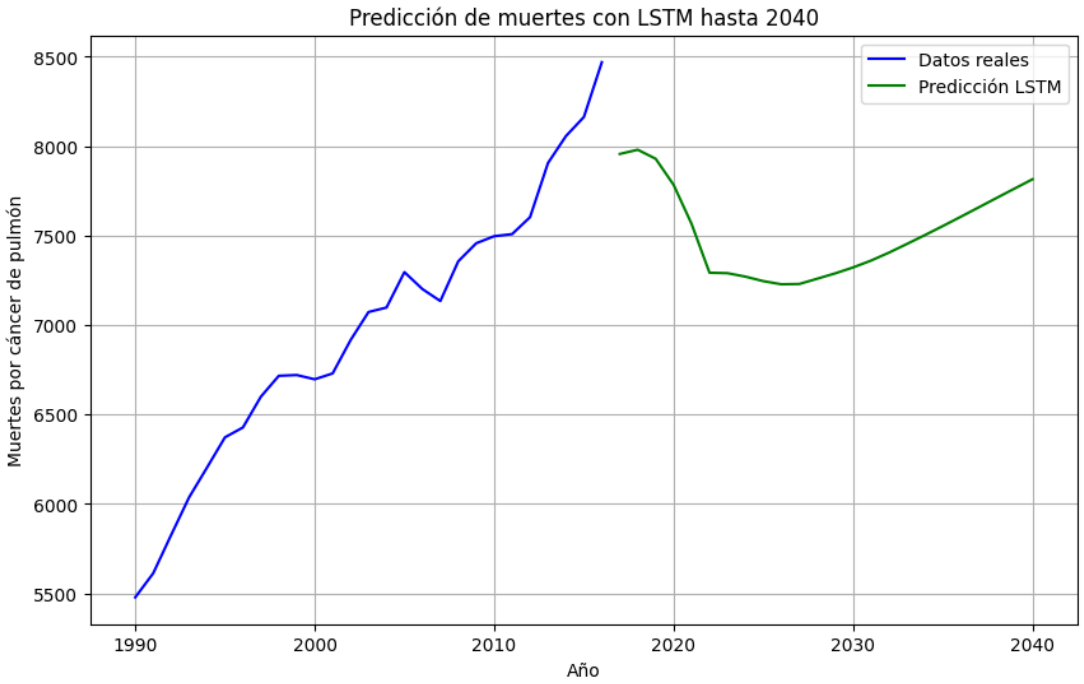


Modelo de Regresión Lineal

Como se puede apreciar, la evolución es completamente lineal.

3.2.2. Red Neuronal Recurrente (LTSM)

Por parte de la red neuronal concurrente, su predicción es de **7815.07**, arrojándonos el siguiente gráfico:



Modelo LSTM

A diferencia del modelo de regresión lineal, la red neuronal LSTM muestra un comportamiento menos lineal, adaptándose al conjunto de datos estudiados.

3.2.3. Comparación entre modelos

Existe cierta cantidad de cálculo para la comparación entre diversos modelos, tales son: Error Absoluto Medio (MAE, por sus siglas en inglés), Error Cuadrático Medio (MSE, por sus siglas en inglés), Coeficiente de Determinación (R^2); no obstante, en esta ocasión, solo nos centraremos en el MSE para realizar la comparación de desempeño y resultados entre ambos modelos.

$$\text{MSE} = \frac{1}{n}\sum(Y_i - \hat{Y}_i)^2$$

Fórmula MSE

El porqué radica en que en general, un MSE más bajo indica que el modelo está haciendo predicciones más precisas, ya que el error cuadrático castiga los errores grandes, lo que impulsa a los modelos a evitar grandes desviaciones. Es particularmente útil cuando deseas evitar errores muy grandes en las predicciones, como es el caso con las muertes causadas por cáncer de pulmón, donde un error grande podría tener consecuencias graves.

En este momento, estamos enfocados en observar qué modelo cuenta con un MSE menor.

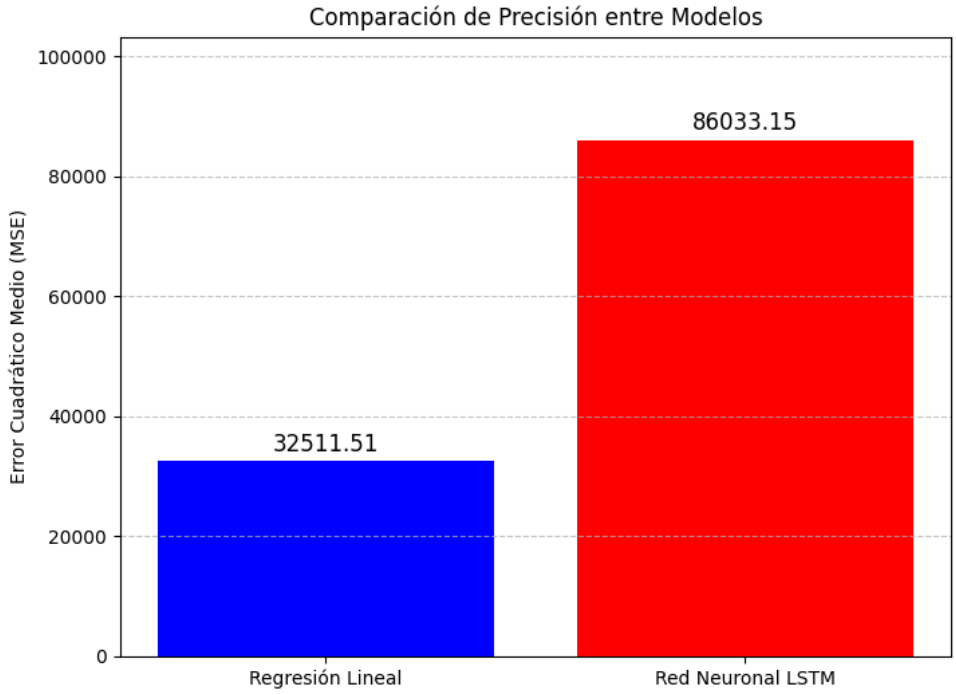
El cálculo del MSE tanto en el modelo de regresión lineal y la red neuronal LSTM fue realizado con la librería sklearn.metrics, haciendo uso de su método: "mean squared error".

```
mse = mean_squared_error(y_test, y_pred)
```

Por parte de la red neuronal, primero se desnormalizaron las predicciones para estar dentro del mismo rango de valores y posterior a eso, se aplicó el método de la librería:

```
# Hacer predicciones con el modelo LSTM en el conjunto de prueba
y_pred_lstm = model.predict(X_test)

# Desnormalizar las predicciones de la LSTM
y_pred_lstm_original = scaler_y.inverse_transform(y_pred_lstm)
y_test_original = scaler_y.inverse_transform(y_test)

# Calcular MSE con valores originales
mse_lstm = mean_squared_error(y_test_original, y_pred_lstm_original)
```

Dichos cálculos fueron útiles para realizar un gráfico y comparar sus resultados.



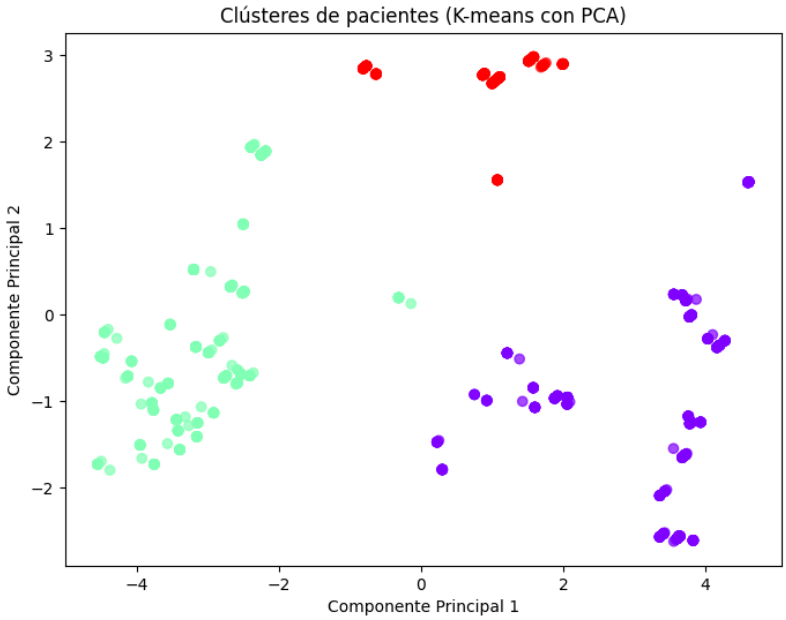
MSE en ambos modelos

Señalando que el modelo de regresión lineal, es más apropiado para este caso.

3.3. Resultados de agrupamiento

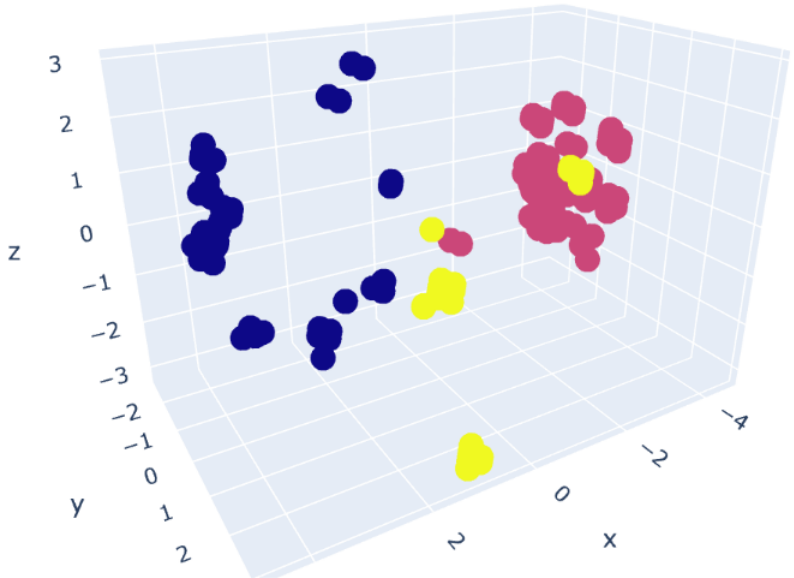
3.3.1. Resultados de k-means

Después de aplicar el método k-means con 3 *clústeres* y la reducción de dimensionalidad obtuvimos las siguientes gráficas



k-means con PCA en 2D

En la gráfica de *k-means* con *PCA* en 2D podemos apreciar que se observan significativamente menos de 1,000 puntos, esto se debe a la pérdida y distorsión de datos que se sufrió al aplicar *PCA*, pues como se aprecia en la gráfica de su varianza, solo se recupera el 51.32 %.

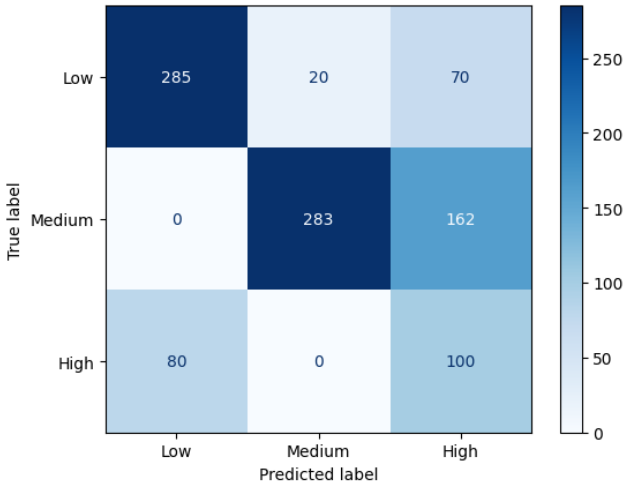


k-means con PCA en 3D

En la gráfica de *k-means* con *PCA* en 3D parecieria que el número de observaciones aumentes, lo cuál tiene sentido ya que la varianza total aumentó a 60.90 % al realizar *PCA* para 3 componentes, sin embargo seguimos teniendo una pérdida de datos bastante significativa.

3.3.2. Evaluación del modelo

A continuación mostramos la matriz de confusión que obtuvimos con respecto al agrupamiento de *k-means*



Matriz de confusión del modelo *k-means*

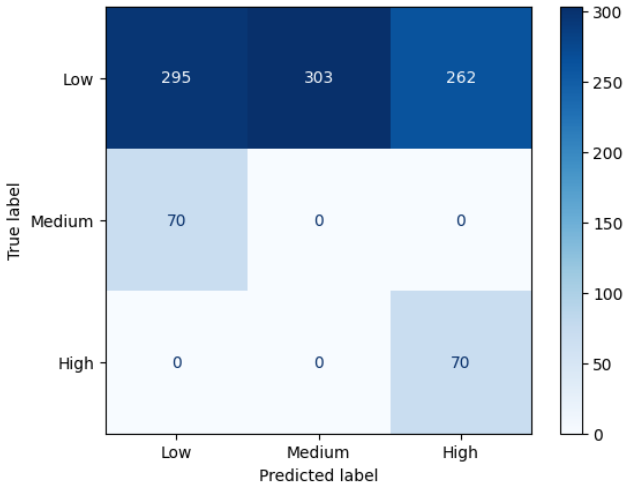
De la cual podemos se obtiene el siguiente reporte:

	precision	recall	f1-score	support
Low	0.78	0.76	0.77	375
Medium	0.93	0.64	0.76	445
High	0.30	0.56	0.39	180
accuracy			0.67	1000
macro avg	0.67	0.65	0.64	1000
weighted avg	0.76	0.67	0.70	1000

Reporte de clasificación de *k-means*

3.3.3. Evaluación de DBSCAN

Pare el método *DBSCAN* obtuvimos la siguiente matriz de confusión



Matriz de confusión para *DBSCAN*

	precision	recall	f1-score	support
Low	0.81	0.34	0.48	860
Medium	0.00	0.00	0.00	70
High	0.21	1.00	0.35	70
accuracy			0.36	1000
macro avg	0.34	0.45	0.28	1000
weighted avg	0.71	0.36	0.44	1000

Reporte de clasificación de *DBSCAN*

4. Conclusiones

El objetivo del apartado de clasificación fue desarrollar un modelo con los conocimientos y técnicas adquiridos a lo largo del *Samsung Innovation Campus*, que pudiera distinguir entre tres tipos de cáncer de pulmón, dada una imagen histopatológica del tumor. Para esto se propusieron 8 diferentes modelos, 3 contenidos en la libreria *sklearn* y 5 modelos de redes neuronales, desarrollados con la ayuda de las librerías *keras* y *tensorflow*. Estos modelos fueron evaluados con diferentes métricas para medir y cuantificar su desempeño, algunas métricas fueron: el *accuracy*, la matriz de confusión, las gráficas *Training vs Validation* entre otras.

En la sección de resultados del apartado de clasificación, se muestra la **Tabla comparativa** de las principales métricas de estos modelos. Podemos observar que el modelo de *Transfer Learning* con *MobileNetV2* fue el modelo con mejores métricas, consiguiendo un 0.99 de *precision*, *recall* y *F1-score*. Esto podría ser indicio de que este modelo fue el que obtuvo mejores resultados, pero al ser valores casi perfectos, es probable que exista un sobre ajuste a los datos de entrenamiento y por lo tanto el modelo no generalice bien con imágenes ajenas al *dataset* de entrenamiento. Los modelos de *XGBClassifier*, la CNN de *Transfer Learning* con *VGG16* y la CNN propuesta, al tener los 3 métricas superiores a 0.9, es más probable que tengan una mejor generalización de las características aprendidas del conjunto de entrenamiento. Los modelos con peor desempeño fueron la Regresión Logística y el modelo de *Transfer Learning* con ResNetRS101, teniendo el primero métricas de 0.68 y 0.69, y el segundo métricas de 0.63. El modelo de *Support Vector Classifier* tuvo un desempeño moderado acercándose a bueno con métricas de 0.88.

Al observar las diferentes matrices de confusión y los reportes de clasificación, podemos distinguir una mayor facilidad de todos los modelos para clasificar los tumores benignos. Todos tuvieron dificultades en la distinción de las clases adenocarcinoma y carcinoma escamocelular, aún cuando las imágenes se convirtieron a escala de grises justamente para evitar esto. Aunque, al observar diferentes imágenes del *dataset* usado, sí existe una clara diferencia entre las imágenes de tumor benigno y las otras 2 clases a simple vista.

Las diferencias de desempeño entre los modelos son coherentes con lo visto a lo largo del curso. El modelo de Regresión Logística es uno sencillo y poco potente para tareas de clasificación sencilla, por lo que su bajo desempeño en la clasificación de imágenes, no es una sorpresa. El modelo de *Transfer Learning* con *ResNetRS101* tuvo un desempeño pobre, esto es debido probablemente a las dimensiones de la red y a la cantidad de datos pues se trató de una cantidad moderada. Su posterior mejoría usando la técnica de *Fine Tuning*, era de esperarse, pues una mayor cantidad de pesos permitió que se adaptará mejor a los datos de entrenamiento. El *Support Vector Classifier* tuvo un desempeño aceptable, como era de esperarse para uno de los modelos más robustos de *Machine Learning*. El probable sobre ajuste del modelo de *Transfer Learning* con *MobileNetV2*, sí fue un poco sorprendente, pues se trata de un modelo pequeño. El sobre ajuste podría deberse a que más de la mitad de los parámetros de la red completa, eran parámetros entrenables, y el resto estaban pre-entrenados, pero esto es solo una hipótesis. Por último, el resto de los modelos tuvó un excelente desempeño sin llegar a un sobre ajuste: del modelo *XGBClassifier* era de esperarse pues se trata de un modelo robusto de *Machine Learning*, el modelo de *Transfer Learning* con *VGG16* suele tener resultados pobres, pero sorprendentemente, obtuvo buenos resultados y la CNN propuesta era un modelo de complejidad media y de tamaño mediano que se adaptó desde cero a los datos de entrenamiento. Este último modelo es un claro ejemplo del poder computacional de los modelos de *Deep Learning* a la hora de realizar tareas de clasificación, incluso con imágenes.

Algo que nos pareció pertinente comentar, es la homogeneidad de las imágenes en el *dataset*. Si uno observa ejemplos de imágenes de las 3 clases, se puede observar que absolutamente todas parecen tener exactamente el mismo acercamiento en el microscopio. Lo que podría impedir una mayor generalización de los modelos al intentar predecir imágenes tomadas con diferentes objetivos (*zoom*) en el microscopio, sin importar el tipo de tumor de la fotografía.

Al haber trabajado con un *dataset* que no es exclusivo del cáncer de pulmón, pues cuenta también con imágenes de cáncer de colon, fue complicado hallar estudios similares al nuestro en cuanto a modelos de clasificación compete. Pero se logró hallar estudios que trabajaron, al menos una parte de estos, únicamente con cáncer de pulmón, he aquí un cuadro comparativo entre esos modelos y los propuestos en este proyecto:

Modelo	Accuracy	Precision	Recall	F1-score
SVC	0.88	0.88	0.88	0.88
Regresión Logística	0.68	0.68	0.69	0.68
XGBClassifier	**0.90**	**0.90**	**0.90**	**0.90**
TL MobileNetV2	0.99	0.99	0.99	0.99
TL VGG16	**0.97**	**0.97**	**0.97**	**0.97**
TL ResNetRS101	0.63	0.63	0.63	0.63
FT ResNetRS101	0.72	0.72	0.73	0.72
CNN propuesta	0.91	0.91	0.91	0.91
CNN (**CNN1**)	0.972	0.9733	0.9733	0.9733
CNN (**CNN2**)	0.9789	-	-	-
XGBClassifier (**modelo˙XGBClassifier**)	0.9953	0.9933	0.9933	0.9933

Tabla comparativa de métricas de los modelos propuestos y de otros estudios

Podemos observar que las métricas del modelo de *Transfer Learning* con *MobileNetV2*, superó los modelos de CNN propuestos en otros estudios. Pero como se señaló anteriormente, se teme un sobre ajuste de este modelo, por lo que consideraremos el segundo mejor modelo, el modelo de *Transfer Learning* con *VGG16*. Este modelo tiene aproximadamente las mismas métricas que los modelos propuestos en otros estudios, por lo que nuestro modelo está dentro de lo esperado en la literatura. Por otro lado, para el modelo de *XGBClassifier*, el modelo propuesto en otro estudio tuvo métricas casi perfectas y el propuesto en este proyecto tuvo métricas peores en comparación, aunque muy buenas. Teniendo en cuenta que la diferencia entre ambos modelos es de menos del 10 %, podemos decir que nos encontramos dentro del desempeño esperado.

Si hablamos de la aplicación de estos modelos en la práctica profesional, existen limitaciones. Se habló previamente de la homogeneidad de las imágenes del *dataset*, parece que todas las fotografías fueron tomadas con el mismo objetivo del microscopio (*zoom*). Lo que implica que los modelos solo saben clasificar imágenes con este grado de acercamiento, hipótesis que se comprobó al intentar predecir imágenes ajenas al *dataset* con un grado de acercamiento diferente. Los modelos tuvieron una buena precisión con imágenes de objetivo similar, pero una muy baja con cualquier otro objetivo. Por lo que no se recomendaría utilizarlos en la práctica profesional, pues existiría una probabilidad importante de una negligencia médica.

Podemos decir con entusiasmo que existe oportunidad de mejora en el desempeño o generalización de estos modelos. Concerniente a los modelos de *Machine Learning*, se pueden explorar otros modelos como *KNN* o *RandomForest* y realizar *GridSearchCV* más extensos, con una cantidad de parámetros mayor, en consecuencia de combinaciones. Por el lado de los modelos de *Deep Learning*, podemos en primer lugar realizar una validación cruzada a los modelos con posible sobre ajuste. Con estos modelos se pueden realizar diversas propuestas de *CNN* tanto propias como con técnicas de *Transfer Learning* y *Fine Tuning*, y dado que existe una buena cantidad de modelos base, y podemos "congelar" y "descongelar" pesos de manera arbitraria, las posibilidades son abundantes.

Otra forma de mejorar la generalización de los modelos es mejorar la cantidad, y en este caso la diversidad de los datos. Ya se habló repetidamente de la homogeneidad existente en el *dataset*. Agregar una mayor variabilidad de imágenes histopatológicas de estos tipos de cáncer, en particular, con diferentes objetivos (*zoom*), enriquecería mucho los datos, provocando una probable mejoría en el desempeño de los modelos.

Por último leyendo la literatura, se pueden usar técnicas de selección de características para mejorar el desempeño de los modelos. También se realizó una "normalización" de las imágenes dividiendo el valor de los píxeles entre 255, pero en la librería de *tensorflow* existen otros métodos y funciones que se podrían usar para realizar una normalización más extensa, lo que podría generar un mayor desempeños de los modelos.

En conclusión, en este proyecto pudimos aplicar los conocimientos adquiridos a lo largo del *Samsung Innovation Campus* y también demuestra el potencial de los modelos de *Machine Learning* y *Deep Learning* en la clasificación de tipos de cánceres de pulmón, mostrando que la inteligencia artificial puede ser una herramienta poderosa en el diagnóstico médico. Sin embargo, aún quedan desafíos por resolver antes de su aplicación en entornos clínicos reales.

El cáncer de pulmón representa un reto crítico para la salud pública tanto en México como a nivel mundial. Aunque en México ocupa el séptimo lugar en cuanto a frecuencia, se le reconoce como el tumor más letal, siendo la principal causa de muerte por cáncer. De acuerdo con el Dr. Omar Macedo Pérez, oncólogo del Instituto Nacional de Cancerología (INCan), anualmente fallecen cerca de ocho mil mexicanos por esta enfermedad, y se registran alrededor de nueve mil casos nuevos, de los cuales el 85 % están relacionados con el consumo de tabaco.

Este panorama se refleja también a nivel internacional. En la última década, la incidencia del cáncer de pulmón ha incrementado en un 30 %, con más de 2 millones de casos nuevos estimados tan solo en el año 2020 y alrededor de 1.8 millones de muertes. Si esta tendencia continúa, se espera que para el año 2030 se reporten más de 2.7 millones de casos nuevos anualmente. Estas cifras posicionan al cáncer de pulmón como una de las enfermedades oncológicas más agresivas y demandantes de atención prioritaria.

En el contexto mexicano, según datos recientes, en 2020 se registraron 7,811 nuevos casos y 6,733 muertes atribuibles a esta enfermedad. Tales datos refuerzan la relevancia del presente trabajo y la necesidad de contar con herramientas predictivas precisas que permitan anticipar el comportamiento de esta enfermedad en el futuro.

A partir del desarrollo de este proyecto y con base en la metodología aplicada para los modelos de predicción —específicamente, la regresión lineal y la red neuronal recurrente (LSTM)— se puede afirmar que los resultados obtenidos son coherentes con las estadísticas actuales. Las predicciones realizadas por ambos modelos se aproximan significativamente a las cifras reales reportadas en el país, lo que respalda su utilidad y validez para escenarios futuros.

1. La evolución de las muertes por cáncer de pulmón en México sigue una tendencia lineal ascendente, reflejando un incremento sostenido año tras año. Esta tendencia podría agravarse en el futuro debido al creciente consumo de productos alternativos que contienen nicotina, como los vapeadores electrónicos y los sobres de nicotina (snus), los cuales representan un riesgo emergente para la salud pública.

2. El modelo de regresión lineal mostró un buen desempeño en términos de precisión, aprovechando la estructura temporal de los datos disponibles. No obstante, al incrementar el volumen de datos y al incorporar variables adicionales relevantes —como edad, género, comorbilidades, o hábitos de consumo—, las redes neuronales recurrentes podrían superar el rendimiento de los modelos tradicionales, al capturar patrones no lineales y relaciones más complejas en la evolución de la mortalidad.

Estos hallazgos demuestran el potencial de las técnicas de aprendizaje automático y profundo como herramientas valiosas en el análisis epidemiológico y la toma de decisiones en salud pública. Su implementación puede contribuir significativamente al diseño de políticas preventivas y a la asignación más efectiva de recursos sanitarios en el país.

Durante la investigación que se realizó para este trabajo se encontraron diversos artículos en los cuales se mencionan algunas de las principales causas del cáncer del pulmón, aunque algunos mencionan información relevante, como el factor de riesgo que presentan el alcohol y el tabaco también encontramos información sobre cáncer de pulmón en personas que jamás han fumado. Esta diversidad en las causas que se discuten nos llevo a aplicar los métodos aprendimos en el *Samsung Innovation Campus* para averiguar si podíamos agrupar a los pacientes en base al nivel de su enfermedad con las características recopiladas.

En la sección de resultados podemos apreciar que nuestro objetivo no es posible, al menos con los datos y técnicas que trabajamos, en la matriz de confusión y el reporte de clasificación para *DBSCAN* se aprecia que el modelo clasifica casi a todos los pacientes con un nivel bajo de la enfermedad lo cual ocasiona que tenga una precisión sumamente baja (36 %).

Si bien, para el modelo de agrupamiento con *k-means* se observa una mejoría significativa, sigue sin ser adecuada, pues en particular el modelo agrupa de manera muy pobre a los pacientes con un nivel alto de cáncer de pulmón, y esto es un elemento de suma importancia, además de que el agrupamiento para los pacientes con niveles bajos y medios de cáncer tampoco es lo suficientemente preciso. Aunque se podría considerar que el modelo tiene cierta utilidad, puesto que el recall para el nivel *Low* fue de 76 % lo que nos indica que el 76 % de los pacientes que tienen un nivelo bajo de cáncer de pulmón fueron correctamente clasificados, esto sigue estando lejos de una precisión aceptable, y en lo general, la precisión del agrupamiento con *k-means* (67 %) nos permite descartarlo.