MEMORIA TRABAJO FIN DE MÁSTER

# Clasificación de Cáncer de Mama de tipo Carcinoma Ductal Invasivo: Un enfoque para la Mejora del Diagnóstico Temprano


Componentes:

Álvaro Ladrón de Guevara Garcés

Ángel Martín Heras

Diego Muñoz Herranz

Tutores:

Teno González Dos Santos

David Sanz Díaz


Madrid, 7 de julio 2023

# RESUMEN

El objetivo principal de este proyecto es desarrollar un modelo capaz de identificar si las masas tumorales son benignas o malignas en el cáncer de mama. Para lograrlo, se plantean una serie de objetivos específicos. En primer lugar, se recopilará y preparará un dataset de imágenes médicas etiquetadas que contenga casos positivos y negativos de cáncer de mama. A continuación, se utilizarán modelos preentrenados para reducir el volumen de datos del dataset inicial.

Se llevará a cabo un análisis exploratorio de los datos para comprender mejor sus características. Luego, se construirán distintos modelos basados en redes convolucionales (CNN) capaces de detectar el cáncer de mama en imágenes médicas, extrayendo propiedades relevantes de las mismas.

El modelo se entrenará utilizando los datos preparados y se ajustarán los parámetros de la red convolucional, midiendo la precisión del algoritmo con el conjunto de datos de entrenamiento. Se buscarán hiperparámetros (neuronas, capas, iteraciones) adecuados para las imágenes. Posteriormente, se validarán los resultados utilizando un conjunto de datos reservado para pruebas.

# ABSTRACT

The main objective of this project is to develop a model capable of identifying whether tumor masses in breast cancer are benign or malignant. To achieve this, a series of specific objectives are proposed. Firstly, a labeled dataset of medical images containing positive and negative cases of breast cancer will be collected and prepared. Pretrained models will then be utilized to reduce the volume of data from the initial dataset.

An exploratory data analysis will be conducted to better understand the characteristics of the data. Subsequently, different models based on convolutional neural networks (CNNs) will be constructed to detect breast cancer in medical images by extracting relevant properties from them.

The model will be trained using the prepared data, and the parameters of the convolutional network will be adjusted, measuring the algorithm's accuracy with the training dataset. Suitable hyperparameters such as neurons, layers, and iterations will be sought for the images. Subsequently, the results will be validated using a separate dataset reserved for testing purposes.

# ÍNDICE

# 1. INTRODUCCIÓN

El cáncer de mama se posiciona como el tipo de tumos maligno más prevalente en la población femenina[1],[2]. Es de suma importancia reconocer la relevancia de la detección temprana de esta enfermedad mediante la implementación de métodos como la mamografía y otras técnicas, ya que su aplicación incide directamente en el pronóstico del paciente.

En este sentido, los avances en el campo de la inteligencia artificial han demostrado su capacidad para contribuir a la detección temprana de este tipo de tumores, centrándonos particularmente en el carcinoma ductal invasivo (IDC, por sus siglas en inglés) [3]

Este cáncer comienza en el revestimiento de los conductos galactóforos (conductos de la mama) los cuales son responsables de transportar la leche desde los lóbulos mamarios hasta el pezón durante la lactancia[1]. Estos conductos están revestidos por células epiteliales que normalmente se dividen y se renuevan de manera controlada. Sin embargo, en ocasiones, debido a diversos factores, estas células pueden experimentar cambios en su material genético, lo que desencadena un proceso de transformación maligna.

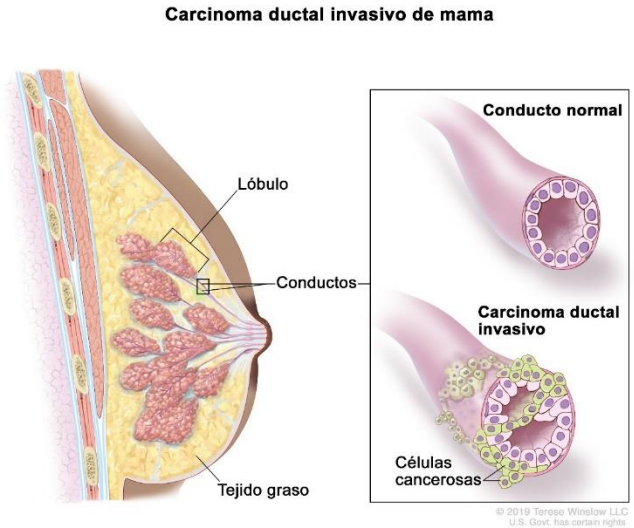



*Figura 1. Formación del carcinoma*

Al utilizar redes neuronales convolucionales en imágenes de IDC, se logra una mayor precisión y sensibilidad en la identificación de posibles tumores. Estas redes son capaces de analizar minuciosamente los detalles de las imágenes provenientes de escáneres y mamografías, destacando regiones sospechosas y proporcionando una evaluación más confiable

# 2. ESTADO ACTUAL DE LA INVESTIGACIÓN

El uso de machine learning en el estudio de imágenes de cáncer de mama se está volviendo cada vez más frecuente, ofreciendo nuevas oportunidades para la detección temprana y el análisis preciso de la enfermedad.

En general, para cualquier algoritmo que se quiera implementar para ésta tarea se utiliza el enfoque CLAHE[1],[2] para mejorar la calidad de las imágenes y eliminar el posible ruido.

Adicionalmente, el rango de algoritmos que se usan para la investigación de este campo es muy grande, usando modelos de SVM, Random Forest y redes neuronales, aplicados tras identificar y delimitar las regiones de interés en las imágenes con las que se van a entrenar dichos modelos.

En el campo del estudio de imágenes de cáncer de mama, los avances en el uso de machine learning han permitido no solo el desarrollo de modelos para la detección temprana de la enfermedad, sino también para determinar la etapa en la que se encuentra. Además, estos modelos han demostrado ser eficaces en evaluar la efectividad del tratamiento[3]. Mediante el análisis de imágenes médicas, se pueden extraer características relevantes y utilizar algoritmos de aprendizaje automático para realizar una evaluación precisa de la progresión del cáncer y para determinar la respuesta al tratamiento. Esta capacidad de los modelos de machine learning brinda nuevas oportunidades para un diagnóstico más preciso y una planificación de tratamiento personalizada en la lucha contra el cáncer de mama.

# 3. SOFTWARE UTILIZADO

Para realizar el presente trabajo, utilizaremos el lenguaje de programación Python. Este es un lenguaje de programación interpretado y de alto nivel que se caracteriza por su legibilidad y simplicidad. Por defecto Python[5] lleva consigo una gran cantidad de módulos y paquetes lo que le permite ser muy versátil en todos los ámbitos, al ser un lenguaje de código abierto (Open Source), permite que la comunidad de desarrolladores pueda crear nuevas librerías para mejorar su uso en algunos ámbitos. Algunas de ellas son las que trataremos en esta investigación, en particular las siguientes:

- TensorFlow[6]: framework que utilizaremos para construir y entrenar redes neuronales profundas.
- PyTorch[7]: este framework proporciona una interfaz flexible y fácil de usar para entrenar modelos.
- Keras[8]: framework de aprendizaje profundo.
- OpenCV[9]: es una biblioteca de visión por computadora y procesamiento de imágenes de código abierto. Proporciona una amplia gama de algoritmos y funciones para el procesamiento de imágenes y vídeos.
- Flask[10]: es un microframework web que permite construir aplicaciones web de manera rápida y sencilla. Es comúnmente usado para desarrollar prototipos, API (Aplication Programming Interfaces) y pequeñas webs.

Parte de estas librerías se usarán las librerías comunes como Pandas, Numpy, SKLearn, Pillow con el objetivo de realizar un análisis exhaustivo del proyecto.

El sistema de control de versiones Git[11] y la plataforma de alojamiento GitHub se han utilizado para gestionar y almacenar el código de este proyecto. Git es un sistema de control de versiones ampliamente utilizado que permite rastrear y administrar cambios en el código fuente a lo largo del tiempo. Proporciona un historial completo de todas las modificaciones realizadas en el código, lo que facilita la colaboración y el seguimiento de los cambios realizados por diferentes personas.

GitHub, por otro lado, es una plataforma basada en la nube que ofrece servicios de alojamiento para repositorios de Git. Permite a los desarrolladores almacenar, compartir y colaborar en proyectos de software de manera eficiente. GitHub proporciona una interfaz web que facilita la visualización y navegación del código, así como herramientas para gestionar problemas, solicitudes de extracción y otras funciones de colaboración.

Al utilizar Git y GitHub, el código del proyecto se guarda en un repositorio Git alojado en GitHub. Esto proporciona un entorno centralizado para el código, donde los desarrolladores pueden colaborar, realizar cambios, rastrear el historial de versiones y administrar el desarrollo del proyecto de manera eficiente.

# 4. INTRODUCCIÓN AL DATASET

El conjunto de datos de imágenes utilizado es el Invasive Ductal Carcinoma (IDC) Histology Image Dataset, el cual se utiliza para la detección y clasificación de cáncer de mama. Este conjunto de datos está compuesto por imágenes histopatológicas digitalizadas de muestras de tejido mamario, específicamente de casos de carcinoma ductal invasivo. El carcinoma ductal invasivo es el tipo más común de cáncer de seno, representando aproximadamente el 80% de los casos. Este tipo de carcinoma se origina en las células que revisten los conductos de leche en el seno y puede invadir la pared del conducto y crecer en los tejidos mamarios cercanos. Además, tiene la capacidad de propagarse a otras partes del cuerpo a través del sistema linfático y el torrente sanguíneo.

El conjunto de imágenes original consta de 162 muestras de cáncer de mama (BCa) escaneadas a 40x. A partir de este conjunto, se generaron un total de 277,524 imágenes, de las cuales 198,738 son negativas (benignas) y 78,786 son positivas (malignas).

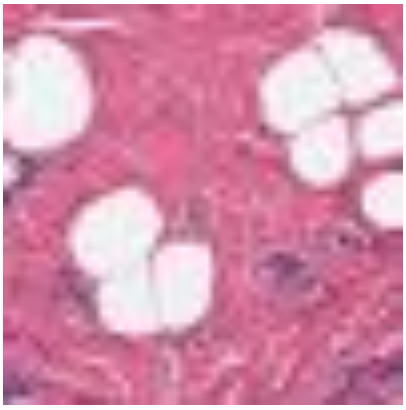


*Figura 2. Imagen de masa tumoral*

*Figura 3.Imagen de masa no tumoral*

Las imágenes se almacenan en carpetas que contienen el ID del paciente, y dentro de cada carpeta se encuentran dos subcarpetas identificadas como "0" o "1". Las imágenes ubicadas en la carpeta "0" corresponden a imágenes benignas, mientras que las imágenes en la carpeta "1" corresponden a imágenes malignas.

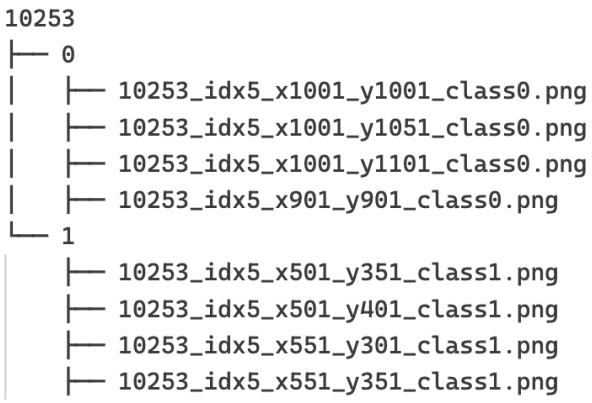



*Figura 4. Estructura de Carpetas*

Con el objetivo de mejorar nuestro trabajo, hemos tomado la decisión de abordar el dataset de manera más cuidadosa. Para ello, hemos realizado la conversión de las imágenes a archivos .dat, lo que nos permite evitar conversiones frecuentes al utilizarlos.

Además, hemos procedido a dividir este conjunto de datos en dos conjuntos: X_train y X_test. Con el fin de asegurar una evaluación adecuada de nuestro modelo, hemos asignado el 30% de los datos al conjunto de validación o pruebas.

El código para el preprocesamiento de datos se puede encontrar en el siguiente enlace: https://github.com/diego-tech/TFM_Breast_Cancer_Classification/tree/main/Code/Preprocessing.

# 5. TRANSFER LEARNING

El aprendizaje por transferencia, o transfer learning en inglés, es un enfoque poderoso en el campo de la inteligencia artificial que ha demostrado su eficacia en la capacitación de nuevos modelos. En lugar de entrenar un modelo desde cero en un conjunto de datos específico, el transfer learning permite aprovechar el conocimiento adquirido previamente por modelos de inteligencia artificial ya entrenados en conjuntos de datos más grandes y diversos.

Una de las principales ventajas del transfer learning es que permite el entrenamiento de manera más eficiente, especialmente cuando los datos de entrenamiento son limitados. Los modelos pre-entrenados, como las redes neuronales convolucionales (CNN) utilizadas para la visión por computadora o los modelos de lenguaje para el procesamiento del lenguaje natural, han sido entrenados en conjuntos de datos masivos y pueden capturar características generales y representaciones significativas de los datos.

Cuando se utiliza el transfer learning, se toma un modelo pre-entrenado y se ajusta os se "descongela" algunas de sus capas para adaptarse a un nuevo conjunto de datos objetivo. Las capas iniciales, que aprenden características básicas y de bajo nivel, suelen mantenerse intactas, mientras que las capas posteriores se adaptan al nuevo dominio o tarea específica.

Esto permite que el modelo aproveche el conocimiento previo para extraer características relevantes y acelerar el proceso de entrenamiento.

Además de la eficiencia en el entrenamiento, el transfer learning también puede ayudar a mejorar el rendimiento de los modelos. Los modelos pre-entrenados han aprendido representaciones útiles de datos complejos y pueden capturar patrones que son transferibles a nuevas tareas. Al ajustar estos modelos en conjuntos de datos más pequeños y específicos, se puede lograr un rendimiento superior en comparación con el entrenamiento desde cero.

Otra de las ventajas del transfer learning en su capacidad para abordar problemas de falta de datos. En muchas aplicaciones de inteligencia artificial, la disponibilidad de datos de entrenamiento puede ser limitada. Sin embargo, los modelos pre-entrenados han sido entrenados en conjuntos de datos más grandes, lo que les permite capturar conocimientos generales y patrones relevantes. Al ajustar estos modelos en conjuntos de datos más pequeños, se puede superar la falta de datos y lograr resultados satisfactorios.

En resumen, el transfer learning es una técnica valiosa en el campo de la inteligencia artificial que permite aprovechar el conocimiento previo adquirido por modelos pre-entrenados. Esta técnica

ofrece eficiencia en el entrenamiento, mejora del rendimiento y capacidad para abordar problemas de falta de datos.

Con relación al caso de uso en cuestión, hemos desarrollado tres algoritmos basados en aprendizaje por transferencia. Esto nos ha brindado la capacidad de reducir significativamente el tiempo de cálculo necesario para utilizar y probar los modelos.

# 6. PRESENTACIÓN DE LOS MODELOS

Las redes neuronales convolucionales (CNN de ahora en adelante) son un tipo especializado de arquitectura de redes neuronales diseñadas específicamente para el procesamiento de datos de imágenes y reconocimiento visual. Se han convertido en una herramienta poderosa en el campo del aprendizaje automático y la visión por computadora, permitiendo tareas como la clasificación de imágenes, detección de objetos, segmentación semántica y más.

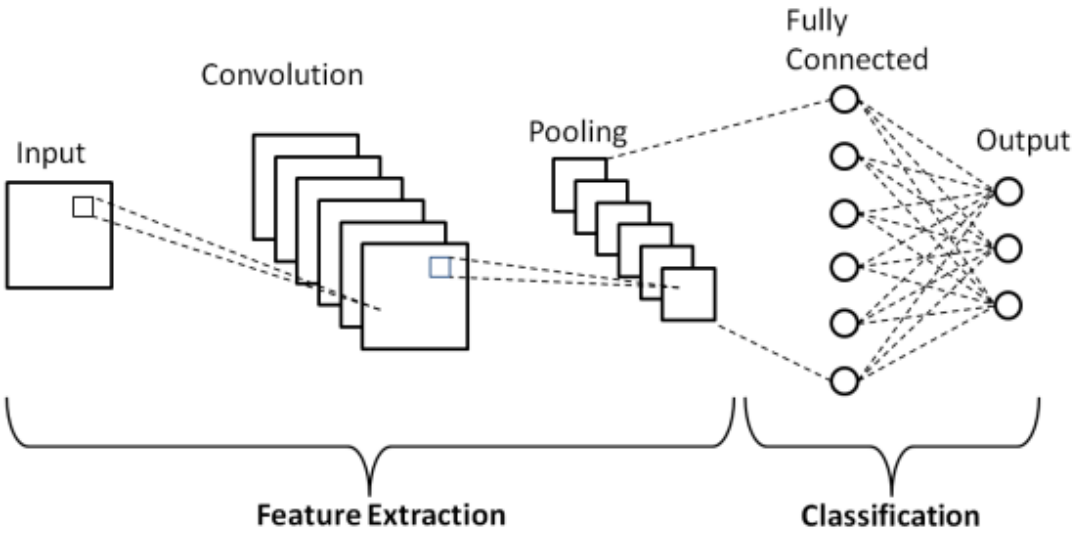



*Figura 5. Diagrama de una CNN*

Definimos ahora los conceptos básicos de las CNN y otros que serán mencionados durante el presente trabajo.

**Convolución** (convolution): Consiste en aplicar a una imagen un filtro para extraer características especiales. Este filtro se desliza sobre la imagen, realizando multiplicaciones punto a punto entre sus valores y los valores correspondientes de la región de la imagen cubierta por el filtro. Luego, se suman todos los productos resultantes para obtener un único valor que representa la activación

de la neurona convolucional y, tras aplicar la convolución por toda la imagen, se creará un mapa de características.

Mediante las distintas capas se aplicarán convoluciones y se descubrirán patrones (como por ejemplo bordes o texturas) que facilitarán la tarea de la clasificación de una imagen.

**Agregación** (pooling): Operación de las CNN para reducir la dimensionalidad de los mapas de características producidos durante la etapa de convolución. Lo que se hace en esta fase es agrupar las activaciones de neuronas vecinas para así conservar la actividad más relevante (es decir, las características más relevantes).

**Fully Connected** (completamente conectado): Las capas completamente conectadas son capas en las redes neuronales donde todas las neuronas están conectadas a todas las neuronas de la capa anterior. Estas capas son responsables de combinar las características extraídas por las capas convolucionales y de realizar la clasificación o regresión final en la red neuronal.

Función de activación: Transmite la información generada por la combinación lineal de los pesos y las entradas en una red neuronal. Por tanto, es la forma de transmitir la información por las conexiones de salida.

**Rectifiied Lineal Unit** (ReLU): Función de activación que funciona anulando los valores de entrada negativos y sin realizar transformaciones sobre los positivos. Es decir, devuelve el máximo entre 0 y el valor de la entrada. Esta función destaca por su buen desempeño con imágenes y redes convolucionales, y suele usarse en las capas ocultas.

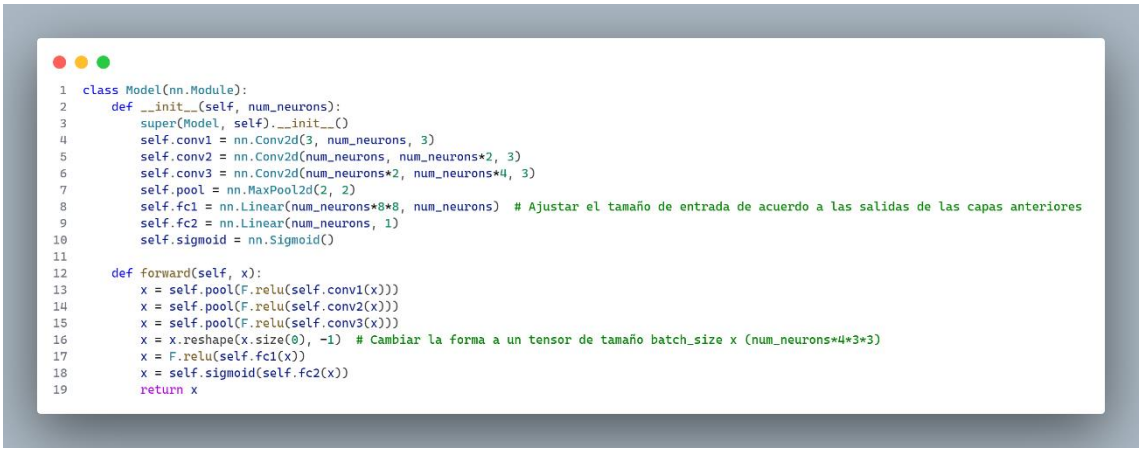
**Sigmoide**: Función de activación que transforma los datos de entrada a una escala entre 0 y 1. De esta forma, los valores muy bajos tienden de manera asintótica a 0, mientras que los más elevados lo hacen a 1. Es útil en los modelos de clasificación binaria y suele usarse en las capas de salida.

En este trabajo, nos enfocaremos en el desarrollo de un modelo clásico de red convolucional, así como en los modelos que utilizan Transfer Learning mencionado anteriormente. Estos son:

- CNN clásica con tres y cuatro capas de convolución.
- Modelo basado en VGG19.
- Modelo basado en ResNet50.
- Modelo basado en DenseNet201.

## 6.1. MODELO CLÁSICO CNN

Las CNN que hemos utilizado en el modelo constan de la siguiente estructura:



```python
class Model(nn.Module):
    def __init__(self, num_neurons):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(3, num_neurons, 3)
        self.conv2 = nn.Conv2d(num_neurons, num_neurons*2, 3)
        self.conv3 = nn.Conv2d(num_neurons*2, num_neurons*4, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(num_neurons*8*8, num_neurons)  # Ajustar el tamaño de entrada de acuerdo a las salidas de las capas anteriores
        self.fc2 = nn.Linear(num_neurons, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.reshape(x.size(0), -1)  # Cambiar la forma a un tensor de tamaño batch_size x (num_neurons*4*3*3)
        x = F.relu(self.fc1(x))
        x = self.sigmoid(self.fc2(x))
        return x
```

*Figura 6. Código de la CNN de 3 capas del modelo.*

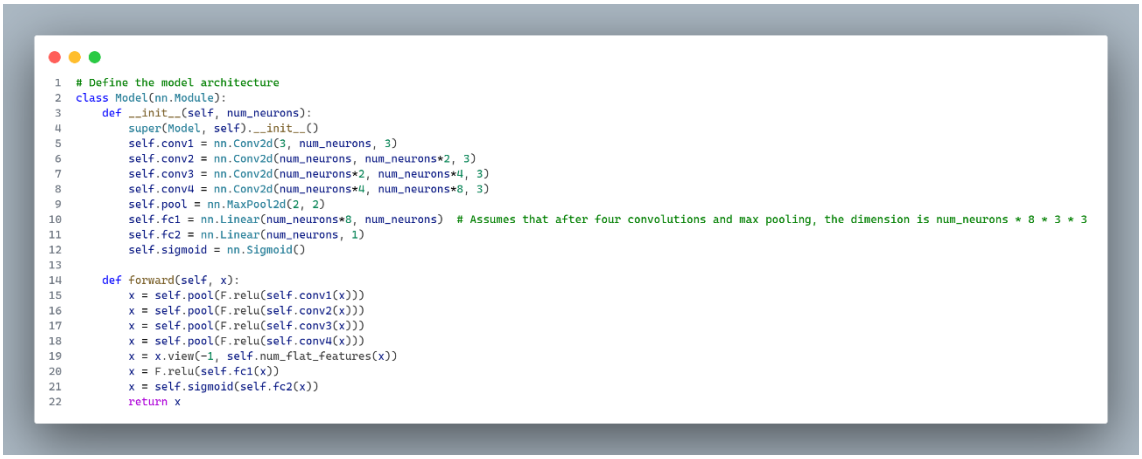
Hacemos uso de tres capas de convolución en 2 dimensiones, cada una con 3 parámetros: el primer parámetro es el número de canales de entrada, el segundo es el número de neuronas (filtros) por capa y el tercero es el tamaño del filtro (en este caso será 3x3, es decir, los filtros se irán aplicando en regiones de 3x3 pixeles y recorriendo toda la imagen).

También existe una capa de pooling con dos parámetros: el primero representa el tamaño de la ventana de pooling – la capa busca el valor máximo dentro de una región de 2x2 pixeles en cada mapa de características y el segundo parámetro es el paso, en este caso 2 lo que indica que esta ventana se moverá de 2 en 2 pixeles para ir aplicando.

Además, utilizamos dos capas fully connected con dos parámetros cada una, dimensión de entrada y dimensión de salida de la capa.

En la parte del forward tenemos el llamado a esas capas, utilizamos el método reshape para cambiar la forma al tensor que introduzcamos y por último el sigmoide, que nos dará la probabilidad de que la etiqueta que ha dado la CNN sea cierta.

Además, hemos hecho otra red neuronal convolucional con 4 capas:



```python
# Define the model architecture
class Model(nn.Module):
    def __init__(self, num_neurons):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(3, num_neurons, 3)
        self.conv2 = nn.Conv2d(num_neurons, num_neurons*2, 3)
        self.conv3 = nn.Conv2d(num_neurons*2, num_neurons*4, 3)
        self.conv4 = nn.Conv2d(num_neurons*4, num_neurons*8, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(num_neurons*8, num_neurons)  # Assumes that after four convolutions and max pooling, the dimension is num_neurons * 8 * 3 * 3
        self.fc2 = nn.Linear(num_neurons, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = self.sigmoid(self.fc2(x))
        return x
```

*Figura 7. Código de la CNN con 4 capas.*

De forma análoga, esta CNN tiene 4 capas de convolución con distintos parámetros, una capa de pooling y dos capas fully connected con diferente dimensión de entrada al modelo de 3 capas. El rendimiento de los modelos y la comparación se analizará en capítulos posteriores.

## 6.2. MODELO BASADO EN ResNet50

El modelo ResNet50 es otro modelo de red neuronal convolucional ampliamente reconocido en el campo del reconocimiento visual. Fue propuesto por el equipo de investigación de Microsoft en el año 2015 como una solución innovadora para abordar el desafío de entrenar redes muy profundas sin sufrir degradación en el rendimiento.

El nombre "ResNet50" se deriva de "Residual Network 50", lo cual hace referencia a su arquitectura residual y al número de capas que lo componen. La arquitectura residual se basa en la idea de que las capas adicionales de una red neuronal deberían poder aprender las diferencias o "residuos" entre las representaciones de entrada y salida de las capas anteriores, en lugar de tener que aprender completamente nuevas representaciones. Esta idea se implementa mediante la introducción de conexiones de salto o conexiones residuales en la red, que permiten que los gradientes fluyan directamente a través de las capas sin obstáculos.

Este modelo consta de un total de 50 capas, incluyendo capas convolucionales, capas de agrupación y capas completamente conectadas. Se caracteriza por su estructura en bloques, donde cada bloque contiene múltiples capas convolucionales. La clave de la arquitectura residual es la inclusión de bloques residuales, que utilizan conexiones de salto para permitir que los gradientes se propaguen de manera más eficiente y ayudar a aliviar el problema de degradación del rendimiento en redes muy profundas.

La principal ventaja del modelo ResNet50 es su capacidad para entrenar y aprovechar redes neuronales extremadamente profundas. A medida que se agregan más capas a la red, se espera que el rendimiento mejore continuamente en lugar de empeorar, gracias a las conexiones residuales.

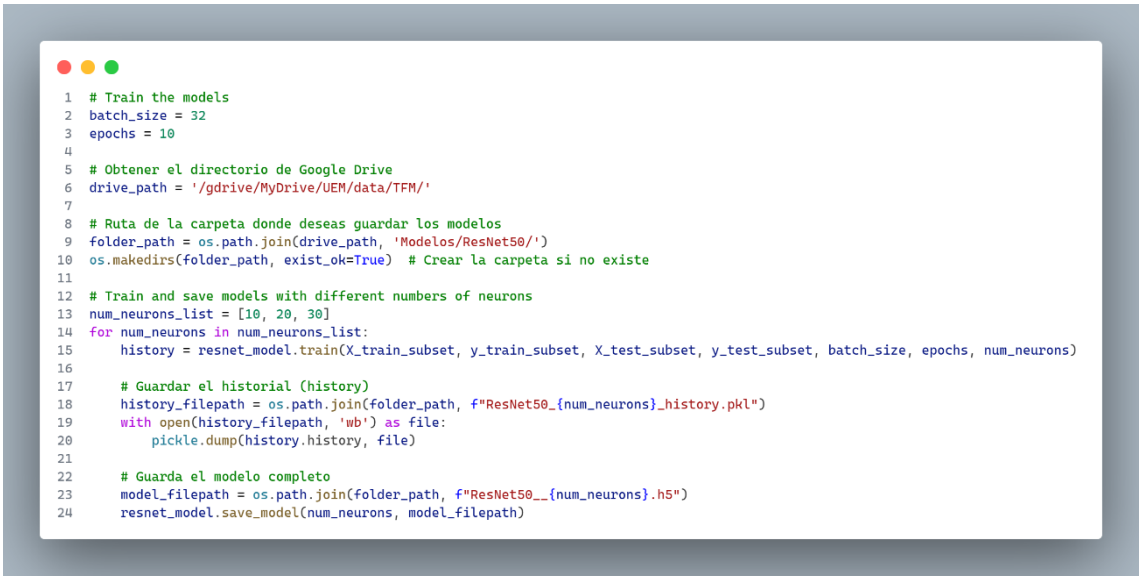
Una vez explicado el modelo base, procedemos a introducir el modelo creado. Este consta de las siguientes características:

```python
class BreastCancerModel:
    def __init__(self, name, base_model, input_shape):
        self.name = name
        self.base_model = base_model
        self.input_shape = input_shape
        self.models = {}

    def build_model(self, num_neurons):
        x = Flatten()(self.base_model.output)
        x = Dense(num_neurons, activation='relu')(x)
        x = Dense(num_neurons // 2, activation='relu')(x)
        predictions = Dense(1, activation='sigmoid')(x)
        model = Model(self.base_model.input, predictions)
        return model

    def train(self, X_train, y_train, X_val, y_val, batch_size, epochs, num_neurons):
        model = self.build_model(num_neurons)
        model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
        history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
                            validation_data=(X_val, y_val))
        self.models[num_neurons] = model
        return history

    def save_model(self, num_neurons, filepath):
        if num_neurons in self.models:
            model = self.models[num_neurons]
            model.save(filepath)
            print("Model saved successfully.")
        else:
            print("Model not found for the specified number of neurons.")

# Define the input shape
input_shape = (50, 50, 3)  # Adjust the number of channels if necessary

# Create ResNet50 model
resnet_base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)
resnet_model = BreastCancerModel("ResNet50", resnet_base_model, input_shape)

```

*Figura 8. Código Modelo ResNet50*

Utilizamos tres capas de convolución, adicionales a las capas del modelo base. Esto se define en el apartado build_model. Las dos primeras capas son capas densas con número de neuronas especificado por num_neurons, utilizando dicha cantidad de neuronas para la primera y la mitad para la segunda. Estas utilizan función de activación ReLU (Rectified Linear Unit). La última

capa es de salida y consta de una neurona, y hace uso de una función de activación sigmoide, acorde a la clasificación binaria que nos ocupa.
Se han utilizado 10 épocas para el entrenamiento, y valores de 10, 20 y 30 para el parámetro num_neurons. El modelo espera recibir imágenes de 50x50 píxeles y tres canales (RGB), de ahí el input_shape definido.
Creamos el modelo ResNet50 preentrenado con pesos de Imagenet y excluyendo la capa superior, mediante include_top=False.



```python
# Train the models
batch_size = 32
epochs = 10

# Obtener el directorio de Google Drive
drive_path = '/gdrive/MyDrive/UEM/data/TFM/'

# Ruta de la carpeta donde deseas guardar los modelos
folder_path = os.path.join(drive_path, 'Modelos/ResNet50/')
os.makedirs(folder_path, exist_ok=True)  # Crear la carpeta si no existe

# Train and save models with different numbers of neurons
num_neurons_list = [10, 20, 30]
for num_neurons in num_neurons_list:
    history = resnet_model.train(X_train_subset, y_train_subset, X_test_subset, y_test_subset, batch_size, epochs, num_neurons)

    # Guardar el historial (history)
    history_filepath = os.path.join(folder_path, f"ResNet50_{num_neurons}_history.pkl")
    with open(history_filepath, 'wb') as file:
        pickle.dump(history.history, file)

    # Guarda el modelo completo
    model_filepath = os.path.join(folder_path, f"ResNet50__{num_neurons}.h5")
    resnet_model.save_model(num_neurons, model_filepath)
```

*Figura 9. Código Entrenamiento ResNet50*

Ahora procedemos a entrenar el modelo, iterando sobre los números de neuronas establecidos.

## 6.3. MODELO BASADO EN DenseNet201

DenseNet201 es un modelo de red neuronal preentrenado que ha demostrado ser altamente eficaz en una variedad de tareas de visión por computadora. Fue introducido por Huang en 2017 como parte de la familia de redes DenseNet (redes densamente conectadas).
La característica distintiva de DenseNet201 es su estructura densamente conectada, que difiere de las arquitecturas de redes neuronales convencionales. En lugar de tener capas que se conectan de forma secuencial, DenseNet201 presenta conexiones directas entre todas las capas, permitiendo un flujo de información más eficiente y una mejor propagación de los gradientes durante el entrenamiento.
Estas conexiones directas hacen que el modelo sea extremadamente profundo y densamente conectado, lo que le permite capturar y combinar características en múltiples escalas y niveles

de abstracción. Esto resulta en una mayor capacidad para capturar detalles finos y características globales en una imagen, lo que puede ser beneficioso para tareas de clasificación, detección de objetos y segmentación semántica.

Otra característica clave de este modelo es su uso de bloques de capas conocidos como "bloques densos". Estos bloques están compuestos por capas de convolución y capas de agrupación, seguidas por una conexión directa que concatena la entrada con todas las salidas anteriores. Esto promueve el flujo de información denso en la red y facilita el aprendizaje de representaciones más ricas y robustas.

Este modelo ha sido entrenado en conjuntos de datos masivos, como el anteriormente mencionado ImageNet, lo que le permite aprender características generales y de alto nivel a partir de una amplia variedad de imágenes. Como resultado, se ha convertido en una opción popular para la transferencia de aprendizaje, donde se utiliza como punto de partida para adaptar y aplicar modelos preentrenados en tareas específicas con conjuntos de datos más pequeños.

Explicamos ahora nuestro modelo basado en DenseNet201.

```python
def build_model(backbone, num_neurons, lr=1e-4):
    model = Sequential()
    model.add(backbone)
    model.add(GlobalAveragePooling2D())
    model.add(Dropout(0.5))
    model.add(BatchNormalization())

    # Additional layers
    model.add(Dense(num_neurons, activation='relu'))

    model.add(Dense(1, activation='sigmoid'))  # Cambio a 1 salida con activación sigmoidal

    model.compile(
        loss='binary_crossentropy',  # Cambio a binary_crossentropy
        optimizer=Adam(learning_rate=lr),
        metrics=['accuracy']
    )

    return model
```

*Figura 10. Código Modelo DenseNet201*

Utitilizamos cinco capas de convolución, además de las correspondientes al modelo base DenseNet201. Estas son:

- Capa base (backbone). Corresponde al modelo DenseNet201
- Capa de promediado global (GlobalAveragePooling2D).
- Capa de Dropout (Dropout).
- Capa de normalización por lotes (BatchNormalization).

- Capa oculta (Dense). El total de neuronas depende del parámetro num_neurons y utiliza función de activación ReLU.
- Capa de salida (Dense). Contiene una sola neurona y utiliza función de activación sigmoide para la clasificación binaria.

Tal y como en el resto de los modelos, se espera recibir imágenes de 50x50 píxeles y tres canales (RGB). Creamos el modelo DenseNet201 preentrenado con los mismos parámetros que para ResNet50.

Definimos los parámetros utilizados por el modelo, destacando un total de diez épocas de entrenamiento y la realización de pruebas para 10, 20 y 30 neuronas en la capa Dense.

```python
for num_neurons in num_neurons_list:
    K.clear_session()
    gc.collect()

    # Build the model with different numbers of neurons
    model = build_model(resnet, num_neurons, lr)

    # Learning Rate Reducer
    learn_control = ReduceLROnPlateau(monitor='val_accuracy', patience=5,
                                      verbose=1, factor=0.2, min_lr=1e-7)

    # Checkpoint
    filepath = f"weights.best_{num_neurons}.hdf5"
    checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,
                                 save_best_only=True, mode='max')

    # Train the model
    history = model.fit(
        X_train_subset, y_train_subset,
        batch_size=BATCH_SIZE,
        epochs=epochs,
        validation_data=(X_test_subset, y_test_subset),
        callbacks=[learn_control, checkpoint]
    )

    # Save the history as a pickle file
    history_filepath = f"DenseNet201_history_{num_neurons}.pkl"
    with open(history_filepath, 'wb') as file:
        pickle.dump(history.history, file)

    # Save the trained model
    model_filepath = f"DenseNet201_Model_{num_neurons}.h5"
    model.save(model_filepath)
```

*Figura 11. Código Entrenamiento DenseNet201*

Por último, entrenamos el modelo iterando sobre los distintos números de neuronas establecidos.

## 6.4. MODELO BASADO EN VGG19

El modelo VGG-19 es un modelo de red neuronal convolucional ampliamente utilizado en el campo del reconocimiento visual. Fue desarrollado por el equipo de investigación de la Universidad de Oxford, conocido como Visual Geometry Group (VGG), en el año 2014.

El objetivo principal de VGG-19 es abordar el desafío de la clasificación de imágenes a gran escala.

Este modelo destaca por su arquitectura profunda y homogénea. Consiste en un total de 19 capas, incluyendo capas convolucionales y capas completamente conectadas (o fully-connected). La estructura se caracteriza por tener capas convolucionales con filtros de tamaño reducido (3x3) y una longitud de paso de convolución igual a 1, lo que permite capturar características detalladas de las imágenes. Además, utiliza capas de agrupación con ventanas de 2x2 y una longitud de paso de 2 para reducir la resolución espacial y aumentar la robustez a las deformaciones locales.

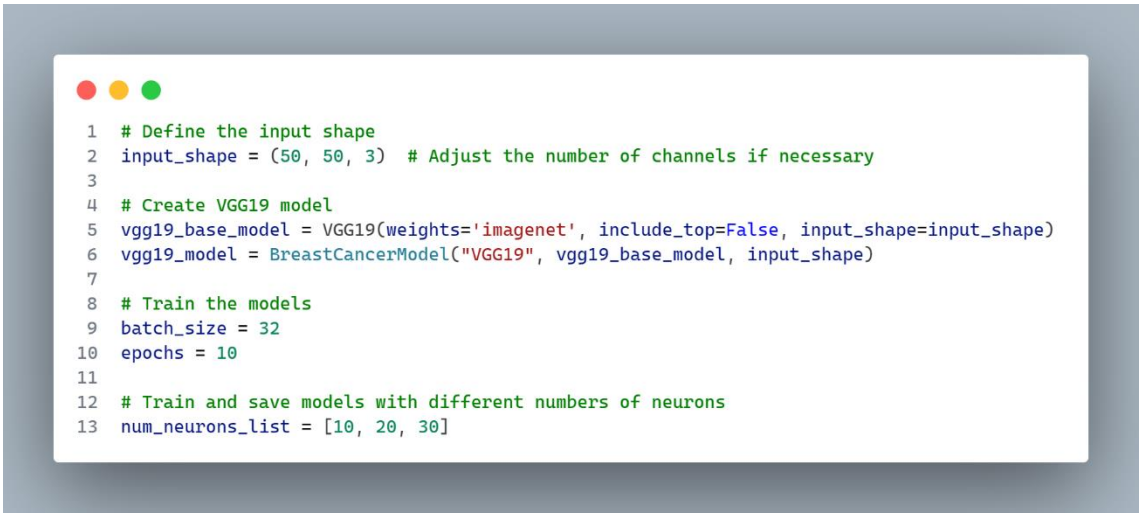
Ahora vamos a introducir nuestro modelo.

```python
class BreastCancerModel:
    def __init__(self, name, base_model, input_shape):
        self.name = name
        self.base_model = base_model
        self.input_shape = input_shape
        self.models = {}

    def build_model(self, num_neurons):
        x = Flatten()(self.base_model.output)
        x = Dense(num_neurons, activation='relu')(x)
        x = Dense(num_neurons // 2, activation='relu')(x)
        predictions = Dense(1, activation='sigmoid')(x)
        model = Model(self.base_model.input, predictions)
        return model

    def train(self, X_train, y_train, X_val, y_val, batch_size, epochs, num_neurons):
        model = self.build_model(num_neurons)
        model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
        history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
                            validation_data=(X_val, y_val))
        self.models[num_neurons] = model
        return history

    def save_models(self, filepath):
        with open(filepath, 'wb') as file:
            pickle.dump(self.models, file)

```

*Figura 12. Código Modelo VGG19*

El modelo hace uso de tres capas densas y una capa de aplanamiento, en total. Primero, la capa de aplanamiento toma la salida del modelo base, que será VGG19 en este caso. Después, el modelo pasa por dos capas densas con un número de neuronas dependientes del parámetro num_neurons, siendo este valor para la primera y la mitad para la segunda, tal y como ocurre con modelos anteriores. Ambas capas tienen función de activación ReLU. Por último, hay una capa densa de una sola neurona con función de activación sigmoide.

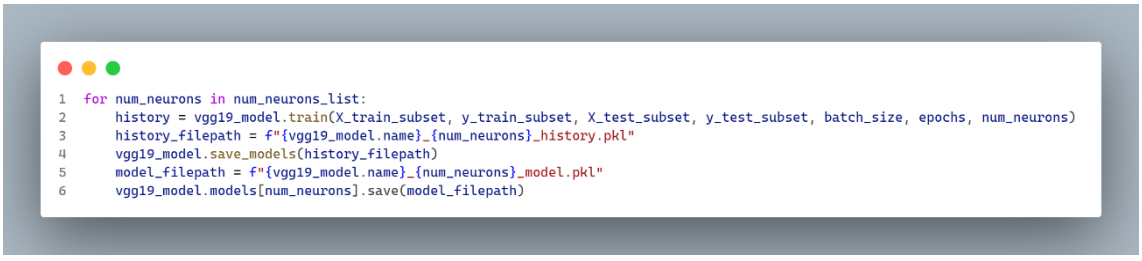
Todo esto queda recogido en el módulo build_model, que devuelve el modelo construido.



```python
# Define the input shape
input_shape = (50, 50, 3)  # Adjust the number of channels if necessary

# Create VGG19 model
vgg19_base_model = VGG19(weights='imagenet', include_top=False, input_shape=input_shape)
vgg19_model = BreastCancerModel("VGG19", vgg19_base_model, input_shape)

# Train the models
batch_size = 32
epochs = 10

# Train and save models with different numbers of neurons
num_neurons_list = [10, 20, 30]
```

*Figura 13. Código Modelo VGG19*

Análogamente a los otros modelos, en este apartado definimos el modelo base y los parámetros referentes a las épocas y neuronas a usar, que son los mismos que en el resto de los algoritmos basados en modelos preentrenados.



```python
for num_neurons in num_neurons_list:
    history = vgg19_model.train(X_train_subset, y_train_subset, X_test_subset, y_test_subset, batch_size, epochs, num_neurons)
    history_filepath = f"{vgg19_model.name}_{num_neurons}_history.pkl"
    vgg19_model.save_models(history_filepath)
    model_filepath = f"{vgg19_model.name}_{num_neurons}_model.pkl"
    vgg19_model.models[num_neurons].save(model_filepath)
```

*Figura 14. Código Entrenamiento VGG19*

Terminamos con el entrenamiento del modelo.

Además de VGG19, existe un modelo de Transfer Learning del equipo Visual Geometry Group denominado VGG16, el cual tiene los mismos fundamentos que el anterior, pero con la diferencia de utilizar 16 capas de convolución en lugar de 19. En el desarrollo del presente trabajo, se ha desarrollado un modelo basado en VGG16 de manera análoga a VGG19 y se han obtenido

resultados prácticamente idénticos. Por la similitud de resultados y de su desarrollo respecto al código, finalmente se ha omitido su introducción y despliegue.

# 7. RESULTADOS Y COMPARACIÓN DEL RENDIMIENTO ENTRE MODELOS

Para comparar los modelos vamos a utilizar dos argumentos:

- Precisión obtenida en el modelo tras el entrenamiento
- Curva ROC

Se define como precisión al porcentaje de casos que el modelo ha predicho de forma correcta, esto es: etiquetas 0 que ha predicho como 0 y etiquetas 1 que ha predicho como 1 (verdaderos positivos y verdaderos negativos).

$$\frac{\#VP + \#VN}{\#TOTAL}$$

Con #VP el número de verdaderos positivos y #VN el número de verdaderos negativos.

La curva ROC es una representación gráfica que muestra la capacidad discriminativa de un modelo de clasificación binaria. Esta curva traza los verdaderos positivos frente a los falsos positivos

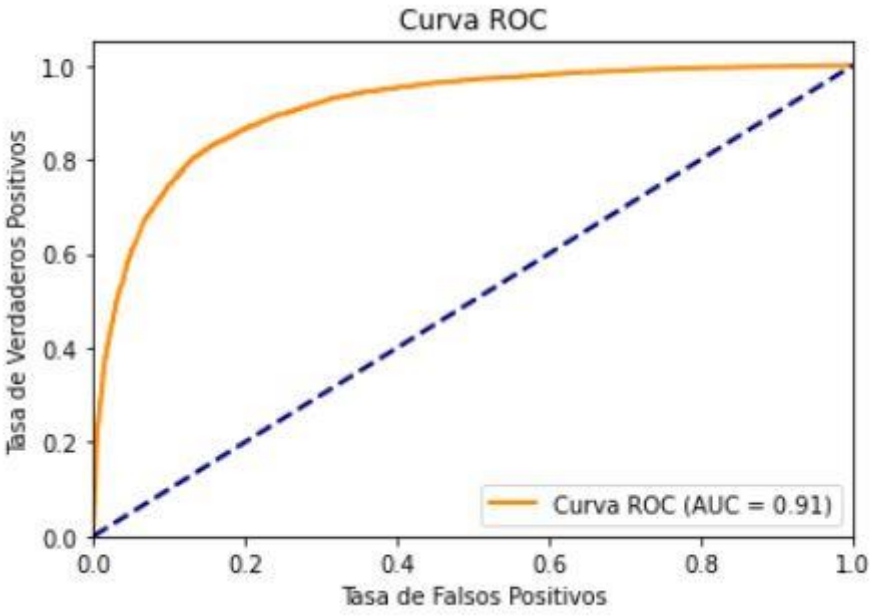



*Figura 15. Ejemplo de curva ROC para un modelo de Machine Learning*

Un parámetro relevante en relación con la curva ROC es el área dada bajo la curva (AUC). El área bajo la curva es una medida numérica que se utiliza para evaluar la capacidad de discriminación de un modelo de clasificación. El AUC se obtiene calculando el área bajo la curva ROC y su valor resultante varía entre 0 y 1. Un AUC igual a 0.5 indica que el modelo no tiene capacidad de discriminación y está clasificando al azar. Un AUC cercano a 1 indica un excelente poder de discriminación, lo que significa que el modelo es capaz de distinguir de manera efectiva entre las clases positivas y negativas. Por otro lado, un AUC cercano a 0 indica un mal rendimiento del modelo, ya que está clasificando de manera inversa o confundiendo las clases.

En cuanto a los parámetros utilizados en las redes en busca de los mejores resultados, hemos decidido considerar el número de iteraciones para entrenar el modelo (epoch), que estará en un rango de 1 a 10. También variaremos el número de neuronas, utilizando valores de 10, 20 y 30 para cada modelo. Además, tendremos en cuenta las peculiaridades específicas de cada modelo. Por ejemplo, crearemos una CNN con 3 y 4 capas, así como implementaremos una VGG19, DeseNet201 y ResNet50. Con los conceptos establecidos, vamos a ver uno a uno cada modelo y sus parámetros.

## 7.1. MODELO CLÁSICO CNN

Generamos una gráfica que muestra la precisión para cada modelo generado, considerando 10, 20 y 30 neuronas, junto con el número de iteraciones utilizadas para entrenarlos. Además, creamos una gráfica de la pérdida de validación correspondiente a cada modelo.

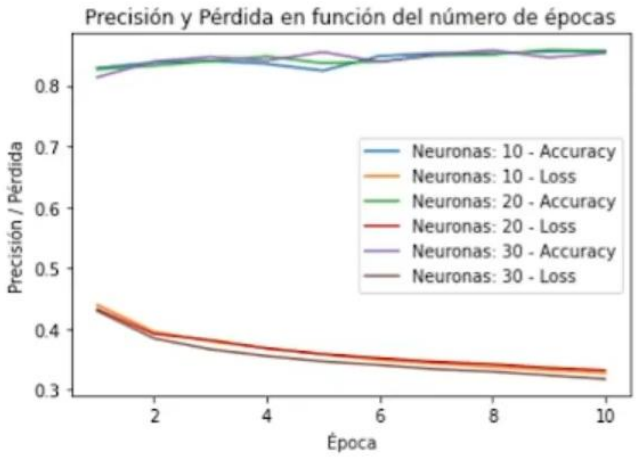



*Figura 16. Precisión y pérdida CNN 4 capas*

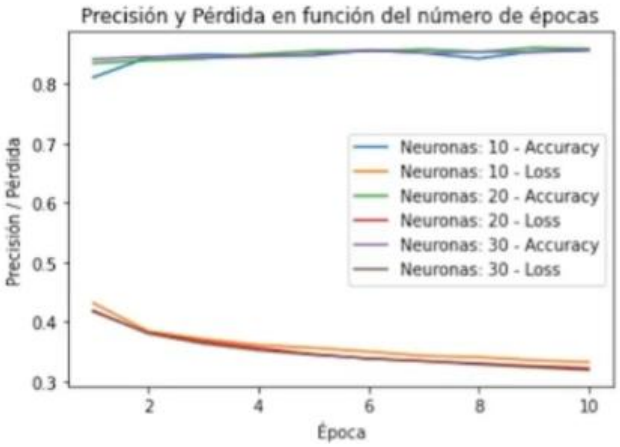



*Figura 17. Precisión y pérdida CNN 3 capas*

Si comparamos ambas gráficas observamos que para la CNN de 3 capas obtenemos una precisión más estable mientras aumenta el número de iteraciones, al contrario que en la CNN de 4 capas. Pese a que alcanzan una precisión similar cercana al 86%, viendo que las pérdidas son casi iguales y que ambas llegan a la precisión más alta en 20 neuronas y 9 iteraciones, decidimos quedarnos con estos parámetros para las siguientes métricas que saquemos para este modelo.

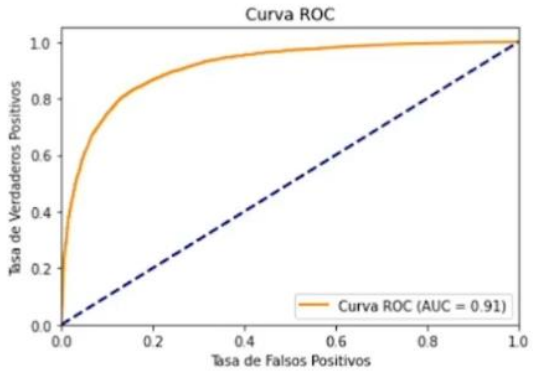



*Figura 18. Curva ROC CNN 3 Capas*

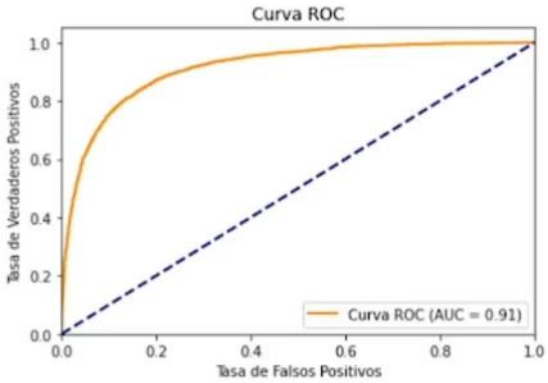



*Figura 19. Curva ROC CNN 4 Capas*

Si observamos las curvas ROC, obtenemos un área bajo la curva igual para ambos casos. Por lo tanto, agregar una capa adicional no está aumentando la cantidad de casos en los que el modelo realiza predicciones correctas. Considerando que añadir más capas al modelo conlleva un mayor costo computacional, hemos decidido quedarnos con el modelo CNN de 3 capas para el estudio de estas imágenes.

## 7.2. MODELO BASASO EN ResNet50

Tratamos ahora de valorar los resultados obtenidos para el modelo diseñado sobre ResNet50[12]. Veamos primero unas gráficas referentes a la precisión y pérdida del modelo.

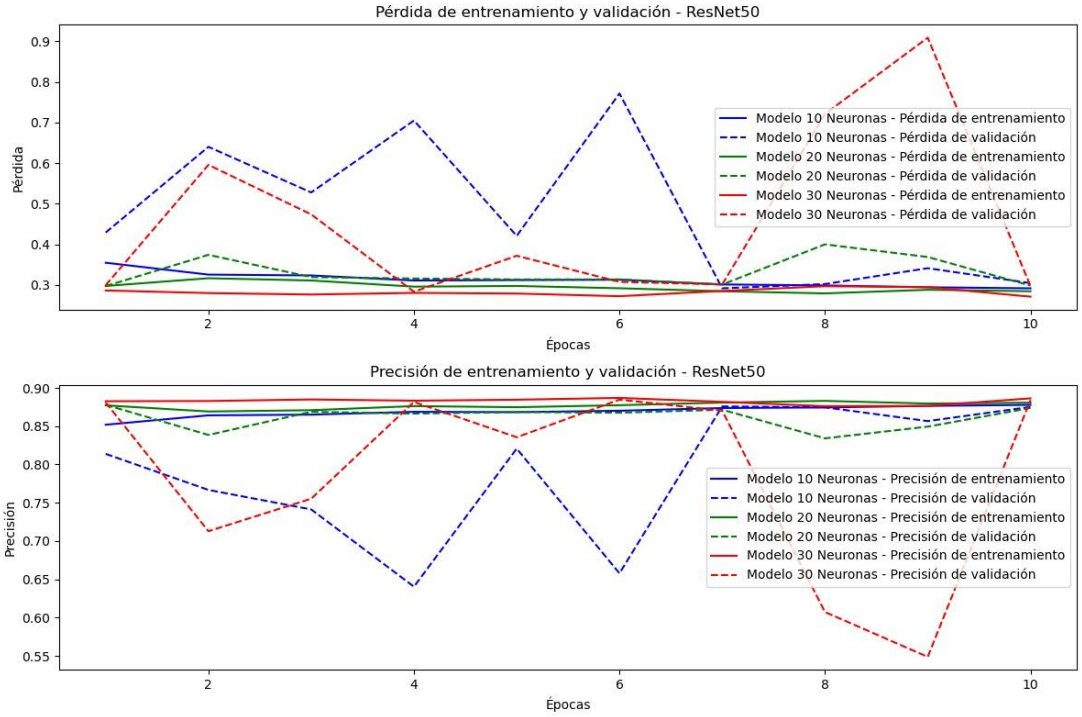



*Figura 20. Comparativa Precisión y Pérdida - ResNet50*

Se observa una precisión de entrenamiento elevada del modelo para los distintos números de neuronas pasados por parámetro. En los tres casos observamos que el modelo alcanza una precisión en torno al 88% en la última época de entrenamiento. En la fase de validación, existe una menor linealidad a medida que aumentan las iteraciones, sobre todo para los casos de 10 y 30 neuronas. Sin embargo, se encuentra la estabilidad en le décima época para los tres casos, también sobre el 88%.

En el estudio de la pérdida de entrenamiento y validación, se dan resultados acordes a las precisiones obtenidas. En la fase de entrenamiento encontramos una tendencia descendente en los tres casos, estabilizándose sobre el 30% en la última época. Esta tendencia descendente se pierde en el caso de validación, aunque hay una convergencia hacia el 30%, aproximadamente, en todas las opciones.

Pasamos ahora al estudio de la curva ROC para los casos de 10, 20 y 30 neuronas.

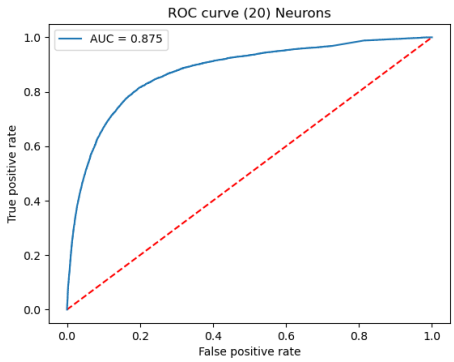



*Figura 21. Curva ROC 20 Neuronas - ResNet50*

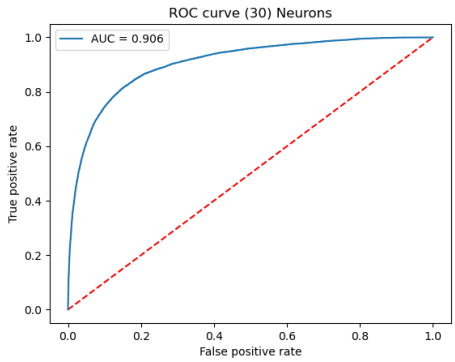



*Figura 22. Curva ROC 30 Neuronas - ResNet50*

Para 20 y 30 neuronas se obtiene un área bajo la curva de 0,875 y 0,906, respectivamente. Además, se observa una mayor estabilidad en la curva ROC para el modelo basado en utilizar 30 neuronas.

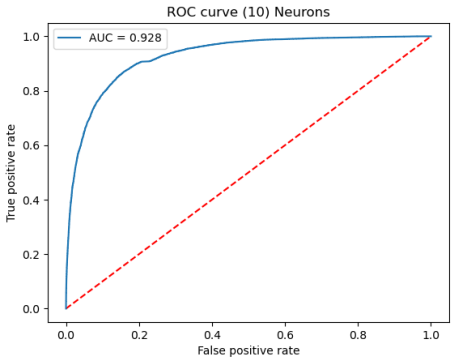



*Figura 23. Curva ROC 10 Neuronas - ResNet50*

Sin embargo, el mejor resultado se da utilizando 10 neuronas, pues obtenemos un área bajo la curva de 0,928. Por tanto, este modelo tiene un 92,8% de probabilidad de distinguir entre un tumor maligno y benigno, utilizando 10 neuronas como valor para el parámetro num_neurons.

## 7.3. MODELO BASADO EN DenseNet201

Comenzamos analizando las gráficas de precisión y pérdida del modelo.

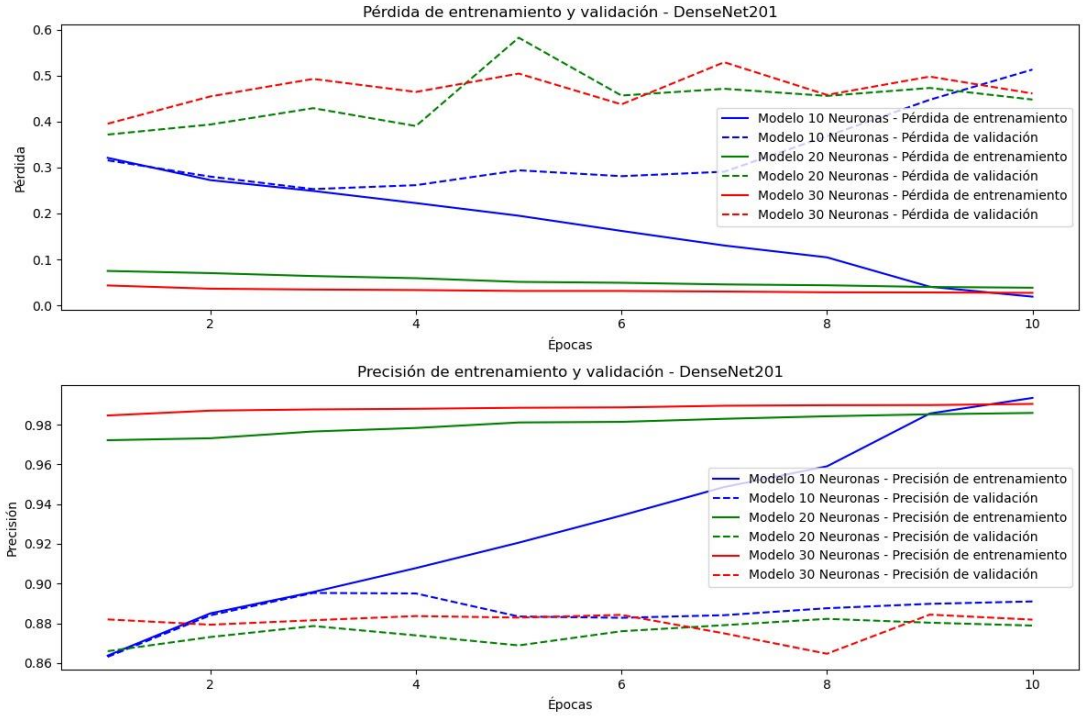



*Figura 24. Comparativa Precisión y Pérdida - DenseNet201*

En el modelo entrenado sobre DenseNet201 observamos unas precisiones muy elevadas para los datos de entrenamiento. En este ámbito, se tienen precisiones en torno al 98% en todas las épocas de entrenamiento para 20 y 30 neuronas. Para 10 neuronas, hay un mayor crecimiento a medida que se dan las iteraciones, llegando incluso a superar a los algoritmos con 20 y 30.

También se observan muy buenas precisiones en la fase de validación. Tal y como ocurre con ResNet50, hay saltos entre diferentes épocas, aunque estos son menores.

La pérdida del modelo, tanto en fase de entrenamiento como de validación, es acorde a las precisiones dadas. Encontramos convergencia para training hacia el 5%, aproximadamente, en los tres casos; y sobre un 40% para testing.

Procedemos a realizar un análisis sobre las curvas ROC.

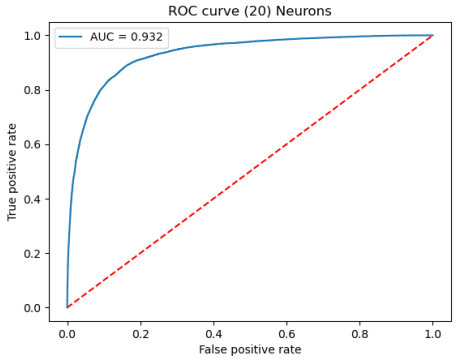



*Figura 25. Curva ROC 20 Neuronas - DenseNet201*

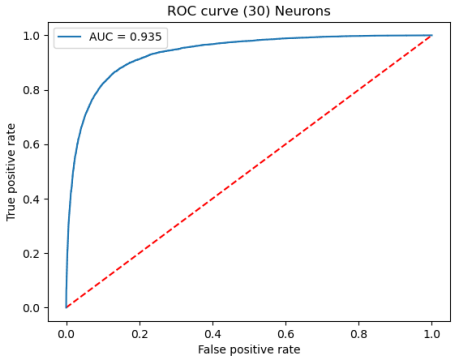



*Figura 26. Curva ROC 30 Neuronas - DenseNet201*

Observamos buena estabilidad en las tres gráficas y un valor del parámetro AUC superior al 90% en los tres casos.

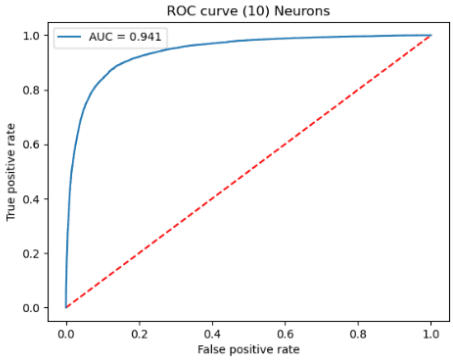



*Figura 27. Curva ROC 10 Neuronas - DenseNet201*

Sin embargo, el mejor valor obtenido para el área bajo la curva se da con 10 neuronas, como en el caso de ResNet50. Con lo cual, las mejores prestaciones de este modelo se dan con el valor 10 para el parámetro num_neurons, obteniendo una precisión de clasificación del 94,1%.

## 7.4. MODELO BASADO EN VGG19

El modelo entrenado sobre VGG19 ofrece resultados inferiores al resto de modelos de Transfer Learning.

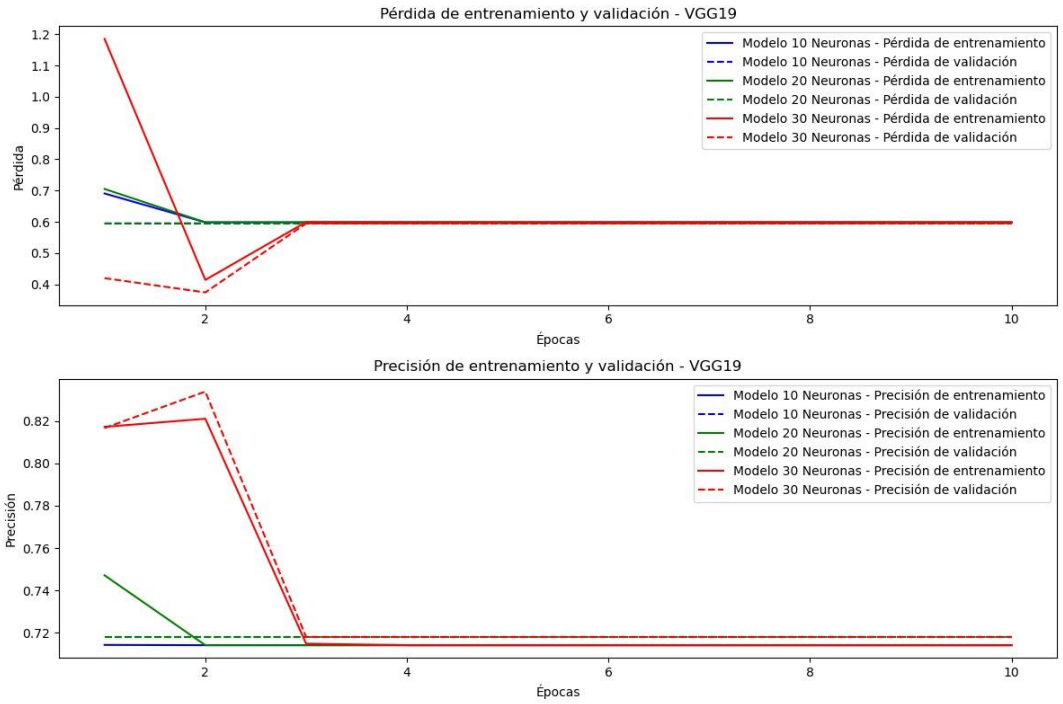



*Figura 258. Comparativa Precisión y Pérdida - VGG19*

Se observan valores prácticamente idénticos en las fases de entrenamiento y validación de cada caso, tanto para el concepto de pérdida como para el de precisión.

Este modelo tiene la particularidad de que las precisiones no mejoran en ningún momento a medida que aumentan las épocas, sino que empeoran o se mantienen. En el caso de 10 y 20 neuronas hay cierta homogeneidad, pero existe una disminución en torno al 10% para 30 neuronas. En los tres casos, se da una convergencia sobre un 72% de precisión.

Los valores de pérdida del modelo también empeoran a medida que se dan las iteraciones, estabilizándose sobre un 60% en todos los casos.

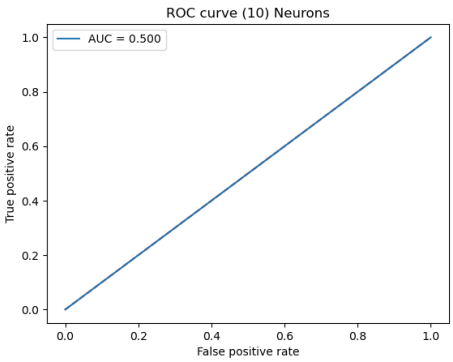



*Figura 29. Curva ROC 10 Neuronas - VGG19*

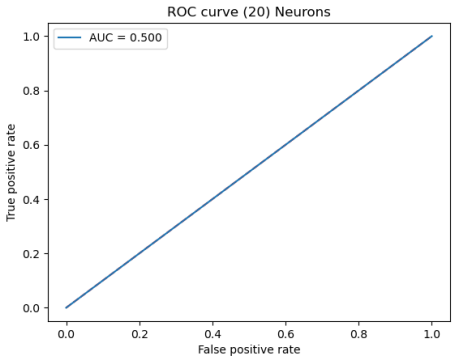



*Figura 3026. Curva ROC 20 Neuronas - VGG19*

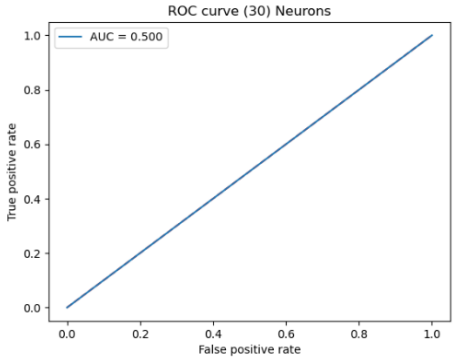



*Figura 3127. Curva ROC 30 Neuronas - VGG19*

Los resultados obtenidos referentes a las curvas ROC son insatisfactorios, pues en todos los casos obtenemos un AUC del 50%, el cual es el peor resultado posible. Esto significa que el modelo clasifica las imágenes de manera aleatoria.

# 8. CONCLUSIONES

Después de realizar un estudio exhaustivo de todos los modelos propuestos utilizando métodos estadísticos, hemos llegado a las siguientes conclusiones:

- Las redes neuronales convolucionales son herramientas altamente efectivas para la clasificación de imágenes de IDC. Tanto una red convolucional clásica como las redes más sofisticadas basadas en convoluciones demostraron buen desempeño, excepto en el caso de las redes VGG.

- No es necesario tener un número muy elevado de neuronas para entrenar estos modelos. Todos los modelos alcanzaron altas precisiones con tan solo 20 neuronas, y no se observó una mejora significativa al agregar más neuronas. Además, es importante destacar que estos modelos son muy estables en términos de iteraciones. A partir de alrededor de 8-9 iteraciones, la precisión y la pérdida se estabilizan.

- Las curvas ROC de los modelos que no se basan en VGG muestran un área bajo la curva (AUC) muy alta, generalmente alrededor de 0.9. Esto indica que estos modelos son confiables para la clasificación de imágenes de este tipo, ya que un valor de 0.9 implica que el 90% de las etiquetas predichas por el modelo son correctas

- Las redes del tipo VGG no son el modelo óptimo para este problema de clasificación, ya que generan predicciones aleatorias según se puede observar en las curvas ROC.

En resumen, los modelos basados en redes neuronales convolucionales, a excepción de las redes VGG, han demostrado ser efectivos y confiables para la clasificación de imágenes de IDC.

# 9. LÍNEAS FUTURAS DE INVESTIGACIÓN

**Extrapolar estas redes para otros tipos de imágenes de diferentes cánceres (melanoma, pulmonar, etc.):**

La extrapolación de estas redes se refiere a aplicar el mismo enfoque o metodología utilizada en el análisis de imágenes de un tipo específico de cáncer, como el de mama, a otros tipos de cánceres, como el melanoma o el cáncer de pulmón. Esto implica adaptar y entrenar las redes neuronales utilizadas para reconocer y clasificar imágenes de un tipo específico de cáncer, para que puedan analizar imágenes de otros tipos de cáncer. Al hacerlo, se podría aprovechar el conocimiento y las técnicas aprendidas de un tipo de cáncer para ayudar en el diagnóstico y tratamiento de otros tipos de cáncer.

**Identificar el estadio del tumor cancerígeno y ver la viabilidad de la cirugía:**

El estadio del tumor cancerígeno se refiere a la clasificación que indica la extensión y la gravedad del cáncer en el momento del diagnóstico. Al utilizar redes neuronales y técnicas de

procesamiento de imágenes, se puede intentar identificar el estadio del tumor cancerígeno basándose en características visuales y patrones identificados en las imágenes médicas. Además, se puede evaluar la viabilidad de la cirugía examinando la información proporcionada por las imágenes y otros datos clínicos. Esto puede ayudar a los médicos a tomar decisiones informadas sobre el plan de tratamiento más adecuado para el paciente.

**Incorporación de datos del paciente al estudio para dar más fiabilidad a los resultados (hábitos, análisis genéticos, etc.):**

Para mejorar la confiabilidad de los resultados, se pueden incorporar datos adicionales del paciente al estudio. Esto puede incluir información sobre los hábitos del paciente, como el historial de tabaquismo o la exposición a carcinógenos, que pueden tener un impacto en el desarrollo del cáncer y su progresión. Además, los análisis genéticos pueden proporcionar información sobre mutaciones genéticas específicas que pueden estar relacionadas con el cáncer y que podrían influir en el enfoque de tratamiento recomendado. Al combinar los datos de imagen con datos clínicos y genéticos, se puede obtener una imagen más completa y precisa de la enfermedad, lo que puede ayudar a personalizar el tratamiento y mejorar la precisión del diagnóstico.

**Estudio del dataset con nuevas redes neuronales:**

El estudio del dataset con nuevas redes neuronales implica aplicar métodos de aprendizaje automático, como el uso de nuevas arquitecturas de redes neuronales, al conjunto de datos recopilado para el análisis de imágenes de cáncer. Esto puede implicar la exploración de enfoques más avanzados, como redes neuronales convolucionales mejoradas o redes neuronales recurrentes, que pueden ser más eficientes en el procesamiento y la extracción de características de las imágenes. Al utilizar nuevas redes neuronales, se puede buscar mejorar la precisión de la clasificación, la detección de características relevantes y la comprensión general de los datos de imagen, lo que puede conducir a avances en el diagnóstico y tratamiento del cáncer.

# 10. ANEXOS

## 10.1. DIARIO DE PROYECTO

- Semana 1: Del 05/06/2023 al 11/06/2023.
    - Determinar un buen conjunto de datos sobre el que trabajar.
    - Familiarizarnos con conceptos relevantes: red neuronal, red convolucional, transfer learning, limpieza de ruido en fotografías, convertir fotografías en arrays.
    - Carga de los datos a Python. Limpieza correcta del ruido en las imágenes, aunque era escaso. Conversión las fotografías a numpy-arrays para su manipulación.

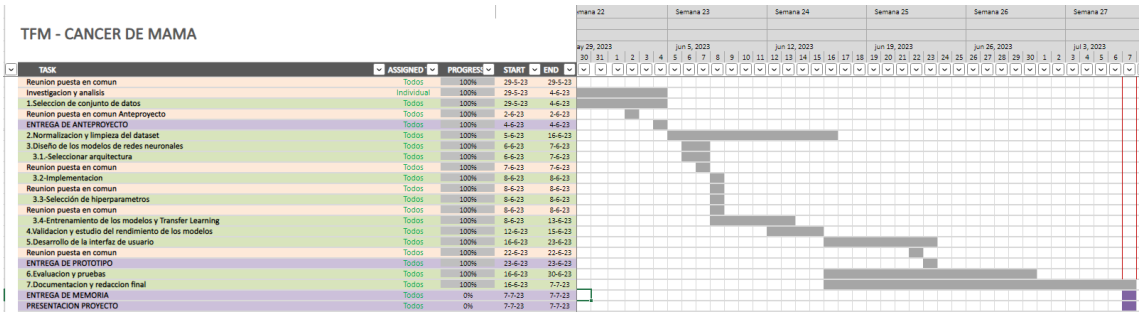
- Semana 2: Del 12/06/2023 al 18/06/2023.
    - Almacenamiento de las imágenes en formato “pickle”.
    - Iniciación del desarrollo de un modelo de red convolucional. Primeras pruebas con una capa, con resultados poco satisfactorios.
    - Investigación sobre modelos preentrenados de Transfer Learning. Decantación por el uso de los modelos VGG16, VGG19, DenseNet201 y ResNet50.
    - Pruebas en el modelo de red convolucional. Se ha probado a usar diferentes números de capas y de neuronas por capa. Cabe destacar que los mejores resultados se obtienen con tres y cuatro capas, pues a menor número hay mala precisión, y a mayor número no hay mejoras. También hay que tener en cuenta que el modelo necesita bastante tiempo de entrenamiento cuando hacemos uso de muchas imágenes (varias horas, incluso).
    - Primeras pruebas con modelos de Transfer Learning: VGG 16, VGG 19, DenseNet201 y ResNet50.
- Semana 3: Del 19/06/2023 al 25/03/2023.
    - Progresos en los modelos preentrenados. Se obtiene buena precisión en los modelos DenseNet201 y ResNet50, mientras que los modelos VGG16 y VGG19 dan como resultado niveles de precisión poco satisfactorios.
    - En el modelo de red CNN, nos decantamos por el uso de tres capas, pues ofrece resultados similares a usar cuatro capas, siendo computacionalmente menos costoso.
    - Obtención de predicciones en todos los modelos, con buenos resultados excepto en VGG16 y VGG19. Creemos que estos dos últimos modelos dan malos

resultados debido a que son modelos de aprendizaje muy profundo y necesitan una mayor cantidad de imágenes, y una mejor repartición de estas.

  - Obtención de resultados y visualizaciones en los modelos CNN, DenseNet201 y ResNet50: gráficas comparativas, precisión en función de las iteraciones efectuadas, curva ROC, matriz de confusión. Estudio de los casos en función del número de neuronas usadas.

- Semana 4: Del 26/06/2023 al 02/07/2023.
  - Obtención de resultados y conclusiones.
  - Creación API web.
- Semana 5: Del 03/07/2023 al 07/07/2023.
  - Escritura de la memoria.
  - Preparación de exposición.

## 10.2. PLANIFICACIÓN




*Incluimos enlace al Excel en la parte bibliográfica*[13]

## 10.3. MANUAL DE INSTALACIÓN PARA EL USO LOCAL DE LA APLICACIÓN WEB

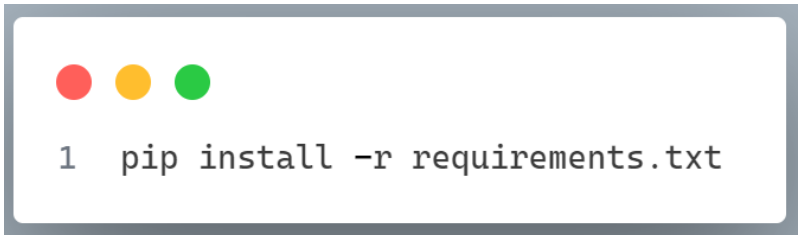
### Configuración del Entorno

#### Windows

1. Descarga e instala Python desde el sitio web oficial: https://www.python.org/downloads/
2. Abre el Símbolo del sistema (Command Prompt).
3. Navega hasta la carpeta raíz del proyecto utilizando el comando **cd**.
4. Crea un entorno virtual ejecutando el siguiente comando:


5. Activa el entorno virtual con el siguiente comando:

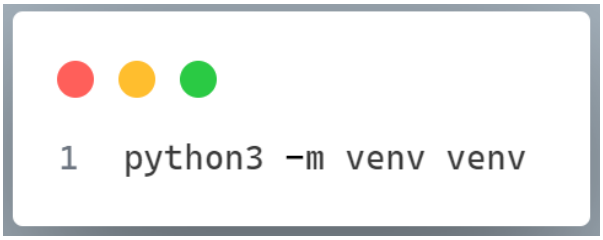

6. Instala las dependencias enumeradas en el archivo requirements.txt utilizando el siguiente comando:




7. Ejecuta el programa Flask con el siguiente comando:

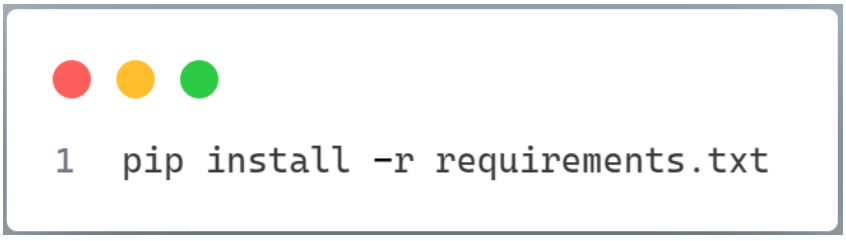

**macOs**

1. Abre la Terminal.
2. Navega hasta la carpeta raíz del proyecto utilizando el comando cd.
3. Crea un entorno virtual ejecutando el siguiente comando:




4. Activa el entorno virtual con el siguiente comando:

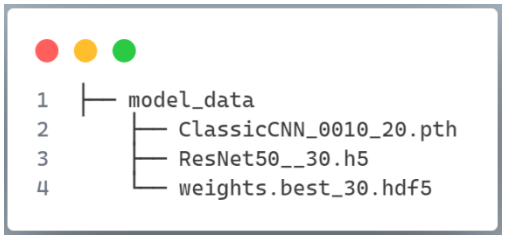

5. Instala las dependencias enumeradas en el archivo requirements.txt utilizando el siguiente comando:




6. Ejecuta el programa Flask con el siguiente comando:

## Uso

Puedes encontrar los archivos del modelo en el siguiente enlace (https://1drv.ms/u/s!ApTJouCVOUYZlJYYG2ODo6jwgZYLiA?e=2fttC5)*. Una vez que hayas descargado el archivo ZIP, asegúrate de descomprimirlo en la carpeta \model_data de manera que esté estructurado de la siguiente manera:




*Si no se te permite hacerlo, por favor contáctanos.

Una vez que el programa haya iniciado, puedes ingresar tu imagen a clasificar y al hacer clic en el botón de predicción, aparecerá la predicción para los modelos con la imagen seleccionada.

Este sitio web está relacionado con el repositorio (https://github.com/diego-tech/TFM_Breast_Cancer_Classification) que alberga el código de los modelos.

# BIBLIOGRAFÍA


- [1] Diccionario de cáncer del NCI. (n.d.). Instituto Nacional Del Cáncer. https://www.cancer.gov/espanol/publicaciones/diccionarios/diccionario-cancer/def/carcinoma-ductal-invasivo\
- [2] Siegel, R. L., Miller, K. A., Fuchs, H. E., & Jemal, A. (2021). Cancer Statistics, 2021. CA: A Cancer Journal for Clinicians, 71(1), 7–33. https://doi.org/10.3322/caac.21654
- [3] Invasive Ductal Carcinoma (IDC): Grade, Symptoms & Diagnosis. (n.d.). https://www.breastcancer.org/types/invasive-ductal-carcinoma
- [4] Sammut, S. J., Crispin-Ortuzar, M., Chin, S. F., Provenzano, E., Bardwell, H. A., Ma, W., Cope, W., Dariush, A., Dawson, S. J., Abraham, J. E., Dunn, J., Hiller, L., Thomas, J., Cameron, D. A., Bartlett, J. M. S., Hayward, L., Pharoah, P. D., Markowetz, F., Rueda, O. M., Earl, H. M., & Caldas, C. (2023). Multi-omic machine learning predictor of breast cancer therapy response. Nature Communications, 14(1), 1234.
- https://www.nature.com/articles/s41586-021-04278-5
- [5] Documentación oficial de Python: https://www.python.org/
- [6] Documentación oficial de TensorFlow: https://www.tensorflow.org/api_docs
- [7] Documentación oficial de PyTorch: https://pytorch.org/docs/stable/index.html
- [8] Documentación oficial de Keras: https://keras.io/api/
- [9] Documentación oficial de OpenCV: https://docs.opencv.org/4.x/
- [10] Documentación oficial de Flask https://flask.palletsprojects.com/en/2.3.x/
- [11] Documentación oficial de Git https://git-scm.com/doc
- [12] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 770-778). Retrieved from https://arxiv.org/abs/1512.03385
- [13] Cronograma de planificación CANCER DE MAMA_TFM_2023_rev.xlsx