



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



Escola Tècnica  
Superior d'Enginyeria  
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica  
Universitat Politècnica de València

# **Creación de modelos para la detección de enfermedades y su ejecución en plataformas serverless**

Trabajo Fin de Máster  
**Máster en Big Data Analytics**

**Autor:** Sergio Langarita Benítez

**Tutor:** Germán Moltó

**Curso:** 2021-2022

Máster en Big Data Analytics



# Resumen

Se han obtenido datos médicos anónimos de la plataforma *Kaggle* y se han usado para obtener modelos predictivos de diferentes enfermedades, como el cáncer de mama, detectando si un tumor es benigno o maligno. También para enfermedades respiratorias como el COVID-19, neumonía o tuberculosis a través de una radiografía del tórax, o el Alzheimer y su desarrollo en el paciente mediante una resonancia magnética del cerebro. Estos modelos se han insertado en contenedores para facilitar su uso y despliegue. También se han adaptado para su ejecución sobre una plataforma serverless como es OSCAR. Además, se han expuesto los modelos por medio de interfaces gráficas, simulando un entorno de producción.

**Palabras clave:** *Serverless, Cloud Computing, FaaS, support vector machine, regresión logística, regresión lineal, K-means, red neuronal convolucional (CNN)*



## *Abstract*

Anonymous health data has been obtained from the Kaggle repositories. These data have been used in order to obtain several models. They predict diseases, such as breast cancer, classifying tumors as benign or malignant. Also, respiratory diseases like COVID-19, pneumonia, or tuberculosis from a chest x-ray image, or Alzheimer, developed from Magnetic Resonance Images. Those models have been packaged into containers to facilitate their use and deployment. Also, they have been adapted into a serverless platform like OSCAR. Moreover, the models have been exposed via a graphical interface, simulating a production environment.

**Keywords** : Serverless, Cloud Computing, FaaS, support vector machine, logistic regression, linear regression, K-means, Convolutional neural network (CNN)



# Tabla de contenidos

1. Introducción	11
2. Estado del arte	12
2.1. Cloud Computing	12
2.2. Serverless	13
2.3. OSCAR y FDL Composer	13
2.4. Infrastructure Manager (IM)	14
2.5. Otras Herramientas utilizadas	16
3. Desarrollo	17
3.1. Modelos y métricas	17
3.2. Obtención y procesamiento de los datos	18
3.3. Detección de cáncer de mama	18
3.3.1. Support Vector Machine	23
3.3.2. Support Vector Machine con Cross Validation	24
3.3.3. Support Vector Machine con Cross Validation y Shuffle Split	25
3.3.4. Regresión Logística	26
3.3.5. Regresión Logística con Cross Validation	27
3.3.6. Regresión Logística con Cross Validation y Shuffle Split	28
3.3.7. Kmeans	29
3.3.8. Árbol de decisión	30
3.3.9. Regresión Lineal sin normalizar	31
3.3.10. Regresión Lineal normalizado	32
3.3.11. Regresión Lineal con Cross Validation	23
3.3.12. Regresión Lineal con Cross Validation y Shuffle Split	34
3.3.13. Random Forest	35
3.3.14. Resultados	36
3.4. Predicción de enfermedades respiratorias mediante radiografías	39

3.5. Predicción de Alzheimer mediante imagen de resonancia magnética	42
3.6. Despliegue de OSCAR con IM	45
3.7. Creación y despliegue de los servicios de OSCAR	50
4. Caso de Uso	54
4.1. Despliegue de Gradio en Kubernetes	55
5. Conclusiones y Trabajo futuro	58
6. ODS (OBJETIVOS DE DESARROLLO SOSTENIBLE)	59
7. Referencias Bibliográficas	60
8. Anexos	66



## Índice de Figuras

Figura 1: Arquitectura de OSCAR	14
Figura 2: Arquitectura de IM	15
Figura 3: Matriz de confusión del modelo SVM	23
Figura 4: Matriz de confusión del modelo de regresión logística	26
Figura 5: Matriz de confusión del modelo de regresión lineal sin normalizar	31
Figura 6: Matriz de confusión del modelo de regresión lineal normalizado	32
Figura 7: Comparación de los accuracy entre los diferentes modelos	37
Figura 8: Matriz de confusión del modelo de regresión lineal seleccionado	38
Figura 9: Distribución de las radiografías en las diferentes clases del dataset	39
Figura 10: Distribución de las radiografías en las diferentes clases del dataset para el modelo final	40
Figura 11: Recall y Precision de las diferentes clases del dataset de enfermedades respiratorias	42
Figura 12: Recall y Precision de las diferentes clases del dataset para la predicción de Alzheimer.	44
Figura 13: Credenciales almacenadas en IM	48
Figura 14: Infraestructuras desplegadas en IM	49
Figura 15: Endpoints en IM	50
Figura 16: Flujos de trabajo creados con FDL Composer	53
Figura 17: Diagrama de la arquitectura	54
Figura 18: Cluster de OSCAR con los 4 servicios desplegados	56
Figura 19: Interfaz gráfica construida con Gradio	57



# Índice de Tablas

Tabla 1: Descripción del dataset de cáncer de mama	19
Tabla 2: Correlación entre el diagnóstico y las demás variables	19
Tabla 3: Frecuencia de las variables según su diagnóstico	20
Tabla 4: Correlación entre las diferentes variables	21,22
Tabla 5: Resultado del modelo SVM	23
Tabla 6: Resultado de SVM con Cross Validation	24
Tabla 7: Resultado SVM con Cross Validation y Shuffle Split	25
Tabla 8: Resultado del modelo de regresión logística	26
Tabla 9: Resultado del modelo de regresión logística con Cross Validation	27
Tabla 10: Resultado del modelo de regresión logística con Cross Validation con Shuffle Split	28
Tabla 11: Resultado del modelo Kmeans en varias ejecuciones	29
Tabla 12: Resultado del modelo árbol de decisión	30
Tabla 13: Resultado de un modelo de regresión lineal sin normalizar	31
Tabla 14: Resultado del modelo de regresión lineal	32
Tabla 15: Resultado del modelo de regresión lineal con Cross Validation	33
Tabla 16: Comparación de los modelos creados con Cross Validation.	33
Tabla 17: Resultado del modelo de regresión lineal con Cross Validation y Shuffle Split.	34
Tabla 18: Comparación de los modelos creados con Cross Validation y Shuffle Split.	34
Tabla 19: Resultado del modelo de Random Forest.	35
Tabla 20: Comparación entre las diferentes medias	36
Tabla 21: Modelo de regresión lineal seleccionado	38
Tabla 22: Distribución de las imágenes en las diferentes clases del dataset	39
Tabla 23: Matriz de confusión sobre el resultado de la red neuronal	41
Tabla 24: Resultados de la red neuronal	41



Tabla 25: Matriz de confusión sobre el resultado de la red neuronal	43
Tabla 26: Resultados de la red neuronal	43
Tabla 27: Nombre de los servicios en diferentes proveedores Cloud	45
Tabla 28: Limitaciones de los proveedores Cloud	46
Tabla 29: Coste de los servicios Cloud	47
Tabla 30: Simulación de los costes	47

# 1. Introducción

---

En un entorno clínico, la predicción temprana de una enfermedad es un factor clave. Aunque no siempre se cumple, debido a que el tiempo entre la realización de las pruebas y la obtención del diagnóstico suele ser amplio. Aun así, el sistema de salud puede verse beneficiado con un diagnóstico temprano implementando el uso de modelos predictivos, reduciendo los tiempos de espera, priorizando los casos importantes, con una mejor planificación. Además con el uso de una medicación menos agresiva, aliviando el dolor del paciente y reduciendo los costes.

Como una contribución a esta problemática, desde el ámbito de la computación y el procesamiento de datos, se ha propuesto el objetivo de crear tres modelos para la predicción de enfermedades y estudiar las diferentes técnicas en la creación de modelos médicos con datos abiertos. Crear un primer modelo que realice la predicción de cáncer de mama en relación a un tumor. Crear un segundo modelo que prediga la existencia de una enfermedad respiratoria como Covid-19, neumonía o tuberculosis con respecto a una radiografía. Y, finalmente, crear un tercer modelo que efectúe una predicción de Alzheimer y su estado respecto a una imagen de resonancia magnética.

Además de adaptar los modelos a un sistema de *Cloud* computing para tener alta disponibilidad de ellos. Entre todos los tipos de servicios ofertados en un sistema *Cloud* se ha decidido implementar los modelos en una infraestructura *Serverless* para que se efectúe una explotación más efectiva de los recursos con una gestión dinámica de éstos. También se pretende adaptar el flujo de trabajo y facilitar su uso al usuario final mediante una interfaz gráfica.



## 2. Estado del arte

Se han utilizado diferentes tecnologías para el desarrollo del proyecto. En los siguientes apartados se da una breve descripción de las tecnologías usadas.

### 2.1. Cloud Computing

Según la NIST (National Institute of Standards and Technology) [25] *cloud computing* es el acceso mediante la red a un conjunto de recursos configurables compartidos, que se utilizan y liberan bajo demanda. Las principales características son:

- El usuario puede utilizar los recursos del sistema sin necesidad de intervención humana.
- El sistema debe ser accesible a través de la red.
- El sistema puede ser usado por varios usuarios donde los recursos son dinámicamente asignados respecto a la demanda del usuario. La localización exacta de los recursos no puede ser controlada por el usuario, pero sí se puede seleccionar un nivel más alto como un centro de datos, zona o país.
- Dar la capacidad al usuario de aprovisionar y liberar recursos elásticamente, es decir, de forma automática y a demanda.
- Los recursos del sistema pueden ser monitorizados.

En *cloud computing* hay diferentes modelos de servicios:

- Infraestructura como servicio (*IaaS*): donde se contratan recursos como máquinas virtuales o almacenamiento.
- Plataforma como servicio (*PaaS*): donde el usuario ejecuta una aplicación sin administrar la infraestructura, solo configurando el entorno de ejecución.
- Software como servicio (*SaaS*): donde el producto no necesita ninguna configuración y el software está listo para utilizar como *One Drive* o *Google Docs*.

En *cloud computing* hay diferentes modelos de despliegue:

- Cloud privadas: infraestructura exclusiva de una entidad.
- Clouds community: infraestructura para una comunidad, o varias organizaciones que comparten objetivos.
- Cloud públicas: Infraestructuras abiertas al público en general, por ejemplo, *Amazon Web Services*, *Microsoft Azure* o *Google Cloud*.

- Clouds híbridas: infraestructura compuesta por dos o más tipos de infraestructuras clouds.

## 2.2. Serverless

Es la computación en la nube con una asignación dinámica de recursos mediante demanda sin la necesidad de escalar, administrar o mantener la infraestructura por parte del desarrollador [30]. Con este paradigma de computación se ha creado el modelo servicio *Function as a service (FaaS)* donde se asignan los recursos necesarios a un proceso que se crea, ejecuta el código deseado y, al terminar, libera los recursos.

Cada proveedor de servicios en la nube ofrece sus propios servicios de *FaaS*. En Amazon Web Services se presenta con el nombre *AWS Lambda* [26]. En la nube de *Microsoft*, *Azure* tiene el nombre de *Azure Functions* [27], y en *Google Cloud* se presenta como *Google Cloud functions* [28].

También hay bases de datos *serverless*, donde la escalabilidad es gestionada automáticamente por el proveedor. Este es el caso de Amazon Aurora (Base de datos relacional), Amazon DynamoDB (Base de datos no relacional) o Google Firestore (Base de datos no relacional).

## 2.3. OSCAR y FDL Composer

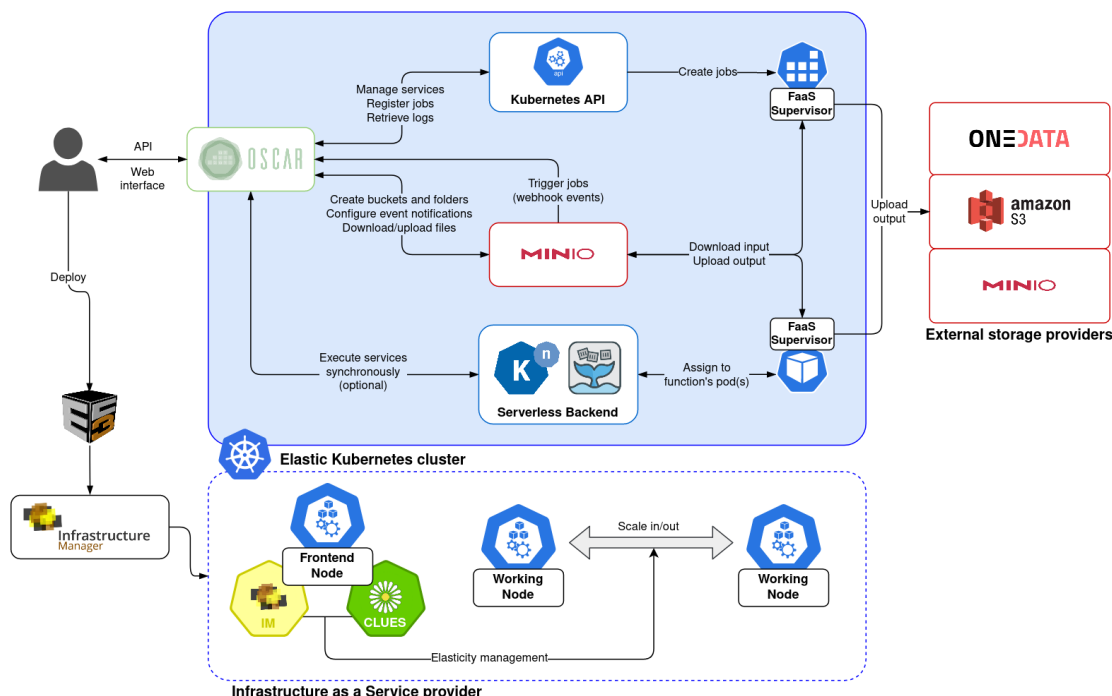
OSCAR<sup>1</sup> [2] es una plataforma de código abierto para la ejecución de aplicaciones *serverless* para el procesamiento de datos, basado en Docker, mediante un despliegue de Kubernetes elásticos. Es posible desplegar OSCAR tanto en un proveedor de nube pública como privada. OSCAR está implementado sobre la API de Kubernetes [7], una herramienta para la administración de contenedores -programas encapsulados con sus dependencias-, y MinIO [29], un servicio de almacenamiento de objetos. Los objetos se almacenan en *buckets* que, a su vez, se organizan de forma interna con un sistema de directorios.

El funcionamiento de OSCAR es el siguiente: Al subir un fichero a un bucket de MinIO [29], es decir al subir un fichero a una carpeta de un servidor de almacenamiento, se desencadena la creación de un contenedor predefinido, teniendo como argumento de entrada el propio fichero y ejecutando el código deseado dentro del contenedor, depositando en el bucket de salida el resultado. Este bucket puede accionar otro servicio de OSCAR, creando así flujos de trabajo (*workflows*), que pueden ser descritos mediante el lenguaje FDL [4] (Functions Definition Language).

---

<sup>1</sup> OSCAR - <https://oscar.grycap.net>





**Figura 1: Arquitectura de OSCAR**

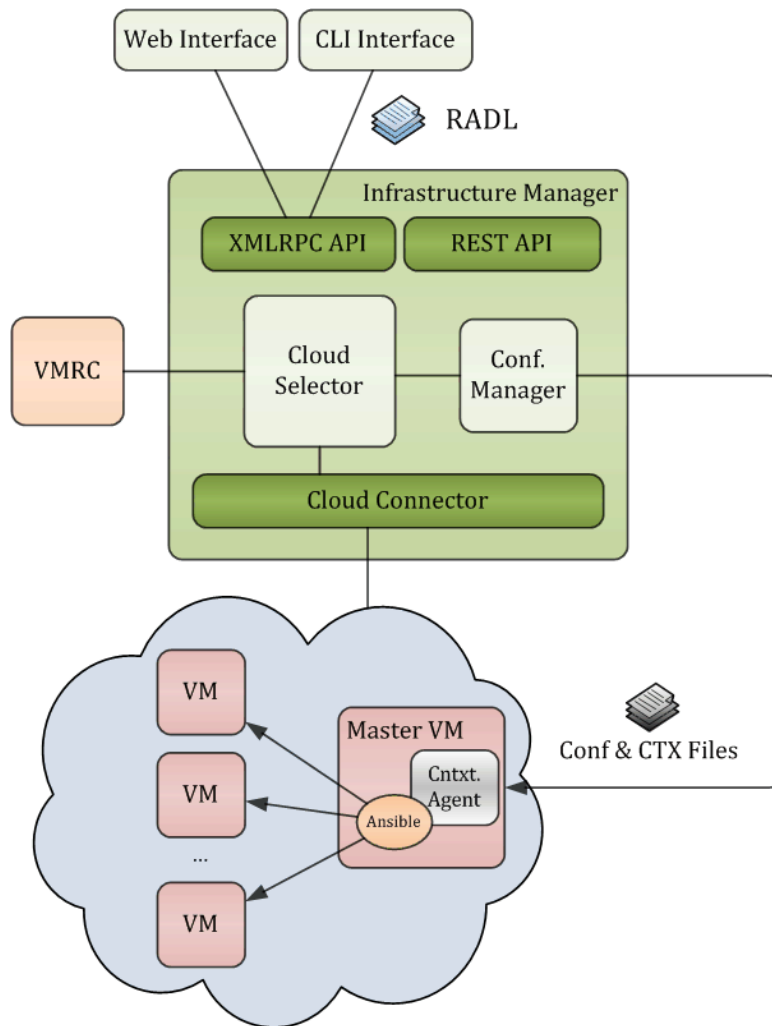
FDL (Functions Definition Language) Composer [4] es una herramienta para la ayuda de creación de ficheros de despliegue automático de flujos de trabajo para OSCAR. Tanto OSCAR, como FDL Composer son herramientas desarrolladas por el grupo de investigación GRyCAP [5] de la Universitat Politècnica de València.

## 2.4. Infrastructure Manager (IM)

Infrastructure Manager (IM) [3] es una herramienta de código abierto para la gestión y despliegue de infraestructuras virtualizadas en nubes públicas o privadas. También mencionar IM Dashboard [23], un cliente web que facilita la interacción con IM para el despliegue de infraestructuras.

El primer componente de IM es una API para interactuar con la infraestructura. Gracias a esta API, IM Dashboard accede al IM. El segundo componente a mencionar es *Cloud Selector*, componente que interactúa con VMRC y *Cloud Connector*. VMRC está compuesto de una lista de las imágenes de máquinas virtuales disponibles en la plataforma *Cloud*. *Cloud Connector*, es una capa diseñada para realizar una conexión homogénea en diferentes proveedores

*cloud*. El último componente es *Configuration Manager* que administra y monitoriza las infraestructuras ya creadas.



**Figura 2: Arquitectura de IM**

OSCAR se despliega sobre un clúster virtual elástico basado en Kubernetes. La creación automática de este clúster es realizada por IM, y OSCAR se puede desplegar en varios clouds gracias al componente *Cloud Connector* de IM. IM e IM Dashboard son herramientas desarrolladas por el grupo de investigación GRyCAP [5].

## 2.5. Otras Herramientas utilizadas

Además, se han utilizado otras herramientas para el desarrollo de este proyecto:

- Docker [6] y Kubernetes [7]: Docker es una herramienta para encapsular programas, junto con sus dependencias mínimas en contenedores. Por su parte, Kubernetes es una herramienta diseñada para el uso de los contenedores de forma automática.
- Scikit-learn [8] y TensorFlow [9]: Son librerías para el aprendizaje automático. Se han utilizado submódulos como `tensorflow_datasets` o `keras`.
- Numpy [10] y Pandas [11]: Se han utilizado las librerías Numpy y Pandas, para el manejo de los datos.
- Gradio [12]: Librería de Python para crear interfaces web para modelos de Inteligencia Artificial.
- Kaggle [13,14,15]: Web relacionada con la ciencia de datos, donde hay competiciones, un foro para dudas y publicaciones, cursos para aprender y repositorios de datos, también conocidos como datasets.
- Google Colab [16]: Plataforma de Google para ejecutar cuadernos de Python con el uso de *GPUs* sin la necesidad de configuración.
- splitfolders [17]: Es una librería de Python utilizada para separar un dataset de imágenes concentradas en una carpeta en *train*, *test* y *validation*.
- OpenNebula [19]: Herramienta para la administración y gestión de las plataformas *cloud*.



## 3. Desarrollo

En los siguientes puntos se describe el desarrollo de este trabajo, una breve introducción de los modelos predictivos utilizados así como las métricas usadas para la evaluación de los modelos. Seguidamente se describen los procesos realizados con la manipulación de datos, la obtención de los diferentes modelos y la encapsulación dentro de contenedores.

### 3.1. Modelos y métricas

Los modelos han sido seleccionados debido a que son los más comunes, partiendo de un modelo inicial e intentando mejorar el resultado del modelo anterior, ya sea seleccionando otro modelo o combinándolo con técnicas de separación del dataset. Los modelos utilizados son los siguientes:

- *Support Vector Machine* (SVM): Aprendizaje supervisado donde la separación de las diferentes clases se realiza mediante un hiperplano.
- Regresión logística: Modelo de predicción de una variable categórica entre dos clases.
- Regresión lineal: Modelo de aprendizaje supervisado para la predicción de una variable, mediante la creación de una función lineal derivada de los datos.
- Random forest: Modelo de predicción de una variable mediante la creación de árboles de decisión de forma aleatoria.
- Red neuronal convolucional: Modelo de predicción mediante una red neuronal donde las neuronas buscan patrones mediante el procesamiento de datos en forma de matriz. Es un modelo muy efectivo en la clasificación de imágenes.
- *Cross Validation*: Separación de un dataset en K bloques. Donde en K iteraciones, un bloque será parte del test, en cada iteración será diferente y el resto del dataset formará parte del entrenamiento.
- *Cross Validation Shuffle Split*: Mismo procedimiento de entrenamiento que *Cross Validation* pero con la separación de los datos en bloques de forma aleatoria.
- Árbol de decisión: Modelo de predicción de una variable mediante la creación de un esquema lógico.



Las medidas para la evaluación de los diferentes modelos:

- *Accuracy*: Número total de aciertos dividido entre el número total de la predicción.
- *Recall*: Verdaderos positivos dividido entre Verdaderos positivos y falso positivo. Indica cuánto ha acertado respecto al total de muestras de una clase..
- *Precision*: Verdaderos positivos dividido entre verdaderos positivo y falso negativo. Se interpreta de la siguiente manera: Sobre todas las clasificaciones hechas a una clase, cuánto porcentaje de acierto existe.

## 3.2. Obtención y procesamiento de los datos

Los datos obtenidos se encontraban en repositorios de datos públicos en la web de *Kaggle*. En el primer *dataset* [13] se expone la predicción de cáncer de mama dado un tumor y las características de éste. El segundo *dataset* [14] posee imágenes de radiografías del tórax donde se da una enfermedad. El objetivo es predecir la enfermedad respiratoria que tiene el paciente. El tercer y último *dataset* [15] almacena imágenes de resonancias magnéticas del cerebro con Alzheimer, en diferentes fases de la enfermedad. Se busca predecir en qué estado de la enfermedad se encuentra el paciente.

## 3.3. Detección de cáncer de mama

El primer *dataset*, contiene información sobre tumores, el radio, la textura, el perímetro, el área, su uniformidad y el diagnóstico. Consta de seis columnas y 569 filas. En ningún campo del dataset se han encontrado datos faltantes. La descripción de las columnas es la siguiente:

- *diagnosis*: diagnóstico, 1 si es un tumor maligno, y 0 si es benigno.
- *mean\_radius*: media de la distancia desde el centro al perímetro, del tumor.
- *mean\_texture*: desviación estándar de los valores en la escala de grises.
- *mean\_perimeter*: media del perímetro del tumor.
- *mean\_area*: media de la variación local en longitudes de radio.

- *mean\_smoothness*: media de la variación local de las longitudes del radio.

En la primera fase se ha inspeccionado el dataset de forma general, obteniendo el porcentaje de cada clase, los posibles campos faltantes y se ha separado el *dataset* en *train* y *test* en una proporción de 80-20. Tanto para la exploración de los datos como para la creación de los distintos modelos se ha usado la librería de Python Scikit-learn.

Número total de filas del <i>dataset</i>	569
Número total de diagnóstico a 0	212 (37,25% del total)
Número total de diagnóstico a 1	357 (62,74% del total)
Null en el dataset:	0
Número total de filas del train	455 (80%)
Número total de filas del test	114(20)

**Tabla 1: Descripción del *dataset* de cáncer de mama**

Se ha procedido a normalizar las variables por columnas, porque cambiando los valores entre 0 y 1, se simplifican las variables de entrada en los modelos, excepto la variable *mean\_smoothness* ya que siempre está entre el rango 0 y 1. Se extraen los valores máximos de cada columna de la parte de *train* y con ellos se divide cada componente de la columna. Después se ha obtenido la correlación entre las diferentes variables y el diagnóstico.

	<i>mean_radius</i>	<i>mean_perimeter</i>	<i>mean_texture</i>	<i>mean_area</i>	<i>mean_smoothness</i>
diagnosis	-0.727037	-0.421403	-0.739597	-0.703594	-0.361859

**Tabla 2: Correlación entre el diagnóstico y las demás variables.**

Se han realizado diferentes visualizaciones sobre la frecuencia de las variables respecto a su diagnóstico. En la tabla 3, se puede observar varios gráficos, donde el eje x es la variable normalizada de la celda superior y el eje y es la frecuencia con la que aparece dicho valor en la variable. Se ha realizado una impresión de

los datos respecto al diagnóstico final, donde el color rojo es un diagnóstico maligno y el verde un diagnóstico benigno.

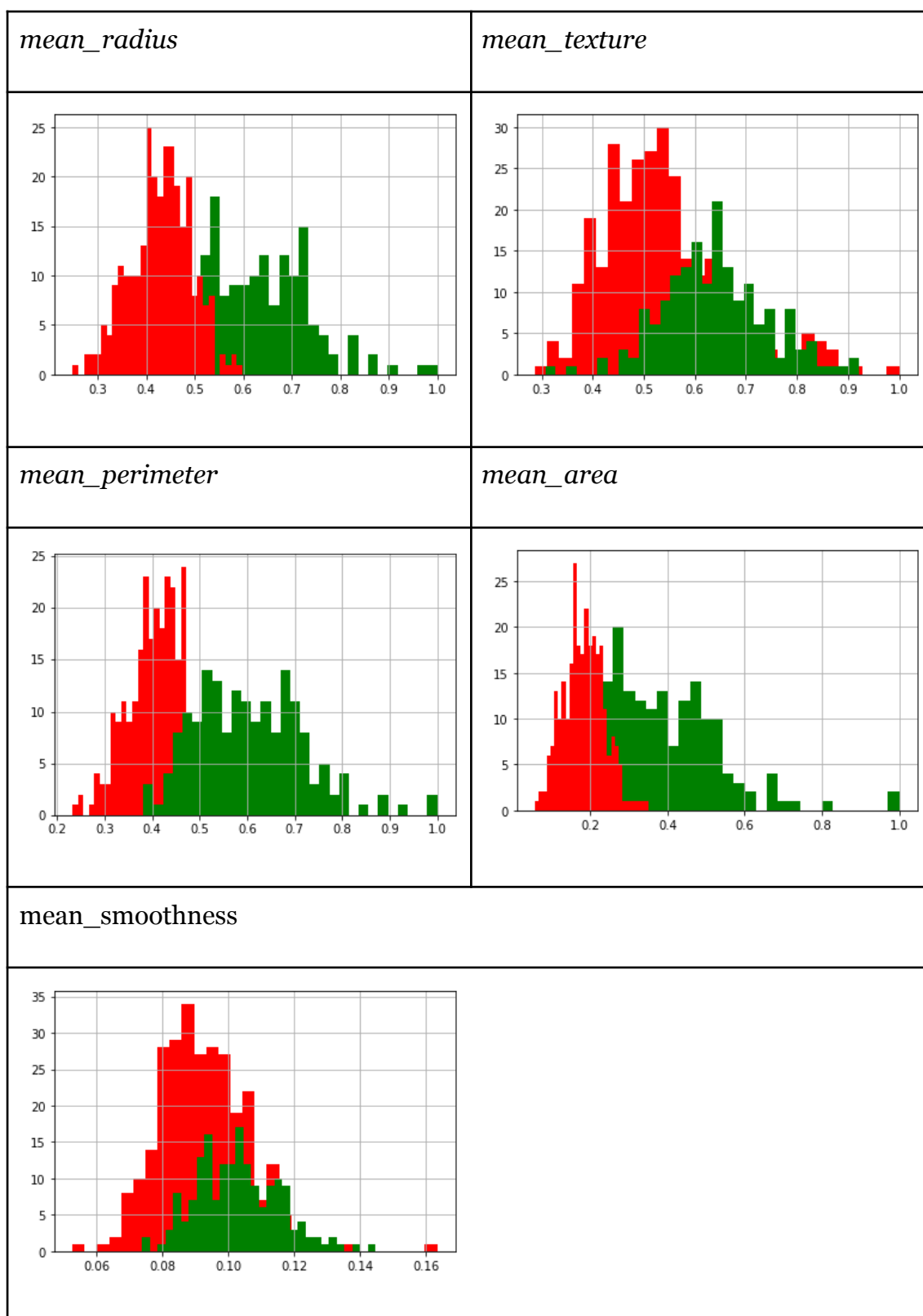
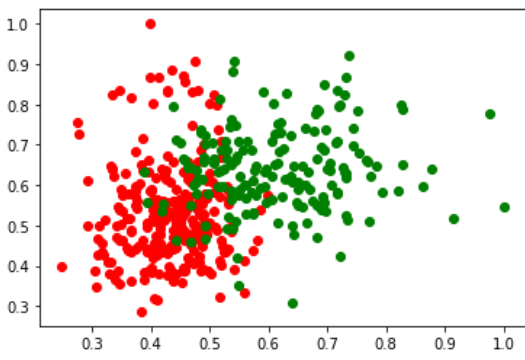
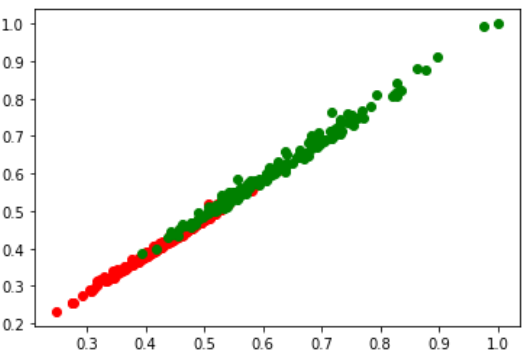
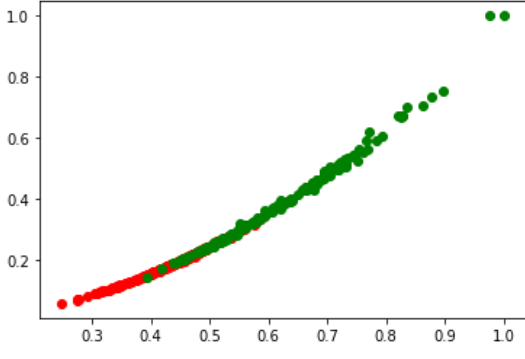
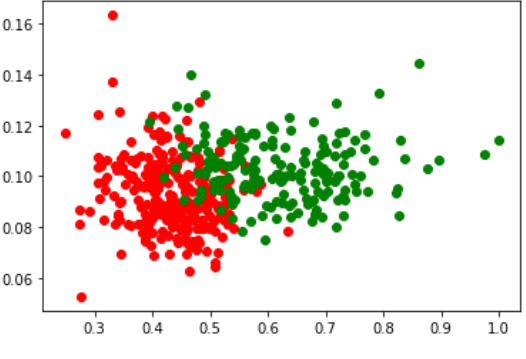
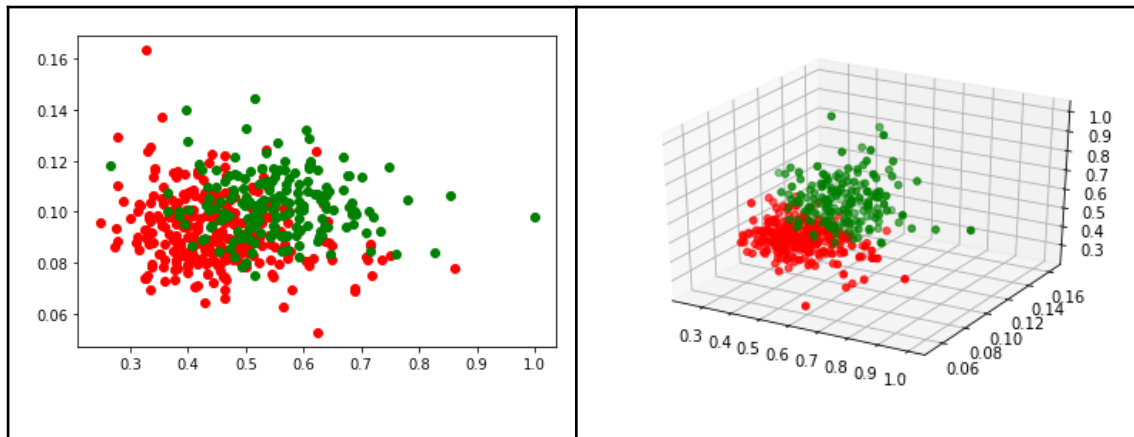


Tabla 3: Frecuencia de las variables según su diagnóstico.

La siguiente visualización, tabla 4, expresa la correlación entre varias variables junto al diagnóstico. Y como se puede observar las variables *mean\_radius*, *mean\_perimeter* y *mean\_area* tienen una correlación muy alta. Esta correlación dificulta la creación del modelo, con datos de entrada redundantes. Así que, para simplificar los datos de entrada y quitar esta redundancia se deberán desechar dos de las tres columnas. El último gráfico de la tabla 4 es la representación en 3D de las variables *mean\_radius*, *mean\_texture*, *mean\_smoothness* y con un color distinto respecto al diagnóstico.

<i>mean_radius</i> y <i>mean_texture</i>	<i>mean_radius</i> y <i>mean_perimeter</i>
	
<i>mean_radius</i> y <i>mean_area</i>	<i>mean_radius</i> y <i>mean_smoothness</i>
	
<i>mean_texture</i> y <i>mean_smoothness</i>	<i>mean_radius</i> , <i>mean_texture</i> y <i>mean_smoothness</i>



**Tabla 4: Correlación entre las diferentes variables.**

En el último gráfico, tabla 4, se puede distinguir claramente el diagnóstico y sus clases. Para la realización de los siguientes modelos se han desechado las columnas *mean\_perimeter* y *mean\_area*, debido a la alta correlación con la variable *mean\_radius*, y así simplificar las variables de entrada. Solo se han utilizado las columnas *mean\_radius*, *mean\_texture* y *mean\_smoothness*.

Una vez se han analizado los datos, se han desechado las columnas con datos redundantes y se han encontrado diferencias entre las diferentes clases a predecir. A partir de esta limpieza de los datos se han implementado y comparado entre sí diferentes modelos predictivos. Para realizar estos modelos se ha utilizado la librería de *Python* Scikit-learn.

### 3.3.1. Support Vector Machine

Se ha realizado un modelo de SVM, con la librería de *Python* Scikit-learn, utilizando las columnas *mean\_radius*, *mean\_texture* y *mean\_smoothness*, normalizadas. Un modelo SVM realiza una representación de los datos y establece un hiperplano separando las diferentes clases. Se ha usado esta técnica debido a que en los gráficos de las tablas 3 y 4 se podían distinguir las diferentes clases. El resultado obtenido es 92% de *Accuracy*, 83% de *Precisión* y 97% de *Recall*. Donde se podría interpretar que, de todos los ejemplos que hay en el *test* de la clase 1, acierta el 97%. Y de todas las predicciones etiquetadas como clase 1 sólo acierta el 83%. El resultado puede variar ya que la parte del *train* y *test* han sido seleccionadas al azar.

Support Vector Machine	Resultados
<i>Accuracy</i>	92,98%
<i>Precision</i>	83,33%
<i>Recall</i>	97,22%

Tabla 5: Resultado del modelo SVM

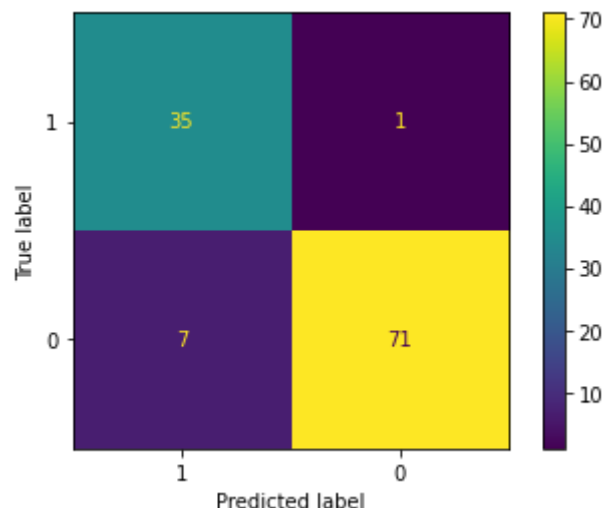


Figura 3: Matriz de confusión del modelo SVM

Una matriz de confusión es la representación de un conjunto de predicciones realizadas por un modelo en una matriz  $N \times N$ , donde  $N$  representa el número de clases a clasificar por el modelo. En un eje está la representación de las predicciones realizadas por el modelo y en el otro eje es la clase verdadera a la que pertenece.

### 3.3.2. Support Vector Machine con Cross Validation

Debido a que el modelo de SVM está supeditado al split aleatorio realizado al *dataset*, el modelo obtenido puede tener algún resquicio de aleatoriedad. Un modelo predictivo debe ser fiable y consistente. Si los datos seleccionados al azar, en distintas ejecuciones, crean un modelo que puede dar tanto un resultado del 100% como de un 0% de *Accuracy*, el modelo no es fiable. Para eliminar esta posibilidad, se ha realizado un modelo SVM con *Cross Validation* con todo el *dataset*.

Support Vector Machine con Cross Validation	Media	Máximo
<i>Accuracy</i>	87,32%	96,55%
<i>Precision</i>	75,94%	92,30%
<i>Recall</i>	92,27%	100%

**Tabla 6: Resultado de SVM con Cross Validation**

Se ha obtenido un *Accuracy* máximo de 96% y una media de 87% de *Accuracy*, con 9,23% puntos en porcentaje de entre la media y el máximo.



### 3.3.3. Support Vector Machine con Cross Validation y Shuffle Split

Otra alternativa realizada ha sido SVM con *Cross Validation y Shuffle Split*. La separación del dataset en train y test se produce de forma aleatoria, y crea un modelo determinado. Y este proceso es repetido varias veces para ver la variación de los resultados del modelo respecto a varias particiones al azar. Para los casos iterables es mejor Shuffle split. En caso de que la separación del dataset se realice una única vez en dos partes train y test, es mejor con la función "train\_test\_split". Al realizar una partición única se crean varios datasets respecto al dataset principal. En cambio al realizar varias iteraciones Shuffle Split crea índices de forma aleatoria, no hace falta guardar en una variable el nuevo dataset. Si el modelo es fiable tendrá pocas variaciones en los resultados y se seleccionará el mejor modelo obtenido respecto a todos los modelos creados.

En el apartado SVM con *Cross validation* la parte de *train* y *test* se realiza partiendo el *dataset* en trozos fijos. Si en una de las partes se concentra una clase en concreto, esa partición será determinante en la clasificación. Con una selección al azar de los datos, el resultado es aleatorio. Pero con la creación de varios modelos se puede verificar si el modelo es fiable.

Support Vector Machine con Cross Validation Shuffle Split	Media	Máximo
<i>Accuracy</i>	91,49%	94,73%
<i>Precision</i>	85,25%	91,30%
<i>Recall</i>	91,93%	100%

**Tabla 7: Resultado SVM con Cross Validation y Shuffle Split**

El resultado obtenido es 94,73% máximo de *accuracy* y 91,49% de media de *accuracy* con 3,24% puntos en porcentaje entre la media y el máximo. Es un menor porcentaje que SVM con *Cross Validation*.

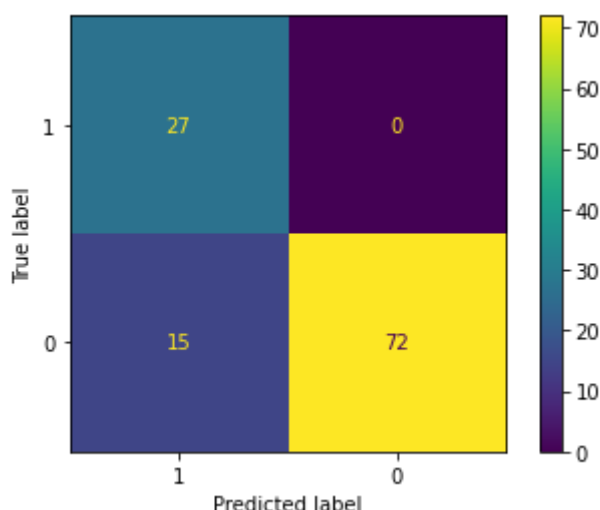


### 3.3.4. Regresión Logística

Se ha buscado comparar el uso del modelo SVM con otro modelo distinto en este *dataset*. Para ello se ha realizado un modelo de regresión logística, donde predice, en este caso, una variable categórica. La variable categórica a predecir es el diagnóstico, si el sujeto padece o no cáncer de mama.

Regresión Logística	
<i>Accuracy</i>	86,84%
<i>Precision</i>	64,28%
<i>Recall</i>	100%

**Tabla 8: Resultado del modelo de regresión logística**



**Figura 4: Matriz de confusión del modelo de regresión logística**

El resultado obtenido es un peor modelo respecto a SVM, con peores resultados en todas las medidas de evaluación. Con una partición aleatoria, el modelo creado puede ser impreciso. Para ello, lo mejor es realizar los mismos procedimientos efectuados anteriormente.

### 3.3.5. Regresión Logística con Cross Validation

La comparación de la Regresión Logística con *Cross Validation* respecto a SVM con *Cross Validation* es:

- Un *accuracy* de 84,72% de media y de 96,55% como máximo, frente a 87,32% de media y un máximo de 96,55% en SVM.
- Una *Precision* de media de 70,27 % y 84,61% de máximo en regresión logística. En SVM una media de 75,94% y un máximo de 92,30%.
- Un *Recall* de 91,64% de media y 100% como máximo en regresión logística. En SVM da como *Recall* 92,27% de media y 100% como máximo.

El resultado SVM con *Cross Validation* es mejor que una regresión lineal con *Cross Validation* ya que tiene mayor *Accuracy*, *Recall* y *Precision*.

Pero hay que destacar una mayor variación entre el *Accuracy* medio con el *Accuracy* máximo en la regresión logística con *Cross Validation* (11,82%) respecto a SVM con *Cross Validation* (9,23%).

Regresión Logística con <i>Cross Validation</i>	Media	Máximo
<i>Accuracy</i>	84,72%	96,55%
<i>Precision</i>	70,27%	84,61%
<i>Recall</i>	91,64%	100%

Tabla 9: Resultado del modelo de regresión logística con *Cross Validation*.

### 3.3.6. Regresión Logística con Cross Validation y Shuffle Split

La comparación entre Regresión Logística con *Cross Validation y Shuffle Split* y SVM *Cross validation y Shuffle Split* es la siguiente:

- Un *Accuracy* de media 86,66% y 91,22% de máximo. Un 91,49% de media y 94,73% de máxima en SVM.
- Un *Precision* de 66,90% como media y 78,94% de máxima. Y 85,25% de media y 91,30% de máximo en SVM.
- Un *Recall* de 96,80% de media y 91,93% de media en SVM. Ambos con 100% máximo de *Recall*.

SVM con *Cross Validation y Shuffle Split* obtiene mejores resultados que una regresión logística con *Cross Validation y Shuffle Split*. Aunque se obtenga un mejor *Recall*, todas las demás medidas de evaluación son inferiores.

Regresión Logística con <i>Cross Validation y Shuffle Split</i>	Media	Máximo
<i>Accuracy</i>	86,66%	91,22%
<i>Precision</i>	66,90%	78,94%
<i>Recall</i>	96,80%	100%

**Tabla 10: Resultado del modelo de regresión logística con *Cross Validation con Shuffle Split*.**

Se obtiene un *accuracy* de media de alrededor de 84-86%, tanto en regresión logística con *Cross Validation* con y sin *Shuffle Split*. El resultado obtenido es siempre un mejor *Recall* que *Precision*. Teniendo en cuenta que hay dos clases, el fallo obtenido es debido a que la clase con la etiqueta 1 se clasifican como clase 0.

### 3.3.7. K-means

Se ha utilizado *K-means* para la evaluación del *dataset*, pero se ha descartado rápidamente debido a que es muy inestable, y en la ejecución de un mismo código puede dar resultados muy distintos. La idea principal era realizar una agrupación en dos conjuntos de manera no supervisada. Pero con la ejecución del mismo código donde lo único que varía es la selección del *train* y *test*, se han obtenido resultados completamente distintos. El *accuracy* puede llegar a ser de más del 80% o incluso no llegar a 10%. Esto ocurre debido a que la inicialización de los puntos se realiza al azar y no llega a distinguir las diferentes clases.

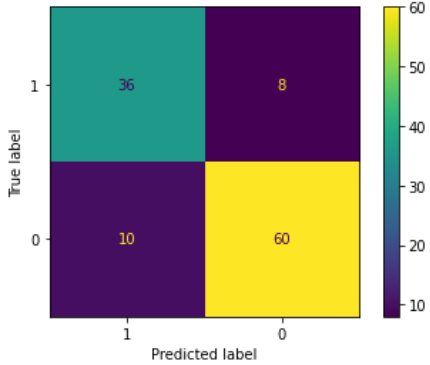
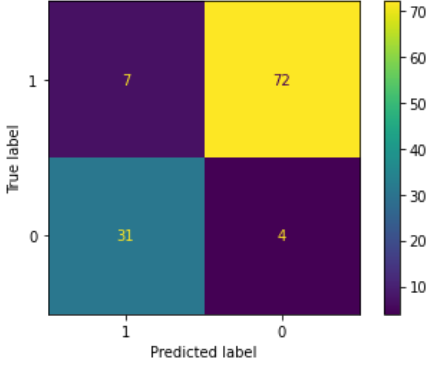
Kmeans	Ejecución buena	Ejecución mala																		
<i>Accuracy</i>	84,21%	09,64%																		
<i>Precision</i>	78,26%	18,42%																		
<i>Recall</i>	81,81%	8,86%																		
Gráficos	 <p>Confusion matrix for 'Ejecución buena':</p> <table border="1"> <thead> <tr> <th></th> <th>Predicted 1</th> <th>Predicted 0</th> </tr> </thead> <tbody> <tr> <th>True 1</th> <td>36</td> <td>8</td> </tr> <tr> <th>True 0</th> <td>10</td> <td>60</td> </tr> </tbody> </table>		Predicted 1	Predicted 0	True 1	36	8	True 0	10	60	 <p>Confusion matrix for 'Ejecución mala':</p> <table border="1"> <thead> <tr> <th></th> <th>Predicted 1</th> <th>Predicted 0</th> </tr> </thead> <tbody> <tr> <th>True 1</th> <td>7</td> <td>72</td> </tr> <tr> <th>True 0</th> <td>31</td> <td>4</td> </tr> </tbody> </table>		Predicted 1	Predicted 0	True 1	7	72	True 0	31	4
	Predicted 1	Predicted 0																		
True 1	36	8																		
True 0	10	60																		
	Predicted 1	Predicted 0																		
True 1	7	72																		
True 0	31	4																		

Tabla 11: Resultado del modelo *K-means* en varias ejecuciones.

### 3.3.8. Árbol de decisión

Otro modelo a explorar es un árbol de decisión con el *dataset* completo, sin la normalización, ni la manipulación alguna de los datos. Una de las observaciones en este modelo es la variación del resultado. Sin llegar a ser tan grande como en el caso de *K-means*, hay una variación entre 85% y 95%.

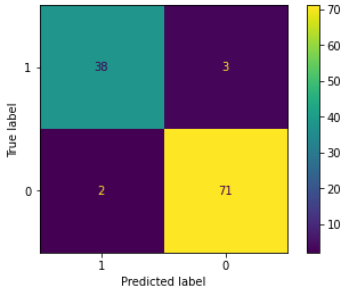
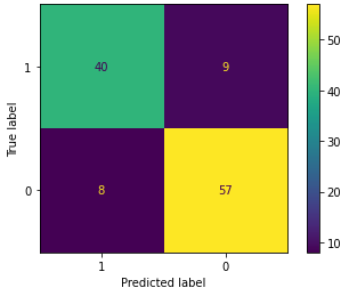
Árbol de decisión	Ejecución buena	Ejecución mala																		
<i>Accuracy</i>	95,61%	85,08%																		
Precision	95%	81,63%																		
<i>Recall</i>	92,58%	83,33%																		
	 <p>Confusion matrix for 'Ejecución buena':</p> <table border="1"> <thead> <tr> <th></th> <th>Predicted label 1</th> <th>Predicted label 0</th> </tr> </thead> <tbody> <tr> <th>True label 1</th> <td>38</td> <td>3</td> </tr> <tr> <th>True label 0</th> <td>2</td> <td>71</td> </tr> </tbody> </table>		Predicted label 1	Predicted label 0	True label 1	38	3	True label 0	2	71	 <p>Confusion matrix for 'Ejecución mala':</p> <table border="1"> <thead> <tr> <th></th> <th>Predicted label 1</th> <th>Predicted label 0</th> </tr> </thead> <tbody> <tr> <th>True label 1</th> <td>40</td> <td>9</td> </tr> <tr> <th>True label 0</th> <td>8</td> <td>57</td> </tr> </tbody> </table>		Predicted label 1	Predicted label 0	True label 1	40	9	True label 0	8	57
	Predicted label 1	Predicted label 0																		
True label 1	38	3																		
True label 0	2	71																		
	Predicted label 1	Predicted label 0																		
True label 1	40	9																		
True label 0	8	57																		

Tabla 12: Resultado del modelo árbol de decisión.

### 3.3.9. Regresión Lineal sin normalizar

Un modelo de regresión lineal realiza una predicción de una variable dependiente en relación con el resto de variables independientes. Se ha utilizado el modelo de regresión lineal de diferentes formas. En este apartado se ha usado el *dataset* sin normalizar y utilizando solo las columnas *mean\_radius*, *mean\_texture* y *mean\_smoothness*, dando un resultado de alrededor de un 86% en *accuracy*. Pero lo mejor es interpretar el resultado comparando la regresión lineal donde el *dataset* no está normalizado con una opción donde sí lo esté. Esta comparación se realiza en el siguiente apartado. Así, es posible observar cómo influye esta modificación de las columnas en los resultados. Si el resultado es mayor o igual a 0,5 se tomará como resultado un tumor maligno, y si el resultado es inferior a 0,5 el resultado será benigno.

Regresión lineal sin normalizar	
<i>Accuracy</i>	92,98%
<i>Precision</i>	86%
<i>Recall</i>	97,72%

Tabla 13: Resultado de un modelo de regresión lineal sin normalizar.

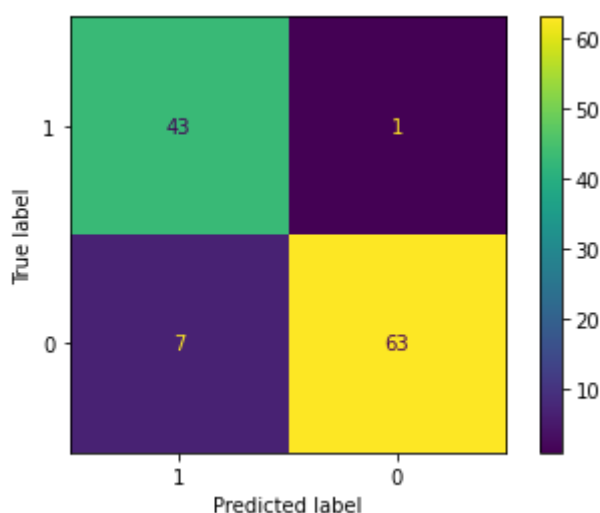


Figura 5: Matriz de confusión de un modelo de regresión lineal sin normalizar.

## 3.3.10. Regresión Lineal normalizado

Siguiendo con el uso del modelo de regresión lineal, se ha normalizado el *dataset* y se han mantenido solo las columnas *mean\_radius*, *mean\_texture* y *mean\_smoothness*. Comparando este modelo donde los parámetros están normalizados, con el modelo de regresión lineal donde los parámetros no están normalizados.

Regresión lineal	Dataset normalizado	Dataset sin normalizar
<i>Accuracy</i>	92,10%	92,98%
<i>Precision</i>	78,57%	86%
<i>Recall</i>	100%	97,72%

Tabla 14: Resultado del modelo de regresión lineal.

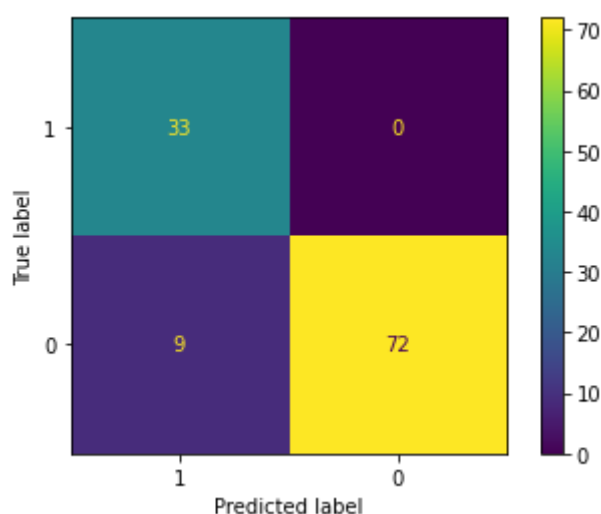


Figura 6: Matriz de confusión del modelo de regresión lineal normalizado.

Dependiendo de la ejecución los resultados pueden cambiar, pero después de varias ejecuciones, comparando los modelos de regresión lineal con los datos normalizados y sin normalizar la normalización y simplificación de los datos de entrada al modelo para obtener mejores resultados debido a que hay una menor variación.



### 3.3.11. Regresión Lineal con Cross Validation

Se ha aplicado *Cross Validation* al modelo de regresión lineal con el dataset normalizado y el uso de las columnas *mean\_radius*, *mean\_texture* y *mean\_smoothness*, con una separación del dataset en 10 partes. Si se comparan los resultados por la media y el valor máximo de los distintos parámetros, da mejores resultados que el resto de los modelos anteriores. Pero en la Figura 7 se puede ver que dos de los modelos obtenidos reflejan valores atípicos (*outliers*).

Regresión lineal con Cross Validation	Media	Máximo
<i>Accuracy</i>	91,48%	96,55%
<i>Precision</i>	82,91%	92,30%
<i>Recall</i>	95,77%	100%

**Tabla 15: Resultado del modelo de regresión lineal con *Cross Validation*.**

Cross Validation	SVM	Regresión logística	Regresión lineal
<i>Accuracy</i>	87,32%	84,72%	91,48%
<i>Precision</i>	75,94%	70,27%	82,91%
<i>Recall</i>	92,27%	91,64%	95,77%
<i>Accuracy</i> medio - <i>Accuracy</i> máximo	9,23%	11,82%	5,06%

**Tabla 16: Comparación de los modelos creados con *Cross Validation*.**

Los valores máximos en todos los modelos con *Cross Validation* son los mismos, 96,55% *Accuracy*, 100% *Recall* y 84,61% o 92,30% *Precision*. Esto se debe a que el *Recall* y *Precision* máximos no están vinculados al modelo con mayor *Accuracy*, si no que es el resultado máximo que se puede obtener. La



justificación del resultado 100% *Recall* es debido a una clasificación de la parte test a una clase predeterminada de forma directa en caso de duda. Los de *Accuracy* y *Precision* obtenidos coinciden debido a que existen ciertos bloques donde todos los datos tienen una fácil clasificación menos uno o dos elementos. Así que para evaluar cual de los modelos es mejor realizar una comparación de las medias.

### 3.3.12. Regresión Lineal con Cross Validation y Shuffle Split

Para finalizar con el modelo de regresión lineal se ha aplicado *Cross Validation* y *Shuffle split* a las columnas *mean\_radius*, *mean\_texture* y *mean\_smoothness*, normalizadas.

Regresión Lineal con <i>Cross Validation</i> y <i>Shuffle Split</i>	Media	Máximo
<i>Accuracy</i>	85,96%	90,35%
<i>Precision</i>	66,72%	78,94%
<i>Recall</i>	95,44%	100%

**Tabla 17: Resultado del modelo de regresión lineal con *Cross Validation* y *Shuffle Split*.**

Cross Validation Shuffle Split	SVM	Regresión logística	Regresión lineal
<i>Accuracy</i>	91,49%	86,66%	85,96%
<i>Precision</i>	85,25%	66,90%	66,72%
<i>Recall</i>	91,93%	96,80%	95,44%
<i>Accuracy</i> medio - <i>Accuracy</i> máximo	3,24%	4,56%	4,39%

**Tabla 18: Comparación de los modelos creados con *Cross Validation* y *Shuffle Split*.**

La comparación de los distintos modelos creados mediante Cross Validation con Shuffle Split destaca SVM frente a los modelos de regresión lineal y regresión logística. Se obtiene un mayor *Accuracy*, unas medidas de Precision y Recall más balanceadas. Además, el modelo de SVM da un resultado de 85% Precision y 91% Recall. Respecto a los demás modelos donde el resultado es de alrededor de 65% Precision y 95% Recall. A todo esto se le suma una menor variación entre el Accuracy medio y el Accuracy máximo.

### 3.3.13. Random Forest

El último modelo aplicado a todo el dataset sin ninguna modificación ha sido *Random Forest*. Se ha comparado el resultado con el modelo árbol de decisión, ya que un modelo de *Random Forest* construye árboles de decisión de forma aleatoria. El resultado que se obtiene es una variación parecida a la encontrada en el modelo del árbol de decisión, alrededor de 85% y 95%.

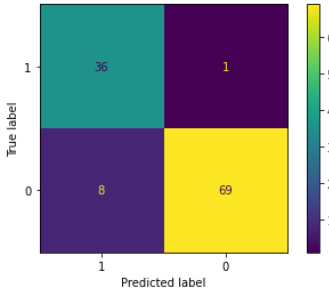
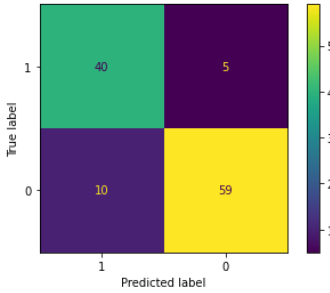
<i>Random Forest</i>	Ejecución buena	Ejecución mala
<i>Accuracy</i>	92,10%	84,21%
<i>Precision</i>	97,29%	80%
<i>Recall</i>	81,81%	88,88%
		

Tabla 19: Resultado del modelo de *Random Forest*.

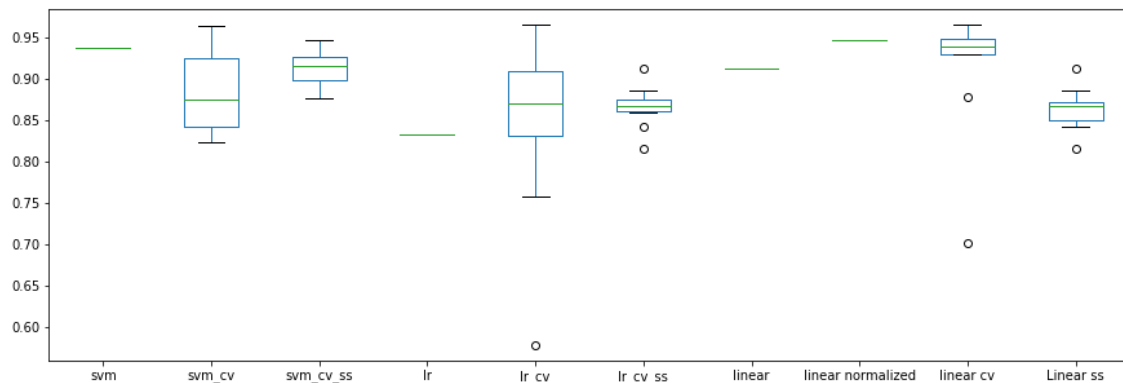
### 3.3.14. Resultados

En la tabla 18 se expresa la media de *Accuracy*, *Recall* y *Precision* de todos los modelos. Los resultados pueden cambiar respecto a los apartados anteriores debido a que la extracción de los resultados de la tabla 18 se ha realizado en ejecuciones distintas.

<b>Modelos</b>	<b><i>Accuracy</i> Media</b>	<b><i>Precision</i> Media</b>	<b><i>Recall</i> Media</b>
<i>Support Vector Machine</i>	92,98%	83,33%	97,22%
<i>Support Vector Machine CV</i>	87,32%	75,94%	92,27%
<i>Support Vector Machine CV Shuffle Split</i>	91,49%	85,25%	91,93%
Regresión Logística	86,84%	64,28%	100%
Regresión Logística <i>CV</i>	84,72%	70,27%	91,64%
Regresión Logística <i>CV Shuffle Split</i>	86,66%	67,50%	96,48%
<i>Kmeans</i>	93,85%	91,66%	89,18%
Árbol de decisión	92,98%	96%	88,88%
Regresión Lineal	92,98%	86%	97,72%
Regresión Lineal normalizado	92,10%	78,57%	100%
Regresión Lineal <i>CV</i>	91,48%	82,91%	95,77%
Regresión Lineal <i>CV Shuffle Split</i>	85,96%	66,72%	95,44%
<i>Random Forest</i>	86,84%	80%	88,88%

Tabla 20: Comparación entre las diferentes medias.

La distribución de los *accuracy* de los diferentes modelos está representada en la figura 7 mediante diagramas de cajas. Los resultados puede que no se adapten a los expresados en los apartados anteriores, ya que el gráfico se ha realizado en una ejecución distinta. Aun así los resultados no difieren mucho de los expuestos anteriormente.



**Figura 7: Comparación de los *accuracy* entre los diferentes modelos.**

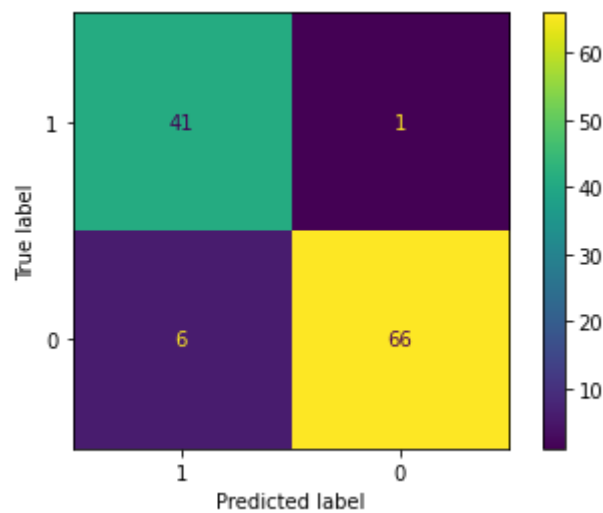
Se han descartado los modelos de Árbol de decisión, Random Forest y K Means, por su gran variabilidad. El resto de los modelos se han descartado por los siguientes motivos:

- Regresión Logística en todas sus formas ya que se obtienen malos resultados y un gran rango de *accuracy* (etiquetas lr, lr\_cv y lr\_cv\_ss en la figura 7).
- SVM con Cross validation tiene rango muy alto de *accuracy* (etiqueta svm\_cv en la figura 7).
- Regresión lineal sin los datos normalizados para simplificar la entrada (etiqueta linear en la figura 7).
- Regresión lineal con Cross Validation debido a la obtención de modelos con valores anómalos muy bajos (etiqueta linear cv en la figura 7).
- Regresión lineal con Cross Validation Shuffle Split por un mal resultado (etiqueta linear ss en la figura 7).
- Otros modelos que se podrían seleccionar son SVM y SVM con Cross Validation y Shuffle Split (etiqueta svm y svm\_cv\_ss en la figura 7) pero se han descartado por los resultados ligeramente inferiores.

Así que el modelo seleccionado ha sido Regresión lineal con el dataset normalizado. Se ha vuelto a crear el modelo con la finalidad de guardarlo para encapsular en una imagen de contenedor Docker. También se ha vuelto a predecir la parte de *test* con el resultado, en la tabla 19 y Figura 8.

Modelo seleccionado	Regresión lineal normalizado
<i>Accuracy</i>	93,85%
<i>Precision</i>	87,23%
<i>Recall</i>	97,61%

**Tabla 21: Modelo de regresión lineal seleccionado.**



**Figura 8: Matriz de confusión del modelo de regresión lineal seleccionado.**

### 3.4. Predicción de enfermedades respiratorias mediante radiografías

Este dataset [14] contiene imágenes de radiografías del tórax que se clasifican en cuatro clases: normal donde el paciente no tiene ninguna enfermedad; COVID, neumonía y tuberculosis, donde el paciente padece una de estas enfermedades, la distribución de las imágenes está representada en la tabla 20. Para este *dataset* se ha usado la capa gratuita de Google Colab, para entrenar las redes neuronales mediante GPUs.

	<i>Covid</i>	<i>Normal</i>	<i>Neumonía</i>	<i>Tuberculosis</i>
<i>Train</i>	460	1341	3875	650
<i>Test</i>	106	234	390	41
<i>Validation</i>	10	8	8	12

Tabla 22: Distribución de las imágenes en las diferentes clases del *dataset*.

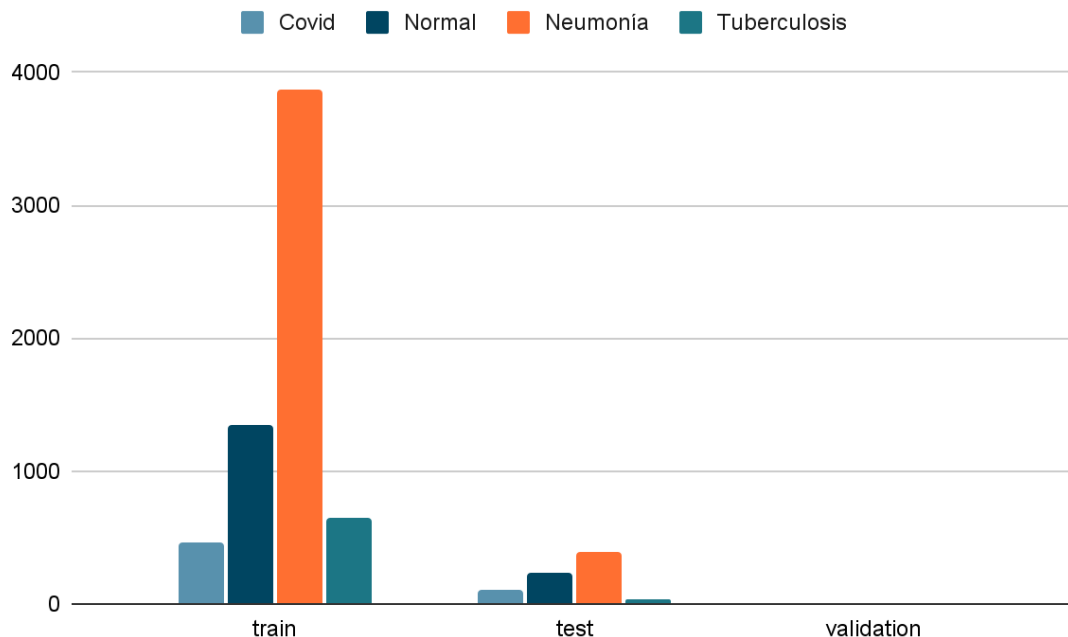
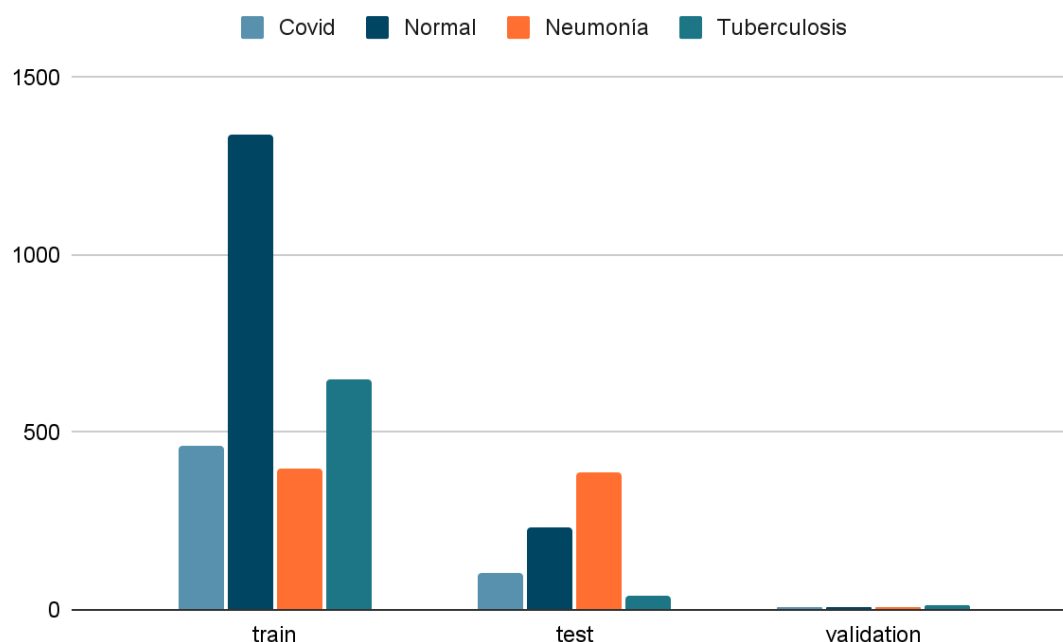


Figura 9: Distribución de las radiografías en las diferentes clases del *dataset*.

Para la clasificación de imágenes, las redes neuronales convolucionales dan un buen rendimiento, ya que buscan patrones alrededor de cada *píxel*. Las redes neuronales se han entrenado con las GPUs que utiliza Google Colab en la capa gratuita, utilizando la biblioteca de TensorFlow. Y las métricas que proporciona por Google Colab es el uso del disco de 23,46 GB y el uso de RAM de 4,35 GB, no especifica si es RAM del sistema o RAM de la GPU. El primer entrenamiento se ha realizado con todas imágenes de la parte de *train*. El resultado obtenido era un *accuracy* menor a 85%. El principal problema encontrado era entre la clase normal y la clase neumonía, ya que las imágenes de la clase normal se califican como neumonía. Para corregir este problema, se ha pensado en darle más peso a la clase normal, insertando imágenes giradas de la clase normal, tanto de forma aleatoria en cualquier dirección como solo cambiando de lado. Con un resultado peor, se ha optado por la estrategia contraria, quitarle valor a la clase neumonía. Para realizar este propósito se han descartado imágenes de la clase neumonía hasta tener 400. Porque la clase neumonía tiene 3875 imágenes, y es la clase con mayor número de imágenes casi triplicando el número de imágenes que tiene la clase normal. El inconveniente de quitar imágenes de una clase da la posibilidad de una mala clasificación debido a datos insuficientes. Una mejor estrategia sería introducir imágenes del resto de clases en vez de quitarlas, pero en este caso no es posible.



**Figura 10: Distribución de las radiografías en las diferentes clases del *dataset* para el modelo final.**

Para perfeccionar la red neuronal [Anexo 8.1] se ha cambiado su tamaño introduciendo nuevas capas hasta llegar a un total de 53.094.120 parámetros donde se entrenan 53.084.992 y 9.128 de los parámetros no son entrenados.



Además, se ha ajustado el entrenamiento a 25 *epochs*, y 100 *batch\_size*. Los Epoch son el número de ciclos en el que todos los datos de entrenamiento pasan por la red neuronal. Y *batch\_size* es el número de entradas de datos que se prueban en la red neuronal entre actualización de los pesos.

Así que la red neuronal se entrena introduciendo todos los datos de entrenamiento 25 veces y cada 100 imágenes procesadas cambia los pesos de la red neuronal.

El problema entre la clase normal y la neumonía sigue existiendo, pero es mucho menor llegando a un *accuracy* del 90%.

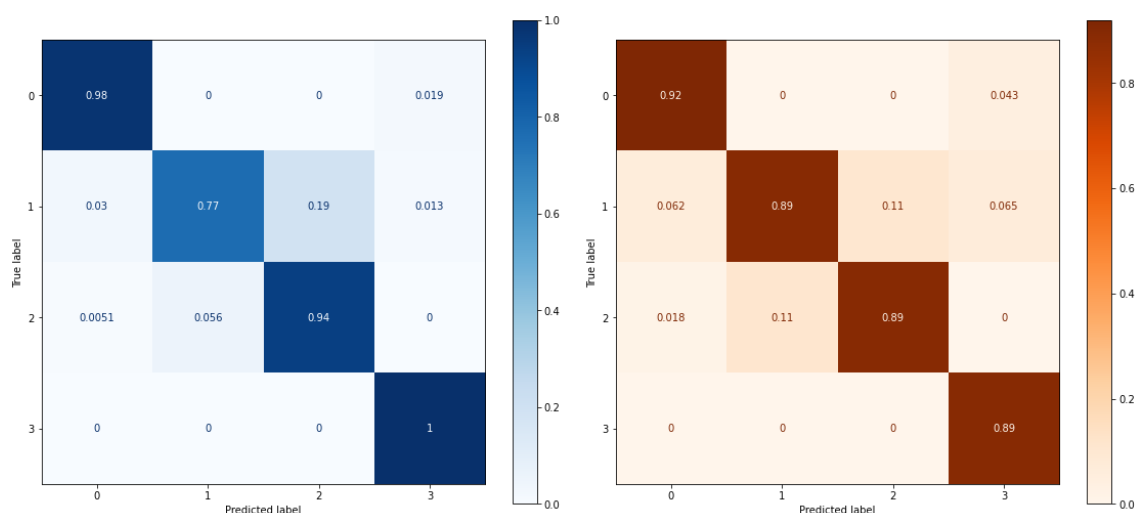
	Predicción Covid	Predicción Normal	Predicción Neumonía	Predicción Tuberculosis
Covid	103	0	0	2
Normal	7	180	44	3
Neumonía	2	22	366	0
Tuberculosis	0	0	0	41

**Tabla 23: Matriz de confusión sobre el resultado de la red neuronal.**

	<i>Precision</i>	<i>Recall</i>	<i>Accuracy</i>
Covid	90%	98%	
Normal	89%	77%	
Neumonía	89%	94%	
Tuberculosis	89%	100%	
Total			90%

**Tabla 24: Resultados de la red neuronal.**





**Figura 11: Recall y Precision de las diferentes clases del dataset de enfermedades respiratorias.**

### 3.5. Predicción de Alzheimer mediante imagen de resonancia magnética.

El objetivo de este último modelo es realizar una predicción sobre un paciente respecto si tiene o no Alzheimer y el desarrollo de la enfermedad sobre el paciente. Para este dataset se ha usado la capa gratuita de *Google Colab*, para entrenar las redes neuronales mediante *GPUs*. En el último dataset se encuentran 6400 imágenes de resonancia magnética del cerebro, distribuidas en 4 clases:

- Demencia leve con 896 imágenes.
- Demencia moderada con 64 imágenes.
- No demencia con 3200 imágenes.
- Demencia muy leve con 2240 imágenes.

Todas las imágenes se encuentran dentro de una carpeta. Se debe realizar un desdoble entre *train* y *test*. Para realizar este desdoble se ha utilizado la librería *splitfolders* de Python en una proporción de 80% *train*, 10% validación y 10% *test*. La red neuronal convolucional [Anexo 8.2] se compone de un total de 72.744.680 parámetros, de los cuales 72.739.648 son parámetros entrenados y 5.032 no. El entrenamiento ha sido de 20 *epochs* y un *batch size* de 100 para obtener un buen resultado. Si los parámetros de entrenamiento son diferentes

los resultados serán peores, ya que la red neuronal sería distinta, con otros pesos que realicen una peor predicción. Según los datos en Google Colab, se usan 25,05 GB de disco y 4,97 GB de RAM. Y se ha llegado a un *accuracy* del 96%. La clase con mayor posibilidad de fallo es demencia muy leve con una *Precision* de 94% y un *Recall* de 95%.

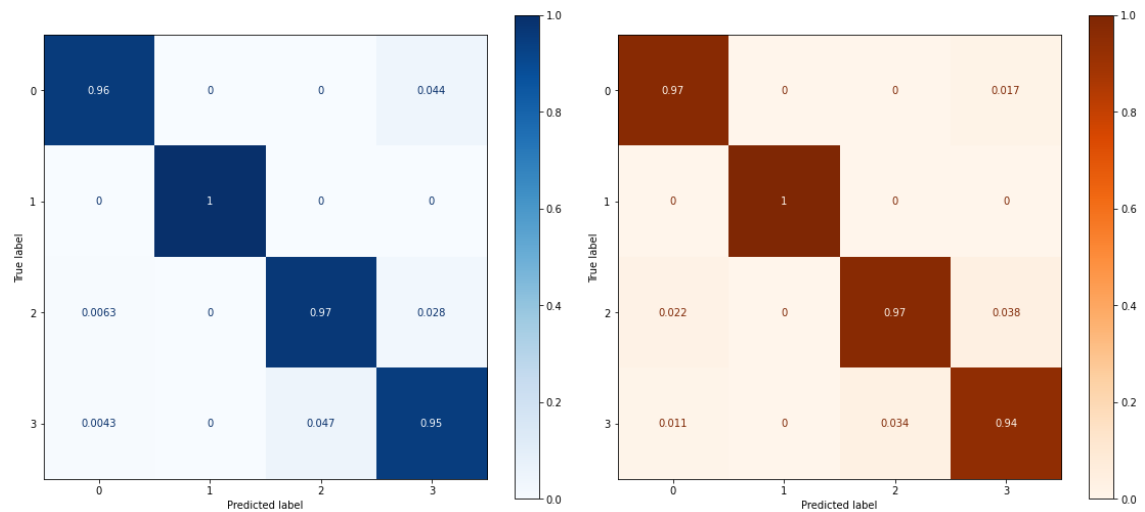
	Predicción Demencia leve	Predicción Demencia moderada	Predicción No demencia	Predicción Demencia muy leve
Demencia leve	86	0	0	4
Demencia moderada	0	7	0	0
No demencia	2	0	309	9
Demencia muy leve	1	0	11	222

**Tabla 25: Matriz de confusión sobre el resultado de la red neuronal.**

	<i>precisión</i>	<i>recall</i>	<i>accuracy</i>
Demencia leve	97%	96%	
Demencia moderada	100%	100%	
No demencia	97%	97%	
Demencia muy leve	94%	95%	
Total			96%

**Tabla 26: Resultados de la red neuronal.**





**Figura 12: Recall y Precision de las diferentes clases del dataset para la predicción de Alzheimer.**

## 3.6. Despliegue de OSCAR con IM

Una vez obtenidos los modelos se busca adaptar estos modelos para que su despliegue y uso, sean fáciles y rápidos en un entorno *Cloud*. Dentro de los modelos de servicio que un entorno *Cloud* ofrece es interesante aplicar computación serverless, pero aplicado a la ejecución de aplicaciones basadas en contenedores. Esto permite delegar el aprovisionamiento de recursos de cómputo en la plataforma serverless, facilitando así el despliegue y operación de aplicaciones basadas en contenedores para la ejecución de la fase de predicción de los modelos creados.

Los principales proveedores *Cloud* dan un servicio de *FaaS* con el uso de contenedores [31, 32, 33, 34], también comparten las diferentes maneras de desencadenar [35, 36, 37] estas funciones, con llamadas http, eventos en el sistema de almacenamiento de objetos o con el manejo de eventos en otros servicios. Pero cada proveedor identifica sus servicios con diferentes nombres.

	Amazon Web Services	Microsoft Azure	Google Cloud	On-premises
Herramientas FaaS	Lambda	Microsoft Azure Function	Cloud Run	OSCAR
Llamadas Http	Sí	Sí	Sí	Sí
Sistema de almacenamiento de documentos	Sí, S3	Sí, Azure Blob Storage	Sí, Cloud Storage	MinIO
Eventos	Sí, Event Bridge	Sí, Azure Event Grid	Sí, Eventarc	No

**Tabla 27: Nombre de los servicios en diferentes plataformas Cloud.**

La ejecución *FaaS* en *Google Cloud* se realiza mediante *Google Cloud Function*. Pero *Google Cloud Function* no soporta la ejecución de contenedores. Para la ejecución de contenedores es necesario usar el servicio de *Google, Cloud Run* con la gestión de los eventos que hace *Eventarc*. El equivalente a *Google Run* en *Amazon Web Services* es *Faregate* pero el precio es un inconveniente ya que según la calculadora de *Amazon* [45] “El precio es por segundo, con un mínimo de 1 minuto (*Linux*) y de 15 minutos (*Windows*).” Si la ejecución del container



dura 1 segundo la facturación es de 1 minuto. En *Microsoft Azure* la ejecución *serverless* de contenedores conforme a demanda, se realiza con *Azure Kubernetes Service* usando nodos virtuales [46] pero tiene un coste mínimo en la creación y el mantenimiento de un clúster de *Kubernetes*. Presentados todos los servicios posibles a utilizar se han descartado *Fargate* y *Azure Kubernetes* por el coste y *Google Cloud Function* por no ser compatible con contenedores. Los servicios restantes son, el servicio *Lambda* de *Amazon Web Services*, el servicio *Microsoft Azure Function* de *Microsoft Azure*. Y *Cloud Run* de *Google Cloud*. Aunque la finalidad de los servicios y la ejecución del código tiene el mismo resultado, cada *Cloud* tiene diferentes limitaciones y costes. En cambio, con el uso de *Cloud* privadas, las limitaciones expuestas son las del propio servidor o la cuota impuesta al usuario. Comparando estas limitaciones y costes, el uso de OSCAR está justificado en este proyecto.

Límite de recursos	<i>Amazon Web Services</i> [38]	<i>Microsoft Azure (Consumption Plan)</i> [40]	<i>Google Cloud</i> [39]	OSCAR
Memoria asignada	1.769 MB por CPU	1,5GB	32 GiB	Memoria del sistema
Tiempo máximo de ejecución	15 minutos	10 minutos	Ilimitado	Ilimitado
Ejecuciones concurrentes	1.000	200	100	Tantas como el sistema soporte
Tamaño del contenedor	10 GB	No especifica	No especifica	Indefinido

**Tabla 28: Limitaciones de los proveedores Cloud.**

Respecto al coste de la infraestructura hay que tener en cuenta que cada *Cloud* tiene un sistema de facturación distinto. Por ejemplo, *Microsoft Azure* tiene diferentes planes de pago [41] dependiendo de las necesidades. En cambio, en *Google Cloud* y *Amazon Web Services* el precio del servicio cambia dependiendo de la localización del servidor, el tiempo de ejecución y recursos utilizados. Además en *Amazon Web Services* también influye la arquitectura donde se ejecuta la aplicación.

En la tabla 27 se compara la capa gratuita y el coste de los servicios, el coste está expresado en *USD*, dólares estadounidenses. Pero como el coste puede variar en la misma *Cloud*, se han tomado como referencia las siguientes características:

- En *Google Cloud*, servidores con una localización de nivel 1.

- En *Microsoft Azure*, se ha consultado el plan *Consumption*.
- En *Amazon Web Services* se han consultado los costes con los servidores del norte de Virginia con una arquitectura x86 y con un precio válido hasta los primeros 6 mil millones de GB/segundo por mes.

Precio	<i>Lambda</i> [42]	<i>Microsoft Azure Function</i> [43]	<i>Cloud Run</i> [44]
Capa Gratuita (millón de ejecuciones y GB/s)	1 y 400.000 GB/s	1 y 400.000 GB/s	2 y 360.000 GB/s
Precio por millón de ejecuciones	0,20	0,20	0,40
Precio por tiempo de ejecución (GB/segundo)	$4,1667 \cdot 10^{-6}$	$1,6 \cdot 10^{-5}$	$2,50 \cdot 10^{-6}$

**Tabla 29: Coste de los servicios *Cloud*.**

Utilizando las calculadoras de cada *Cloud* se pueden comparar los costes de los diferentes servicios con los mismos requisitos. En la tabla 28 se ha realizado una comparación de los servicios con 500.000 ejecuciones al mes y 45 segundos de ejecución. Algunas celdas están en blanco porque en *Microsoft Azure Function* en el plan *Consumption* tiene un máximo de 1,5 GB de memoria RAM y en *Google Cloud* solo se puede asignar una cantidad de memoria predeterminada.

	1GB	1,5GB	2GB	3GB	4GB
<i>Lambda</i> [47]	368,33	555,83	743,33	1118,34	1493,34
<i>Microsoft Azure Function</i> [48]	353.60	533.60			
<i>Cloud Run</i> [49]	66.78		99.86		166.68

**Tabla 30: Simulación de los costes en \$.**

Se ha decidido utilizar OSCAR [2] porque:

- Es código abierto y utiliza licencia Apache 2.0

- El uso de contenedores permite realizar una configuración completa del entorno de ejecución.
- El uso on-premises no tiene una limitación en la capacidad de los contenedores, en el tiempo de ejecución y a coste o, si bien depende de la capacidad de la infraestructura computacional disponible en la organización.
- Fácil despliegue tanto en local como en diferentes proveedores *Clouds* gracias a IM.
- Interacción con un cluster de OSCAR mediante una API y también a través de herramientas de línea de comandos.

Se ha utilizado una de las plataformas *Cloud on-premises* del grupo de investigación GRyCAP, implementada sobre OpenNebula, para minimizar costes al no utilizar los servicios de un proveedor Cloud. El despliegue de OSCAR se ha realizado mediante el cliente web IM Dashboard [23] que interactúe con IM.

Para el despliegue de OSCAR mediante IM es necesario tener credenciales de la plataforma *cloud* en la que vas a desplegar. En el apartado “*cloud credential*” se pueden visualizar, operar con las credenciales que se dispone, o introducir otras credenciales. Hay diferentes proveedores disponibles con los que IM interactúa, desde los principales proveedores Cloud públicos como Amazon Web Services, Microsoft Azure, Google Cloud o con un cluster on-premises implementado sobre OpenNebula.

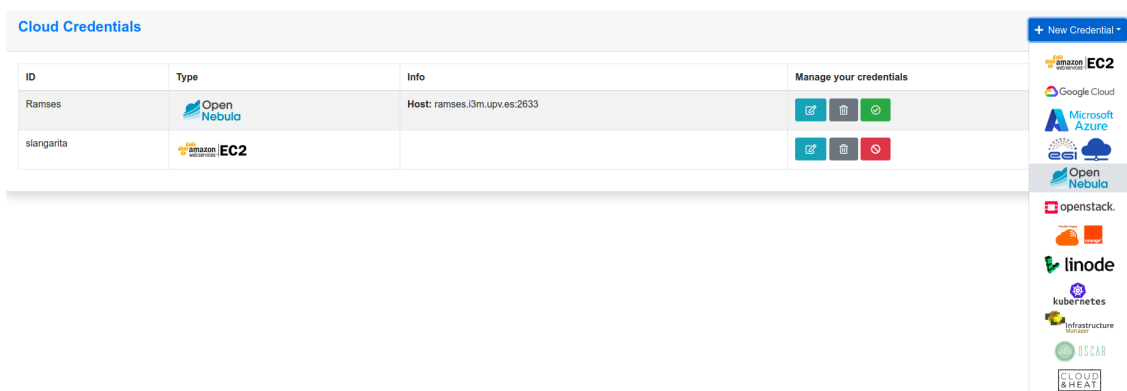
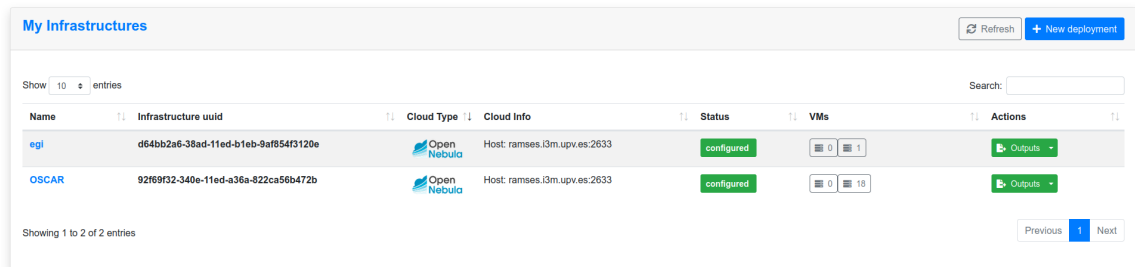


Figura 13: Credenciales almacenadas en IM.

Una vez se disponga de las credenciales de uno de los proveedores, las limitaciones del uso de esa infraestructura vendrán determinadas por las limitaciones impuestas a las credenciales introducidas. En la pantalla principal la opción “Launch an OSCAR Virtual Cluster” despliega un clúster de OSCAR. Esta es una plataforma serverless implementada sobre un cluster de Kubernetes elásticos junto con MinIO como servidor de almacenamiento y al subir un



fichero a unos almacenes de objetos, llamados *buckets*, desencadena la invocación de un servicio, haciendo crecer y decrecer el tamaño los trabajos activos en el cluster de Kubernetes conforme a demanda.



Name	Infrastructure uuid	Cloud Type	Cloud Info	Status	VMs	Actions
egi	d64bb2a6-38ad-11ed-b1eb-9af854f3120e	Open Nebula	Host: ramses.3m.upv.es:2633	configured	0 / 1	Outputs
OSCAR	92f69f32-340e-11ed-a36a-822ca56b472b	Open Nebula	Host: ramses.3m.upv.es:2633	configured	0 / 18	Outputs

**Figura 14: Infraestructuras desplegadas en IM.**

En el apartado Infraestructuras, se visualizan las infraestructuras asignadas al usuario. La información que se expone de una infraestructura es un nombre asignado, una cadena de caracteres como identificador, la *Cloud* en la cual está desplegada y su información, el estado en el que se encuentra la infraestructura, el número de máquinas que compone la infraestructura y las acciones que se aplican al clúster.

En una explicación más completa del funcionamiento de las infraestructuras creadas en IM [3], el nodo *o* es el nodo principal y seleccionándolo se obtiene la información necesaria para conectarse a él (el usuario, el dominio o dirección ip y el fichero con la clave de acceso que se debe descargar) mediante *ssh*, protocolo de acceso remoto. El clúster estará completamente desplegado y disponible cuando su estado sea “*configured*”, teniendo en cuenta que puede tardar un tiempo prolongado. Con respecto a la pestaña “*outputs*”, se encuentran los puntos de accesos a los diferentes recursos del clúster. Cuando se despliega un clúster de OSCAR se accede a la interfaz web de OSCAR usando el enlace junto a *oscarui\_endpoint*.

Outputs	
Field Name	Value
console_minio_endpoint	<a href="https://console.minio.determined-liskov7.im.grycap.net/">https://console.minio.determined-liskov7.im.grycap.net/</a>
dashboard_endpoint	<a href="https://determined-liskov7.im.grycap.net/dashboard/">https://determined-liskov7.im.grycap.net/dashboard/</a>
minio_endpoint	<a href="https://minio.determined-liskov7.im.grycap.net/">https://minio.determined-liskov7.im.grycap.net/</a>
oscarui_endpoint	<a href="https://determined-liskov7.im.grycap.net/">https://determined-liskov7.im.grycap.net/</a>

Close

Figura 15: *Endpoints* de OSCAR que IM proporciona.

## 3.7. Creación y despliegue de los servicios de OSCAR.

Para exponer los modelos en OSCAR se deben crear contenedores en un entorno de ejecución que contengan estos modelos ya entrenados. Un servicio de OSCAR permite ejecutar un contenedor que empaqueta un modelo junto con las dependencias necesarias para su ejecución. Al realizar esta invocación del servicio y ejecución del contenedor obtendrá una predicción respecto a los datos de entrada. Se han creado cuatro servicios de OSCAR para el despliegue de los tres modelos:

- Dos para la predicción de cáncer de mama.
- Uno para la predicción de enfermedades respiratorias.
- Uno para la predicción de Alzheimer.

En la predicción de cáncer de mama se ha usado un modelo de regresión lineal y se han creado dos servicios: el primero realiza una predicción con el modelo, y el segundo devuelve gráficos con la distribución de los datos del *dataset* y los datos de entrada. Los modelos de predicción de enfermedades respiratorias y predicción de Alzheimer clasifican imágenes. La única diferencia entre los contenedores a crear es el modelo introducido, porque todos ellos tienen las mismas dependencias. Para la creación de los servicios de OSCAR es necesario crear un contenedor Docker con el código a ejecutar dentro y un script de ejecución. También se ha creado un archivo YAML con la herramienta FDL Composer para el despliegue automático dentro de nuestras infraestructuras.

El contenedor donde se ejecutará la predicción de cáncer de mama está construido con la imagen Docker de referencia de Python 3.9. Se copian los archivos necesarios dentro del contenedor, un fichero json para normalizar los datos, un fichero pkl que tiene el modelo de regresión lineal, y el programa Python que ejecuta normaliza los datos y realiza una predicción sobre el modelo. Por último se actualiza el contenedor y se instalan las dependencias de panda y sklearn.

```
FROM python:3.9
COPY json_data.json /opt/json_data.json
COPY Pickle_RL_Model.pkl /opt/Pickle_RL_Model.pkl
COPY main.py /opt/main.py
RUN apt-get update
RUN apt-get install -y python3 python3-pip
RUN pip3 install pandas==1.4.3 sklearn==0.0
```

El contenedor que realiza los gráficos toma como referencia una imagen de Ubuntu 20.04. Se copia dentro del contenedor el programa python que realiza los gráficos y los datos para dibujar los gráficos. Por último se actualiza el contenedor y se instalan las dependencias de python, zip, pandas y matplotlib.

```
FROM ubuntu:20.04
COPY graph.py /opt/graph.py
COPY Breast_cancer_data.csv /opt/Breast_cancer_data.csv
RUN apt-get update
RUN apt-get install -y python3 python3-pip zip
RUN pip3 install pandas==1.4.3 matplotlib==3.5.2
```

Los contenedores que realizan una predicción sobre imágenes parten de una imagen Python 3.9, se copia el modelo y programa Python a ejecutar. Después se crean dos carpetas para adaptar el sistema de ficheros al necesario. Y en el último paso se instalan dependencias, TensorFlow y tensorflow\_datasets.

```
FROM python:3.9.0
COPY cnnmodel /opt/cnnmodel
RUN mkdir /opt/image
RUN mkdir /opt/image/class
COPY main.py /opt/main.py
RUN pip install --no-cache-dir tensorflow
tensorflow_datasets
```

Si se invocan de forma asíncrona los servicios seguirán uno de los siguientes flujos de trabajo:

1. Al subir un archivo en el *bucket* input: bcp/input.
  - 1.1. Ejecutará dos servicios en relación con la predicción de cáncer de mama y un script dentro de cada contenedor realizará el siguiente procedimiento.
    - 1.1.1. El programa Python normalizará los datos de entrada, cargará el modelo y predecirá un resultado. Y dejará resultado en el *bucket* output: bcp/output/model.
    - 1.1.2. El programa Python creará los gráficos y los empaquetará en un zip. Dejando el zip en el *bucket* output: bcp/output/grap.
2. Al subir una imagen a los *bucket* chestray/input o alzheimer/input.
  - 2.1. Se ejecutará el programa Python donde cambiará la imagen al formato numpy array, cargará el modelo, realizará la predicción y decodificará el resultado. Depositará el resultado en el *bucket* de output: chestray/output o alzheimer/output.

Para la ayuda del despliegue de forma automática se ha usado la herramienta FDL Composer. FDL Composer es una herramienta para crear ficheros los cuales contienen la descripción de un servicio de OSCAR o varios servicios de OSCAR creando flujos de trabajo (*workflows*). Se ha creado un archivo YAML donde están todas las especificaciones necesarias para la configuración del servicio OSCAR especificando el nombre del servicio, nombre de la imagen del contenedor, el archivo script, recursos asignados, *input bucket* y *output bucket*, etc. También se especifica la configuración de MinIO.

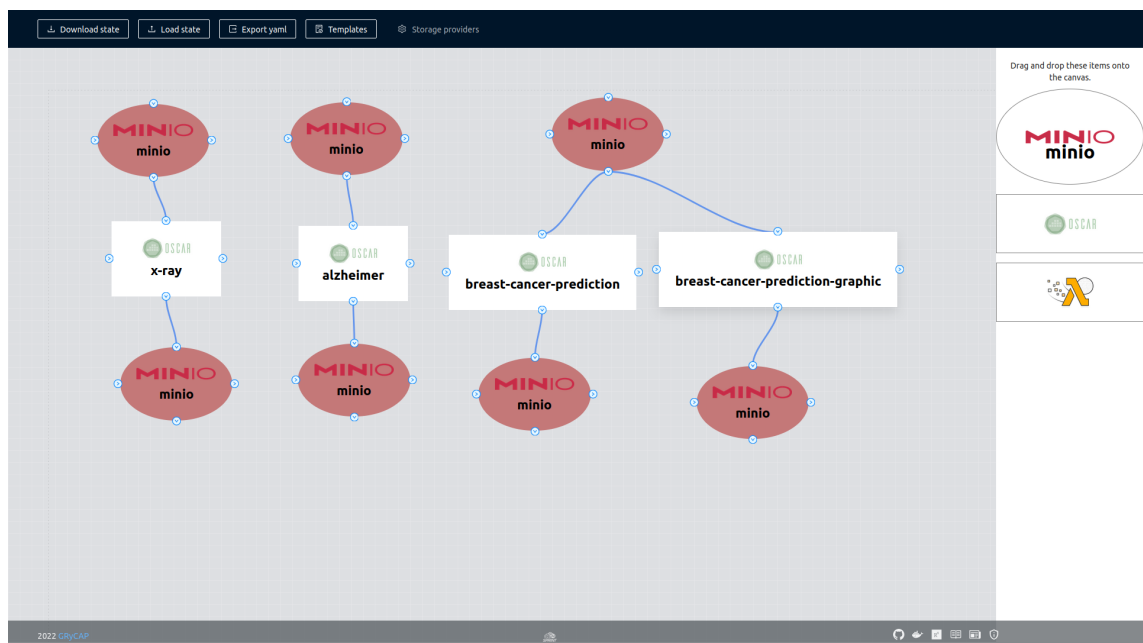
```
functions:
  oscar:
    - oscar-cluster:
      name: x-ray
      memory: 3Gi
      cpu: '1'
      image: ghcr.io/sergiolangaritabenitez/chest_x_ray
      script: script_x_ray.sh
      log_level: CRITICAL
      input:
        - path: chestray/input
          storage_provider: minio.minio
      output:
        - path: chestray/output
          storage_provider: minio.minio
```

```

storage_providers:
minio:
  minio:
  endpoint:
'https://minio.<public-domain>.im.grycap.net'
  region: us-east-1
  access_key:
  secret_key:
  verify: true

```

Se han creado cuatro flujos de trabajo iniciados desde tres buckets distintos como se indica en la figura.



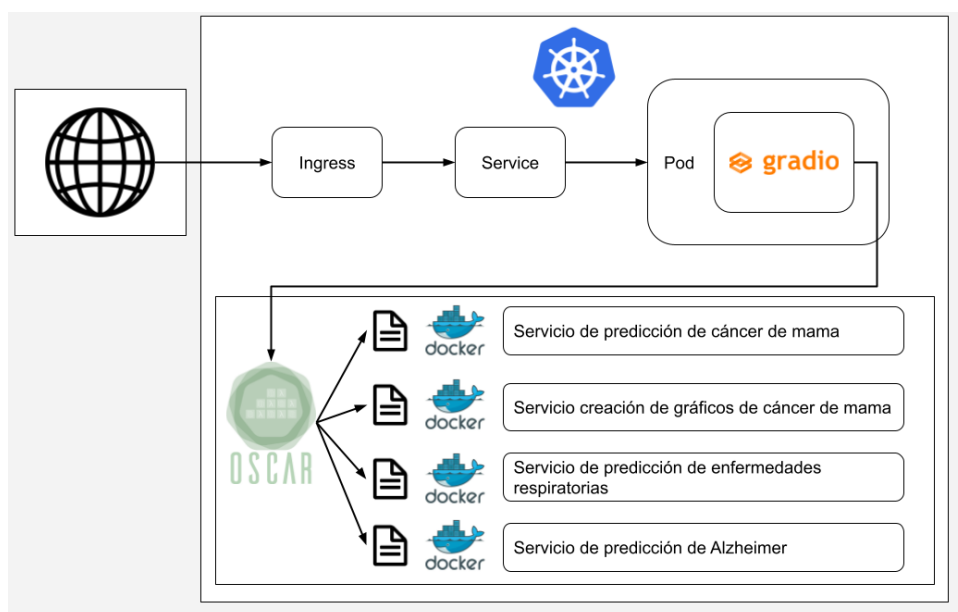
**Figura 16: Flujos de trabajo creados con FDL Composer.**

Los archivos YAML generados por FDL Composer se pueden usar con OSCAR-CLI [50], herramienta de línea de comandos para interactuar con un clúster de OSCAR, para desplegar todos los servicios descritos de forma automática.

## 4. Caso de Uso

La ejecución de un servicio OSCAR se puede hacer de múltiples formas, por ejemplo, con el uso del terminal mediante la herramienta OSCAR-CLI [24] pero el uso del terminal no es algo común excepto para usuarios avanzados. Así que el siguiente objetivo es facilitar al usuario el uso de estos servicios dentro de una infraestructura serverless. Sin la necesidad de conocer la tecnología. Mejorando la usabilidad.

Para ello, la interfaz web de OSCAR nos da la posibilidad de gestionar los servicios, pero la visualización de los resultados de los servicios es algo más complejo. Para simplificar la ejecución se ha creado una interfaz web a modo de demostración con Gradio [12]. Gradio es una librería de Python que tiene como objetivo ayudar a crear una interfaz gráfica de forma sencilla y rápida para facilitar el uso de aplicaciones de inteligencia artificial. Esto es posible gracias a que es fácil interactuar con los componentes y no requiere conocimientos de diseño. Pero la personalización es limitada respecto a tecnologías diseñadas para la creación de interfaces de usuario. En este caso, a modo de demostración es suficiente con un diseño genérico que interactúe con los servicios de OSCAR y exponga los resultados obtenidos como se puede visualizar en la figura 17.



**Figura 17: Diagrama de la arquitectura**

Toda comunicación con el cluster se hace mediante llamadas a la API REST de OSCAR. El usuario se autentica con las credenciales del servidor de OSCAR. Al autenticarse de forma correcta el usuario accede a una interfaz gráfica donde tendrá a su disposición un formulario de cada problema a resolver, predicción de cáncer de mama, predicción de enfermedades respiratorias y predicción de Alzheimer. Se pueden ejecutar estos modelos de manera independiente y

paralela, siempre que el clúster disponga de suficientes recursos. Todas las invocaciones que se realizan son de forma asíncrona. Pero OSCAR también da la posibilidad de ejecutar los servicios de forma síncrona con una llamada a su API REST. Las funciones asíncronas se deben invocar con cuatro argumentos.

- Los datos de entrada.
- El *bucket* de entrada donde se subirá el archivo.
- El *bucket* de salida donde el programa se quedará esperando.
- El nombre que se le pondrá al fichero cuando se descargue, todo esto es transparente al usuario.

Las invocaciones asíncronas solo necesitan acceso a MinIO:

- *Endpoint*: Dirección en la que se encuentra el sistema de almacenamiento de MinIO, para ello se debe añadir subdominio “minio” al dominio de OSCAR.
- *Access key*: es la identificación del usuario.
- *Secret key*: es la contraseña del usuario, que complementa al *access key*.

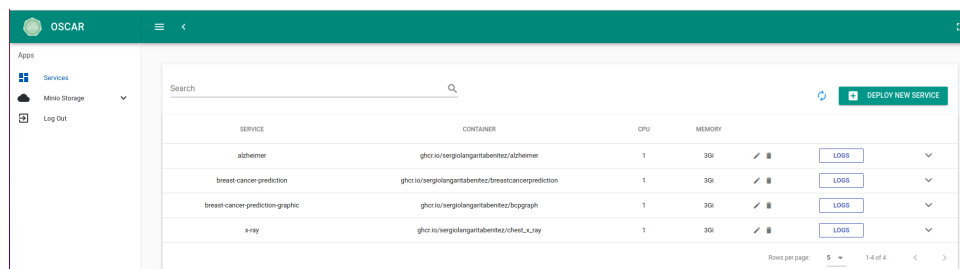
Se dejará en el bucket de entrada el archivo a procesar. Y el proceso se quedará bloqueado escuchando las notificaciones del *bucket* de salida; después se descargará el resultado. En caso de fallo en el servicio de OSCAR y no finalizar el servicio, la interfaz de Gradio no notificará ningún fallo y seguirá esperando la respuesta. Este posible fallo es debido a que se ha usado una implementación de llamadas asíncronas interactuando con el bucket de MinIO. Estas interacciones con MinIO se han realizado de forma más ágil gracias a la API de MinIO. Para finalizar se ha introducido el programa de Gradio en un contenedor y facilitar el despliegue dentro de un cluster de Kubernetes.

## 4.1. Despliegue de Gradio en Kubernetes

Una vez la aplicación de Gradio se ejecuta correctamente en local, se ha encapsulado esa aplicación en una imagen de contenedor Docker para realizar un despliegue sobre el clúster de Kubernetes en el que se encuentra OSCAR.

Para el despliegue en Kubernetes se ha creado un *Deployment* el cual especifica: La imagen del contenedor del programa de Gradio. Con dos variables de entorno. El *endpoint* donde se encuentra OSCAR y el puerto en el que Gradio se despliega. Se ha creado un *Service* que apunta a los *Pods*, instancias de los contenedores, creados por el *Deployment*. Y se ha creado un *Ingress* para

acceder a Gradio, donde se especifica subdominio en el que se encuentra se expone la interfaz gráfica, y se ha modificado el *timeout* en *nginx* de tiempo de espera para la lectura para asegurarse que OSCAR devuelve la respuesta.



The screenshot shows the OSCAR dashboard with a sidebar on the left containing 'Apps', 'Services', 'Mini Storage', and 'Log Out'. The main area has a search bar and a 'DEPLOY NEW SERVICE' button. Below is a table with the following data:

SERVICE	CONTAINER	CPU	MEMORY		
alzheimerr	ghcr.io/sergiolagartabentez/alzheimer	1	3Gi	✓	LOGS
breast-cancer-prediction	ghcr.io/sergiolagartabentez/breastcancerprediction	1	3Gi	✓	LOGS
breast-cancer-prediction-graphic	ghcr.io/sergiolagartabentez/bcpgaph	1	3Gi	✓	LOGS
x-ray	ghcr.io/sergiolagartabentez/chest_x-ray	1	3Gi	✓	LOGS

At the bottom right of the table, it says 'Rows per page: 5' and '1-4 of 4'.

**Figura 18: Cluster de OSCAR con los 4 servicios desplegados**

En la figura 19 se puede visualizar la demostración construida en Gradio con los resultados obtenidos. Se pueden observar los parámetros de entrada como un conjunto de campos los cuales se deben de rellenar o como una imagen a subir. También se han añadido imágenes de muestra, para facilitar la demostración al usuario. Es posible que el usuario no disponga de una radiografía del tórax o una resonancia magnética del cerebro. Los resultados finales imprimen la predicción dada por el modelo y como información complementaria, las posibilidades con las que el modelo clasifica el resultado o gráficos con la representación de los datos junto a los datos de entrada. Todo el proceso de extracción de información de los componentes de entrada, ejecución de los servicios y exposición de los resultados en los componentes de salida, se realiza después de presionar los botones. La información complementaria se ha añadido como ayuda para la toma de decisiones.

El tiempo total de ejecución que se le presenta al usuario es la suma de la latencia hasta el servidor, creación del contenedor, tiempo de ejecución y respuesta. La influencia de estos tiempos dependen de las especificaciones del clúster de OSCAR y los recursos asignados a servicios, que en este caso el clúster se compone de un nodo principal con 4 CPUs y 3,7 GB RAM y cuatro nodos trabajadores con 2 CPUs y 3,7 GB RAM y los recursos asignados a cada servicio es de 1 CPU y 3 GiB. Los tiempos de ejecución se han obtenido respecto la interfaz gráfica de OSCAR, así que no refleja la latencia hasta el servidor. La creación del contenedor es de 2 a 4 segundos siempre y cuando no haya necesidad de pedir el contenedor y sin tener trabajos por delante. Cada servicio de OSCAR tiene un tiempo de ejecución distinto:

- El servicio de OSCAR con el modelo de predicción de cáncer de mama realiza la predicción en un tiempo de ejecución de 5 segundos.
- El servicio de OSCAR que crea los gráficos tiene un tiempo de ejecución de 4 o 5 segundos.
- El servicio de OSCAR para la predicción de enfermedades respiratorias tiene un tiempo de ejecución de 23 a 45 segundos.



- El servicio de OSCAR con el modelo para la predicción de Alzheimer tiene un tiempo de ejecución entre 42 y 68 segundos.

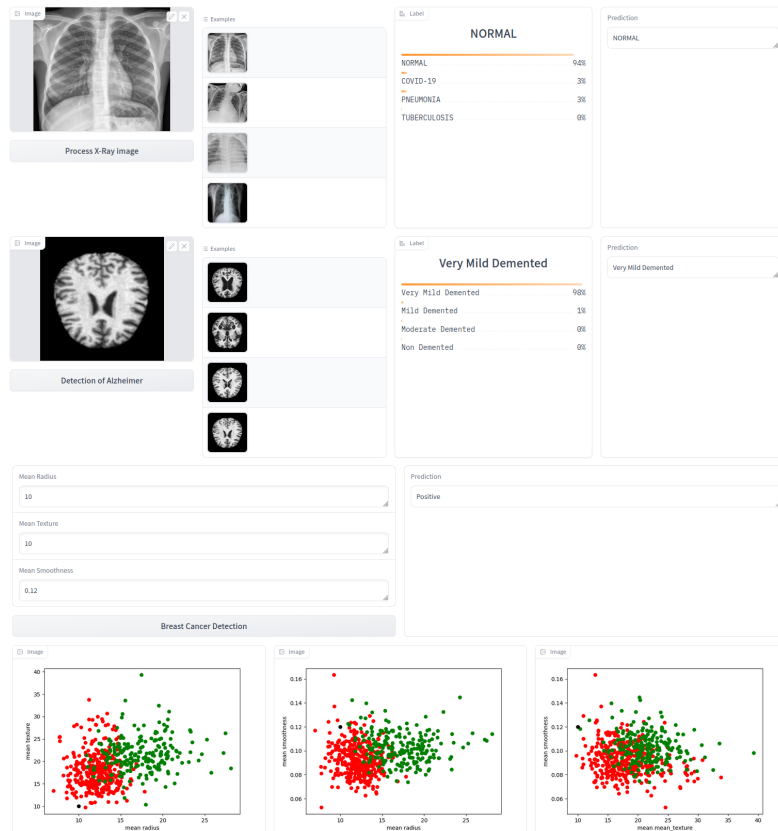


Figura 19: Interfaz gráfica construida con Gradio

## 5. Conclusiones y Trabajo futuro

El trabajo realizado ha sido la extracción de datos médicos de un repositorio público como Kaggle. A partir de los datos obtenidos se han creado diferentes modelos para la predicción de enfermedades, usando Google Colab con las librerías de Python como Scikit-learn, TensorFlow, Numpy y Pandas. Estos modelos predictivos se han encapsulado y adaptado a una plataforma serverless como es OSCAR. Por último se ha buscado facilitar el uso de estos modelos, mediante una exposición de los mismos al usuario con una interfaz gráfica con Gradio.

Los siguientes pasos a llevar a cabo en proyectos futuros es la verificación de los modelos con una puesta en producción de los mismos. En caso de un resultado positivo se podría ampliar el número de servicios con la creación de más modelos o la adaptación de otros modelos ya creados como por ejemplo los realizados por *Deep Health* [20]. Una vez verificado que el modelo es robusto para ponerlo en producción. Se debería sustituir Gradio por una interfaz de usuario más profesional y personalizada que dependa del rol, especialización y con permisos.

Además, este trabajo puede servir de guía para aplicar esta metodología en el Atlas de Imágenes en Cáncer del proyecto EUCAIM [21], asignado al Instituto de Investigación Sanitaria La Fe por la Comisión Europea, dentro del programa Horizonte Europa, donde el grupo de investigación GRyCAP participa.

Este trabajo de fin de Máster se ha elaborado bajo el proyecto OSCARISER (Open Serverless Computing for the Adoption of Rapid Innovation on Secure Enterprise-ready Resources). Proyecto PDC2021-120844-I00 financiado por MCIN/AEI/10.13039/501100011033 y por la Unión Europea Next GenerationEU/PRTR.

## 6. ODS (OBJETIVOS DE DESARROLLO SOSTENIBLE)

---

Además, se busca cumplir con algunos Objetivos de Desarrollo Sostenible (ODS) impulsados por la Organización de Naciones Unidas (ONU) [22]:

- **Objetivo 3:** Garantizar una vida sana y promover el bienestar para todos en todas las edades. Como los modelos para la predicción temprana de enfermedades son libres y cualquier persona puede utilizarlos, su uso promueve la anticipación y prevención ante una enfermedad para mejorar el bienestar de los ciudadanos.
- **Objetivo 4:** Una educación de calidad con acceso igualitario. A una formación técnica, profesional y superior de calidad, incluida la enseñanza universitaria. El contenido del trabajo es libre para su uso y la enseñanza, facilitando el acceso a quién esté interesado.
- **Objetivo 9:** Industria, innovación e infraestructura inclusivas y sostenibles, donde promuevan las nuevas tecnologías, faciliten el comercio internacional y permitan el uso eficiente de los recursos. Con el uso de plataformas Serverless se utilizan solo recursos necesarios reduciendo los gastos.




## 7. Referencias Bibliográficas

---

- [1] Serverless computing - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online], (“Serverless computing”) Available from: [https://en.wikipedia.org/wiki/Serverless\\_computing](https://en.wikipedia.org/wiki/Serverless_computing)
- [2] Pérez, A., Risco, S., Naranjo, D. M., Caballer, M., & Moltó, G. (2019, July). On-premises serverless computing for event-driven data processing applications. In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD) (pp. 414-421). IEEE. 2022. OSCAR [online], Available from: <https://ieeexplore.ieee.org/document/8814513/>
- [3] Caballer, M., Blanquer, I., Moltó, G., & de Alfonso, C. (2015). Dynamic management of virtual infrastructures. Journal of Grid Computing, 13(1), 53-70. GRYCAP, INSTITUTE, 2022, IM - Infrastructure Manager a TOSCA Cloud Orchestrator. Grycap.upv.es [online]. 2022. [Accessed 13 October 2022]. Available from: <https://www.grycap.upv.es/im/index.php>
- [4] GitHub - grycap/fdl-composer, 2022. GitHub [online], Available from: <https://github.com/grycap/fdl-composer>
- [5] GRyCAP – Grupo de Grid y Computación de Altas Prestaciones, 2022. Grycap.upv.es [online], Available from: <https://www.grycap.upv.es/>
- [6] Docker (software) - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online], Available from: [https://es.wikipedia.org/wiki/Docker\\_\(software\)](https://es.wikipedia.org/wiki/Docker_(software))
- [7] Kubernetes - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online], Available from: <https://es.wikipedia.org/wiki/Kubernetes>
- [8] Scikit-learn - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online], Available from: <https://es.wikipedia.org/wiki/Scikit-learn>
- [9] TensorFlow - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online], Available from: <https://es.wikipedia.org/wiki/TensorFlow>
- [10] NumPy - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online], Available from:
- [11] Pandas (software) - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online], Available from: [https://es.wikipedia.org/wiki/Pandas\\_\(software\)](https://es.wikipedia.org/wiki/Pandas_(software))
- [12] GitHub - gradio-app/gradio: Create UIs for your machine learning model in Python in 3 minutes, 2022. GitHub [online], Available from: <https://github.com/gradio-app/gradio>

- [13] Breast Cancer Prediction Dataset, 2022. Kaggle.com [online], Available from: <https://www.kaggle.com/datasets/merishnasuwal/breast-cancer-prediction-dataset>
- [14] Chest X-Ray (Pneumonia, Covid-19, Tuberculosis), 2022. Kaggle.com [online], Available from: <https://www.kaggle.com/datasets/jtiptj/chest-xray-pneumoniacovid19tuberculosis?select=train>
- [15] Alzheimer MRI Preprocessed Dataset, 2022. Kaggle.com [online], Available from: <https://www.kaggle.com/datasets/sachinkumar413/alzheimer-mri-dataset>
- [16] Google Colab, 2022. Research.google.com [online], Available from: <https://research.google.com/colaboratory/intl/es/faq.html>
- [17] split-folders, 2022. PyPI [online], Available from: <https://pypi.org/project/split-folders/>
- [18] Formato de compresión ZIP - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online], Available from: [https://es.wikipedia.org/wiki/Formato\\_de\\_compresi%C3%B3n\\_ZIP](https://es.wikipedia.org/wiki/Formato_de_compresi%C3%B3n_ZIP)
- [19] Opennebula - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online], Available from: <https://es.wikipedia.org/wiki/Opennebula>
- [20] Deep Health – Deep-Learning and HPC to Boost Biomedical Applications for Health, 2022. Deephealth-project.eu [online], Available from: <https://deephealth-project.eu/>
- [21] FE, IIS, 2022, El IIS La Fe liderará la dirección científica del Nodo Central del Atlas de Imágenes en Cáncer dentro del proyecto europeo EUCAIM | Noticias. IIS La Fe [online]. 2022. [Accessed 13 October 2022]. Available from: <https://www.iislafe.es/es/actualidad/noticias/3301/el-iis-la-fe-liderara-la-direccion-cientifica-del-nodo-central-del-atlas-de-imagenes-en-cancer-dentro-del-proyecto-europeo-eucaim>
- [22] United Nations - Objetivos de desarrollo sostenible, 2022. <https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible>
- [23] Infrastructure manager dashboard. Home [online]. [Accessed 20 October 2022]. Available from: <https://im.egi.eu/>
- [24] CLI. OSCAR [online]. [Accessed 20 October 2022]. Available from: <https://docs.oscar.grycap.net/oscar-cli/>

- [25] The NIST definition of cloud computing. [online]. [Accessed 17 November 2022]. Available from: <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf>
- [26] EMAZ, Antoine and PÉREZ-RUIZ Una. ES. *Amazon* [online]. 1998. [Accessed 17 November 2022]. Available from: <https://aws.amazon.com/es/lambda/>
- [27] Azure functions overview. *Azure Functions Overview | Microsoft Learn* [online]. [Accessed 17 November 2022]. Available from: <https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview>
- [28] Cloud functions | google cloud. *Google* [online]. [Accessed 17 November 2022]. Available from: <https://cloud.google.com/functions>
- [29] MINIO, Inc. High performance, kubernetes native object storage. *MinIO* [online]. [Accessed 17 November 2022]. Available from: <https://min.io/>
- [30] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. 2022. Serverless Computing: A Survey of Opportunities, Challenges, and Applications. *ACM Comput. Surv.* 54, 11s, Article 239 (January 2022), 32 pages. <https://doi.org/10.1145/3510611>
- [31] Creación de Funciones de Azure en Linux mediante una imagen personalizada. *Creación de funciones de Azure en Linux mediante una imagen personalizada | Microsoft Learn* [online]. [Accessed 17 November 2022]. Available from: <https://learn.microsoft.com/es-es/azure/azure-functions/functions-create-function-linux-custom-image?tabs=in-process%2Cbash%2Cazure-cli&pivots=programming-language-csharp>
- [32] AHMED, Omer Farooq. Docker container on azure functions with python. *DEV Community*  [online]. 13 October 2020. [Accessed 17 November 2022]. Available from: <https://dev.to/omer95/docker-container-on-azure-functions-with-python-1lgd>
- [33] HENDRIX, Roger W. Lambda. *Amazon* [online]. 1983. [Accessed 17 November 2022]. Available from: <https://docs.aws.amazon.com/lambda/latest/dg/images-create.html>
- [34] What is cloud run | cloud run documentation | google cloud. *Google* [online]. [Accessed 17 November 2022]. Available from: <https://cloud.google.com/run/docs/overview/what-is-cloud-run>
- [35] HENDRIX, Roger W. Lambda. *Amazon* [online]. 1983. [Accessed 17 November 2022]. Available from:

<https://docs.aws.amazon.com/lambda/latest/operatorguide/lambda-event-driven-paradigm.html>

[36] CREA UN activador para cloud run | Eventarc | google cloud. *Google* [online]. [Accessed 17 November 2022]. Available from: <https://cloud.google.com/eventarc/docs/creating-triggers?hl=es-419>

[37] Desencadenadores y enlaces en azure functions. *Desencadenadores y enlaces en Azure Functions* | *Microsoft Learn* [online]. [Accessed 17 November 2022]. Available from: <https://learn.microsoft.com/es-es/azure/azure-functions/functions-triggers-bindings?tabs=csharp#supported-bindings>

[38] HENDRIX, Roger W. Lambda. *Amazon* [online]. 1983. [Accessed 17 November 2022]. Available from: <https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html>

[39] Cloud run quotas and limits | cloud run documentation | google cloud. *Google* [online]. [Accessed 17 November 2022]. Available from: <https://cloud.google.com/run/quotas>

[40] Azure functions scale and hosting. *Azure Functions scale and hosting* | *Microsoft Learn* [online]. [Accessed 17 November 2022]. Available from: <https://learn.microsoft.com/en-us/azure/azure-functions/functions-scale#service-limits>

[41] Azure functions pricing. *Microsoft Learn* [online]. [Accessed 17 November 2022]. Available from: <https://learn.microsoft.com/en-us/azure/azure-functions/pricing>

[42] HENDRIX, Roger W. Lambda. *Amazon* [online]. 1983. [Accessed 17 November 2022]. Available from: <https://aws.amazon.com/lambda/pricing/>

[43] Precios - functions: Microsoft Azure. *Precios - Functions* | *Microsoft Azure* [online]. [Accessed 17 November 2022]. Available from: <https://azure.microsoft.com/es-es/pricing/details/functions/>

[44] Precios | cloud run | google cloud. *Google* [online]. [Accessed 17 November 2022]. Available from: <https://cloud.google.com/run/pricing?hl=es>

[45] *AWS Pricing Calculator* [online]. [Accessed 17 November 2022]. Available from: <https://calculator.aws>

[46] SEAN MCKENNA PRINCIPAL PROGRAM MANAGER. Bringing serverless to Azure Kubernetes Service. *Azure Blog and Updates* | *Microsoft Azure* [online]. [Accessed 17 November 2022]. Available from:



<https://azure.microsoft.com/en-us/blog/bringing-serverless-to-azure-kubernetes-service/>

[47] *AWS Pricing Calculator* [online]. [Accessed 17 November 2022]. Available from: [https://calculator.aws/#/addService/Lambda?refid=ft\\_card](https://calculator.aws/#/addService/Lambda?refid=ft_card)

[48] Calculadora de Precios. *Microsoft Azure* [online]. [Accessed 17 November 2022]. Available from: <https://azure.microsoft.com/es-mx/pricing/calculator/>

[49] Cloud pricing calculator. *Cloud Pricing Calculator* [online]. [Accessed 17 November 2022]. Available from: <https://cloudpricingcalculator.appspot.com/>

[50] CLI. *OSCAR* [online]. [Accessed 18 November 2022]. Available from: <https://docs.oscar.grycap.net/oscar-cli/>



## 8. Anexos

### 8.1.-Red neuronal de predicción de enfermedades respiratorias.

```
def conv_block_1(x, filters, ks = (3, 3)):
    x = keras.layers.Conv2D(filters=filters,
                             kernel_size=ks,
                             strides=(1, 1),
                             padding='same',
                             activation=None)(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Activation('relu')(x)
    return x

def cnn_1(input_shape = None, num_labels = None):
    if input_shape is None:
        raise Exception('input_shape must be provided
as a tuple, e.g., (28, 28)')
    if num_labels is None:
        raise Exception('num_labels must be provided as
an integer')
    inputs = keras.Input(shape = input_shape)
    x = inputs
    x = conv_block_1(x, filters=16, ks=(3, 3))
    x = conv_block_1(x, filters=64, ks=(3, 3))
    x = keras.layers.MaxPool2D(pool_size=(2, 2),
padding='same')(x)
    x = conv_block_1(x, filters=128, ks=(3, 3))
    x = conv_block_1(x, filters=256, ks=(3, 3))
    x = keras.layers.MaxPool2D(pool_size=(2, 2),
padding='same')(x)
    x = conv_block_1(x, filters=256, ks=(3, 3))
    x = conv_block_1(x, filters=256, ks=(3, 3))
    x = keras.layers.MaxPool2D(pool_size=(2, 2),
padding='same')(x)
    x = conv_block_1(x, filters=512, ks=(3, 3))
    x = conv_block_1(x, filters=512, ks=(3, 3))
```



```
x = keras.layers.MaxPool2D(pool_size=(2, 2),
padding='same')(x)
x = conv_block_1(x, filters=1024, ks=(3, 3))
x = conv_block_1(x, filters=1024, ks=(3, 3))

x = keras.layers.Flatten()(x)

x = keras.layers.Dense(516)(x)
x = keras.layers.BatchNormalization()(x)
x = keras.layers.Activation('relu')(x)

outputs = keras.layers.Dense(num_labels,
activation='softmax')(x)
model = keras.Model(inputs, outputs)
model.compile(

optimizer=keras.optimizers.Adam(learning_rate=1.0e-5
),
    loss='sparse_categorical_crossentropy',

metrics=[tf.keras.metrics.SparseCategoricalCrossentr
opy(),tf.keras.metrics.SparseCategoricalAccuracy()],
)
model.summary()
return model
```

## 8.2.-Red neuronal de predicción de Alzheimer.

```
def conv_block_1(x, filters, ks = (3, 3)):
x = keras.layers.Conv2D(filters=filters,
                        kernel_size=ks,
                        strides=(1, 1),
                        padding='same',
                        activation=None)(x)
x = keras.layers.BatchNormalization()(x)
x = keras.layers.Activation('relu')(x)
return x
```

```

def cnn_1(input_shape = None, num_labels = None):
    if input_shape is None:
        raise Exception('input_shape must be provided
as a tuple, e.g., (28, 28)')
    if num_labels is None:
        raise Exception('num_labels must be provided as
an integer')
    inputs = keras.Input(shape = input_shape)
    x = inputs
    x = conv_block_1(x, filters=16, ks=(3, 3))
    x = conv_block_1(x, filters=64, ks=(3, 3))
    x = keras.layers.MaxPool2D(pool_size=(2, 2),
padding='same')(x)
    x = conv_block_1(x, filters=128, ks=(3, 3))
    x = conv_block_1(x, filters=256, ks=(3, 3))
    x = keras.layers.MaxPool2D(pool_size=(2, 2),
padding='same')(x)
    x = conv_block_1(x, filters=256, ks=(3, 3))
    x = conv_block_1(x, filters=256, ks=(3, 3))
    x = keras.layers.MaxPool2D(pool_size=(2, 2),
padding='same')(x)
    x = conv_block_1(x, filters=512, ks=(3, 3))
    x = conv_block_1(x, filters=512, ks=(3, 3))

    x = keras.layers.Flatten()(x)

    x = keras.layers.Dense(516)(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Activation('relu')(x)

    outputs = keras.layers.Dense(num_labels,
activation='softmax')(x)
    model = keras.Model(inputs, outputs)
    model.compile(

optimizer=keras.optimizers.Adam(learning_rate=1.0e-5
),
    loss='sparse_categorical_crossentropy',

```



```
metrics=[tf.keras.metrics.SparseCategoricalCrossentropy(),tf.keras.metrics.SparseCategoricalAccuracy()],  
)  
model.summary()  
return model
```