eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Máster Universitario en Ingeniería Computacional y Sistemas Inteligentes

Departamento de Ciencias de la Computación e Inteligencia Artificial

Tesis de Máster

Análisis del impacto del aprendizaje federado en entornos desbalanceados

Autora

Mirai Herrera Piñeiro

Septiembre de 2021

Director
Basilio Sierra (EHU)

Tutor en empresa
Daniel Estepa (Ikerlan)

Konputazio Ingeniaritza eta Sistema Adimentsuak Unibertsitate Masterra

Máster Universitario en Ingeniería Computacional y Sistemas Inteligentes

ikerlan
MEMBER OF BASQUE RESEARCH & TECHNOLOGY ALLIANCE

Resumen

Con este proyecto se ha querido estudiar sobre el aprendizaje federado (*federated learning*), un campo bastante reciente dentro del aprendizaje profundo (*deep learning*). Esta aproximación puede ser una apuesta muy interesante para muchas empresas, ya que el aprendizaje federado permite hacer uso de datos sin la necesidad de tener que compartirlos, de modo que los datos se mantienen en privado.

Más específicamente, el objetivo de este proyecto ha sido hacer un análisis sobre el impacto que este tipo de aprendizaje tiene sobre entornos desbalanceados, haciendo uso de diferentes distribuciones de datos y diferentes algoritmos de agregación.

Es bien sabido que un aprendizaje federado no podrá alcanzar un nivel de precisión como el de uno centralizado. Es por ello que se ha querido analizar en que situaciones se le podría sacar el mayor provecho, teniendo en cuenta si la perdida de precisión podría considerarse aceptable o no.

Palabras clave: aprendizaje federado, aprendizaje profundo, privacidad, redes neuronales, desbalance de datos, agregadores.

Abstract

This project studies federated learning, a fairly recent field within deep learning. Since federated learning allows the use of data without the need to share it, this approach can be a very interesting bet for many companies, so their data is kept private.

More specifically, this project analyzes the impact this type of learning has on unbalanced environments, making use of different data distributions and different aggregation algorithms.

It is well known that a federated-learned model will not be able to achieve a level of precision a centralized-learned one can achieve. This gives us a reason to analyze in which situations we could get the most out of it, taking into account if the loss of precision could be considered acceptable or not.

Keywords: federated learning, deep learning, privacy, neural networks, data imbalance, aggregators.

Laburpena

Proiektu honen bidez ikasketa federatua (*federated learning*) landu nahi izan da, ikasketa sakonaren (*deep learning*) barruan azken urteotan agertu den esparru berri bat hain zuzen ere. Ikasketa federatuak datuak konpartitu beharrik gabe datuak erabiltzea ahalbidetzen duenez, enpresa askorentzat apostu oso interesgarria izan daiteke, modu honetan datuak pribatutasunean mantenduko dituztelako.

Zehatzagoak izateko, proiektu honen helburua ikasketa mota honek ingurune desorekatuetan daukan eraginaren analisi bat egitea izan da, datu-banaketa desberdinak eta bateratze-algoritmo desberdinak erabiliz.

Ikasketa federatu batek ezingo du ikasketa zentralizatu batek lortzen duen doitasun-maila gainditu, hori bistakoa da. Arrazoibide honek ordea, ikasketa honi etekin handiena zein egoeretan atera dakiokeen ikertu nahi izatera bultzatu gaitu, doitasun-galera onargarria izan daitekeen ala ez kontuan hartuz.

Hitz gakoak: ikasketa federatua, ikasketa sakona, pribatutasuna, neurona-sareak, datuen desoreka, bateratzaileak.

Índice general

Índice de figuras

1. CAPÍTULO

Introducción

Este documento es la memoria de mi trabajo de fin de máster. El trabajo lo he realizado en el centro tecnológico vasco Ikerlan [1], una cooperativa que es parte de la corporación Mondragon [2] (Arrasate). Ikerlan fue fundada en 1974, y desde entonces trabaja en desarrollar soluciones para que sus clientes sean cada vez más competitivos, adaptándose a las necesidades que tengan.

Figura 1.1: Olandixo, la sede principal del centro tecnológico Ikerlan en Arrasate/Mondragon.

En la Figura 1.1 se ve la sede de Ikerlan en donde he estado trabajando durante dos meses y medio a tiempo completo realizando este proyecto. Dentro de la empresa me han facilitado un ordenador y un puesto de trabajo con todo el equipo necesario, como también un entorno muy amigable.

1.1. Contexto

En los recientes años ha surgido un nuevo campo dentro del *deep learning*. Este campo se llama *federated learning* (aprendizaje federado), y propone un concepto muy interesante para ciertas situaciones.

A veces una empresa o una institución no dispone de suficientes datos para hacer un entrenamiento robusto, o bien sus datos son demasiado heterogéneos (es decir, la diferencia entre las cantidades de muestras de ciertas clases es muy grande). Esto se arreglaría juntando más datos de otras empresas e instituciones, pero por diversos motivos la mayoría no pueden permitirse compartir sus datos.

El aprendizaje federado propone una solución a este dilema. Cada institución entrena un modelo sobre sus propios datos y lo envía a un servidor que se encarga de fusionarlos todos en un único modelo. El servidor devuelve a cada institución el modelo fusionado, y estos vuelven a entrenarlo sobre sus propios datos. Mediante la sucesiva repetición de estos pasos, se logra entrenar un modelo global entre todas las instituciones sin que ninguna tenga que poner sus datos a la vista de los demás.

1.2. Objetivos y enfoque

Este proyecto cuenta con tres objetivos principales:

- Investigar sobre el aprendizaje federado y su estado del arte.
- Analizar el impacto del aprendizaje federado en entornos desbalanceados de datos.
- Comparar diferentes métodos para hacer la agregación de modelos entrenados.

Para cumplir con el primer objetivo, primero he realizado un estudio sobre el *deep learning* y después he empezado a explorar profundamente el concepto del *federated learning*. Dentro del aprendizaje federado he indagado en diferentes *frameworks* y diferentes tipologías de agregación.

Para abordar el segundo objetivo, he simulado diferentes niveles de desbalanceo de datos mediante diferentes distribuciones de datos, he entrenado varios modelos de aprendizaje sobre estas distribuciones, y he sacado conclusiones midiendo el rendimiento de los modelos entrenados.

Como último objetivo, he vuelto a entrenar los mismos modelos del segundo objetivo con diferentes algoritmos de agregación, entre los que también se encuentra uno creado por mi misma. Además, también he participado en un desafío internacional sobre el aprendizaje federado, con el objetivo de profundizar más en el tema y sacar más conclusiones.

1.3. Planificación

Para poder llevar a cabo todos los objetivos propuestos he planificado las diferentes fases del proyecto. Para ello he definido unas cuantas tareas y he indicando desde que semana hasta que semana se debería realizar cada una de ellas, mediante el cronograma que se ve en la Figura 1.2.

Esta es la lista de las diferentes tareas que he definido:

- Bloque 1 ⟶ Estudio.
  - Tarea 1 ⟶ Repasar el tema del *deep learning* mediante un curso de Coursera.
  - Tarea 2 ⟶ Buscar información y estudiar el *federated learning*.
  - Tarea 3 ⟶ Investigar el estado del arte del *federated learning*.
  - Tarea 4 ⟶ Informarse sobre OpenFL y entender todo su código fuente.
- Bloque 2 ⟶ Caso de uso K-MNIST.
  - Tarea 1 ⟶ Adaptar el código fuente para crear diferentes distribuciones de datos.
  - Tarea 2 ⟶ Entrenar un modelo con cada diferente distribución de datos.
  - Tarea 3 ⟶ Entrenar un modelo con cada método de agregación.
  - Tarea 4 ⟶ Visualizar y analizar los resultados.
- Bloque 3 ⟶ Caso de uso *FeTS Challenge*.
  - Tarea 1 ⟶ Informarse sobre el *FeTS Challenge* y entender todo su código fuente.
  - Tarea 2 ⟶ Crear un nuevo método de agregación.
  - Tarea 3 ⟶ Entrenar un modelo con cada método de agregación.
  - Tarea 4 ⟶ Visualizar y analizar los resultados.
- Bloque 4 ⟶ Documentación.
  - Tarea 1 ⟶ Documentar las tareas diariamente.
  - Tarea 2 ⟶ Escribir la memoria.
  - Tarea 3 ⟶ Revisar la memoria.
  - Tarea 4 ⟶ Presentar el proyecto ante tribunal.

El tiempo que he estado trabajando en la empresa ha sido 12 semanas, desde mediados de mayo hasta finales de julio. He planificado todas las tareas para realizarlas durante ese tiempo, teniendo en cuenta que seguramente la memoria la tendría que escribir durante agosto, y por lo tanto dejando todo agosto como margen para terminarlo.

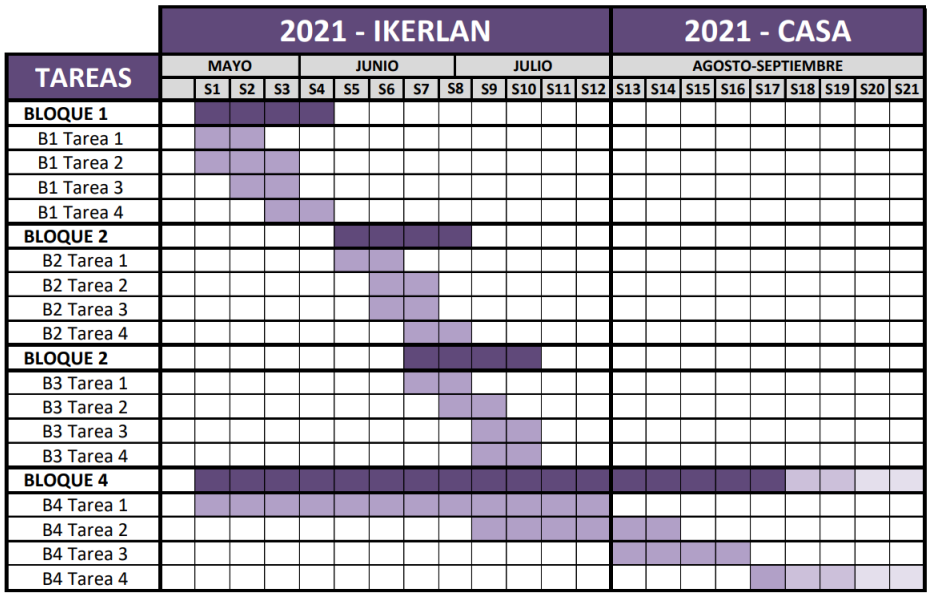

Figura 1.2: El cronograma de la planificación del proyecto.

Para realizar la defensa ante el tribunal, el último día que la universidad permite es el 30 de septiembre. Sin embargo, también se ha planificado hacer una presentación del trabajo en la propia empresa durante septiembre, antes de hacer la defensa.

1.4. Resumen de lo obtenido

Mediante este proyecto, por un lado se ha obtenido un análisis detallado sobre el aprendizaje federado, con todas sus ventajas y desventajas principales. Por otro lado, el proyecto también ofrece una lista de diferentes *frameworks* y otra de diferentes tipologías de agregación, sacadas del estado del arte.

Aparte de la información teórica, también se han expuesto dos casos de su uso como ejemplos prácticos, explicando detalladamente como se han llevado a cabo, analizando los resultados y sacando conclusiones.

1.5. Sumario de los capítulos

La memoria se divide en seis apartados principales:

1. **Introducción:** explica el motivo del proyecto, sus objetivos, y la metodología aplicada en el trabajo.
2. **Estudio del tema:** explora el tema del *deep learning* y del *federated learning* profundamente, dando a conocer diferentes *framework*s y tipologías de agregación de modelos entrenados.
3. **Herramientas de trabajo:** presenta las herramientas que se han utilizado para realizar los casos de uso.
4. **Caso de uso I - Análisis del impacto del aprendizaje federado:** expone un caso de uso del aprendizaje federado poniendo la teoría en practica, entrenando varios modelos y analizando los resultados obtenidos.
5. **Caso de uso II - Detección federada de tumores cerebrales:** expone otro caso de uso más participando en una competición internacional del aprendizaje federado y analizando los resultados obtenidos.
6. **Conclusiones finales:** explica las conclusiones finales que se han sacado realizando este proyecto.

2. CAPÍTULO

Estudio del tema

Para poder entender qué es el *federated learning* hace falta comprender bien el *deep learning*. Para ello, antes de abordar el aprendizaje federado, se ha realizado un estudio-repaso sobre el aprendizaje profundo, para así recordar lo estudiado durante el grado y el máster.

Una vez repasado el *deep learning*, se ha procedido a estudiar y entender profundamente el *federated learning*, buscando información en artículos y leyendo *paper*s sobre el tema.

2.1. Deep Learning

El aprendizaje profundo (*deep learning*) es un conjunto de algoritmos de aprendizaje automático (*machine learning*) que intenta modelar abstracciones de alto nivel en datos usando arquitecturas computacionales que admiten transformaciones no lineales múltiples e iterativas de datos expresados en forma matricial o tensorial. [3]

Los algoritmos de aprendizaje profundo se basan en las redes neuronales artificiales con aprendizaje de características (*feature learning*). El aprendizaje puede ser supervisado, semi-supervisado y no-supervisado. [4]

Varias arquitecturas de *deep learning*, como por ejemplo las redes neuronales recurrentes (*recurrent neural network*) y las redes neuronales convolucionales (*convolutional neural network*), han sido aplicadas a campos como visión por computador, reconocimiento automático del habla, procesamiento de lenguaje natural, traducción automática, bioinformática, etc., y han mostrado producir resultados de vanguardia en varias tareas. [5]

2.1.1. Formación

Para poder repasar el tema de forma ordenada y eficiente, además de realizar lecturas de blogs y *paper*s, decidí inscribirme al curso online *Deep Learning Specialization* en la plataforma de educación virtual Coursera [6] como complemento a lo ya aprendido en el máster. Este curso está dirigido por el informático americano Andrew Ng, el mismísimo cocreador de la plataforma en la cual se ofrece este curso.

El curso contiene vídeos explicativos sobre el tema y ejercicios de implementación en Python mediante *notebook*s de Jupyter. Los temas que trata son las redes neuronales básicas, las redes neuronales profundas, las redes neuronales convolucionales, algoritmos de optimización y técnicas para mejorar el rendimiento de las redes, como por ejemplo, la afinación de hiperparámetros, los métodos de regularización, la normalización por lotes el chequeo de gradientes, etc.

2.1.2. Redes neuronales

Las redes neuronales artificiales son sistemas computacionales basadas en las redes neuronales biológicas que constituyen el cerebro de los animales. Esta redes artificiales consisten en un conjunto de unidades llamadas neuronas artificiales las cuales están conectadas entre sí para poder transmitirse señales de una a otra. Al atravesar una red neuronal, la información de entrada se somete a diversas operaciones y produce unos valores de salida.

El objetivo de una red neuronal es resolver los problemas de la misma manera que el cerebro humano los resolvería, solo que las redes neuronales son más abstractas. Actualmente las redes neuronales más complejas suelen contener desde miles a millones de neuronas. Estos sistemas aprenden por su propia cuenta mediante el entrenamiento con datos, formándose a sí mismos.

El entrenamiento consiste en procesar ejemplos anotados y en formar asociaciones ponderadas por probabilidad entre un ejemplo y su correspondiente anotación. Las asociaciones formadas se almacenan dentro de la estructura de datos de la propia red. Durante cada ronda del entrenamiento la red intenta predecir el resultado de cada ejemplo, y al final de la ronda lo compara al resultado real (*ground truth*), a lo que viene a ser la correspondiente anotación. La diferencia entre los dos resultados es el error que la red utiliza en cada ronda para ajustar sus asociaciones ponderadas de acuerdo a una regla de aprendizaje que se haya establecido mediante hiperparámetros. Los ajustes sucesivos harán que la red neuronal produzca un resultado que sea cada vez más similar al resultado real. Después de un número suficiente de estos ajustes, el entrenamiento puede terminarse en base a ciertos criterios. Este proceso de entrenamiento se conoce como aprendizaje supervisado.

Las neuronas se organizan en varias capas (*layers*), y solamente pueden estar conectadas de una capa a otra, de la manera que se ve en la Figura 2.1. Es decir, una neurona no se puede conectar con las neuronas de su misma capa. La capa inicial, la cual recibe información externa, se denomina capa de entrada (*input layer*). La capa final, la cual produce el resultado definitivo, se denomina capa de salida (*output layer*). Entre estas dos capas puede haber cero o más capas, denominadas capas ocultas (*hidden layers*). A la hora de decir cuántas capas tiene una red, la capa inicial no se tiene en cuenta.

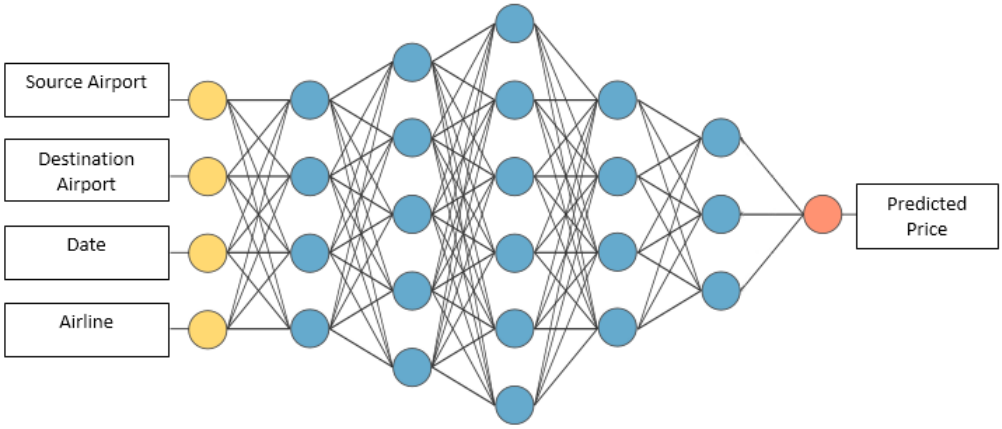

Figura 2.1: Ejemplo de una red neuronal de 6 capas, con sus neuronas *input* (amarillas), *output* (naranja) y *hidden* (azules).

Cada neurona está conectada con otras de una capa a otra a través de unos enlaces. En estos enlaces el valor de salida de la neurona anterior es multiplicado por un valor de peso (*weight*). A esta multiplicación también se le suma el valor *bias*, un valor constante que sirve para compensar el resultado. Los pesos en los enlaces pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. Del mismo modo, a la salida de la neurona, puede existir una función limitadora o umbral, que modifica el valor resultado o impone un límite que no se debe sobrepasar antes de propagarse a otra neurona. Esta función se conoce como función de activación (*activation function*).

A la hora de realizar el aprendizaje automático, normalmente se intenta minimizar una función de pérdida (*loss function*) que evalúa la red en su total. Por poner un ejemplo, el *cross-entropy loss function* mide el rendimiento de un modelo de clasificación cuya salida es un valor de probabilidad entre 0 y 1. El valor de la función aumenta a medida que la probabilidad predicha diverge del resultado real (*ground truth*). Los valores de los pesos de las neuronas se van actualizando buscando reducir el valor de la función de pérdida, proceso que se realiza mediante la propagación hacia atrás (*backpropagation*).

Los enlaces que hay entre dos capas pueden ser de diferentes tipos. Una de las opciones es que sean completamente conectadas (*fully connected*), donde todas las neuronas de una capa están conectadas con todas las neuronas de la siguiente capa. Otra de las opciones es que sean agrupadas (*pooling*), donde un grupo de neuronas de una capa está conectada a una sola neurona de la siguiente capa, reduciendo así la cantidad de neuronas en la siguiente capa. Este último tipo de enlace se utiliza en redes convolucionales.

2.1.3. Redes neuronales convolucionales

Las redes neuronales convolucionales (*convolutional neural networks*) son un tipo de redes neuronales artificiales que normalmente se aplican para analizar imágenes visuales. [7] Como el nombre lo indica, estas redes utilizan una operación matemática llamada convolución en al menos una de sus capas. Al utilizar la convolución en lugar de la multiplicación de matrices, se reduce consideradamente la cantidad de parámetros que hay que entrenar.

En una red neuronal convolucional, las capas ocultas incluyen capas que realizan convoluciones, como por ejemplo la red de la Figura 2.2. Normalmente suelen ser capas que realizan un producto escalar entre el filtro de convolución y la matriz de entrada de la capa. A medida que el filtro se desliza a lo largo de la matriz de entrada, la operación de convolución genera un mapa de características (*feature map*), lo cual a su vez contribuye a la entrada de la siguiente capa. A estas capas les suelen seguir otras como pueden ser las capas de agrupación (*pooling*) y/o las capas completamente conectadas (*fully connected*). También es muy común añadir la función de activación ReLU después de cada convolución y cada capa de *fully connected* y la función Softmax en la capa de salida.

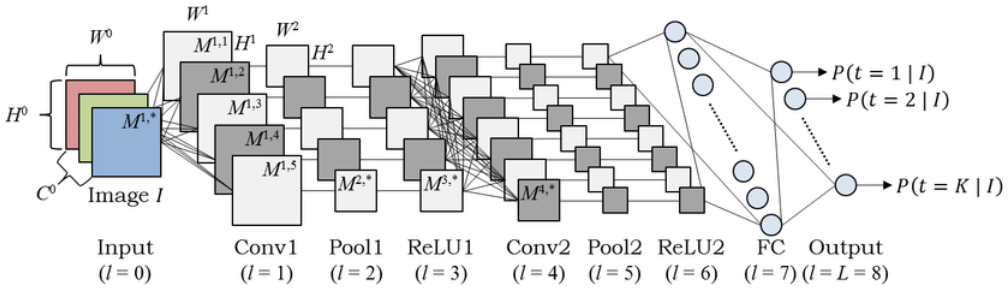

Figura 2.2: Ejemplo de una red neuronal convolucional de 8 capas que contiene 2 convoluciones y una capa *fully connected*.

Las redes convolucionales utilizan ”tensores”, un objeto algebraico que describe una relación multilineal entre conjuntos de objetos algebraicos relacionados con un espacio vectorial. Estos tensores suelen ser de cuatro dimensiones: [número de *input*s] x [altura de los *input*s] x [anchura de los *input*s] x [número de canales de los *input*s]. Al número de canales también se le suele llamar profundidad. Lo normal es que la altura y la anchura vayan reduciéndose, mientras que el número de canales va creciendo, como se puede ver en el ejemplo de la Figura 2.3. La altura y la anchura en las capas *fully connected* siempre es de 1.

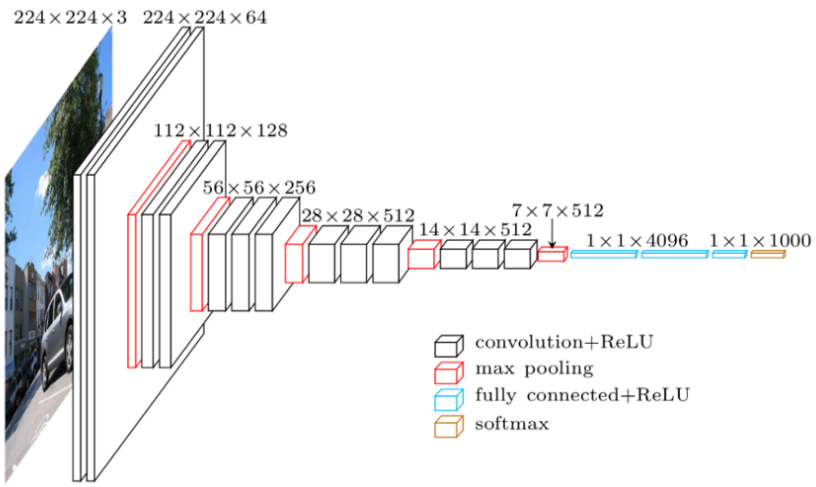

Figura 2.3: Ejemplo de como van cambiando las dimensiones de un *input* en una red neuronal convolucional.

Las capas convolucionales son el componente básico de una red convolucional. Los parámetros de la capa consisten en un conjunto de filtros que se pueden aprender. Los filtros se deslizan a lo largo de la matriz de entrada generando un mapa de características (*feature map*), de la manera que se ve en la Figura 2.4, teniendo el cuenta hiperparámetros como el *stride* (zancada) y el *padding* (relleno). Como resultado, la red aprende filtros que se activan cuando detectan algún tipo específico de característica en alguna posición espacial en el *input*.

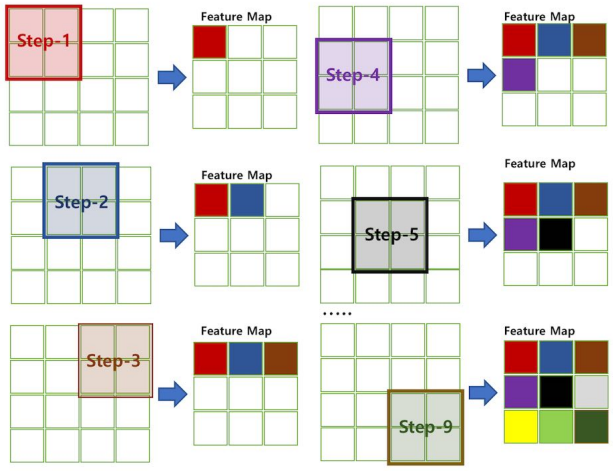

Figura 2.4: Ejemplo de un filtro deslizandose a lo largo de la matriz de entrada, con un *stride* de 1 y un *padding* de 0.

Asumiendo que si un detector de características (es decir, un filtro) es útil en alguna posición espacial seguramente también lo sea en otras posición espaciales, las capas convolucionales utilizan un sistema de compartición de parámetros. De esta manera, la cantidad de parámetros que la red debe aprender se reduce consideradamente.

En cambio, las capas *pooling* sirven para reducir las dimensiones de los datos, combinando grupos de neuronas en una sola neurona. Hay diferentes tipos de *pooling*, como el *max pooling*, el *min pooling* y el *average pooling*. Depende del tipo que se utiliza, la neurona resultante al combinar un grupo de neuronas será el valor mayor, el valor menor o el valor promedio del grupo, como su nombre en inglés lo indica.

Después de hacer varias convoluciones y varios *pooling*s, la clasificación final se realiza a través de capas *fully connected*, capas completamente conectadas como las que se usan en las redes no convolucionales.

Otros elementos muy comunes en las redes convolucionales son las funciónes de activación (*activation function*). Una de las más usadas es ReLU, la cual se suele utilizar al final de cada convolución para eliminar los valores negativos del mapa de activación poniéndolos a cero. Otra de las funciones típicas es Softmax, la cual se utiliza en la capa final *output* para representar la distribución categórica del resultado final, en otras palabras, para predecir las probabilidades de cada clase con valores entre 0 y 1.

2.2. Federated Learning

El aprendizaje federado, también conocido como aprendizaje colaborativo, es una técnica de aprendizaje automático que entrena un algoritmo a través de una arquitectura descentralizada formada por múltiples dispositivos periféricos o servidores, los cuales contienen sus propios datos locales y no los intercambian con los demás. Esta técnica empezó a surgir como tema de investigación a partir de 2015 [8] [9], siendo Google el primero en proponer un *framework* para ello en 2016. [10]

El enfoque del aprendizaje federado contrasta con las técnicas tradicionales de aprendizaje automático centralizado donde todos los conjuntos de datos locales se cargan en un servidor, ya que en un modelo federado cada participante mantiene sus datos locales en privado. De la misma manera, también contrasta con los enfoques descentralizados más clásicos que distribuyen las muestras de datos locales de manera idéntica, ya que en un modelo federado lo más probable es que cada participante tenga un conjunto de datos bastante diferente al resto.

Una de las diferencias entre el aprendizaje federado y el aprendizaje distribuido son las suposiciones que se hacen sobre los datos. El aprendizaje distribuido distribuye los datos de manera homogénea entre los participantes, con el objetivo de paralelizar el rendimiento de cómputo. En cambio, el aprendizaje federado trabaja con conjuntos de datos muy heterogéneos de tamaños que pueden variar muchísimo de un participante a otro. Por esta razón, se dice que los datos son *non-idd* (*non-independent and non-identically distributed*). Además, los colaboradores involucrados en el aprendizaje federado pueden no ser confiables, ya que normalmente dependen de medios de comunicación menos potentes como el Wi-Fi, y por lo tanto están sujetos a más posibles fallos. En comparación, en el aprendizaje distribuido los nodos suelen ser centros de procesamiento de datos con poderosas capacidades computacionales conectados entre sí con redes muy rápidas. [11]

2.2.1. Arquitectura

La premisa básica detrás del aprendizaje federado es que, en vez de que los datos se muevan al lugar donde esté el modelo, el modelo se mueve al lugar donde estén los datos. De este modo, el único movimiento de datos que se realiza en un esquema federado son los parámetros del modelo que en cada ronda van actualizándose, permitiendo así abordar problemas críticos como la privacidad y seguridad de los datos, los derechos de acceso a los datos y el acceso a datos heterogéneos.

El aprendizaje consiste en que cada nodo entrene un modelo local sobre sus propias muestras de datos locales, y que con cierta frecuencia intercambie parámetros del modelo con el resto de nodos. De esta manera se genera un modelo global compartido por todos los nodos. Los parámetros que se comparten, dependiendo del método que se utiliza, pueden ser los pesos de la red neuronal o también los gradientes que se computan mediante un algoritmo de *backpropagation*.

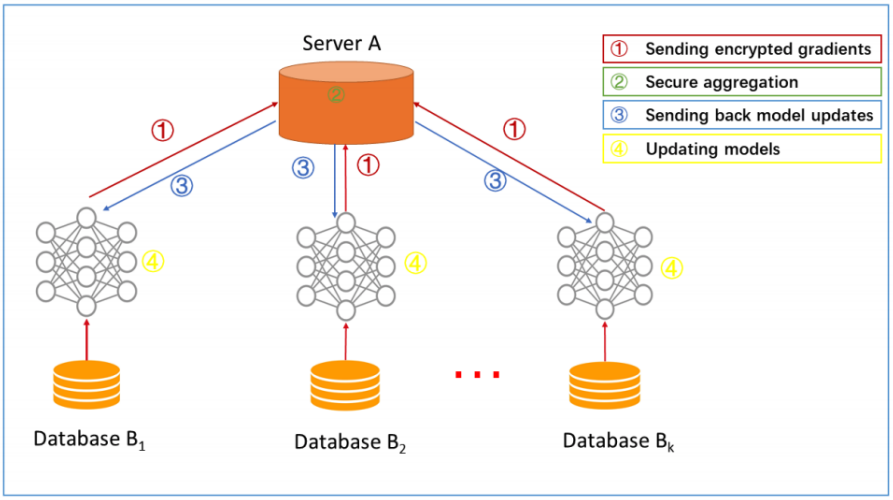

Figura 2.5: Arquitectura de un aprendizaje federado centralizado.

El proceso de entrenamiento se puede resumir en varios pasos que se repiten de forma iterativa:

1. El servidor central inicializa un modelo de aprendizaje y comienza la primera ronda.
2. El servidor realiza una selección de colaboradores entre todos los nodos locales disponibles.
3. Los colaboradores seleccionados reciben el modelo, mientras que el resto esperan a la siguiente ronda.
4. Los colaboradores entrenan el modelo recibido sobre sus datos locales, mientras el servidor espera.
5. Los colaboradores envían el modelo entrenado al servidor central.

6. El servidor recibe los modelos actualizados y realiza una agregación, generando un único modelo global.
7. El servidor finaliza la ronda y comienza a una nueva, volviendo a la selección de colaboradores.
8. El aprendizaje termina una vez el criterio de terminación predefinido se cumpla.

El servidor central ejerce de director para coordinar todos los nodos, y se encarga de llevar a cabo cada paso del algoritmo. Durante el entrenamiento del modelo es posible que algún nodo falle, y por tanto que su modelo actualizado nunca llegue. El servidor también se encarga de estos casos, dejando de esperar al colaborador pasado un tiempo razonable y en caso de que se vea necesario abandonando la ronda actual.

Se dice que el aprendizaje federado es centralizado cuando un servidor central es parte de la arquitectura del sistema, como se ve en la Figura 2.5. De la misma manera, también es posible realizar el aprendizaje sin ningún tipo de servidor central, solamente coordinando los nodos entre ellos para obtener el modelo global. A esto se le llama aprendizaje federado descentralizado.

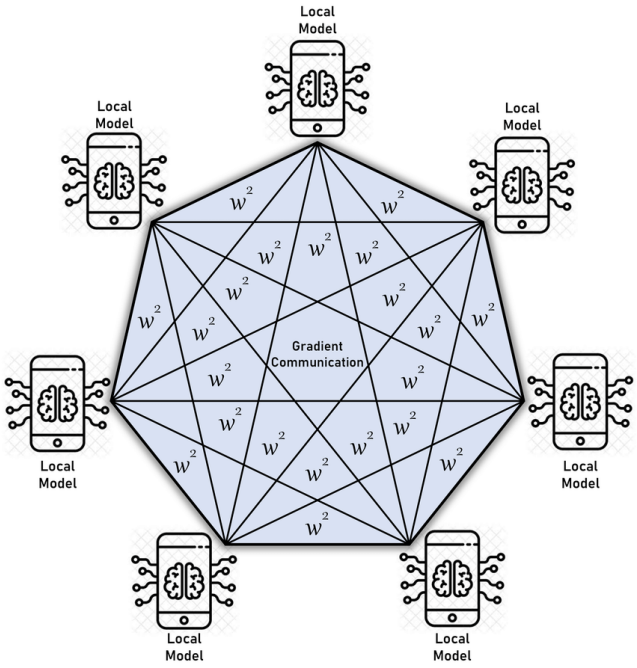

Figura 2.6: Arquitectura de un aprendizaje federado descentralizado.

El esquema descentralizado puede usar diferentes topologías de red, todos los nodos no tienen por qué estar conectados entre ellos como se ve en la Figura 2.6. Cada nodo comparte los parámetros actualizados con sus nodos vecino, lo que a menudo resulta en que se reduzca el trafico de comunicación y del tiempo de entrenamiento. Cada nodo actualiza su modelo haciendo una agregación de sus propios parámetros y los parámetros que recibe de sus vecinos. [12]

Para hacer la agregación de los parámetros actualizados, ya sea en una arquitectura centralizada o descentralizada, hay diferentes formas posibles. Las técnicas más comunes son los variantes del método del descenso de gradiente estocástico (FedSGD: *federated stochastic gradient descent*), en donde los gradientes son promediados proporcionalmente al número de muestras de entrenamiento que hay en cada nodo y utilizados para realizar un paso de descenso de gradiente.

En el método anterior, si todos los nodos locales comienzan con la misma inicialización, promediar los gradientes es estrictamente equivalente a promediar los pesos mismos. Por esta misma razón, otro método muy común es el promedio federado (FedAvg: *federated averaging*), el cual es una generalización del método anterior que promedia los pesos en vez de promediar los gradientes. [13]

Recientemente, con el objetivo de abordar clientes heterogéneos los cuales están equipados con capacidades de comunicación y computación muy diferentes, una nueva arquitectura de aprendizaje federado llamada HeteroFL ha sido propuesta. Esta técnica permite el entrenamiento de modelos locales heterogéneos con complejidades de computación que varían dinámicamente, al igual que también permite que los modelos sean actualizados de manera asíncrona durante el proceso de entrenamiento. En comparación con los enfoques síncronos en los que los modelos locales se intercambian una vez que los cálculos para todas las capas de la red neuronal se han realizado, los asíncronos aprovechan las propiedades de las redes neuronales para intercambiar actualizaciones de modelos tan pronto como los cálculos de una determinada capa estén disponibles. [14]

2.2.2. Ventajas y desventajas

La mayor y principal ventaja del aprendizaje federado es la seguridad y privacidad de los datos. Con el sistema de intercambio de parámetros, al no tener que compartir datos directamente, se garantiza la confidencialidad de cada uno de los nodos y sus datos privados.

Sin embargo, al igual que tiene sus ventajas, también tiene sus desventajas o limitaciones. Una de ellas es la comunicación. El aprendizaje federado requiere de una comunicación frecuente entre los nodos durante el proceso de aprendizaje, por lo que requiere de un gran ancho de banda. No obstante, los dispositivos que normalmente se emplean en el aprendizaje federado suelen ser dispositivos IoT o teléfonos inteligentes que están conectados a redes Wi-Fi, por lo que a pesar de que transmitir parámetros suele ser mucho menos costoso que transmitir datos, es probable que aun así el ancho de banda no sea suficiente. [11]

La capacidad del dispositivo local para ser entrenado también puede generar un problema, ya que algunos dispositivos puede que no tengan la capacidad de cómputo necesaria para poder llevar a cabo el entrenamiento que el aprendizaje federado requiere.

La heterogeneidad entre los diferentes conjuntos de datos locales es una de las mayores limitaciones al entrenar un modelo. Cada nodo puede tener algún sesgo con respecto a la población general, y el tamaño de los conjuntos de datos puede variar significativamente, lo que dificulta la convergencia. Los datos locales de cada nodo pueden estar actualizándose periódicamente, pudiendo también generar una heterogeneidad temporal.

Otra de las desventajas es la vulnerabilidad del modelo global. Un fallo en un nodo, como por ejemplo la pérdida parcial o total de los parámetros actualizados, puede terminar afectando al modelo global. De la misma manera, un atacante que participa en el aprendizaje como un nodo más puede llegar a dañar gravemente el modelo global utilizando datos locales falsos y haciendo que el modelo aprenda lo que al atacante le convenga. [15]

2.2.3. Privacidad

Sin lugar a dudas, la privacidad es el punto mas esencial del aprendizaje federado. Al fin y al cabo, la principal ventaja de utilizar un enfoque federado para realizar el aprendizaje automático es poder garantizar la privacidad de los datos. Los datos se mantienen localmente, ni se suben a ningún sitio ni se comparten con nadie. El conjunto de datos entero que se utiliza en un entrenamiento federado lo forman datos locales que en ningún momento salen de su respectivo nodo.

Lo único que se intercambia durante un aprendizaje federado son los parámetros que se van aprendiendo. Además es posible cifrar estos parámetros antes de compartirlos entre las rondas de aprendizaje, ampliando así la privacidad aun más. También se puede utilizar un cifrado homomórfico para realizar los cálculos directamente sobre los datos cifrados, sin tener que descifrarlos de antemano. Aun así, a pesar de todas estas medidas de protección, los parámetros pueden filtrar información sobre los datos indirectamente. [16] Para abordar problemas como este, diferentes técnicas de protección de datos han sido y se siguen desarrollando. [17]

Cualquier nodo que contribuya durante el entrenamiento federado puede lanzar un ataque diferencial, pudiendo revelar mediante el análisis del modelo distribuido la contribución de un colaborador durante el entrenamiento y la información sobre su conjunto de datos. Para abordar este problema se propone el algoritmo de la privacidad diferencial (*differential privacy*), con el objetivo de ocultar las contribuciones de los colaboradores durante el entrenamiento, teniendo en cuenta el equilibrio entre la pérdida de privacidad y el rendimiento del modelo. [18]

Utilizando la privacidad diferencial se introduce ruido en los parámetros, cosa que agrega incertidumbre a cualquier intento de revelar datos privados de colaboradores, pero también reduce la precisión del modelo compartido, lo que limita la escala útil de utilizar esta técnica. Para abordar este problema se propone utilizar la computación segura multipartita (*secure multiparty computation*) en conjunto con la privacidad diferencial, con el objetivo de generar modelos más precisos. [19]

2.2.4. Ámbitos de uso

El aprendizaje federado generalmente se aplica cuando clientes individuales necesitan entrenar modelos en conjuntos de datos más grandes que los suyos pero no pueden permitirse compartir sus propios datos con el resto. Esto hace que el aprendizaje federado tenga un ámbito de uso relativamente pequeño, dado que en el caso de que compartir los datos no suponga ningún problema generalmente otros tipos de aprendizaje automático consiguen mejores resultados.

A pesar de todo, cada vez están apareciendo mas ámbitos en los que la utilización del aprendizaje federado genera muchas ventajas. Estas son las principales áreas en las que hoy en día se esta utilizando:

Industria 4.0: El uso de las técnicas de aprendizaje automático en la industria es algo que se está volviendo fundamental para mejorar la eficiencia y la eficacia de los procesos industriales al tiempo que se garantiza un alto nivel de seguridad. [20] Sin embargo, la privacidad de los datos confidenciales es de suma importancia para las industrias y las empresas de fabricación. Los algoritmos de aprendizaje federado pueden ser aplicados a estos problemas, dado que no revelan ningún dato sensible.

Transporte: Los vehículos autónomos funcionan gracias a muchas tecnologías de aprendizaje automático. La visión por computadora es esencial para analizar obstáculos, permitiendo al vehículo adaptar su ritmo al entorno mediante el aprendizaje automático. Debido a la alta cantidad de vehículos autónomos que se espera que haya, y la necesidad de que respondan rápidamente a situaciones del mundo real, el enfoque del aprendizaje tradicional de la nube puede generar grandes riesgos de seguridad. El aprendizaje federado puede ser una gran solución para limitar el volumen de las transferencias de datos y acelerar el proceso de aprendizaje de los vehículos. [21]

Medicina: El campo médico y sanitario es un gran sector en el que el aprendizaje federado puede sacar su máximo potencial. El enfoque estándar actual de centralizar los datos de múltiples centros en una misma nube genera preocupaciones críticas sobre la privacidad del paciente y la protección de datos. El aprendizaje federado resuelve este problema, ya que con un esquema federado las instituciones médicas pueden entrenar un modelo de aprendizaje sobre un enorme conjunto de datos sin comprometer la seguridad de sus propios datos. Todo esto supone un gran avance, y hoy en día se está trabajando intensamente en ello. [22]

2.2.5. Frameworks

Los *framework*s o marcos de trabajo para el aprendizaje federado son proyectos muy recientes que la mayoría aún tiene mucho que pulir. Por esta razón, los repositorios GitHub de estos proyectos no paran de recibir actualizaciones. Estos son los marcos de trabajo más importantes por el momento:

OpenFL - Open Federated Learning: OpenFL es un *framework* de código abierto escrito en Python y desarrollado por Intel. Sirve para entrenar modelos simulando un esquema federado. OpenFL funciona con modelos de entrenamiento creados mediante PyTorch, TensorFlow y Keras, y se puede extender fácilmente a otros marcos de aprendizaje profundo. Contiene tutoriales intuitivos y ofrece la posibilidad de utilizar funciones de agregación personalizadas. [23]. Este *framework* se ha lanzado este mismo año, y desde entonces ha recibido constantes actualizaciones. Su primer uso se ha dado en la primera competición computacional jamás propuesta sobre el aprendizaje federado, en el desafío internacional *FeTS 2021*. [24]

TFF - TensorFlow Federated: TFF es otro *framework* de código abierto escrito en Python y desarrollado por Google. Permite simular algoritmos de aprendizaje federado en modelos de entrenamiento creados mediante TensorFlow o Keras, a la vez que también permite experimentar con algoritmos nuevos. Los componentes básicos que TFF proporciona también se pueden utilizar para implementar cálculos que no sean de aprendizaje, como por ejemplo estadísticas agregadas sobre datos descentralizados. La pagina web contiene guías y mucha información útil para todo lo relacionado con el aprendizaje federado. [25]

IBMFL - IBM Federated Learning: IBMFL es un *framework* escrito en Python para el entorno empresarial. Proporciona una estructura básica para el aprendizaje federado, a la que se le pueden agregar funciones avanzadas. No depende de ningún marco de aprendizaje automático específico, y admite diferentes topologías de aprendizaje. Soporta tanto modelos de redes neuronales como modelos de técnicas clásicas del aprendizaje automático, ya sea de modo supervisado, de modo no supervisado, o con aprendizaje por refuerzo. También soporta múltiples algoritmos de agregación recientes, y ofrece la opción de poder diseñar agregadores propios. [26] Cuenta con guías y tutoriales para poder entender la configuración, y proporciona herramientas para implementar un escenario real. [27]

PaddleFL - Paddle Federated Learnig: PaddleFL es un *framework* de código abierto basado en la plataforma PaddlePaddle (*parallel distributed deep learning*) desarrollado originalmente por Baidu. Permite replicar y comparar fácilmente diferentes algoritmos de aprendizaje federados. Facilita la implementación de un sistema de aprendizaje federado en clústeres distribuidos a gran escala. Proporciona varias estrategias de aprendizaje federado con aplicación en visión por computadora, procesamiento del lenguaje natural, etc. Estrategias de aprendizaje tradicional también son proporcionados, como el aprendizaje multitarea y el aprendizaje por transferencia en entornos federados. [28]

PySyft: PySyft es una librería de Python de código abierto desarrollada por OpenMined y orientada al aprendizaje profundo seguro y privado. PySyft separa los datos privados y el entrenamiento del modelo, utilizando el aprendizaje federado, la privacidad diferencial y la computación encriptada dentro de los principales marcos de aprendizaje profundo como PyTorch y TensorFlow. [29]

Flower: Flower es un *framework* amigable de código abierto que ofrece un enfoque unificado para el aprendizaje federado. Es muy personalizable y permite utilizar cualquier marco de aprendizaje automático y cualquier lenguaje de programación. [30] Cuenta con mucha documentación de ayuda. [31]

FedLab: FedLab es un *framework* flexible de código abierto basado en PyTorch. Ha sido lanzado hace un par de meses, y su principal característica es ser altamente personalizable. FedLab proporciona los módulos necesarios para la simulación del aprendizaje federado, incluida la comunicación, la compresión, la optimización del modelo, la distribución de datos y otros más. Los usuarios pueden crear un entorno de simulación federada con módulos personalizados, como si estuvieran jugando con ladrillos de LEGO. [32] Cuenta con algoritmos implementados de referencia para una mejor comprensión y un uso fácil. [33]

FATE - Federated AI Technology Enabler: FATE es un proyecto de código abierto iniciado por WeBank para proporcionar un *framework* seguro para realizar el aprendizaje federado. Implementa protocolos de seguridad basados en la encriptación homomórfica y la computación multipartita. Además de arquitecturas de aprendizaje federado, también soporta varios algoritmos de aprendizaje automático, incluida la regresión logística, algoritmos basados en árboles, el aprendizaje profundo y el aprendizaje por transferencia. [34]

2.2.6. Tipologías de agregadores

Los agregadores son un elemento esencial en el aprendizaje federado. Se encargan de combinar mediante un algoritmo las actualizaciones de modelo provenientes de múltiples nodos. Los cambios en este algoritmo pueden acelerar la convergencia, reducir el tiempo de entrenamiento o mejorar la solidez del modelo. He aquí una lista de varios tipos de algoritmo del estado del arte:

FedSGD - Federated Stochastic Gradient Descent: FedSGD es el algoritmo base del aprendizaje federado. Se basa en el método del descenso de gradiente estocástico. Cada colaborador comparte sus gradientes locales y el servidor hace un promedio proporcionalmente al número de muestras de entrenamiento de cada colaborador, realizando un paso de descenso de gradiente.

FedAvg - Federated Averaging: FedAvg es el método más básico que se usa comúnmente. Se basa en el algoritmo anterior, solo que promedia los pesos en vez de promediar los gradientes. Los colaboradores realizan varios pasos en el descenso de gradiente, y después envían los pesos actualizados de sus modelos al servidor para que este los promedie. [13]

Secure Aggregation: Los protocolos de agregación segura permiten a múltiples clientes que desconfían mutuamente calcular de manera colaborativa la combinación de sus parámetros privados sin que se revelen los valores de los parámetros. A día de hoy se han propuesto muchas variaciones de este protocolo. Una de ellas proporciona una comunicación eficiente para datos de alta dimensión, y tolera que hasta un tercio de los colaboradores no completen el protocolo. [35] Otra de ellas introduce una nueva métrica para cuantificar las garantías de privacidad del aprendizaje federado en múltiples rondas de entrenamiento, y realiza una selección estratégica de colaboradores para garantizar la privacidad de cada uno a largo plazo. [36]

RFA - Robust Federated Aggregation: RFA es un algoritmo para hacer que el aprendizaje federado tenga una configuración robusta y segura cuando una fracción de los dispositivos puede estar enviando actualizaciones dañadas al servidor. Este enfoque de agregación es independiente del nivel de corrupción que haya: supera la agregación clásica en términos de robustez cuando el nivel de corrupción es alto, manteniéndose competitivo en el caso de baja corrupción. [37]

FedProx: FedProx puede verse como una generalización y reparametrización de FedAvg. A pesar de que solo añade modificaciones pequeñas, FedProx demuestra un comportamiento de convergencia significativamente más estable y preciso en relación con FedAvg, particularmente en entornos muy heterogéneos. Este método proporciona garantías de convergencia al aprender datos de distribuciones no idénticas (heterogeneidad estadística), y se adhiere a las restricciones de los sistemas a nivel de dispositivo permitiendo que cada dispositivo participante realice una cantidad variable de trabajo (heterogeneidad de sistemas). [38]

Krum: Krum es el primer algoritmo que se propuso para hacer frente a las fallas bizantinas (comportamientos arbitrarios y potencialmente adversarios) en las implementaciones del descenso de gradiente estocástico (SGD). Estas fallas pueden ocurrir ya sea por errores de software, por asincronías en la red, por sesgos en los conjuntos de datos locales, o incluso por atacantes que intentan poner en riesgo todo el sistema. [39]

Coordinate-wise median: Al igual que Krum, este algoritmo también tiene como objetivo hacer frente a las fallas bizantinas que particularmente ocurren en entornos descentralizados. Este método es un algoritmo de descenso de gradiente (GD) basado en la mediana de coordenadas, especialmente robusto contra los fallos arbitrarios, y a diferencia de Krum, con un enfoque en lograr un rendimiento estadístico óptimo. [40]

Zeno: Similar a los dos algoritmos anteriores, Zeno también es una técnica para hacer frente a los fallos que pueden dañar el modelo global. Se aplica particularmente en el descenso de gradiente estocástico (SGD), haciendo que el aprendizaje sea tolerante a un número arbitrario de colaboradores defectuosos. Zeno trata a todos los colaboradores como sospechosos, y crea una clasificación de fiabilidad en cada iteración. De esta manera logra sospechar de colaboradores potencialmente defectuosos. [41]

3. CAPÍTULO

Herramientas de trabajo

Para llevar a cabo los dos casos de uso que explicaré en los capítulos 4 y 5, he utilizado unas herramientas específicas como mi entorno de trabajo. Todas estas herramientas son de código abierto, por lo que se pueden utilizar y modificar gratuitamente.

3.1. Anaconda

Lo primero de todo fue instalar Anaconda, una distribución de *software* que viene con el intérprete de Python y varios paquetes relacionados con el aprendizaje automático y la ciencia de datos. Anaconda contiene un sistema llamado Conda para gestionar paquetes y entornos. Este gestor permite tener varios entornos de trabajo con diferentes paquetes en cada uno, de la manera que se aprecia en la Figura 3.1, pudiendo cambiar fácilmente de uno a otro.

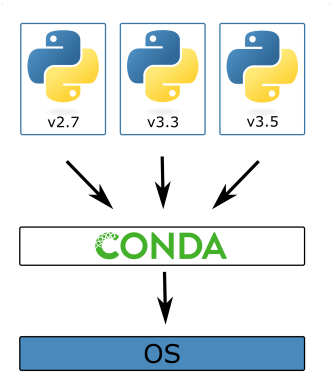

Figura 3.1: Ejemplo de varios entornos, cada uno con una versión diferente de Python.

Las dependencias de algunos paquetes pueden entrar en conflicto con otros paquetes previamente instalados, lo que puede causar muchos errores. A diferencia del gestor de paquetes Pip, al instalar un nuevo paquete en un entorno, Conda comprueba los paquetes que ya están instalados de antes y solamente instala las dependencias que sean compatibles con el entorno, de esta manera teniendo todo bien organizado y libre de conflictos.

3.2. Jupyter Notebook

Mediante Conda creé un entorno de Python y dentro instalé los paquetes de Jupyter, entre los cuales se encontraba Jupyter Notebook, un entorno computacional e interactivo basado en la web para crear y editar documentos de tipo *notebook*, como se muestra en la Figura 3.2.

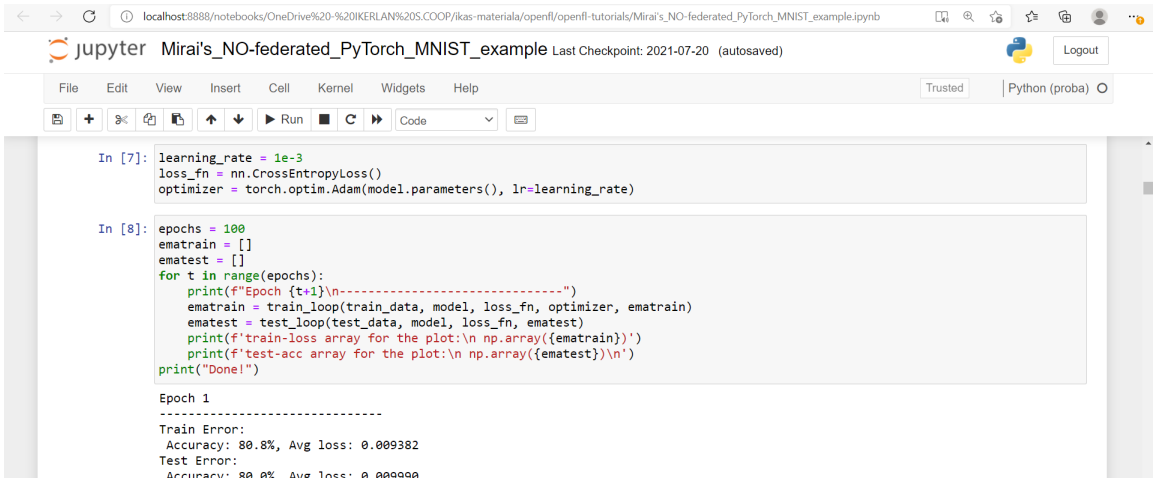

Figura 3.2: Ejemplo de un cuaderno de Jupyter.

Estos documentos están formados por una lista ordenada de celdas que pueden contener código, texto de tipo Markdown, formulas matemáticas, representaciones gráficas y multimedia interactiva. Los *notebook*s pueden conectarse a diferentes núcleos (*kernel*s) para permitir la programación en muchos idiomas. Por defecto se conectan al núcleo de IPython. El núcleo ejecuta el código que hay en las celdas y devuelve el resultado en la propia interfaz debajo de cada celda. Las celdas permiten ejecutar el código paso por paso.

Los *notebook*s de Jupyter han sido la herramienta principal para escribir y ejecutar el código en los dos casos de uso que explicaré en los siguientes capítulos. El núcleo para ejecutar mi código ha sido el entorno de Python que he creado mediante Conda.

3.3. PyTorch

Para escribir el código he usado PyTorch, una librería de código abierto para el aprendizaje automático. PyTorch proporciona la computación de tensores al igual que NumPy, pero con una aceleración fuerte a través de GPUs. A su vez, también proporciona todo tipo de bloques para una fácil construcción de redes neuronales.

3.4. OpenFL

El marco que he utilizado para realizar el aprendizaje federado ha sido OpenFL, uno de los *framework*s más nuevos en el estado del arte. La razón por la que he decidido utilizar este frente a los demás ha sido la posibilidad de usarlo para un caso de uso específico (capítulo 5 de la memoria) y el hecho de que se trata de uno de los *framework*s más novedosos y prometedores para el aprendizaje federado.

OpenFL proporciona unas funciones para entrenar nuestro modelo en una simulación de un esquema federado. Permite elegir la cantidad de colaboradores y el número de rondas. Además, ofrece la opción de personalizar el algoritmo de agregación mediante una función. A día de hoy cuenta seis simples funciones de agregación previamente definidas, pero los desarrolladores tienen la intención de añadir más.

3.5. Git

Para poder utilizar OpenFL como marco de trabajo, he descargado su código fuente desde su repositorio de GitHub [42] mediante Git, un software para rastrear cambios en cualquier conjunto de archivos que generalmente es utilizado para coordinar el trabajo entre programadores que desarrollan código fuente de manera colaborativa durante el desarrollo de un software. De la misma manera, cada vez que ha habido un cambio en el código por parte de los desarrolladores, he utilizado Git para actualizar mis archivos de código.

3.6. Otros

Durante las primeras fases del proyecto se analizó también el uso de otras herramientas como Docker, Keras o TFF aunque finalmente fueron desechadas debido a los requisitos de los casos de uso implementados.

4. CAPÍTULO

Caso de uso I — Análisis del impacto del aprendizaje federado (K-MNIST)

Con la intención de poner en practica la teoría y analizar las situaciones en las que el aprendizaje federado podría ser útil, he realizado unos experimentos con diferentes agregadores y diferentes distribuciones de datos.

4.1. Descripción

El objetivo principal de esta tarea es hacer que cada colaborador entrene los datos de una sola clase y experimentar metiéndole a cada uno diferentes porcentajes de datos de las demás clases. La idea detrás de esto es que en una situación real cada colaborador probablemente tendrá datos muy diferentes comparando al resto. De esta manera, se quiere analizar el impacto que el aprendizaje federado tiene sobre entornos desbalanceados. Por otra parte, también se quiere comparar la eficacia de diferentes métodos de agregación.

4.2. Dataset

PyTorch contiene un paquete de librería llamado TorchVision que consta de varios conjuntos de datos conocidos predefinidos para la visión por computador. [43] Uno de esos *datasets* es el Kuzushiji-MNIST (abreviado como K-MNIST), el cual ha sido utilizado para realizar esta tarea.

K-MNIST, como su nombre lo indica, es una variación del MNIST clásico, siendo la única diferencia que K-MNIST utiliza caracteres *kuzushiji* en lugar de los números del cero al nueve que el MNIST ordinario utiliza.

Los *kuzushiji* son básicamente caracteres japoneses desfigurados, una escritura cursiva que se usó en Japón durante más de mil años y que se ha dejado de usar en los tiempos modernos, especialmente desde la modernización del idioma en 1868. Esta escritura simplifica los caracteres japoneses (ya sean *kanji*s o *kana*s) de tal manera que suelen ser difíciles o incluso imposibles de reconocer. En la actualidad, la mayoría de los japoneses no son capaces de leer una gran parte de documentos históricos por estar escritos de esta manera anticuada. [44]

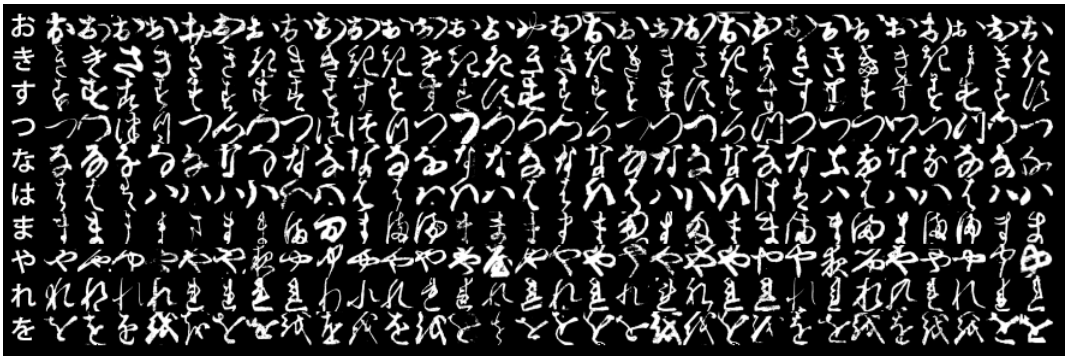

Figura 4.1: Algunos ejemplos de las muestras que contiene el K-MNIST. Cada fila pertenece a una clase, y la primera columna muestra el *kana* moderno equivalente de cada clase.

El conjunto de datos esta formado por imágenes de 28x28 en escala de grises que abarcan 10 clases, y está perfectamente balanceado al igual que el MNIST orginal, con 60.000 muestras de entrenamiento y 10.000 muestras de validación. Como se puede ver en la Figura 4.1, las clases que el *dataset* contiene son 10 *kana*s diferentes: お (o), き (ki), す (su), つ (tsu), な (na), は (ha), ま (ma), や (ya), れ (re) y を (wo).

4.3. Métricas de evaluación

Para evaluar el rendimiento de los modelos de entrenamiento, se han utilizado las mismas métricas que están predefinidas en el propio *framework*:

- ***global-acc*** (valor entre 0 y 1, siendo 1 lo mejor) $\longrightarrow$ Tasa de acierto del modelo global: cada colaborador utiliza el modelo agregado que el servidor le ha mandado para predecir la clase de sus datos de validación.
- ***train-loss*** (valor entre ∞ y 0, siendo 0 lo mejor) $\longrightarrow$ Coste del entrenamiento: cada colaborador entrena el modelo recibido sobre sus datos de entrenamiento y evalúa el coste del entrenamiento.
- ***local-acc*** (valor entre 0 y 1, siendo 1 lo mejor) $\longrightarrow$ Tasa de acierto del modelo local: cada colaborador utiliza el modelo que ha entrenado para predecir la clase de sus datos de validación.

Cada colaborador calcula localmente las tres métricas y se las envía al servidor central. Una vez recibidas las métricas de todos los colaboradores, el servidor calcula las métricas globales haciendo la media de todas.

4.4. Red neuronal

La red principal que se ha utilizado para realizar esta tarea ha sido la misma que viene en el tutorial PyTorch de OpenFL. [45] Es una red muy simple formada de la manera que se muestra en la Figura 4.2.

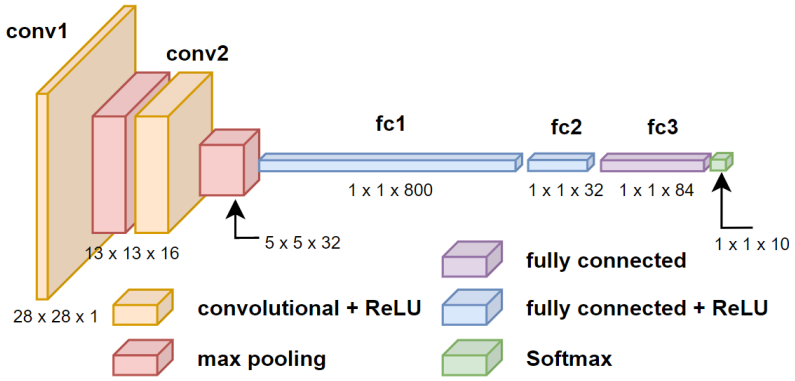

Figura 4.2: Esquema de la red neuronal principal.

Con el objetivo de mejorar esta red, también se ha creado una red secundaria aplicando técnicas de regularización a la red principal. Esta red contiene una capa Dropout después de cada función ReLU, quedándose de la manera que se muestra en la Figura 4.3.

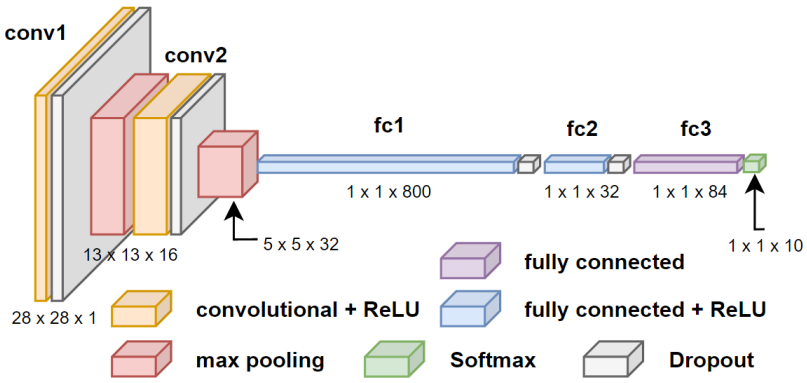

Figura 4.3: Esquema de la red neuronal secundaria.

La técnica Dropout consiste en poner a cero el peso de varias neuronas aleatorias para omitirlas y así intentar evitar el sobreajuste (*overfitting*). El sobreajuste ocurre cuando un modelo aprende demasiado los datos específicos de entrenamiento y por lo tanto no se adapta muy bien a nuevos datos, como se ve en la Figura 4.4.

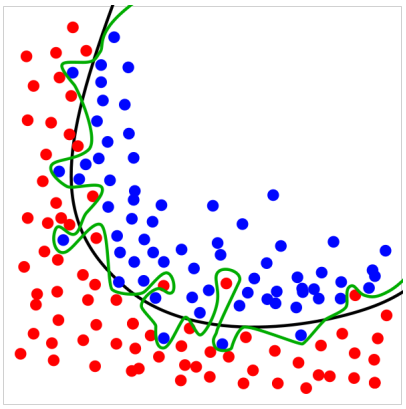

Figura 4.4: La línea verde representa un modelo sobreajustado, mientras que la línea negra representa un modelo ideal.

Otra técnica que también se ha utilizado en conjunto con la red secundaria es la regularización L2. Esta regularización se define dentro de la función de optimización al crear el modelo federado. La regularización L2 intenta estimar la media de los datos para evitar el sobreajuste. La técnica consiste en añadir una pequeña penalización llamada decadencia de pesos (*weight decay*) a la función de pérdida, penalización que se define como la suma de los valores al cuadrado de todos los pesos. [46]

4.5. Entrenamiento de los modelos

Cada modelo se ha entrenado con un modo de agregación y una distribución de datos diferente. A pesar de estas diferencias, todos los modelos tienen en común la cantidad de colaboradores, la cantidad de muestras totales que cada colaborador entrena por cada ronda, y los hiperparámetros de entrenamiento.

4.5.1. Selección de colaboradores

El *dataset* contiene 10 clases, y como ya he mencionado antes, el objetivo de la tarea es que cada colaborador entrene una clase, de modo que se han utilizado 10 colaboradores. Durante las ejecuciones, en todas las rondas federadas, siempre se han seleccionado las 10 clases presentes.

4.5.2. Hiperparámetros

El número de rondas de entrenamiento es uno de los hiperparámetros personalizables, el cual he establecido a 100 para todas las ejecuciones. Cada colaborador solamente hace una ronda de entrenamiento por cada ronda federada, por lo que una ronda de entrenamiento equivale a una ronda federada.

En cambio, el tamaño del lote (*batch size*) lo he establecido a 64. Este parámetro indica la cantidad de muestras que se propagan por la red en una sola iteración de ida y vuelta (*propagation* y *backpropagation*). Para que una ronda de entrenamiento o una época (*epoch*) se complete, se tienen que hacer tantas iteraciones como haga falta hasta propagar todas las muestras. Teniendo en cuenta que cada colaborador tiene 6.000 muestras de entrenamiento, se necesitarán 94 (6000/64) iteraciones para completar una ronda. En el caso del modelo no-federado, al tener todas las muestras en una sola maquina, se necesitarán 938 (60000/64) iteraciones.

Por otro lado, el ratio de aprendizaje (*learning rate*) lo he establecido a $1e{-}3$, a lo que viene a ser 0.001. Este hiperparámetro se utiliza para controlar el ajuste de los pesos de la red neuronal con respecto al gradiente de pérdida. Cuanto menor sea el valor más pequeños serán los pasos que se dan al moverse hacia el mínimo de la función de pérdida, y por lo tanto más lento será el descenso de gradiente. En cambio, cuanto mayor sea el valor más grandes serán los pasos, y por lo tanto habrá peligro de que se pase de largo el mínimo y que la red no consiga converger.

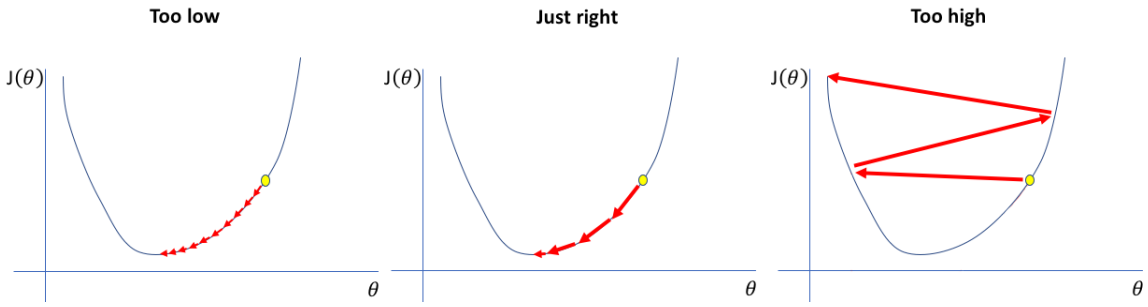

Figura 4.5: Comparativa entre un ratio de aprendizaje demasiado bajo (gráfico de la izquierda), un ratio adecuado (gráfico del medio) y un ratio demasiado alto (gráfico de la derecha).

Por último, otro de los parámetros es la cantidad de rondas que se guardan en la base de datos, la cual he estableciéndolo a 5. Algunos algoritmos de agregación utilizan tensores de las rondas anteriores para computar los tensores agregados. Estableciendo este parámetro a 5 estoy limitando la base de datos a guardar solamente los tensores de las últimas 5 rondas que se han realizado, reduciendo el consumo de RAM del sistema.

4.5.3. Funciones de agregación

En cada ronda las funciones de agregación reciben un tensor por cada colaborador, y devuelven el promedio de todos los tensores como un único tensor. Estos son los 9 algoritmos que he utilizado como funciones:

- ***SimpleAverage*** $\longrightarrow$ Promedio sencillo: devuelve simplemente el promedio de todos los tensores.
- ***WeightedAverage*** $\longrightarrow$ Promedio ponderado: devuelve el promedio de todos los tensores teniendo en cuenta la importancia (la cantidad de muestras de entrenamiento) de cada colaborador. Esta función viene predefinida en el código fuente de OpenFL, y se utiliza por defecto si no se elige ninguna función especifica.
- ***Median*** $\longrightarrow$ Mediana: devuelve la mediana de todos los tensores. Esta función viene predefinida en el código fuente de OpenFL.
- ***GeometricMedian*** $\longrightarrow$ Mediana geométrica: devuelve la mediana geométrica de todos los tensores. Esta función viene predefinida en el código fuente de OpenFL.
- ***ClippedAveraging*** $\longrightarrow$ Promedio recortado: dado un valor alpha y siendo delta las restas entre cada tensor actual y el tensor agregado de la ronda anterior, recorta todos los deltas al percentil de alpha, crea nuevos tensores sumando el tensor agregado de la ronda anterior con cada delta recortado, y devuelve el promedio ponderado de todos los nuevos tensores. Esta función viene predefinida en los tutoriales de OpenFL.
- ***ClippedAveragingFeTS*** $\longrightarrow$ Promedio recortado FeTS: hace lo mismo que el *ClippedAveraging* anterior, pero utilizando la función *np.percentile* de NumPy para recortar los deltas. Esta función viene predefinida en el código del desafío *FeTS 2021* que explico en el segundo caso de uso (capítulo 5 de la memoria).
- ***ExponentialSmoothingAveraging*** $\longrightarrow$ Promedio de suavizado exponencial: dado un valor alpha, aplica la formula del suavizado exponencial [47] sobre el promedio ponderado de todos los tensores. Esta función viene predefinida en los tutoriales de OpenFL.
- ***ConditionalThresholdAveraging*** $\longrightarrow$ Promedio de umbral condicional: dada una formula lambda que actúa como un umbral que va subiendo con cada ronda, devuelve el promedio de los tensores que cumplan el umbral. En caso de que ningún tensor cumpla el umbral, devuelve un promedio ponderado de todos los tensores. Esta función viene predefinida en los tutoriales de OpenFL.
- ***MiraiedAveraging*** $\longrightarrow$ Promedio miraieado: siendo delta las restas entre cada tensor actual y el tensor agregado de la ronda anterior, divide cada delta por el valor mirai, crea nuevos tensores sumando cada tensor actual con cada resultado multiplicado por dos, y devuelve el promedio ponderado de todos los nuevos tensores. Dado que cada tensor es una matriz, la suma de todos los tensores resulta en una única matriz. El valor mirai es la resta entre el valor máximo de esa matriz y la media de todos los valores de la misma matriz. Este algoritmo ha sido inventado por mí, y por ende lleva mi nombre, Mirai. La Figura 4.6 muestra el pseudocódigo de la función.

```
tensor_sum = sum(all_tensors)
mirai = np.max(tensor_sum) - np.mean(tensor_sum)
prev_tensor = aggregated tensor from prev_round
for col_tensor in all_tensors:
    delta = col_tensor - prev_tensor
    new_tensor = col_tensor + 2 * delta / mirai
np.average(miraied_tensors, weights)
```

Figura 4.6: Pseudocódigo del *MiraiedAveraging*.

4.5.4. Distribución de datos

De las 60.000 muestras de entrenamiento totales del *dataset*, cada colaborador siempre ha tenido 6.000 muestras de entrenamiento. De la misma manera, de las 10.000 muestras de validación totales del *dataset*, cada colaborador siempre ha tenido 1.000 muestras de validación.

A cada colaborador se le ha asignado una clase como su clase principal, y el porcentaje de muestras de dicha clase ha sido diferente dependiendo del experimento realizado:

- *100 %* $\longrightarrow$ El 100 % de las muestras son de su clase principal.
- *99 %* $\longrightarrow$ El 99 % de las muestras son de su clase principal, y 1 % del resto de clases.
- *95 %* $\longrightarrow$ El 95 % de las muestras son de su clase principal, y 5 % del resto de clases.
- *90 %* $\longrightarrow$ El 90 % de las muestras son de su clase principal, y 10 % del resto de clases.
- *80 %* $\longrightarrow$ El 80 % de las muestras son de su clase principal, y 20 % del resto de clases.
- *70 %* $\longrightarrow$ El 70 % de las muestras son de su clase principal, y 30 % del resto de clases.
- *60 %* $\longrightarrow$ El 60 % de las muestras son de su clase principal, y 40 % del resto de clases.
- *50 %* $\longrightarrow$ El 50 % de las muestras son de su clase principal, y 50 % del resto de clases.

Los experimentos *100 %* y *80 %* se han hecho tanto con la red principal (*basic NN*) como con la secundaria (*updated NN*), mientras que el resto solamente se han hecho con la red principal.

Para poder realizar estas especificas distribuciones de datos he tenido que cambiar el código fuente del *framework*. Las muestras del resto de clases que cada colaborador tiene están asignadas aleatoriamente, es decir, no tienen por qué contener la misma cantidad de muestras por cada clase no principal.

4.5.5. Elementos añadidos

Los pesos iniciales de un modelo se establecen aleatoriamente, pero la aleatoriedad de una computadora siempre se tiene que basar en algo, ya que no son capaces de tomar decisiones aleatorias. Por lo tanto, he añadido una semilla (*seed*) fija para que las ejecuciones de un mismo modelo en unas mismas condiciones ejecuciones siempre den los mismos resultados. Lo que una semilla hace es fijar una base para la aleatoriedad, de modo que los resultados aleatorios siempre serán los mismos cuando se use una misma semilla.

4.6. Resultados

He entrenado un modelo por cada distribución y agregación diferente, lo que suma 90 modelos en total. Si contamos el modelo no-federado que también he entrenado para comparar el aprendizaje automático ordinario con los modelos federados, serían 91 modelos en total, como se puede ver en la Figura 4.7. Al haber hecho tantos experimentos, solamente enseñaré los resultados más significativos.

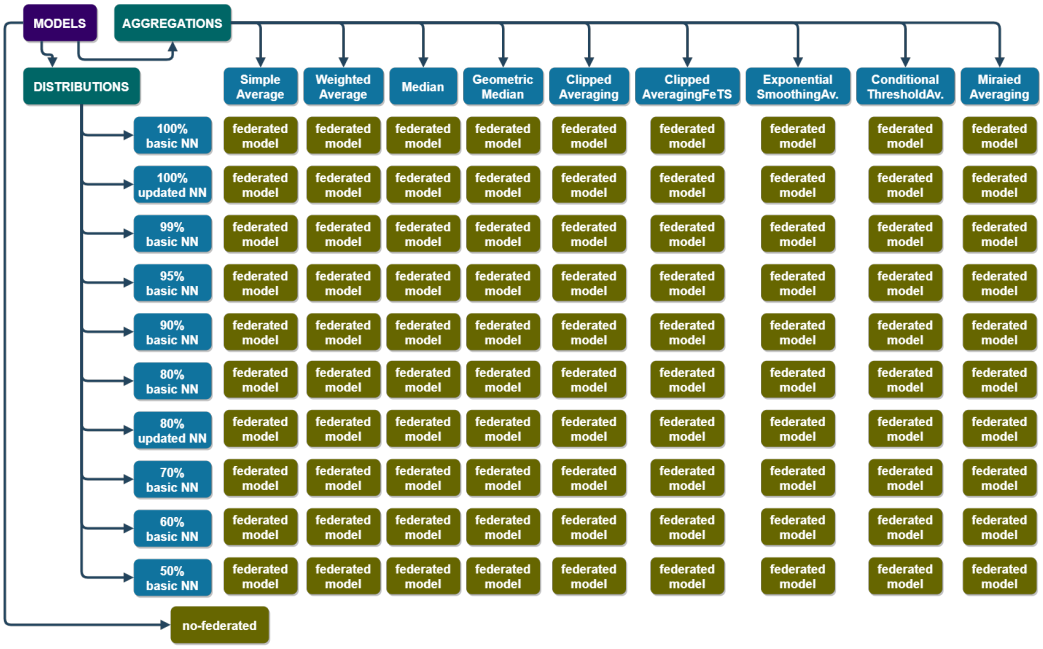

Figura 4.7: Modelos entrenados para el primer caso de uso.

4.6.1. Modelos federados con diferentes distribuciones de datos

Como se aprecia en la Figura 4.8, cuanto más bajo sea el porcentaje de la distribución más sube la tasa de acierto. Cuanto más variedad de datos se tiene mejores resultados se obtienen, es completamente lógico.

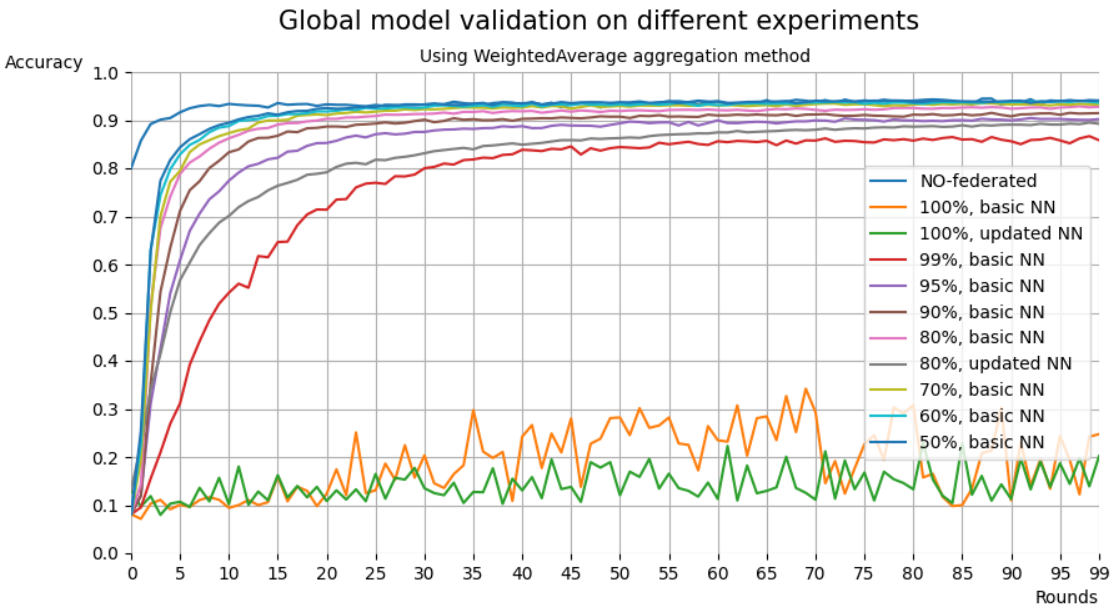

Figura 4.8: La tasa de acierto del modelo global en diferentes distribuciones de datos usando *WeightedAverage*.

Los resultados de la distribucion *100 %* son muy malos, algo que era de esperar, ya que cada colaborador solamente ha entrenado una clase. En cambio, los resultados mejoran drásticamente solamente con que cada colaborador entrene un 1 % de otras clases, llegando a una tasa de acierto del 85 % con la distribución *99 %*. Visto esto, está claro que para que el entrenamiento tenga sentido es necesario que cada colaborador cuente con un mínimo de muestras por cada clase presente en el *dataset* entero.

Con la distribución *95 %* se consigue una tasa de acierto del 90 %, y bajando el porcentaje de la distribución se logra llegar a alrededor del 93-94 %. En comparación, el modelo no-federado llega al 94 %. Viendo estos resultados, se puede concluir que con que cada colaborador entrene un 5 % de otras clases ya es suficiente para obtener resultados optimos.

También es interesante ver que como los resultados de los modelos entrenados con la red secundaria (*updated NN*) son bastante peores que los entrenados con la red principal (*basic NN*). Por ejemplo, la distribución *80 %* con la red secundaria da peores resultados que las distribuciones *95 %* y *90 %* con la red principal, cuando debería dar mejores por ser un porcentaje más bajo. A pesar de haber utilizado varias técnicas de regularización con la intención de mejorar la red, los resultados han demostrado no haber funcionado en ningún experimento.

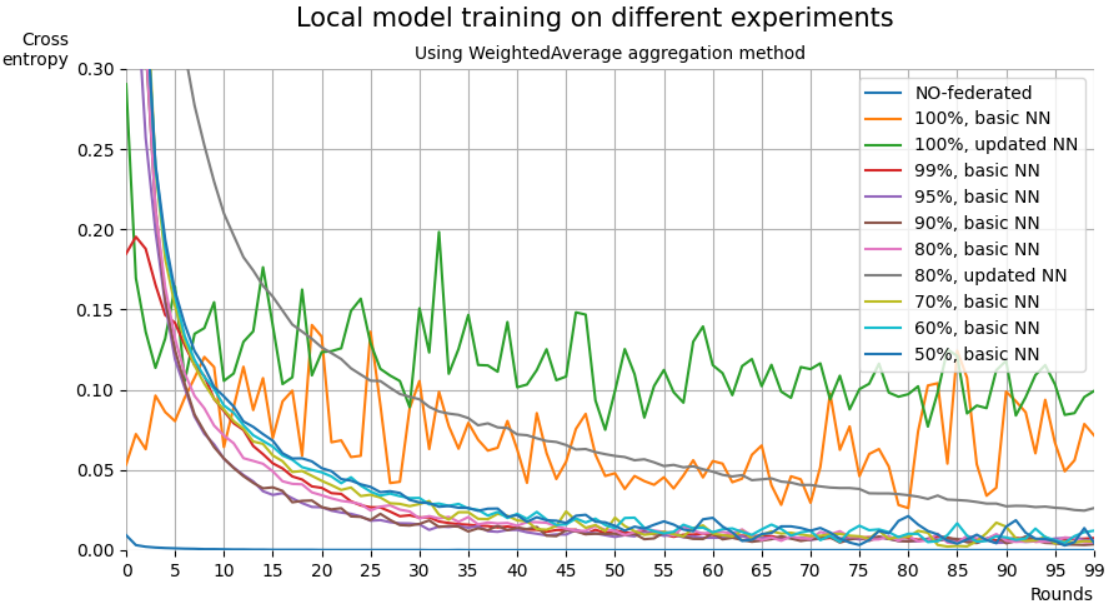

Figura 4.9: El coste del entrenamiento en diferentes distribuciones de datos usando *WeightedAverage*.

En lo que se refiere al coste del entrenamiento, como se puede ver en la Figura 4.9, el coste sube cuanto más bajo sea el porcentaje de la distribución, es decir, cuanto más variedad de datos se tiene. También cabe destacar que la red secundaria tiene un coste de entrenamiento bastante mayor que la red principal. Por otro lado, el coste de las distribuciones *100 %* sube y baja sin sentido, cosa que ocurre porque el modelo no está aprendiendo nada.

4.6.2. Modelos federados con diferentes algoritmos de agregación

Comparando los algoritmos de agregación, en la Figura 4.11, Figura 4.12, Figura 4.13 y Figura 4.14 se puede ver que *Median* y *GeometricMedian* son los peores, mientras que el resto dan resultados bastante parecidos.

Entre el resto, el *MiraiedAveraging* converge muy rápido, siendo muchas veces el más rápido de todos. Sin embargo, por desgracia, su tasa de acierto queda un poco por debajo del resto. Por otra parte, curiosamente, *ClippedAveraging* y *ClippedAveragingFeTS* difieren bastante en las primeras 30-40 rondas, cuando en teoría son el mismo algoritmo implementado de dos maneras diferentes.

De la misma manera, *SimpleAverage* y *WeightedAverage* también difieren un poquito entre sí. Dado que cada colaborador entrena la misma cantidad de muestras, todos los colaboradores tienen la misma importancia, y por lo tanto teóricamente el promedio ponderado es lo mismo que el promedio sencillo. Sin embargo, en la practica los resultados difieren. Esto ocurre porque los calculos se hacen de diferente manera y generan una pequeña diferencia en los decimales, como se puede ver en la Figura 4.10. Esta pequeña diferencia va creciendo poco a poco ronda tras ronda.

```
import numpy as np
tensors = np.array([1.2, 2.8, 3.4, 4.1, 5.9])
weights = np.array([0.1, 0.1, 0.1, 0.1, 0.1])
noweights = np.array([1, 1, 1, 1, 1]) #weights=None
avg = sum(tensors * weights) / sum(weights)
avg2 = sum(tensors * noweights) / sum(noweights)
print(f'Weighted average: {avg}')
print(f'Simple average: {avg2}')
```

```
Weighted average: 3.48
Simple average: 3.4799999999999995
```

Figura 4.10: La pequeña diferencia que se genera al hacer los mismos cálculos en dos diferentes maneras.

Dicho esto, si hubiera que nominar un agregador como ganador, este sería *ExponentialSmoothingAveraging* seguido de muy cerca por *ConditionalThresholdAveraging*. Por otra parte, es interesante ver que la distribución *70 %* casi llega a superar el modelo no-federado, mientras que la distribución *50 %* sí llega a superarlo.

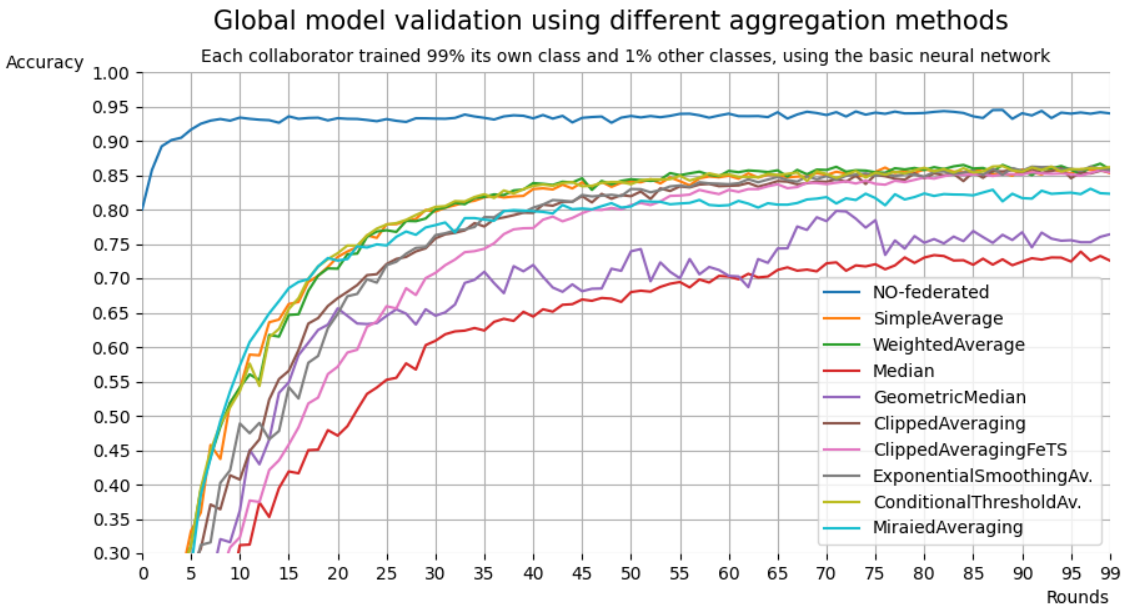

Figura 4.11: La tasa de acierto del modelo global usando diferentes agregaciones en la distribución *99 %*.

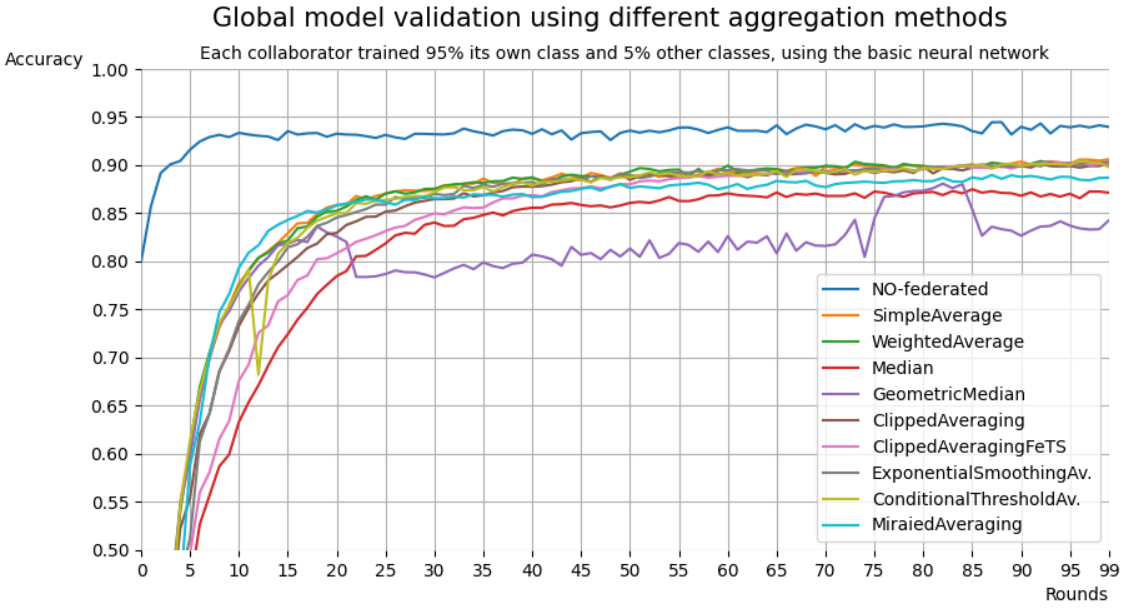

Figura 4.12: La tasa de acierto del modelo global usando diferentes agregaciones en la distribución *95 %*.

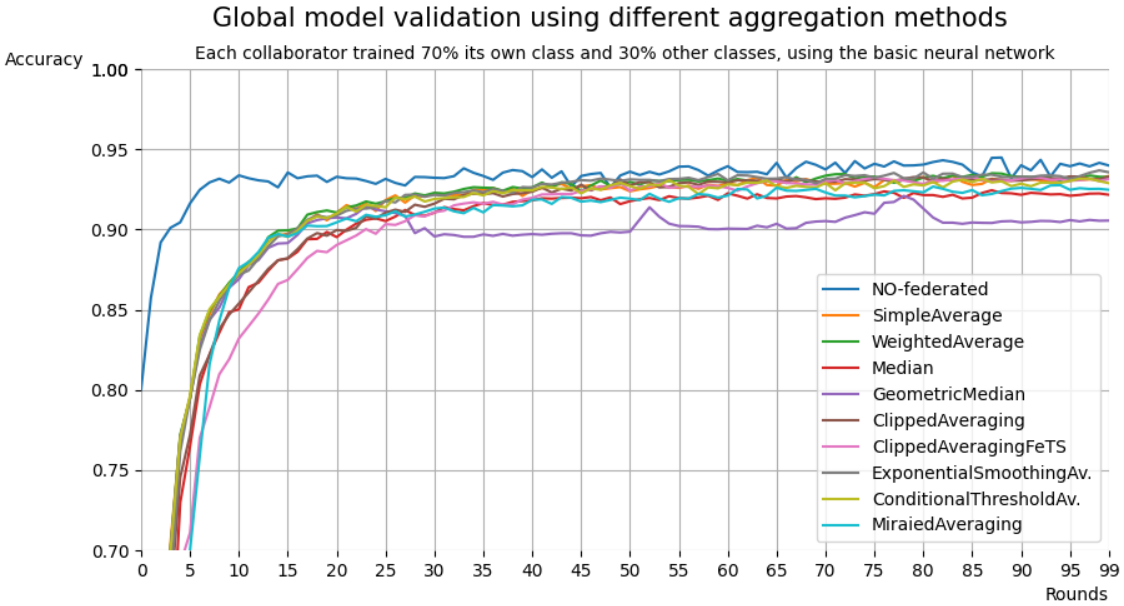

Figura 4.13: La tasa de acierto del modelo global usando diferentes agregaciones en la distribución *70 %*.

Viendo que en la Figura 4.14 algunos algoritmos llegan incluso a superar la tasa de acierto del modelo no-federado, se puede concluir que el aprendizaje federado es tan óptimo como el aprendizaje tradicional cuando al menos la mitad de las muestras de datos que cada colaborador entrena son variadas.

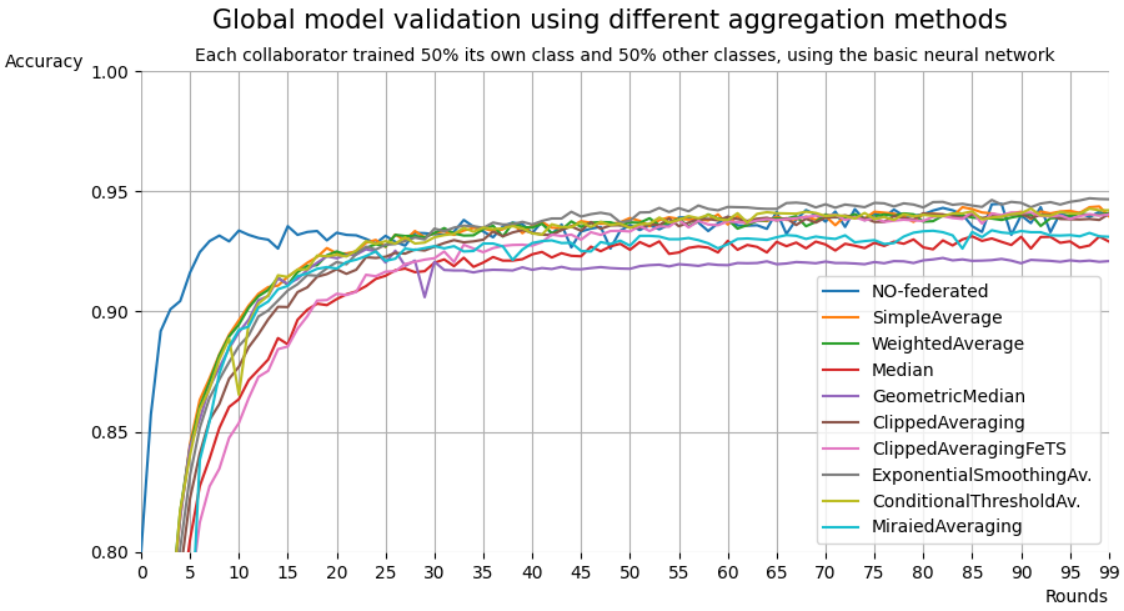

Figura 4.14: La tasa de acierto del modelo global usando diferentes agregaciones en la distribución *50 %*.

En cuanto al coste del entrenamiento, como se puede ver en la Figura 4.15, Figura 4.16 y Figura 4.17, los agregadores con más coste han sido *Median* y *GeometricMedian*, justo los mismos que han obtenido las peores tasas de acierto. Por otro lado, el agregador con menos coste ha sido *MiraiedAveraging* con bastante diferencia.

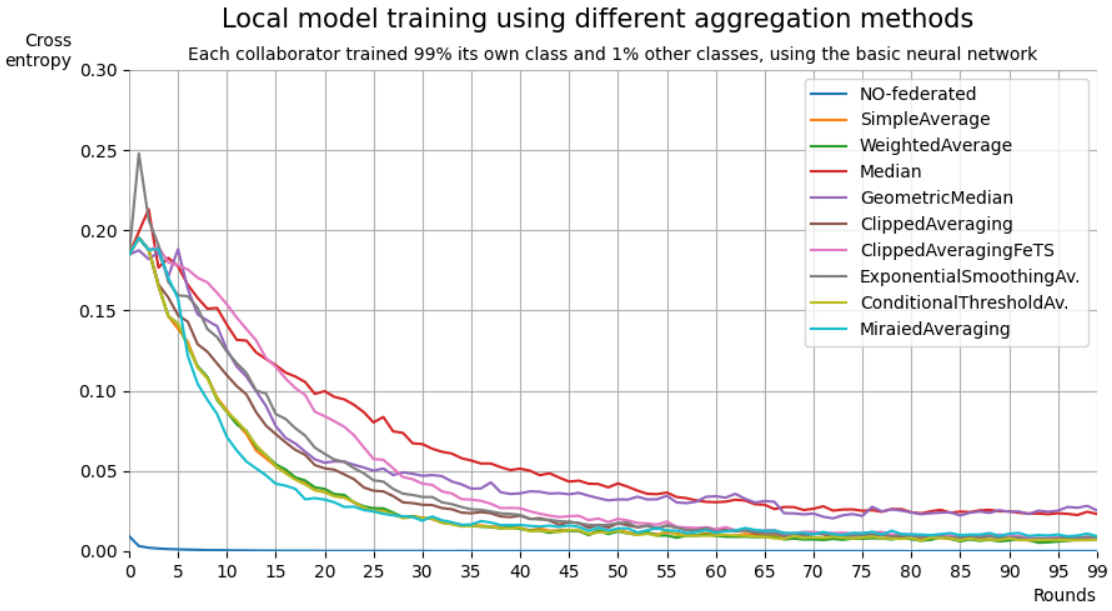

Figura 4.15: El coste del entrenamiento usando diferentes agregaciones en la distribución *99 %*.

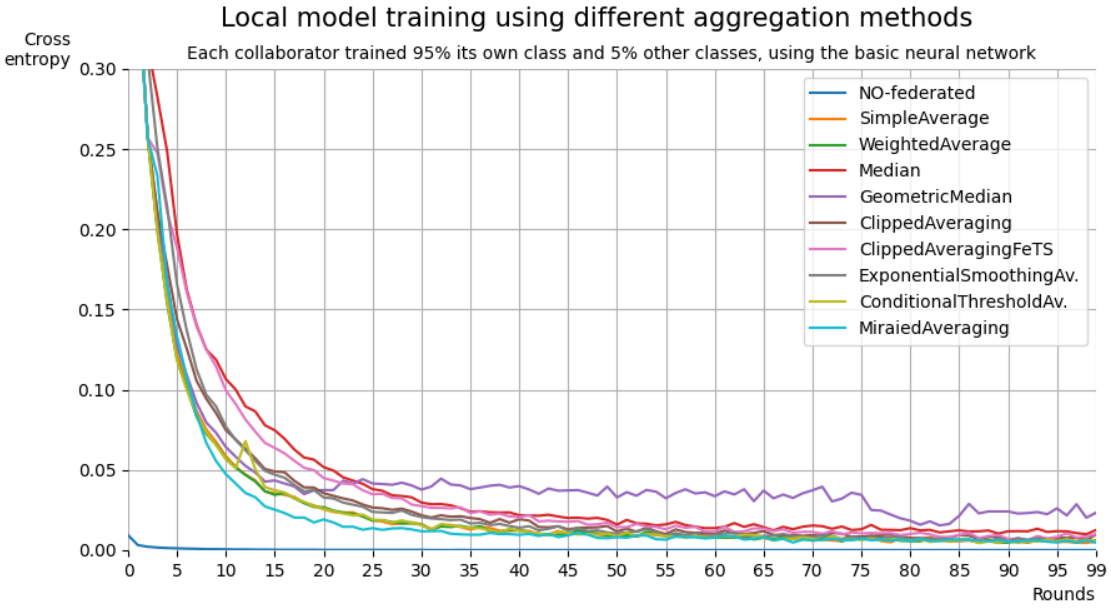

Figura 4.16: El coste del entrenamiento usando diferentes agregaciones en la distribución *95 %*.

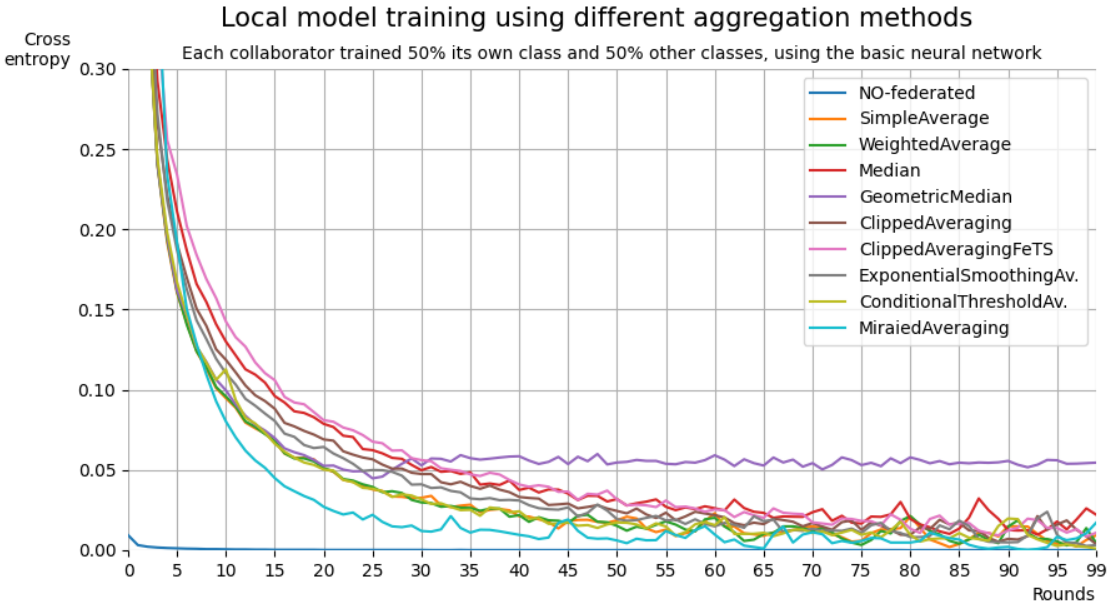

Figura 4.17: El coste del entrenamiento usando diferentes agregaciones en la distribución *50 %*.

Viendo todos estos resultados, se puede concluir que en general el mejor algoritmo de agregación ha sido *MiraiedAveraging*, ya que aunque su tasa de acierto no sea la mejor de todas, converge el más rápido y es el que menos cuesta entrenar entre todos los modelos federados.

4.6.3. Conclusiones

Resumiendo todo lo anterior, estas son las conclusiones a las que se ha llegado:

- Cada colaborador tiene que tener un mínimo de muestras por cada clase presente en el conjunto de datos, de lo contrario el entrenamiento no tiene sentido.
- La red neuronal que se utiliza para el entrenamiento afecta bastante en los resultados, como se ha visto en las diferencias entre la *basic NN* y la *updated NN*.
- El aprendizaje federado es igual de óptimo que el aprendizaje tradicional cuando al menos la mitad de las muestras de datos que cada colaborador entrena son variadas.
- Entre los algoritmos de agregación que se han utilizado, el más óptimo es el *MiraiedAveraging*. A pesar de que su tasa de acierto no es la mejor de todas, converge el más rápido y es el que menos cuesta entrenar.

5. CAPÍTULO

Caso de uso II — Detección federada de tumores cerebrales (FeTS Challenge)

Durante las primeras fases del TFM conocimos esta competición sobre el aprendizaje federado en la que decidimos participar debido a su alto encaje con el objetivo del proyecto, aprovechando así la oportunidad de aplicar lo aprendido en un entorno de datos reales.

5.1. Descripción

El *Federated Tumor Segmentation Challenge 2021*, abreviado como *FeTS Challenge* o *FeTS 2021*, es la primera competición computacional sobre el aprendizaje federado. [48]

El *FeTS Challenge* se centra en el aprendizaje federado y en la solidez de los cambios de distribución entre instituciones médicas para la tarea de segmentación de tumores cerebrales. Este desafío tiene como base el *BraTS Challenge*, un desafío que se lleva haciendo anualmente desde 2012. [49]

Los desafíos internacionales se han convertido en el estándar para la validación de los métodos de análisis de imágenes biomédicas. Sin embargo, desde *FeTS 2021* sostienen que el rendimiento real de incluso los algoritmos ganadores en datos clínicos del ”mundo real” a menudo no está claro, ya que los datos incluidos en estos desafíos generalmente se adquieren en entornos muy controlados y en pocas instituciones. La solución aparentemente obvia de simplemente recopilar cada vez más datos de instituciones geográficamente más distintas en tales desafíos no se escala bien debido a la privacidad, la propiedad y los obstáculos técnicos. [50]

El *FeTS Challenge* es el primer desafío que se ha propuesto para el aprendizaje federado en medicina, y tiene la intención de abordar los obstáculos recién mencionados, tanto para la creación como para la evaluación de modelos de segmentación tumoral. En concreto, el *FeTS 2021* utiliza escáneres de resonancia magnética multiinstitucionales adquiridos clínicamente del *BraTS 2020*, así como de varias instituciones independientes remotas incluidas en la red colaborativa de una federación del mundo real, la federación *fets.ai*. [24]

El desafío está construido sobre el *framework* OpenFL, una infraestructura de código abierto de Intel preparada para realizar el entrenamiento de un modelo sobre la paradigma del aprendizaje federado. [23]

El *FeTS Challenge* se centra en la construcción y evaluación de un modelo de consenso para la segmentación de tumores cerebrales intrínsecamente heterogéneos (en apariencia, forma e histología) conocidos como gliomas. [51] En comparación con el *BraTS Challenge*, el objetivo final del *FeTS Challenge* es la creación de un modelo de segmentación de consenso que haya obtenido conocimiento de los datos de múltiples instituciones sin agrupar sus datos (es decir, reteniendo los datos dentro de cada institución), y la evaluación de modelos de segmentación en dicha configuración federada (es decir, en la naturaleza). [52] El *challenge* se estructura en dos tareas explícitas:

- *Task 1 - Federated training (FL Weight Aggregation Methods)* $\longrightarrow$ Entrenamiento federado: dado un algoritmo de segmentación predefinido para el entrenamiento, tiene como objetivo definir métodos efectivos de agregación de pesos para la creación de un modelo de consenso, mientras que (opcionalmente) también tiene en cuenta las interrupciones de la red.
- *Task 2 - Federated evaluation (Generalization "In The Wild")* $\longrightarrow$ Evaluación federada: tiene como objetivo crear algoritmos de segmentación robustos, evaluados durante la fase de prueba en conjuntos de datos invisibles de varias instituciones independientes remotas de la red colaborativa de la federación *fets.ai*.

Los participantes son libres de escoger que tareas quieren realizar. Personalmente, yo solo he participado en el *Task 1*, así que solo explicaré las partes del *challenge* que he realizado.

El enfoque específico de la tarea *Task 1* es identificar la mejor manera de agregar el conocimiento proveniente de modelos de segmentación entrenados en las instituciones individuales, en lugar de identificar el mejor método de segmentación. Más específicamente, la atención se centra en las partes metodológicas específicas del aprendizaje federado (en la agregación, en la selección de colaboradores, en el entrenamiento por ronda...) y no en el desarrollo de algoritmos de segmentación (en lo que se centra el *BraTS Challenge*). [50]

Para realizar dicha tarea se exigen unos requisitos de sistema:

- Git.
- Python con sistema de gestión de entorno virtual (se recomienda utilizar Anaconda).
- Solo Windows: el paquete Pickle5 requiere al menos Microsoft C++ 14.0 (C++ Build Tools).

Con el fin de facilitar la tarea, se proporciona a todos los participantes una infraestructura existente para la segmentación federada de tumores que utiliza el promedio federado (*federated averaging*) como método de agregación. Esta infraestructura también indica los lugares exactos donde los participantes tienen permitido y se espera que realicen cambios. La infraestructura se puede encontrar en GitHub. [53]

Estas son las partes del código en las que se espera que los participantes realicen cambios:

- La función de agregación para fusionar los modelos actualizados que los colaboradores han entrenado en cada ronda federada.
- Qué colaboradores se eligen para entrenar en cada ronda federada.
- Los parámetros de entrenamiento para cada ronda federada.
- Las métricas de validación que se calcularán en cada ronda (y que después también pueden ser utilizadas como *input* para las otras funciones).

El objetivo principal de esta tarea consiste en la agregación de modelos locales de segmentación dada la partición de los datos de acuerdo a su distribución en el mundo real.

5.2. Dataset

El conjunto de datos utilizado en el este desafío (FeTS 2021) es un subconjunto del desafío del año pasado del *Brain Tumor Segmentation* (BraTS 2020).

Los datos que tenemos en el conjunto son imágenes multiparamétricas por resonancia magnética (*mpMRI: multi-parametric magnetic resonance imaging*) que radiográficamente parecen tener glioblastoma (GBM), un tumor cerebral que produce cáncer. Dichas imágenes son escaneos amplios, multiinstitucionales, rutinarios, clínicamente adquiridos, preoperatorios y estándares. [54]

Los escaneos están disponibles en archivos NIfTI (*Neuroimaging Informatics Technology Initiative*) con extensión *.nii.gz*, y son de 4 tipos diferentes. Cada dato de paciente cuenta con un escaneo de cada tipo:

- _t1.nii.gz (T1) ⟶ Imagen nativa ponderada en T1[1].
- _t1ce.nii.gz (T1CE) ⟶ Imagen realzada en poscontraste de gadolinio (Gd) ponderada en T1.
- _t2.nii.gz (T2) ⟶ Imagen nativa ponderada en T2[2].
- _flair.nii.gz (FLAIR) ⟶ Imagen de recuperación de inversión atenuada por líquido ponderada en T2[3].

También tenemos un archivo más por cada paciente, el cual para los participantes solo es visible en la fase de entrenamiento:

- _seg.nii.gz ⟶ Imagen segmentada (*ground truth*).

Los escaneos han sido manualmente segmentados por 1-4 evaluadores. Las imágenes segmentadas son anotaciones (*ground truth*) que han sido aprobadas por neurorradiólogos expertos certificados. Las anotaciones constan de 3 etiquetas, y cada etiqueta indica una subregión tumoral:

- *Label 1: NCR* ⟶ Núcleo del tumor necrótico.
- *Label 2: ED* ⟶ Tejido peritumoral edematoso/invadido.
- *Label 4: ET / AT*[4] ⟶ Tumor realzado en poscontraste de gadolinio (Gd).

Estas anotaciones fueron creadas por múltiples expertos, y aunque a cada institución que aportó datos se le proporcionó un protocolo de anotación muy específico, se observaron estilos de anotación ligeramente diferentes para varios evaluadores involucrados en el proceso. Por lo tanto, todas las etiquetas finales incluidas en el conjunto de datos de BraTS también fueron revisados más a fondo para verificar la coherencia y el cumplimiento del protocolo de anotación por un solo neurorradiólogo certificado por la junta con más de 15 años de experiencia. [57]

[1] El T1 de un tejido es el tiempo que tarda en recuperar el 63 % de la magnetización longitudinal. [55] Las imágenes ponderadas en T1 resaltan la grasa del tejido. [56]

[2] El T2 de un tejido es el tiempo que emplea en perder el 63 % de su magnetización transversal. [55] Las imágenes ponderadas en T2 resaltan la grasa y el agua del tejido. [56]

[3] T2-FLAIR: *T2 Fluid-Attenuated Inversion Recovery*

[4] En el paper del *BraTS Challenge* lo llaman "*Active tumor (AT)*", mientras que en la página web del *FeTS Challenge* (y en el código) lo llaman "*Enhancing tumor (ET)*". Las dos nomenclaturas se refieren a lo mismo. [57]

Las ediciones anteriores al BraTS 2017 contaban con otra etiqueta más (*Label 3: NET*), pero esta etiqueta fue eliminada debido a que presentaba una parcialidad en los resultados. [57] Por lo tanto, a partir de 2017 el desafío consta de las siguientes subregiones:

- *Enhancing tumor (ET)* $\longrightarrow$ Tumor realzado: define las regiones de tumor activo, y en base a esto, la extensión de la resección quirúrgica (equivalente a la etiqueta *label 4*).
- *Tumor core (TC)* $\longrightarrow$ Núcleo del tumor: define lo que normalmente se reseca durante un procedimiento quirúrgico (unión de las etiquetas *label 1* y *label 4*).
- *Whole tumor (WT)* $\longrightarrow$ Tumor entero: define la extensión total del tumor, incluido el tejido edematoso peritumoral y el área altamente invadida (unión de las etiquetas *label 1*, *label 2* y *label 4*).

Para poder imaginarme mejor con que datos estoy trabajando, he decidido visualizarlos mediante una aplicación llamada Mango (*Multi-image Analysis GUI*) [58], un visor de imágenes de investigación médica que proporciona herramientas de análisis y una interfaz de usuario para navegar por los volúmenes de las imágenes como la que se ve en la Figura 5.1.

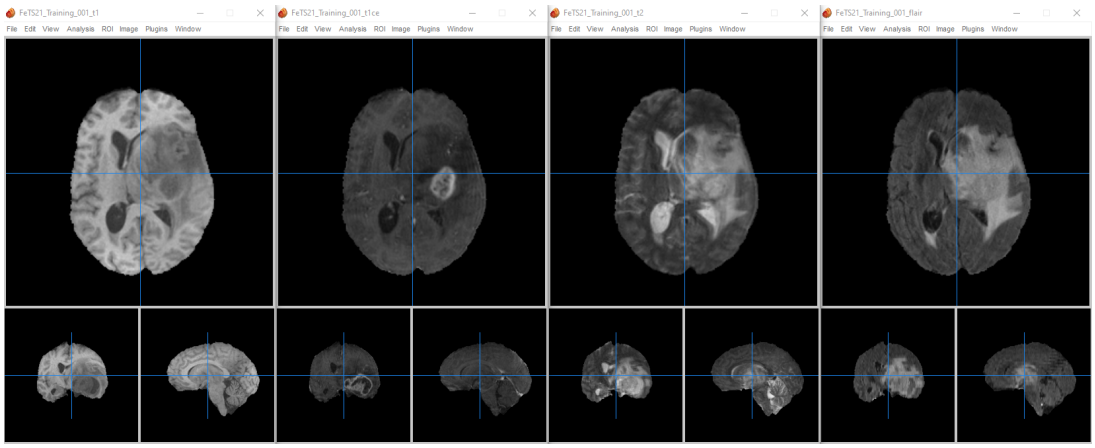

Figura 5.1: Los diferentes escaneos del cerebro de un paciente (archivo *FeTS21_Training_001.nii.gz*) visualizados mediante la aplicación Mango. Las cuatro ventanas, de izquierda a derecha, pertenecen a los escaneos T1, T1CE, T2 y FLAIR.

Los datos de imágenes están acompañados por un archivo de valores separados por comas (.csv), el cual incluye información sobre la partición de datos de acuerdo con el origen de adquisición para cada uno de los datos pseudoidentificados, para de esa forma facilitar aún más la investigación sobre el aprendizaje federado. [54] Cada partición contará como un colaborador para realizar el entrenamiento, mientras que cada paciente será una muestra de datos. Al principio se han ofrecido dos estrategias para particionar los datos:

- *partitioning_1.csv* $\longrightarrow$ Particionamiento natural: cada *Partition_ID* representa una institución de origen, mientras que cada *Subject_ID* se refiere a los registros de un paciente.

  $\Rightarrow$ 17 particiones y 341 pacientes.

- *partitioning_2.csv* ⟶ Particionamiento artificial: lo mismo que el anterior, pero después de dividir en más particiones cada una de las 5 instituciones más grandes de acuerdo a la información extraída de las imágenes (en concreto, se midió el tamaño del "tumor entero" (*WT: whole tumor*) para todos los registros y se utilizó la mediana como umbral para crear las particiones adicionales).

  ⇒ 22 particiones y 341 pacientes.

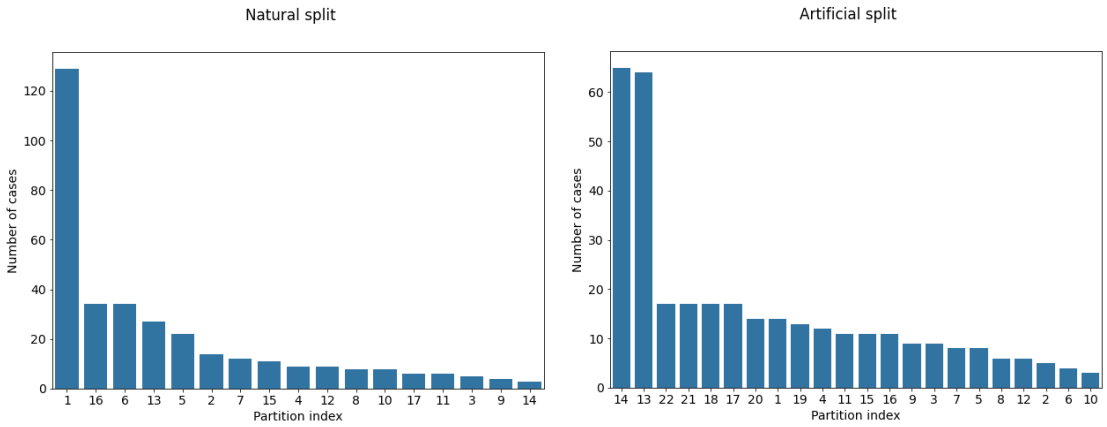

Figura 5.2: El número de pacientes (*number of cases*) por cada partición del conjunto de entrenamiento. Particionamiento natural (*partitioning_1.csv*) a la izquierda, y particionamiento artificial (*partitioning_2.csv*) a la derecha.

Las distribuciones de dichas particiones se muestran gráficamente en la Figura 5.2. Adicionalmente, conforme iba avanzando el *challenge*, se ofreció un particionamiento más pequeño para que los participantes pudiesen hacer ejecuciones mas rápidas:

- *small_split.csv* ⟶ Particionamiento pequeño: una muestra pequeña del particionamiento natural.

  ⇒ 3 particiones y 12 pacientes.

También se ha ofrecido la posibilidad de que los participantes creasen y explorasen más estrategias de particionamiento, haciendo uso de cualquier información que considerasen relevante para contribuir a la generalización del método propuesto. [54] En mi caso, por limitaciones de *hardware*, también he creado un particionamiento más pequeño basado en el *small_split.csv*:

- *mysplit.csv* ⟶ Mi propio particionamiento: una muestra aún más pequeña del particionamiento natural.

  ⇒ 3 particiones y 6 pacientes.

Aparte de los particionamientos ya mencionados, existe otro más que los organizadores han utilizado durante la fase de testeo, y que por lo tanto no se encuentra al alcance de los participantes. [53]

5.3. Métricas de evaluación

El *challenge* utiliza seis[5] métricas específicas para evaluar el rendimiento de los modelos de entrenamiento. A pesar de ello, las métricas que se tienen en cuenta para elegir la persona ganadora de la competición solo son las primeras tres. [50] Estas son las métricas en cuestión:

- ***Dice Similarity Coefficient*** (valor entre 0 y 1, siendo 1 lo mejor) ⟶ Coeficiente de similitud de Dice: la media armónica entre la precisión y la exhaustividad (*recall*), métrica que también se conoce como F1-Score o coeficiente de Sorensen-Dice. [59] Se utiliza para evaluar el rendimiento de la segmentación. Se calcula para cada subregión (*enhancing tumor*, *tumor core* y *whole tumor*).
- ***95% Hausdorff Distance*** (valor entre ∞ y 0, siendo 0 lo mejor) ⟶ Percentil 95 de la distancia de Hausdorff: distancia máxima posible entre dos subconjuntos, en este caso entre la región de tumores y la región de no tumores. [60] El percentil 95 significa que la distancia será solamente más larga que el 95 % de todas las distancias posibles entre las dos regiones. Se utiliza para evitar que los valores atípicos tengan demasiado peso. Se calcula para cada subregión (*enhancing tumor*, *tumor core* y *whole tumor*).
- ***Simulated Time*** (valor entre 0 y ∞, siendo 0 lo mejor) ⟶ Tiempo simulado: la simulación del tiempo que se está tardando en ejecutar todo el algoritmo. Se computa por colaborador, siendo el resultado de cada ronda el tiempo del colaborador que más haya tardado. El tiempo de cada colaborador se computa mediante la suma de los siguientes valores:
  - Tiempo que se tarda en descargar del servidor el modelo compartido.
  - Tiempo que se tarda en validar el modelo compartido.
  - Tiempo que se tarda en entrenar el modelo.
  - Tiempo que se tarda en validar el modelo entrenado.
  - Tiempo que se tarda en subir al servidor el modelo actualizado.

  Para que la competición sea justa y no favorezca a los que tengan más potencia computacional, todos los tiempos están previamente definidos en base a la información real de un subconjunto de hospitales. [61]
- ***Sensitivity*** (valor entre 0 y 1, siendo 1 lo mejor) ⟶ Sensibilidad: proporción de regiones de tumor correctamente identificados, métrica que también se conoce como tasa de verdaderos positivos (TPR[6]).
- ***Specificity*** (valor entre 0 y 1, siendo 1 lo mejor) ⟶ Especificidad: proporción de regiones sin tumor correctamente identificados, métrica que también se conoce como tasa de verdaderos negativos (TNR[6]).
- ***Convergence Score*** (valor entre 0 y 1, siendo 1 lo mejor) ⟶ Calificación de convergencia: valor que indica que tal se está convergiendo el algoritmo. Se computa calculando la media de los tres coeficientes de *Dice* a lo largo del tiempo simulado.

[5] La descripción del *challenge* solo indica cinco [50], pero el algoritmo predefinido computa una métrica más: el *Convergence Score*.
[6] TPR (*true positive rate*) y TNR (*true negative rate*). [62]

5.4. Red neuronal

La red que se ha usado para la tarea ha sido proporcionada por el propio *challenge*. Es una red que está formada por varios módulos de segmentación que realizan diferentes operaciones (submuestreo, codificación, decodificación y sobremuestreo), creando una arquitectura estándar de tipo U-Net [63], como la que se ve en la Figura 5.3. Las redes U-Net están basadas en las redes totalmente convolucionales (*fully convolutional networks*) que se utilizan para la segmentación de imágenes. [64] Estas redes reciben una imagen como *input* y devuelven otra imagen del mismo tamaño como *output*.

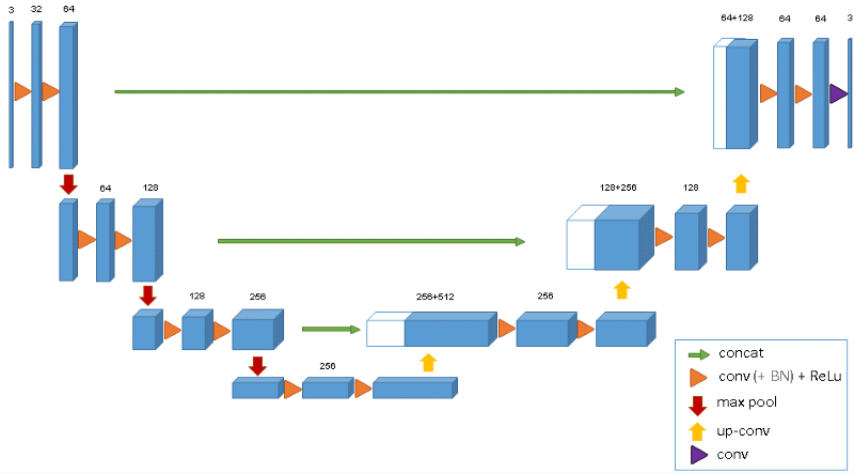

Figura 5.3: Arquitectura de una red U-Net.

La red del *challenge* realiza un proceso de codificación y un proceso de decodificación mediante convoluciones que van aprendiendo la información de los datos. Los módulos de codificación se intercalan con módulos de submuestreo, mientras que los módulos de decodificación se intercalan con módulos de sobremuestreo.

Los módulos se definen mediante unos hiperparámetros constantes más el número de filtros *input* y el número de filtros *output*. La cantidad de filtros se duplica en cada codificación, mientras que se reduce a la mitad en cada decodificación, dando a la red una forma de U.

La arquitectura U-Net permite añadir unos elementos llamados *skip connections* de una capa a otra capa simétrica, para así poder estabilizar el entrenamiento y la convergencia saltando algunas capas durante el *back-propagation*. Al actualizar los pesos mediante el *backpropagation*, los gradientes suelen ir disminuyendo por cada neurona que cruzan (el problema del *vanishing gradient*). Los *skip connections* permiten propagar gradientes más grandes a las capas iniciales, haciendo que las capas iniciales aprenden tan rápido como las capas finales, y pudiendo entrenar redes más profundas.

5.5. Entrenamiento de los modelos

En esta tarea solamente he entrenado tres diferentes modelos. Dado que por limitaciones de *hardware* ha sido imposible utilizar los particionamientos *partitioning_1.csv* y *partitioning_2.csv*, el entrenamiento final de estos modelos lo he realizado con el *small_split.csv*, mientras que para la mayoría de ejecuciones de prueba he utilizado el *mysplit.csv* para ahorrar tiempo.

5.5.1. Selección de colaboradores

Debido a que los particionamientos utilizados solo cuentan con tres colaboradores, un número que de por sí ya me parecía muy pequeño, en cada ronda federada siempre he seleccionado todos los colaboradores. También cabe mencionar que en este caso cada colaborador solamente cuenta con dos muestras de entrenamiento.

Dado que tanto el servidor como todos los colaboradores se simulan dentro de una sola maquina, los colaboradores solo realizan tres tareas: validar el modelo compartido, entrenar el modelo, y validar el modelo entrenado.

5.5.2. Hiperparámetros

El número de rondas de entrenamiento que en principio he establecido ha sido 100, aunque ninguna ejecución ha podido llegar a esa cifra principalmente por falta de suficiente memoria RAM. Al igual que en el anterior caso de uso, cada colaborador solamente hace una ronda de entrenamiento por cada ronda federada, por lo que una ronda de entrenamiento equivale a una ronda federada.

El tamaño de lote (*batch size*) está fijado a 1 por parte de los organizadores del *challenge*, y no se permite cambiarlo. Esto está hecho así para poder utilizar el descenso de gradiente estocástico, ya que para ello es necesario que el *batch size* sea de 1. Por lo tanto, dado que todas las muestras de datos se propagan por la red de una sola tirada, una ronda de entrenamiento se completa en una sola iteración de ida y vuelta.

En cambio, el ratio de aprendizaje (*learning rate*) está establecido por defecto a $5e{-}5$, a lo que viene a ser 0.00005. A pesar de que se permite cambiarlo, lo he mantenido con dicho valor.

Aparte, al igual que en el anterior caso de uso, la cantidad de rondas que se guardan en la base de datos en esta tarea también la he establecido a 5.

5.5.3. Funciones de agregación

La única diferencia entre los tres diferentes modelos que he entrenado yace en el algoritmo utilizado para la agregación de tensores. La explicación de cada algoritmo se encuentra en el anterior caso de uso (apartado 4.5 de la memoria). Estos son los tres algoritmos que he utilizado en este caso de uso:

- *WeightedAverage* $\longrightarrow$ Promedio ponderado.
- *ClippedAveragingFeTS* $\longrightarrow$ Promedio recortado FeTS.
- *MiraiedAveraging* $\longrightarrow$ Promedio miraieado.

5.5.4. Distribución de datos

Dado que las muestras son datos muy grandes, por limitaciones de *hardware*, las distribuciones han tenido que ser muy pequeñas. En el particionamiento *small_split.csv* el primer colaborador contiene 4 muestras, el segundo 5 muestras, y el tercero 3 muestras, haciendo que la importancia de cada uno sea 0.333, 0.444 y 0.222 respectivamente. En cambio, en el particionamiento *mysplit.csv* los tres contienen 2 muestras, haciendo que todos tengan la misma importancia.

5.5.5. Elementos añadidos

Aparte de las métricas predefinidas por los desarrolladores del *challenge*, personalmente he añadido la métrica ***Accuracy*** (valor entre 0 y 1, siendo 1 lo mejor) que indica la precisión del modelo para predecir el resultado. Cabe mencionar que excepto *Simulated Time* y *Convergence Score*, todas las métricas se calculan dentro de cada colaborador, y que las métricas que el servidor devuelve al final de cada ronda solamente son la media de todos los colaboradores. Es decir, el modelo que se agrega al final de la ronda número 3 se evalúa con las métricas de la ronda número 4.

Otra cosa que también he añadido ha sido una semilla (*seed*) fija para que las ejecuciones de un mismo modelo en unas mismas condiciones siempre diesen los mismos resultados. Más adelante, esta medida también ha sido adoptada por los desarrolladores del *challenge* en una actualización posterior.

5.6. Resultados

En este apartado se muestran los resultados obtenidos entrenando los tres modelos diferentes. En este caso, los tres modelos han sido entrenados sobre el particionamiento *small_split.csv*.

5.6.1. Modelo federado con WeightedAverage

En el caso del modelo entrenado con el algoritmo de agregación *WeightedAverage* se han conseguido los siguientes resultados que se muestran en la tabla de la Figura 5.4.

REAL TIME	ROUND	Simulation Time	Convergence Score	Sensitivity	Specificity	Accuracy
START [09:50]	WeightedAv.	Seed(0)				
[10:21]	ROUND 0	13.71 minutes	0.006817776	0.003340537	0.998794377	0.750832617
[10:50]	ROUND 1	26.39 minutes	0.009100882	0.004091504	0.998960435	0.758599997
[11:18]	ROUND 2	36.03 minutes	0.012762877	0.004842311	0.998894155	0.813504159
[11:51]	ROUND 3	47.72 minutes	0.020071104	0.006449182	0.998878539	0.871637642
[12:27]	ROUND 4	58.77 minutes	0.027422526	0.007885248	0.998819053	0.910286486
[13:00]	ROUND 5	70.4 minutes	0.132879516	0.023555582	0.999121606	0.934399307
[13:31]	ROUND 6	79.42 minutes	0.267599518	0.076424807	0.999628544	0.948547304
[14:00]	ROUND 7	90.63 minutes	0.283819251	0.111199521	0.999700248	0.953692913
[14:29]	ROUND 8	99.95 minutes	0.303730604	0.150667682	0.999858618	0.957554400
[14:59]	ROUND 9	110.99 minutes	0.336573807	0.215407237	0.999891281	0.962711334
[15:32]	ROUND 10	123.39 minutes	0.336573807	0.202016428	0.999919593	0.964471757
[16:07]	ROUND 11	135.29 minutes	0.383698469	0.271710098	0.999809563	0.968108118
[16:43]	ROUND 12	147.5 minutes	0.383698469	0.249884531	0.999829233	0.964555979
[17:13]	ROUND 13	159.92 minutes	0.408090938	0.311874360	0.999759734	0.972584546
[17:41]	ROUND 14	169.97 minutes	0.408090938	0.314126343	0.999804020	0.970972240
[18:11]	ROUND 15	181.48 minutes	0.448075105	0.350388199	0.999686420	0.979646862
[18:46]	ROUND 16	191.77 minutes	0.448075105	0.332153469	0.999755681	0.981465340
...	...	...	...	...	...	...
[09:29]	ROUND 44	510.57 minutes	0.743561315	0.565209687	0.999543011	0.998658657
[10:01]	ROUND 45	520.81 minutes	0.747552314	0.583566904	0.999558032	0.998737156
[10:31]	ROUND 46	532.78 minutes	0.759402085	0.606500924	0.999651015	0.998851240
[11:00]	ROUND 47	542.97 minutes	0.775427937	0.585270047	0.999724865	0.998828113
[11:30]	ROUND 48	553.9 minutes	0.775427937	0.592570066	0.999712884	0.998892009
[12:00]	ROUND 49	563.66 minutes	0.775427937	0.613759339	0.999637783	0.998886406
[12:33]	ROUND 50	575.83 minutes	0.775427937	0.607008755	0.999612272	0.998892128
[12:46]	ROUND 51	OUT OF RAM				

Figura 5.4: Resumen de los resultados del modelo entrenado con *WeightedAverage* sobre *small_split.csv*.

El modelo converge bastante rápido, y su precisión sube del 75 % al 90-95 % en las primeras pocas rondas. En las siguientes rondas estos valores siguen subiendo, pero cada vez más despacio, llegando al 99 % al cabo de unas 20 rondas. A partir de ahí los resultado apenas cambian, y al llegar a la ronda número 51 la ejecución se corta por falta de suficiente memoria RAM.

Por cada ronda, el tiempo simulado de cada colaborador es 2 minutos, 4-5 minutos y 10-13 minutos respectivamente. Para el computo del tiempo simulado solamente se tiene en cuenta el colaborador que más tarda, dado que en una situación real estos se estarían ejecutando en paralelo.

En la primera columna de la tabla se puede ver que en realidad cada ronda tarda más o menos 30 minutos, mientras que la ejecución de todos los colaboradores en serie debería ser solamente unos 15-20 minutos. Es más, debería ser bastante menos de eso, ya que OpenFL está simulando el servidor dentro de la propia maquina y no necesita ni subir ni bajar el modelo. Esta diferencia tan grande entre el tiempo real y el tiempo simulado ocurre por limitaciones de *hardware*.

Mientras que en la Figura 5.4 se han omitido algunas métricas para que la tabla se ajustase a la estructura de la hoja, la Figura 5.5 y la Figura 5.6 muestran gráficamente la evolución de todas las métricas de evaluación a medida que avanzan las rondas.

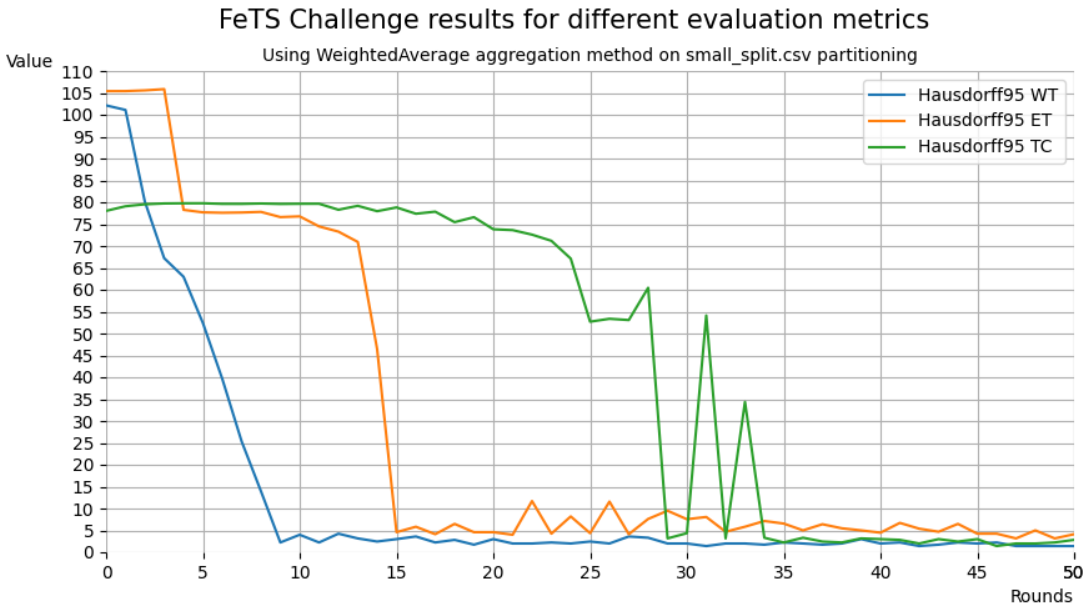

Figura 5.5: Los valores de las distancias de Hausdorff del modelo entrenado con *WeightedAverage* sobre *small_split.csv*.

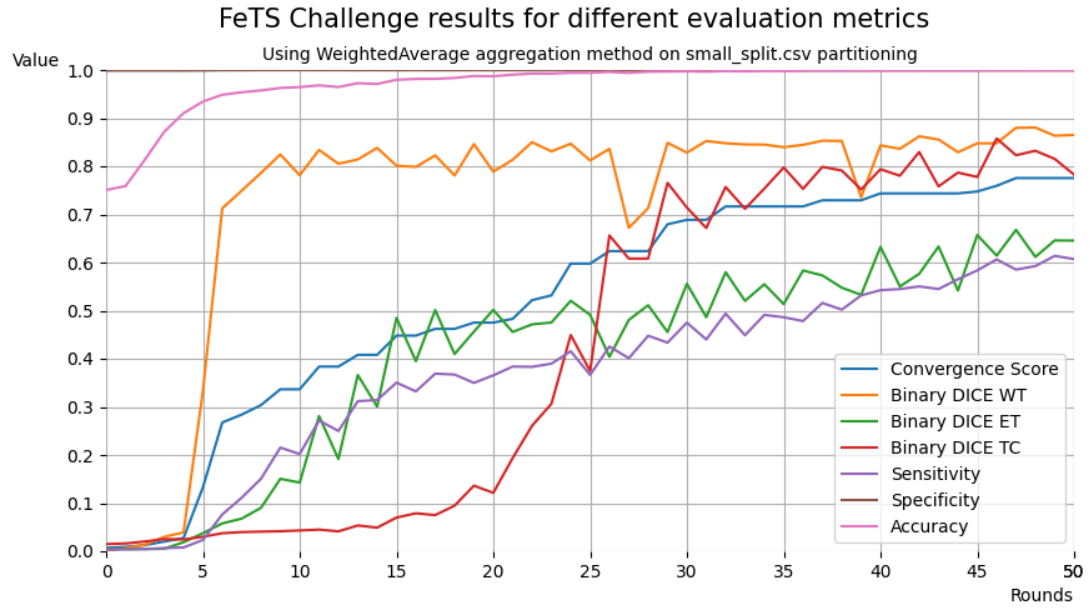

Figura 5.6: Los valores del resto de las métricas del modelo entrenado con *WeightedAverage* sobre *small_split.csv*.

En la Figura 5.5 primera se puede ver como la distancia de Hausdorff disminuye mucho en las tres subregiones, lo cual significa que el modelo ha aprendido a no tener en cuenta los valores atípicos, cosa que es buena. En las subregiones del tumor total (*whole tumor*) y del tumor realzado (*enhancing tumor*) disminuye muy rápido, mientras que en la subregion del núcleo del tumor (*tumor core*) tarda un poquito más.

En cambio, en la Figura 5.6 se ve como el coeficiente de Dice incrementa en las tres subregiones, en cada una más o menos al mismo ritmo que su respectiva distancia de Hausdorff disminuye en la figura anterior. El coeficiente de la subregión del tumor realzado no llega a incrementar tanto como el resto, al igual que su respectiva distancia de Hausdorff disminuye un poco menos que el resto de distancias en la figura anterior.

Por otro lado, la precisión empieza en un valor bastante alto subiendo enseguida a valores máximos, y la especificidad empieza en un valor máximo desde el principio, mientras que la sensibilidad va subiendo poco a poco, bastante acorde con la calificación de convergencia.

5.6.2. Modelo federado con ClippedAveragingFeTS

En lo que se refiere al modelo entrenando con el algoritmo de agregación *ClippedAveragingFeTS*, como se puede ver en la Figura 5.7, no se ha podido realizar más de dos rondas debido a un error en la ejecución.

REAL TIME	ROUND	Simulation Time	Convergence Score	Sensitivity	Specificity	Accuracy
START [18:25]	ClippedAv.	Seed(0)				
[18:57]	ROUND 0	13.71 minutes	0.006817776	0.003340537	0.998794377	0.750832617
[19:34]	ROUND 1	26.39 minutes	0.009100882	0.004091504	0.998960435	0.758599997
[20:01]	ROUND 2	DELTA ERROR				

Figura 5.7: Resumen de los resultados del modelo entrenado con *ClippedAveragingFeTS* sobre *small_split.csv*.

Los resultados de las primeras dos rondas son iguales a los obtenidos con el algoritmo anterior. Esto se debe a que en esas rondas el valor de los deltas es cero, ya que para calcularlos se utiliza el tensor agregado anterior a la ronda anterior, es decir, el tensor agregado de hace dos rondas, y este aún no existe.

De todas formas, a pesar de que ya en la ronda número 2 los deltas no serían cero, los resultados seguirían siendo los mismos que del algoritmo anterior dado que las métricas se calculan antes de hacer la fusión de los tensores. Resumiendo, los resultados empezarían a variar del algoritmo anterior a partir de la ronda número 3, pero debido a unas dificultades imprevistas no lo he podido demostrar.

5.6.3. Modelo federado con MiraiedAveraging

En el caso del modelo entrenado con el algoritmo de agregación *MiraiedAveraging*, como se puede ver en la Figura 5.8, los resultados obtenidos han vuelto a ser los mismos que en los modelos anteriores.

REAL TIME	ROUND	Simulation Time	Convergence Score	Sensitivity	Specificity	Accuracy
START [20:25]	MiraiedAv.	Seed(0)				
[20:59]	ROUND 0	13.71 minutes	0.006817776	0.003340537	0.998794377	0.750832617
[21:35]	ROUND 1	26.39 minutes	0.009100882	0.004091504	0.998960435	0.758599997
[22:13]	ROUND 2	36.03 minutes	0.012762877	0.004842311	0.998894155	0.813504159
[22:13]	ROUND 3	NAN ERROR				

Figura 5.8: Resumen de los resultados del modelo entrenado con *MiraiedAveraging* sobre *small_split.csv*.

Los resultados de las primeras tres rondas son iguales a los obtenidos con el primer algoritmo. Esto se debe a la misma razón explicada en el algoritmo anterior. Los resultados empezarían a variar a partir de la ronda número 3, pero una vez más debido a unas dificultades imprevistas no lo he podido demostrar.

Esta vez el problema ha surgido con valores NaN. Dado que el algoritmo funciona impecablemente en el primer caso de uso, todo apunta a que el problema está en que el algoritmo no se adapta muy bien a la red compleja del *challenge*. La aparición de los valores NaN puede ser debido al problema del *exploding gradient*, donde los gradientes llegan a valores infinitos. Algunos participantes del *challenge* también han reportado tener aparentemente el mismo problema. [65]

5.7. Limitaciones

El *challenge* estaba pensado para hacerlo con GPUs, pero por desgracia las GPUs de Ikerlan no estaban preparadas para usarlas en Windows y no ha habido tiempo de hacer la adaptación necesaria para que funcionasen, así que no he tenido otra opción que realizar las ejecuciones mediante una CPU. A consecuencia de ello, dado que los CPUs son muchísimo más lentos que los GPUs, los tiempos de ejecución han sido muy largos y han limitado mucho mi trabajo.

De hecho, en un intento de entrenar un modelo sobre el particionamiento *partitioning_1.csv*, la ejecución ha tardado 9 horas en que solamente 2 colaboradores sobre 17 realizasen sus tareas (validación global, entrenamiento y validación local). Dos colaboradores que sus tiempos simulados han sido 0.93 y 2.08 minutos. Si hacemos los cálculos podemos ver que para completar una sola ronda entera la ejecución hubiera tardado más de 3 días, cuando el tiempo simulado de la ronda entera seguramente sería menos de una hora.

Por otro lado, la Figura 5.9 muestra los resultados de una de las primeras ejecuciones que hice de prueba, ejecutada sobre el particionamiento *mysplit.csv*. El ordenador estuvo una semana entera realizando una ejecución que solamente ha llegado hasta la ronda número 25. En cambio, el tiempo simulado nos dice que en una situación real esas 26 rondas se hubieran hecho en 289.49 minutos, es decir, en 4.82 horas. Aquí también, la diferencia con el tiempo real es abismal, pero en este caso no debería ser de tanto.

REAL TIME	ROUND	Simulation Time	Convergence Score	Sensitivity	Specificity	Accuracy
START [12:33]	WeightedAv.	NO SEED				
DAY 1 [13:08]	ROUND 0	13.52 minutes	0.003297028	0.002777969	0.999034882	0.789937735
[13:43]	ROUND 1	26.03 minutes	0.007007386	0.003367161	0.999078512	0.788967311
[14:16]	ROUND 2	35.48 minutes	0.014645807	0.004198550	0.999147236	0.790067911
[15:05]	ROUND 3	47.0 minutes	0.025824560	0.005525015	0.999266446	0.792147458
[16:07]	ROUND 4	57.86 minutes	0.030807541	0.006381996	0.999331653	0.794804633
[18:00]	ROUND 5	69.31 minutes	0.041490425	0.008229921	0.999495327	0.799805641
[22:53]	ROUND 6	78.14 minutes	0.047050003	0.009757124	0.999557316	0.804520905
[02:08]	ROUND 7	89.17 minutes	0.052849962	0.011294879	0.999657691	0.810758054
[05:57]	ROUND 8	98.3 minutes	0.061382659	0.013346850	0.999735892	0.818558514
[10:04]	ROUND 9	109.17 minutes	0.061382659	0.013752542	0.999800384	0.821202815
DAY 2 [14:36]	ROUND 10	121.38 minutes	0.067281105	0.016313149	0.999823749	0.829040110
[19:05]	ROUND 11	133.11 minutes	0.067281105	0.016843092	0.999888420	0.833300531
[23:20]	ROUND 12	145.13 minutes	0.075096532	0.019336557	0.999889672	0.838764369
[03:40]	ROUND 13	157.37 minutes	0.075096532	0.018917968	0.999927819	0.840130329
[09:04]	ROUND 14	167.24 minutes	0.078496903	0.021389538	0.999931633	0.845083952
DAY 3 [15:45]	ROUND 15	178.57 minutes	0.078496903	0.020376367	0.999956608	0.844871998
[23:54]	ROUND 16	188.67 minutes	0.085697325	0.024161058	0.999954998	0.850055039
[08:57]	ROUND 17	200.92 minutes	0.085697325	0.022821328	0.999973297	0.851950645
DAY 4 [20:01]	ROUND 18	212.59 minutes	0.085697325	0.024957329	0.999972403	0.853433907
[04:35]	ROUND 19	223.31 minutes	0.085697325	0.025583677	0.999979436	0.860416174
DAY 5 [18:15]	ROUND 20	233.79 minutes	0.085697325	0.026559018	0.999980211	0.858237743
[03:17]	ROUND 21	245.65 minutes	0.086957042	0.029114259	0.999985039	0.869472504
DAY 6 [21:00]	ROUND 22	256.36 minutes	0.093751545	0.028114097	0.999978721	0.869079411
[12:04]	ROUND 23	268.44 minutes	0.093751545	0.029877014	0.999990702	0.877130270
DAY 7 [23:06]	ROUND 24	279.08 minutes	0.110859893	0.033632617	0.999982178	0.881964982
DAY 8 [13:23]	ROUND 25	289.49 minutes	0.110859893	0.035804395	0.999987125	0.889242947
[?????]	ROUND 26	STOPPED				

Figura 5.9: Resultados de una ejecución de prueba con *WeightedAverage* sobre *mysplit.csv*.

Esta diferencia ha ocurrido por diversas razones. Una de ella es que, como he explicado antes, OpenFL simula un entorno federado en una sola maquina, lo que significa que los colaboradores se ejecutan en serie en vez de en paralelo como lo sería en una situación real. Por lo tanto, teniendo tres colaboradores, es normal que la ejecución llegue incluso a tardar tres veces más. Por otro lado, como ya he mencionado antes, utilizar CPUs limita mucho el rendimiento del ordenador, haciendo que las ejecuciones necesiten más tiempo.

De todas formas, en el caso de esta ejecución y muchas otras que he hecho de prueba, el tiempo de ejecución va creciendo de forma exponencial, cosa que no debería ocurrir. La razón de esto es que el *challenge* estaba guardando todos los modelos entrenados ronda tras ronda, en vez de solamente guardar el modelo de la última ronda. Una razón que, por muy sencilla que parezca, me ha llevado un tiempo averiguarlo, ya que los desarrolladores no han parado de actualizar el código añadiendo y cambiando muchas cosas.

Precisamente, eso ha sido otro dato que ha dificultado mi trabajo. Si bien es verdad que los desarrolladores han añadido cosas muy útiles como un parámetro para ajustar cada cuantas rondas se deberían evaluar las métricas (para acelerar las ejecuciones) y un sistema de *checkpoint*s (para interrumpir y continuar las ejecuciones desde la última ronda guardada), estas novedades han llegado bastante tarde y esparcidas.

Teniendo las limitaciones de *hardware* que he explicado y un tiempo limitado para realizar el trabajo, se hubiera agradecido tener una versión completa y sin cambios repentinos desde el principio, aunque hasta cierto punto los cambios de última hora son completamente normales en una competición como esta que se celebra por primera vez.

Por último, cabe mencionar que los organizadores del *challenge* han facilitado una plataforma para evaluar los modelos entrenados mediante un particionamiento de testeo oculto, pero que lamentablemente esta plataforma solo ha estado disponible durante un tiempo limitado y no he tenido la oportunidad de utilizarla.

6. CAPÍTULO

Conclusiones finales

El *federated learning* aún es un concepto muy nuevo, eso está claro. Sin embargo, no cabe duda de que también es algo que puede aportar mucho en numerosos ámbitos. Viendo en la Figura 6.1 la cantidad de *paper*s que se han publicado en los dos últimos años [66], está claro que el aprendizaje federado está generando cada vez más interés, por lo que se espera que haya muchos avances en los próximos años.

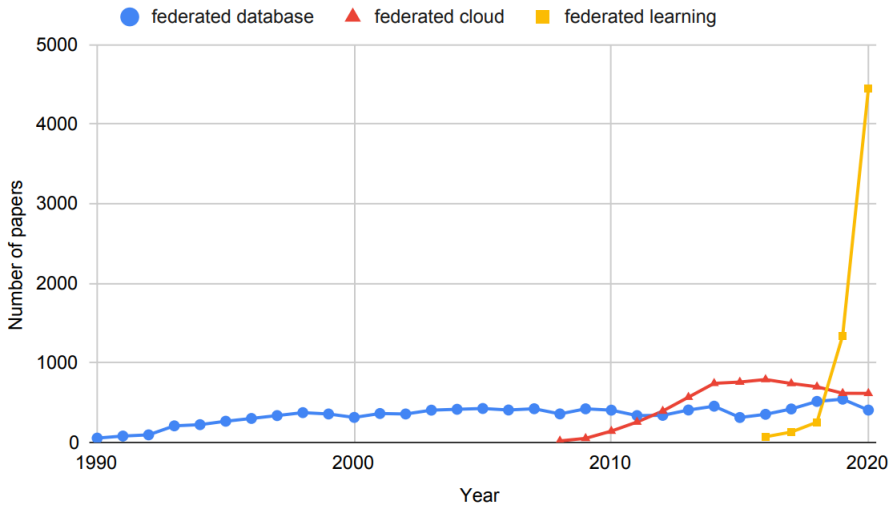

Figura 6.1: La cantidad de *paper*s que contienen las palabras clave ”base de datos federada” (azul), ”nube federada” (rojo) y ”aprendizaje federado” (amarillo) publicadas por año en Google Schoolar.

Teniendo en cuenta que la gente está siendo cada vez más consciente sobre el tema de la privacidad, tarde o temprano a las empresas no les quedará otro remedio que adoptar el aprendizaje federado. Por otro lado, la posibilidad de poder colaborar con diferentes instituciones sin tener que compartir datos privados hará que las empresas pequeñas no necesiten ayuda de las grandes, quitando importancia a tales como Amazon, Facebook o Google.

En cuanto a los *framework*s por lo visto todos están muy verdes, y aunque cada uno presenta sus especialidades ninguno sobresale realmente. De todas formas es bastante seguro que estos evolucionarán bastante rápido, y de momento puedo decir que OpenFL, el *framework* que he utilizado en este proyecto, se ve muy prometedor. Actualmente sus desarrolladores están trabajando en añadir una forma más fácil de modificar el método de agregación y en ofrecer unos cuantos métodos robustos previamente definidos. [67]

La arquitectura más común para el aprendizaje federado es utilizar un servidor central que dirige los colaboradores. Sin embargo, uno de los temas de investigación actuales es realizar este aprendizaje sin necesidad de ningún servidor central, haciendo que los colaboradores se coordinen entre sí en un esquema descentralizado. A raíz de esto también ha nacido un nuevo paradigma llamado aprendizaje asistido (*assisted learning*), el cual tiene como objetivo proporcionar protocolos para que los colaboradores aprendan de forma autónoma optimizándose los unos a los otros sin utilizar un modelo global. [68]

En lo que se refiere a los resultados que se obtienen mediante el aprendizaje federado, está claro que estos dependen mucho de lo desbalanceado que está el entorno. Aún así, en general se puede concluir que en desbalanceos habituales los resultados son muy buenos y que están bastante a la par de los que se obtienen mediante un aprendizaje tradicional.

Para finalizar, como punto negativo se podría decir que por un lado los colaboradores necesitan una mínima capacidad de computo que puede que algunos dispositivos no la tengan, y que por otro lado los *framework*s actuales están poco maduros. Sin embargo, la solución de ambos problemas solamente es cuestión de tiempo. Visto lo visto, no cabe duda de que el aprendizaje federado va a evolucionar mucho en los próximos años.

6.1. Conclusiones personales

Como conclusión personal, este trabajo no solo me ha servido para aprender sobre el aprendizaje federado, sino que también para aprender sobre competiciones y proyectos reales. Dadas las dificultades que he tenido con el segundo caso de uso, al final no he podido completar a tiempo las tareas que el *FeTS Challenge* proponía, por lo que me he quedado descalificada de la competición. Sin embargo, estoy segura de que la experiencia obtenida no ha sido en vano, y que la próxima vez podré actuar mejor ante dichas dificultades.

En cuanto al aprendizaje federado, teniendo en cuenta que me he aventurado en ello sin tener la más mínima idea de lo que trataba, tengo la sensación de que he aprendido mucho pero que a la vez todo lo aprendido solamente es la punta de un iceberg muy grande. Habiendo llegado hasta este punto, sería interesante por mi parte continuar profundizando en este tema usando este proyecto como base.

Bibliografía

[1] IKERLAN. Teknologia, gure izaera. https://www.ikerlan.es/eu/ [Visitado el 13/09/2021].

[2] MONDRAGON Corporation. Mondragon corporation. https://www.mondragon-corporation.com/eu/ [Visitado el 13/09/2021].

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *CoRR*, 2014. arXiv:1206.5538.

[4] Jürgen Schmidhuber. Deep Learning in Neural Networks: An Overview. *CoRR*, 2014. arXiv:1404.7828.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Association for Computing Machinery*, 2017. DOI:10.1145/3065386.

[6] Andrew Ng — Coursera. Deep Learning Specialization. https://www.coursera.org/specializations/deep-learning [Visitado el 19/08/2021].

[7] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, and N.I. Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, 2020. DOI:10.1016/j.matcom.2020.04.031.

[8] Jakub Konečný, Brendan McMahan, and Daniel Ramage. Federated Optimization: Distributed Optimization Beyond the Datacenter. *CoRR*, 2015. arXiv:1511.03575.

[9] Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. *CoRR*, 2016. arXiv:1610.02527.

[10] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated Learning of Deep Networks using Model Averaging. *CoRR*, 2016. arXiv:1602.05629.

[11] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben

Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and Open Problems in Federated Learning. *CoRR*, 2019. arXiv:1912.04977.

[12] Wei Liu, Li Chen, and Wenyi Zhang. Decentralized Federated Learning: Balancing Communication and Computing Costs. *CoRR*, 2021. arXiv:2107.12048.

[13] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated Learning of Deep Networks using Model Averaging. *CoRR*, 2016. arXiv:1602.05629.

[14] Enmao Diao, Jie Ding, and Vahid Tarokh. HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients. *CoRR*, 2020. arXiv:2010.01264.

[15] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How To Backdoor Federated Learning. *CoRR*, 2018. arXiv:1807.00459.

[16] Haiqin Weng, Juntao Zhang, Feng Xue, Tao Wei, Shouling Ji, and Zhiyuan Zong. Privacy Leakage of Real-World Vertical Federated Learning. *CoRR*, 2020. arXiv:2011.09290.

[17] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated Machine Learning: Concept and Applications. *CoRR*, 2019. arXiv:1902.04885.

[18] Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially Private Federated Learning: A Client Level Perspective. *CoRR*, 2017. arXiv:1712.07557.

[19] David Byrd and Antigoni Polychroniadou. Differentially Private Secure Multi-Party Computation for Federated Learning in Financial Applications. *CoRR*, 2020. arXiv:2010.05867.

[20] Melissa Crooks — Chatbots Journal. Applications Of Machine Learning In Industry 4.0, March 3, 2020. https://chatbotsjournal.com/applications-of-machine-learning-in-industry-4-0-ca469435a1cf [Visitado el 30/08/2021].

[21] Ahmet M. Elbir, Burak Soner, and Sinem Coleri. Federated Learning in Vehicular Networks. *CoRR*, 2020. arXiv:2006.01412.

[22] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletarì, Holger R. Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N. Galtier, Bennett A. Landman, Klaus Maier-Hein, Sébastien Ourselin, Micah Sheller, Ronald M. Summers, Andrew Trask, Daguang Xu, Maximilian Baust, and M. Jorge Cardoso. The future of digital health with federated learning. *NPJ Digit Med.*, 2020. DOI:10.1038/s41746-020-00323-1.

[23] G Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, Jason Martin, Brandon Edwards, Micah J. Sheller, Sarthak Pati, Prakash Narayana Moorthy, Shih han Wang, Prashant Shah, and Spyridon Bakas. OpenFL: An open-source framework for Federated Learning. *CoRR*, 2021. arXiv:2105.06413.

[24] Sarthak Pati, Ujjwal Baid, Maximilian Zenk, Brandon Edwards, Micah Sheller, G. Anthony Reina, Patrick Foley, Alexey Gruzdev, Jason Martin, Shadi Albarqouni, Yong Chen, Russell Taki Shinohara, Annika Reinke, David Zimmerer, John B. Freymann, Justin S. Kirby, Christos Davatzikos, Rivka R. Colen, Aikaterini Kotrotsou, Daniel Marcus, Mikhail Milchenko, Arash Nazeri, Hassan Fathallah-Shaykh, Roland Wiest, Andras Jakab, Marc-Andre Weber, Abhishek Mahajan, Lena Maier-Hein, Jens Kleesiek, Bjoern Menze, Klaus Maier-Hein, and Spyridon Bakas. The Federated Tumor Segmentation (FeTS) Challenge. *CoRR*, 2021. arXiv:2105.05874.

[25] TensorFlow Developers. TensorFlow Federated. https://www.tensorflow.org/federated [Visitado el 30/08/2021].

[26] Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Jeremy Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, Mark Purcell, Ambrish Rawat, Tran Ngoc Minh, Naoise Holohan, Supriyo Chakraborty, Shalisha Witherspoon, Dean Steuer, Laura Wynter, Hifaz Hassan, Sean Laguna, Mikhail Yurochkin, Mayank Agarwal, Ebube Chuba, and Annie Abay. IBM Federated Learning: an Enterprise Framework White Paper v0.1. *CoRR*, 2020. arXiv:2007.10987.

[27] IBM — GitHub. IBM/federated-learning-lib: A library for federated learning (a distributed machine learning process) in an enterprise environment. https://github.com/IBM/federated-learning-lib [Visitado el 30/08/2021].

[28] PaddlePaddle — GitHub. PaddlePaddle/PaddleFL: Federated Deep Learning in PaddlePaddle. https://github.com/PaddlePaddle/PaddleFL [Visitado el 30/08/2021].

[29] OpenMined — GitHub. OpenMined/PySyft: A library for answering questions using data you cannot see. https://github.com/OpenMined/PySyft [Visitado el 30/08/2021].

[30] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. Flower: A Friendly Federated Learning Research Framework. *CoRR*, 2020. arXiv:2007.14390.

[31] Adap — GitHub. adap/flower: Flower - A Friendly Federated Learning Framework. https://github.com/SMILELab-FL/FedLab [Visitado el 30/08/2021].

[32] Dun Zeng, Siqi Liang, Xiangjing Hu, and Zenglin Xu. FedLab: A Flexible Federated Learning Framework. *CoRR*, 2021. arXiv:2107.11621.

[33] SMILELab-FL — GitHub. SMILELab-FL/FedLab: A flexible Federated Learning Framework based on PyTorch, simplifying your Federated Learning research. https://github.com/SMILELab-FL/FedLab [Visitado el 30/08/2021].

[34] FederatedAI — GitHub. FederatedAI/FATE: An Industrial Grade Federated Learning Framework. https://github.com/FederatedAI/FATE [Visitado el 04/09/2021].

[35] Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical Secure Aggregation for Federated Learning on User-Held Data. *CoRR*, 2016. arXiv:1611.04482.

[36] Jinhyun So, Ramy E. Ali, Basak Guler, Jiantao Jiao, and Salman Avestimehr. Securing Secure Aggregation: Mitigating Multi-Round Privacy Leakagein Federated Learning. *CoRR*, 2021. arXiv:2106.03328.

[37] Krishna Pillutla, Sham M. Kakade, and Zaid Harchaoui. Robust Aggregation for Federated Learning. *CoRR*, 2019. arXiv:1912.13445.

[38] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the Convergence of Federated Optimization in Heterogeneous Networks. *CoRR*, 2018. arXiv:1812.06127.

[39] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. *In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc.*, 2017. DOI:10.5555/3294771.3294783.

[40] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter L. Bartlett. Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. *CoRR*, 2018. arXiv:1803.01498.

[41] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Zeno: Byzantine-suspicious stochastic gradient descent. *CoRR*, 2018. arXiv:1805.10032.

[42] Intel — GitHub. intel/openfl: An open framework for Federated Learning. https://github.com/intel/openfl [Visitado el 01/09/2021].

[43] PyTorch. torchvision.datasets — Torchvision 0.10.0 documentation. https://pytorch.org/vision/stable/datasets.html [Visitado el 05/09/2021].

[44] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep Learning for Classical Japanese Literature. *CoRR*, 2018. arXiv:1812.01718.

[45] OpenFL — GitHub. openfl/Federated_Pytorch_MNIST_Tutorial.ipynb at develop. https://github.com/intel/openfl/blob/develop/openfl-tutorials/Federated_Pytorch_MNIST_Tutorial.ipynb [Visitado el 06/09/2021].

[46] Dhaval Taunk — Analytics Vidhya — Medium. L1 vs L2 Regularization: The intuitive difference, March 15, 2020. https://medium.com/analytics-vidhya/l1-vs-l2-regularization-which-is-better-d01068e6658c [Visitado el 07/09/2021].

[47] Wikipedia. Exponential smoothing. https://en.wikipedia.org/wiki/Exponential_smoothing [Visitado el 01/09/2021].

[48] CBICA — Perelman School of Medicine at the University of Pennsylvania. MICCAI Federated Tumor Segmentation (FeTS) Challenge 2021. https://www.med.upenn.edu/cbica/fets/miccai2021/ [Visitado el 27/07/2021].

[49] CBICA — Perelman School of Medicine at the University of Pennsylvania. RSNA-ASNR-MICCAI Brain Tumor Segmentation (BraTS) Challenge 2021. https://www.med.upenn.edu/cbica/brats2021/ [Visitado el 27/07/2021].

[50] The FeTS 2021 organizing committee. Challenge Description — FeTS Challenge 2021. https://fets-ai.github.io/Challenge/tasks/ [Visitado el 27/07/2021].

[51] Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras, Michel Bilello, Martin Rozycki, Justin S. Kirby, John B. Freymann, Keyvan Farahani, and Christos Davatzikos. Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features. *Nature Scientific Data*, 2017. DOI:10.1038/sdata.2017.117.

[52] Micah J. Sheller, Brandon Edwards, G. Anthony Reina, Jason Martin, Sarthak Pati, Aikaterini Kotrotsou, Mikhail Milchenko, Weilin Xu, Daniel Marcus, Rivka R. Colen, and Spyridon Bakas. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. *Nature Scientific Reports*, 2020. DOI:10.1038/s41598-020-69250-1.

[53] FETS-AI — GitHub. Challenge/Task_1 at main. https://github.com/FETS-AI/Challenge/tree/main/Task_1/ [Visitado el 27/07/2021].

[54] The FeTS 2021 organizing committee. Data — FeTS Challenge 2021. https://fets-ai.github.io/Challenge/data/ [Visitado el 27/07/2021].

[55] Sociedad Española de Imagen Cardíaca. ¿Qué es el T1 y el T2? https://ecocardio.com/documentos/biblioteca-preguntas-basicas/preguntas-al-radiologo/914-que-es-t1-y-t2.html [Visitado el 27/07/2021].

[56] Radiology Masterclass. MRI interpretation - T1 v T2 images. https://www.radiologymasterclass.co.uk/tutorials/mri/t1_and_t2_images [Visitado el 27/07/2021].

[57] Spyridon Bakas, Mauricio Reyes, Andras Jakab, Stefan Bauer, Markus Rempfler, Alessandro Crimi, Russell Takeshi Shinohara, Christoph Berger, Sung Min Ha, Martin Rozycki, Marcel Prastawa, Esther Alberts, Jana Lipkova, John Freymann, Justin Kirby, Michel Bilello, Hassan Fathallah-Shaykh, Roland Wiest, Jan Kirschke, Benedikt Wiestler, Rivka Colen, Aikaterini Kotrotsou, Pamela Lamontagne, Daniel Marcus, Mikhail Milchenko, Arash Nazeri, Marc-Andre Weber, Abhishek Mahajan, Ujjwal Baid, Elizabeth Gerstner, Dongjin Kwon, Gagan Acharya, Manu Agarwal, Mahbubul Alam, Alberto Albiol, Antonio Albiol, Francisco J. Albiol, Varghese Alex, Nigel Allinson, Pedro H. A. Amorim, Abhijit Amrutkar, Ganesh Anand, Simon Andermatt, Tal Arbel, Pablo Arbelaez, Aaron Avery, Muneeza Azmat, Pranjal B., W Bai, Subhashis Banerjee, Bill Barth, Thomas Batchelder, Kayhan Batmanghelich, Enzo Battistella, Andrew Beers, Mikhail Belyaev, Martin Bendszus, Eze Benson, Jose Bernal, Halandur Nagaraja Bharath, George Biros, Sotirios Bisdas, James Brown, Mariano Cabezas, Shilei Cao, Jorge M. Cardoso, Eric N Carver, Adrià Casamitjana, Laura Silvana Castillo, Marcel Catà, Philippe Cattin, Albert Cerigues, Vinicius S. Chagas, Siddhartha Chandra, Yi-Ju Chang, Shiyu Chang, Ken Chang, Joseph Chazalon, Shengcong Chen, Wei Chen, Jefferson W Chen, Zhaolin Chen, Kun Cheng, Ahana Roy Choudhury, Roger Chylla, Albert Clérigues, Steven Colleman, Ramiro German Rodriguez Colmeiro, Marc Combalia, Anthony Costa, Xiaomeng Cui, Zhenzhen Dai, Lutao Dai, Laura Alexandra Daza, Eric Deutsch, Changxing Ding, Chao Dong,

Shidu Dong, Wojciech Dudzik, Zach Eaton-Rosen, Gary Egan, Guilherme Escudero, Théo Estienne, Richard Everson, Jonathan Fabrizio, Yong Fan, Longwei Fang, Xue Feng, Enzo Ferrante, Lucas Fidon, Martin Fischer, Andrew P. French, Naomi Fridman, Huan Fu, David Fuentes, Yaozong Gao, Evan Gates, David Gering, Amir Gholami, Willi Gierke, Ben Glocker, Mingming Gong, Sandra González-Villá, T. Grosges, Yuanfang Guan, Sheng Guo, Sudeep Gupta, Woo-Sup Han, Il Song Han, Konstantin Harmuth, Huiguang He, Aura Hernández-Sabaté, Evelyn Herrmann, Naveen Himthani, Winston Hsu, Cheyu Hsu, Xiaojun Hu, Xiaobin Hu, Yan Hu, Yifan Hu, Rui Hua, Teng-Yi Huang, Weilin Huang, Sabine Van Huffel, Quan Huo, Vivek HV, Khan M. Iftekharuddin, Fabian Isensee, Mobarakol Islam, Aaron S. Jackson, Sachin R. Jambawalikar, Andrew Jesson, Weijian Jian, Peter Jin, V Jeya Maria Jose, Alain Jungo, B Kainz, Konstantinos Kamnitsas, Po-Yu Kao, Ayush Karnawat, Thomas Kellermeier, Adel Kermi, Kurt Keutzer, Mohamed Tarek Khadir, Mahendra Khened, Philipp Kickingereder, Geena Kim, Nik King, Haley Knapp, Urspeter Knecht, Lisa Kohli, Deren Kong, Xiangmao Kong, Simon Koppers, Avinash Kori, Ganapathy Krishnamurthi, Egor Krivov, Piyush Kumar, Kaisar Kushibar, Dmitrii Lachinov, Tryphon Lambrou, Joon Lee, Chengen Lee, Yuehchou Lee, M Lee, Szidonia Lefkovits, Laszlo Lefkovits, James Levitt, Tengfei Li, Hongwei Li, Wenqi Li, Hongyang Li, Xiaochuan Li, Yuexiang Li, Heng Li, Zhenye Li, Xiaoyu Li, Zeju Li, XiaoGang Li, Wenqi Li, Zheng-Shen Lin, Fengming Lin, Pietro Lio, Chang Liu, Boqiang Liu, Xiang Liu, Mingyuan Liu, Ju Liu, Luyan Liu, Xavier Llado, Marc Moreno Lopez, Pablo Ribalta Lorenzo, Zhentai Lu, Lin Luo, Zhigang Luo, Jun Ma, Kai Ma, Thomas Mackie, Anant Madabushi, Issam Mahmoudi, Klaus H. Maier-Hein, Pradipta Maji, CP Mammen, Andreas Mang, B. S. Manjunath, Michal Marcinkiewicz, S McDonagh, Stephen McKenna, Richard McKinley, Miriam Mehl, Sachin Mehta, Raghav Mehta, Raphael Meier, Christoph Meinel, Dorit Merhof, Craig Meyer, Robert Miller, Sushmita Mitra, Aliasgar Moiyadi, David Molina-Garcia, Miguel A.B. Monteiro, Grzegorz Mrukwa, Andriy Myronenko, Jakub Nalepa, Thuyen Ngo, Dong Nie, Holly Ning, Chen Niu, Nicholas K Nuechterlein, Eric Oermann, Arlindo Oliveira, Diego D. C. Oliveira, Arnau Oliver, Alexander F. I. Osman, Yu-Nian Ou, Sebastien Ourselin, Nikos Paragios, Moo Sung Park, Brad Paschke, J. Gregory Pauloski, Kamlesh Pawar, Nick Pawlowski, Linmin Pei, Suting Peng, Silvio M. Pereira, Julian Perez-Beteta, Victor M. Perez-Garcia, Simon Pezold, Bao Pham, Ashish Phophalia, Gemma Piella, G.N. Pillai, Marie Piraud, Maxim Pisov, Anmol Popli, Michael P. Pound, Reza Pourreza, Prateek Prasanna, Vesna Prkovska, Tony P. Pridmore, Santi Puch, Élodie Puybareau, Buyue Qian, Xu Qiao, Martin Rajchl, Swapnil Rane, Michael Rebsamen, Hongliang Ren, Xuhua Ren, Karthik Revanuru, Mina Rezaei, Oliver Rippel, Luis Carlos Rivera, Charlotte Robert, Bruce Rosen, Daniel Rueckert, Mohammed Safwan, Mostafa Salem, Joaquim Salvi, Irina Sanchez, Irina Sánchez, Heitor M. Santos, Emmett Sartor, Dawid Schellingerhout, Klaudius Scheufele, Matthew R. Scott, Artur A. Scussel, Sara Sedlar, Juan Pablo Serrano-Rubio, N. Jon Shah, Nameetha Shah, Mazhar Shaikh, B. Uma Shankar, Zeina Shboul, Haipeng Shen, Dinggang Shen, Linlin Shen, Haocheng Shen, Varun Shenoy, Feng Shi, Hyung Eun Shin, Hai Shu, Diana Sima, M Sinclair, Orjan Smedby, James M. Snyder, Mohammadreza Soltaninejad, Guidong Song, Mehul Soni, Jean Stawiaski, Shashank Subramanian, Li Sun, Roger Sun, Jiawei Sun, Kay Sun, Yu Sun, Guoxia Sun, Shuang Sun, Yannick R Suter, Laszlo Szilagyi, Sanjay Talbar, Dacheng Tao, Dacheng Tao, Zhongzhao Teng, Siddhesh Thakur, Meenakshi H Thakur, Sameer Tharakan, Pallavi Tiwari, Guillaume Tochon, Tuan Tran, Yuhsiang M. Tsai, Kuan-Lun Tseng, Tran Anh Tuan, Vadim Turlapov, Nicholas Tustison, Maria Vakalopoulou, Sergi Valverde, Rami Vanguri, Evgeny

Vasiliev, Jonathan Ventura, Luis Vera, Tom Vercauteren, C. A. Verrastro, Lasitha Vidyaratne, Veronica Vilaplana, Ajeet Vivekanandan, Guotai Wang, Qian Wang, Chiatse J. Wang, Weichung Wang, Duo Wang, Ruixuan Wang, Yuanyuan Wang, Chunliang Wang, Guotai Wang, Ning Wen, Xin Wen, Leon Weninger, Wolfgang Wick, Shaocheng Wu, Qiang Wu, Yihong Wu, Yong Xia, Yanwu Xu, Xiaowen Xu, Peiyuan Xu, Tsai-Ling Yang, Xiaoping Yang, Hao-Yu Yang, Junlin Yang, Haojin Yang, Guang Yang, Hongdou Yao, Xujiong Ye, Changchang Yin, Brett Young-Moxon, Jinhua Yu, Xiangyu Yue, Songtao Zhang, Angela Zhang, Kun Zhang, Xuejie Zhang, Lichi Zhang, Xiaoyue Zhang, Yazhuo Zhang, Lei Zhang, Jianguo Zhang, Xiang Zhang, Tianhao Zhang, Sicheng Zhao, Yu Zhao, Xiaomei Zhao, Liang Zhao, Yefeng Zheng, Liming Zhong, Chenhong Zhou, Xiaobing Zhou, Fan Zhou, Hongtu Zhu, Jin Zhu, Ying Zhuge, Weiwei Zong, Jayashree Kalpathy-Cramer, Keyvan Farahani, Christos Davatzikos, Koen van Leemput, and Bjoern Menze. Identifying the Best Machine Learning Algorithms for Brain Tumor Segmentation, Progression Assessment, and Overall Survival Prediction in the BRATS Challenge. *CoRR*, 2018. arXiv:1811.02629.

[58] Research Imaging Institute. Mango. ric.uthscsa.edu/mango/index.html [Visitado el 27/07/2021].

[59] Wikipedia. Sørensen–Dice coefficient. https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient [Visitado el 02/09/2021].

[60] Wikipedia. Hausdorff distance. https://en.wikipedia.org/wiki/Hausdorff_distance [Visitado el 02/09/2021].

[61] FETS-AI — GitHub. How Simulated Time is computed - challenge/Task_1 at main. https://github.com/FETS-AI/Challenge/tree/main/Task_1#how-simulated-time-is-computed [Visitado el 02/09/2021].

[62] Wikipedia. Sensibilidad y especificidad. https://es.wikipedia.org/wiki/Sensibilidad_y_especificidad [Visitado el 02/09/2021].

[63] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3D U-Net: Learning Dense Volumetric segmentation from Sparse Annotation. *CoRR*, 2016. arXiv:1606.06650.

[64] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. *CoRR*, 2014. arXiv:1411.4038.

[65] FETS-AI — GitHub. Task-1: ValueError: A model output tensor was found to have nan values — Discussion #113. https://github.com/FETS-AI/Challenge/discussions/113 [Visitado el 04/09/2021].

[66] Qinbin Li, Zeyi Wen, and Bingsheng He. Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection. *CoRR*, 2019. arXiv:1907.09693.

[67] Intel — GitHub. Projects intel/openfl. https://github.com/intel/openfl/projects [Visitado el 15/09/2021].

[68] Xun Xian, Xinran Wang, Jie Ding, and Reza Ghanadan. Assisted Learning and Imitation Privacy. *CoRR*, 2020. arXiv:2004.00566.