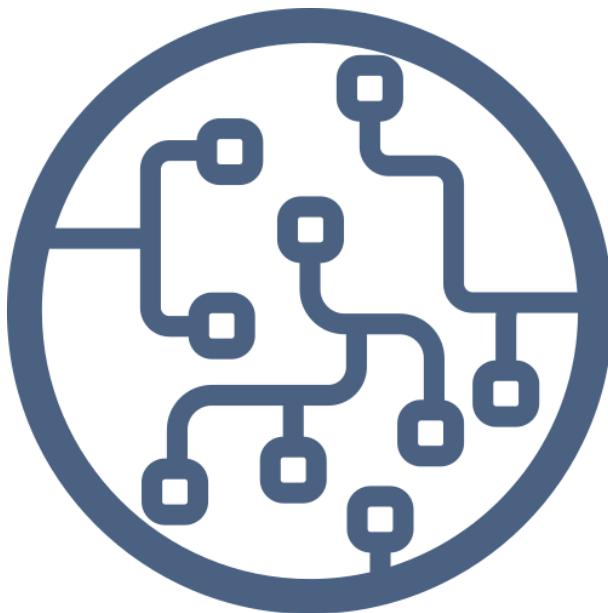


MÁSTER UNIVERSITARIO EN INGENIERÍA DE CONTROL,
AUTOMATIZACIÓN Y ROBÓTICA

TRABAJO FIN DE MASTER

***DISEÑO Y SIMULACIÓN INTELIGENTES DE
SISTEMAS FLUIDODINÁMICOS EN ESTADO NO
ESTACIONARIO MEDIANTE DEEP LEARNING***



Estudiante: Abucide Armas, Álvaro

Director/Directora: Zulueta Guerrero, Ekaitz

Curso: 2022/2023

Fecha: Bilbao, 10 de julio de 2023

Resumen

La dinámica de fluidos computacional (CFD, del inglés *Computational Fluid Dynamics*) aplicada a flujos turbulentos ha sido objeto de estudio durante los últimos años. Sin embargo, mediante las simulaciones CFD los costes computacionales se elevan considerablemente, siendo inviable el uso de estas técnicas para la resolución de ciertos problemas. Actualmente, se están desarrollando múltiples técnicas de *Deep Learning* (DL) aplicadas a los problemas de CFD. Una de las principales aplicaciones del DL al CFD es el empleo de redes neuronales convolucionales (CNN, del inglés *Convolutional Neural Networks*) para predecir los valores de las magnitudes fluido dinámicas deseadas alrededor de una geometría determinada. En este caso, se ha desarrollado una CNN que predice los campos de velocidades paralela al flujo y vertical y de presión. Se toman dos enfoques: la predicción de los instantes futuros partiendo de uno inicial y la predicción del instante inicial. La CNN adquiere como entradas las características geométricas de la forma analizada y el instante anterior, en el caso del enfoque temporal, siendo las salidas de la red los campos de ambas velocidades y de la presión, obtenidos mediante simulaciones CFD.

Abstract

The application of computational fluid dynamics (CFD) to turbulent flow has lately been a considerable topic of research. Nonetheless, the use of CFD tools results in large computational costs, which implies that, for some applications, CFD may be inviable. Several authors have conducted research applying deep learning (DL) techniques to CFD-based simulations to date. One of the main applications of DL with CFD is the use of convolutional neural networks (CNN) to predict the values of the desired fluid-dynamic magnitudes around a concrete geometry. In this case, a CNN which predicts the streamwise and vertical velocities and pressure fields has been developed. Two approaches are considered: the prediction of the future instants on the basis of an initial sample and the prediction of the initial sample. The CNN takes as inputs the geometric features of the analyzed form and the previous instant, in the case of the temporal approach, being the outputs of the net the coupled velocities and the pressure, obtained by CFD simulations.

Laburpena

Fluxu zurrubilotsuei aplikatutako fluido konputazionalen dinamika (CFD, ingelese-tik *Computational Fluid Dynamics*) aztertu da azken urteetan. CFD simulazioen bidez, ordea, kostu konputazionalak nabarmen igotzen dira, eta bidera ezina da teknika horiek erabiltzea zenbait problema ebazteko. Gaur egun, *Deep Learning* (DL) teknika ugari garatzen ari dira, CFDren arazoei aplikatuta. DLk CFDn duen aplikazio nagusietako bat sare neuronal konboluzionalak (CNN, inglesetik *Convolutional Neural Networks*) erabiltzea da, geometria jakin baten inguruaren nahi diren magnitude fluido dinamikoen balioak iragartzeko. Kasu honetan, CNN bat garatu da fluxuarekiko paraleloa eta bertikala diren abiadura-eremuak eta presio-eremua iragartzeko. Bi ikuspegi hartzen dira: hasierako une batetik abiatuz etorkizuneko uneen iragarpena eta hasierako unearren iragarpena. CNNk sarrera gisa hartzen ditu aztertutako formaren ezaugarri geometrikoak, eta aurreko unea denbora-ikuspegiaren kasuan. Sareko irteerak bi abiaduren eta presioaren eremuak dira, CFD simulazioen bidez lortuak.

Palabras clave:

Deep Learning, dinámica de fluidos computacional, redes neuronales convolucionales, U-Net, flujo turbulento

Índice

Lista de tablas	V
Lista de ilustraciones	VII
Acrónimos	IX
1. Introducción y contexto	1
2. Alcance y objetivos	3
3. Antecedentes bibliográficos o estado del arte	5
3.1. Introducción a la Inteligencia Artificial y el <i>Machine Learning</i>	5
3.1.1. Parámetros e hiper-parámetros	9
3.1.2. Tipos y características de ANN	11
3.2. Dinámica de fluidos Computacional	14
4. Desarrollo de la solución	17
4.1. Geometría circular con velocidad de entrada variable	17
4.1.1. Simulaciones numéricas	17
4.1.2. Red neuronal convolucional	19
4.1.3. Entradas de la red neuronal	22
4.1.4. Parámetros del entrenamiento	24
4.2. Geometrías variables	26
4.2.1. Simulaciones numéricas para geometrías variables	26
4.2.2. Red neuronal convolucional	27
4.2.3. Parámetros del entrenamiento	28
4.2.4. <i>Data augmentation</i>	29
5. Análisis de resultados	33
5.1. Resultados de la red aplicada a una geometría circular con velocidad de entrada variable	33
5.2. Resultados de la red que predice los instantes futuros aplicada a geometrías variables	35
5.3. Resultados de la red que predice la muestra inicial para geometrías variables	45
5.4. Análisis del coste computacional	54

6. Conclusiones y trabajos futuros	57
6.1. Discusión general	57
6.2. Conclusiones específicas	57
6.3. Líneas de trabajo futuras	58
7. Presupuesto y planificación	59
7.1. Presupuesto	59
7.2. Planificación	60
Referencias bibliográficas	61
A. Esquemas y programas fuente	67
A.1. Carga de archivos CSV e interpolación	67
A.2. Generar las matrices de SDF y FRC	69
A.3. <i>Data augmentation</i>	72
A.4. Generar los datos de entrenamiento	77
A.5. CNN	81
A.6. Test de los modelos neuronales y análisis de los resultados	114

Lista de tablas

4.1. Conjunto de valores seleccionados para la búsqueda de los hiper-parámetros adecuados para la red que predice los instantes futuros para velocidades de entrada al dominio variables.	25
4.2. Conjunto de valores seleccionados para la búsqueda de los hiper-parámetros adecuados para la red que predice los instantes futuros de los campos con geometrías variables.	29
4.3. Conjunto de valores seleccionados para la búsqueda de los hiper-parámetros adecuados para la red que predice la muestra inicial. . . .	29
5.1. Identificador de cada modelo, combinación de los hiper-parámetros utilizados en cada uno de los modelos entrenados y duración de los entrenamientos de cada variable.	34
5.2. Media aritmética y desviación estándar de las simulaciones CFD y las predicciones de la CNN por cada velocidad de entrada al dominio. . . .	35
5.3. 10 mejores entrenamientos para predicción de los instantes futuros del campo de velocidad paralela al flujo.	36
5.4. 10 mejores entrenamientos para la predicción de los instantes futuros del campo de velocidad vertical.	36
5.5. 10 mejores entrenamientos para predicción de los instantes futuros del campo de presión.	36
5.6. Media aritmética y desviación estándar de las 50 muestras predichas por la CNN y simuladas mediante CFD.	45
5.7. 10 mejores entrenamientos para la velocidad paralela al flujo. . . .	52
5.8. 10 mejores entrenamientos para la velocidad vertical.	53
5.9. 10 mejores entrenamientos para la presión.	54
5.10. Media aritmética y desviación estándar de las predicciones de la CNN de la primera muestra y de las simulaciones CFD de la muestra inicial. .	54
5.11. Comparación del tiempo de cálculo de los campos de velocidades paralela al flujo y vertical y de presión.	56
7.1. Amortizaciones.	59
7.2. Presupuesto.	59

Lista de ilustraciones

3.1.	Diagrama que representa una ANN simple, que contiene las capas de entrada, oculta y de salida. Si tuviese varias capas ocultas sería una red de DL.	6
3.2.	Diagrama que muestra que el DL se engloba dentro del ML y, que a su vez, este último, es un subconjunto de IA.	7
3.3.	Organigrama que muestra cómo las diferentes partes de un sistema de IA se relacionan entre ellas en las diferentes disciplinas de la IA. Las cajas en gris muestran los componentes que son capaces de aprender de los datos.	8
3.4.	Diagrama de una neurona con sus respectivas entradas, pesos sinápticos, función de activación y salida.	9
3.5.	Diagramas de las arquitecturas de los principales tipos de redes neuronales: a) MLP, b) CNN, c) RNN y d) GAN.	12
3.6.	Diagrama de una convolución [16].	13
3.7.	Ejemplos de operaciones de <i>pooling</i>	14
4.1.	Dominio computacional de las simulaciones con velocidad de entrada variable (sin escalar).	18
4.2.	Distribución del mallado alrededor de la geometría circular.	19
4.3.	Arquitectura U-Net.	20
4.4.	Arquitectura detallada de la CNN.	22
4.5.	Diagrama representativo del FRC.	23
4.6.	SDF de un círculo.	24
4.7.	Dominio computacional de la simulaciones con geometrías variables (sin escalar).	26
4.8.	Ejemplo de la distribución del mallado alrededor de una geometría.	27
4.9.	Arquitectura U-Net con 3 decodificadores.	28
4.10.	Arquitectura detallada de la CNN con 3 decodificadores.	28
5.1.	Errores medios de cada modelo neuronal entrenado para cada una de las velocidades de entrada al dominio. a) Velocidad paralela al flujo, b) velocidad vertical y c) presión.	37
5.2.	Errores máximos de cada modelo neuronal entrenado para cada una de las velocidades de entrada al dominio. a) Velocidad paralela al flujo, b) velocidad vertical y c) presión.	38

5.3. Predicciones de la geometría circular con una velocidad de entrada de 5 m/s de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.	39
5.4. Predicciones de la geometría circular con una velocidad de entrada de 15 m/s de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.	40
5.5. Predicciones de la geometría circular con una velocidad de entrada de 25 m/s de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.	41
5.6. Histogramas de la distribución de los datos de CFD y de la CNN del campo de velocidad paralela al flujo alrededor de la geometría circular para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.	42
5.7. Histogramas de la distribución de los datos de CFD y de la CNN del campo de velocidad vertical alrededor de la geometría circular para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.	43
5.8. Histogramas de la distribución de los datos de CFD y de la CNN del campo de presión alrededor de la geometría circular para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.	44
5.9. Predicciones de la geometría circular de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.	46
5.10. Predicciones de la elipse de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.	47
5.11. Predicciones del cuadrado de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.	48
5.12. Histogramas de la distribución de los datos de CFD y de la CNN del campo de velocidad paralela al flujo para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.	49
5.13. Histogramas de la distribución de los datos de CFD y de la CNN del campo de velocidad vertical para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.	50
5.14. Histogramas de la distribución de los datos de CFD y de la CNN del campo de presión para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.	51
5.15. Predicciones de la CNN para a) círculo, b) cilindro, c) cuadrado, d) rectángulo, e) triángulo y f) triángulo equilátero.	55
5.16. Distribución de los datos para: a) v_x , b) v_y y c) p.	56

Acrónimos

- ADAM** Estimación del momento adaptativo, del inglés *Adaptative Moment Estimation*
- ANN** Red neuronal artificial, del inglés *Artificial Neural Network*
- ASO** Optimización de formas aerodinámicas, del inglés *Aerodynamic Shape Optimization*
- CFD** Dinámica de fluidos computacional, del inglés *Computational Fluid Dynamics*
- CNN** Red neuronal convolucional, del inglés *Convolutional Neural Network*
- DL** *Deep Learning*
- DNN** Red neuronal profunda, del inglés *Deep Neural Network*
- FRC** Canal de la región de flujo, del inglés *Flow Region Channel*
- GAN** Red neuronal generativa adversaria, del inglés *Generative Adversarial Networks*
- HPC** Recursos computacionales de alta eficiencia, del inglés *high-performance computing*
- IA** Inteligencia Artificial
- LGR** Representación geométrica latente, del inglés *Latent Geometry Representation*
- MAE** Error medio absoluto, del inglés *Mean Absolute Error*
- MSE** Error medio cuadrático, del inglés *Mean Squared Error*
- RANS** Navier-Stokes promediadas por Reynolds, del inglés *Reynolds-Averaged Navier-Stokes*
- ReLU** Unidad lineal rectificada, del inglés *rectified linear unit*
- RMSE** Raíz del error cuadrático medio, del inglés *Root Mean Square Error*

RMSprop Propagación de la raíz cuadrática media, del inglés *root mean square propagation*

RNN Red neuronal recurrente, del inglés *Recurrent Neural Network*

SDF Función de distancia con signo, del inglés *Signed Distance Function*

SGD Gradiente de descenso estocástico, del inglés *stochastic gradient descent*

URANS Inestables de Navier-Stokes promediadas por Reynolds, del inglés *Unsteady Reynolds-Averaged Navier-Stokes*

Introducción y contexto

La Inteligencia Artificial (IA) se ha convertido en los últimos años en una tecnología de gran utilidad para la resolución de diversos problemas prácticos. Esto se debe, fundamentalmente, a las capacidades de aprendizaje, razonamiento y adaptación de los sistemas inteligentes, que permiten que los métodos de IA estén alcanzando niveles de rendimiento sin precedentes a la hora de resolver problemas computacionales complejos [6]. La IA se ha postulado como una herramienta de valiosa utilidad en aplicaciones como el procesamiento del lenguaje natural, el procesamiento de imágenes, los sistemas de visión artificial, el análisis de imágenes médicas, la mecánica computacional o la ingeniería aeroespacial.

La optimización de formas aerodinámicas (ASO, del inglés *Aerodynamic Shape Optimization*) es una forma efectiva de automatizar los procesos de diseño. Apoyándose en la dinámica de fluidos computacional (CFD, del inglés *Computational Fluid Dynamics*) y en recursos computacionales de alta eficiencia (HPC, del inglés *high-performance computing*), las técnicas de ASO permiten diseñar colas, alas o góndolas del motor [28]. Las técnicas de DL aplicadas a la dinámica de fluidos se emplean para el cálculo de parámetros como, por ejemplo, el coeficiente de resistencia (C_D) o la relación entre sustentación y avance o para el cálculo directo de los campos de flujo. Las aplicaciones de DL necesitan de una gran cantidad de datos, que, en el ámbito de la dinámica de fluidos, se extraen de los experimentos y simulaciones numéricas.

En este TFM, se aplica una arquitectura concreta de red neuronal convolucional (CNN, del inglés *Convolutional Neural Networks*) a una serie de situaciones fluido dinámicas de flujo turbulento. Se analiza la capacidad de las redes neuronales de predecir los campos de velocidades paralela al flujo y vertical y de presión, cuando se enfrentan a diferentes velocidades de entrada del fluido o a geometrías variables en tipo, tamaño y orientación.

El SW empleado consiste en una CPU Intel Xeon Gold 5120 para la obtención de todas las simulaciones CFD, una GPU NVIDIA Quadro RTX 6000 para el entrenamiento de la red neuronal, una CPU Intel Core i7-10750H para la generación de los diferentes códigos auxiliares. Los programas empleados son el Star-CCM+ para la generación de las simulaciones CFD y Python 3.9.6 y Matlab 2020b para el desarrollo de SW. Python 3.9.6 se ha ejecutado en el entorno Anaconda.

El CFD requiere de una capacidad computacional muy elevada. Para ciertas aplicaciones, se precisan resultados rápidos para la extracción de las primeras conclusiones

respecto a un problema determinado. Este TFM se focaliza en la aplicación de diferentes técnicas de DL que ahondan en la obtención de resultados rápidos lo más precisos posibles para evaluar los posibles caminos a seguir a la hora de resolver un problema concreto.

Alcance y objetivos

Cada simulación fluido dinámica puede presentar unas características muy diferentes respecto al resto. En un análisis lo más básico posible, se puede seleccionar el tipo de fluido y sus valores de velocidad, temperatura, si es un flujo laminar o turbulento... También la geometría a la que se enfrenta el fluido, modificando, por ejemplo, su forma, tamaño, posición u orientación. Todo ello unido a los elevados recursos computacionales que consumen las simulaciones CFD, implica la necesidad de búsqueda de nuevas técnicas que efectúen las simulaciones fluido dinámicas.

Debido a esta problemática, diferentes técnicas de DL se han postulado como alternativas eficientes a las simulaciones fluido dinámicas clásicas. Son muchas y variadas las aplicaciones en las que el DL y en concreto, las redes neuronales, han conseguido resultados con un coste computacional muy reducido y errores, con respecto a las simulaciones clásicas, lo suficientemente pequeños como para considerar al DL como una alternativa eficiente, para ciertas aplicaciones, a las simulaciones CFD.

Este TFM busca aplicar una arquitectura concreta de una CNN para resolver problemas fluido dinámicos con diferentes condiciones del fluido y las geometrías analizadas. En concreto, se buscan soluciones a los siguientes planteamientos:

En primer lugar, se busca predecir un número concreto de instantes futuros partiendo de un estado inicial. Este problema se enfoca en dos situaciones diferentes. En la primera, se consideran velocidades variables de entrada al dominio evaluado. La interacción del fluido se considera siempre sobre una geometría circular, invariable en tamaño y posición. Por tanto, el primer objetivo reside en analizar la capacidad de una CNN para predecir instantes futuros a partir de uno inicial, para velocidades del fluido de entrada al dominio variables. En la segunda situación, el objetivo reside en evaluar si esa misma CNN es capaz de predecir los instantes futuros para un conjunto de geometrías (círculo, elipse, rectángulo, cuadrado, triángulo y triángulo equilátero). Asimismo, se analiza el comportamiento para tamaños, orientaciones y posiciones variables de las geometrías mencionadas.

Las aplicaciones mencionadas en el párrafo anterior dependen en todo momento de un estado inicial a partir del cual se predicen los instantes futuros. Para solucionar este inconveniente, se emplea nuevamente la CNN diseñada para predecir el instante inicial necesario para las otras aplicaciones.

Estas tres aplicaciones persiguen el objetivo global de obtener resultados relativamente precisos que involucren un consumo reducido de recursos computacionales. Para

lograr este objetivo global, se tienen que alcanzar objetivos intermedios, expuestos a continuación.

- El primer objetivo reside en la generación de las simulaciones CFD, necesarias para el entrenamiento de las redes neuronales.
- Posteriormente, se ha de manejar correctamente los datos de las simulaciones CFD para que sirvan de salidas de la CNN. Asimismo, se han de seleccionar el resto de entradas y salidas necesarias.
- El siguiente objetivo consiste en la selección de la arquitectura de la CNN y los valores de los parámetros e hiper-parámetros de la red más convenientes.
- Por último, se han de manejar los resultados obtenidos para extraer las conclusiones pertinentes.

Antecedentes bibliográficos o estado del arte

3.1 Introducción a la Inteligencia Artificial y el Machine Learning

La Inteligencia Artificial (IA) tiene múltiples enfoques respecto a su definición y a lo que realmente representa. Existen definiciones que hacen referencia a los procesos mentales y al razonamiento y otras, se centran en la conducta. Asimismo, algunas de estas definiciones miden la capacidad de una máquina de igualar el comportamiento humano, mientras que otras se basan en la racionalidad, es decir, la facultad de obrar de forma correcta, según el conocimiento de la máquina [42]. Tomando sistemas que piensan como humanos, aparecen las siguientes definiciones:

- “El nuevo y excitante esfuerzo de hacer que los computadores piensen... máquinas con mentes, en el más sentido literal” [18].
- “[La automatización de] actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje...” [7].

Respecto al enfoque de sistemas que piensan racionalmente, dos definiciones de referencia son las siguientes:

- “El estudio de las facultades mentales mediante el uso de modelos computacionales” [8].
- “El estudio de los cálculos que hacen posible percibir, razonar y actuar” [51].

El tercer grupo se compone de definiciones que refieren a sistemas que actúan como humanos:

- “El arte de desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren de inteligencia” [25].
- “El estudio de como lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor” [40].

Por último, el grupo de definiciones que se centran en la actuación racional de los sistemas:

- “La Inteligencia Computacional es el estudio del diseño de agentes inteligentes” [37].
- “IA... está relacionada con conductas inteligentes en artefactos” [35].

Para Flach [11], el Aprendizaje Automático (ML, del inglés *Machine Learning*) es el estudio sistemático de los algoritmos y sistemas que mejoran su conocimiento o su desempeño con la experiencia. El ML se engloba dentro del campo de la IA. Torres [48] expresa de forma generalizada que el ML consiste en desarrollar para cada problema un algoritmo de predicción para un caso de uso particular.

Una red neuronal artificial (ANN, del inglés *Artificial Neural Network*) consiste en una capa de entrada de neuronas, una o más capas ocultas de neuronas, y una capa final de salida [49]. Cuando la red presenta una única capa oculta, se les denomina ANN superficiales, como la del diagrama de la figura 3.1, mientras que las ANN profundas tienen un gran número de capas ocultas. Es en este segundo caso donde entra en juego el concepto de DL, que es un subgrupo del ML, como muestra el diagrama de la figura 3.2.

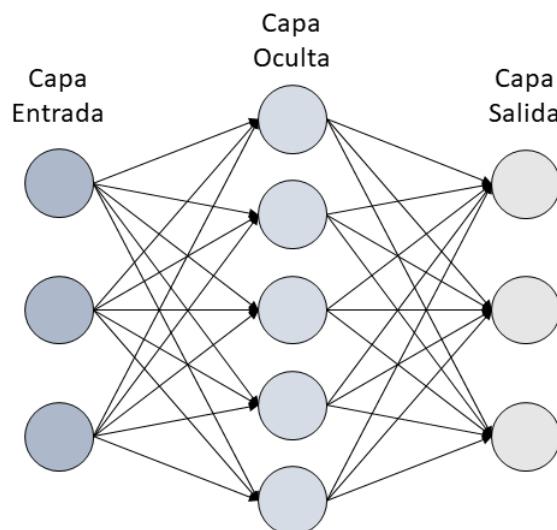


Figura 3.1: Diagrama que representa una ANN simple, que contiene las capas de entrada, oculta y de salida. Si tuviese varias capas ocultas sería una red de DL.

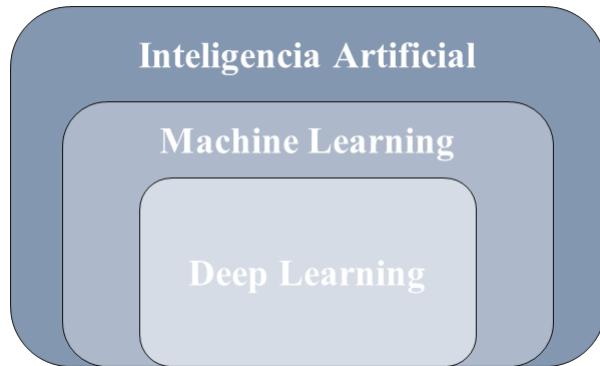


Figura 3.2: Diagrama que muestra que el DL se engloba dentro del ML y, que a su vez, este último, es un subconjunto de IA.

LeCun et al. [26] definen el DL como la herramienta que permite a los modelos computacionales compuestos de múltiples capas de procesamiento aprender representaciones de los datos con múltiples niveles de abstracción. El DL consigue descubrir estructuras complejas en grandes conjuntos de datos utilizando el algoritmo de retropropagación (del inglés *backpropagation*) para indicar cómo una máquina debería cambiar sus parámetros internos usados para el cálculo de la representación en cada capa desde la representación de la capa anterior. El concepto de DL junto con los modelos computacionales y los algoritmos que incorpora trata de imitar la arquitectura de las redes neuronales biológicas del cerebro. Cuando el cerebro recibe nueva información, la compara con los conocimientos previamente existentes e intenta darle sentido a este nuevo estímulo. El cerebro descifra la información asignando los elementos a varias categorías [20]. El organigrama de la figura 3.3 muestra de forma concisa cómo se organizan los diferentes pasos que se toman en las diversas disciplinas de IA. En el caso de DL, se aprecia la búsqueda de características simples en un primer lugar, para luego, ceñirse a las particularidades de mayor nivel de abstracción. Por último, se lleva a cabo el mapeo de esas características para obtener la salida deseada. A diferencia del DL, cuando se trata de una ANN simple, únicamente se buscan las características en una sola parte y, en el ML, la computadora simplemente se dedica al mapeo de las características.

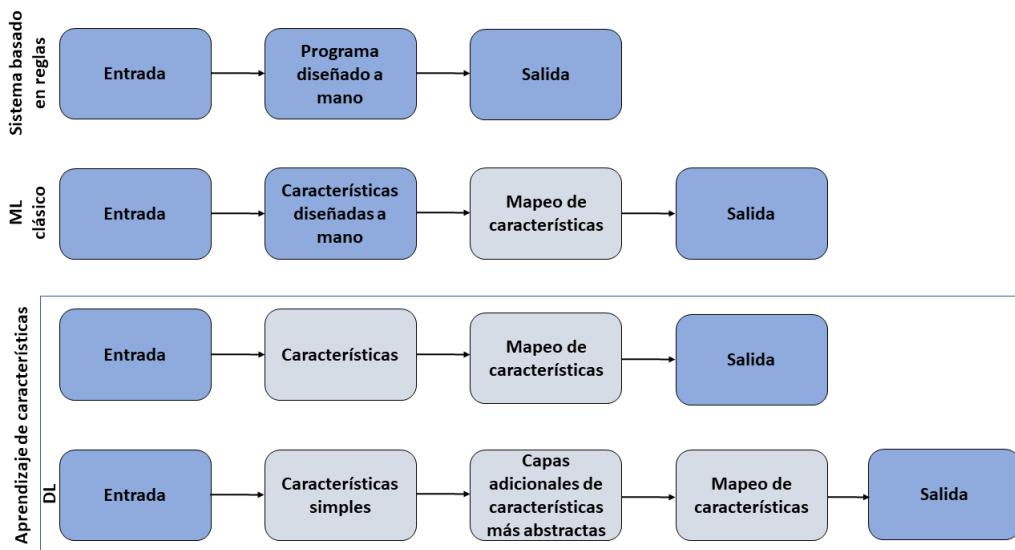


Figura 3.3: Organigrama que muestra cómo las diferentes partes de un sistema de IA se relacionan entre ellas en las diferentes disciplinas de la IA. Las cajas en gris muestran los componentes que son capaces de aprender de los datos.

Las neuronas en DL se pueden definir como los nodos por los que fluyen los datos. El funcionamiento de las neuronas es el siguiente:

- Reciben las señales de una o más entradas, provenientes de los datos de entrada de las salidas de las neuronas anteriores.
- En segundo lugar, realizan una serie de cálculos según la ecuación 3.1.
- Por último, expulsan el resultado a la salida.

$$y = \phi \left(\sum_{j=1}^n x_j \omega_j \right), \quad (3.1)$$

donde y es la salida de la neurona, ϕ representa el resultado de la función de activación, n es el número de entradas totales, x_j representa cada una de las entradas de la neurona y ω_j , cada uno de los pesos sinápticos. Para $j=0$, x_0 es igual a 1 y ω_0 es el sesgo (también se simboliza como b). La 3.4 muestra un diagrama de una neurona artificial.

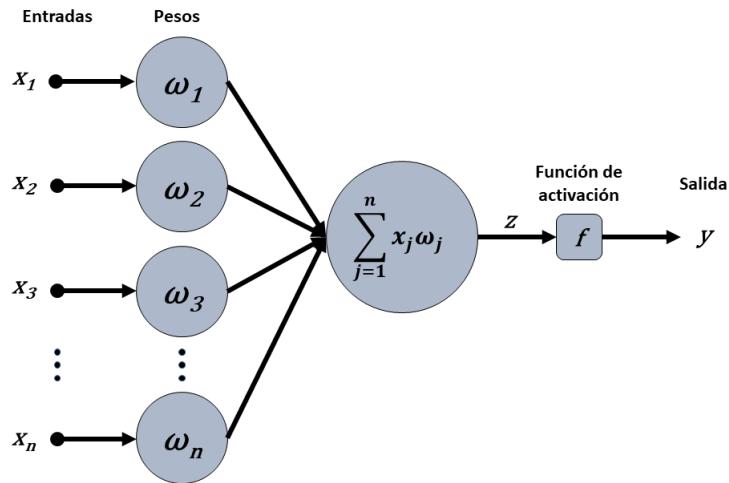


Figura 3.4: Diagrama de una neurona con sus respectivas entradas, pesos sinápticos, función de activación y salida.

3.1.1 Parámetros e hiper-parámetros

En los modelos de ML existen dos tipos de parámetros: los propiamente denominados parámetros, que se inicializan y actualizan durante el proceso de aprendizaje de la red neuronal; y los hiper-parámetros, que han de ser inicializados previamente al inicio del entrenamiento del modelo [24].

Dentro del primer grupo, el llamado parámetros, se encuentran los pesos sinápticos y el sesgo. Los pesos son números reales que indican la pendiente de la recta. El sesgo representa cómo de lejos están las predicciones con respecto del valor de salida deseado.

Los hiper-parámetros, según Yang y Shami [53], se pueden clasificar en dos categorías: aquellos que están relacionados con la construcción de un modelo de DL, llamada hiper-parámetros de diseño de modelo; y los hiper-parámetros de optimización, que se encargan de la optimización y el proceso de entrenamiento del modelo de DL.

Dentro del grupo de hiper-parámetros de diseño de modelo, aparecen algunos como el número de capas ocultas, el número de neuronas por capa, la función de coste, la función de activación y el tipo de optimizador. Como se ha mencionado previamente, las redes neuronales presentan como mínimo tres capas, donde dos de ellas son las de entrada y salida. El concepto de profundidad de una red neuronal depende de dos hiperparámetros, que son el número de capas ocultas y el número de neuronas por capa. Estos dos hiperparámetros dependen de la complejidad y tamaño de los conjuntos de datos del modelo. El tipo de función de coste empleada es un hiperparámetro que adquiere gran relevancia. Algunos ejemplos de función de coste son la entropía cruzada binaria (del inglés *binary cross-entropy*) para los

problemas de clasificación; la entropía cruzada multicategoría (del inglés *multi-class cross-entropy*) para los problemas de clasificación multivariable, y el error medio absoluto (MAE, del inglés *Mean Absolute Error*), también llamado *L1 loss*, el error medio cuadrático (MSE, del inglés *Mean Squared Error*), también *L2 loss*, y la raíz del error cuadrático medio (RMSE, del inglés *Root Mean Square Error*) para los problemas de regresión. Las expresiones de las funciones de coste vienen dadas por las expresiones 3.2, 3.3 y 3.4 respectivamente.

$$MAE = \frac{1}{m} \sum_{i=1}^m |h(x^{(i)}) - y^{(i)}|, \quad (3.2)$$

donde m es el número de muestras, $x^{(i)}$ es la muestra i del conjunto de datos, $h(x^{(i)})$ es la predicción para la muestra i e $y^{(i)}$ es la etiqueta de la muestra i.

$$MSE = \frac{1}{m} \sum_{i=1}^m |y^{(i)} - \hat{y}^{(i)}|, \quad (3.3)$$

donde m es el número de muestras, $y^{(i)}$ es la etiqueta de la muestra i e $\hat{y}^{(i)}$ es la etiqueta predecida para la muestra i.

$$RMSE = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})}, \quad (3.4)$$

donde m es el número de muestras, $x^{(i)}$ es la muestra i del conjunto de datos, $h(x^{(i)})$ es la predicción para la muestra i e $y^{(i)}$ es la etiqueta de la muestra i.

El hiperparámetro función de activación se emplea para la propagación de la salida de una neurona hacia delante. Algunos ejemplos de funciones de activación son la función lineal, la sigmoide, la tangente hiperbólica (\tanh), la softmax, la unidad lineal rectificada (ReLU, del inglés *rectified linear unit*) o la *softsign*. Las expresiones de estas funciones vienen dadas por las ecuaciones 3.5, 3.6, 3.7, 3.8, 3.9 y 3.10 respectivamente.

$$f(x) = x, \quad (3.5)$$

$$f(x) = \frac{1}{1 + e^{-x}}, \quad (3.6)$$

$$f(x) = \tanh(x), \quad (3.7)$$

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \text{ para } i = 1, \dots, n, \quad (3.8)$$

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0, \end{cases} \quad (3.9)$$

$$f(x) = \frac{x}{1 + |x|}. \quad (3.10)$$

Por último, el tipo de optimizador a emplear. La optimización hace referencia a la tarea de minimizar o maximizar una función $f(x)$ alterando el valor de x . La función $f(x)$ a minimizar o maximizar es la previamente explicada función de coste [14]. Algunos ejemplos típicos de optimizadores son el gradiente de descenso estocástico (SGD, del inglés *stochastic gradient descent*), la estimación de momento adaptativo (ADAM, del inglés *adaptive moment estimation*), la propagación de la raíz cuadrática media (RMSprop, del inglés *root mean square propagation*).

Dentro del segundo grupo, el más importante es la tasa de aprendizaje (del inglés *learning rate*), que determina el tamaño de paso en cada iteración para permitir que la función de coste converja. Si la tasa de aprendizaje es muy grande, el proceso de aprendizaje se acelera, pero puede provocar que el gradiente oscile sobre un mínimo local o que no llegue a converger; mientras que una tasa de aprendizaje muy pequeña provoca un incremento del tiempo de aprendizaje. El *dropout* es una técnica que previene el sobreajuste (del inglés *overfitting*). Se encarga de eliminar algunas neuronas y conexiones entre neuronas de forma temporal de una red neuronal [45]. El *mini-batch size* es un hiperparámetro que representa el número de muestras procesadas previas a la actualización del modelo y el número de épocas indica el número de veces que se itera sobre el conjunto de datos completos. Un último hiperparámetro sería el *early stopping*, que sirve para finalizar el entrenamiento del modelo cuando el error de validación no cambia durante una serie de épocas consecutivas.

3.1.2 Tipos y características de ANN

En este apartado, se describen algunas de las estructuras de DL más básicas, en concreto las relacionadas con la aplicación del DL a la dinámica de fluidos. La principal diferencia entre ellas radica en la arquitectura de la red, es decir, la forma en la que las neuronas se organizan dentro de la red. Según la arquitectura, la siguiente lista muestra los tres tipos principales:

- Perceptrón multicapa (MLP, del inglés *Multi-Layer Perceptron*): es un sistema de neuronas simples interconectadas. Se trata de una representación del modelo de mapeo no lineal de un vector de entradas en uno de salidas [12]. Las capas están densamente conectadas entre sí, es decir, cada nodo está conectado con

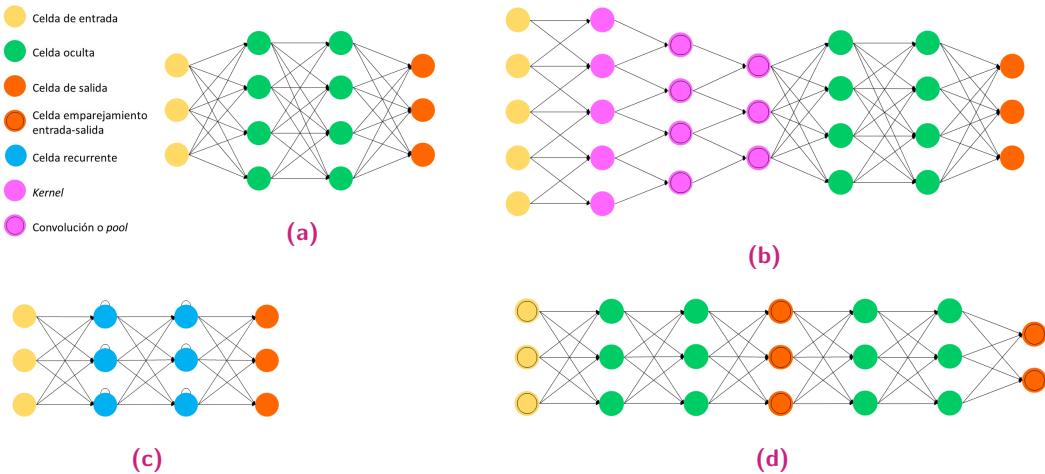


Figura 3.5: Diagramas de las arquitecturas de los principales tipos de redes neuronales: a) MLP, b) CNN, c) RNN y d) GAN.

todas las neuronas de la capa anterior y posterior. La figura 3.5a muestra un diagrama de un MLP con dos capas ocultas.

- Las CNN son un tipo de ANN que están compuestas como mínimo por una capa convolucional. Suelen llevar a cabo operaciones de convolución, *pooling*, tener capas densamente conectadas (del inglés *fully-connected*). Su ventaja principal con respecto a las ANN tradicionales reside en la disminución del número de parámetros necesarios [3]. La figura 3.5b muestra un ejemplo de una arquitectura de red de tipo CNN.
- Las redes neuronales recurrentes (RNN, del inglés *Recurrent Neural Networks*) presentan un estado interno creado con los datos de entrada ya empleados por la red. Posteriormente, como salida, proporciona una combinación entre el estado interno y la nueva entrada. A su vez, el estado interno se modifica en cada iteración para los nuevos datos entrantes. Se emplean en tareas que requieren datos secuenciales [43]. En la figura 3.5c se muestra un esquema de una RNN.
- Las red neuronales generativas adversarias (GAN, del inglés *Generative Adversarial Networks*) están formadas por dos modelos: un modelo generativo, G, que captura la distribución de los datos, y un modelo discriminativo D que estima la probabilidad de que una muestra provenga del entrenamiento de los datos antes que de G. El procedimiento de entrenamiento de G es maximizar la probabilidad de D de fallar [15]. En la figura 3.5d se visualiza un ejemplo de una estructura de tipo GAN.

Las capas principales de una CNN son las capas de entrada y salida y las capas de convolución y *pooling*.

La operación de convolución consiste en detectar y aprender determinados patrones locales en ventanas de dos dimensiones dentro de una imagen. Una vez reconocido un patrón específico en una imagen, se puede identificar esa característica en cualquier parte de la misma. Asimismo, una capa convolucional inicial es capaz de aprender patrones sencillos como aristas, colores o líneas. Posteriormente, otra capa convolucional se vale de esos patrones para aprender jerarquías de patrones. De esta forma, haciendo uso de varias capas de convolución, la red neuronal llega a comprender patrones muy complejos. La figura 3.6 muestra un diagrama que representa la capa de convolución.

Las capas de *pooling* se emplean comúnmente tras la serie de capas convolucionales. A rasgos generales, la operación de *pooling* realiza una simplificación de la información recogida por la capa convolucional y crea una versión condensada de la misma. Para ello, esta capa divide en pequeñas regiones de igual tamaño la capa convolucional logrando reducir el número de conexiones para las siguientes capas. No realiza ningún aprendizaje por sí misma, sino que reduce el número de parámetros a aprender durante las siguientes capas. Como se muestra en la figura 3.7, existen diferentes tipos de *pooling*. Los más típicos son el *max-pooling*, en el que se toma el mayor valor de la región, y el *average-pooling*, donde se toma el valor medio de la región.

Tanto la capa de convolución como la de *pooling* tienen su respectiva operación inversa. Para las capas de convolución, existen las de convolución traspuesta, también denominadas deconvoluciones. Para las capas de *pooling* se encuentran las capas de *unpooling*.

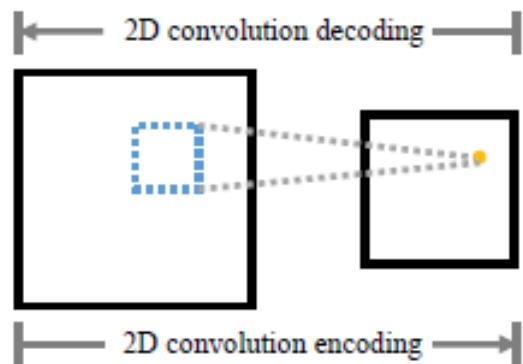


Figura 3.6: Diagrama de una convolución [16].

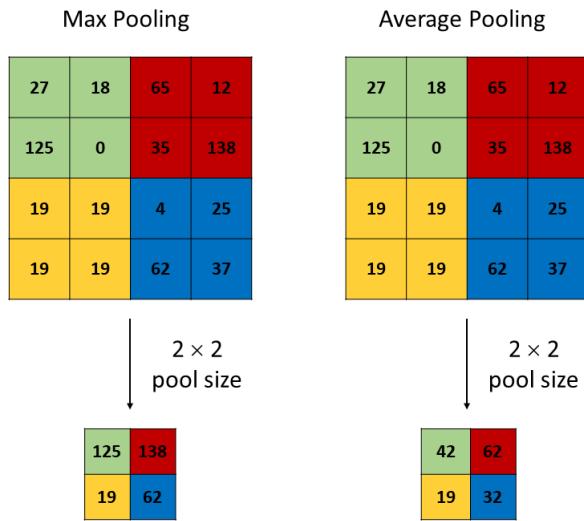


Figura 3.7: Ejemplos de operaciones de *pooling*.

3.2 Dinámica de fluidos Computacional

Las características físicas de cualquier fluido siguen tres leyes fundamentales: la conservación de la masa, $F=m \cdot a$ (segunda ley de Newton) y la conservación de la energía. Estos principios fundamentales se pueden expresar en forma de ecuaciones matemáticas, generalmente a través de ecuaciones diferenciales. El CFD se basa en sustituir soluciones analíticas de ecuaciones en forma cerrada por números. El producto final del CFD es una gran cantidad de números que proporcionan una descripción cuantitativa del problema.

Las soluciones de las simulaciones en CFD requieren generalmente la manipulación reiterada de miles, o incluso millones, de números. Debido a ello, la capacidad computacional necesaria y la demanda de memoria es muy elevada. A pesar de los avances exponenciales que se llevan produciendo durante las últimas décadas en la informática, que habilitan la resolución de problemas con CFD cada vez más detallados y sofisticados, sigue suponiendo una limitación para el desarrollo de productos en un rango variado de aplicaciones, como pueden ser la optimización del diseño aerodinámico y la interacción fluido-estructura [50]. Este hecho, unido al crecimiento de la inteligencia artificial en los últimos años, ha resultado en que muchos autores empleen técnicas de DL para obtener aproximaciones de los resultados de las simulaciones CFD, en los casos en los que se necesitan geometrías muy complejas o mallados muy finos. Otro inconveniente de las aplicaciones CFD es su clara dependencia de la capacidad del usuario para la generación del mallado y el modelo de turbulencia.

Para resolver estos problemas, el DL ha sido un área de estudio enfocada a las simulaciones CFD. Por ejemplo, Tao y Sun [46], Zhang et al. [54] y Yan et al.

[52] alcanzaron la optimización aerodinámica, mejorando la eficiencia de varias geometrías con DL. Dentro de las técnicas de DL aplicadas en el ámbito de la computación de fluidos, destacan dos enfoques diferentes. El primero tiene como objetivo la reducción tiempo de computación de las simulaciones en las que se emplea mallas gruesas. Por ejemplo, Bao et al. [5] aplicaron un enfoque en las características físicas del sistema para mejorar el modelado y la capacidad de simulación de una malla gruesa, y Hanna et al. [17] diseñaron un algoritmo de DL para predecir y mejorar el error de los resultados obtenidos en una malla gruesa.

El segundo enfoque implica el cálculo directo de las características deseadas del fluido. Guo et al [16] crearon una CNN que predice los campos de flujo estacionarios alrededor de objetos sólidos, logrando resultados no tan precisos, pero con predicciones muy rápidas. Ribeiro et al [39], tomando la CNN de Guo et al [16] como referencia, diseñaron una CNN muy precisa para predecir los campos de velocidad y presión de fluidos estacionarios alrededor obstáculos con formas simples, reduciendo el coste computacional en torno a 3 y 5 órdenes de magnitud con respecto a las simulaciones CFD. Kashefi et al [21] diseñaron una ANN para obtener los mismos campos que los estudios mencionados previamente. Los campos se obtuvieron con ligeras modificaciones de las geometrías, ya que este es un parámetro esencial a la hora de optimizar los diseños.

Asimismo, existen otros estudios donde se llevan a cabo predicciones de características del flujo más específicas. Por ejemplo, Ling et al [29] utilizó un enfoque de DL para el modelado de las turbulencias a través de las ecuaciones de Navier-Stokes promediadas por Reynolds (RANS, del inglés Reynolds-Averaged Navier-Stokes). De esta forma, modeló el tensor de anisotropía de Reynolds a través de una red neuronal profunda (DNN, del inglés Deep Neural Network), consiguiendo una mejora importante comparada con los resultados obtenidos en las simulaciones CFD. Lee and You [27] predijeron el desprendimiento de vórtices laminares no estacionarios en un cilindro circular mediante una GAN. Liu et al. [30] y Deng et al. [9], diseñaron métodos para la detección de impactos y vórtices respectivamente mediante técnicas de DL.

Los estudios anteriores están centrados en la predicción de flujos laminares. Este tipo de flujo es más fácil de predecir para una DNN debido a que las partículas del fluido se desplazan de forma paralela en láminas ordenadas mientras que, en los flujos turbulentos, las partículas presentan un movimiento aleatorio y caótico. La arquitectura más habitual en los estudios que contemplan los flujos turbulentos son las CNN. Especialmente con una estructura tipo *autoencoder*, en la que las entradas a la red se reducen a una representación geométrica latente (LGR, del inglés *Latent Geometry Representation*). El LGR se trata de una representación de las características básicas de las entradas iniciales a la red. De esta forma, el *encoder* de la CNN puede mapear de forma más sencilla las características del fluido deseadas. Asimismo, la

estructura U-net creada por Ronneberger et al. [41], en la que se aplica la estructura *autoencoder* conectando cada bloque codificador a su respectivo decodificador. Estos autores emplean dicha estructura para la segmentación de imágenes biomédicas. No obstante, los estudios comentados a continuación demuestran la flexibilidad de aplicación de esta arquitectura a la dinámica de fluidos para flujos turbulentos. Por ejemplo, Fang et al. [10] predijeron el tensor de esfuerzos de Reynolds para flujo turbulento con una DNN, mejorando los resultados obtenidos con el modelo lineal y con el modelo de viscosidad de Eddy cuadrática. Thuerey et al. [47] utiliza una CNN con arquitectura U-net para aproximar los campos de velocidad y presión del modelo de turbulencia Spalart-Allmaras basado en las ecuaciones RANS para un perfil aerodinámico. Abucide-Armas et al. [1] logra tasas de error reducidas en la predicción de los campos de velocidad y presión, a través de una CNN, para fluidos turbulentos y velocidades de entrada al dominio variables. Además, añade una técnica de data-augmentation novedosa aplicando la semejanza dinámica y Portal-Porras et al. [38] desarrolla varias estructuras basadas en CNN para predecir los campos de velocidad para flujos turbulentos.

A pesar de la existencia de algunos estudios que analizan dominios tridimensionales, como los estudios Guo et al. [16] y el de Nowruzi et al. [36], la gran mayoría de estudios de este tipo se centran en geometrías bidimensionales. Esto se debe principalmente a los limitados recursos computacionales existentes para las geometrías 3D [34]. Para evitar este problema, Mohan et al. [34] desarrollaron una infraestructura basada en DL que reduce la geometría para analizar posteriormente las características del fluido.

En la mayoría de los estudios mencionados previamente no se tiene en cuenta la evolución temporal de las características del fluido. En DL, se emplean RNN para los casos en los que las entradas y salidas de la red dependen de instantes anteriores. Agostini [2] predice la evolución temporal de la velocidad paralela al flujo con un modelo *autoencoder*. Por otro lado, Maulik et al. [33], Gonzalez y Balajewicz [13] y King et al. [22] predijeron algunas propiedades del fluido con un enfoque basado en la evolución temporal de las mismas.

En este trabajo, se utiliza una CNN basada en la arquitectura U-net con una estructura de tipo *autoencoder*. La CNN propuesta es entrenada con los datos de los campos de velocidad horizontal y vertical y de presión obtenidos con técnicas de CFD. El objetivo es aprovechar estos datos para entrenar la CNN para conseguir predicciones de los instantes futuros con tasas de error pequeñas, teniendo en cuenta la dependencia del estado actual de las características de un fluido con respecto al estado anterior. En cuanto al testeo de la red, se toma un instante inicial aleatorio de los datos de CFD a partir del cual se calculan las predicciones de los siguientes 10 instantes. Cada resultado obtenido es comparado con su correspondiente instante calculado con CFD para obtener la precisión de la red.

Desarrollo de la solución

El trabajo se focaliza en la utilización de técnicas de DL para simulaciones fluido dinámicas, principalmente para flujos turbulentos. En primer lugar, se estudia la capacidad de una CNN de predecir instantes futuros de los campos de velocidades paralela al flujo y vertical y de presión para velocidades de entrada al dominio computacional variables. El segundo caso se enfoca en la predicción de dichos campos variando las geometrías analizadas y sus características. Asimismo, se genera un modelo neuronal para la predicción del primer instante a partir del cual se generan los instantes futuros.

4.1 Geometría circular con velocidad de entrada variable

4.1.1 Simulaciones numéricas

En este trabajo, se estudian las oscilaciones periódicas a largo plazo en un flujo turbulento. Por ello, las simulaciones se llevan a cabo en base a las ecuaciones inestables de Navier-Stokes promediadas por Reynolds (URANS, del inglés *Unsteady Reynolds-Averaged Navier-Stokes*). Las ecuaciones URANS se obtienen mediante el siguiente proceso:

Las ecuaciones de Navier-Stokes para un flujo incompresible se filtran a lo largo del tiempo de acuerdo a la ecuación 4.1.

$$\frac{\partial \langle u_i \rangle}{\partial t} + \frac{\partial}{\partial x_j} (\langle u_j u_i \rangle) = -\frac{1}{\rho} \frac{\partial \langle p \rangle}{\partial x_i} + v \frac{\partial^2 \langle u_i \rangle}{\partial x_k^2}. \quad (4.1)$$

Después, se introduce en la ecuación 4.1 el tensor de esfuerzos turbulentos, dado por la ecuación 4.2.

$$\tau_{ij} = \langle u_i \rangle \langle u_j \rangle - \langle u_j u_i \rangle. \quad (4.2)$$

La ecuación URANS final queda:

$$\frac{\partial \langle u_i \rangle}{\partial t} + \frac{\partial}{\partial x_j} (\langle u_i \rangle \langle u_j \rangle) = -\frac{1}{\rho} \frac{\partial \langle \tau_{ij} \rangle}{\partial x_i} + \frac{\partial \langle p \rangle}{\partial x_j} + v \frac{\partial^2 \langle u_i \rangle}{\partial x_k^2}. \quad (4.3)$$

Explicaciones más detalladas sobre las ecuaciones URANS se proporcionan, por ejemplo, en el estudio de Iaccarino et al. [19].

Star-CCM+ [44] ha sido empleado para la generación de las simulaciones CFD. Cada una de las simulaciones es de un 1 segundo con una frecuencia de muestreo de $2 \cdot 10^{-4}$. Esto proporciona un total de 5000 muestras por cada simulación. Para cada muestra, se han calculado los campos de las velocidades paralela al flujo y vertical y el campo de presión. Estos datos se guardan en archivos de tipo CSV, donde cada uno de ellos contiene los valores de presión y velocidades horizontal y vertical para cada punto del dominio dado por las coordenadas X e Y. Los resultados de las simulaciones CFD son los empleados para el entrenamiento, validación y testeo de la CNN. El dominio considerado es un rectángulo bidimensional con un cilindro circular en el medio del dominio, ver Aramendia et al. [4]. El flujo avanza desde la entrada al dominio, situada en la parte izquierda del mismo, hasta la salida, situada en el extremo derecho del dominio. Los extremos superior e inferior y el cilindro circular no tienen asignadas condiciones de deslizamiento. El diámetro del círculo (D) es de 10 mm, y su centro se encuentra a una distancia 5D de la entrada al dominio y de las paredes deslizantes. El tamaño del dominio es un rectángulo de dimensiones iguales a 100×256 mm. La vista detallada del dominio computacional y sus dimensiones se proporciona en la figura 4.1.

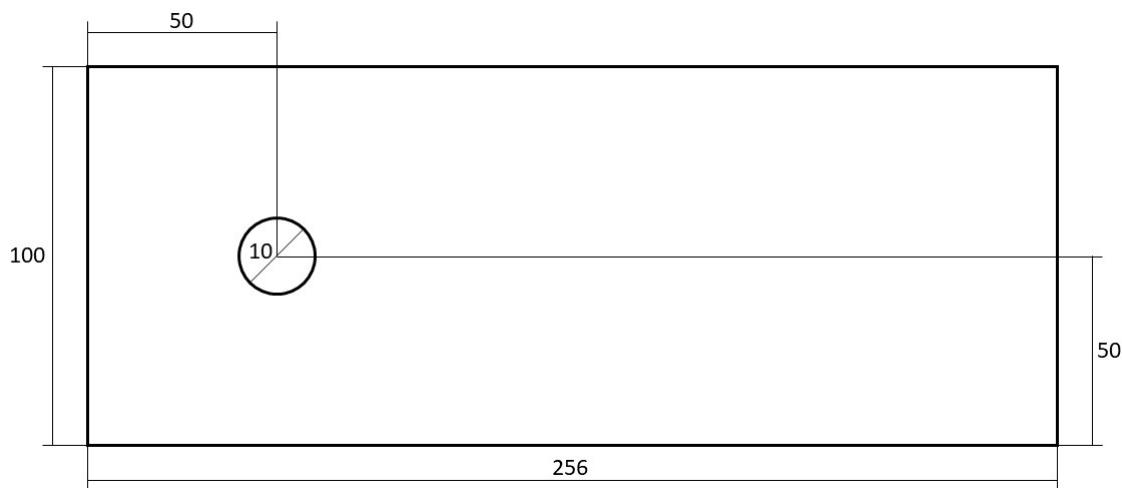


Figura 4.1: Dominio computacional de las simulaciones con velocidad de entrada variable (sin escalar).

Dentro del dominio descrito en el párrafo anterior, se ha construido un malla compuesto de celdas poliédricas. La mayoría de celdas se ubican en la zona posterior al obstáculo en el sentido de movimiento del flujo, buscando una mejor captura de los vórtices situados detrás del círculo. Asimismo, un control volumétrico ha sido diseñado para refinar la malla alrededor del obstáculo para poder mantener el valor y^+ por debajo de 1. La figura 4.2 muestra el malla descrito.

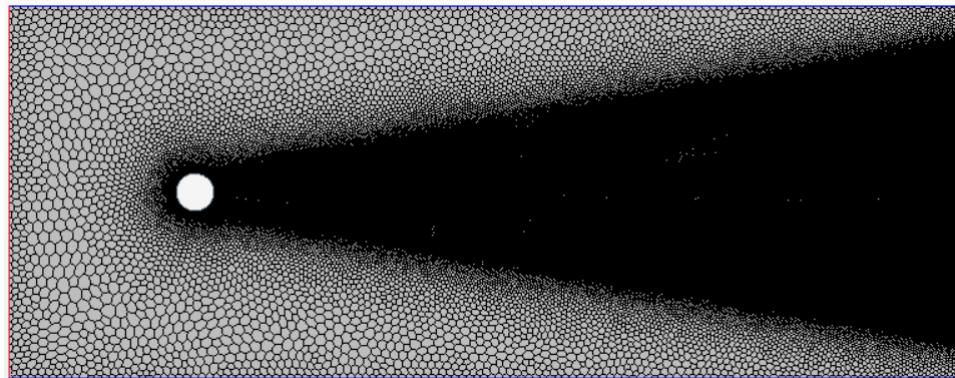


Figura 4.2: Distribución del mallado alrededor de la geometría circular.

Con respecto al fluido, se considera un flujo de aire incompresible turbulento inestable. La densidad (ρ) del fluido elegido es $1,18415 \text{ kg/m}^3$ y su viscosidad dinámica (μ) es igual a $1,85508 \cdot 10^{-5} \text{ Pa}\cdot\text{s}$. Ambas magnitudes se asumen como constantes. La velocidad en la entrada (u_∞) varía entre 5 m/s y 25 m/s , con un intervalo de 5 m/s entre cada simulación. Dando un total de 5 simulaciones. El número de Reynolds de las simulaciones varía entre 3200 y 16000 , dependiendo del caso y de acuerdo a la ecuación 4.4.

$$Re = u_\infty D / \nu, \quad (4.4)$$

donde u_∞ indica la velocidad del fluido, D el diámetro del círculo y ν , la viscosidad cinemática del fluido. Para que el tamaño de los datos de CFD encajen con el tamaño de las entradas de la CNN, se interpolan en una red de 79×172 .

4.1.2 Red neuronal convolucional

La red neuronal escogida para este estudio es una CNN. Estas redes, como se ha descrito previamente son muy eficaces en la obtención de patrones de imágenes a nivel de píxel a partir de unos determinados datos de entrada. Las imágenes digitales son, en esencia, matrices. Los campos de velocidad y presión también son matrices; por tanto, la ventaja de encontrar patrones es la que permite la predicción de los campos analizados.

En este caso, el problema ha sido analizado con un enfoque basado en el tiempo. Este enfoque se justifica en que, en la dinámica de fluidos, el estado de un fluido en un instante de tiempo, t , es fuertemente dependiente de su estado anterior, $t-1$. Es decir, el transitorio de los estados de un fluido adquiere una gran importancia a la hora de analizar las características del mismo. Este concepto se implementa en la CNN mediante la adición de una nueva entrada a la red neuronal que contenga el estado del fluido analizado en el instante $t-1$.

La arquitectura considerada es una U-Net propuesta por Ronneberger et al. [41] para la segmentación de imágenes médicas. Ribeiro et al. [39] demostraron que este tipo

de arquitectura es perfectamente aplicable a la predicción de los campos de velocidad y presión de un fluido. La ventaja de la U-Net reside en que reproduce resultados de mayor precisión gracias a las conexiones directas entre las características de la geometría codificada y las capas decodificadoras. Esta arquitectura consiste en una serie de capas de convolución que llevan a cabo una compresión de la información de las geometrías para obtener una versión condensada de los datos, provocando que la CNN sea capaz de detectar los patrones destacables existentes en el conjunto de datos con una mayor sencillez. Esta versión reducida de las geometrías recibe el nombre de LGR. Posteriormente, por medio de capas de deconvolución se realiza el mapeo entre la LGR y las variables de interés, en este caso, u_x , u_y y p . Estas operaciones de deconvolución transcurren hasta que el LGR adquiere el tamaño del CFD original. Además, el número de canales de salida es igual al número de variables de interés. En el caso en estudio, a pesar de existir tres variables de interés, únicamente existe un canal de salida debido a que cada una de las tres variables se estudia por separado. La adición de un enfoque temporal provoca que un análisis de las tres variables de forma simultánea sea erróneo, ya que se mezclarían informaciones de las variables de interés entre ellas.

La arquitectura de la red presenta una estructura de tipo *autoencoder*, puesto que las entradas de la red se codifican hasta proporcionar el LGR que, posteriormente, se decodifica hasta obtener las salidas deseadas. La figura 4.3 representa de manera gráfica la arquitectura U-Net con una estructura de tipo *autoencoder*.

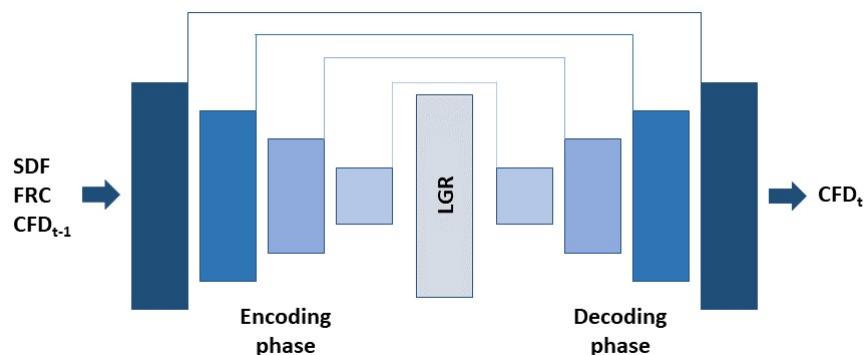


Figura 4.3: Arquitectura U-Net.

A continuación, se describe en profundidad la arquitectura del *autoencoder*. La parte de codificación está constituida por 4 bloques codificadores, donde cada uno de ellos está compuesto por 3 capas de convolución. Los parámetros de las capas de convolución son los siguientes:

- Filtros: 8, 16, 32 y 32 para cada bloque codificador respectivamente.

- Tamaños de las ventanas: 3×3 , 5×5 o 7×7 , dependiendo de cada uno de los modelos entrenados.

Cada bloque codificador tiene el siguiente orden en sus capas:

- En primer lugar, una capa de convolución con la entrada igual al número de filtros del anterior bloque y la salida igual al número de filtros actual más una capa ReLU. En el primer bloque codificador, la primera subcapa contiene 3 filtros en su entrada, que corresponde con el número de entradas a la red neuronal (la función de distancia con signo (SDF, del inglés *Signed Distance Function*), el canal de la región de flujo (FRC, del inglés *Flow Region Channel*) y CFD_{t-1}).
- Posteriormente, una operación de convolución con igual número de filtros en la entrada y la salida más una capa ReLU.
- Por último, se lleva a cabo el mismo proceso que el anterior punto añadiendo, tras la capa ReLU , una capa de *maxpooling*.

La parte decodificadora está formada por 4 bloques decodificadores formados por sucesivas capas de deconvolución, *maxunpooling* y ReLU para mapear la LGR en la salida deseada en cada caso. Las características de las capas de deconvolución son las mismas que las de convolución de la parte codificadora a excepción de los filtros que van en orden inverso para obtener el mapeo de la LGR de forma correcta. El orden de las capas en cada uno de los decodificadores es el siguiente:

- Operación de deconvolución con la entrada igual al doble de filtros correspondiente y la salida igual al número de filtros correspondiente más una capa ReLU.
- Una capa de deconvolución con entrada y salida igual al número de filtros más una capa ReLU.
- Una capa de maxunpooling más una capa de deconvolución con entrada igual al número de filtros correspondiente y salida igual al número de filtros del bloque siguiente más una capa ReLU. En el ultimo bloque decodificador, la capa de *maxunpooling* no existe y la deconvolución tiene una única salida, la variables u_x , u_y o p dependiendo del caso a evaluar.

La figura 4.4 representa la arquitectura de la U-Net de forma detallada.

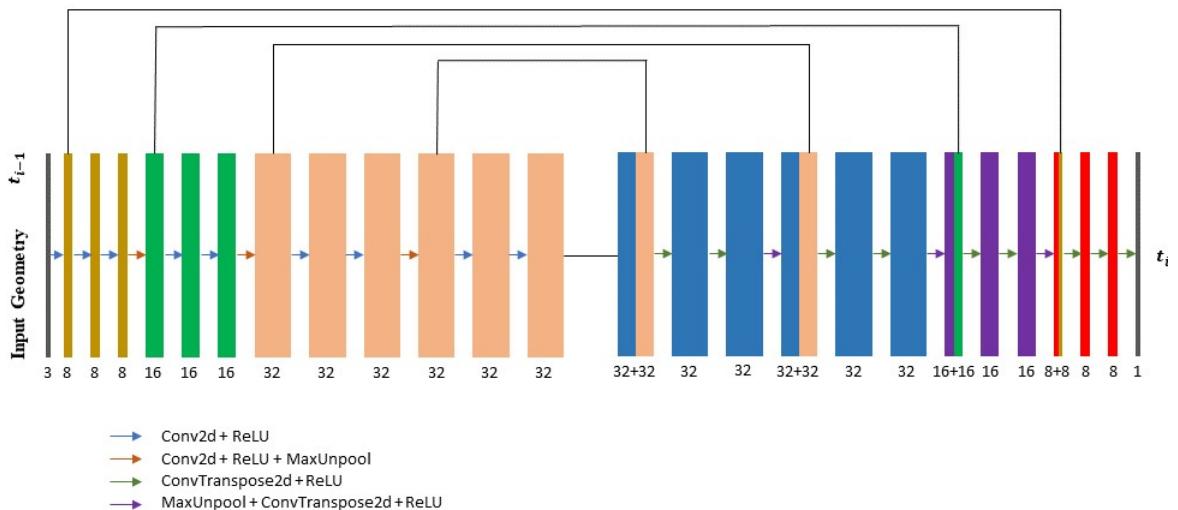


Figura 4.4: Arquitectura detallada de la CNN.

4.1.3 Entradas de la red neuronal

Para esta red neuronal se consideran 3 entradas. El primer par corresponde con la SDF y el FRC. La entrada SDF hace referencia a la forma geométrica tanto del obstáculo como de la malla mientras que en el FRC se indican las características del fluido a lo largo de las diferentes posiciones de la malla. La entrada restante representa la muestra de la variable analizada en el instante anterior (CFD_{t-1}).

La capa de entrada FRC es un canal de múltiples categorías que presenta información sobre las condiciones de contorno del dominio. Se trata de una matriz donde, a través de 5 categorías diferentes, se indican las zonas por las que puede transcurrir un fluido. Las categorías son las siguientes:

- 0 para el obstáculo.
- 1 para la región de flujo libre.
- 2 para las paredes superior e inferior que cumplen la hipótesis de no deslizamiento.
- 3 para la condición de velocidad constante de entrada.
- 4 para la condición límite de flujo cero.

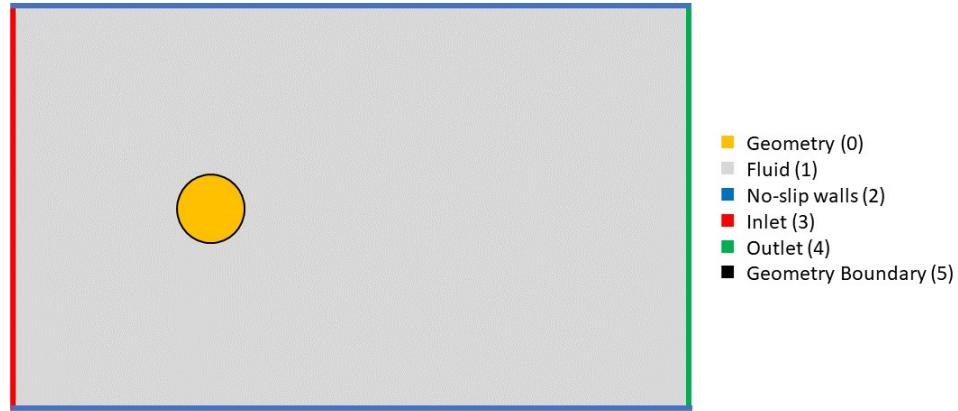


Figura 4.5: Diagrama representativo del FRC.

La entrada SDF consiste en una función matemática que mide la distancia relativa entre cualquier punto de la rejilla y cualquier punto del contorno de una figura geométrica cerrada. Según Guo et al. [16], la SDF proporciona una representación universal para diferentes figuras geométricas y funciona eficientemente con las redes neuronales. Otro tipo de representaciones de las geometrías como los límites o los parámetros geométricos no son efectivas para las redes neuronales. La expresión matemática de esta función viene dada por:

$$SDF(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in \Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega^c, \end{cases} \quad (4.5)$$

donde Ω es un subconjunto de un espacio métrico, X , con métrica, d , y $\partial\Omega$ es el límite de Ω . Para cada $x \in X$:

$$d(x, \partial\Omega) := \inf_{y \in \partial\Omega} d(x, y), \quad (4.6)$$

donde \inf es el ínfimo. A las posiciones interiores del obstáculo (Ω^c) se le asignan valores de distancia negativos. La figura 4.6 muestra la SFD generada para este estudio con un código de MATLAB [32]. Este permite elegir la posición y tamaño del círculo y el tamaño de la rejilla.

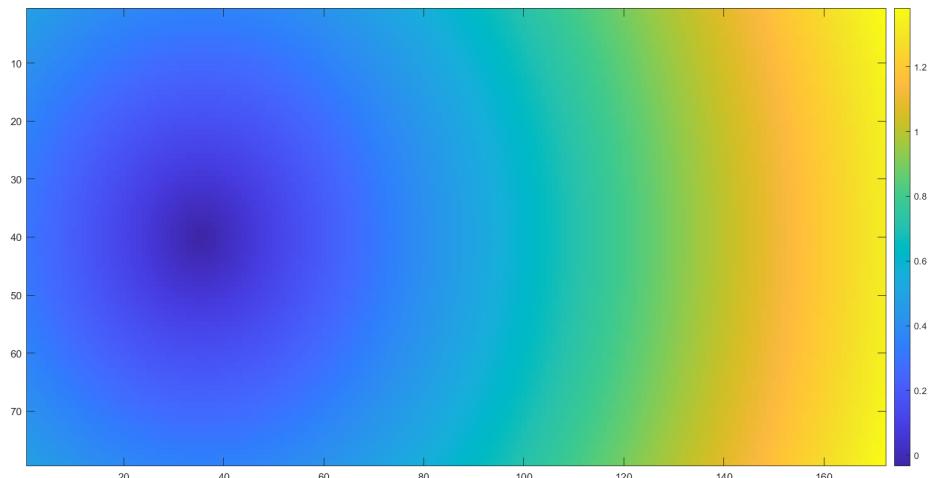


Figura 4.6: SDF de un círculo.

Durante el entrenamiento de la CNN, los resultados obtenidos de las simulaciones CFD se utilizan como la tercera entrada. La muestra del instante $t-1$ con respecto a la muestra de CFD analizada en ese momento es el introducido como entrada. Esto permite a la CNN aprender los patrones existentes durante los transitorios de un estado a otro. A la hora del testeo de la CNN, el resultado predicho en el instante anterior ($t-1$) se emplea como referencia para la nueva predicción; es decir, en el testeo las entradas son la SDF, el FRC y la predicción en $t-1$.

4.1.4 Parámetros del entrenamiento

El optimizador escogido para el entrenamiento de la red es el AdamW, dado por el algoritmo [1]. Es un algoritmo basado en el algoritmo Adam, que actualiza el vector de gradiente y el gradiente cuadrático utilizando la media móvil exponencial [23]. Los coeficientes β_1 y β_2 son los factores de olvido (*forgetting factors*) para los gradientes y los segundos momentos de los gradientes, respectivamente. El valor proporcionado a ambos factores de olvido es de 0,5. El optimizador AdamW es una versión actualizada del optimizador Adam, que mejora la regularización desacoplando el decaimiento de los pesos de la actualización basada en gradiente [31].

Algorithm 1 Algoritmo AdamW

Entradas: $\gamma(\text{lr})$, β_1 , $\text{beta}_2(\text{betas})$, $\theta_0(\text{params})$, $f(\theta)$ (objetivo), $\epsilon\lambda(\text{weight decay})$, amsgrad , maximize

Iniciar: $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment)

for $t=1$ to ... **do**

if maximize :

$g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$

else

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ $\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\theta_{t-1}$

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)g_t$

$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$

$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$

$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$

if amsgrad

$\widehat{v}_t^{max} \leftarrow \max(\widehat{v}_t^{max}, \widehat{v}_t)$

$\theta_t \leftarrow \theta_t - \gamma\widehat{m}_t / (\sqrt{\widehat{v}_t^{max}} + \epsilon)$

else

$\theta_t \leftarrow \theta_t - \gamma\widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$

end

return θ_t

Los valores de los hiper-parámetros empleados en el entrenamiento de los diferentes modelos se muestran en la tabla 4.1.

Parámetro	Valores		
Arquitectura	[8, 16, 32, 32]		
Tamaño de las ventanas	3	5	7
Función de coste	L1-norm		
Tasa de aprendizaje (lr)	0,001	0,0001	
Decaimiento de pesos	0,005		
Batch size	32	64	
Ratio entrenamiento-test	0,7-0,3		
Nº de épocas	1000		

Tabla 4.1: Conjunto de valores seleccionados para la búsqueda de los hiper-parámetros adecuados para la red que predice los instantes futuros para velocidades de entrada al dominio variables.

4.2 Geometrías variables

4.2.1 Simulaciones numéricas para geometrías variables

Nuevamente se emplea Star-CCM+ [44] para simular los campos de velocidades paralela al flujo y vertical y de presión. Se consideran las siguientes geometrías: círculo, elipse, cuadrado, rectángulo, triángulo y triángulo equilátero. La duración de cada una de las simulaciones es de 0,1 segundos con una frecuencia de muestreo de $2 \cdot 10^{-4}$. Se generan 500 muestras para cada geometría, proporcionando un total de 3000 muestras. En cada muestra se calculan los valores de velocidad y presión para cada punto del dominio (X, Y). Estos valores se almacenan en un archivo con formato CSV. El dominio considerado es un rectángulo bidimensional, donde la geometría y las paredes se consideran como superficies no deslizantes. El tamaño del dominio es de 128×256 mm. La figura 4.7 representa el dominio computacional y sus dimensiones.

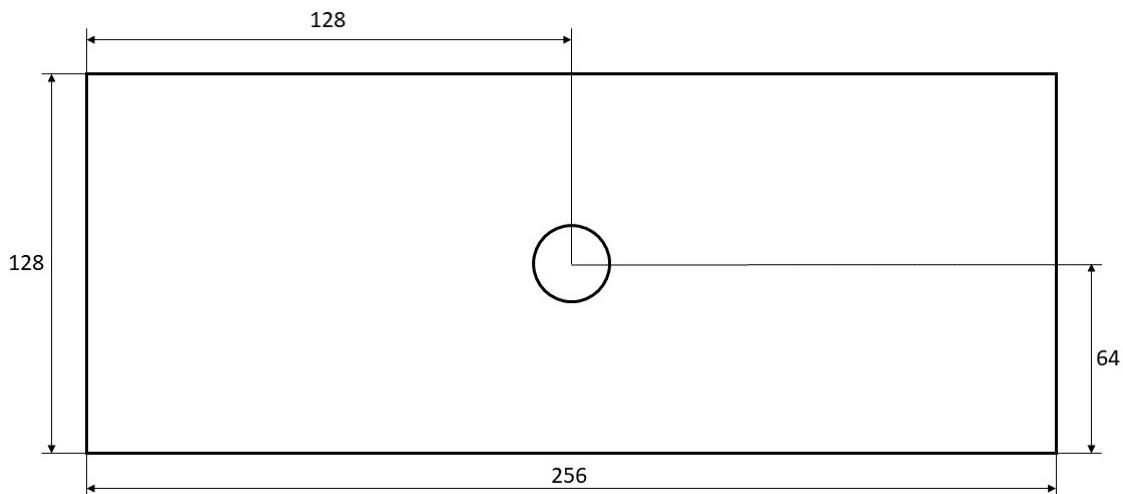


Figura 4.7: Dominio computacional de la simulaciones con geometrías variables (sin escalar).

Se ha construido un mallado compuesto de celdas poliédricas dentro del dominio descrito. La mayoría de celdas se ubican alrededor de la geometría y en las paredes del dominio. Asimismo, se ha sido diseñado un control volumétrico para refinar la malla alrededor del obstáculo para poder mantener el valor y^+ por debajo de 1. La figura 4.8 muestra el mallado descrito.

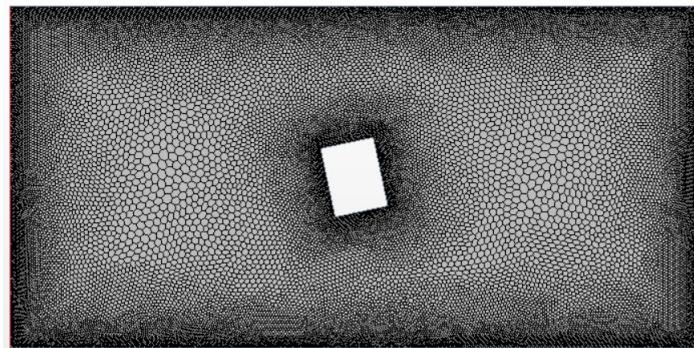


Figura 4.8: Ejemplo de la distribución del mallado alrededor de una geometría.

El fluido considerado es un flujo de aire incompresible turbulento inestable con una densidad (ρ) de $1,18415 \text{ kg/m}^3$ y una viscosidad dinámica (μ) igual a $1,85508 \cdot 10^{-5} \text{ Pa}\cdot\text{s}$. Ambas magnitudes se asumen como constantes. La velocidad de entrada es de 5 m/s. Para que el tamaño de los datos de CFD encajen con el tamaño de las entradas de la CNN, se interpolan en una red de 79×172 .

4.2.2 Red neuronal convolucional

La arquitectura U-Net empleada en 4.1.2 se toma como referencia para los entrenamientos de esta sección. Aquí se analizan dos variantes de esta arquitectura. En primer lugar, se reutiliza el enfoque temporal para predecir los estados futuros de los campos de velocidades y presión. Posteriormente, se focaliza en la predicción de un instante determinado de la terna de campos a partir de las entradas de la red, que representan las características geométricas del obstáculo y el dominio computacional.

La primera variante corresponde con la representada en la figura 4.3. Se presentan como entradas las matrices SDF y FRC y la muestra $t-1$ del campo analizado. Un único decodificador mapea la información en la LGR, que se decodifica para obtener el campo estudiado en el instante t .

La segunda variante se diferencia de la primera en que, a través de tres decodificadores, se obtienen los tres campos analizados. Asimismo, las entradas de la red son únicamente las matrices SDF y FRC. La figura 4.9 representa gráficamente el *autoencoder* con tres decodificadores. Los entrenamientos con esta segunda variante se realizan para una estructura *autoencoder* de 4 bloques codificadores/decodificadores y para otra de 5. La figura 4.10 representa de forma detallada la estructura U-Net con una configuración de 4 bloques codificadores/decodificadores y 3 decodificadores.

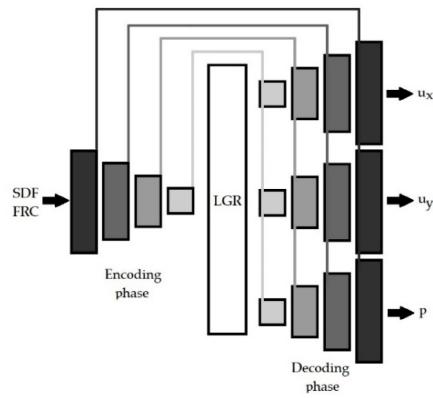


Figura 4.9: Arquitectura U-Net con 3 decodificadores.

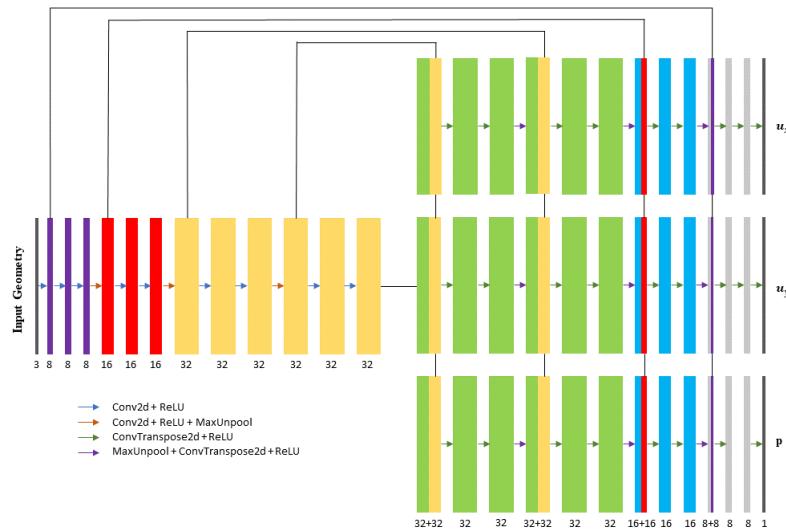


Figura 4.10: Arquitectura detallada de la CNN con 3 decodificadores.

4.2.3 Parámetros del entrenamiento

El optimizador seleccionado es el AdamW, dado por el algoritmo [1]. Ambos factores de olvido se inicializan en 0,5. Para buscar la mejor combinación de los hiperparámetros para la red que predice las muestras futuras y la que predice la muestra inicial, se entrena las redes con todas las combinaciones posibles de los valores dados por las tablas [4.2] y [4.3] respectivamente.

Parámetro	Valores		
Arquitectura	[8, 16, 32, 32]		
Tamaño de las ventanas	3	5	7
Función de coste	L1-norm		
Tasa de aprendizaje (lr)	0,001	0,0001	
Decaimiento de pesos	0,005		
<i>Batch size</i>	32	64	128
Ratio entrenamiento-test	0,7-0,3		
Nº de épocas	2000		

Tabla 4.2: Conjunto de valores seleccionados para la búsqueda de los hiper-párametros adecuados para la red que predice los instantes futuros de los campos con geometrías variables.

Parámetro	Valores			
Arquitectura	[8, 16, 32, 32]		[8, 16, 16, 32, 32]	
Tamaño de las ventanas	3	5	7	9
Función de coste	L1-norm			
Tasa de aprendizaje (lr)	0,001	0,0001		
Decaimiento de pesos	0,005			
<i>Batch size</i>	32	64		128
Ratio entrenamiento-test	0,7-0,3			
Nº de épocas	1000	2000		

Tabla 4.3: Conjunto de valores seleccionados para la búsqueda de los hiper-párametros adecuados para la red que predice la muestra inicial.

4.2.4 *Data augmentation*

El entrenamiento de la red para predecir la primera muestra de los campos de velocidades y presión necesita de una gran cantidad de datos. No obstante, tomando un instante determinado, únicamente se tienen seis muestras, una por cada geometría. Por ello, se opta por aplicar una técnica de *data augmentation* que permita incrementar dicha cantidad de datos para el entrenamiento.

Las técnicas de *data augmentation* se emplean asiduamente en aplicaciones de DL. Consisten en la generación de datos sintéticos realistas con el objetivo de aumentar la cantidad de datos para el proceso de aprendizaje de una red neuronal. La técnica más utilizada consiste en añadir transformaciones geométricas y perturbaciones a los datos reales. Sin embargo, este sencillo procedimiento no puede ser aplicado en este estudio. Como solución a este inconveniente, se aplica la técnica de *data augmentation* expuesta en el estudio de Abucide et al. [1].

Esta técnica de *data augmentation* se aplica teniendo en cuenta la teoría de la semejanza de la dinámica de fluidos. Para cualquier simulación, si el número de Reynolds se mantiene constante en cada caso, la nueva velocidad de entrada al dominio y las nuevas velocidades y presiones de los campos pueden ser calculadas. El número de Reynolds se obtiene a partir de la [4.4](#).

Asumiendo que el número de Reynolds es constante en cada uno de los casos, la nueva velocidad de entrada para cada caso se puede calcular mediante la ecuación [4.7](#). Esto es debido a que el fluido y las condiciones de contorno se mantienen constantes y a que la densidad y la viscosidad dinámica no ejercen influencia ninguna sobre la velocidad. Como consecuencia, aplicando ligeras modificaciones al tamaño de las geometrías, la cantidad de datos de entrada para la CNN se incrementa considerablemente.

$$u_{\infty i}^* = \frac{D_1}{D_i} u_{\infty 1} \quad (4.7)$$

Las expresiones de los nuevos campos de velocidades y presión vienen dadas por las ecuaciones [4.8](#), [4.9](#) y [4.10](#).

$$\hat{u}_{xi}(\hat{x}, \hat{y}) = \frac{u_x i}{u_{\infty i}^*} \left(\frac{x_i}{D_i}, \frac{y_i}{D_i} \right) \quad (4.8)$$

$$\hat{u}_{yi}(\hat{x}, \hat{y}) = \frac{u_y i}{u_{\infty i}^*} \left(\frac{x_i}{D_i}, \frac{y_i}{D_i} \right) \quad (4.9)$$

$$\hat{p}_i(\hat{x}, \hat{y}) = \frac{p_x i}{u_{\infty i}^{*2} \rho} \left(\frac{x_i}{D_i}, \frac{y_i}{D_i} \right) \quad (4.10)$$

donde \hat{u}_{xi} , \hat{u}_{yi} y \hat{p}_i representan los nuevos campos de velocidades y presión y $\frac{x_i}{D_i}$ y $\frac{y_i}{D_i}$, las nuevas coordenadas dentro del dominio. Como establece la teoría de la semejanza, el tamaño del dominio cambia proporcionalmente a la modificación del tamaño de la geometría. Esto se muestra en la ecuación [4.11](#), donde se muestra la equivalencia para dos puntos concretos del mallado.

$$\hat{x} = \frac{x_1}{D_1} = \frac{x_2}{D_2} \quad (4.11)$$

Con esta información y mediante un código que interpola los campos originales, la cantidad de datos se incrementa fácilmente. La ventaja de este método de *data augmentation* reside en que evita la generación de nuevas simulaciones CFD de larga duración y elevados recursos computacionales. Se han tomado las primera

muestra de los campos de velocidades paralela al flujo y vertical y de presión simulados mediante CFD para cada una de las seis geometrías. Después, se han multiplicado las dimensiones de las geometrías por cincuenta valores diferentes de una variable llamada factor de tamaño, que varían entre 0,93 y 1,07. A continuación, se ha aplicado el procedimiento explicado en este apartado y se han obtenido 300 muestras de CFD para el entrenamiento.

Análisis de resultados

5.1 Resultados de la red aplicada a una geometría circular con velocidad de entrada variable

La red neuronal ha sido entrenada para las distintas combinaciones de los valores de los hiper-parámetros dados por la tabla 4.1. En la tabla 5.1 se muestran todos los modelos neuronales entrenados con los valores de los hiper-parámetros y la duración de los entrenamientos para cada variable. Asimismo, en la figuras 5.1 y 5.2 se muestran las gráficas que comparan los errores medios y máximos obtenidos en los testeos de cada modelo. Por cada variable se genera un modelo neuronal individual. Esto permite escoger tres modelos independientes según que combinación de los hiper-parámetros haya proporcionado mejores resultados. Cada modelo neuronal ha sido entrenado con los mismos datos para el entrenamiento y para el test. En este caso, los modelos seleccionados son los de ID igual a 1, 9 y 8 para la velocidad paralela al flujo, la velocidad vertical y la presión, respectivamente. La siguiente configuración de los filtros ha sido empleada para todos los entrenamientos: [8, 16, 32, 32].

Los mejores modelos para la predicción de las muestras futuras de la velocidad paralela al flujo son el 1 y el 6. Pese a que el modelo 6 presenta menor error medio, el error máximo es elevado. Por tanto, el primer modelo es el seleccionado. En el caso de la velocidad vertical, los mejores modelos son el 9 y el 10. Se escoge el modelo 9 por proporcionar un error máximo menor que las predicciones del décimo modelo. En el caso de la presión, los modelos 7, 8 y 9 son los que proporcionan los mejores resultados. Se escoge el modelo con ID igual a 8, debido a presentar menor error medio que el modelo 7 y menor error máximo que el modelo 9.

ID	Tamaño ventana	Lr	Batch size	N.º épocas	Duración	Duración	Duración
					entrena-miento v_x (h)	entrena-miento v_y (h)	entrena-miento p (h)
1	3	0.001	32	1000	6,73	6,61	6,61
2	5	0.001	32	1000	7,21	7,09	6,95
3	7	0.001	32	1000	9,47	9,30	9,32
4	3	0.001	64	1000	5,56	5,55	5,51
5	5	0.001	64	1000	6,40	6,18	6,29

ID	Tamaño ventana	Lr	Batch size	N.º épocas	Duración	Duración	Duración
					entrena-miento v _x (h)	entrena-miento v _y (h)	entrena-miento p (h)
6	7	0.001	64	1000	8,97	8,69	8,74
7	3	0.0001	32	1000	7,41	6,97	6,98
8	5	0.0001	32	1000	9,59	7,26	7,26
9	7	0.0001	32	1000	7,10	9,41	9,44
10	3	0.0001	64	1000	7,40	6,98	6,93
11	5	0.0001	64	1000	9,60	7,30	7,27
12	7	0.0001	64	1000	7,11	9,43	9,43

Tabla 5.1: Identificador de cada modelo, combinación de los hiper-parámetros utilizados en cada uno de los modelos entrenados y duración de los entrenamientos de cada variable.

La figuras 5.3, 5.4 y 5.5 muestran la comparativa gráfica de los resultados obtenidos mediante las predicciones de la CNN y de las simulaciones CFD. Asimismo, las figuras 5.6, 5.7 y 5.8 muestran los histogramas en los que se comparan la distribución de los datos de las predicciones de la CNN y los valores de CFD. La tabla 5.2 muestra la comparativa de la media aritmética y la desviación estándar para los valores de las simulaciones CFD y la predicciones de la CNN, para las tres variables analizadas. Puesto que las predicciones dependen la predicción de la muestra inicial, en las predicciones más cercanas al instante 50 se incrementan considerablemente los errores absolutos. Específicamente, hasta la muestra 20 los resultados son bastante precisos. Despues, aparece un incremento de los errores absolutos, especialmente en la velocidad paralela al flujo y en los casos con velocidades de entrada al dominio elevadas. Las velocidades de entrada superiores influyen en que los cambios de los valores de los campos entre dos muestras consecutivas sean mayores. Esto dificulta que la CNN prediga los valores de los instantes posteriores, provocando un incremento del error absoluto.

Velocidad entrada (m/s)	Método	CFD			CNN		
		v _x (m/s)	v _y (m/s)	p (Pa)	v _x (m/s)	v _y m/s	p (Pa)
5	μ	4,9951	0,0075	2.1723	5,4124	-0,0285	2,8495
	σ	1,2170	1,1167	4,3208	1,3490	1,1234	5,1211
10	μ	9,9913	-0,0013	5,2600	10,1065	-0,0376	6,5888
	σ	2,3040	1,8506	14,1083	2,2233	1,8575	14,5295
15	μ	14,9948	0,0190	23,3207	14,7106	-0,0316	28,1998
	σ	3,0165	3,6250	33,4826	2,9791	3,5796	38,5308

20	μ	19,9890	0,0665	44,3182	18,8420	-0,1063	49,2844
	σ	4,6631	4,7345	64,6778	4,6450	4,4262	69,8038
25	μ	24,9808	0,0353	71,2020	22,2070	-0,2575	57,7787
	σ	6,3783	5,8095	105,2168	6,2542	5,4596	85,2597

Tabla 5.2: Media aritmética y desviación estándar de las simulaciones CFD y las predicciones de la CNN por cada velocidad de entrada al dominio.

5.2 Resultados de la red que predice los instantes futuros aplicada a geometrías variables

La tabla 4.2 indica los valores de los hiper-parámetros que se han seleccionado para el entrenamiento de los modelos neuronales. Mediante combinaciones de los valores escogidos se han generado una serie de modelos neuronales. Después, se comprueba los valores de error medio y máximo en cada caso. En las tablas 5.3, 5.4 y 5.5 se muestran los mejores modelos ordenados según el mínimo error medio conseguido para cada una de las tres variables. En este caso, se pueden seleccionar diferentes combinaciones de los hiper-parámetros para el entrenamiento de cada variable, puesto que se generan tres modelos independientes. Todos los entrenamientos presentan 2000 épocas con la arquitectura [8, 16, 32, 32]. El criterio seguido para escoger el modelo adecuado se basa en los errores máximos y mínimos proporcionados por los testeos de cada modelo neuronal. Además, las muestras para entrenamiento y test han sido las mismas para cada modelo neuronal.

En los casos de la velocidad vertical y la presión, el error medio y el error máximo mínimos corresponden los modelos neuronales con ID igual a 5 y 12 respectivamente. Sin embargo, para la velocidad paralela al flujo, los dos primeros modelos neuronales con el menor error medio, proporcionan errores máximos demasiados elevados. Los modelos neuronales con ID igual a 12 y 15 son relativamente similares, proporcionando cada uno menor error medio y mayor error máximo y viceversa. El modelo neuronal con ID 15 es el elegido, por ser el de menor error máximo.

ID	Tamaño ventana	Lr	Batch size	Duración entrenamiento (h)	Error medio v_x (m/s)	Error máx. v_x (m/s)
10	3	0,0001	32	1,70	0,1413	33,4453
13	3	0,0001	64	1,35	0,1561	43,7074
12	7	0,0001	32	2,83	0,1715	15,6595
15	7	0,0001	64	2,60	0,2053	14,8938
4	3	0,001	64	1,37	0,2188	2705,84
11	5	0,0001	32	1,67	0,2217	21,416

ID	Tamaño ventana	Lr	Batch size	Duración entrenamiento (h)	Error medio v_x (m/s)	Error máx. v_x (m/s)
3	7	0,001	32	2,74	0,2510	73,7285
14	5	0,0001	64	1,51	0,2847	48,7618
6	7	0,001	64	2,59	0,3729	1848,65
5	5	0,001	64	1,50	0,5728	39398,8

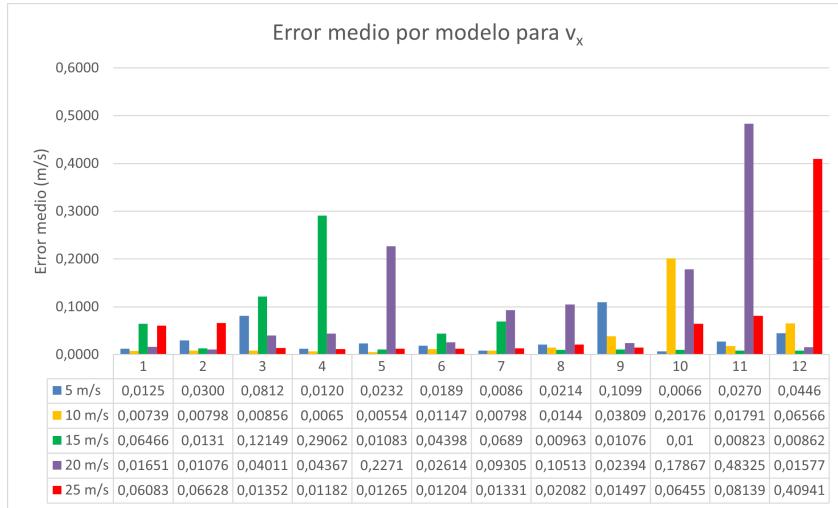
Tabla 5.3: 10 mejores entrenamientos para predicción de los instantes futuros del campo de velocidad paralela al flujo.

ID	Tamaño ventana	Lr	Batch size	Duración entrenamiento (h)	Error medio v_y (m/s)	Error máx. v_y (m/s)
5	5	0,001	64	1,45	0,0585	9,7446
15	7	0,0001	64	2,55	0,0651	10,5586
12	7	0,0001	32	2,72	0,0651	32,4277
2	5	0,001	32	1,56	0,0674	21,7514
14	5	0,0001	64	1,46	0,0976	18,6669
7	3	0,001	128	1,40	0,0992	35,5557
9	7	0,001	128	2,69	0,0999	15,4869
1	3	0,001	32	1,59	0,1016	802,375
8	5	0,001	128	1,49	0,1094	90,4429
16	3	0,0001	128	1,41	0,1358	34,6491

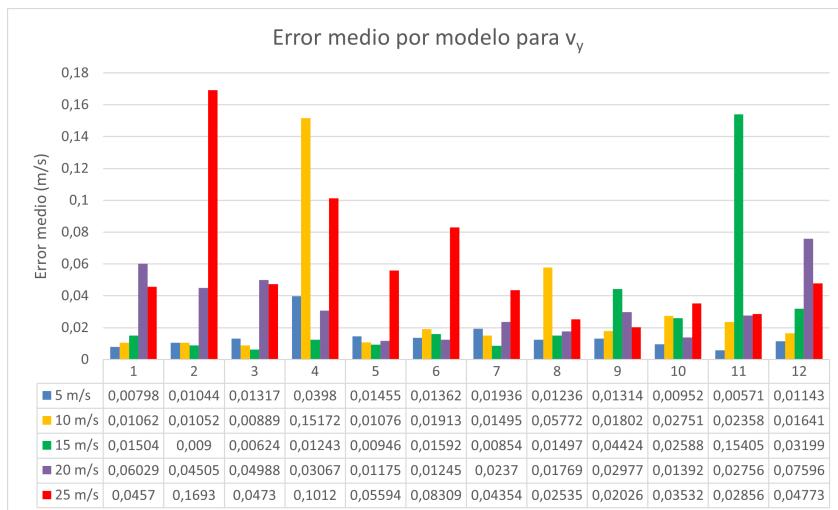
Tabla 5.4: 10 mejores entrenamientos para la predicción de los instantes futuros del campo de velocidad vertical.

ID	Tamaño ventana	Lr	Batch size	Duración entrenamiento (h)	Error medio p (Pa)	Error máx. p (Pa)
12	7	0,0001	32	2,72	0,7139	112,09
4	3	0,001	64	1,24	0,8828	132,73
11	5	0,0001	32	1,57	0,9542	119,92
1	3	0,001	32	1,58	0,9748	466,44
14	5	0,0001	64	1,46	0,9933	117,70
5	5	0,001	64	1,44	1,0869	356,00
15	7	0,0001	64	2,54	1,0900	212,32
7	3	0,001	128	1,40	1,1218	549,21
18	7	0,0001	128	2,71	1,1887	108,50
2	5	0,001	32	1,55	1,3195	196,38

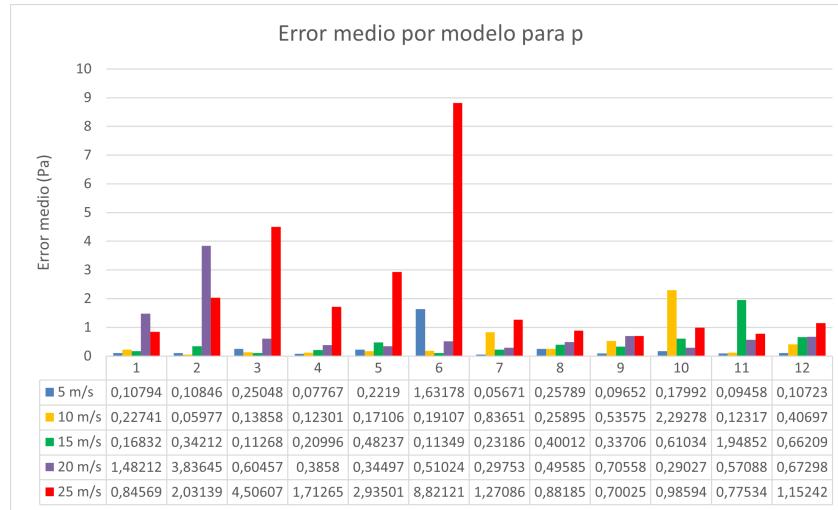
Tabla 5.5: 10 mejores entrenamientos para predicción de los instantes futuros del campo de presión.



(a)

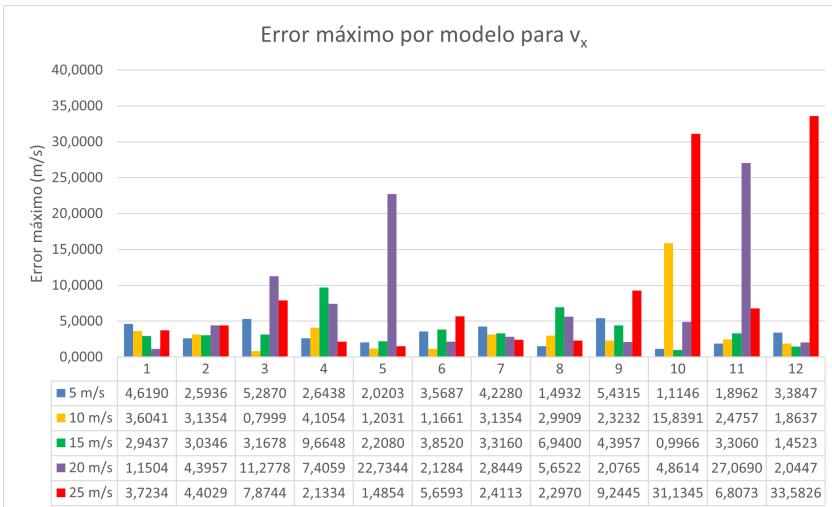


(b)

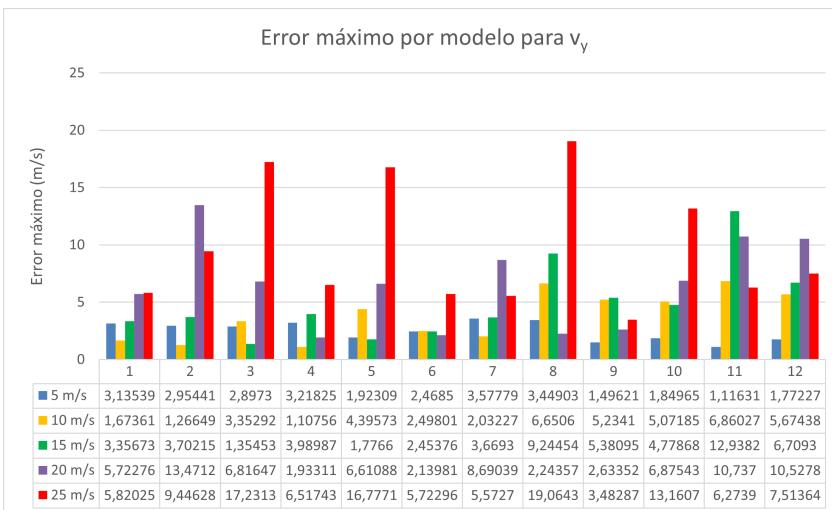


(c)

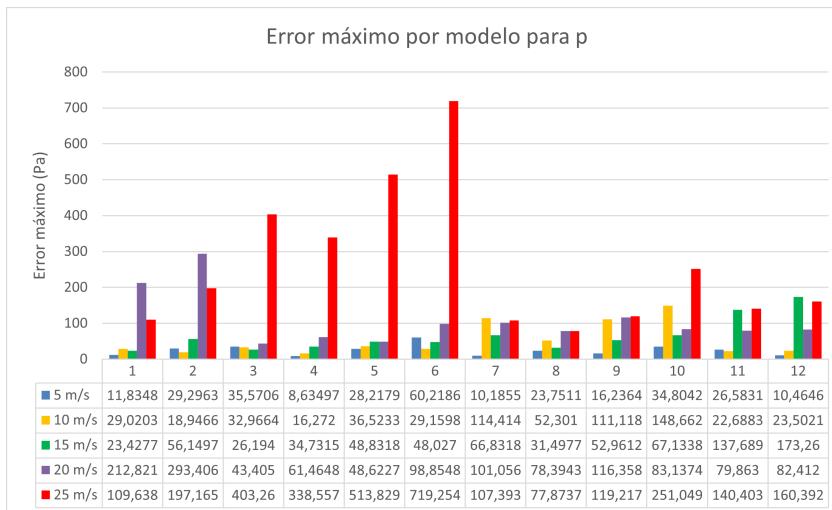
Figura 5.1: Errores medios de cada modelo neuronal entrenado para cada una de las velocidades de entrada al dominio. a) Velocidad paralela al flujo, b) velocidad vertical y c) presión.



(a)



(b)



(c)

Figura 5.2: Errores máximos de cada modelo neuronal entrenado para cada una de las velocidades de entrada al dominio. a) Velocidad paralela al flujo, b) velocidad vertical y c) presión.

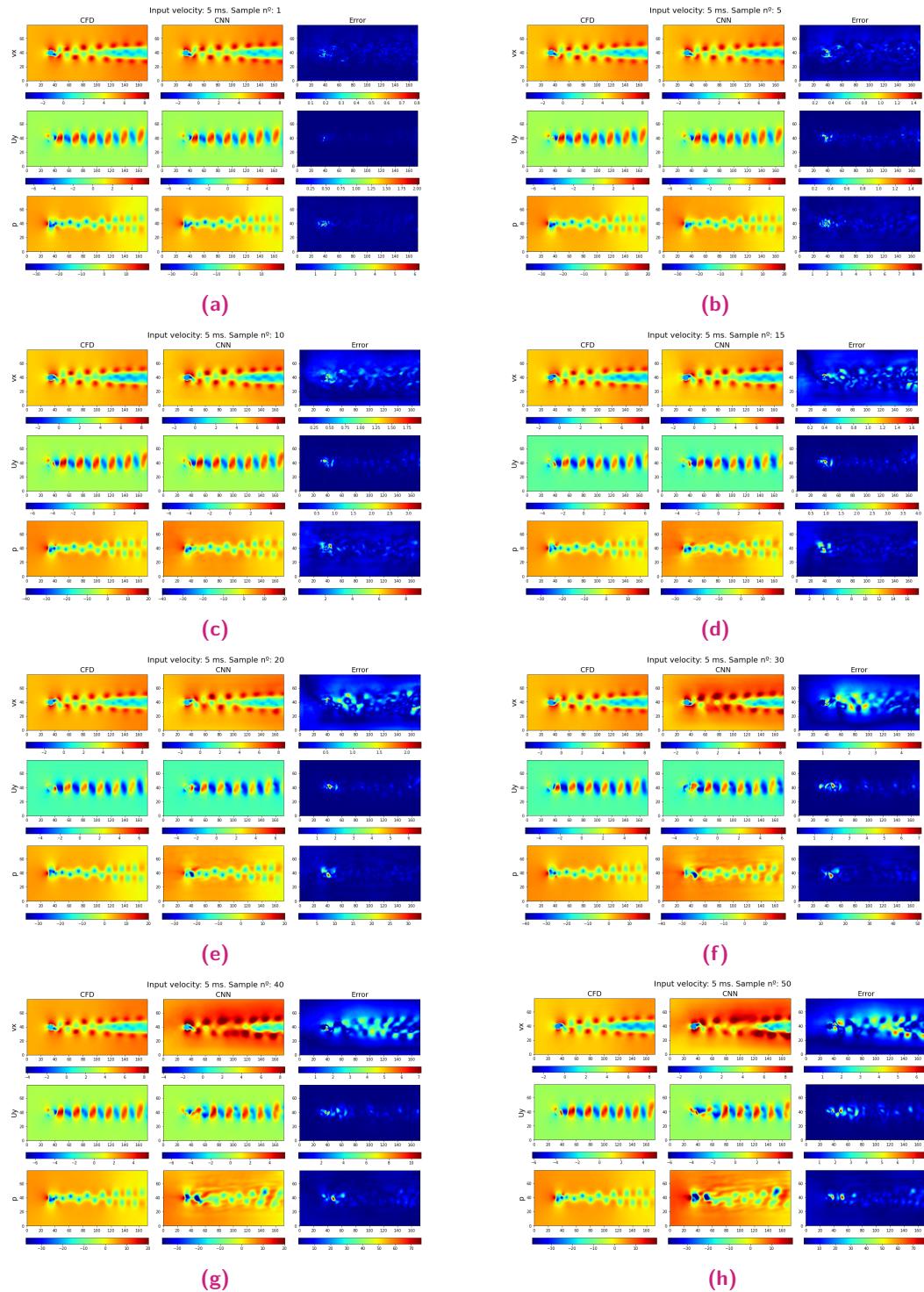


Figura 5.3: Predicciones de la geometría circular con una velocidad de entrada de 5 m/s de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.

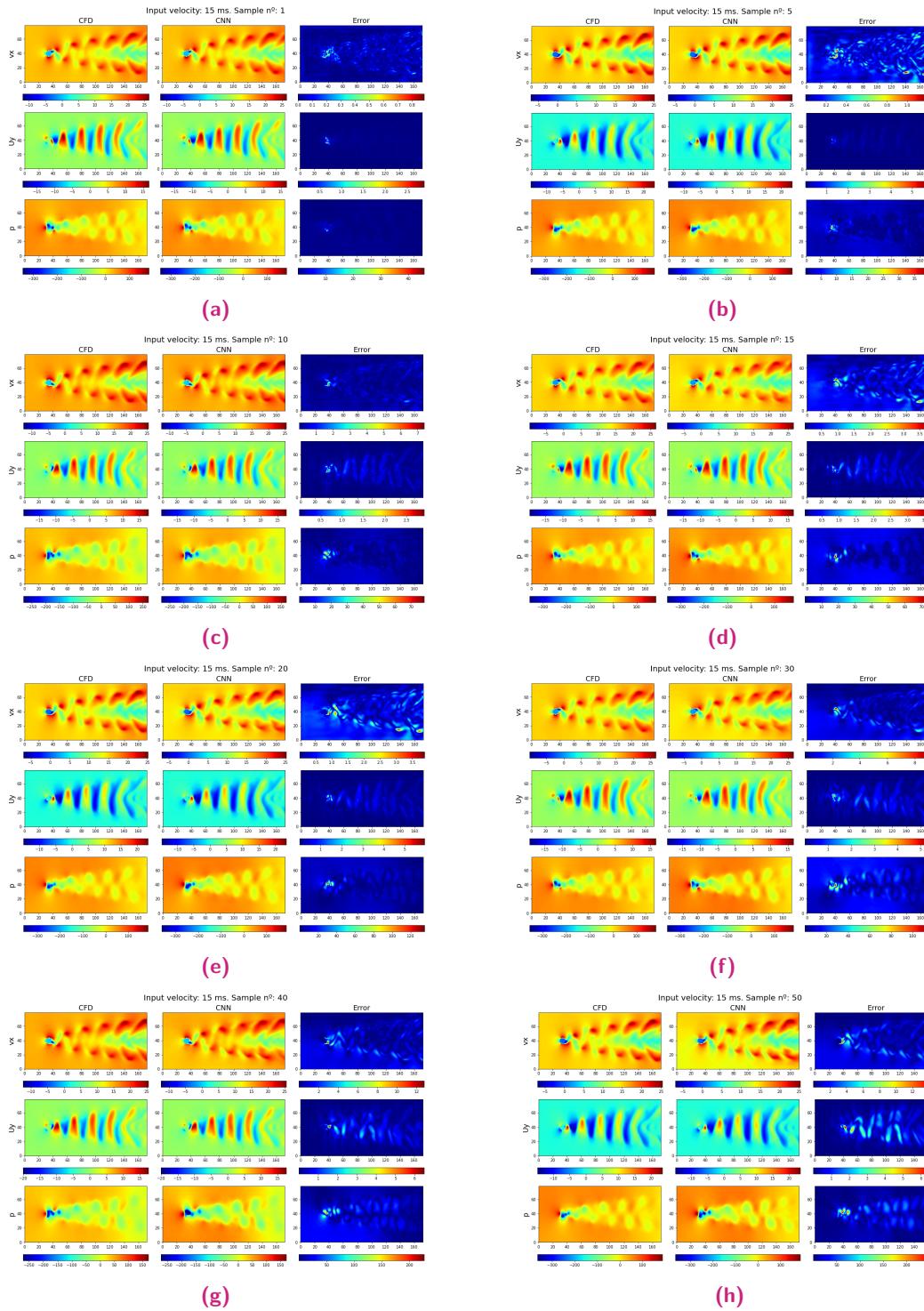


Figura 5.4: Predicciones de la geometría circular con una velocidad de entrada de 15 m/s de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.

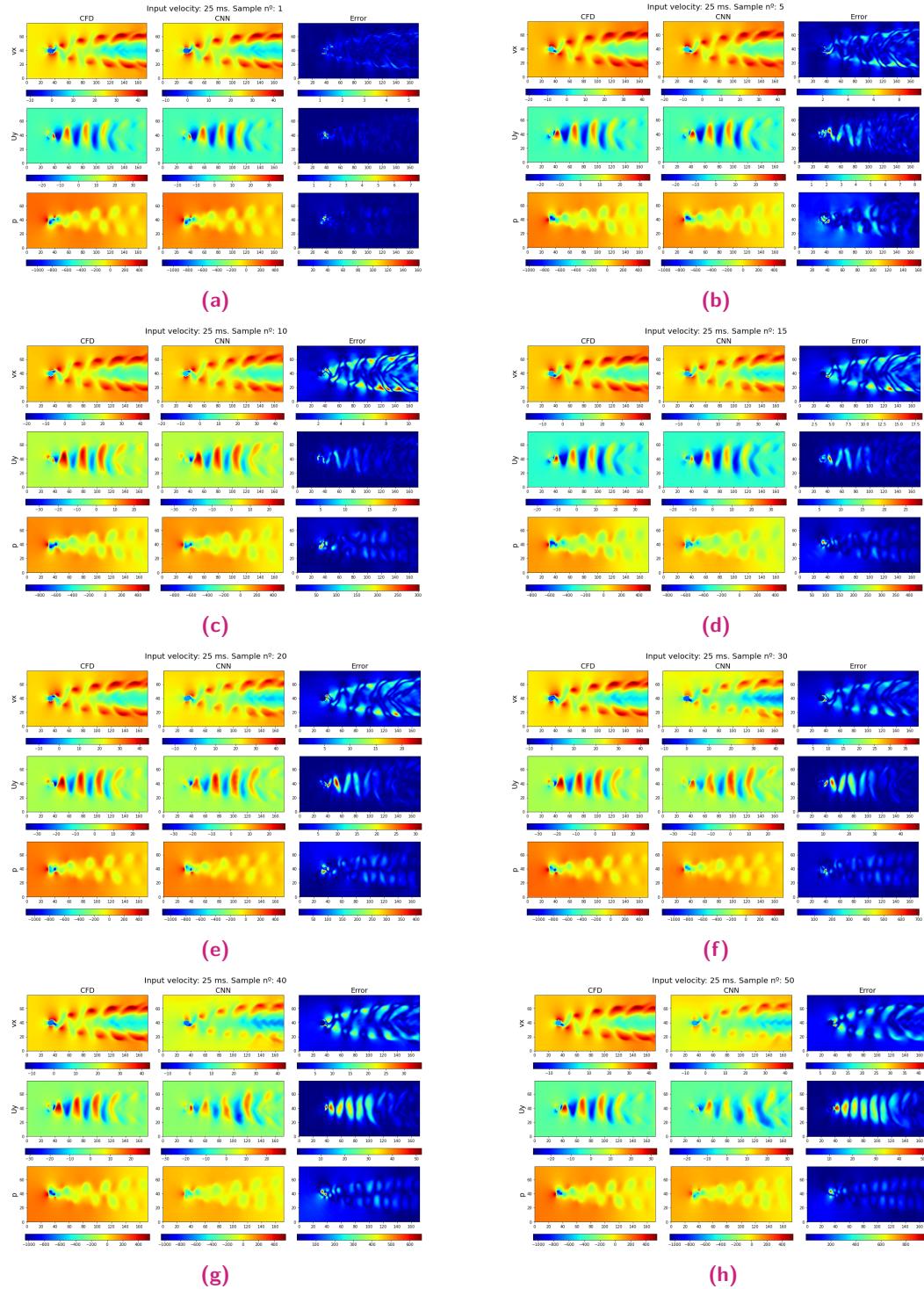


Figura 5.5: Predicciones de la geometría circular con una velocidad de entrada de 25 m/s de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.

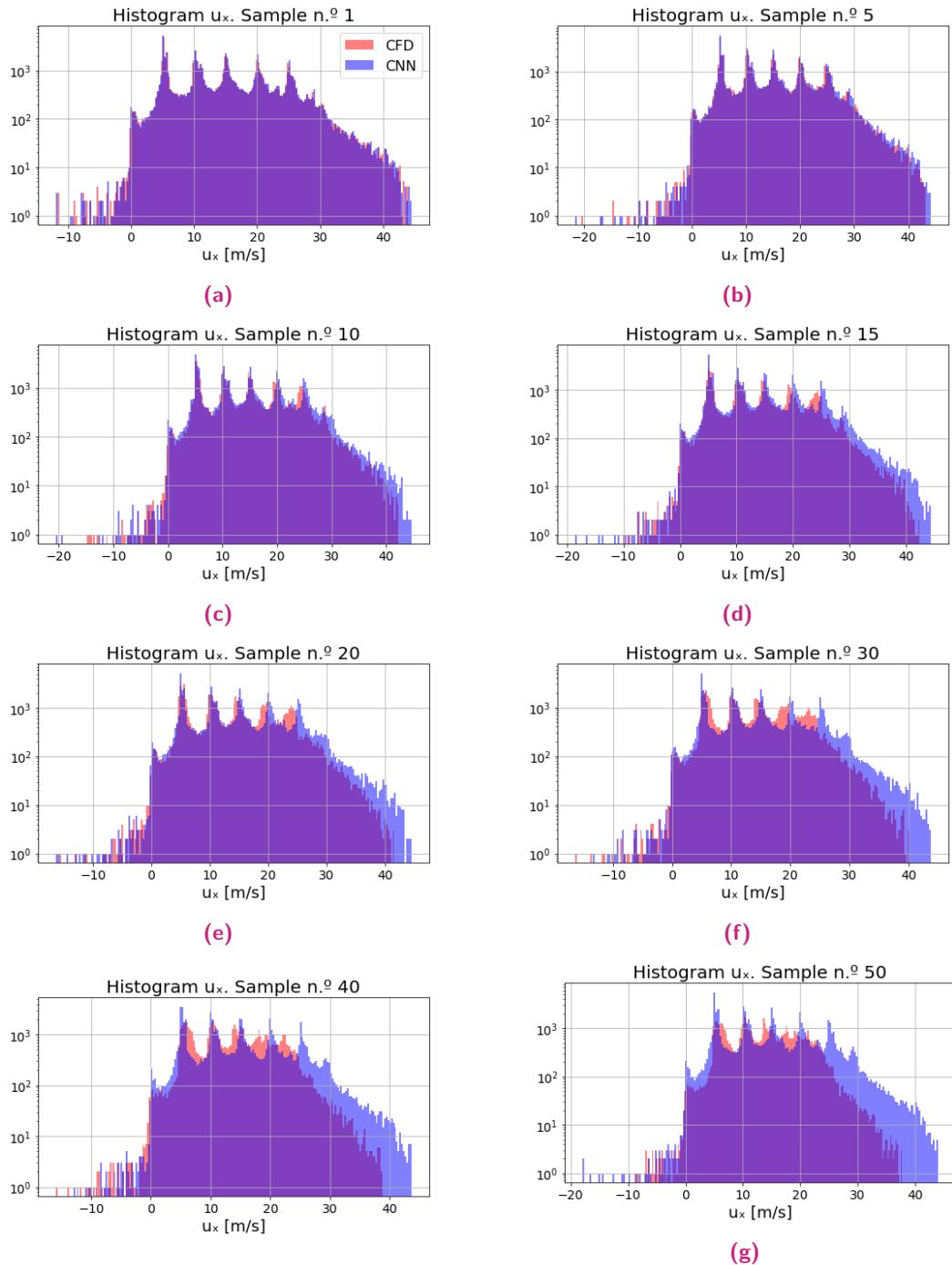


Figura 5.6: Histogramas de la distribución de los datos de CFD y de la CNN del campo de velocidad paralela al flujo alrededor de la geometría circular para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.

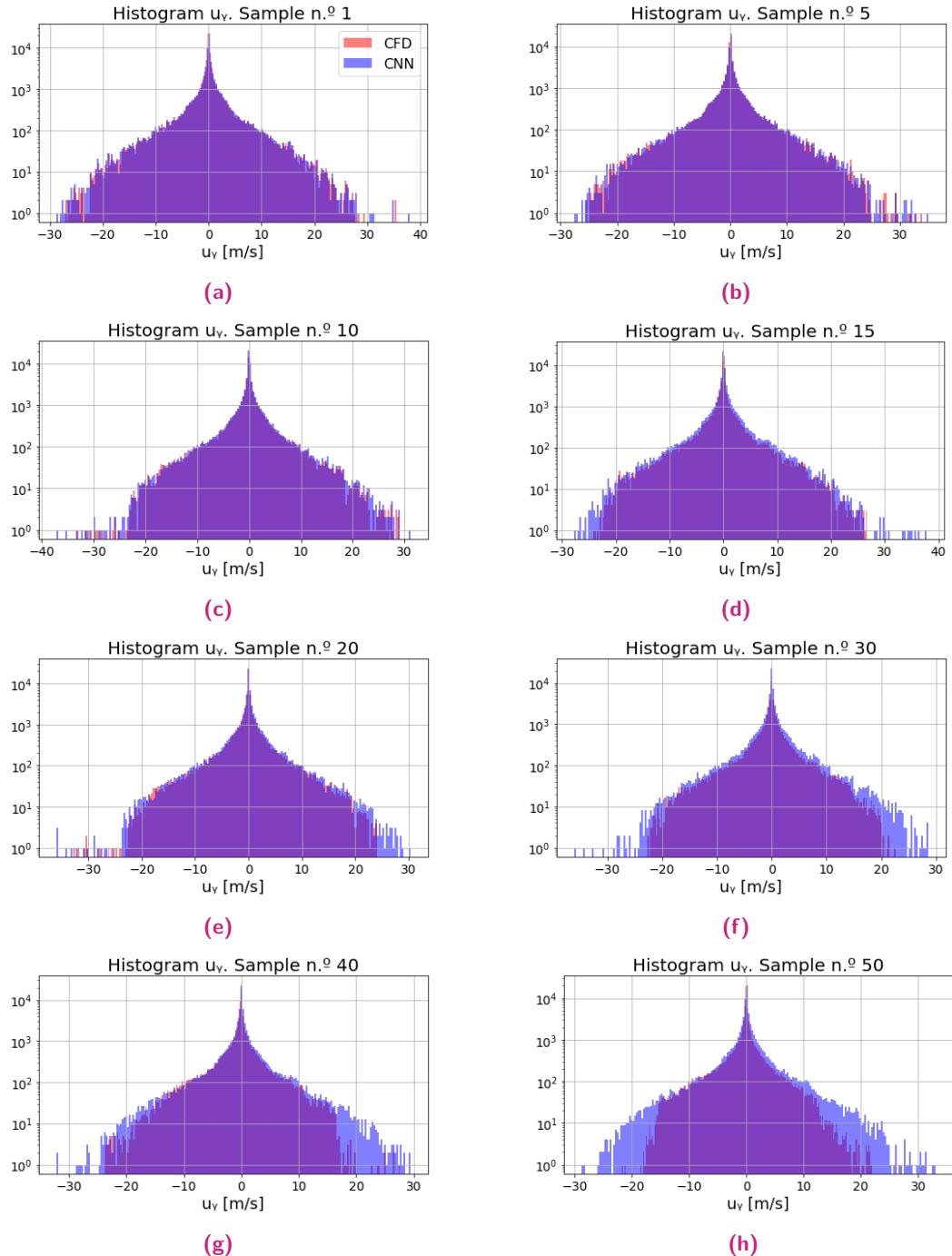


Figura 5.7: Histogramas de la distribución de los datos de CFD y de la CNN del campo de velocidad vertical alrededor de la geometría circular para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.

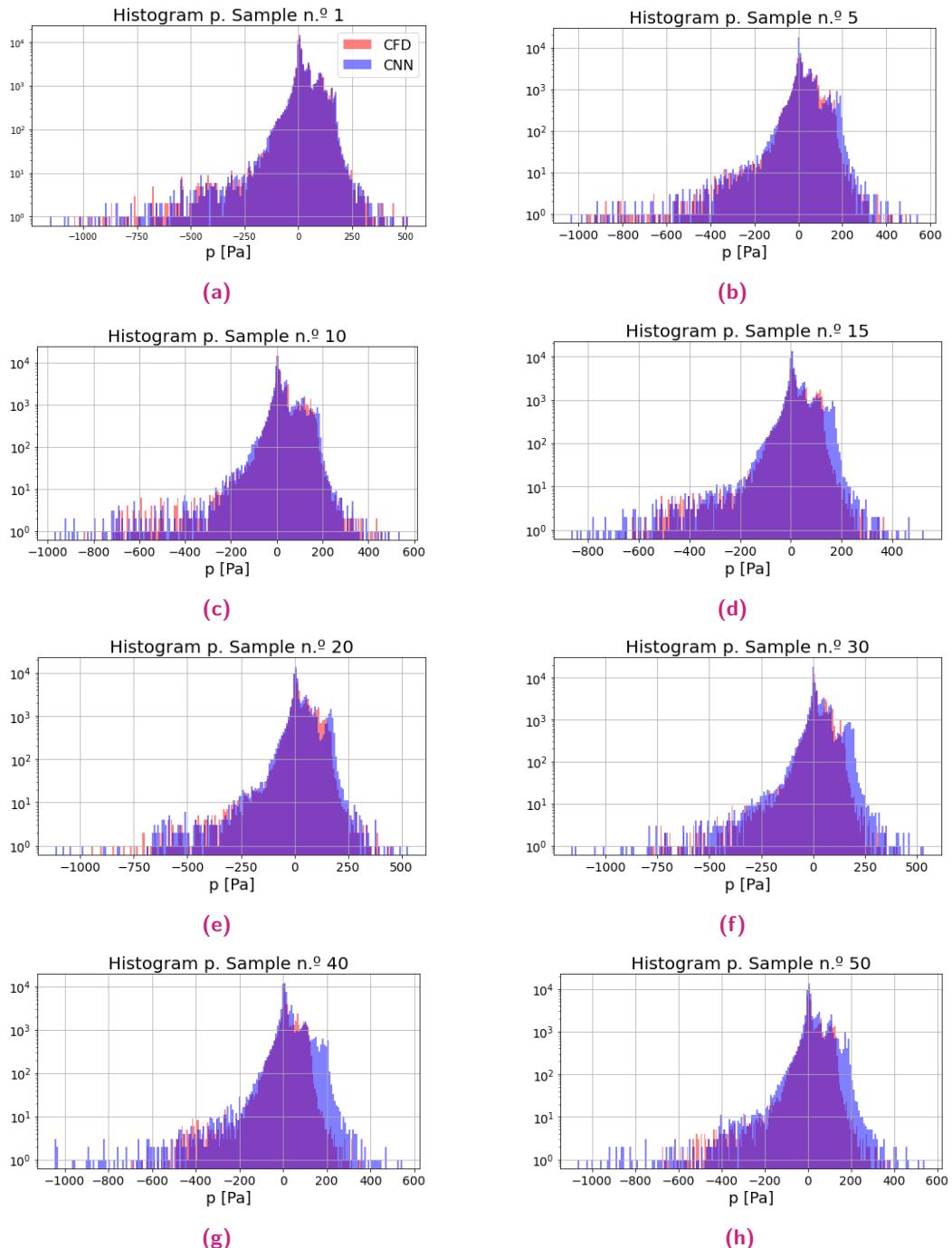


Figura 5.8: Histogramas de la distribución de los datos de CFD y de la CNN del campo de presión alrededor de la geometría circular para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.

En las figuras 5.9, 5.10 y 5.11 se muestra gráficamente la comparación de los resultados obtenidos por la CNN con respecto a las simulaciones proporcionadas por el CFD para el círculo, la elipse y el cuadrado. Asimismo, en las figuras 5.12, 5.13 y 5.14 se muestran los histogramas para las tres variables, que comparan todos los valores dados por la CNN con respecto al CFD. Por otro lado, la tabla 5.6 contiene los valores cuantitativos de la media aritmética y la desviación estándar de las simulaciones CFD y de los tests de la CNN, donde se observa que el modelo neuronal presenta una precisión considerable. Los errores absolutos en las predicciones de los tres campos se incrementan a lo largo de las muestras, puesto que dependen de un estado inicial y la aparición de un error excesivo en un cierto punto, influye considerablemente en el error que producirán las predicciones futuras en dicho punto. Durante las primeras 20 muestras predichas, el error absoluto es relativamente pequeño; sin embargo, en la zona posterior al contorno de las geometrías, surgen errores absolutos elevados, que se acumulan a lo largo de las predicciones. En este caso, los vórtices son captados adecuadamente. La CNN proporciona tasas de error reducidas en esta zona, incluso en los instantes más avanzados.

Método	CFD			CNN		
	v _x (m/s)	v _y (m/s)	p (Pa)	v _x (m/s)	v _y (m/s)	p (Pa)
Media aritmética (μ)	5,0283	-0,0538	8,4882	5,0413	-0,0452	8,5360
Desviación estándar (σ)	1,9109	1,7205	17,0865	1,9940	1,7725	16,9417

Tabla 5.6: Media aritmética y desviación estándar de las 50 muestras predichas por la CNN y simuladas mediante CFD.

5.3 Resultados de la red que predice la muestra inicial para geometrías variables

La red ha sido entrenada para todas las combinaciones de los valores de los hiperparámetros proporcionados por la tabla 4.3. En las tablas 5.7, 5.8 y 5.9 se muestran los resultados de los 10 mejores entrenamientos para las variables v_x, v_y y p respectivamente. A la hora de escoger el modelo entrenado óptimo, se observa que la siguiente combinación de los valores de los hiper-parámetros proporciona los mejores resultados:

- Arquitectura: [8, 16, 16, 32, 32]
- Tamaño de ventana: 3.
- Lr: 0,001.
- *Batch size*: 32

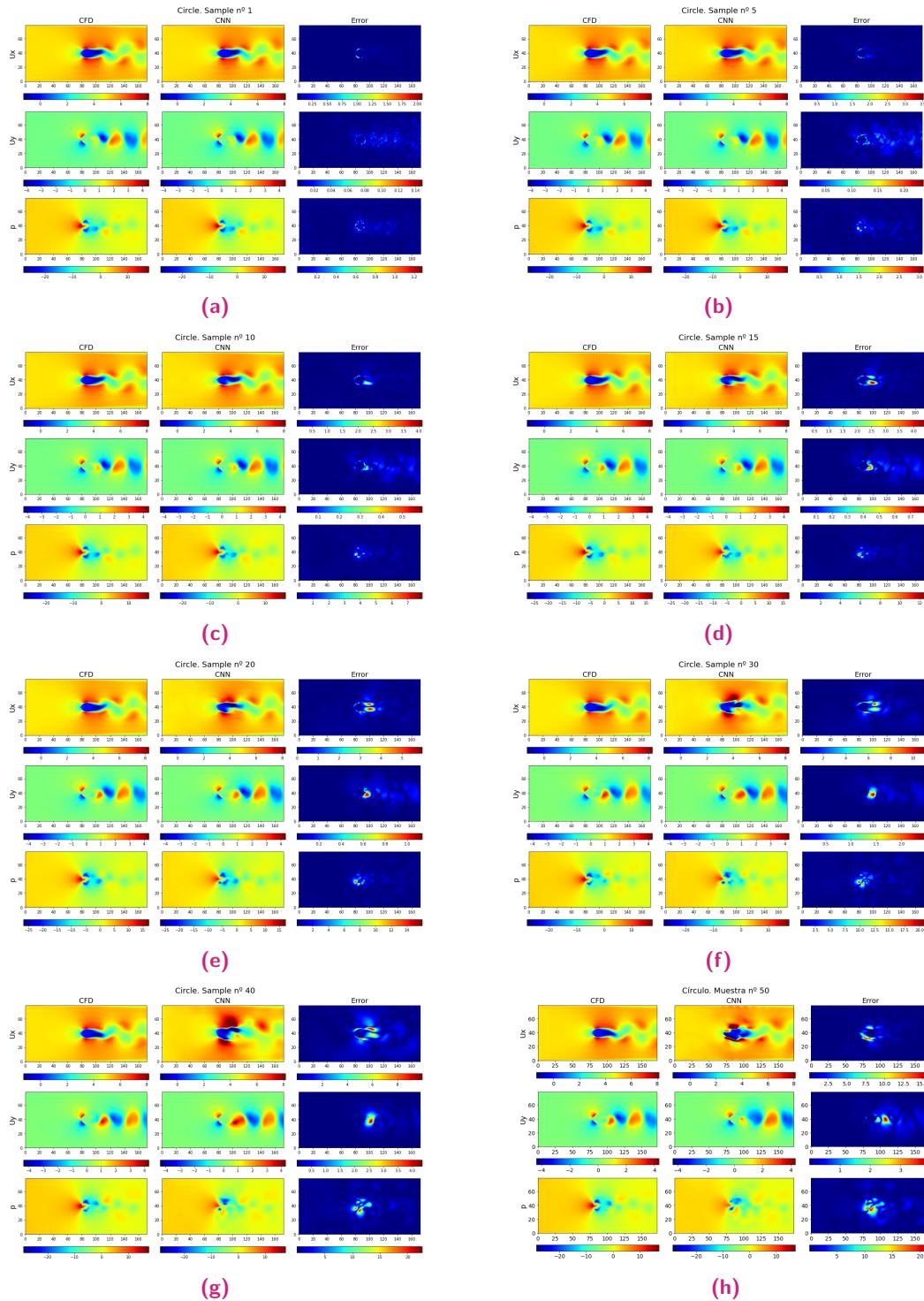


Figura 5.9: Predicciones de la geometría circular de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.

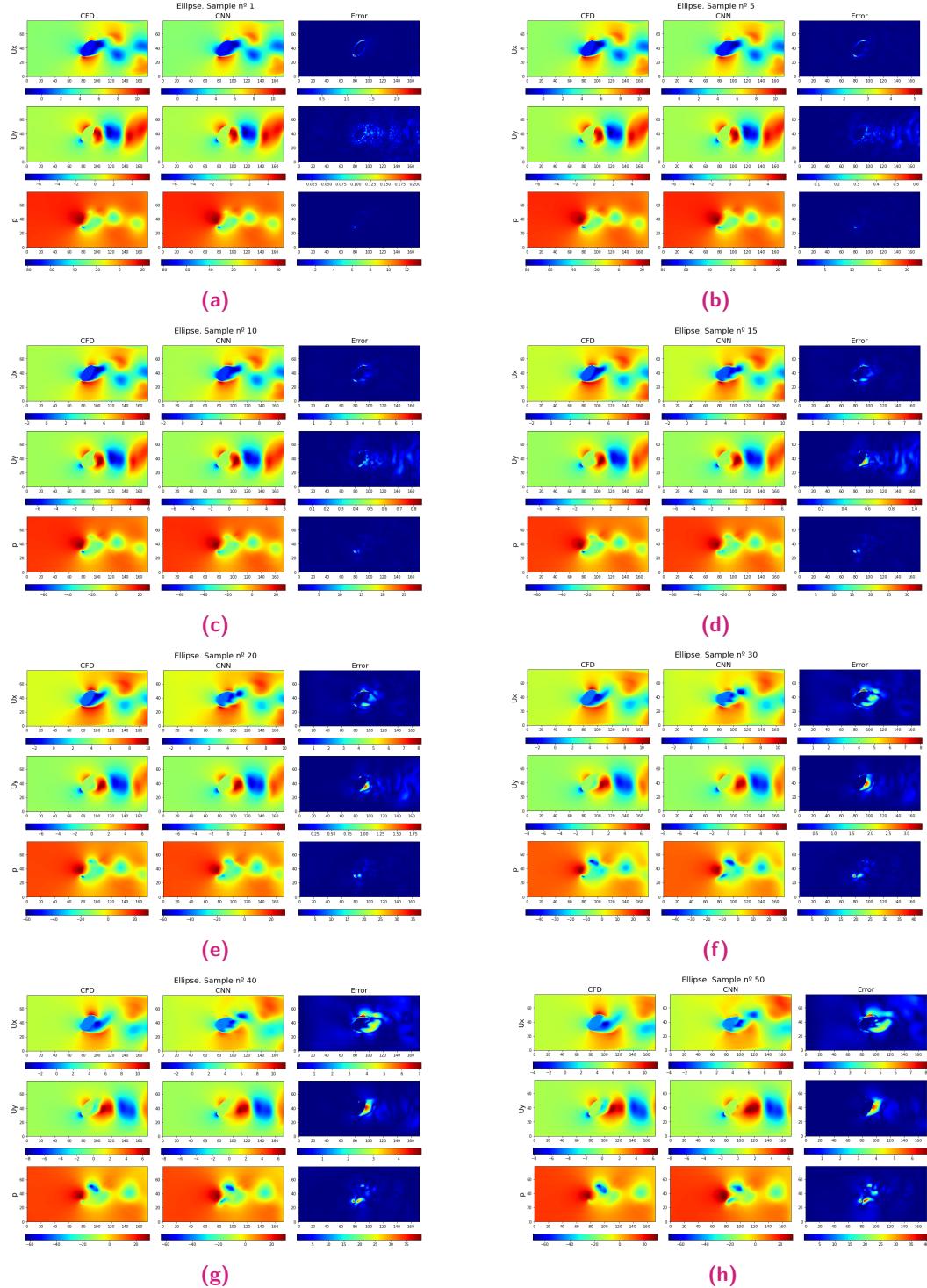


Figura 5.10: Predicciones de la elipse de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.

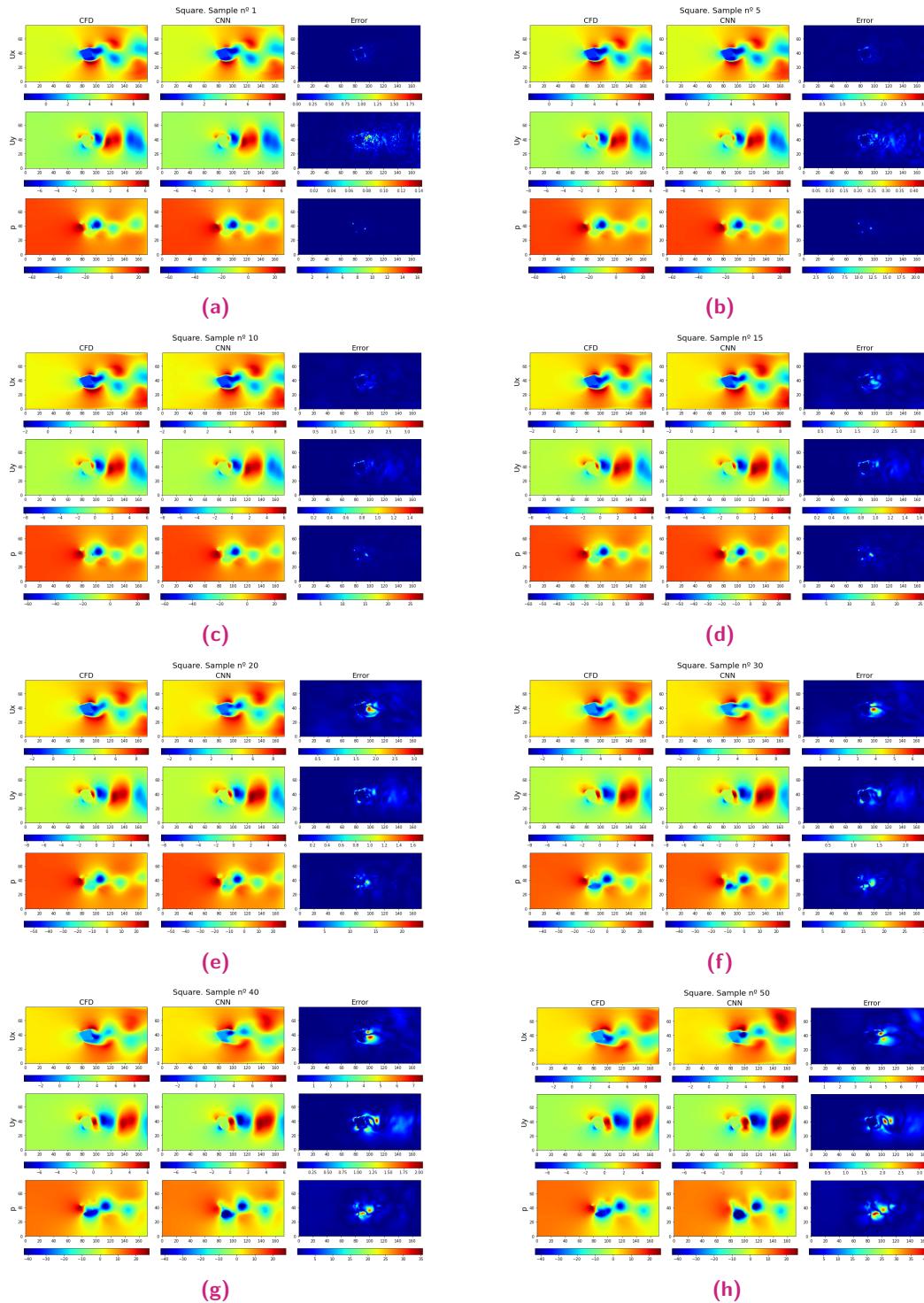


Figura 5.11: Predicciones del cuadrado de las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40, h) 50.

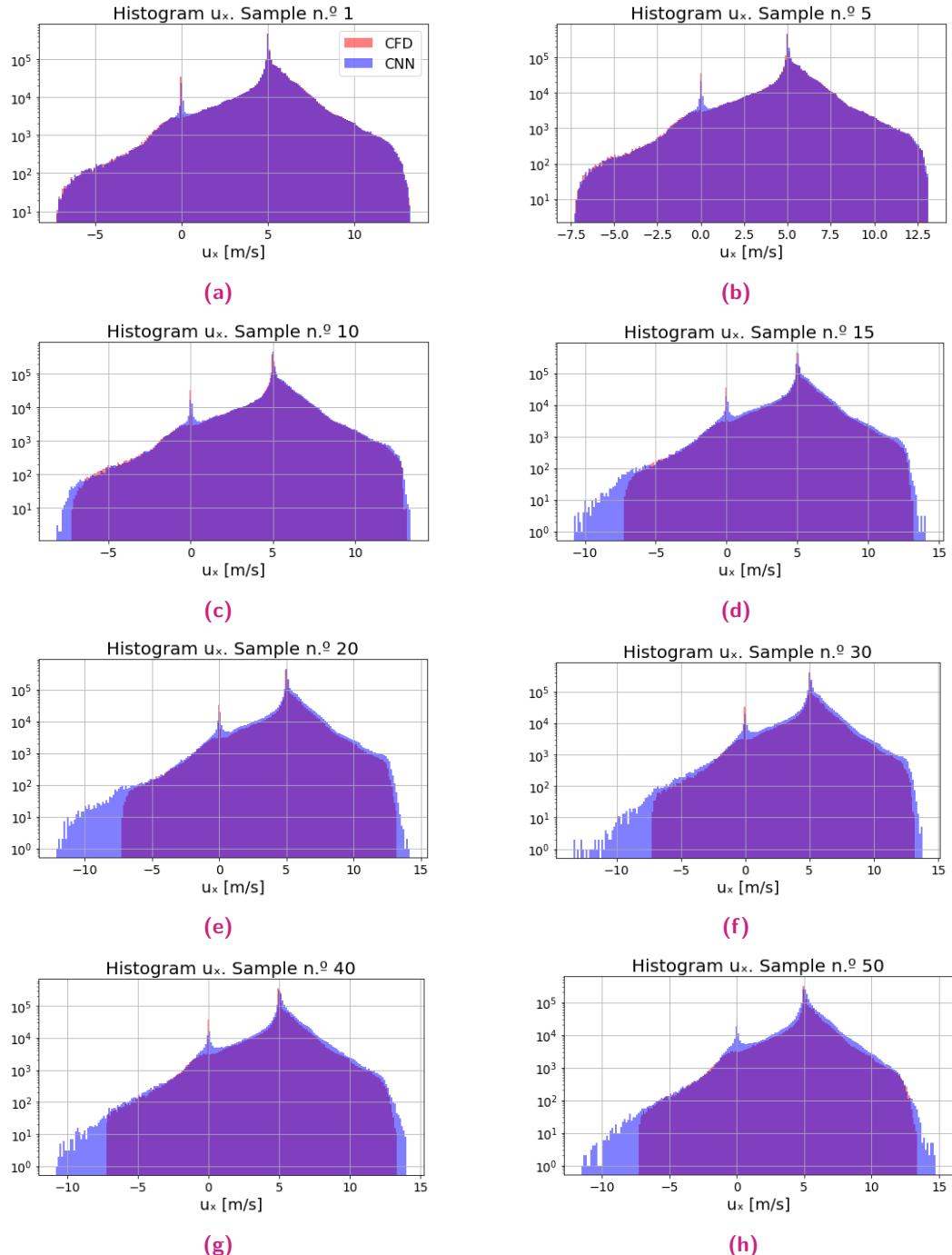


Figura 5.12: Histogramas de la distribución de los datos de CFD y de la CNN del campo de velocidad paralela al flujo para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.

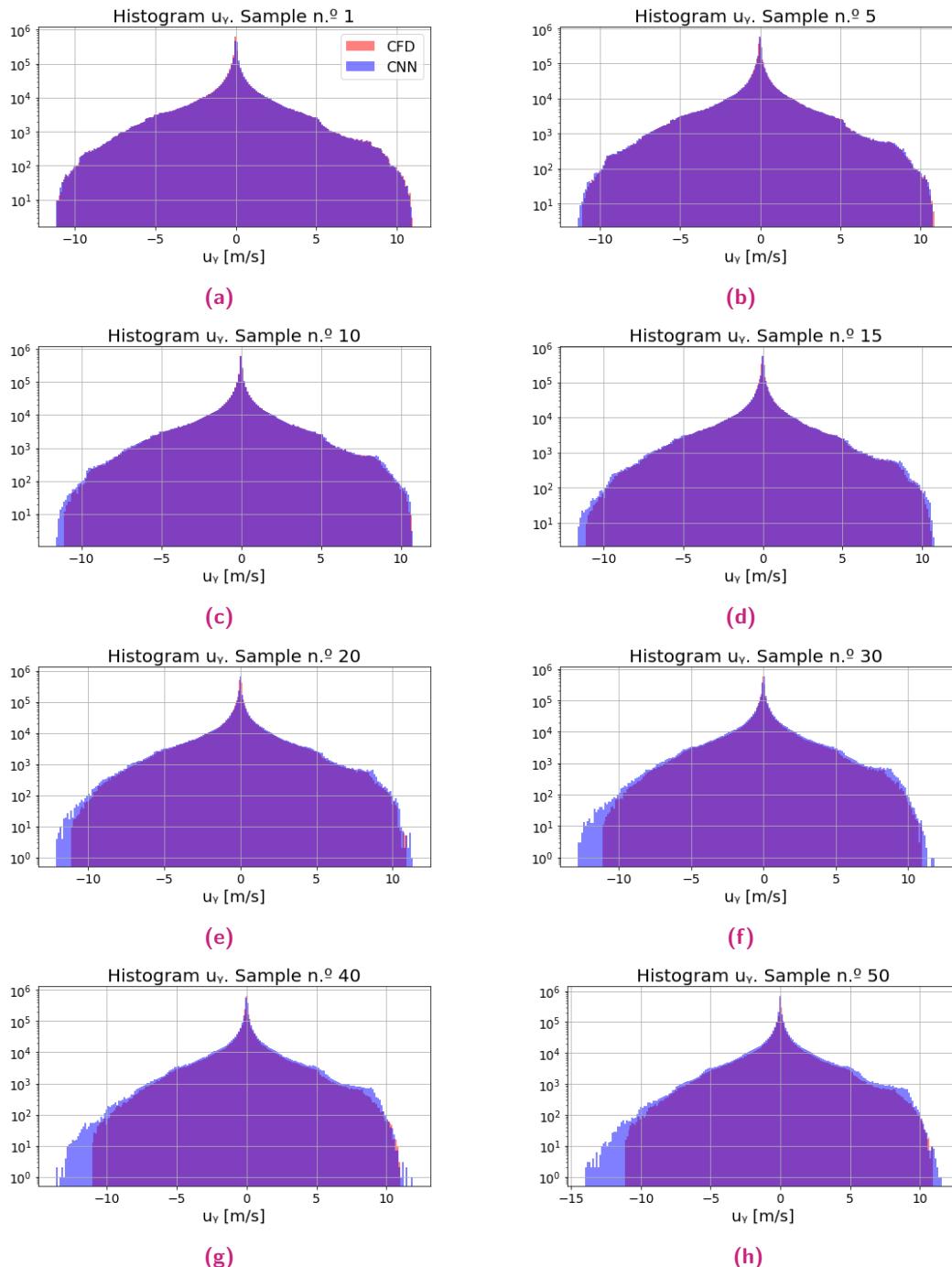


Figura 5.13: Histogramas de la distribución de los datos de CFD y de la CNN del campo de velocidad vertical para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.

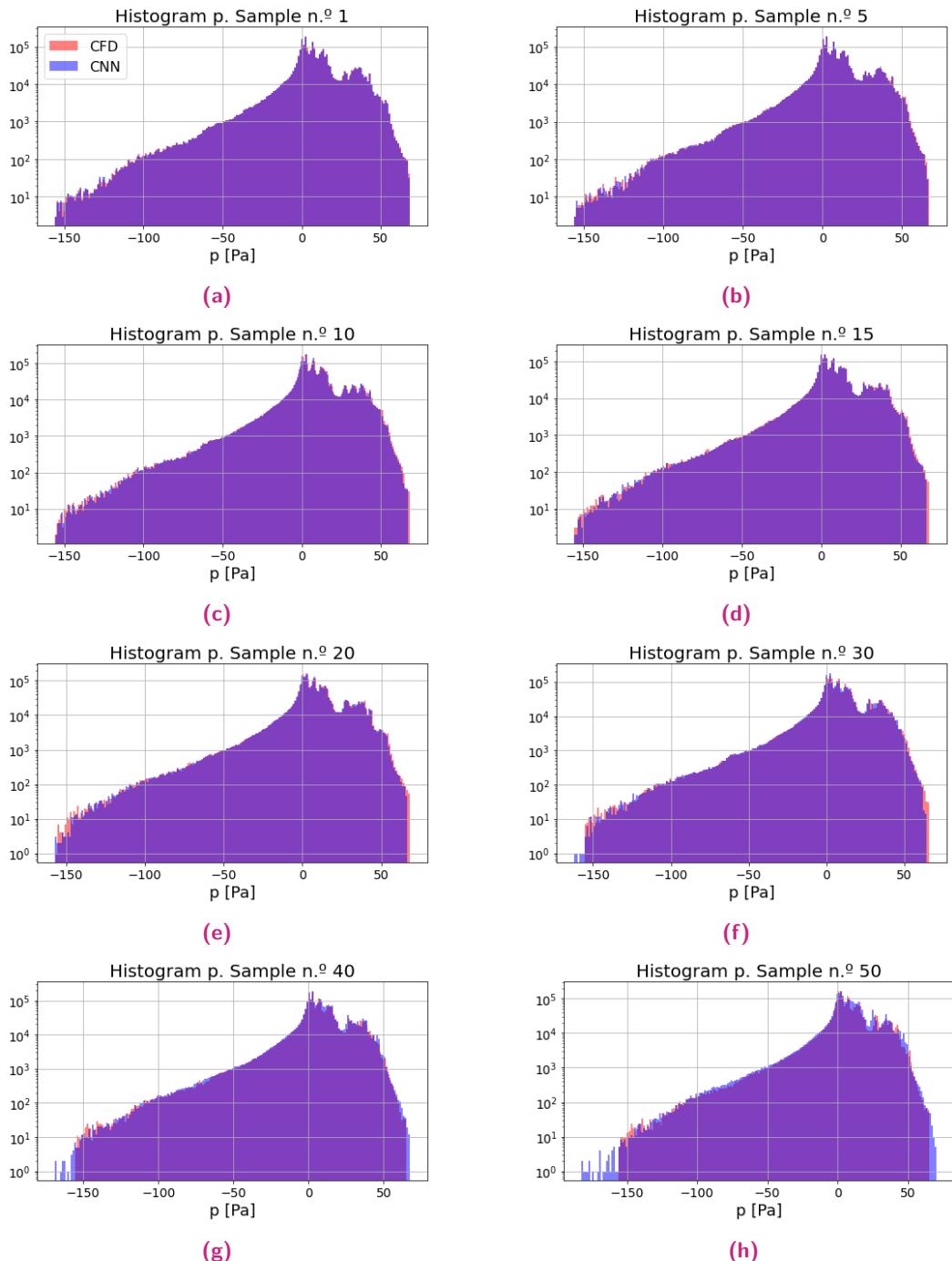


Figura 5.14: Histogramas de la distribución de los datos de CFD y de la CNN del campo de presión para las muestras número: a) 1, b) 5, c) 10, d) 15, e) 20, f) 30, g) 40 y h) 50.

- Número de épocas: 2000

Esta combinación es la adecuada, puesto que se trata del mejor modelo para la velocidad paralela al flujo y la presión y el segundo mejor, para la velocidad vertical. Cada modelo neuronal ha sido ejecutado con las mismas muestras para entrenamiento y test.

Arqui-tectura	Tamaño ventana	Lr	Batch size	N.º épocas	Duración entrenamiento (min)	Error medio v_x (m/s)	Error máx. v_x (m/s)
[8, 16, 16, 32, 32]	3	0,001	32	2000	19,66	0,0528	2,4436
[8, 16, 32, 32]	5	0,001	32	2000	22,33	0,0550	2,5234
[8, 16, 16, 32, 32]	5	0,001	32	2000	21,60	0,0678	3,0944
[8, 16, 32, 32]	3	0,001	32	2000	19,45	0,0698	3,2475
[8, 16, 32, 32]	7	0,001	32	1000	13,84	0,0705	2,5926
[8, 16, 32, 32]	9	0,0001	32	2000	37,20	0,0707	2,9895
[8, 16, 16, 32, 32]	5	0,001	32	1000	10,88	0,0746	2,9101
[8, 16, 32, 32]	7	0,0001	32	2000	27,85	0,0793	4,3594
[8, 16, 32, 32]	3	0,001	128	2000	17,02	0,0811	4,0057
[8, 16, 16, 32, 32]	9	0,0001	32	2000	34,19	0,0831	3,7460

Tabla 5.7: 10 mejores entrenamientos para la velocidad paralela al flujo.

Arqui-tectura	Tamaño ventana	Lr	Batch size	N.º épocas	Duración entrenamiento (min)	Error medio v_y (m/s)	Error máx. v_y (m/s)
[8, 16, 32, 32]	7	0,0001	32	2000	27,85	0,0281	2,2092
[8, 16, 16, 32, 32]	3	0,001	32	2000	19,66	0,0288	2,6517

Arquitectura	Tamaño ventana	Lr	Batch size	N.º épocas	Duración entrenamiento (min)	Error medio v _y (m/s)	Error máx. v _y (m/s)
[8, 16, 32, 32]	5	0,001	32	2000	22,33	0,0292	2,4026
[8, 16, 32, 32]	7	0,001	32	1000	13,84	0,0310	2,4049
[8, 16, 32, 32]	3	0,001	32	2000	19,45	0,0333	3,9388
[8, 16, 32, 32]	5	0,001	64	2000	20,31	0,0343	2,6450
[8, 16, 16, 32, 32]	5	0,001	32	2000	21,60	0,0353	2,0954
[8, 16, 32, 32]	9	0,0001	32	2000	37,20	0,0365	2,2067
[8, 16, 32, 32]	5	0,001	32	1000	11,19	0,0371	1,8295
[8, 16, 16, 32, 32]	7	0,001	32	1000	12,88	0,0397	4,2255

Tabla 5.8: 10 mejores entrenamientos para la velocidad vertical.

Arquitectura	Tamaño ventana	Lr	Batch size	N.º épocas	Duración entrenamiento (min)	Error medio p (Pa)	Error máx. p (Pa)
[8, 16, 16, 32, 32]	3	0,001	32	2000	19,66	0,2474	16,1953
[8, 16, 32, 32]	7	0,001	32	1000	13,84	0,2604	18,4768
[8, 16, 32, 32]	5	0,001	32	2000	22,33	0,2643	16,4145
[8, 16, 16, 32, 32]	9	0,001	64	2000	32,99	0,2736	19,4038
[8, 16, 16, 32, 32]	5	0,001	32	2000	21,60	0,2770	22,1191
[8, 16, 32, 32]	5	0,001	64	2000	20,31	0,2814	17,5238
[8, 16, 32, 32]	3	0,001	32	2000	19,45	0,3048	17,0342

Arqui-tectura	Tamaño ventana	Lr	Batch size	N.º épocas	Duración entrenamiento (min)	Error medio p (Pa)	Error máx. p (Pa)
[8, 16, 32, 32]	7	0,0001	32	2000	27,85	0,3080	23,0767
[8, 16, 16, 32, 32]	9	0,0001	32	2000	34,19	0,3269	27,6613
[8, 16, 32, 32]	5	0,001	32	1000	11,19	0,3314	19,8795

Tabla 5.9: 10 mejores entrenamientos para la presión.

En la figura 5.15 se muestra una predicción del modelo escogido para cada una de las geometrías analizadas. Para evaluar cuantitativamente la precisión del modelo neuronal, mediante la tabla 5.10 se muestran los valores de la media aritmética y la desviación estándar de las simulaciones con CFD y con la CNN. Asimismo, en la figura 5.16 se muestra la distribución de los valores de los campos analizados obtenidos con la CFD y la CNN. Ambas permiten alcanzar la conclusión de que el modelo neuronal es considerablemente preciso con respecto a las simulaciones CFD.

Método	CFD			CNN		
	v _x (m/s)	v _y (m/s)	p (Pa)	v _x (m/s)	v _y (m/s)	p (Pa)
Media aritmética (μ)	5,0674	-0,0644	5,9574	5,0414	-0,0628	6,0475
Desviación estándar (σ)	1,6867	1,5074	15,0737	1,6797	1,5142	15,0853

Tabla 5.10: Media aritmética y desviación estándar de las predicciones de la CNN de la primera muestra y de las simulaciones CFD de la muestra inicial.

5.4 Análisis del coste computacional

En esta sección se lleva a cabo la comparativa de los tiempos computacionales necesarios para obtener los resultados de los campos de velocidades y de presión mediante las simulaciones CFD y las predicciones de los modelos neuronales.

En el caso de los modelos neuronales que predicen los instantes futuros del fluido con velocidad de entrada variable, el tiempo de entrenamiento ha sido de 6,73, 9,41 y 7,26 horas, dando un total de 23,40 horas de entrenamiento. La predicción de

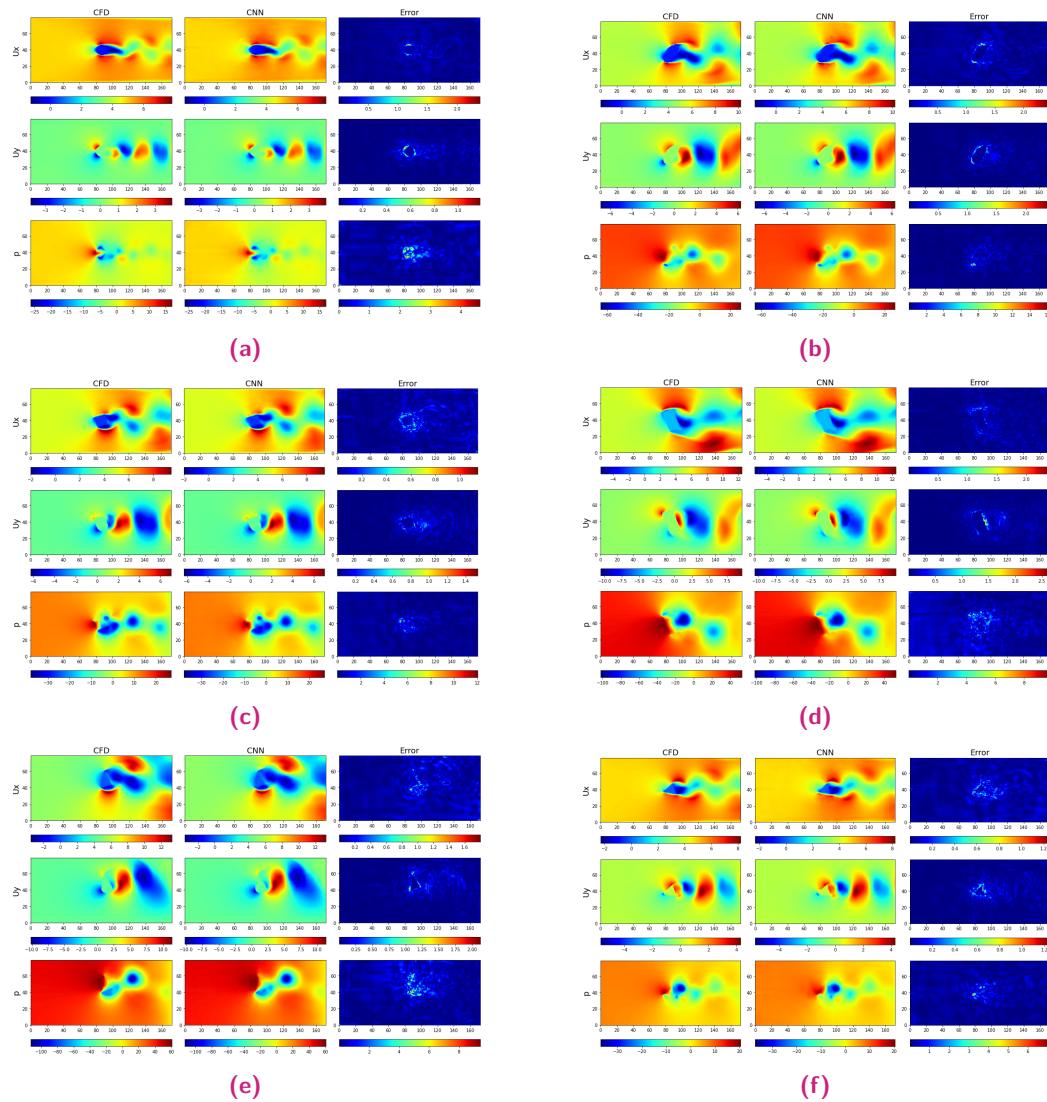


Figura 5.15: Predicciones de la CNN para a) círculo, b) cilindro, c) cuadrado, d) rectángulo, e) triángulo y f) triángulo equilátero.

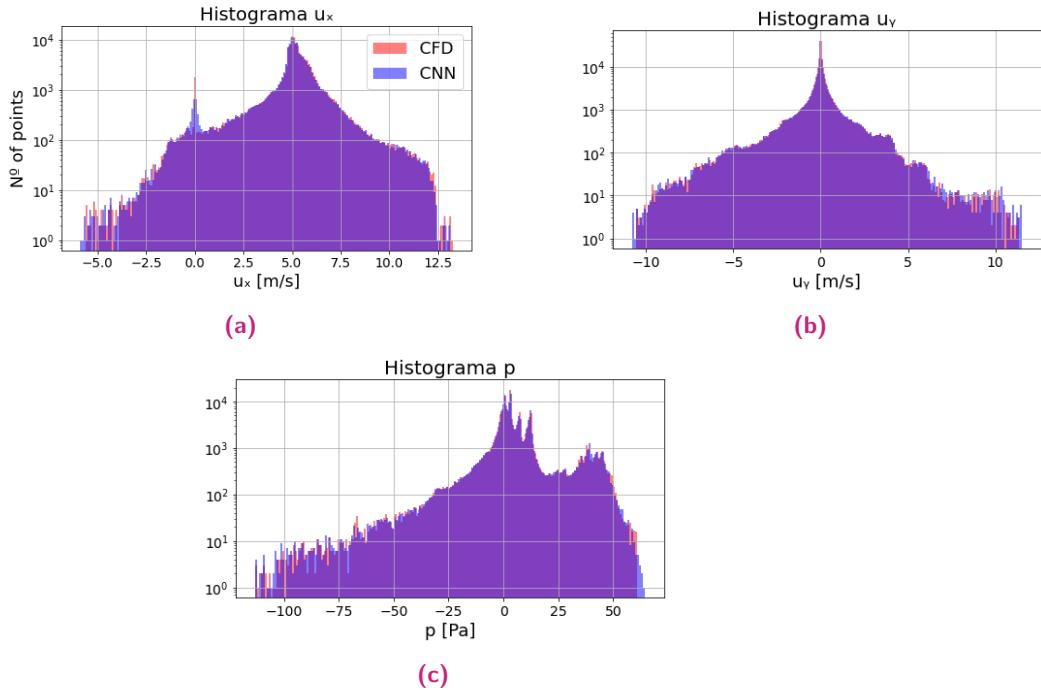


Figura 5.16: Distribución de los datos para: a) v_x , b) v_y y c) p .

50 instantes tarda 1,56, 3,21 y 1,84 segundos. El entrenamiento de los modelos neuronales de la red que predice los instantes futuros para geometrías variables, proporciona unos tiempos de entrenamiento de 2,60, 1,45 y 2,72 horas, dando un total de 6,77 horas. La predicción de 50 instantes lleva 1,76, 3,77 y 6,06 segundos. Por último, el modelo neuronal que predice el instante inicial ha sido entrenado durante 19,66 minutos y las predicciones son de 0,08 segundos. En la tabla 5.11 se muestran las comparativas de la duración de las predicciones por parte de los modelos neuronales con respecto a la duración de las simulaciones CFD. En el caso de las predicciones de las muestras futuras, se compara el tiempo necesario para predecir 50 muestras de las tres variables. Para la predicción de la muestra inicial, se compara el tiempo necesario para predecir una única muestra. Además, se ha de tener en cuenta el tiempo ahorrado gracias a la aplicación de la técnica de *data augmentation*, ya que evita la generación de un mayor número de simulaciones CFD. Esta técnica ahorra la generación de 50 nuevas simulaciones por cada geometría, que se traduce en un ahorro aproximado de 120 segundos por geometría.

Red	Duración simulaciones CFD (s)	Duración predicciones (s)	Reducción de tiempo
Velocidades de entrada variables	540,00	6,61	81,69
Geometrías variables	720,00	11,59	62,12
Muestra inicial	720,00	0,08	9000

Tabla 5.11: Comparación del tiempo de cálculo de los campos de velocidades paralela al flujo y vertical y de presión.

Conclusiones y trabajos futuros

6.1 Discusión general

Las simulaciones CFD son de gran utilidad para el estudio del comportamiento de los flujos turbulentos al enfrentarse a determinadas geometrías. No obstante, el coste computacional de estas simulaciones es demasiado elevado para ciertas aplicaciones en las que se requieren resultados rápidos. Asimismo, la precisión de estas simulaciones dependen también de la influencia que tiene el usuario en la generación del mallado y el modelo de turbulencia. Las técnicas de DL, como numerosos estudios demuestran, son eficaces para lograr aproximaciones rápidas y relativamente precisas.

En este estudio, se ha demostrado la flexibilidad de la estructura U-Net en su aplicación a diferentes situaciones fluido dinámicas. Esta estructura es capaz de predecir los campos de velocidades paralela al flujo y vertical y de presión para características variables del flujo. Asimismo, se demuestra no solo la capacidad de predecir una muestra de las variables analizadas, sino que se logra predecir, con errores relativamente pequeños hasta un número concreto de muestras, los instantes futuros del fluido.

6.2 Conclusiones específicas

En primer lugar, se han logrado unas aproximaciones relativamente precisas de los campos de velocidades paralela al flujo y vertical y de presión con una geometría circular y velocidad de entrada del fluido variable durante 20 instantes, partiendo de una inicial. A partir de la muestra 20, aparece un destacado incremento de los errores absolutos, especialmente en la velocidad paralela al flujo y en las velocidades de entrada más elevadas. En la velocidad vertical y la presión, las aproximaciones son generalmente más fidedignas, incluso para las muestras de los instantes más avanzados.

En el caso de la predicción de los instantes futuros de las geometrías variables, ocurre de forma similar al de la velocidad de entrada variable. Hasta la muestra del instante número 20, las predicciones son relativamente fiables. Los errores son más elevados en la velocidad paralela al flujo, que en la velocidad vertical y la presión. En este caso, los vórtices son captados adecuadamente, quedando las mayores tasas de error en las zonas contiguas a las geometrías.

Se confirma que la técnica de *data augmentation* empleada es eficaz para incrementar el número de muestras necesarias para el entrenamiento, de forma que no sea necesario llevar a cabo nuevas simulaciones CFD para el aumento de los datos. Gracias a ello, el modelo neuronal generado predice con precisión los campos de velocidades paralela al flujo, vertical y de presión.

Respecto al objetivo de reducir el coste computacional, el primer modelo neuronal predice 81,69 veces más rápido, el segundo 62,12 veces y el tercero, 9000. En los tres casos se cumple el objetivo de reducir el coste computacional, pero gracias a la técnica de *data augmentation* aplicada, en la predicción de la primera muestra el coste computacional se reduce drásticamente.

6.3 Líneas de trabajo futuras

El campo de la aplicación de DL a las simulaciones CFD es muy amplio. Son muchas y variadas las características fluido-dinámicas con las que se puede trabajar. El comportamiento del fluido varía ampliamente según el tipo de fluido estudiado y sus magnitudes de temperatura, densidad, velocidad, número de Reynolds...

Respecto a futuros estudios, cabe la posibilidad de aplicar la CNN con estructura U-Net a geometría con un comportamiento aerodinámico superior, como los perfiles aerodinámicos. Otro posible enfoque correspondería con la aplicación de técnicas de DL para predecir la mejor forma de un perfil aerodinámico con un flap de Gurney añadido, en busca de la optimización aerodinámica.

Presupuesto y planificación

7.1 Presupuesto

Concepto	Amortización ordenador portátil	Amortización ordenador remoto
Coste producto	1000 €	3000 €
Tiempo de amortización	4 años	4 años
Tiempo de trabajo	1700 horas/año	1700 horas/año
Tiempo de trabajo total	6800 horas	6800 horas
Coste por hora	0,15 €/hora	0,44 €/hora

Tabla 7.1: Amortizaciones.

Concepto	Coste unitario	Cantidad	Coste total	Porcentaje
Costes directos			25630,20 €	72,46 %
Coste tiempo invertido	50 €/hora	420 horas	21000,00 €	81,93 %
Coste simulaciones CFD	400 €/sim.	11 sim.	4400,00 €	17,17 %
Amortización ordenador portátil	0,15 €/hora	420 horas	63,00 €	0,25 %
Amortización ordenador remoto	0,44 €/hora	380 horas	167,20 €	0,65 %
Costes indirectos			3844,53 €	10,87 %
		15 % directos	3844,53 €	100,00 %
TOTAL Costes			29474,73 €	
Beneficio industrial			5894,95 €	16,67 %
		20 % costes	5894,95 €	100,00 %
TOTAL			35369,68 €	100,00 %
IVA		21 % total	7427,63 €	
TOTAL + IVA			42797,31 €	

Tabla 7.2: Presupuesto.

7.2 Planificación

Tarea	2022				2023																		
	Octubre		Noviembre		Diciembre		Enero			Febrero			Marzo			Abril			Mayo				
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3
CNN para velocidades de entrada variables																							
Estado del arte de CFD																							
Desarrollo de la aplicación																							
Tests iniciales																							
Solución																							
Ánalisis de resultados																							
Documentación																							
CNN para geometrías variables																							
Estado del arte de CFD																							
Desarrollo de la aplicación																							
Tests iniciales																							
Solución																							
Ánalisis de resultados																							
Documentación																							
Documentación final																							
Documentación final																							

Referencias bibliográficas

- [1]Alvaro Abucide-Armas, Koldo Portal-Porras, Unai Fernandez-Gamiz, Ekaitz Zulueta y Adrian Teso-Fz-Betoño. „A Data Augmentation-Based Technique for Deep Learning Applied to CFD Simulations“. En: *Mathematics* 9.16 (4 de ago. de 2021), pág. 1843 (vid. págs. [16, 29]).
- [2]Lionel Agostini. „Exploration and prediction of fluid dynamical systems using auto-encoder technology“. En: *Physics of Fluids* 32.6 (1 de jun. de 2020), pág. 067103 (vid. pág. [16]).
- [3]Saad Albawi, Tareq Abed Mohammed y Saad Al-Zawi. „Understanding of a convolutional neural network“. En: *2017 International Conference on Engineering and Technology (ICET)*. 2017 International Conference on Engineering and Technology (ICET). Antalya: IEEE, ago. de 2017, págs. 1-6 (vid. pág. [12]).
- [4]Iñigo Aramendia, Unai Fernandez-Gamiz, Ekaitz Zulueta Guerrero, Jose Lopez-Gude y Javier Sancho. „Power Control Optimization of an Underwater Piezoelectric Energy Harvester“. En: *Applied Sciences* 8.3 (7 de mar. de 2018), pág. 389 (vid. pág. [18]).
- [5]Han Bao, Jinyong Feng, Nam Dinh y Hongbin Zhang. „Computationally efficient CFD prediction of bubbly flow using physics-guided deep learning“. En: *International Journal of Multiphase Flow* 131 (oct. de 2020), pág. 103378 (vid. pág. [15]).
- [6]Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser et al. „Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI“. En: *Information Fusion* 58 (jun. de 2020), págs. 82-115 (vid. pág. [1]).
- [7]Richard Bellman. *An introduction to artificial intelligence: can computers think?* San Francisco: Boyd & Fraser Pub. Co, 1978. 146 págs. (vid. pág. [5]).
- [8]Eugene Charniak y Drew V. McDermott. *Introduction to artificial intelligence*. Reading, Mass: Addison-Wesley, 1985. 701 págs. (vid. pág. [5]).
- [9]Liang Deng, Yueqing Wang, Yang Liu et al. „A CNN-based vortex identification method“. En: *Journal of Visualization* 22.1 (feb. de 2019), págs. 65-78 (vid. pág. [15]).
- [10]Rui Fang, David Sondak, Pavlos Protopapas y Sauro Succi. „Deep learning for turbulent channel flow“. En: *arXiv:1812.02241 [physics]* (5 de dic. de 2018). arXiv: [1812.02241](https://arxiv.org/abs/1812.02241) (vid. pág. [16]).
- [11]Peter A. Flach. *Machine learning: the art and science of algorithms that make sense of data*. OCLC: ocn795181906. Cambridge ; New York: Cambridge University Press, 2012. 396 págs. (vid. pág. [6]).

- [12]M.W Gardner y S.R Dorling. „Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences“. En: *Atmospheric Environment* 32.14 (ago. de 1998), págs. 2627-2636 (vid. pág. [11]).
- [13]Francisco J. Gonzalez y Maciej Balajewicz. „Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems“. En: *arXiv:1808.01346 [physics]* (22 de ago. de 2018). arXiv: [1808.01346](https://arxiv.org/abs/1808.01346) (vid. pág. [16]).
- [14]Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016. 775 págs. (vid. pág. [11]).
- [15]Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza et al. *Generative Adversarial Networks*. arXiv:1406.2661. type: article. arXiv, 10 de jun. de 2014. arXiv: [1406.2661\[cs, stat\]](https://arxiv.org/abs/1406.2661) (vid. pág. [12]).
- [16]Xiaoxiao Guo, Wei Li y Francesco Iorio. „Convolutional Neural Networks for Steady Flow Approximation“. En: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco California USA: ACM, 13 de ago. de 2016, págs. 481-490 (vid. págs. [13], [15], [16], [23]).
- [17]Botros N. Hanna, Nam T. Dinh, Robert W. Youngblood e Igor A. Bolotnov. „Coarse-Grid Computational Fluid Dynamic (CG-CFD) Error Prediction using Machine Learning“. En: *arXiv:1710.09105 [physics]* (25 de oct. de 2017). arXiv: [1710.09105](https://arxiv.org/abs/1710.09105) (vid. pág. [15]).
- [18]John Haugeland. *Artificial intelligence: the very idea*. 6. print. Bradford books. Cambridge, Mass.: MIT Press, 1993. 287 págs. (vid. pág. [5]).
- [19]G. Iaccarino, A. Ooi, P.A. Durbin y M. Behnia. „Reynolds averaged simulation of unsteady separated flow“. En: *International Journal of Heat and Fluid Flow* 24.2 (abr. de 2003), págs. 147-156 (vid. pág. [18]).
- [20]D. Jakhar e I. Kaur. „Artificial intelligence, machine learning and deep learning: definitions and differences“. En: *Clinical and Experimental Dermatology* 45.1 (ene. de 2020), págs. 131-132 (vid. pág. [7]).
- [21]Ali Kashefi, Davis Rempe y Leonidas J. Guibas. „A Point-Cloud Deep Learning Framework for Prediction of Fluid Flow Fields on Irregular Geometries“. En: *Physics of Fluids* 33.2 (1 de feb. de 2021), pág. 027104. arXiv: [2010.09469](https://arxiv.org/abs/2010.09469) (vid. pág. [15]).
- [22]Ryan King, Oliver Hennigh, Arvind Mohan y Michael Chertkov. „From Deep to Physics-Informed Learning of Turbulence: Diagnostics“. En: *arXiv:1810.07785 [nlin, physics:physics, stat]* (5 de dic. de 2018). arXiv: [1810.07785](https://arxiv.org/abs/1810.07785) (vid. pág. [16]).
- [23]Diederik P. Kingma y Jimmy Ba. „Adam: A Method for Stochastic Optimization“. En: (2014). Publisher: arXiv Version Number: 9 (vid. pág. [24]).
- [24]Max Kuhn y Kjell Johnson. *Applied predictive modeling*. OCLC: ocn827083441. New York: Springer, 2013. 600 págs. (vid. pág. [9]).
- [25]Ray Kurzweil. *The age of intelligent machines*. 3. print. Cambridge, Mass: MIT Press, 1999. 565 págs. (vid. pág. [5]).
- [26]Yann LeCun, Yoshua Bengio y Geoffrey Hinton. „Deep learning“. En: *Nature* 521.7553 (28 de mayo de 2015), págs. 436-444 (vid. pág. [7]).

- [27]Sangseung Lee y Donghyun You. „Prediction of laminar vortex shedding over a cylinder using deep learning“. En: *arXiv:1712.07854 [physics]* (21 de dic. de 2017). arXiv: [1712.07854](https://arxiv.org/abs/1712.07854) (vid. pág. 15).
- [28]Jichao Li, Xiaosong Du y Joaquim R. R. A. Martins. *Machine Learning in Aerodynamic Shape Optimization*. arXiv:2202.07141. type: article. arXiv, 14 de feb. de 2022. arXiv: [2202.07141 \[physics\]](https://arxiv.org/abs/2202.07141) (vid. pág. 1).
- [29]Julia Ling, Andrew Kurzawski y Jeremy Templeton. „Reynolds averaged turbulence modelling using deep neural networks with embedded invariance“. En: *Journal of Fluid Mechanics* 807 (25 de nov. de 2016), págs. 155-166 (vid. pág. 15).
- [30]Yang Liu, Yutong Lu, Yueqing Wang et al. „A CNN-based shock detection method in flow visualization“. En: *Computers & Fluids* 184 (abr. de 2019), págs. 1-9 (vid. pág. 15).
- [31]Ilya Loshchilov y Frank Hutter. „Decoupled Weight Decay Regularization“. En: (2017). Publisher: arXiv Version Number: 3 (vid. pág. 24).
- [32]*MATLAB - MathWorks*. URL: <https://www.mathworks.com/products/matlab.html> (visitado 20 de nov. de 2022) (vid. pág. 23).
- [33]Romit Maulik, Bethany Lusch y Prasanna Balaprakash. „Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders“. En: *Physics of Fluids* 33.3 (1 de mar. de 2021), pág. 037106. arXiv: [2002.00470](https://arxiv.org/abs/2002.00470) (vid. pág. 16).
- [34]Arvind Mohan, Don Daniel, Michael Chertkov y Daniel Livescu. „Compressed Convolutional LSTM: An Efficient Deep Learning framework to Model High Fidelity 3D Turbulence“. En: *arXiv:1903.00033 [nlin, physics:physics]* (4 de mar. de 2019). arXiv: [1903.00033](https://arxiv.org/abs/1903.00033) (vid. pág. 16).
- [35]Nils J. Nilsson. *Artificial intelligence: a new synthesis*. 5th print. San Francisco, Calif: Kaufmann, 2003. 513 págs. (vid. pág. 6).
- [36]Hashem Nowruzi, Hassan Ghassemi y Mahmoud Ghiasi. „Performance predicting of 2D and 3D submerged hydrofoils using CFD and ANNs“. En: *Journal of Marine Science and Technology* 22.4 (dic. de 2017), págs. 710-733 (vid. pág. 16).
- [37]David L. Poole, Alan K. Mackworth y Randy Goebel. *Computational intelligence: a logical approach*. New York: Oxford University Press, 1998. 558 págs. (vid. pág. 6).
- [38]Koldo Portal-Porras, Unai Fernandez-Gamiz, Ainara Ugarte-Anero, Ekaitz Zulueta y Asier Zulueta. „Alternative Artificial Neural Network Structures for Turbulent Flow Velocity Field Prediction“. En: *Mathematics* 9.16 (14 de ago. de 2021), pág. 1939 (vid. pág. 16).
- [39]Mateus Dias Ribeiro, Abdul Rehman, Sheraz Ahmed y Andreas Dengel. „DeepCFD: Efficient Steady-State Laminar Flow Approximation with Deep Convolutional Neural Networks“. En: *arXiv:2004.08826 [physics]* (2 de mayo de 2020). arXiv: [2004.08826](https://arxiv.org/abs/2004.08826) (vid. págs. 15, 19).
- [40]Elaine Rich y Kevin Knight. *Artificial intelligence*. 2. ed. New York: McGraw-Hill, 1991. 621 págs. (vid. pág. 5).

- [41]Olaf Ronneberger, Philipp Fischer y Thomas Brox. „U-Net: Convolutional Networks for Biomedical Image Segmentation“. En: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. por Nassir Navab, Joachim Hornegger, William M. Wells y Alejandro F. Frangi. Vol. 9351. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, págs. 234-241 (vid. págs. [16, 19]).
- [42]Stuart J. Russell, Peter Norvig y Ernest Davis. *Artificial intelligence: a modern approach*. 3rd ed. Prentice Hall series in artificial intelligence. Upper Saddle River: Prentice Hall, 2010. 1132 págs. (vid. pág. [5]).
- [43]Alex Sherstinsky. „Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network“. En: *Physica D: Nonlinear Phenomena* 404 (mar. de 2020), pág. 132306 (vid. pág. [12]).
- [44]Siemens Software. URL: <https://www.plm.automation.siemens.com/global/en/> (visitado 20 de nov. de 2022) (vid. págs. [18, 26]).
- [45]Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever y Ruslan Salakhutdinov. „Dropout: a simple way to prevent neural networks from overfitting“. En: *The journal of machine learning research* 15.1 (2014). Publisher: JMLR.org, págs. 1929-1958 (vid. pág. [11]).
- [46]Jun Tao y Gang Sun. „Application of deep learning based multi-fidelity surrogate model to robust aerodynamic design optimization“. En: *Aerospace Science and Technology* 92 (sep. de 2019), págs. 722-737 (vid. pág. [14]).
- [47]Nils Thuerey, Konstantin Weißenow, Lukas Prantl y Xiangyu Hu. „Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows“. En: *AIAA Journal* 58.1 (ene. de 2020), págs. 25-36 (vid. pág. [16]).
- [48]Jordi Torres. *Python deep learning: introducción práctica con Keras y TensorFlow 2*. OCLC: 1148195914. Barcelona: Marcombo, 2020 (vid. pág. [6]).
- [49]Sun-Chong Wang. *Interdisciplinary Computing in Java Programming*. OCLC: 852790067. Boston, MA: Springer US, 2003 (vid. pág. [6]).
- [50]John F. Wendt, John D. Anderson y Von Karman Institute for Fluid Dynamics, eds. *Computational fluid dynamics: an introduction*. 3rd ed. OCLC: ocn288984495. Berlin ; [London]: Springer, 2008. 332 págs. (vid. pág. [14]).
- [51]Patrick Henry Winston. *Artificial intelligence*. 3rd ed. Reading, Mass: Addison-Wesley Pub. Co, 1992. 737 págs. (vid. pág. [5]).
- [52]Xinghui Yan, Jihong Zhu, Minchi Kuang y Xiangyang Wang. „Aerodynamic shape optimization using a novel optimizer based on machine learning techniques“. En: *Aerospace Science and Technology* 86 (mar. de 2019), págs. 826-835 (vid. pág. [15]).
- [53]Li Yang y Abdallah Shami. „On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice“. En: *Neurocomputing* 415 (nov. de 2020), págs. 295-316. arXiv: [2007.15745 \[cs, stat\]](https://arxiv.org/abs/2007.15745) (vid. pág. [9]).
- [54]Xinshuai Zhang, Fangfang Xie, Tingwei Ji, Zaoxu Zhu y Yao Zheng. „Multi-fidelity deep neural network surrogate model for aerodynamic shape optimization“. En: *Computer Methods in Applied Mechanics and Engineering* 373 (ene. de 2021), pág. 113485 (vid. pág. [14]).

**MÁSTER UNIVERSITARIO EN INGENIERÍA DE CONTROL,
AUTOMATIZACIÓN Y ROBÓTICA**

TRABAJO FIN DE MÁSTER

ANEXO A: ESQUEMAS Y PROGRAMAS FUENTE

***DISEÑO Y SIMULACIÓN INTELIGENTES DE
SISTEMAS FLUIDODINÁMICOS EN ESTADO NO
ESTACIONARIO MEDIANTE DEEP LEARNING***

Estudiante Abucide, Armas, Álvaro
Director/Directora Zulueta, Guerrero, Ekaitz
Departamento Ingeniería de Sistemas y Automática
Curso académico 2022-2023

Bilbao, 10 de julio de 2023

Esquemas y programas fuente

A.1 Carga de archivos CSV e interpolación

```
1 import pickle
2 import matplotlib.pyplot as plt
3 from scipy.io import loadmat
4 import numpy as np
5 # from functions.create_dataX_function import create_dataX
6 # from functions.load_CSVfunc_TIME import loadCSV
7 # from functions.load_CSVfunc_fromHardDisc import loadCSV
8 from scipy.interpolate import griddata
9
10 import pandas as pd
11 from os import listdir
12 from pathlib import Path
13 import csv
14 from numba import njit
15
16 @njit
17 def visualize3(sample_y, s):
18
19     minu = np.min(sample_y[s, 0, :, :])
20     maxu = np.max(sample_y[s, 0, :, :])
21
22     minv = np.min(sample_y[s, 1, :, :])
23     maxv = np.max(sample_y[s, 1, :, :])
24
25     minp = np.min(sample_y[s, 2, :, :])
26     maxp = np.max(sample_y[s, 2, :, :])
27
28     plt.figure()
29     fig = plt.gcf()
30     fig.set_size_inches(15, 10)
31     plt.suptitle("Time instant " + str(s+1), fontsize=20)
32     plt.subplot(3, 1, 1)
33     plt.title('CFD', fontsize=18)
```

```

34     plt.imshow((sample_y[s, 0, :, :]), cmap='jet', vmin = minu,
35     ↪   vmax = maxu, origin='lower', extent=[0,172,0,79])
36     plt.colorbar(orientation='horizontal')
37     plt.ylabel('Ux', fontsize=18)
38
39     plt.subplot(3, 1, 2)
40     plt.imshow((sample_y[s, 1, :, :]), cmap='jet', vmin = minv,
41     ↪   vmax = maxv, origin='lower', extent=[0,172,0,79])
42     plt.colorbar(orientation='horizontal')
43     plt.ylabel('Uy', fontsize=18)
44
45     plt.subplot(3, 1, 3)
46     plt.imshow((sample_y[s, 2, :, :]), cmap='jet', vmin = minp,
47     ↪   vmax = maxp, origin='lower', extent=[0,172,0,79])
48     plt.colorbar(orientation='horizontal')
49     plt.ylabel('p', fontsize=18)
50     plt.tight_layout()
51     plt.show()
52
53 def data_manip(dataY):
54
55     Y=np.zeros([len(dataY),3,81,174])
56     for i in range(len(dataY)):
57         for j in range (3):
58             points=dataY[i][:,4:6] #X e Y.
59             values=dataY[i][:,j] #Pressure, velocity[i],
60             ↪   velocity[j]
61
62             # Interpolación:
63             grid_x,
64             ↪   grid_y=np.mgrid[min(points[:,0]):max(points[:,0]):174j,
65             ↪   min(points[:,1]):max(points[:,1]):81j]
66             data_interpolated=griddata(points, values, (grid_x,
67             ↪   grid_y), method='linear')
68             data_interpolated=np.moveaxis(data_interpolated,0,-1)
69             Y[i,j,:,:]=data_interpolated
70             print("Interpolación nº: " + str(i + 1))
71             Y=Y[:, :, 1:80, 1:173]
72             ## Cambiar el orden de pressure y velocities para que estén en
73             ↪   la misma forma
74             ## que y en la red neuronal.

```

```

68     ## 1º Velocity[i], 2º Velocity[j], 3º Pressure.
69     Y=Y[:,[1,2,0],:,:,]
70
71     return Y
72
73 if __name__=="__main__":
74     dataY=[]
75     for i in range (3000,5001,1):
76         file =
77             pd.read_csv('./Unai-ANN-Abucide-casos20-25-2alamenos4/' +
78                         'XYZ_Internal_Table_Datos_25ms__' +
79                         str(i) +
80                         '.csv', header=0)
81         data_array = file.to_numpy()
82         dataY.append(data_array)
83         print ("Archivo nº" + str(i))
84         print(file)
85     with open('csv_25ms_3000_5000.pkl', 'wb') as f:
86         pickle.dump(dataY, f)

```

A.2 Generar las matrices de SDF y FRC

```

84 function [px, py] = geometry_points_v2(geometry, aug_factor,
85     ↪ size_factor, rotation_angle, position)
86     switch geometry
87         case "circle"
88             rx = 10*172/(180);
89             ry = 10*79/(180);
90             a = 172/2; % Center in x
91             b = 79/2; % Center in y
92             th = 0:pi/50:2*pi;
93             px = rx * cos(th) + a;
94             py = ry * sin(th) + b;
95         case "triangle"
96             px = [89 85 98];
97             py = [59 40 43];
98         case "triangle_eq"
99             px = [77 92 87];
100            py = [37 34 46];
101        case "ellipse"

```

```

101      px = [77 77 78 78 78 79 80 81 82 83 84 85 86 87 88 89
102          ↪ 90 91 92 ...
103          93 94 94 94 95 95 95 95 95 95 95 95 94 93 92 92 90
104          ↪ 89 88 ...
105          87 86 85 84 83 83 82 81 80 80 79 79 78 78 78];
106      py = [35 34 33 32 31 30 30 29 29 30 30 30 31 32 32 33
107          ↪ 34 35 ...
108          36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 50 51
109          ↪ 51 50 ...
110          50 50 49 48 48 47 46 45 44 43 42 41 40 39 38 37];
111      case "square"
112          px = [77 82 93 89];
113          py = [42 32 36 46];
114      case "rectangle"
115          px = [78 84 96 87];
116          py = [48 26 30 52];
117      otherwise
118          disp("Geometry not considered.")
119      end
120
121      % Cambiar la posición de la figura
122      px = px + position(1);
123      py = py + position(2);
124
125      % Cambiar el tamaño de la figura para el data augmentation
126      p = polyshape(px, py);
127      [cx, cy] = centroid(p);
128      p2 = scale(p, aug_factor, [cx, cy]);
129      px = p2.Vertices(:,1)';
130      py = p2.Vertices(:,2)';
131
132      % Cambiar el tamaño de la figura
133      p = polyshape(px, py);
134      [cx, cy] = centroid(p);
135      p2 = scale(p, size_factor, [cx, cy]);
136      px = p2.Vertices(:,1)';
137      py = p2.Vertices(:,2)';
138
139      % Cambiar la orientación de la figura
140      p = polyshape(px, py);
141      [cx, cy] = centroid(p);
142      p2 = rotate(p, rotation_angle, [cx, cy]);

```

```

139     px = p2.Vertices(:,1)';
140     py = p2.Vertices(:,2)';
141
142 end

143 geometries = ["circle", "ellipse", "rectangle", "square",
144   ↪ "triangle", "triangle_eq"];
145 % aug_factor = linspace(0.93, 1.07, 50);
146 aug_factor = 1;
147 size_factor = 1;
148 rotation_angle = 1;
149 position = [1, 1; 1, 1];
150 for geom = 1:length(geometries)
151   for aug = 1:length(aug_factor)
152     for size = 1:length(size_factor)
153       for rot_angle = 1:length(rotation_angle)
154         for pos = 1:length(position)
155           [px, py] = geometry_points_v2(geometries(geom),
156             ↪ aug_factor(aug), size_factor(size), ...
157             ↪ rotation_angle(rot_angle),
158             ↪ position(pos,:));
159           Z_matrix = sdf(px, py)';
160           % save("Datos para entrenamiento/SDF_geometrias/" +
161             ↪ geometries(geom) + ".mat", "Z_matrix")
162           % save("Datos para
163             ↪ entrenamiento/SDF_geometrias_data_augmentation/Test/" +
164             ↪ + geometries(geom) + "_" + aug_factor(aug) +
165             ↪ "size_factor" + size_factor(size) + "rot_angle" +
166             ↪ rotation_angle(rot_angle) + "pos_x" +
167             ↪ position(pos,1) + "pos_y" + position(pos,2) +
168             ↪ ".mat", "Z_matrix")
169           frc = FRC_geometries(px, py)';
170           figure
171           imagesc(frc')
172           % save("Datos para entrenamiento/FRC_geometrias/" +
173             ↪ geometries(geom) + ".mat", "frc")

```

```

163    % save("Datos para
164        →  entrenamiento/FRC_geometrias_data_augmentation/Test/"
165        →  + geometries(geom) + "_" + aug_factor(aug) +
166        →  "size_factor" + size_factor(size) + "rot_angle" +
167        →  rotation_angle(rot_angle) + "pos_x" +
168        →  position(pos,1) + "pos_y" + position(pos,2) +
169        →  ".mat", "frc")
170    end
171
172    end
173
174 end

```

A.3 *Data augmentation*

Generar los datos adicionales de las entradas de la CNN:

```

169 from scipy.io import loadmat
170 import numpy as np
171 from os import listdir
172 import pickle
173
174 def generate_dataX():
175
176     SDF = []
177     FRC = []
178     filepaths = [f for f in
179                 →  listdir("SDF_geometrias_data_augmentation/Train") if
180                 →  f.endswith('.mat')]
181     for file in filepaths:
182
183         →  SDF.append(loadmat("SDF_geometrias_data_augmentation/Train/" +
184         →  + file)['Z_matrix'])
185     filepaths = [f for f in
186                 →  listdir("FRC_geometrias_data_augmentation/Train") if
187                 →  f.endswith('.mat')]
188     for file in filepaths:
189
190         →  FRC.append(loadmat("FRC_geometrias_data_augmentation/Train/" +
191         →  + file)['frc'])

```

```

184     SDF2 = pickle.load(open("dataX.pkl", "rb"))[len(filepaths), 2,
185     ↪   :, :]
186
187     dataX = np.zeros([len(filepaths), 3, 172, 79])
188
189     dataX[:,0,:,:] = np.array(SDF)
190     dataX[:,1,:,:] = np.array(FRC)
191     dataX[:,2,:,:] = SDF2
192
193     return dataX

```

Generar los datos adicionales de los campos de velocidades paralela al flujo y vertical y de presión:

```

193 import numpy as np
194 import pickle
195 from scipy.interpolate import griddata
196 from scipy.io import loadmat
197
198 geom_index = ["1", "63", "121", "723", "1305", "1923"]
199
200 # a = np.linspace(0.93, 1.07, 50)
201
202 a = loadmat("aug_factor.mat")["aug_factor"]
203 a = np.reshape(a, [50])
204
205 for geom in geom_index:
206     dataY = pickle.load(open('E:/NuevasGeometrias/csv_geom' + geom
207     ↪   + '_1001_1500.pkl', 'rb'))
208     # Y=np.zeros([len(dataY)*len(a),3,172, 79])
209     Y=np.zeros([len(a),3,172, 79])
210
211     print("Geometry " + geom)
212
213     for i in range(3):
214         points=dataY[i][:,5:7] #X e Y.
215         values_p=dataY[i][:,1] # Pressure (Pa)
216         values_vx=dataY[i][:,2] # Velocity x (m/s)
217         values_vy=dataY[i][:,3] # Velocity y (m/s)
218
219         print("Interpolación nº : " + str(i + 1) + " out of " +
220           ↪   str(len(dataY)))

```

```

219
220     for k in range(len(a)):
221         # for k in range(1):
222
223         grid_x_p, grid_y_p =
224             np.mgrid[min(points[:,0]):max(points[:,0]):256j/a[k],
225                     min(points[:,1]):max(points[:,1]):128j/a[k]]
226         data_interpolated_p = griddata(points, values_p,
227             (grid_x_p, grid_y_p), method='linear')/a[k]**2
228
229         print("Pressure data augmentation: " + str(k) + " out
230               of " + str(len(a)))
231
232         grid_x_vx, grid_y_vx =
233             np.mgrid[min(points[:,0]):max(points[:,0]):256j/a[k],
234                     min(points[:,1]):max(points[:,1]):128j/a[k]]
235         data_interpolated_vx = griddata(points, values_vx,
236             (grid_x_vx, grid_y_vx), method='linear')/a[k]
237
238         print("Vx data augmentation: " + str(k) + " out of " +
239               str(len(a)))
240
241         grid_x_vy, grid_y_vy =
242             np.mgrid[min(points[:,0]):max(points[:,0]):256j/a[k],
243                     min(points[:,1]):max(points[:,1]):128j/a[k]]
244         data_interpolated_vy = griddata(points, values_vy,
245             (grid_x_vy, grid_y_vy), method='linear')/a[k]
246
247         print("Vy data augmentation: " + str(k) + " out of " +
248               str(len(a)))
249
250         points_b_p = np.array([grid_x_p.flatten(),
251                               grid_y_p.flatten()]).T
252         values_b_p = data_interpolated_p.flatten()
253
254         points_b_vx = np.array([grid_x_vx.flatten(),
255                               grid_y_vx.flatten()]).T
256         values_b_vx = data_interpolated_vx.flatten()
257
258         points_b_vy = np.array([grid_x_vy.flatten(),
259                               grid_y_vy.flatten()]).T
260         values_b_vy = data_interpolated_vy.flatten()

```

```

246
247     print("Pressure interpolation initialized")
248
249     grid_x_b_p, grid_y_b_p =
250         np.mgrid[min(points_b_p[:,0]):max(points_b_p[:,0]):174j,
251             min(points_b_p[:,1]):max(points[:,1]):81j]
252     data_interpolated_b_p = griddata(points_b_p, values_b_p,
253         (grid_x_b_p, grid_y_b_p), method='linear')
254     data_interpolated_b_p =
255         data_interpolated_b_p[1:173,1:80]
256
257     print("Pressure interpolation finished")
258     print("Vx interpolation initialized")
259
260     grid_x_b_vx, grid_y_b_vx =
261         np.mgrid[min(points_b_vx[:,0]):max(points_b_vx[:,0]):174j,
262             min(points_b_vx[:,1]):max(points[:,1]):81j]
263     data_interpolated_b_vx = griddata(points_b_vx,
264         values_b_vx, (grid_x_b_p, grid_y_b_vx),
265         method='linear')
266     data_interpolated_b_vx =
267         data_interpolated_b_vx[1:173,1:80]
268
269     print("Vx interpolation finished")
270     print("Vy interpolation initialized")
271
272     grid_x_b_vy, grid_y_b_vy =
273         np.mgrid[min(points_b_vy[:,0]):max(points_b_vy[:,0]):174j,
274             min(points_b_vy[:,1]):max(points[:,1]):81j]
275     data_interpolated_b_vy = griddata(points_b_vy,
276         values_b_vy, (grid_x_b_vy, grid_y_b_vy),
277         method='linear')
278     data_interpolated_b_vy =
279         data_interpolated_b_vy[1:173,1:80]
280
281     print("Vy interpolation finished")
282
283     Y[k, 0, :, :] = data_interpolated_b_vx
284     Y[k, 1, :, :] = data_interpolated_b_vy
285     Y[k, 2, :, :] = data_interpolated_b_p

```

```

273     with open('dataY_ ' + geom + '_geometries_augmented_f.pkl',
274             'wb') as f:
275         pickle.dump(Y, f)
276
277
278
279
280 def randomize_CFD_input_data():
281
282     y_circle =
283         → pickle.load(open("./dataY_1_geometries_augmented_e.pkl",
284                         "rb"))[:50] # CFD
285     y_ellipse =
286         → pickle.load(open("./dataY_1305_geometries_augmented_e.pkl",
287                         "rb"))[:50] # CFD
288     y_rectangle =
289         → pickle.load(open("./dataY_723_geometries_augmented_e.pkl",
290                         "rb"))[:50] # CFD
291     y_square =
292         → pickle.load(open("./dataY_63_geometries_augmented_e.pkl",
293                         "rb"))[:50] # CFD
294     y_triangle =
295         → pickle.load(open("./dataY_1923_geometries_augmented_e.pkl",
296                         "rb"))[:50] # CFD
297     y_triangle_eq =
298         → pickle.load(open("./dataY_121_geometries_augmented_e.pkl",
299                         "rb"))[:50] # CFD
300
301     y_list = [y_circle, y_ellipse, y_rectangle, y_square,
302             → y_triangle, y_triangle_eq]
303     length = len(y_circle) + len(y_ellipse) + len(y_rectangle) +
304             → len(y_square) + len(y_triangle) + len(y_triangle_eq)
305     length_list = [len(y_circle), len(y_ellipse), len(y_rectangle),
306                   → len(y_square), len(y_triangle), len(y_triangle_eq)]
307     length1 = len(y_circle)
308     length2 = length1 + len(y_ellipse)
309     length3 = length2 + len(y_rectangle)
310     length4 = length3 + len(y_square)
311     length5 = length4 + len(y_triangle)
312     print("Long. 1: ", length1)

```

```

298     print("Long. 2: ", length2)
299     print("Long. 3: ", length3)
300     print("Long. 4: ", length4)
301     print("Long. 5: ", length5)
302
303     print("Tamaño total: ", length)
304
305     geometries = ["circle", "ellipse", "rectangle", "square",
306                   "triangle", "triangle_eq"]
306
307     data_x = generate_dataX()
308
309     lengths = [length1, length2, length3, length4, length5]
310
311     x = np.concatenate((data_x[:length1,:,:,:],
312                         data_x[length1:length2,:,:,:],
313                         data_x[length2:length3,:,:,:],
314                         data_x[length3:length4,:,:,:],
315                         data_x[length4:length5,:,:,:],
316                         data_x[length5:,:,:,:]))
317
318
319     y = np.zeros([length, 3, 172, 79])
320     y[:length1,:,:,:] = y_list[0]
321     y[length1:length2,:,:,:] = y_list[1]
322     y[length2:length3,:,:,:] = y_list[2]
323     y[length3:length4,:,:,:] = y_list[3]
324     y[length4:length5,:,:,:] = y_list[4]
325     y[length5:,:,:,:] = y_list[5]
326
327     return y, x

```

A.4 Generar los datos de entrenamiento

Código que organiza los datos de entrada y salida para el entrenamiento de la CNN:

```

328 import numpy as np
329 import pickle
330 import random
331 from create_dataX import *

```

```

332
333 def randomize_CFD_input_data(variable, n_test):
334     velocities = [5, 10, 15, 20, 25]
335
336     y_circle = pickle.load(open("./Y_geom1_interpolated.pkl",
337                               "rb")) # CFD
337     y_ellipse = pickle.load(open("./Y_geom1305_interpolated.pkl",
338                               "rb")) # CFD
338     y_rectangle = pickle.load(open("./Y_geom723_interpolated.pkl",
339                               "rb")) # CFD
339     y_square = pickle.load(open("./Y_geom63_interpolated.pkl",
340                               "rb")) # CFD
340     y_triangle = pickle.load(open("./Y_geom1923_interpolated.pkl",
341                               "rb")) # CFD
341     y_triangle_eq = pickle.load(open("./Y_geom121_interpolated.pkl",
342                               "rb")) # CFD
342
343     y_list = [y_circle, y_ellipse, y_rectangle, y_square,
344               y_triangle, y_triangle_eq]
344     length = len(y_circle) + len(y_ellipse) + len(y_rectangle) +
345               len(y_square) + len(y_triangle) + len(y_triangle_eq)
345     length_list = [len(y_circle), len(y_ellipse), len(y_rectangle),
346                   len(y_square), len(y_triangle), len(y_triangle_eq)]
346     length1 = len(y_circle)-1
347     length2 = length1 + len(y_ellipse) - 1
348     length3 = length2 + len(y_rectangle) - 1
349     length4 = length3 + len(y_square) - 1
350     length5 = length4 + len(y_triangle) - 1
351     print("Long. 1: ", length1)
352     print("Long. 2: ", length2)
353     print("Long. 3: ", length3)
354     print("Long. 4: ", length4)
355     print("Long. 5: ", length5)
356
357     length_list_b = [length1, length2, length3, length4, length5]
358
359     # y = np.zeros([length-5, 1, 172, 79])
360     print("Tamaño total: ", length)
361
362     geometries = ["circle", "ellipse", "rectangle", "square",
363                   "triangle", "triangle_eq"]
363

```

```

364     data_x = generate_dataX(geometries)
365
366     x=np.zeros([length,2,79,172])
367     x[:length1+1,:,:,:]=data_x[0]
368     x[length1+1:length2+2,:,:,:]=data_x[1]
369     x[length2+2:length3+3,:,:,:]=data_x[2]
370     x[length3+3:length4+4,:,:,:]=data_x[3]
371     x[length4+4:length5+5,:,:,:]=data_x[4]
372     x[length5+5:,:,:,:]=data_x[5]
373
374     x_new = np.zeros([length-6,3,79,172])
375     y_new = np.zeros([length-6,1,79,172])
376
377     y_new[:length1], x_new[:length1] = add_t_1_instant(y_list[0],
378             ↵ x[:length1+1], variable)
378     y_new[length1:length2], x_new[length1:length2] =
379             ↵ add_t_1_instant(y_list[1], x[length1+1:length2+2],
380             ↵ variable)
380     y_new[length2:length3], x_new[length2:length3] =
381             ↵ add_t_1_instant(y_list[2], x[length2+2:length3+3],
382             ↵ variable)
382     y_new[length3:length4], x_new[length3:length4] =
383             ↵ add_t_1_instant(y_list[3], x[length3+3:length4+4],
384             ↵ variable)
384     y_new[length4:length5], x_new[length4:length5] =
385             ↵ add_t_1_instant(y_list[4], x[length4+4:length5+5],
386             ↵ variable)
386     y_new[length5:], x_new[length5:] = add_t_1_instant(y_list[5],
387             ↵ x[length5+5:], variable)
388
389     test_index_list = []
390     y_train = y_new
391     x_train = x_new
392     for i in range(n_test):
393         y_train, x_train, test_index, length_list_b =
394             ↵ test_samples(y_train, x_train, length_list,
395             ↵ length_list_b)
396         test_index_list.append(test_index)
397         print("Test n°: ", i+1)
398     return y_train, x_train, y_new, x_new, test_index_list,
399             ↵ length_list
400

```

```

393 def add_t_1_instant(y, x, variable):
394     print("Tamaño original: ", y.shape)
395     if variable == "vx":
396         y2=y[:, :, 1, :, :] # Select the variable to simulate vx(:1),
397             ↪ vy(1:2), p(2:3)
398     elif variable == "vy":
399         y2=y[:, 1:2, :, :] # Select the variable to simulate
400             ↪ vx(:1), vy(1:2), p(2:3)
401     else:
402         y2=y[:, 2:3, :, :] # Select the variable to simulate
403             ↪ vx(:1), vy(1:2), p(2:3)
404     print("Tamaño tras selección de la variable: ", y2.shape)
405     y=y2
406     x2=np.zeros([len(y), 3, 79, 172])
407     x2[:, :, 2, :, :] = x
408     # y=np.moveaxis(y, 2, 3)
409     x2[:, 2:3, :, :] = y # Add the t-1 instant to the input data
410     x=x2[:-1, :, :, :]
411     y=y[1:, :, :, :]
412     print("Tamaño tras eliminar una muestra: ", y.shape)
413     return(y, x)

414 def test_samples(y_train, x_train, length_list, length_list_b):

415     test_index = []

416     random_value_circle = (random.randint(0, length_list[0]-50))
417     random_value_ellipse = (random.randint(0, length_list[1]-50))
418     random_value_rectangle = (random.randint(0, length_list[2]-50))
419     random_value_square = (random.randint(0, length_list[3]-50))
420     random_value_triangle = (random.randint(0, length_list[4]-50))
421     random_value_triangle_eq = (random.randint(0,
422                                         ↪ length_list[5]-50))

423

424     test_index = [random_value_circle, random_value_ellipse,
425                   ↪ random_value_rectangle,
426                   random_value_square, random_value_triangle,
427                   ↪ random_value_triangle_eq]
428     y_train = np.delete(y_train, np.add(test_index[1:],
429                           ↪ length_list_b), 0)
430     print("length y: ", y_train.shape)

```

```

428     x_train = np.delete(x_train, np.add(test_index[1:], 
429     ↪   length_list_b), 0)
430     print("length x: ", x_train.shape)
431     length_list_b = np.add(length_list_b, [-1 -2 -3 -4 -5])
432     # length_list_b = length_list_b.to_list()
433     print("Lista de longitudes de array:", length_list_b)
434     return y_train, x_train, test_index, length_list_b
435
436
437 # from random_samples_function import *
438 from data_generation_for_CNN_first_sample_data_augmentation import
439     *
440
441 # y, x, y_all, x_all, test_index_list, length_list =
442     ↪   randomize_CFD_input_data("p",50)
443 y, x = randomize_CFD_input_data()
444
445
446 with open("training_test_data_geometries_first_sample_f.pkl", "wb")
447     ↪   as file:
448         # pickle.dump([y, x, y_all, x_all, test_index_list,
449             ↪   length_list], file)
450         pickle.dump([y, x], file)

```

A.5 CNN

El modelo de la CNN viene dado por los siguientes códigos:

```

444 import torch
445 import torch.nn as nn
446 import torch.nn.functional as F
447 from torch.nn.utils import weight_norm
448
449 def create_layer(in_channels, out_channels, kernel_size, wn=True,
450     ↪   bn=True,
451             activation=nn.ReLU, convolution=nn.Conv2d):
452     assert kernel_size % 2 == 1
453     layer = []
454     conv = convolution(in_channels, out_channels, kernel_size,
455         ↪   padding=kernel_size // 2)
456     if wn:
457         conv = weight_norm(conv)
458     layer.append(conv)

```

```

457     if activation is not None:
458         layer.append(activation())
459     if bn:
460         layer.append(nn.BatchNorm2d(out_channels))
461     return nn.Sequential(*layer)
462
463
464 class AutoEncoder(nn.Module):
465     def __init__(self, in_channels, out_channels, kernel_size=3,
466      ↪ filters=[16, 32, 64],
467      ↪ weight_norm=True, batch_norm=True,
468      ↪ activation=nn.ReLU, final_activation=None):
469     super().__init__()
470     assert len(filters) > 0
471     encoder = []
472     decoder = []
473     for i in range(len(filters)):
474         if i == 0:
475             encoder_layer = create_layer(in_channels,
476             ↪ filters[i], kernel_size, weight_norm,
477             ↪ batch_norm, activation, nn.Conv2d)
478             decoder_layer = create_layer(filters[i],
479             ↪ out_channels, kernel_size, weight_norm, False,
480             ↪ final_activation, nn.ConvTranspose2d)
481         else:
482             encoder_layer = create_layer(filters[i-1],
483             ↪ filters[i], kernel_size, weight_norm,
484             ↪ batch_norm, activation, nn.Conv2d)
485             decoder_layer = create_layer(filters[i],
486             ↪ filters[i-1], kernel_size, weight_norm,
487             ↪ batch_norm, activation, nn.ConvTranspose2d)
488         encoder = encoder + [encoder_layer]
489         decoder = [decoder_layer] + decoder
490     self.encoder = nn.Sequential(*encoder)
491     self.decoder = nn.Sequential(*decoder)
492
493     def forward(self, x):
494         return self.decoder(self.encoder(x))
495
496
497     import torch
498     import torch.nn as nn
499     import torch.nn.functional as F

```

```

488 from torch.nn.utils import weight_norm
489 from Models.AutoEncoder import create_layer
490
491
492 def create_encoder_block(in_channels, out_channels, kernel_size,
493     ↪ wn=True, bn=True,
494         activation=nn.ReLU, layers=2):
495     encoder = []
496     for i in range(layers):
497         _in = out_channels
498         _out = out_channels
499         if i == 0:
500             _in = in_channels
501         encoder.append(create_layer(_in, _out, kernel_size, wn, bn,
502             ↪ activation, nn.Conv2d))
503     return nn.Sequential(*encoder)
504
505
506 def create_decoder_block(in_channels, out_channels, kernel_size,
507     ↪ wn=True, bn=True,
508         activation=nn.ReLU, layers=2, final_layer=False):
509     decoder = []
510     for i in range(layers):
511         _in = in_channels
512         _out = in_channels
513         _bn = bn
514         _activation = activation
515         if i == 0:
516             _in = in_channels * 2
517         if i == layers - 1:
518             _out = out_channels
519             if final_layer:
520                 _bn = False
521                 _activation = None
522             decoder.append(create_layer(_in, _out, kernel_size, wn, _bn,
523                 ↪ _activation, nn.ConvTranspose2d))
524     return nn.Sequential(*decoder)
525
526
527 def create_encoder(in_channels, filters, kernel_size, wn=True,
528     ↪ bn=True, activation=nn.ReLU, layers=2):
529     encoder = []
530     for i in range(len(filters)):

```

```

525     if i == 0:
526         encoder_layer = create_encoder_block(in_channels,
527             ↳ filters[i], kernel_size, wn, bn, activation,
528             ↳ layers)
529     else:
530         encoder_layer = create_encoder_block(filters[i-1],
531             ↳ filters[i], kernel_size, wn, bn, activation,
532             ↳ layers)
533     encoder = encoder + [encoder_layer]
534
535 return nn.Sequential(*encoder)

536
537 def create_decoder(out_channels, filters, kernel_size, wn=True,
538     ↳ bn=True, activation=nn.ReLU, layers=2):
539     decoder = []
540     for i in range(len(filters)):
541         if i == 0:
542             decoder_layer = create_decoder_block(filters[i],
543                 ↳ out_channels, kernel_size, wn, bn, activation,
544                 ↳ layers, final_layer=True)
545         # elif i == len(filters):
546         #     decoder_layer = create_decoder_block(velocity_channel,
547         #         ↳ filters[i], kernel_size, wn, bn, activation, layers,
548         #         ↳ final_layer=True)
549         else:
550             decoder_layer = create_decoder_block(filters[i],
551                 ↳ filters[i-1], kernel_size, wn, bn, activation,
552                 ↳ layers, final_layer=False)
553     decoder = [decoder_layer] + decoder
554
555 return nn.Sequential(*decoder)

556
557 class UNetEx(nn.Module):
558     def __init__(self, in_channels, out_channels, kernel_size=3,
559         ↳ filters=[16, 32, 64], layers=3,
560         ↳ weight_norm=True, batch_norm=True, activation=nn.ReLU,
561         ↳ final_activation=None):
562         super().__init__()
563         assert len(filters) > 0
564         self.final_activation = final_activation
565         self.encoder = create_encoder(in_channels, filters,
566             ↳ kernel_size, weight_norm, batch_norm, activation,
567             ↳ layers)
568         decoders = []

```

```

552     for i in range(out_channels):
553         decoders.append(create_decoder(1, filters, kernel_size,
554                                     ↴ weight_norm, batch_norm, activation, layers))
555     self.decoders = nn.Sequential(*decoders)
556
557     def encode(self, x):
558         tensors = []
559         indices = []
560         sizes = []
561         for encoder in self.encoder:
562             x = encoder(x)
563             sizes.append(x.size())
564             tensors.append(x)
565             x, ind = F.max_pool2d(x, 2, 2, return_indices=True)
566             indices.append(ind)
567         return x, tensors, indices, sizes
568
569     def decode(self, _x, _tensors, _indices, _sizes):
570         y = []
571         for _decoder in self.decoders:
572             x = _x
573             tensors = _tensors[:]
574             indices = _indices[:]
575             sizes = _sizes[:]
576             for decoder in _decoder:
577                 tensor = tensors.pop()
578                 size = sizes.pop()
579                 ind = indices.pop()
580                 x = F.max_unpool2d(x, ind, 2, 2, output_size=size)
581                 x = torch.cat([tensor, x], dim=1)
582                 x = decoder(x)
583                 y.append(x)
584         return torch.cat(y, dim=1)
585
586     def forward(self, x):
587         x, tensors, indices, sizes = self.encode(x)
588         x = self.decode(x, tensors, indices, sizes)
589         if self.final_activation is not None:
590             x = self.final_activation(x)
591
592         return x

```

Funciones para el entrenamiento de la red neuronal:

```

591 import copy
592 import torch
593
594 def generate_metrics_list(metrics_def):
595     list = {}
596     for name in metrics_def.keys():
597         list[name] = []
598     return list
599
600 def epoch(scope, loader, on_batch=None, training=False):
601     model = scope["model"]
602     optimizer = scope["optimizer"]
603     loss_func = scope["loss_func"]
604     metrics_def = scope["metrics_def"]
605     scope = copy.copy(scope)
606     scope["loader"] = loader
607
608     metrics_list = generate_metrics_list(metrics_def)
609     total_loss = 0
610     if training:
611         model.train()
612     else:
613         model.eval()
614     for tensors in loader:
615         if "process_batch" in scope and scope["process_batch"] is
616             not None:
617             tensors = scope["process_batch"](tensors)
618         if "device" in scope and scope["device"] is not None:
619             tensors = [tensor.to(scope["device"]) for tensor in
620                         tensors]
621         loss, output = loss_func(model, tensors)
622         if training:
623             optimizer.zero_grad()
624             loss.backward()
625             optimizer.step()
626         total_loss += loss.item()
627         scope["batch"] = tensors
628         scope["loss"] = loss
629         scope["output"] = output
630         scope["batch_metrics"] = {}
631         for name, metric in metrics_def.items():
632             value = metric["on_batch"](scope)

```

```

631         scope["batch_metrics"][name] = value
632         metrics_list[name].append(value)
633     if on_batch is not None:
634         on_batch(scope)
635     scope["metrics_list"] = metrics_list
636     metrics = {}
637     for name in metrics_def.keys():
638         scope["list"] = scope["metrics_list"][name]
639         metrics[name] = metrics_def[name]["on_epoch"](scope)
640     return total_loss, metrics
641
642 def train(scope, train_dataset, val_dataset, patience=10,
643           ↪ batch_size=256, print_function=print, eval_model=None,
644           ↪ on_train_batch=None, on_val_batch=None,
645           ↪ on_train_epoch=None, on_val_epoch=None,
646           ↪ after_epoch=None):
647     epochs = scope["epochs"]
648     model = scope["model"]
649     metrics_def = scope["metrics_def"]
650     scope = copy.copy(scope)
651
652     scope["best_train_metric"] = None
653     scope["best_train_loss"] = float("inf")
654     scope["best_val_metrics"] = None
655     scope["best_val_loss"] = float("inf")
656     scope["best_model"] = None
657
658     train_loader = torch.utils.data.DataLoader(train_dataset,
659           ↪ batch_size=batch_size, shuffle=True)
660     val_loader = torch.utils.data.DataLoader(val_dataset,
661           ↪ batch_size=batch_size, shuffle=False)
662     skips = 0
663     for epoch_id in range(1, epochs + 1):
664         scope["epoch"] = epoch_id
665         print_function("Epoch #" + str(epoch_id))
666         # Training
667         scope["dataset"] = train_dataset
668         train_loss, train_metrics = epoch(scope, train_loader,
669             ↪ on_train_batch, training=True)
670         scope["train_loss"] = train_loss
671         scope["train_metrics"] = train_metrics
672         print_function("\tTrain Loss = " + str(train_loss))

```

```

667     for name in metrics_def.keys():
668         print_function("\tTrain " + metrics_def[name]["name"] +
669             " = " + str(train_metrics[name]))
670
671     if on_train_epoch is not None:
672         on_train_epoch(scope)
673
674     del scope["dataset"]
675
676     # Validation
677     scope["dataset"] = val_dataset
678
679     with torch.no_grad():
680         val_loss, val_metrics = epoch(scope, val_loader,
681             → on_val_batch, training=False)
682
683     scope["val_loss"] = val_loss
684     scope["val_metrics"] = val_metrics
685
686     print_function("\tValidation Loss = " + str(val_loss))
687
688     for name in metrics_def.keys():
689         print_function("\tValidation " +
690             → metrics_def[name]["name"] + " = " +
691             → str(val_metrics[name]))
692
693     if on_val_epoch is not None:
694         on_val_epoch(scope)
695
696     del scope["dataset"]
697
698     # Selection
699
700     is_best = None
701
702     if eval_model is not None:
703         is_best = eval_model(scope)
704
705     if is_best is None:
706         is_best = val_loss < scope["best_val_loss"]
707
708     if is_best:
709
710         scope["best_train_metric"] = train_metrics
711         scope["best_train_loss"] = train_loss
712         scope["best_val_metrics"] = val_metrics
713         scope["best_val_loss"] = val_loss
714
715         scope["best_model"] = copy.deepcopy(model)
716         print_function("Model saved!")
717
718         skips = 0
719
720     else:
721
722         skips += 1
723
724     if after_epoch is not None:
725         after_epoch(scope)
726
727
728     return scope["best_model"], scope["best_train_metric"],
729             → scope["best_train_loss"], \

```

```

    scope["best_val_metrics"], scope["best_val_loss"]

705
706 def train_model(model, loss_func, train_dataset, val_dataset,
707     ↪ optimizer, process_batch=None, eval_model=None,
708         ↪ on_train_batch=None, on_val_batch=None,
709             ↪ on_train_epoch=None, on_val_epoch=None,
710                 ↪ after_epoch=None,
711                     epochs=100, batch_size=256, patience=10, device=0,
712                         ↪ **kwargs):
713     model = model.to(device)
714     scope = {}
715     scope["model"] = model
716     scope["loss_func"] = loss_func
717     scope["train_dataset"] = train_dataset
718     scope["val_dataset"] = val_dataset
719     scope["optimizer"] = optimizer
720     scope["process_batch"] = process_batch
721     scope["epochs"] = epochs
722     scope["batch_size"] = batch_size
723     scope["device"] = device
724     metrics_def = {}
725     names = []
726     for key in kwargs.keys():
727         parts = key.split("_")
728         if len(parts) == 3 and parts[0] == "m":
729             if parts[1] not in names:
730                 names.append(parts[1])
731     for name in names:
732         if "m_" + name + "_name" in kwargs and "m_" + name +
733             ↪ "_on_batch" in kwargs and "m_" + name + "_on_epoch" in
734             ↪ kwargs:
735             metrics_def[name] = {
736                 "name": kwargs["m_" + name + "_name"],
737                 "on_batch": kwargs["m_" + name + "_on_batch"],
738                 "on_epoch": kwargs["m_" + name + "_on_epoch"],
739             }
740         else:
741             print("Warning: " + name + " metric is incomplete!")
742     scope["metrics_def"] = metrics_def
743     return train(scope, train_dataset, val_dataset,
744         ↪ eval_model=eval_model, on_train_batch=on_train_batch,

```

```

738         on_val_batch=on_val_batch, on_train_epoch=on_train_epoch,
739             ↪ on_val_epoch=on_val_epoch, after_epoch=after_epoch,
batch_size=batch_size, patience=patience)

```

Funciones necesarias para la ejecución del código de la red neuronal.

```

740 import torch.nn as nn
741 import numpy as np
742 from matplotlib import pyplot as plt
743
744 def split_tensors(*tensors, ratio):
745     assert len(tensors) > 0
746     split1, split2 = [], []
747     count = len(tensors[0])
748     for tensor in tensors:
749         assert len(tensor) == count
750         split1.append(tensor[:int(len(tensor) * ratio)])
751         split2.append(tensor[int(len(tensor) * ratio):])
752     if len(tensors) == 1:
753         split1, split2 = split1[0], split2[0]
754     return split1, split2
755
756 def initialize(model, gain=1, std=0.02):
757     for module in model.modules():
758         if type(module) in [nn.Linear, nn.Conv1d, nn.Conv2d,
759             ↪ nn.Conv3d]:
760             nn.init.xavier_normal_(module.weight, gain)
761             if module.bias is not None:
762                 nn.init.normal_(module.bias, 0, std)
763
764 def visualize(sample_y, out_y, error, s, variable, n_sample,
765             ↪ velocity):
766
767     minu = np.min(sample_y[s, 0, :, :])
768     maxu = np.max(sample_y[s, 0, :, :])
769
770     mineu = np.min(error[s, 0, :, :])
771     maxeu = np.max(error[s, 0, :, :])
772
773     plt.figure()
774     fig = plt.gcf()

```

```

773     plt.suptitle("Input velocity: " + velocity + " ms. Sample nº: "
774         + str(n_sample+s), fontsize=20)
775     fig.set_size_inches(15, 10)
776     plt.subplot(3, 3, 1)
777     plt.title('CFD', fontsize=18)
778     plt.imshow(np.transpose(sample_y[s, 0, :, :]), cmap='jet', vmin
779         = minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
780     plt.colorbar(orientation='horizontal')
781     plt.ylabel(variable, fontsize=18)
782     plt.subplot(3, 3, 2)
783     plt.title('CNN', fontsize=18)
784     plt.imshow(np.transpose(out_y[s, 0, :, :]), cmap='jet', vmin =
785         minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
786     plt.colorbar(orientation='horizontal')
787     plt.subplot(3, 3, 3)
788     plt.title('Error', fontsize=18)
789     plt.imshow(np.transpose(error[s, 0, :, :]), cmap='jet', vmin =
790         mineu, vmax = maxeu, origin='lower', extent=[0,172,0,79])
791     plt.colorbar(orientation='horizontal')
792     plt.tight_layout()
793     plt.show()

```

Código para el entrenamiento de la red que predice los instantes futuros con velocidades de entrada variables:

```

790 import os
791 import json
792 import torch
793 import pickle
794 import time
795 from train_functions import *
796 from functions import *
797 import torch.optim as optim
798 from torch.utils.data import TensorDataset
799 from Models.UNetEx import UNetEx
800 from Models.UNet import UNet
801
802 import numpy as np
803 from create_channels_weights_1variable import
804     create_channels_weights
805 from random_samples_function import *

```

```

806 import random
807
808 import gc
809
810 # torch.cuda.empty_cache()
811 # del variables
812 # gc.collect()
813 epoch = [1000]
814 ___batchsize = [32, 64]
815 learning_rate = [0.0001]
816 kernelsize = [3, 5, 7]
817 variable = ["vx", "vy", "p"]
818 for vb in variable:
819     cont = 25
820     for bs in range(len(batchsize)):
821         for lr in range(len(learning_rate)):
822             for ep in range(len(epoch)):
823                 for ks in range(len(kernelsize)):
824                     device = torch.device("cuda" if
825                         torch.cuda.is_available() else "cpu")
826                     with open('training_test_data_' + vb + '.pkl',
827                         'rb') as f:
828                         y, x, y_all, x_all, test_index_list,
829                         length_list = pickle.load(f)
830
831                         # Remove the possible NaN data
832                         # y_b=y
833                         # y[np.isnan(y)] = 0
834                         # x[np.isnan(x)] = 0
835
836                         x = torch.FloatTensor(x)
837                         y = torch.FloatTensor(y)
838                         # y_b = torch.FloatTensor(y_b)
839
840                         # Initial weights calculation
841                         # channels_weights =
842                         #     torch.sqrt(torch.mean(y.permute(0, 2, 3,
843                         #         1).reshape((len(y)*172*79,1)) ** 2,
844                         #         dim=0)).view(1, -1, 1, 1).to(device)
845                         channels_weights=create_channels_weights(y,
846                         device)

```

```

841     print("Channel weights: ", channels_weights)

842

843     # Simulation files directory
844     simulation_directory = "./Run/"
845     if not os.path.exists(simulation_directory):
846         os.makedirs(simulation_directory)

847

848     train_data, test_data = split_tensors(x, y,
849                                         ← ratio=0.7)

850

851     train_dataset, test_dataset =
852         ← TensorDataset(*train_data),
853         ← TensorDataset(*test_data)
854     test_x, test_y = test_dataset[:]

855

856     # Parameters
857     torch.manual_seed(0)
858     lr = learning_rate[l_r]
859     kernel_size = kernelsize[ks]
860     filters = [8, 16, 32, 32]
861     bn = False
862     wn = False
863     wd = 0.005
864     beta1 = 0.5
865     beta2 = 0.5

866

867     model = UNetEx(3, 1, filters=filters,
868                     ← kernel_size=kernel_size, batch_norm=bn,
869                     ← weight_norm=wn)
870     # model.load_state_dict(torch.load(
871     ← "Modelo_vx_Final.py")) #Cargamos los pesos
872     ← anteriores

873

874     # Define optimizer
875     optimizer =
876         ← torch.optim.AdamW(model.parameters(), lr=lr,
877         ← weight_decay=wd)

878

879     config = {}
880     train_loss_curve = []
881     test_loss_curve = []
882     train_ux_curve = []

```

```

874     test_ux_curve = []
875
876     def after_epoch(scope):
877
878         ↪ train_loss_curve.append(scope["train_loss"])
879         test_loss_curve.append(scope["val_loss"])
880         train_ux_curve.append(scope[
881             ↪ "train_metrics"]["ux"])
882         test_ux_curve.append(scope[
883             ↪ "val_metrics"]["ux"])
884
885
886     def loss_func(model, batch):
887         x, y = batch
888         output = model(x)
889
890         lossu = torch.abs((output[:,0,:,:] -
891                         y[:,0,:,:]).reshape((output.shape[0],1,
892                         output.shape[2],output.shape[3])))
893         loss = lossu / channels_weights
894         return torch.sum(loss), output
895
896
897     ti = time.time()
898
899     # Training model
900     DeepCFD, train_metrics, train_loss,
901         ↪ test_metrics, test_loss = train_model(model,
902         ↪ loss_func, train_dataset, test_dataset,
903         ↪ optimizer,
904             epochs=epoch[ep], batch_size=batchsize[ep],
905             ↪ device=device,
906             m_ux_name=vb + "MSE",
907             m_ux_on_batch=lambda scope:
908                 ↪ float(torch.sum(scope["output"][:,0,:,:]
909                     ↪ - scope["batch"][1][:,:,0,:,:])),
910             m_ux_on_epoch=lambda scope:
911                 ↪ sum(scope["list"]) /
912                 ↪ len(scope["dataset"]),
913             patience=25, after_epoch=after_epoch)
914
915
916     duration = time.time() - ti

```

```

901     print("Training time: " + str(duration//60) +
902           "min and " +
903           str("{:.2f}".format(duration%60)) + "sec")
904
905
906     # Guardar los el modelo
907     torch.save(model.state_dict(),"Modelo_" + vb +
908                 str(cont) + ".py")
909
910
911     metrics = {}
912     metrics["train_metrics"] = train_metrics
913     metrics["train_loss"] = train_loss
914     metrics["test_metrics"] = test_metrics
915     metrics["test_loss"] = test_loss
916     curves = {}
917     curves["train_loss_curve"] = train_loss_curve
918     curves["test_loss_curve"] = test_loss_curve
919     curves["train_ux_curve"] = train_ux_curve
920     curves["test_ux_curve"] = test_ux_curve
921     config["metrics"] = metrics
922     config["curves"] = curves
923
924
925     net_param = {}
926     net_param["num_epochs"] = epoch[ep]
927     net_param["learning_ratio"] =
928         learning_rate[l_r]
929     net_param["kernel_size"] = kernelsize[ks]
930     net_param["batch_size"] = batchsize[bs]
931     config["net_param"] = net_param
932     config["training_duration"] = duration
933
934
935     velocity = ["05", "10", "15", "20", "25"] # The
936         simulated velocity
937     variable = vb # Simulated variable
938     with open(simulation_directory + "results_ms_"
939                 + vb + str(cont) + ".json", "w") as file:
940         json.dump([config, duration, net_param],
941                   file)
942
943
944     # Test 5ms
945     y_5ms = y_all[:length_list[0]]
946     x_5ms = x_all[:length_list[0]]

```

```

936     # Test 10ms
937     y_10ms =
938         ↳ y_all[length_list[0]:sum(length_list[:2])]
939     x_10ms =
940         ↳ x_all[length_list[0]:sum(length_list[:2])]
941     # Test 15ms
942     y_15ms =
943         ↳ y_all[sum(length_list[:2]):sum(length_list[:3])]
944     x_15ms =
945         ↳ x_all[sum(length_list[:2]):sum(length_list[:3])]
946     # Test 20ms
947     y_20ms =
948         ↳ y_all[sum(length_list[:3]):sum(length_list[:4])]
949     x_20ms =
950         ↳ x_all[sum(length_list[:3]):sum(length_list[:4])]
951     # Test 25ms
952     y_25ms = y_all[sum(length_list[:4]):]
953     x_25ms = x_all[sum(length_list[:4]):]

954
955
956
957
958
959
960
961
962
963
964
965

def test_CNN(n_instantes, x_ms, y_ms,
             ↳ simulation_directory, velocity, variable,
             ↳ n_test, vb, cont):

    ti = time.time()

    out = torch.empty((n_instantes, 3, 172, 79))
    x_ms = torch.FloatTensor(x_ms)
    y_ms = torch.FloatTensor(y_ms)
    out[:, 0:2, :, :] = x_ms[0, :2, :, :]

    out[0, 2, :, :] =
        ↳ DeepCFD(x_ms[n_test:n_test+1].to(device))

    for i in range (1,n_instantes,1):
        print("Sample: " + str(i))
        out[i, 2, :, :] =
            ↳ DeepCFD(out[i-1:i].to(device))
        error =
            ↳ (torch.abs((out[:, 2:3, :, :].cpu() -
                         ↳ y_ms[n_test:n_test+n_instantes].cpu())))
        out2=out[:, 2:3, :, :]

```

```

966         out2=out2.reshape([n_instantes,1,172,79])

967

968     duration = time.time() - ti
969     print("Training time: " + str(duration//60)
970     ↪ + "min and " +
971     ↪ str("{:.2f}").format(duration%60)) +
972     ↪ "sec")

973

974     out = out2.cpu().detach().numpy()
975     error = error.cpu().detach().numpy()
976     y = y_ms[ n_test:n_test+n_instantes]
977     ↪ .cpu().detach().numpy()

978

979     with open(simulation_directory + "plots_" +
980     ↪ velocity + "ms_" + vb + str(cont) +
981     ↪ ".pkl", "wb") as file:
982         pickle.dump([out, error, y], file)

983

984     n_instantes = 50
985     for test in range(len(test_index_list)):
986         test_CNN(n_instantes, x_5ms, y_5ms,
987         ↪ simulation_directory, velocity[0],
988         ↪ variable, test_index_list[test][0],
989         ↪ vb, cont)
990         test_CNN(n_instantes, x_10ms, y_10ms,
991         ↪ simulation_directory, velocity[1],
992         ↪ variable, test_index_list[test][1],
993         ↪ vb, cont)
994         test_CNN(n_instantes, x_15ms, y_15ms,
995         ↪ simulation_directory, velocity[2],
996         ↪ variable, test_index_list[test][2],
997         ↪ vb, cont)
998         test_CNN(n_instantes, x_20ms, y_20ms,
999         ↪ simulation_directory, velocity[3],
1000         ↪ variable, test_index_list[test][3],
1001         ↪ vb, cont)
1002         test_CNN(n_instantes, x_25ms, y_25ms,
1003         ↪ simulation_directory, velocity[4],
1004         ↪ variable, test_index_list[test][4],
1005         ↪ vb, cont)

1006         cont = cont+1

```

Código para el entrenamiento de la red que predice los instantes futuros para geometrías variables:

```
992 import os
993 import json
994 import torch
995 import pickle
996 import time
997 from train_functions import *
998 from functions import *
999 import torch.optim as optim
1000 from torch.utils.data import TensorDataset
1001 from Models.UNetEx import UNetEx
1002 from Models.UNet import UNet
1003
1004 import numpy as np
1005 from create_channels_weights_1variable import
1006     ↵ create_channels_weights
1007 from random_samples_function import *
1008
1009 import random
1010
1011 # import gc
1012
1013 # torch.cuda.empty_cache()
1014 # del variables
1015 # gc.collect()
1016
1017 epoch = [2000] # 1000
1018 batchsize = [32, 64, 128] # 32
1019 learning_rate = [0.001, 0.0001] # 0.0001
1020 kernelsize = [3, 5, 7] # 3
1021 variable = ["vy", "p"]
1022
1023 # epoch = [1000]
1024 # batchsize = [32]
1025 # learning_rate = [0.001]
1026 # kernelsize = [3]
1027 # variable = ["vx"]
1028
1029 for vb in variable:
    cont = 200
```

```

1030     for bs in range(len(batchsize)):
1031         for l_r in range(len(learning_rate)):
1032             for ep in range(len(epoch)):
1033                 for ks in range(len(kernelsize)):
1034                     a = variable.index(vb) + bs + l_r + ep + ks
1035                     # if cont<20:
1036                     #     cont = cont + 1
1037
1038                     # else:
1039                     device = torch.device("cuda" if
1040                         torch.cuda.is_available() else "cpu")
1041                     with open('training_test_data_geometries_' + vb
1042                         + '.pkl', 'rb') as f:
1043                         y, x, y_all, x_all, test_index_list,
1044                             length_list = pickle.load(f)
1045
1046                     print(ep, l_r, ks)
1047
1048                     # Remove the possible NaN data
1049                     # y_b=y
1050                     # y[np.isnan(y)] = 0
1051                     # x[np.isnan(x)] = 0
1052
1053                     # Function that scales data in the (0, 1)
1054                         interval
1055                     def minmax_norm(df):
1056                         return (df - df.min()) / (df.max() -
1057                             df.min())
1058
1059                     # x=minmax_norm(x)
1060                     # y=minmax_norm(y)
1061
1062                     x = torch.FloatTensor(x)
1063                     y = torch.FloatTensor(y)
1064                     # y_b = torch.FloatTensor(y_b)

# Initial weights calculation

```

```

1065     # channels_weights =
1066     ↵ torch.sqrt(torch.mean(y.permute(0, 2, 3,
1067     ↵ 1).reshape((len(y)*172*79,1)) ** 2,
1068     ↵ dim=0)).view(1, -1, 1, 1).to(device)
1069     channels_weights=create_channels_weights(y,
1070     ↵ device)
1071     print("Channel weights: ", channels_weights)

1072
1073     # Simulation files directory
1074     simulation_directory = "./Run/"
1075     if not os.path.exists(simulation_directory):
1076         os.makedirs(simulation_directory)

1077
1078     plots_directory = "Run/Plots2/"
1079     if not os.path.exists(plots_directory):
1080         os.makedirs(plots_directory)

1081
1082     plots_subdirectory = "Run/Plots2/" + vb +
1083     ↵ str(cont) + "/"
1084     if not os.path.exists(plots_subdirectory):
1085         os.makedirs(plots_subdirectory)

1086     modelo_directory = "./Modelo2/"
1087     if not os.path.exists(modelo_directory):
1088         os.makedirs(modelo_directory)
1089     # Spliting dataset into 95% train and 5% test
1090     train_data, test_data = split_tensors(x, y,
1091     ↵ ratio=0.9)

1092
1093     train_dataset, test_dataset =
1094     ↵ TensorDataset(*train_data),
1095     ↵ TensorDataset(*test_data)
1096     test_x, test_y = test_dataset[:]

1097
1098     # Parameters
1099     torch.manual_seed(0)
1100     lr = learning_rate[l_r]
1101     kernel_size = kernelsize[ks]
1102     filters = [8, 16, 32, 32]
1103     bn = False
1104     wn = False
1105     wd = 0.005

```

```

1099         beta1 = 0.5
1100         beta2 = 0.5
1101
1102     model = UNetEx(3, 1, filters=filters,
1103                     ↵ kernel_size=kernel_size, batch_norm=bn,
1104                     ↵ weight_norm=wn)
1105     #
1106     ↵ model.load_state_dict(torch.load("Modelo_vx_Final.py"))#Ca
1107
1108     # Define opotimizer
1109     optimizer =
1110         ↵ torch.optim.AdamW(model.parameters(), lr=lr,
1111         ↵ weight_decay=wd)
1112
1113     config = {}
1114     train_loss_curve = []
1115     test_loss_curve = []
1116     train_ux_curve = []
1117     test_ux_curve = []
1118
1119     def after_epoch(scope):
1120
1121         ↵ train_loss_curve.append(scope["train_loss"])
1122         test_loss_curve.append(scope["val_loss"])
1123
1124         ↵ train_ux_curve.append(scope["train_metrics"]["ux"])
1125
1126         ↵ test_ux_curve.append(scope["val_metrics"]["ux"])
1127
1128     def loss_func(model, batch):
1129
1130         x, y = batch
1131         output = model(x)
1132
1133
1134         lossu = torch.abs((output[:,0,:,:] -
1135                         ↵ y[:,0,:,:])).reshape((output.shape[0],1,output.shape[2],
1136                         output.shape[3]))
1137         loss = lossu / channels_weights
1138         return torch.sum(loss), output
1139
1140
1141         ti = time.time()
1142
1143     # Training model

```

```

1131     DeepCFD, train_metrics, train_loss,
1132         ↵ test_metrics, test_loss = train_model(model,
1133             ↵ loss_func, train_dataset, test_dataset,
1134             ↵ optimizer,
1135                 epochs=epoch[ep], batch_size=batchsize[bs],
1136                     ↵ device=device,
1137                         m_ux_name="Ux MSE",
1138                         m_ux_on_batch=lambda scope:
1139                             ↵ float(torch.sum(scope["output"][:,0,:,:]
1140                                 ↵ - scope["batch"][1][:,:,0,:,:])),
1141                         m_ux_on_epoch=lambda scope:
1142                             ↵ sum(scope["list"]) /
1143                                 ↵ len(scope["dataset"]),
1144                         patience=25, after_epoch=after_epoch)

1145
1146     duration = time.time() - ti
1147     print("Training time: " + str(duration//60) +
1148         ↵ "min and " +
1149         ↵ str("{:.2f}".format(duration%60)) + "sec")

1150     # Guardar los el modelo
1151     torch.save(model.state_dict(), modelo_directory
1152         ↵ + "Modelo_geometries_" + vb + str(cont) +
1153         ↵ ".py")

1154     metrics = {}
1155     metrics["train_metrics"] = train_metrics
1156     metrics["train_loss"] = train_loss
1157     metrics["test_metrics"] = test_metrics
1158     metrics["test_loss"] = test_loss
1159     curves = {}
1160     curves["train_loss_curve"] = train_loss_curve
1161     curves["test_loss_curve"] = test_loss_curve
1162     curves["train_ux_curve"] = train_ux_curve
1163     curves["test_ux_curve"] = test_ux_curve
1164     config["metrics"] = metrics
1165     config["curves"] = curves

1166
1167     net_param = {}
1168     net_param["num_epochs"] = epoch[ep]
1169     net_param["learning_ratio"] =
1170         ↵ learning_rate[l_r]

```

```

1160     net_param["kernel_size"] = kernelsize[ks]
1161     net_param["batch_size"] = batchsize[bs]
1162     config["net_param"] = net_param
1163     config["training_duration"] = duration
1164
1165     geometries = ["circle", "ellipse", "rectangle",
1166                   "square", "triangle", "triangle_eq"] # The
1167                   ← simulated velocity
1168     # variable = "p" # Simulated variable
1169     # n_test = "1" # Number of test with this
1170           ← variable at this velocity
1171     with open(simulation_directory + "results_" +
1172               ← vb
1173               + "_geometries_" + str(cont) +
1174               ← ".json", "w") as file:
1175         json.dump(config, file)
1176
1177
1178     # Test circle
1179     y_circle = y_all[:length_list[0]]
1180     x_circle = x_all[:length_list[0]]
1181
1182     # Test ellipse
1183     y_ellipse =
1184         ← y_all[length_list[0]:sum(length_list[:2])]
1185     x_ellipse =
1186         ← x_all[length_list[0]:sum(length_list[:2])]
1187
1188     # Test rectangle
1189     y_rectangle =
1190         ← y_all[sum(length_list[:2]):sum(length_list[:3])]
1191     x_rectangle =
1192         ← x_all[sum(length_list[:2]):sum(length_list[:3])]
1193
1194     # Test square
1195     y_square =
1196         ← y_all[sum(length_list[:3]):sum(length_list[:4])]
1197     x_square =
1198         ← x_all[sum(length_list[:3]):sum(length_list[:4])]
1199
1200     # Test triangle
1201     y_triangle =
1202         ← y_all[sum(length_list[:4]):sum(length_list[:5])]
1203     x_triangle =
1204         ← x_all[sum(length_list[:4]):sum(length_list[:5])]
1205
1206     # Test equilateral triangle

```

```

1189     y_triangle_eq = y_all[sum(length_list[:5]):]
1190     x_triangle_eq = x_all[sum(length_list[:5]):]
1191
1192
1193     def test_CNN(n_instantes, x_ms, y_ms,
1194         ↵ simulation_directory, geometries,
1195             variable, n_test, plots_directory,
1196                 ↵ plots_subdirectory):
1197
1198         ti = time.time()
1199
1200         out = torch.empty((n_instantes,3,79,172))
1201         x_ms = torch.FloatTensor(x_ms)
1202         y_ms = torch.FloatTensor(y_ms)
1203         out[:,0:2,:,:]=x_ms[0,:2,:,:]
1204
1205         out[0,2,:,:] =
1206             ↵ DeepCFD(x_ms[n_test:n_test+1].to(device))
1207
1208         for i in range (1,n_instantes,1):
1209             print("Sample: " + str(i))
1210             out[i,2,:,:] =
1211                 ↵ DeepCFD(out[i-1:i].to(device))
1212             error =
1213                 ↵ (torch.abs((out[:,2:3,:,:].cpu() -
1214                     ↵ y_ms[n_test:n_test+n_instantes].cpu())))
1215             out2=out[:,2:3,:,:]
1216             out2=out2.reshape([n_instantes,1,79,172])
1217
1218             duration = time.time() - ti
1219             print("Training time: " + str(duration//60)
1220                 + "min and " +
1221                     str("{:.2f}".format(duration%60)) +
1222                         "sec")
1223
1224             # for s in range (n_instantes):
1225                 #
1226                     ↵ visualize3(y_ms[n_test:n_test+n_instantes].cpu().detach())
1227                 #
1228                     ↵ out2.cpu().detach().numpy(),
1229                     ↵ error.cpu().detach().numpy(), s,

```

```

1218         #           variable, n_test,
1219         #           ↵ geometries)
1220         out = out2.cpu().detach().numpy()
1221         error = error.cpu().detach().numpy()
1222
1223
1224         # with open(plots_subdirectory + "plots_" +
1225         #           ↵ geometries + "_" + vb + str(n_test) +
1226         #           ↵ ".pkl", "wb") as file:
1227             # pickle.dump([out, error, y], file)
1228
1229
1230     return out, error, y, geometries, n_test
1231
1232
1233
1234     plots = {}
1235
1236
1237     n_instantes = 50
1238     test_index_list[7][-1] = 300 # Este número de
1239     #           ↵ test estaba fuera de rango
1240     test_index_list[21][-1] = 265
1241     for test in range(30):
1242         print(test)
1243         out, error, y, geom, n_test =
1244             test_CNN(n_instantes, x_circle,
1245             #           ↵ y_circle, simulation_directory,
1246             #           ↵ geometries[0],
1247             #           ↵ vb, test_index_list[test][0],
1248             #           ↵ plots_directory,
1249             #           ↵ plots_subdirectory)
1250         plots[vb + '_' + geom + '_' + str(n_test)]
1251             = [y, out, error]
1252
1253         out, error, y, geom, n_test =
1254             test_CNN(n_instantes, x_ellipse,
1255             #           ↵ y_ellipse, simulation_directory,
1256             #           ↵ geometries[1],
1257             #           ↵ vb, test_index_list[test][1],
1258             #           ↵ plots_directory,
1259             #           ↵ plots_subdirectory)
1260         plots[vb + '_' + geom + '_' + str(n_test)]
1261             = [y, out, error]

```

```

1243     out, error, y, geom, n_test =
1244         ↳ test_CNN(n_instantes, x_rectangle,
1245             ↳ y_rectangle, simulation_directory,
1246             ↳ geometries[2],
1247                 vb, test_index_list[test][2],
1248                     ↳ plots_directory,
1249                         ↳ plots_subdirectory)
1250 plots[vb + '_' + geom + '_' + str(n_test)]
1251     ↳ = [y, out, error]
1252 out, error, y, geom, n_test =
1253     ↳ test_CNN(n_instantes, x_square,
1254         ↳ y_square, simulation_directory,
1255             ↳ geometries[3],
1256                 vb, test_index_list[test][3],
1257                     ↳ plots_directory,
1258                         ↳ plots_subdirectory)
1259 plots[vb + '_' + geom + '_' + str(n_test)]
1260     ↳ = [y, out, error]
1261 out, error, y, geom, n_test =
1262     ↳ test_CNN(n_instantes, x_triangle,
1263         ↳ y_triangle, simulation_directory,
1264             ↳ geometries[4],
1265                 vb, test_index_list[test][4],
1266                     ↳ plots_directory,
1267                         ↳ plots_subdirectory)
1268 plots[vb + '_' + geom + '_' + str(n_test)]
1269     ↳ = [y, out, error]
1270 out, error, y, geom, n_test =
1271     ↳ test_CNN(n_instantes, x_triangle_eq,
1272         ↳ y_triangle_eq, simulation_directory,
1273             ↳ geometries[5],
1274                 vb, test_index_list[test][5],
1275                     ↳ plots_directory,
1276                         ↳ plots_subdirectory)
1277 plots[vb + '_' + geom + '_' + str(n_test)]
1278     ↳ = [y, out, error]
1279
1280 with open(plots_subdirectory +
1281     ↳ "plots_geometries_" + vb + str(cont) +
1282         ↳ ".pkl", "wb") as file:
1283     pickle.dump(plots, file)

```

```
1258
1259         cont = cont+1
```

Código para el entrenamiento de la red que predice el primer instante para geometrías variables:

```
1260 from train_functions import *
1261 from functions import *
1262 import torch.optim as optim
1263 from torch.utils.data import TensorDataset
1264 from Models.UNetEx import UNetEx
1265
1266 import numpy as np
1267 from create_channels_weights import create_channels_weights
1268
1269 # epoch = [1000]
1270 # batchsize = [32, 64, 128]
1271 # learning_rate = [0.001, 0.0001]
1272 # kernelsize = [5, 7, 9]
1273 # filters_ = [[8, 16, 16, 32, 32], [8, 16, 32, 64]]
1274
1275 epoch = [1000, 2000]
1276 batchsize = [32, 64, 132]
1277 learning_rate = [0.001, 0.0001]
1278 kernelsize = [3, 5, 7, 9]
1279 filters_ = [[8, 16, 32, 32], [8, 16, 16, 32, 32]]
1280
1281 cont = 100
1282 for bs in range(len(batchsize)):
1283     for lr in range(len(learning_rate)):
1284         # for sp_rat in range(len(split_ratio)):
1285         for ep in range(len(epoch)):
1286             for ks in range(len(kernelsize)):
1287                 for filt in range(len(filters_)):
1288                     device = torch.device("cuda" if
1289                         torch.cuda.is_available() else "cpu")
1290                     y, x = pickle.load(open(
1291                         "./training_test_data_geometries_first_sample_e.pkl",
1292                         "rb"))
1293
1294                 def minmax_norm(df):
```

```

1292         return (df - df.min()) / ( df.max() -
1293                                     ↵ df.min())
1294
1295     minimize = 0
1296
1297     if minimize == 1:
1298         for j in range (3):
1299             if j == 2:
1300                 y[:,j,:,:]=minmax_norm(y[:,j,:,:])
1301             else:
1302                 x[:,j,:,:]=minmax_norm(x[:,j,:,:])
1303                 y[:,j,:,:]=minmax_norm(y[:,j,:,:])
1304
1305     y_b=y
1306
1307     y[ np.isnan(y) ] = 0
1308
1309
1310     x = torch.FloatTensor(x)
1311     y = torch.FloatTensor(y)
1312     y_b = torch.FloatTensor(y_b)
1313
1314
1315     channels_weights=create_channels_weights(y_b,
1316                                              ↵ device)
1317
1318
1319     # x = x.permute(0,1,3,2)
1320     # y = y.permute(0,1,3,2)
1321     # channels_weights =
1322     #   ↵ torch.sqrt(torch.mean(y_cw)).view(1, -1, 1,
1323     #   ↵ 1).to(device)
1324     # channels_weights =
1325     #   ↵ torch.sqrt(torch.mean(y.permute(0, 2, 3,
1326     #   ↵ 1).reshape((981*172*79,3)) ** 2,
1327     #   ↵ dim=0)).view(1, -1, 1, 1).to(device)
1328     print(channels_weights)
1329
1330
1331     # Simulation files directory
1332     simulation_directory = "./Run/"
1333     if not os.path.exists(simulation_directory):
1334         os.makedirs(simulation_directory)
1335
1336
1337     plots_directory = "Run/Plots_first_sample/"
1338     if not os.path.exists(plots_directory):
1339         os.makedirs(plots_directory)

```

```

1327
1328     plots_subdirectory = "Run/Plots_first_sample/_"
1329     ↵ + str(cont) + "_/"
1330
1331     if not os.path.exists(plots_subdirectory):
1332         os.makedirs(plots_subdirectory)
1333
1334
1335
1336     modelo_directory = "./Modelo_first_sample/"
1337
1338     if not os.path.exists(modelo_directory):
1339         os.makedirs(modelo_directory)
1340
1341
1342     idx = torch.randperm(x.shape[0])
1343     x = x[idx].view(x.size())
1344     y = y[idx].view(y.size())
1345
1346     train_data, test_data = split_tensors(x, y,
1347     ↵ ratio=0.7)
1348
1349     train_dataset, test_dataset =
1350     ↵ TensorDataset(*train_data),
1351     ↵ TensorDataset(*test_data)
1352
1353     test_x, test_y = test_dataset[:]
1354
1355
1356     torch.manual_seed(0)
1357     lr = learning_rate[l_r]
1358     kernel_size = kernelsize[ks]
1359
1360     # filters = [8, 16, 32, 32]
1361     filters = filters_[filt]
1362
1363     bn = False
1364
1365     wn = False
1366
1367     wd = 0.005
1368
1369     beta1 = 0.5
1370
1371     beta2 = 0.5
1372
1373
1374     model = UNetEx(2, 3, filters=filters,
1375     ↵ kernel_size=kernel_size, batch_norm=bn,
1376     ↵ weight_norm=wn)
1377
1378     # Define opotimizer
1379     optimizer =
1380     ↵ torch.optim.AdamW(model.parameters(), lr=lr,
1381     ↵ weight_decay=wd)
1382
1383
1384     config = {}
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
```

```

1361     train_loss_curve = []
1362     test_loss_curve = []
1363     train_mse_curve = []
1364     test_mse_curve = []
1365     train_ux_curve = []
1366     test_ux_curve = []
1367     train_uy_curve = []
1368     test_uy_curve = []
1369     train_p_curve = []
1370     test_p_curve = []

1371
1372     def after_epoch(scope):
1373         train_loss_curve.append(
1374             scope["train_loss"])
1375         test_loss_curve.append( scope["val_loss"])
1376         train_mse_curve.append(
1377             scope["train_metrics"]["mse"])
1378         test_mse_curve.append(
1379             scope["val_metrics"]["mse"])
1380         train_ux_curve.append(
1381             scope["train_metrics"]["ux"])
1382         test_ux_curve.append(
1383             scope["val_metrics"]["ux"])
1384         train_uy_curve.append(
1385             scope["train_metrics"]["uy"])
1386         test_uy_curve.append(
1387             scope["val_metrics"]["uy"])
1388         train_p_curve.append(
1389             scope["train_metrics"]["p"])
1390         test_p_curve.append(
1391             scope["val_metrics"]["p"])

1392
1393     def loss_func(model, batch):
1394         x, y = batch
1395         output = model(x)
1396         lossu = ((output[:,0,:,:] - y[:,0,:,:]) ** 2).reshape((output.shape[0],1,
1397             output.shape[2],output.shape[3]))
1398         lossv = ((output[:,1,:,:] - y[:,1,:,:]) ** 2).reshape((output.shape[0],
1399             1,output.shape[2],output.shape[3]))

```

```

1389         losssp = ((output[:,2,:,:] - y[:,2,:,:]) **  

1390             2).reshape((output.shape[0],1,  

1391             output.shape[2],output.shape[3]))  

1392 # lossu = torch.abs((output[:,0,:,:] -  

1393             y[:,0,:,:])).reshape((output.shape[0],1,  

1394             output.shape[2],output.shape[3]))  

1395 # lossv = torch.abs((output[:,1,:,:] -  

1396             y[:,1,:,:])).reshape((output.shape[0],1,  

1397             output.shape[2],output.shape[3]))  

1398 # lossp = torch.abs((output[:,2,:,:] -  

1399             y[:,2,:,:])).reshape((output.shape[0],1,  

1400             output.shape[2],output.shape[3]))  

1401 loss = (lossu + lossv +  

1402             lossp)/channels_weights  

1403 return torch.sum(loss), output  

1404  

1405 ti = time.time()  

1406  

1407 # Training model  

1408 DeepCFD, train_metrics, train_loss,  

1409     test_metrics, test_loss = train_model(model,  

1410     loss_func, train_dataset, test_dataset,  

1411     optimizer,  

1412     epochs=epoch[ep], batch_size=batchsize[bs],  

1413         device=device,  

1414     m_mse_name="Total MSE",  

1415     m_mse_on_batch=lambda scope:  

1416         float(torch.sum((scope["output"] -  

1417             scope["batch"][1]) ** 2)),  

1418     m_mse_on_epoch=lambda scope:  

1419         sum(scope["list"]) /  

1420         len(scope["dataset"]),  

1421     m_ux_name="Ux MSE",  

1422     m_ux_on_batch=lambda scope:  

1423         float(torch.sum((scope["output"][:,0,:,:] -  

1424             scope["batch"][1][:,0,:,:]) ** 2)),  

1425     m_ux_on_epoch=lambda scope:  

1426         sum(scope["list"]) /  

1427         len(scope["dataset"]),  

1428     m_uy_name="Uy MSE",

```

```

1408     m_uy_on_batch=lambda scope:
1409         ↳ float(torch.sum((scope["output"][:,1,:,:]-
1410             ↳ - scope["batch"][1][:,1,:,:]) ** 2)),
1411     m_uy_on_epoch=lambda scope:
1412         ↳ sum(scope["list"]) /
1413         ↳ len(scope["dataset"]),
1414     m_p_name="p MSE",
1415     m_p_on_batch=lambda scope:
1416         ↳ float(torch.sum((scope["output"][:,2,:,:]-
1417             ↳ - scope["batch"][1][:,2,:,:]) ** 2)),
1418     m_p_on_epoch=lambda scope:
1419         ↳ sum(scope["list"]) /
1420         ↳ len(scope["dataset"]), patience=25,
1421         ↳ after_epoch=after_epoch
1422     )
1423
1424
1425     duration = time.time() - ti
1426     print("Training time: " + str(duration//60) +
1427         ↳ "min and " +
1428         ↳ str("{:.2f}").format(duration%60)) + "sec")
1429
1430
1431     # Guardar los el modelo
1432     torch.save(model.state_dict(), modelo_directory
1433         ↳ + "Modelo_geometries_" + str(cont) +
1434         ↳ "_data_augmentation.py")
1435
1436
1437     metrics = {}
1438     metrics["train_metrics"] = train_metrics
1439     metrics["train_loss"] = train_loss
1440     metrics["test_metrics"] = test_metrics
1441     metrics["test_loss"] = test_loss
1442     curves = {}
1443     curves["train_loss_curve"] = train_loss_curve
1444     curves["test_loss_curve"] = test_loss_curve
1445     curves["train_mse_curve"] = train_mse_curve
1446     curves["test_mse_curve"] = test_mse_curve
1447     curves["train_ux_curve"] = train_ux_curve
1448     curves["test_ux_curve"] = test_ux_curve
1449     curves["train_uy_curve"] = train_uy_curve
1450     curves["test_uy_curve"] = test_uy_curve
1451     curves["train_p_curve"] = train_p_curve
1452     curves["test_p_curve"] = test_p_curve

```

```

1437     config["metrics"] = metrics
1438     config["curves"] = curves
1439
1440     net_param = {}
1441     net_param["num_epochs"] = epoch[ep]
1442     net_param["learning_ratio"] =
1443         ↳ learning_rate[l_r]
1444     net_param["kernel_size"] = kernelsize[ks]
1445     net_param["batch_size"] = batchsize[bs]
1446     net_param["arquitecture"] = filters_[filt]
1447     config["net_param"] = net_param
1448     config["training_duration"] = duration
1449
1450     with open(simulation_directory +
1451             ↳ "results_geometries_" + str(cont) +
1452             ↳ "_data_augmentation.json", "w") as file:
1453         json.dump([config, duration], file)
1454
1455     test_range = list(range(10, 20))
1456     out = DeepCFD(test_x[test_range].to(device))
1457     error = (torch.abs((out.cpu() -
1458             ↳ test_y[test_range].cpu())))/torch.abs(test_y.cpu())
1459     for s in range(10):
1460
1461         ↳ visualize2(test_y[test_range].cpu().detach().numpy(),
1462             ↳ out.cpu().detach().numpy(),
1463                 ↳ error.cpu().detach().numpy(), s,
1464                     ↳ epoch[ep],
1465                     ↳ learning_rate[l_r],
1466                     ↳ batchsize[bs], kernelsize[ks])
1467     out_b = out.cpu().detach().numpy()
1468     error_b = error.cpu().detach().numpy()
1469     test_y_b =
1470         ↳ test_y[test_range].cpu().detach().numpy()
1471     test_x_b =
1472         ↳ test_x[test_range].cpu().detach().numpy()
1473
1474     with open(plots_subdirectory +
1475             ↳ "plots_geometries" + str(cont) +
1476             ↳ "data_augmentation.pkl", "wb") as file:
1477         pickle.dump([out_b, error_b, test_y_b,
1478             ↳ test_x_b, test_y, test_x], file)

```

```
1466  
1467     cont = cont + 1
```

A.6 Test de los modelos neuronales y análisis de los resultados

Test de los modelos neuronales y análisis de los resultados de la CNN que predice los instantes futuros para velocidades variables de entrada:

```
1468 import pickle  
1469 import numpy as np  
1470 import math  
1471 import random  
1472 from matplotlib import pyplot as plt  
1473 import matplotlib.gridspec as gridspec  
1474 import statistics as st  
1475 from sklearn import preprocessing  
1476 from os import listdir  
1477 from tabulate import tabulate  
1478 import json  
1479 import pandas as pd  
1480  
1481 def visualize(sample_y, out_y, error, s, n_sample, geometry):  
1482  
1483     minu = np.min(sample_y[s, 0, :, :])  
1484     maxu = np.max(sample_y[s, 0, :, :])  
1485  
1486     minv = np.min(sample_y[s, 1, :, :])  
1487     maxv = np.max(sample_y[s, 1, :, :])  
1488  
1489     minp = np.min(sample_y[s, 2, :, :])  
1490     maxp = np.max(sample_y[s, 2, :, :])  
1491  
1492     mineu = np.min(error[s, 0, :, :])  
1493     maxeu = np.max(error[s, 0, :, :])  
1494  
1495     minev = np.min(error[s, 1, :, :])  
1496     maxev = np.max(error[s, 1, :, :])  
1497  
1498     minep = np.min(error[s, 2, :, :])  
1499     maxep = np.max(error[s, 2, :, :])
```

```

1500
1501     plt.figure()
1502     fig = plt.gcf()
1503     plt.suptitle(geometry + ". Sample nº " + str(n_samples+s),
1504                   fontsize=20)
1505     fig.set_size_inches(15, 10)
1506     plt.subplot(3, 3, 1)
1507     plt.title('CFD', fontsize=18)
1508     plt.imshow(np.transpose(sample_y[s, 0, :, :]), cmap='jet', vmin=
1509                minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
1510     plt.colorbar(orientation='horizontal')
1511     plt.ylabel('Ux', fontsize=18)
1512     plt.subplot(3, 3, 2)
1513     plt.title('CNN', fontsize=18)
1514     plt.imshow(np.transpose(out_y[s, 0, :, :]), cmap='jet', vmin =
1515                minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
1516     plt.colorbar(orientation='horizontal')
1517     plt.subplot(3, 3, 3)
1518     plt.title('Error', fontsize=18)
1519     plt.imshow(np.transpose(error[s, 0, :, :]), cmap='jet', vmin =
1520                mineu, vmax = maxeu, origin='lower', extent=[0,172,0,79])
1521     plt.colorbar(orientation='horizontal')

1522     plt.subplot(3, 3, 4)
1523     plt.imshow(np.transpose(sample_y[s, 1, :, :]), cmap='jet', vmin=
1524                minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
1525     plt.colorbar(orientation='horizontal')
1526     plt.ylabel('Uy', fontsize=18)
1527     plt.subplot(3, 3, 5)
1528     plt.imshow(np.transpose(out_y[s, 1, :, :]), cmap='jet', vmin =
1529                minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
1530     plt.colorbar(orientation='horizontal')
1531     plt.subplot(3, 3, 6)
1532     plt.imshow(np.transpose(error[s, 1, :, :]), cmap='jet', vmin =
1533                minev, vmax = maxev, origin='lower', extent=[0,172,0,79])
1534     plt.colorbar(orientation='horizontal')

1535
1536     plt.subplot(3, 3, 7)
1537     plt.imshow(np.transpose(sample_y[s, 2, :, :]), cmap='jet', vmin=
1538                minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
1539     plt.colorbar(orientation='horizontal')

```

```

1534     plt.ylabel('p', fontsize=18)
1535     plt.subplot(3, 3, 8)
1536     plt.imshow(np.transpose(out_y[s, 2, :, :]), cmap='jet', vmin =
1537     ↪ minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
1538     plt.colorbar(orientation='horizontal')
1539     plt.subplot(3, 3, 9)
1540     plt.imshow(np.transpose(error[s, 2, :, :]), cmap='jet', vmin =
1541     ↪ minep, vmax = maxep, origin='lower', extent=[0,172,0,79])
1542     plt.colorbar(orientation='horizontal')
1543     plt.tight_layout()
1544     plt.show()

1545
1546
1547 def visualize3(sample_y, s, n_sample, geometry):
1548
1549     minu = np.min(sample_y[s, 0, :, :])
1550     maxu = np.max(sample_y[s, 0, :, :])
1551
1552     minv = np.min(sample_y[s, 1, :, :])
1553     maxv = np.max(sample_y[s, 1, :, :])
1554
1555     minp = np.min(sample_y[s, 2, :, :])
1556     maxp = np.max(sample_y[s, 2, :, :])
1557
1558     plt.figure()
1559     fig = plt.gcf()
1560     plt.suptitle(geometry + ". Sample n.º " + str(n_sample+s),
1561     ↪ fontsize=20)
1562     fig.set_size_inches(15, 10)
1563     plt.suptitle("Time instant " + str(s+1), fontsize=20)
1564     plt.subplot(3, 1, 1)
1565     plt.title('CFD', fontsize=18)
1566     plt.imshow((sample_y[s, 0, :, :]), cmap='jet', vmin = minu,
1567     ↪ vmax = maxu, origin='lower', extent=[0,172,0,79])
1568     plt.colorbar(orientation='horizontal')
1569     plt.ylabel('Ux', fontsize=18)
1570
1571
1572     plt.subplot(3, 1, 2)
1573     plt.imshow((sample_y[s, 1, :, :]), cmap='jet', vmin = minv,
1574     ↪ vmax = maxv, origin='lower', extent=[0,172,0,79])
1575     plt.colorbar(orientation='horizontal')
1576     plt.ylabel('Uy', fontsize=18)

```

```

1571
1572
1573     plt.subplot(3, 1, 3)
1574     plt.imshow((sample_y[s, 2, :, :]), cmap='jet', vmin = minp,
1575     ↪   vmax = maxp, origin='lower', extent=[0,172,0,79])
1576     plt.colorbar(orientation='horizontal')
1577     plt.ylabel('p', fontsize=18)
1578     plt.tight_layout()
1579     plt.show()

1580 def visualize_rel_error(sample_y, out_y, error, s, input_velocity
1581   ):
1582
1583     # Visualize function with relative error added
1584
1585     minu = np.min(sample_y[s, 0, :, :])
1586     maxu = np.max(sample_y[s, 0, :, :])
1587
1588     minv = np.min(sample_y[s, 1, :, :])
1589     maxv = np.max(sample_y[s, 1, :, :])
1590
1591     minp = np.min(sample_y[s, 2, :, :])
1592     maxp = np.max(sample_y[s, 2, :, :])
1593
1594     mineu = np.min(error[s, 0, :, :])
1595     maxeu = np.max(error[s, 0, :, :])
1596
1597     minev = np.min(error[s, 1, :, :])
1598     maxev = np.max(error[s, 1, :, :])
1599
1600     minep = np.min(error[s, 2, :, :])
1601     maxep = np.max(error[s, 2, :, :])
1602
1603     plt.figure()
1604     fig = plt.gcf()
1605     fig.set_size_inches(15, 10)
1606     plt.suptitle("Time instant " + str(s+1), fontsize=20)
1607     plt.subplot(3, 4, 1)
1608     plt.title('CFD', fontsize=18)
1609     plt.imshow(np.transpose(sample_y[s, 0, :, :]), cmap='jet', vmin
1610     ↪   = minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
1611     plt.colorbar(orientation='horizontal')

```

```

1610     plt.ylabel('Ux', fontsize=18)
1611     plt.subplot(3, 4, 2)
1612     plt.title('CNN', fontsize=18)
1613     plt.imshow(np.transpose(out_y[s, 0, :, :]), cmap='jet', vmin =
1614         minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
1615     plt.colorbar(orientation='horizontal')
1616     plt.subplot(3, 4, 3)
1617     plt.title('Error', fontsize=18)
1618     plt.imshow(np.transpose(error[s, 0, :, :]), cmap='jet', vmin =
1619         mineu, vmax = maxeu, origin='lower', extent=[0,172,0,79])
1620     plt.colorbar(orientation='horizontal')
1621     plt.subplot(3, 4, 4)
1622
1623     plt.subplot(3, 4, 5)
1624     plt.imshow(np.transpose(sample_y[s, 1, :, :]), cmap='jet', vmin =
1625         minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
1626     plt.colorbar(orientation='horizontal')
1627     plt.ylabel('Uy', fontsize=18)
1628     plt.subplot(3, 4, 6)
1629     plt.imshow(np.transpose(out_y[s, 1, :, :]), cmap='jet', vmin =
1630         minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
1631     plt.colorbar(orientation='horizontal')
1632     plt.subplot(3, 4, 7)
1633     plt.imshow(np.transpose(error[s, 1, :, :]), cmap='jet', vmin =
1634         minev, vmax = maxev, origin='lower', extent=[0,172,0,79])
1635     plt.colorbar(orientation='horizontal')
1636     plt.subplot(3, 4, 8)
1637
1638     plt.subplot(3, 4, 9)
1639     plt.imshow(np.transpose(sample_y[s, 2, :, :]), cmap='jet', vmin =
1640         minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
1641     plt.colorbar(orientation='horizontal')
1642     plt.ylabel('p', fontsize=18)
1643     plt.subplot(3, 4, 10)
1644     plt.imshow(np.transpose(out_y[s, 2, :, :]), cmap='jet', vmin =
1645         minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
1646     plt.colorbar(orientation='horizontal')
1647     plt.subplot(3, 4, 11)
1648     plt.imshow(np.transpose(error[s, 2, :, :]), cmap='jet', vmin =
1649         minep, vmax = maxep, origin='lower', extent=[0,172,0,79])

```

```

1644     plt.colorbar(orientation='horizontal')
1645     plt.subplot(3, 4, 12)
1646     plt.tight_layout()
1647     plt.show()
1648
1649 def test_mean_error(plots):
1650     error = np.zeros([len(plots), 50]) # Habría que poner esto pero
1651     ↵ me da fallo --> len(plots[id_test][2]))
1652     max_error = np.zeros([len(plots), 50])
1653     variance = np.zeros([len(plots), 50])
1654     col = 0
1655     for key in plots:
1656         for row in range(50):
1657             error[col, row] = np.mean(plots[key][2][row, 0],
1658             ↵ dtype=np.float64)
1659             max_error[col, row] = np.amax(plots[key][2][row, 0])
1660             variance[col, row] = np.var(plots[key][2][row, 0], ddof
1661             ↵ = 1)
1662             col = col + 1
1663     mean_error = np.mean(error, axis = 0, dtype=np.float64)
1664     max_error_all = np.amax(max_error, axis = 0)
1665     variance_all = np.mean(variance, axis = 0)
1666     me_error = np.mean(mean_error)
1667     v_all = np.mean(variance_all)
1668     ma_error = np.amax(max_error_all)
1669     return mean_error, max_error_all, variance_all, me_error, v_all,
1670     ↵ ma_error
1671
1672 def plot_feature(feature_type, upper_limit, legend, plot_title,
1673     ↵ y_label):
1674     for i in range(10):
1675         plt.plot(feature_type[i], '--o', markersize = 3)
1676
1677         plt.grid(axis='both')
1678         plt.ylabel(y_label)
1679         plt.ylim((0,upper_limit))
1680         plt.legend(legend)
1681         plt.title(plot_title)
1682         plt.show()
1683
1684 if __name__ == "__main__":
1685
1686
1687
1688
1689
1690
```

```

1681 plots = []
1682 training_info = []
1683 path = "E:/NuevasGeometrias/Resultados_NuevasGeometrias/"
1684 # path = "C:/Users/alvar/Desktop/Master/ Segundo/TFM/2ª
    ↵ parte/Datos para entrenamiento/Resultados_vx/"
1685
1686 plots_directories = "Plots vx/#, "Plots vy/", "Plots p/"]
1687 statistics_directories = "Estadisticas vx#", "Estadisticas vy",
    ↵ "Estadisticas p]"
1688 # filepaths = [f for f in listdir(str(path +
    ↵ plots_directories)) if f.endswith('.pkl')]
1689 # for file in filepaths:
1690 #     data = pickle.load(open(path + plots_directories + file,
    ↵ "rb"))
1691 #     mean_error, max_error, variance, me_error, var_, ma_error
    ↵ = test_mean_error(data)
1692 #     with open(path + statistics_directories +
    ↵ "/plots_geometries_statistics_" + file[-6:-4] + ".json",
    ↵ "wb") as save_file:
1693 #         pickle.dump([mean_error, max_error, variance,
    ↵ me_error, var_, ma_error], save_file)
1694
1695 filepaths_vx = [f for f in listdir(str(path +
    ↵ plots_directories)) if f.endswith('.pkl')]
1696 data_vx = pickle.load(open(path + plots_directories +
    ↵ filepaths_vx[10], "rb"))
1697 filepaths_vy = [f for f in listdir(str(path + "Plots vy/")) if
    ↵ f.endswith('.pkl')]
1698 data_vy = pickle.load(open(path + "Plots vy/" + filepaths_vy[7],
    ↵ "rb"))
1699 filepaths_p = [f for f in listdir(str(path + "Plots p/")) if
    ↵ f.endswith('.pkl')]
1700 data_p = pickle.load(open(path + "Plots p/" + filepaths_p[5],
    ↵ "rb"))
1701
1702 cfd = np.zeros([50, 3, 79, 172])
1703 cnn = np.zeros([50, 3, 79, 172])
1704 error = np.zeros([50, 3, 79, 172])
1705
1706 geometry = 'circle'
1707 samples = ['244', '349', '369']
1708 # samples = ['289', '374', '279']

```

```

1709     # samples = ['115', '369', '51']
1710     # samples = ['74', '190', '317']
1711     # samples = ['169', '62', '428']
1712     # samples = ['313', '233', '19']
1713
1714     cfd[:,0,:,:] = data_vx['vx_' + geometry + '_' +
1715         ↳ samples[0]] [0] [:,0,:,:]
1716     cfd[:,1,:,:] = data_vy['vy_' + geometry + '_' +
1717         ↳ samples[1]] [0] [:,0,:,:]
1718     cfd[:,2,:,:] = data_p['p_' + geometry + '_' +
1719         ↳ samples[2]] [0] [:,0,:,:]
1720
1721
1722     cnn[:,0,:,:] = data_vx['vx_' + geometry + '_' +
1723         ↳ samples[0]] [1] [:,0,:,:]
1724     cnn[:,1,:,:] = data_vy['vy_' + geometry + '_' +
1725         ↳ samples[1]] [1] [:,0,:,:]
1726     cnn[:,2,:,:] = data_p['p_' + geometry + '_' +
1727         ↳ samples[2]] [1] [:,0,:,:]
1728
1729     error[:,0,:,:] = data_vx['vx_' + geometry + '_' +
1730         ↳ samples[0]] [2] [:,0,:,:]
1731     error[:,1,:,:] = data_vy['vy_' + geometry + '_' +
1732         ↳ samples[1]] [2] [:,0,:,:]
1733     error[:,2,:,:] = data_p['p_' + geometry + '_' +
1734         ↳ samples[2]] [2] [:,0,:,:]
1735
1736     s = [0, 4, 9, 14, 19, 29, 39, 49]
1737
1738     for a in range(len(s)):
1739         visualize(np.moveaxis(cfd, [2,3], [3,2]),
1740                    np.moveaxis(cnn, [2,3], [3,2]),
1741                    np.moveaxis(error, [2,3], [3,2]),
1742                    s[a], 1, "Circle")
1743
1744     cfd_vx = np.zeros([len(data_vx)*50, 1, 79, 172])
1745     out_vx = np.zeros([len(data_vx)*50, 1, 79, 172])
1746
1747     cfd_vy = np.zeros([len(data_vy)*50, 1, 79, 172])
1748     out_vy = np.zeros([len(data_vy)*50, 1, 79, 172])
1749
1750     cfd_p = np.zeros([len(data_p)*50, 1, 79, 172])
1751     out_p = np.zeros([len(data_p)*50, 1, 79, 172])

```

```

1742
1743     c = 0
1744     for key in data_vx:
1745         cfd_vx[c:c+50,:,:,:]=data_vx[key][0]
1746         out_vx[c:c+50,:,:,:]=data_vx[key][1]
1747         c = c + 50
1748
1749     c = 0
1750     for key in data_vy:
1751         cfd_vy[c:c+50,:,:,:]=data_vy[key][0]
1752         out_vy[c:c+50,:,:,:]=data_vy[key][1]
1753         c = c + 50
1754
1755     c = 0
1756     for key in data_p:
1757         cfd_p[c:c+50,:,:,:]=data_p[key][0]
1758         out_p[c:c+50,:,:,:]=data_p[key][1]
1759         c = c + 50
1760
1761     cfd_vx_mean = np.mean(cfd_vx)
1762     cfd_vy_mean = np.mean(cfd_vy)
1763     cfd_p_mean = np.mean(cfd_p)
1764     cfd_vx_std = np.std(cfd_vx)
1765     cfd_vy_std = np.std(cfd_vy)
1766     cfd_p_std = np.std(cfd_p)
1767
1768     out_vx_mean = np.mean(out_vx)
1769     out_vy_mean = np.mean(out_vy)
1770     out_p_mean = np.mean(out_p)
1771     out_vx_std = np.std(out_vx)
1772     out_vy_std = np.std(out_vy)
1773     out_p_std = np.std(out_p)

1774 import os
1775 import json
1776 import torch
1777 import pickle
1778 import time
1779 from train_functions import *
1780 import torch.optim as optim
1781 from torch.utils.data import TensorDataset
1782 from Models.UNetEx import UNetEx

```

```

1783 from Models.UNet import UNet
1784 import numpy as np
1785 from matplotlib import pyplot as plt
1786 from os import listdir
1787 os.environ['KMP_DUPLICATE_LIB_OK']='True'
1788
1789 def visualize(sample_y, out_y, error, s, variable, n_sample,
1790   ↵ velocity):
1791
1792     minu = np.min(sample_y[s, 0, :, :])
1793     maxu = np.max(sample_y[s, 0, :, :])
1794
1795     # minv = np.min(sample_y[s, 1, :, :])
1796     # maxv = np.max(sample_y[s, 1, :, :])
1797
1798     # minp = np.min(sample_y[s, 2, :, :])
1799     # maxp = np.max(sample_y[s, 2, :, :])
1800
1801     mineu = np.min(error[s, 0, :, :])
1802     maxeu = np.max(error[s, 0, :, :])
1803
1804     # minev = np.min(error[s, 1, :, :])
1805     # maxev = np.max(error[s, 1, :, :])
1806
1807     # minep = np.min(error[s, 2, :, :])
1808     # maxep = np.max(error[s, 2, :, :])
1809
1810     plt.figure()
1811     fig = plt.gcf()
1812     plt.suptitle("Input velocity: " + velocity + " ms. Sample nº: "
1813       ↵ + str(n_sample+s), fontsize=20)
1814     fig.set_size_inches(15, 10)
1815     plt.subplot(3, 3, 1)
1816     plt.title('CFD', fontsize=18)
1817     plt.imshow(np.transpose(sample_y[s, 0, :, :]), cmap='jet', vmin
1818       ↵ = minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
1819     plt.colorbar(orientation='horizontal')
1820     plt.ylabel(variable, fontsize=18)
1821     plt.subplot(3, 3, 2)
1822     plt.title('CNN', fontsize=18)
1823     plt.imshow(np.transpose(out_y[s, 0, :, :]), cmap='jet', vmin =
1824       ↵ minu, vmax = maxu, origin='lower', extent=[0,172,0,79])

```

```

1821 plt.colorbar(orientation='horizontal')
1822 plt.subplot(3, 3, 3)
1823 plt.title('Error', fontsize=18)
1824 plt.imshow(np.transpose(error[s, 0, :, :]), cmap='jet', vmin =
    ↪ mineu, vmax = maxeu, origin='lower', extent=[0,172,0,79])
1825 plt.colorbar(orientation='horizontal')

1826
1827 # plt.subplot(3, 3, 4)
1828 # plt.imshow(np.transpose(sample_y[s, 1, :, :]), cmap='jet',
    ↪ vmin = minv, vmax = maxv, origin='lower',
    ↪ extent=[0,172,0,79])
1829 # plt.colorbar(orientation='horizontal')
1830 # plt.ylabel('Uy', fontsize=18)
1831 # plt.subplot(3, 3, 5)
1832 # plt.imshow(np.transpose(out_y[s, 1, :, :]), cmap='jet', vmin
    ↪ = minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
1833 # plt.colorbar(orientation='horizontal')
1834 # plt.subplot(3, 3, 6)
1835 # plt.imshow(np.transpose(error[s, 1, :, :]), cmap='jet', vmin
    ↪ = minev, vmax = maxev, origin='lower', extent=[0,172,0,79])
1836 # plt.colorbar(orientation='horizontal')

1837
1838
1839 # plt.subplot(3, 3, 7)
1840 # plt.imshow(np.transpose(sample_y[s, 2, :, :]), cmap='jet',
    ↪ vmin = minp, vmax = maxp, origin='lower',
    ↪ extent=[0,172,0,79])
1841 # plt.colorbar(orientation='horizontal')
1842 # plt.ylabel('p', fontsize=18)
1843 # plt.subplot(3, 3, 8)
1844 # plt.imshow(np.transpose(out_y[s, 2, :, :]), cmap='jet', vmin
    ↪ = minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
1845 # plt.colorbar(orientation='horizontal')
1846 # plt.subplot(3, 3, 9)
1847 # plt.imshow(np.transpose(error[s, 2, :, :]), cmap='jet', vmin
    ↪ = minep, vmax = maxep, origin='lower', extent=[0,172,0,79])
1848 # plt.colorbar(orientation='horizontal')
1849 plt.tight_layout()
1850 plt.show()

1851
1852 def visualize2(sample_y, s, file):
1853

```

```

1854     plt.figure()
1855     fig = plt.gcf()
1856     # fig.set_size_inches(15, 10)
1857
1858     minu = np.min(sample_y[s, 0, :, :])
1859     maxu = np.max(sample_y[s, 0, :, :])
1860
1861     plt.imshow(np.transpose(sample_y[s, 0, :, :]), cmap='jet', vmin
1862                ↳ = minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
1863     plt.colorbar(orientation='horizontal')
1864     plt.title("Test " + file)
1865     plt.tight_layout()
1866     plt.show()
1867
1868
1869 def test_CNN(n_instantes, x_ms, n_test):
1870
1871     ti = time.time()
1872
1873     out = torch.empty((n_instantes,3,172,79))
1874     x_ms = torch.FloatTensor(x_ms)
1875     out[:,0:2,:,:]=x_ms[0,:2,:,:]
1876
1877     out[0,2,:,:] = model_prueba(x_ms[n_test:n_test+1])
1878
1879     for i in range (1,n_instantes,1):
1880         print("Sample: " + str(i))
1881         out[i,2,:,:] = model_prueba(out[i-1:i])
1882         out2=out[:,2:3,:,:]
1883         out2=out2.reshape([n_instantes,1,172,79])
1884         tout = time.time()
1885
1886         duration = tout-ti
1887
1888
1889     if __name__ == "__main__":
1890
1891         path = "E:/CNN Velocidades/Modelos/"
1892
1893
1894     torch.manual_seed(0)

```

```

1895     lr = 0.0001
1896     # lr = 0.0001
1897     # kernel_size = 3
1898     # kernel_size = 5
1899     kernel_size = 7
1900     filters = [8, 16, 32, 32]
1901     bn = False
1902     wn = False
1903     wd = 0.005
1904     beta1 = 0.5
1905     beta2 = 0.5
1906
1907     model_prueba = UNetEx(3, 1, filters=filters,
1908                           ↪ kernel_size=kernel_size, batch_norm=bn, weight_norm=wn)
1909     # model_prueba.load_state_dict(torch.load(path +
1910     ↪ "Modelo_vy13.py", map_location=torch.device('cpu')))
1911     model_prueba.load_state_dict(torch.load(path + "Modelo_vy27.py",
1912     ↪ map_location=torch.device('cpu')))
1913     # model_prueba.load_state_dict(torch.load(path +
1914     ↪ "Modelo_p26.py", map_location=torch.device('cpu')))
1915     model_prueba.eval()
1916
1917
1918
1919     path = "E:/NuevasGeometrias/Resultados_NuevasGeometrias"
1920     ↪ /Resultados_first_sample/tests_first_sample/"
1921     geom_index = [1, 63, 121, 723, 1305, 1923]
1922     geometry = ["circle", "square", "triangle_eq", "rectangle",
1923     ↪ "ellipse", "triangle"]
1924     cfd_all = np.zeros([len(geometry)*50, 1, 172, 79])
1925     out_all = np.zeros([len(geometry)*50, 1, 172, 79])
1926     for g_i in range(len(geometry)):
1927         # cfd = pickle.load(open(path + "Test geometrias no
1928         ↪ vistas/dataY_1300_1001_1050.pkl", "rb"))
1929         cfd = pickle.load(open("E:/NuevasGeometrias/Y_geom" +
1930         ↪ str(geom_index[g_i]) + "_interpolated.pkl", "rb"))
1931         # filepaths = [f for f in listdir(str(path)) if
1932         ↪ f.endswith('.pkl')]
1933         # for file in filepaths:
1934         #     with open(path + file, 'rb') as f:
1935         #         input_data.append(pickle.load(f))
1936         input_data = []
1937         input_data.append(pickle.load(open(path +

```

```

1928             "input_data_geometry_" +
1929                 ↵   geometry[g_i]
1930             +
1931                 ↵   "_size_factor_1_rotation_angle_1_variable_vx."
1932                 ↵   "rb"))
1933
1934     x = np.zeros([1,3,172,79])
1935     x[:, -1, :, :] = input_data[0][0] [:, 1, :, :].detach().numpy()
1936     x[:, 0, :, :] = input_data[0][1]
1937     x[:, 1, :, :] = input_data[0][2]
1938
1939             # with open(simulation_directory + "plots_" + velocity
1940             ↵   + "ms_" + variable + "_Final_v2" + str(n_test) +
1941             ↵   ".pkl", "wb") as file:
1942                 # pickle.dump([out, error, y], file)
1943
1944     variable = 'vy'
1945     n_instantes = 50
1946     out2, duration = test_CNN(n_instantes, x, 0)
1947     # cfd = cfd[:, [1,2,0], :, :]
1948     cfd = torch.tensor(np.moveaxis(cfd, [2,3], [3,2]))
1949     cfd = cfd[:, 2:3, :, :]
1950     error = abs(cfd[:n_instantes, :, :, :] - out2)
1951     s = [0, 4, 9, 14, 19, 29, 39, 49]
1952     for a in s:
1953         # visualize2(out2.detach().numpy(), s, filepaths[0])
1954         visualize(cfd.detach().numpy(), out2.detach().numpy(),
1955                     error.detach().numpy(), a, variable, 1, "5")
1956
1957     path2 = "E:/NuevasGeometrias"
1958         ↵   /Resultados_NuevasGeometrias/test_nuevas_geometrias/"
1959     with open(path2 + "test_" + geometry[g_i] + '_' + variable
1960             ↵   + ".pkl", "wb") as file:
1961         pickle.dump([cfд.detach().numpy()[:n_instantes],
1962                     ↵   out2.detach().numpy(),
1963                     error.detach().numpy()], file)
1964
1965     cfd_all[g_i*50:(g_i+1)*50] = cfd[:50].detach().numpy()
1966     out_all[g_i*50:(g_i+1)*50] = out2.detach().numpy()
1967
1968     mean_value_cfd = np.mean(cfd_all[:, 0, :, :])
1969     std_cfd = np.std(cfd_all[:, 0, :, :])

```

```

1962     mean_value_cnn = np.mean(out_all[:,0,:,:])
1963     std_cnn = np.std(out_all[:,0,:,:])
1964
1965     sample10 = [x+10 for x in sample1]
1966     sample15 = [x+15 for x in sample1]
1967     sample20 = [x+20 for x in sample1]
1968     sample30 = [x+30 for x in sample1]
1969     sample40 = [x+40 for x in sample1]
1970     sample50 = [x+50 for x in sample1]
1971     sample = [sample1, sample5, sample10, sample15, sample20,
1972             ↵ sample30,
1973             sample40, sample50]
1974     samples = [1, 5, 10, 15, 20, 30, 40, 50]
1975
1976     for s in range(len(sample)):
1977         cfd = cfd_all[sample[s]]
1978         out = out_all[sample[s]]
1979         gen_data_distribution(cfd, out, samples[s])

```

Test de los modelos neuronales y análisis de los resultados de la CNN que predice los instantes futuros para geometrías variables:

```

1979 import pickle
1980 import numpy as np
1981 import math
1982 import random
1983 from matplotlib import pyplot as plt
1984 import matplotlib.gridspec as gridspec
1985 import statistics as st
1986 from sklearn import preprocessing
1987 from os import listdir
1988 from tabulate import tabulate
1989 import json
1990 import pandas as pd
1991
1992 def visualize(sample_y, out_y, error, s, variable, n_sample,
1993               ↵ velocity):
1994
1995     minu = np.min(sample_y[s, :, :])
1996     maxu = np.max(sample_y[s, :, :])
1997
1998     # minv = np.min(sample_y[s, 1, :, :])

```

```

1998 # maxv = np.max(sample_y[s, 1, :, :])
1999
2000 # minp = np.min(sample_y[s, 2, :, :])
2001 # maxp = np.max(sample_y[s, 2, :, :])
2002
2003 mineu = np.min(error[s, :, :])
2004 maxeu = np.max(error[s, :, :])
2005
2006 # minev = np.min(error[s, 1, :, :])
2007 # maxev = np.max(error[s, 1, :, :])
2008
2009 # minep = np.min(error[s, 2, :, :])
2010 # maxep = np.max(error[s, 2, :, :])
2011
2012 plt.figure()
2013 fig = plt.gcf()
2014 plt.suptitle("Input velocity: " + velocity + " ms. Sample nº: "
2015     + str(n_sample+s), fontsize=20)
2016 fig.set_size_inches(15, 10)
2017 plt.subplot(3, 3, 1)
2018 plt.title('CFD', fontsize=18)
2019 plt.imshow(np.transpose(sample_y[s, :, :]), cmap='jet', vmin =
2020     minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2021 plt.colorbar(orientation='horizontal')
2022 plt.ylabel(variable, fontsize=18)
2023 plt.subplot(3, 3, 2)
2024 plt.title('CNN', fontsize=18)
2025 plt.imshow(np.transpose(out_y[s, :, :]), cmap='jet', vmin =
2026     minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2027 plt.colorbar(orientation='horizontal')
2028 plt.subplot(3, 3, 3)
2029 plt.title('Error', fontsize=18)
2030 plt.imshow(np.transpose(error[s, :, :]), cmap='jet', vmin =
2031     mineu, vmax = maxeu, origin='lower', extent=[0,172,0,79])
2032 plt.colorbar(orientation='horizontal')
2033 # plt.subplot(3, 3, 4)
# plt.imshow(np.transpose(sample_y[s, 1, :, :]), cmap='jet',
#     vmin = minv, vmax = maxv, origin='lower',
#     extent=[0,172,0,79])
# plt.colorbar(orientation='horizontal')
# plt.ylabel('Uy', fontsize=18)

```

```

2034 # plt.subplot(3, 3, 5)
2035 # plt.imshow(np.transpose(out_y[s, 1, :, :]), cmap='jet', vmin
2036   ← = minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
2037 # plt.colorbar(orientation='horizontal')
2038 # plt.subplot(3, 3, 6)
2039 # plt.imshow(np.transpose(error[s, 1, :, :]), cmap='jet', vmin
2040   ← = minev, vmax = maxev, origin='lower', extent=[0,172,0,79])
2041 # plt.colorbar(orientation='horizontal')

2042
2043 # plt.subplot(3, 3, 7)
2044 # plt.imshow(np.transpose(sample_y[s, 2, :, :]), cmap='jet',
2045   ← vmin = minp, vmax = maxp, origin='lower',
2046   ← extent=[0,172,0,79])
2047 # plt.colorbar(orientation='horizontal')
2048 # plt.ylabel('p', fontsize=18)
2049 # plt.subplot(3, 3, 8)
2050 # plt.imshow(np.transpose(out_y[s, 2, :, :]), cmap='jet', vmin
2051   ← = minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
2052 # plt.colorbar(orientation='horizontal')
2053 # plt.subplot(3, 3, 9)
2054 # plt.imshow(np.transpose(error[s, 2, :, :]), cmap='jet', vmin
2055   ← = minep, vmax = maxep, origin='lower', extent=[0,172,0,79])
2056 # plt.colorbar(orientation='horizontal')
2057 plt.tight_layout()
2058 plt.show()

2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069 def visualize3(sample_y, s):

2067
2068     minu = np.min(sample_y[s, 0, :, :])
2069     maxu = np.max(sample_y[s, 0, :, :])

2070
2071     minv = np.min(sample_y[s, 1, :, :])
2072     maxv = np.max(sample_y[s, 1, :, :])

2073
2074     minp = np.min(sample_y[s, 2, :, :])
2075     maxp = np.max(sample_y[s, 2, :, :])

2076
2077     plt.figure()
2078     fig = plt.gcf()
2079     fig.set_size_inches(15, 10)

```

```

2070     plt.suptitle("Time instant " + str(s+1), fontsize=20)
2071     plt.subplot(3, 1, 1)
2072     plt.title('CFD', fontsize=18)
2073     plt.imshow((sample_y[s, 0, :, :]), cmap='jet', vmin = minu,
2074     ↳   vmax = maxu, origin='lower', extent=[0,172,0,79])
2075     plt.colorbar(orientation='horizontal')
2076     plt.ylabel('Ux', fontsize=18)
2077
2078     plt.subplot(3, 1, 2)
2079     plt.imshow((sample_y[s, 1, :, :]), cmap='jet', vmin = minv,
2080     ↳   vmax = maxv, origin='lower', extent=[0,172,0,79])
2081     plt.colorbar(orientation='horizontal')
2082     plt.ylabel('Uy', fontsize=18)
2083
2084     plt.subplot(3, 1, 3)
2085     plt.imshow((sample_y[s, 2, :, :]), cmap='jet', vmin = minp,
2086     ↳   vmax = maxp, origin='lower', extent=[0,172,0,79])
2087     plt.colorbar(orientation='horizontal')
2088     plt.ylabel('p', fontsize=18)
2089     plt.tight_layout()
2090     plt.show()
2091
2092 def visualize_rel_error(sample_y, out_y, error, s, input_velocity
2093   ):
2094
2095     # Visualize function with relative error added
2096
2097     minu = np.min(sample_y[s, 0, :, :])
2098     maxu = np.max(sample_y[s, 0, :, :])
2099
2100     minv = np.min(sample_y[s, 1, :, :])
2101     maxv = np.max(sample_y[s, 1, :, :])
2102
2103     minp = np.min(sample_y[s, 2, :, :])
2104     maxp = np.max(sample_y[s, 2, :, :])
2105
2106     mineu = np.min(error[s, 0, :, :])
2107     maxeu = np.max(error[s, 0, :, :])
2108
2109     minev = np.min(error[s, 1, :, :])
2110     maxev = np.max(error[s, 1, :, :])

```

```

2108
2109     minep = np.min(error[s, 2, :, :])
2110     maxep = np.max(error[s, 2, :, :])
2111
2112     plt.figure()
2113     fig = plt.gcf()
2114     fig.set_size_inches(15, 10)
2115     plt.suptitle("Time instant " + str(s+1), fontsize=20)
2116     plt.subplot(3, 4, 1)
2117     plt.title('CFD', fontsize=18)
2118     plt.imshow(np.transpose(sample_y[s, 0, :, :]), cmap='jet', vmin
2119     ↪ = minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2120     plt.colorbar(orientation='horizontal')
2121     plt.ylabel('Ux', fontsize=18)
2122     plt.subplot(3, 4, 2)
2123     plt.title('CNN', fontsize=18)
2124     plt.imshow(np.transpose(out_y[s, 0, :, :]), cmap='jet', vmin =
2125     ↪ minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2126     plt.colorbar(orientation='horizontal')
2127     plt.subplot(3, 4, 3)
2128     plt.title('Error', fontsize=18)
2129     plt.imshow(np.transpose(error[s, 0, :, :]), cmap='jet', vmin =
2130     ↪ mineu, vmax = maxeu, origin='lower', extent=[0,172,0,79])
2131     plt.colorbar(orientation='horizontal')
2132     plt.subplot(3, 4, 4)
2133
2134
2135     plt.subplot(3, 4, 5)
2136     plt.imshow(np.transpose(sample_y[s, 1, :, :]), cmap='jet', vmin
2137     ↪ = minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
2138     plt.colorbar(orientation='horizontal')
2139     plt.ylabel('Uy', fontsize=18)
2140     plt.subplot(3, 4, 6)
2141     plt.imshow(np.transpose(out_y[s, 1, :, :]), cmap='jet', vmin =
2142     ↪ minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
2143     plt.colorbar(orientation='horizontal')
2144     plt.subplot(3, 4, 7)
2145     plt.imshow(np.transpose(error[s, 1, :, :]), cmap='jet', vmin =
2146     ↪ minev, vmax = maxev, origin='lower', extent=[0,172,0,79])
2147     plt.colorbar(orientation='horizontal')
2148     plt.subplot(3, 4, 8)

```

```

2144
2145     plt.subplot(3, 4, 9)
2146     plt.imshow(np.transpose(sample_y[s, 2, :, :]), cmap='jet', vmin
2147     ↪ = minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
2148     plt.colorbar(orientation='horizontal')
2149     plt.ylabel('p', fontsize=18)
2150     plt.subplot(3, 4, 10)
2151     plt.imshow(np.transpose(out_y[s, 2, :, :]), cmap='jet', vmin =
2152     ↪ minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
2153     plt.colorbar(orientation='horizontal')
2154     plt.subplot(3, 4, 11)
2155     plt.imshow(np.transpose(error[s, 2, :, :]), cmap='jet', vmin =
2156     ↪ minep, vmax = maxep, origin='lower', extent=[0,172,0,79])
2157     plt.colorbar(orientation='horizontal')
2158
2159 def test_mean_error(plots):
2160     error = np.zeros([len(plots), 50]) # Habría que poner esto pero
2161     ↪ me da fallo --> len(plots[id_test][2]))
2162     max_error = error
2163     variance = error
2164     col = 0
2165     for key in plots:
2166         for row in range(50): #
2167             error[col, row] = np.mean(plots[key][2][row],
2168             ↪ dtype=np.float64)
2169             max_error[col, row] = np.max(plots[key][2][row])
2170             variance[col, row] = np.var(plots[key][2][row], ddof =
2171             ↪ 1)
2172             col = col + 1
2173     mean_error = np.mean(error, axis = 0, dtype=np.float64)
2174     max_error_all = np.max(max_error, axis = 0)
2175     variance_all = np.mean(variance, axis = 0)
2176     me_error = np.mean(mean_error)
2177     v_all = np.mean(variance_all)
2178     ma_error = np.max(max_error_all)
2179     return mean_error, max_error_all, variance_all, me_error, v_all,
2180     ↪ ma_error

```

```

2178 def plot_feature(feature_type, upper_limit, legend, plot_title,
2179     ↵ y_label):
2180     for i in range(10):
2181         plt.plot(feature_type[i], '--o', markersize = 3)
2182
2183     plt.grid(axis='both')
2184     plt.ylabel(y_label)
2185     plt.ylim((0,upper_limit))
2186     plt.legend(legend)
2187     plt.title(plot_title)
2188     plt.show()
2189
2190 def get_sub(x):
2191     normal =
2192         ↵ "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+-()"
2193     sub_s = "CDGQwZw"
2194     res = x.maketrans(''.join(normal), ''.join(sub_s))
2195     return x.translate(res)
2196
2197 def gen_data_distribution(CFD, out, sample):
2198     ## Distribucion de los datos:
2199     CFD_array = np.array(CFD)
2200     out_array = np.array(out)
2201
2202     fig, ax = plt.subplots()
2203     fig.set_size_inches(7, 4)
2204     # plt.title('Histogram u{}'.format(get_sub('y')) + '. Sample
2205     ↵ n.º ' + str(sample+1), fontsize = 20)
2206     plt.title('Histogram p' + '. Sample n.º ' + str(sample+1),
2207     ↵ fontsize = 20)
2208     fig.tight_layout()
2209     n, bins, patches =
2210         ↵ plt.hist(CFD_array[:,0,:,:].reshape(len(CFD)*79*172),
2211                         200, density=False, facecolor='r',
2212                         alpha=0.5)
2213     n, bins, patches =
2214         ↵ plt.hist(out_array[:,0,:,:].reshape(len(out)*79*172),
2215                         200, density=False, facecolor='b',
2216                         alpha=0.5)
2217     # plt.xlabel('u{}'.format(get_sub('y')) + ' [m/s]', fontsize =
2218     ↵ 18)
2219     plt.xlabel('p [Pa]', fontsize = 18)

```

```

2213     plt.rc('xtick', labelsize = 14)
2214     plt.rc('ytick', labelsize = 14)
2215     if sample == 0:
2216         plt.legend(['CFD', 'CNN'], fontsize = 16)
2217         plt.grid(True)
2218         ax.set_yscale('log')
2219         path = 'C:/Users/alvar/Desktop/Master/Segundo/TFM/2ª
2220             ↵ parte/Imagenes para TFM/Instantes futuros
2221             ↵ geometrias/Histogramas'
2222         plt.savefig(path + '/Histograma p muestra: ' + str(sample+1) +
2223             ↵ '.png', format="png")
2224         plt.show()
2225
2226
2227 if __name__ == "__main__":
2228
2229     mean_error_list = []
2230     max_error_list = []
2231     variance_list = []
2232     me_error_list = []
2233     var_list = []
2234     ma_error_list = []
2235
2236     plots = []
2237     training_info = []
2238     path = "E:/NuevasGeometrias/Resultados_NuevasGeometrias/"
2239     # path = "C:/Users/alvar/Desktop/Master/Segundo/TFM/2ª
2240         ↵ parte/Datos para entrenamiento/Resultados_vy_p/"
2241
2242     statistics_directory = "Estadisticas p/"
2243     plots_directory = "Plots p/"
2244
2245     filepaths = [f for f in listdir(str(path +
2246         ↵ statistics_directory)) if f.endswith('.json')]
2247     for file in filepaths:
2248         mean_error, max_error, variance, me_error, var_, ma_error =
2249             ↵ pickle.load(open(path + statistics_directory + file,
2250                 ↵ "rb")))
2251         mean_error_list.append(mean_error)
2252         max_error_list.append(max_error)
2253         variance_list.append(variance)
2254         me_error_list.append(me_error)
2255         var_list.append(var_)

```

```

2248     ma_error_list.append(ma_error)
2249     filepaths = [f for f in listdir(str(path + plots_directory)) if
2250      ↳  f.endswith('.json')]
2250   for file in filepaths:
2251     data = json.load(open(path + plots_directory + file, "rb"))
2252     training_info.append(data)
2253
2254   xlabel = []
2255   training_duration = []
2256   for i in range(len(training_info)):
2257     xlabel.insert(-1 , 'Test nº' + str(i+3))
2258
2259     ↳  training_duration.append(training_info[i]['training_duration']/3600)
2260   plt.plot(training_duration, '--bo', markersize = 3)
2261   plt.grid(axis='both')
2262   # plt.xlabel(xlabel)
2263   plt.ylabel('Time (min)')
2264   plt.title('Training duration of each model')
2265   plt.show()
2266
2267   table_data = []
2268   col_names = ["Id", "Train time (h)", "Num_epochs", "Lr",
2269    ↳  "Kernel_size", "Batch_size",
2270      ↳  "MeErr " + statistics_directory[-3:-1], "MaxErr "
2271      ↳  + statistics_directory[-3:-1]]
2272   for i in range(len(me_error_list)):
2273     table_data.append([filepaths[i][-7:-5],
2274
2275       ↳  training_info[i]["training_duration"]/3600,
2276
2277       ↳  training_info[i]["net_param"]['num_epochs'],
2278
2279       ↳  training_info[i]["net_param"]['learning_ratio'],
2280
2281       ↳  training_info[i]["net_param"]['kernel_size'],
2282
2283       ↳  training_info[i]["net_param"]['batch_size'],
2284       ↳  me_error_list[i], ma_error_list[i]])
2285   table = pd.DataFrame(table_data, columns = col_names)
2286   # table = tabulate(table_data, headers = col_names)
2287   print(table)
2288

```

```

2281     table.to_csv(path + statistics_directory +
2282                   'training_information.csv')
2283
2284     # plot_feature(mean_error_list, 70, legend, 'Mean error for
2285     # each prediction',
2286     #             'Mean error (m/s)')
2287     # plot_feature(max_error_list, 40, legend, 'Maximum error for
2288     # each prediction',
2289     #             'Max error (m/s)')
2290     # plot_feature(variance_list, 10, legend, 'Variance for each
2291     # prediction',
2292     #             'Variance (m^2/s^2)')
2293
2294
2295     filepaths = [f for f in listdir(str(path + plots_directory)) if
2296                   f.endswith('.pkl')]
2297
2298     # data = pickle.load(open(path + plots_directory +
2299     #                         filepaths[10], "rb"))
2300
2301     # data = pickle.load(open(path + plots_directory + filepaths[7],
2302     #                         "rb"))
2303
2304     data = pickle.load(open(path + plots_directory + filepaths[5],
2305     #                         "rb"))
2306
2307
2308     s = [0, 4, 9, 14, 19, 29, 39, 49]
2309
2310     for a in s:
2311         c = 0
2312
2313         CFD = np.zeros([len(data), 1, 79, 172])
2314         out = np.zeros([len(data), 1, 79, 172])
2315
2316         for key in data:
2317             CFD[c, 0, :, :] = data[key][0][a, 0, :, :]
2318             out[c, 0, :, :] = data[key][1][a, 0, :, :]
2319             c = c+1
2320
2321         gen_data_distribution(CFD, out, a)
2322
2323
2324     import os
2325     import json
2326     import torch
2327     import pickle
2328     import time
2329     from train_functions import *
2330     import torch.optim as optim

```

```

2314 from torch.utils.data import TensorDataset
2315 from Models.UNetEx import UNetEx
2316 from Models.UNet import UNet
2317 import numpy as np
2318 from matplotlib import pyplot as plt
2319 from os import listdir
2320 os.environ['KMP_DUPLICATE_LIB_OK']='True'
2321
2322 def visualize(sample_y, out_y, error, s, variable, n_sample,
2323   ↵ velocity):
2324
2325     minu = np.min(sample_y[s, 0, :, :])
2326     maxu = np.max(sample_y[s, 0, :, :])
2327
2328     # minv = np.min(sample_y[s, 1, :, :])
2329     # maxv = np.max(sample_y[s, 1, :, :])
2330
2331     # minp = np.min(sample_y[s, 2, :, :])
2332     # maxp = np.max(sample_y[s, 2, :, :])
2333
2334     mineu = np.min(error[s, 0, :, :])
2335     maxeu = np.max(error[s, 0, :, :])
2336
2337     # minev = np.min(error[s, 1, :, :])
2338     # maxev = np.max(error[s, 1, :, :])
2339
2340     # minep = np.min(error[s, 2, :, :])
2341     # maxep = np.max(error[s, 2, :, :])
2342
2343     plt.figure()
2344     fig = plt.gcf()
2345     plt.suptitle("Input velocity: " + velocity + " ms. Sample nº: "
2346      ↵ + str(n_sample+s), fontsize=20)
2347     fig.set_size_inches(15, 10)
2348     plt.subplot(3, 3, 1)
2349     plt.title('CFD', fontsize=18)
2350     plt.imshow(np.transpose(sample_y[s, 0, :, :]), cmap='jet', vmin
2351      ↵ = minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2352     plt.colorbar(orientation='horizontal')
2353     plt.ylabel(variable, fontsize=18)
2354     plt.subplot(3, 3, 2)
2355     plt.title('CNN', fontsize=18)

```

```

2353 plt.imshow(np.transpose(out_y[s, 0, :, :]), cmap='jet', vmin =
2354     ↪ minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2355 plt.colorbar(orientation='horizontal')
2356 plt.subplot(3, 3, 3)
2357 plt.title('Error', fontsize=18)
2358 plt.imshow(np.transpose(error[s, 0, :, :]), cmap='jet', vmin =
2359     ↪ mineu, vmax = maxeu, origin='lower', extent=[0,172,0,79])
2360 plt.colorbar(orientation='horizontal')
2361
2362 # plt.subplot(3, 3, 4)
2363 # plt.imshow(np.transpose(sample_y[s, 1, :, :]), cmap='jet',
2364     ↪ vmin = minv, vmax = maxv, origin='lower',
2365     ↪ extent=[0,172,0,79])
2366 # plt.colorbar(orientation='horizontal')
2367 # plt.ylabel('Uy', fontsize=18)
2368 # plt.subplot(3, 3, 5)
2369 # plt.imshow(np.transpose(out_y[s, 1, :, :]), cmap='jet', vmin
2370     ↪ = minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
2371 # plt.colorbar(orientation='horizontal')
2372 # plt.subplot(3, 3, 6)
2373 # plt.imshow(np.transpose(error[s, 1, :, :]), cmap='jet', vmin
2374     ↪ = minev, vmax = maxev, origin='lower', extent=[0,172,0,79])
2375 # plt.colorbar(orientation='horizontal')
2376
2377 # plt.subplot(3, 3, 7)
2378 # plt.imshow(np.transpose(sample_y[s, 2, :, :]), cmap='jet',
2379     ↪ vmin = minp, vmax = maxp, origin='lower',
2380     ↪ extent=[0,172,0,79])
2381 # plt.colorbar(orientation='horizontal')
2382 # plt.ylabel('p', fontsize=18)
2383 # plt.subplot(3, 3, 8)
2384 # plt.imshow(np.transpose(out_y[s, 2, :, :]), cmap='jet', vmin
2385     ↪ = minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
2386 # plt.colorbar(orientation='horizontal')
2387 # plt.subplot(3, 3, 9)
2388 # plt.imshow(np.transpose(error[s, 2, :, :]), cmap='jet', vmin
2389     ↪ = minep, vmax = maxep, origin='lower', extent=[0,172,0,79])
2390 # plt.colorbar(orientation='horizontal')
2391 plt.tight_layout()
2392 plt.show()

```

```

2385 def visualize2(sample_y, s, file):
2386
2387     plt.figure()
2388     fig = plt.gcf()
2389     # fig.set_size_inches(15, 10)
2390
2391     minu = np.min(sample_y[s, 0, :, :])
2392     maxu = np.max(sample_y[s, 0, :, :])
2393
2394     plt.imshow(np.transpose(sample_y[s, 0, :, :]), cmap='jet', vmin
2395     ↪ = minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2396     plt.colorbar(orientation='horizontal')
2397     plt.title("Test " + file)
2398     plt.tight_layout()
2399     plt.show()
2400
2401 def test_CNN(n_instantes, x_ms, n_test):
2402
2403     ti = time.time()
2404
2405     out = torch.empty((n_instantes,3,172,79))
2406     x_ms = torch.FloatTensor(x_ms)
2407     out[:,0:2,:,:]=x_ms[0,:2,:,:]
2408
2409     out[0,2,:,:] = model_prueba(x_ms[n_test:n_test+1])
2410
2411     for i in range (1,n_instantes,1):
2412         print("Sample: " + str(i))
2413         out[i,2,:,:] = model_prueba(out[i-1:i])
2414         out2=out[:,2:3,:,:]
2415         out2=out2.reshape([n_instantes,1,172,79])
2416         tout = time.time()
2417
2418         duration = tout-ti
2419
2420     return out2, duration
2421
2422 def gen_data_distribution(CFD, out, sample):
2423     ## Distribucion de los datos:
2424     CFD_array = np.array(CFD)
2425     out_array = np.array(out)

```

```

2426     fig, ax = plt.subplots()
2427     fig.set_size_inches(7, 4)
2428     # plt.title('Histogram u{}'.format(get_sub('y')) + '. Sample'
2429     #             ' n.º ' + str(sample+1), fontsize = 20)
2430     plt.title('Histogram p' + ' . Sample n.º ' + str(sample),
2431               fontsize = 20)
2432     fig.tight_layout()
2433     n, bins, patches =
2434         plt.hist(CFD_array[:,0,:,:].reshape(len(CFD)*79*172),
2435                  200, density=False, facecolor='r',
2436                  alpha=0.5)
2437     n, bins, patches =
2438         plt.hist(out_array[:,0,:,:].reshape(len(out)*79*172),
2439                  200, density=False, facecolor='b',
2440                  alpha=0.5)
2441     # plt.xlabel('u{}'.format(get_sub('y')) + ' [m/s]', fontsize =
2442     #             18)
2443     plt.xlabel('p [Pa]', fontsize = 18)
2444     plt.rc('xtick', labelsize = 14)
2445     plt.rc('ytick', labelsize = 14)
2446     if sample == 1:
2447         plt.legend(['CFD', 'CNN'], fontsize = 16)
2448     plt.grid(True)
2449     ax.set_yscale('log')
2450     # path = 'C:/Users/alvar/Desktop/Master/Segundo/TFM/2ª
2451     #       parte/Imagenes para TFM/Instantes futuros
2452     #       geometrias/Histogramas'
2453     # plt.savefig(path + '/Histogram p muestra: ' + str(sample+1) +
2454     #             '.png', format="png")
2455     plt.show()

2456 if __name__ == "__main__":
2457
2458     path = "E:/NuevasGeometrias/Resultados_NuevasGeometrias
2459             /Modelos/"
2460
2461     torch.manual_seed(0)
2462     # lr = 0.001
2463     lr = 0.0001
2464     # kernel_size = 3
2465     kernel_size = 5
2466     # kernel_size = 7

```

```

2459     filters = [8, 16, 32, 32]
2460     bn = False
2461     wn = False
2462     wd = 0.005
2463     beta1 = 0.5
2464     beta2 = 0.5
2465
2466     model_prueba = UNetEx(3, 1, filters=filters,
2467         ↪ kernel_size=kernel_size, batch_norm=bn, weight_norm=wn)
2468     # model_prueba.load_state_dict(torch.load(path +
2469         ↪ "Modelo_geometries_vx15.py",
2470         ↪ map_location=torch.device('cpu')))
2471     model_prueba.load_state_dict(torch.load(path +
2472         ↪ "Modelo_geometries_vy207.py",
2473         ↪ map_location=torch.device('cpu')))
2474     # model_prueba.load_state_dict(torch.load(path +
2475         ↪ "Modelo_geometries_p205.py",
2476         ↪ map_location=torch.device('cpu')))
2477     model_prueba.eval()
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

```

2486     input_data.append(pickle.load(open(path +
2487         "input_data_geometry_" + geometry[g_i] +
2488         "_size_factor_1_rotation_angle_1_variable_vx.pkl",
2489         "rb")))
2490
2491     x = np.zeros([1,3,172,79])
2492     x[:, -1, :, :] = input_data[0][0][:, 1, :, :].detach().numpy()
2493     x[:, 0, :, :] = input_data[0][1]
2494     x[:, 1, :, :] = input_data[0][2]
2495
2496     # with open(simulation_directory + "plots_" + velocity
2497     #             + "ms_" + variable + "_Final_v2" + str(n_test) +
2498     #             ".pkl", "wb") as file:
2499     #     pickle.dump([out, error, y], file)
2500
2501     variable = 'vy'
2502     n_instantes = 50
2503     out2, duration = test_CNN(n_instantes, x, 0)
2504     # cfd = cfd[:, [1,2,0], :, :]
2505     cfd = torch.tensor(np.moveaxis(cfd, [2,3], [3,2]))
2506     cfd = cfd[:, 2:3, :, :]
2507     error = abs(cfd[:n_instantes, :, :, :] - out2)
2508     s = [0, 4, 9, 14, 19, 29, 39, 49]
2509     for a in s:
2510         # visualize2(out2.detach().numpy(), s, filepaths[0])
2511         visualize(cfd.detach().numpy(), out2.detach().numpy(),
2512                    error.detach().numpy(), a, variable, 1, "5")
2513
2514     path2 = "E:/NuevasGeometrias/
2515         Resultados_NuevasGeometrias/test_nuevas_geometrias/"
2516     with open(path2 + "test_" + geometry[g_i] + '_' + variable
2517     + ".pkl", "wb") as file:
2518         pickle.dump([cfд.detach().numpy()[:n_instantes],
2519                     out2.detach().numpy(),
2520                     error.detach().numpy()], file)
2521
2522     cfd_all[g_i*50:(g_i+1)*50] = cfd[:50].detach().numpy()
2523     out_all[g_i*50:(g_i+1)*50] = out2.detach().numpy()
2524
2525     # cfd_all = np.array(cfd_all[:].detach().numpy())
2526     # out_all = np.array(out_all.detach().numpy())

```

```

2519     # mean_error_vx =
2520     ↳ np.mean(error[:,0,:,:].cpu().detach().numpy())
2521     # mean_error_vy =
2522     ↳ np.mean(error[:,1,:,:].cpu().detach().numpy())
2523     # mean_error_p = np.mean(error[:,2,:,:].cpu().detach().numpy())
2524
2525
2526     mean_value_cfd = np.mean(cfd_all[:,0,:,:])
2527     std_cfd = np.std(cfd_all[:,0,:,:])
2528
2529
2530     mean_value_cnn = np.mean(out_all[:,0,:,:])
2531     std_cnn = np.std(out_all[:,0,:,:])
2532
2533
2534     sample10 = [x+10 for x in sample1]
2535     sample15 = [x+15 for x in sample1]
2536     sample20 = [x+20 for x in sample1]
2537     sample30 = [x+30 for x in sample1]
2538     sample40 = [x+40 for x in sample1]
2539     sample50 = [x+50 for x in sample1]
2540
2541     sample = [sample1, sample5, sample10, sample15, sample20,
2542               ↳ sample30,
2543                 sample40, sample50]
2544
2545     samples = [1, 5, 10, 15, 20, 30, 40, 50]
2546
2547
2548     for s in range(len(sample)):
2549         cfd = cfd_all[sample[s]]
2550         out = out_all[sample[s]]
2551         gen_data_distribution(cfd, out, samples[s])

```

Test del modelo neuronal y análisis de los resultados de la CNN que predice el primer instante para las geometrías variables:

```

2543 import pickle
2544 import numpy as np
2545 import math
2546 import random
2547 from matplotlib import pyplot as plt
2548 import matplotlib.gridspec as gridspec
2549 import statistics as st
2550 from sklearn import preprocessing
2551 from os import listdir
2552 from tabulate import tabulate
2553 import json

```

```

2554 import pandas as pd
2555
2556
2557 def visualize(sample_y, out_y, error, s, variable, n_sample,
2558   ↪ velocity):
2559
2560     minu = np.min(sample_y[s, :, :])
2561     maxu = np.max(sample_y[s, :, :])
2562
2563     # minv = np.min(sample_y[s, 1, :, :])
2564     # maxv = np.max(sample_y[s, 1, :, :])
2565
2566     # minp = np.min(sample_y[s, 2, :, :])
2567     # maxp = np.max(sample_y[s, 2, :, :])
2568
2569     mineu = np.min(error[s, :, :])
2570     maxeu = np.max(error[s, :, :])
2571
2572     # minev = np.min(error[s, 1, :, :])
2573     # maxev = np.max(error[s, 1, :, :])
2574
2575     # minep = np.min(error[s, 2, :, :])
2576     # maxep = np.max(error[s, 2, :, :])
2577
2578     plt.figure()
2579     fig = plt.gcf()
2580     plt.suptitle("Input velocity: " + velocity + " ms. Sample nº: "
2581       ↪ + str(n_sample+s), fontsize=20)
2582     fig.set_size_inches(15, 10)
2583     plt.subplot(3, 3, 1)
2584     plt.title('CFD', fontsize=18)
2585     plt.imshow(np.transpose(sample_y[s, :, :]), cmap='jet', vmin =
2586       ↪ minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2587     plt.colorbar(orientation='horizontal')
2588     plt.ylabel(variable, fontsize=18)
2589     plt.subplot(3, 3, 2)
2590     plt.title('CNN', fontsize=18)
2591     plt.imshow(np.transpose(out_y[s, :, :]), cmap='jet', vmin =
2592       ↪ minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2593     plt.colorbar(orientation='horizontal')
2594     plt.subplot(3, 3, 3)
2595     plt.title('Error', fontsize=18)

```

```

2592 plt.imshow(np.transpose(error[s, :, :]), cmap='jet', vmin =
2593     ↪ mineu, vmax = maxeu, origin='lower', extent=[0,172,0,79])
2594 plt.colorbar(orientation='horizontal')
2595
2596 # plt.subplot(3, 3, 4)
2597 # plt.imshow(np.transpose(sample_y[s, 1, :, :]), cmap='jet',
2598     ↪ vmin = minv, vmax = maxv, origin='lower',
2599     ↪ extent=[0,172,0,79])
2600 # plt.colorbar(orientation='horizontal')
2601 # plt.ylabel('Uy', fontsize=18)
2602 # plt.subplot(3, 3, 5)
2603 # plt.imshow(np.transpose(out_y[s, 1, :, :]), cmap='jet', vmin
2604     ↪ = minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
2605 # plt.colorbar(orientation='horizontal')
2606 # plt.subplot(3, 3, 6)
2607 # plt.imshow(np.transpose(error[s, 1, :, :]), cmap='jet', vmin
2608     ↪ = minev, vmax = maxev, origin='lower', extent=[0,172,0,79])
2609 # plt.colorbar(orientation='horizontal')
2610
2611 # plt.subplot(3, 3, 7)
2612 # plt.imshow(np.transpose(sample_y[s, 2, :, :]), cmap='jet',
2613     ↪ vmin = minp, vmax = maxp, origin='lower',
2614     ↪ extent=[0,172,0,79])
2615 # plt.colorbar(orientation='horizontal')
2616 # plt.ylabel('p', fontsize=18)
2617 # plt.subplot(3, 3, 8)
2618 # plt.imshow(np.transpose(out_y[s, 2, :, :]), cmap='jet', vmin
2619     ↪ = minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
2620 # plt.colorbar(orientation='horizontal')
2621 plt.tight_layout()
2622 plt.show()

2623
2624 def visualize3(sample_y, s):
2625
2626     minu = np.min(sample_y[s, 0, :, :])
2627     maxu = np.max(sample_y[s, 0, :, :])

```

```

2625     minv = np.min(sample_y[s, 1, :, :])
2626     maxv = np.max(sample_y[s, 1, :, :])
2627
2628     minp = np.min(sample_y[s, 2, :, :])
2629     maxp = np.max(sample_y[s, 2, :, :])
2630
2631     plt.figure()
2632     fig = plt.gcf()
2633     fig.set_size_inches(15, 10)
2634     plt.suptitle("Time instant " + str(s+1), fontsize=20)
2635     plt.subplot(3, 1, 1)
2636     plt.title('CFD', fontsize=18)
2637     plt.imshow((sample_y[s, 0, :, :]), cmap='jet', vmin = minu,
2638                → vmax = maxu, origin='lower', extent=[0,172,0,79])
2639     plt.colorbar(orientation='horizontal')
2640     plt.ylabel('Ux', fontsize=18)
2641
2642     plt.subplot(3, 1, 2)
2643     plt.imshow((sample_y[s, 1, :, :]), cmap='jet', vmin = minv,
2644                → vmax = maxv, origin='lower', extent=[0,172,0,79])
2645     plt.colorbar(orientation='horizontal')
2646     plt.ylabel('Uy', fontsize=18)
2647
2648     plt.subplot(3, 1, 3)
2649     plt.imshow((sample_y[s, 2, :, :]), cmap='jet', vmin = minp,
2650                → vmax = maxp, origin='lower', extent=[0,172,0,79])
2651     plt.colorbar(orientation='horizontal')
2652     plt.ylabel('p', fontsize=18)
2653     plt.tight_layout()
2654     plt.show()
2655
2656 def visualize_rel_error(sample_y, out_y, error, s, input_velocity
2657   ):
2658
2659   # Visualize function with relative error added
2660
2661   minu = np.min(sample_y[s, 0, :, :])
2662   maxu = np.max(sample_y[s, 0, :, :])
2663
2664   minv = np.min(sample_y[s, 1, :, :])
2665   maxv = np.max(sample_y[s, 1, :, :])

```

```

2663
2664     minp = np.min(sample_y[s, 2, :, :])
2665     maxp = np.max(sample_y[s, 2, :, :])
2666
2667     mineu = np.min(error[s, 0, :, :])
2668     maxeu = np.max(error[s, 0, :, :])
2669
2670     minev = np.min(error[s, 1, :, :])
2671     maxev = np.max(error[s, 1, :, :])
2672
2673     minep = np.min(error[s, 2, :, :])
2674     maxep = np.max(error[s, 2, :, :])
2675
2676     plt.figure()
2677     fig = plt.gcf()
2678     fig.set_size_inches(15, 10)
2679     plt.suptitle("Time instant " + str(s+1), fontsize=20)
2680     plt.subplot(3, 4, 1)
2681     plt.title('CFD', fontsize=18)
2682     plt.imshow(np.transpose(sample_y[s, 0, :, :]), cmap='jet', vmin
2683     ↪ = minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2684     plt.colorbar(orientation='horizontal')
2685     plt.ylabel('Ux', fontsize=18)
2686     plt.subplot(3, 4, 2)
2687     plt.title('CNN', fontsize=18)
2688     plt.imshow(np.transpose(out_y[s, 0, :, :]), cmap='jet', vmin =
2689     ↪ minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2690     plt.colorbar(orientation='horizontal')
2691     plt.subplot(3, 4, 3)
2692     plt.title('Error', fontsize=18)
2693     plt.imshow(np.transpose(error[s, 0, :, :]), cmap='jet', vmin =
2694     ↪ mineu, vmax = maxeu, origin='lower', extent=[0,172,0,79])
2695     plt.colorbar(orientation='horizontal')
2696     plt.subplot(3, 4, 4)
2697
2698     plt.subplot(3, 4, 5)
2699     plt.imshow(np.transpose(sample_y[s, 1, :, :]), cmap='jet', vmin
2700     ↪ = minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
    plt.colorbar(orientation='horizontal')
    plt.ylabel('Uy', fontsize=18)
    plt.subplot(3, 4, 6)

```

```

2701 plt.imshow(np.transpose(out_y[s, 1, :, :]), cmap='jet', vmin =
2702     ↪ minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
2703 plt.colorbar(orientation='horizontal')
2704 plt.subplot(3, 4, 7)
2705 plt.imshow(np.transpose(error[s, 1, :, :]), cmap='jet', vmin =
2706     ↪ minev, vmax = maxev, origin='lower', extent=[0,172,0,79])
2707 plt.colorbar(orientation='horizontal')
2708 plt.subplot(3, 4, 8)

2709 plt.subplot(3, 4, 9)
2710 plt.imshow(np.transpose(sample_y[s, 2, :, :]), cmap='jet', vmin
2711     ↪ = minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
2712 plt.colorbar(orientation='horizontal')
2713 plt.ylabel('p', fontsize=18)
2714 plt.subplot(3, 4, 10)
2715 plt.imshow(np.transpose(out_y[s, 2, :, :]), cmap='jet', vmin =
2716     ↪ minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
2717 plt.colorbar(orientation='horizontal')
2718 plt.subplot(3, 4, 11)
2719 plt.imshow(np.transpose(error[s, 2, :, :]), cmap='jet', vmin =
2720     ↪ minep, vmax = maxep, origin='lower', extent=[0,172,0,79])
2721 plt.colorbar(orientation='horizontal')
2722 plt.tight_layout()
2723 plt.show()

2724 def test_mean_error(plots):
2725     error = np.zeros([len(plots), 50]) # Habría que poner esto pero
2726     ↪ me da fallo --> len(plots[id_test][2]))
2727     max_error = error
2728     variance = error
2729     col = 0
2730     for key in plots:
2731         for row in range(50): #
2732             error[col, row] = np.mean(plots[key][2][row],
2733                 ↪ dtype=np.float64)
2734             max_error[col, row] = np.max(plots[key][2][row])
2735             variance[col, row] = np.var(plots[key][2][row], ddof =
2736                 ↪ 1)
2737             col = col + 1
2738     mean_error = np.mean(error, axis = 0, dtype=np.float64)

```

```

2735     max_error_all = np.max(max_error, axis = 0)
2736     variance_all = np.mean(variance, axis = 0)
2737     me_error = np.mean(mean_error)
2738     v_all = np.mean(variance_all)
2739     ma_error = np.max(max_error_all)
2740     return mean_error, max_error_all, variance_all, me_error, v_all,
2741           ↵   ma_error
2742
2743 def plot_feature(feature_type, upper_limit, legend, plot_title,
2744   ↵   y_label):
2745     for i in range(10):
2746       plt.plot(feature_type[i], '--o', markersize = 3)
2747
2748     plt.grid(axis='both')
2749     plt.ylabel(y_label)
2750     plt.ylim((0,upper_limit))
2751     plt.legend(legend)
2752     plt.title(plot_title)
2753     plt.show()
2754
2755 def get_sub(x):
2756   normal =
2757   ↵   "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+-=()"
2758   sub_s = "CDGQwZw"
2759   res = x.maketrans(''.join(normal), ''.join(sub_s))
2760   return x.translate(res)
2761
2762 def gen_data_distribution(CFD, out):
2763   ## Distribucion de los datos:
2764   CFD_array = np.array(CFD)
2765   out_array = np.array(out)
2766
2767   fig, ax = plt.subplots()
2768   fig.set_size_inches(7, 4)
2769   plt.title('Histogram u{}'.format(get_sub('x')), fontsize = 20)
2770   fig.tight_layout()
2771   fig.subplots_adjust(top=0.95)
2772   n, bins, patches =
2773   ↵   plt.hist(CFD_array[:,0,:,:].reshape(10*79*172),
2774               200, density=False, facecolor='r',
2775               alpha=0.5)

```

```

2772     n, bins, patches =
2773         plt.hist(out_array[:,0,:,:].reshape(10*79*172),
2774                         200, density=False, facecolor='b',
2775                         alpha=0.5)
2776     plt.xlabel('u{}' .format(get_sub('x')) + ' [m/s]', fontsize =
2777             18)
2778     plt.ylabel('Nº of points', fontsize = 18)
2779     plt.rc('xtick', labelsize = 14)
2780     plt.rc('ytick', labelsize = 14)
2781     plt.legend(['CFD', 'CNN'], fontsize = 18)
2782     ax.set_yscale('log')
2783     # ax.set_xscale('log')
2784     plt.grid(True)
2785     # fig.savefig('./ResultadosFinales/data_distrubution_vx_' +
2786     #             vel_title + '.svg', bbox_inches='tight')
2787     plt.show()

2788
2789     fig, ax = plt.subplots()
2790     fig.set_size_inches(7, 4)
2791     plt.title('Histogram u{}' .format(get_sub('y')), fontsize = 20)
2792     fig.tight_layout()
2793     n, bins, patches =
2794         plt.hist(CFD_array[:,1,:,:].reshape(10*79*172),
2795                         200, density=False, facecolor='r',
2796                         alpha=0.5)
2797     n, bins, patches =
2798         plt.hist(out_array[:,1,:,:].reshape(10*79*172),
2799                         200, density=False, facecolor='b',
2800                         alpha=0.5)
2801     plt.xlabel('u{}' .format(get_sub('y')) + ' [m/s]', fontsize =
2802             18)
2803     # plt.legend(['CFD', 'CNN'])
2804     plt.rc('xtick', labelsize = 14)
2805     plt.rc('ytick', labelsize = 14)
2806     ax.set_yscale('log')
2807     plt.grid(True)
2808     # fig.savefig('./ResultadosFinales/data_distrubution_vy_' +
2809     #             vel_title + '.svg', bbox_inches='tight')
2810     plt.show()

2811
2812     fig, ax = plt.subplots()

```

```

2807     fig.set_size_inches(7, 4)
2808     plt.title('Histogram p', fontsize = 20)
2809     fig.tight_layout()
2810     n, bins, patches =
2811         ↳ plt.hist(CFD_array[:,2,:,:].reshape(10*79*172),
2812                     200, density=False, facecolor='r',
2813                     alpha=0.5)
2814     n, bins, patches =
2815         ↳ plt.hist(out_array[:,2,:,:].reshape(10*79*172),
2816                     200, density=False, facecolor='b',
2817                     alpha=0.5)
2818
2819     plt.xlabel('p [Pa]', fontsize = 18)
2820     plt.rc('xtick', labelsize = 14)
2821     plt.rc('ytick', labelsize = 14)
2822     # plt.legend(['CFD', 'CNN'])
2823     plt.grid(True)
2824     ax.set_yscale('log')
2825     # fig.savefig('./ResultadosFinales/data_distrubution_p_ +
2826         ↳ vel_title + '.svg', bbox_inches='tight')
2827     plt.show()

2828 if __name__ == "__main__":
2829
2830     # mean_error_list = []
2831     # max_error_list = []
2832     # variance_list = []
2833     # me_error_list = []
2834     # var_list = []
2835     # ma_error_list = []
2836
2837     # plots = []
2838     training_info = []
2839     path = "E:/NuevasGeometrias/Resultados_NuevasGeometrias
2840         ↳ /Resultados_first_sample/"
2841
2842     filepaths = [f for f in listdir(str(path) + "Estadisticas/") if
2843         ↳ f.endswith('.json')]
2844     for file in filepaths:
2845         mean_error, max_error, variance, me_error, var_, ma_error =
2846             ↳ pickle.load(open(path + "Estadisticas/" + file, "rb"))
2847         # mean_error_list.append(mean_error)

```

```

2843     # max_error_list.append(max_error)
2844     # variance_list.append(variance)
2845     # me_error_list.append(me_error)
2846     # var_list.append(var_)
2847     # ma_error_list.append(ma_error)
2848     filepaths = [f for f in listdir(str(path)) if
2849      ↳ f.endswith('.json')]
2850     for file in filepaths:
2851         data = json.load(open(path + file, "rb"))
2852         training_info.append(data)
2853
2854     xlabel = []
2855     training_duration = []
2856     for i in range(len(training_info)):
2857         xlabel.insert(-1 , 'Test nº' + str(i+3))
2858         training_duration.append(training_info[i][1]/3600)
2859     plt.plot(training_duration, '--bo', markersize = 3)
2860     plt.grid(axis='both')
2861     # plt.xlabel(xlabel)
2862     plt.ylabel('Time (min)')
2863     plt.title('Training duration of each model')
2864     plt.show()
2865
2866
2867     table_data = []
2868     col_names = ["Id", "Train time (h)", "Num_epochs", "Lr",
2869      ↳ "Kernel_size", "Batch_size", "Architecture",
2870          "MeErr_vx", "MeErr_vy", "MeErr_p", "MaxErr_vx",
2871          ↳ "MaxErr_vy", "MaxErr_p"]
2872     for i in range(len(mean_error)):
2873         table_data.append([filepaths[i][19:22],
2874                           training_info[i][1]/3600,
2875
2876                           ↳ training_info[i][0]["net_param"]['num_epochs'],
2877
2878                           ↳ training_info[i][0]["net_param"]['learning_ratio'],
2879
2880                           ↳ training_info[i][0]["net_param"]['kernel_size'],
2881
2882                           ↳ training_info[i][0]["net_param"]['batch_size'],

```

```

2877
2878         ↵   training_info[i][0]["net_param"]["arquitecture"] ,
2879         mean_error[i][0], mean_error[i][1],
2880         ↵   mean_error[i][2],
2881         max_error[i][0], max_error[i][1],
2882         ↵   max_error[i][2]))
2883
2884     table = pd.DataFrame(table_data, columns = col_names)
2885     # table = tabulate(table_data, headers = col_names)
2886     print(table)
2887
2888     table_less_mean_error =
2889     ↵   table.sort_values(by=["MeErr_vx"])#[:10]
2890     table_less_max_error =
2891     ↵   table.sort_values(by=["MaxErr_vx"])#[:10]
2892     print(table_less_mean_error)
2893     print(table_less_max_error)
2894
2895     # plot_feature(mean_error_list, 70, legend, 'Mean error for
2896     ↵   each prediction',
2897     #           'Mean error (m/s)')
2898     # plot_feature(max_error_list, 40, legend, 'Maximum error for
2899     ↵   each prediction',
2900     #           'Max error (m/s)')
2901     # plot_feature(variance_list, 10, legend, 'Variance for each
2902     ↵   prediction',
2903     #           'Variance (m^2/s^2)')
2904
2905
2906
2907     #### Histograms
2908
2909
2910     filepaths = [f for f in listdir(str(path)) if
2911     ↵   f.endswith('.pkl')]
2912     best_model_plots = pickle.load(open(path + filepaths[9], "rb"))
2913
2914
2915     cfd = best_model_plots[2]
2916     out = best_model_plots[0]
2917
2918
2919     gen_data_distribution(cfd, out)
2920
2921
2922     import os
2923     import json

```

```

2909 import torch
2910 import pickle
2911 import time
2912 from train_functions import *
2913 import torch.optim as optim
2914 from torch.utils.data import TensorDataset
2915 from Models.UNetEx import UNetEx
2916 from Models.UNet import UNet
2917 import numpy as np
2918 from matplotlib import pyplot as plt
2919 from scipy.io import loadmat
2920 os.environ['KMP_DUPLICATE_LIB_OK']='True'
2921
2922
2923 def visualize(sample_y, out_y, error, s, variable, n_sample,
2924   ↪ velocity):
2925
2926     minu = np.min(sample_y[s, 0, :, :])
2927     maxu = np.max(sample_y[s, 0, :, :])
2928
2929     minv = np.min(sample_y[s, 1, :, :])
2930     maxv = np.max(sample_y[s, 1, :, :])
2931
2932     minp = np.min(sample_y[s, 2, :, :])
2933     maxp = np.max(sample_y[s, 2, :, :])
2934
2935     mineu = np.min(error[s, 0, :, :])
2936     maxeu = np.max(error[s, 0, :, :])
2937
2938     minev = np.min(error[s, 1, :, :])
2939     maxev = np.max(error[s, 1, :, :])
2940
2941     minep = np.min(error[s, 2, :, :])
2942     maxep = np.max(error[s, 2, :, :])
2943
2944     plt.figure()
2945     fig = plt.gcf()
2946     plt.suptitle("Input velocity: " + velocity + " ms. Sample nº: "
2947       ↪ + str(n_sample+s), fontsize=20)
2948     fig.set_size_inches(15, 10)
2949     plt.subplot(3, 3, 1)
2950     plt.title('CFD', fontsize=18)

```

```

2949     plt.imshow(np.transpose(sample_y[s, 0, :, :]), cmap='jet', vmin
2950         ← = minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2951     plt.colorbar(orientation='horizontal')
2952     plt.ylabel(variable, fontsize=18)
2953     plt.subplot(3, 3, 2)
2954     plt.title('CNN', fontsize=18)
2955     plt.imshow(np.transpose(out_y[s, 0, :, :]), cmap='jet', vmin =
2956         ← minu, vmax = maxu, origin='lower', extent=[0,172,0,79])
2957     plt.colorbar(orientation='horizontal')
2958     plt.subplot(3, 3, 3)
2959     plt.title('Error', fontsize=18)
2960     plt.imshow(np.transpose(error[s, 0, :, :]), cmap='jet', vmin =
2961         ← mineu, vmax = maxeu, origin='lower', extent=[0,172,0,79])
2962     plt.colorbar(orientation='horizontal')

2963     plt.subplot(3, 3, 4)
2964     plt.imshow(np.transpose(sample_y[s, 1, :, :]), cmap='jet', vmin
2965         ← = minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
2966     plt.colorbar(orientation='horizontal')
2967     plt.ylabel('Uy', fontsize=18)
2968     plt.subplot(3, 3, 5)
2969     plt.imshow(np.transpose(out_y[s, 1, :, :]), cmap='jet', vmin =
2970         ← minv, vmax = maxv, origin='lower', extent=[0,172,0,79])
2971     plt.colorbar(orientation='horizontal')
2972     plt.subplot(3, 3, 6)
2973     plt.imshow(np.transpose(error[s, 1, :, :]), cmap='jet', vmin =
2974         ← minev, vmax = maxev, origin='lower', extent=[0,172,0,79])
2975     plt.colorbar(orientation='horizontal')

2976     plt.subplot(3, 3, 7)
2977     plt.imshow(np.transpose(sample_y[s, 2, :, :]), cmap='jet', vmin
2978         ← = minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
2979     plt.colorbar(orientation='horizontal')
2980     plt.ylabel('p', fontsize=18)
2981     plt.subplot(3, 3, 8)
2982     plt.imshow(np.transpose(out_y[s, 2, :, :]), cmap='jet', vmin =
2983         ← minp, vmax = maxp, origin='lower', extent=[0,172,0,79])
2984     plt.colorbar(orientation='horizontal')
2985     plt.subplot(3, 3, 9)
2986     plt.imshow(np.transpose(error[s, 2, :, :]), cmap='jet', vmin =
2987         ← minep, vmax = maxep, origin='lower', extent=[0,172,0,79])

```

```

2982     plt.colorbar(orientation='horizontal')
2983     plt.tight_layout()
2984     plt.show()

2985
2986
2987 def visualize2(sample_y, s, geometry, rotation_angle, size_factor,
2988   ↵ variable, path):
2989
2990     plt.figure()
2991     fig = plt.gcf()
2992     # fig.set_size_inches(15, 10)
2993
2994     minu = np.min(sample_y[s, variable, :, :])
2995     maxu = np.max(sample_y[s, variable, :, :])
2996
2997     plt.imshow(np.transpose(sample_y[s, variable, :, :]),
2998       ↵ cmap='jet',
2999         vmin = minu, vmax = maxu, origin='lower',
3000           ↵ extent=[0,172,0,79])
3001     plt.colorbar(orientation='horizontal')
3002     plt.ylabel('vx', fontsize = 14)
3003     plt.title("Test " + geometry + ", size factor: " +
3004       ↵ str(size_factor)
3005         + ", rot. angle: " + str(rotation_angle), fontsize =
3006           ↵ 14)
3007     plt.tight_layout()
3008     plt.savefig(path + "geometry_" + geometry + "_size_factor_" +
3009       ↵ str(size_factor) + "_rotation_angle_"
3010         + str(rotation_angle) + "_variable_vx" + ".png" ,
3011           ↵ bbox_inches = 'tight')
3012     plt.show()

3013
3014 def get_sub(x):
3015   normal =
3016     ↵ "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+-()"
3017   sub_s = "CDGQwZw"
3018   res = x.maketrans(''.join(normal), ''.join(sub_s))
3019   return x.translate(res)

3020
3021 def gen_data_distribution(CFD, out):
3022   ## Distribucion de los datos:
3023   CFD_array = np.array(CFD)

```

```

3016     out_array = np.array(out)

3017
3018     fig, ax = plt.subplots()
3019     fig.set_size_inches(7, 4)
3020     plt.title('Histogram u{}'.format(get_sub('x')), fontsize = 20)
3021     fig.tight_layout()
3022     fig.subplots_adjust(top=0.95)
3023     n, bins, patches = plt.hist(CFD_array[:,0,:,:].reshape(79*172),
3024                                 200, density=False, facecolor='r',
3025                                 alpha=0.5)
3026     n, bins, patches = plt.hist(out_array[:,0,:,:].reshape(79*172),
3027                                 200, density=False, facecolor='b',
3028                                 alpha=0.5)
3029     plt.xlabel('u{}'.format(get_sub('x')) + ' [m/s]', fontsize =
3030     ↪ 18)
3031     plt.ylabel('Nº of points', fontsize = 18)
3032     plt.rc('xtick', labelsize = 14)
3033     plt.rc('ytick', labelsize = 14)
3034     plt.legend(['CFD', 'CNN'], fontsize = 18)
3035     ax.set_yscale('log')
3036     # ax.set_xscale('log')
3037     plt.grid(True)
3038     # fig.savefig('./ResultadosFinales/data_distrubution_vx_ ' +
3039     ↪ vel_title + '.svg', bbox_inches='tight')
3040     plt.show()

3041
3042     fig, ax = plt.subplots()
3043     fig.set_size_inches(7, 4)
3044     plt.title('Histogram u{}'.format(get_sub('y')), fontsize = 20)
3045     fig.tight_layout()
3046     n, bins, patches = plt.hist(CFD_array[:,1,:,:].reshape(79*172),
3047                                 200, density=False, facecolor='r',
3048                                 alpha=0.5)
3049     n, bins, patches = plt.hist(out_array[:,1,:,:].reshape(79*172),
3050                                 200, density=False, facecolor='b',
3051                                 alpha=0.5)
3052     plt.xlabel('u{}'.format(get_sub('y')) + ' [m/s]', fontsize =
3053     ↪ 18)
3054     # plt.legend(['CFD', 'CNN'])
3055     plt.rc('xtick', labelsize = 14)
3056     plt.rc('ytick', labelsize = 14)
3057     ax.set_yscale('log')

```

```

3055     plt.grid(True)
3056     # fig.savefig('./ResultadosFinales/data_distrubution_vy_ ' +
3057     #             ↳ vel_title + '.svg', bbox_inches='tight')
3058     plt.show()
3059
3060
3061     fig, ax = plt.subplots()
3062     fig.set_size_inches(7, 4)
3063     plt.title('Histogram p', fontsize = 20)
3064     fig.tight_layout()
3065     n, bins, patches = plt.hist(CFD_array[:,2,:,:].reshape(79*172),
3066                                 200, density=False, facecolor='r',
3067                                 alpha=0.5)
3068     n, bins, patches = plt.hist(out_array[:,2,:,:].reshape(79*172),
3069                                 200, density=False, facecolor='b',
3070                                 alpha=0.5)
3071
3072     plt.xlabel('p [Pa]', fontsize = 18)
3073     plt.rc('xtick', labelsize = 14)
3074     plt.rc('ytick', labelsize = 14)
3075     # plt.legend(['CFD', 'CNN'])
3076     plt.grid(True)
3077     ax.set_yscale('log')
3078     # fig.savefig('./ResultadosFinales/data_distrubution_p_ ' +
3079     #             ↳ vel_title + '.svg', bbox_inches='tight')
3080     plt.show()
3081
3082 if __name__ == "__main__":
3083
3084     path1 = "E:/NuevasGeometrias/Resultados_NuevasGeometrias/
3085     ↳ Resultados_first_sample/Modelos/"
3086
3087     torch.manual_seed(0)
3088     lr = 0.001
3089     kernel_size = 3
3090     filters = [8, 16, 16, 32, 32]
3091     bn = False
3092     wn = False
3093     wd = 0.005
3094     beta1 = 0.5
3095     beta2 = 0.5

```

```

3094     model_prueba = UNetEx(2, 3, filters=filters,
3095     ↵   kernel_size=kernel_size, batch_norm=bn, weight_norm=wn)
3096     model_prueba.load_state_dict(torch.load(path1 +
3097     ↵   "Modelo_geometries_109_data_augmentation.py",
3098     ↵   map_location=torch.device('cpu')))
3099     model_prueba.eval()
3100
3101     path2 = "C:/Users/alvar/Desktop/Master/Segundo/TFM/2ª
3102     ↵   parte/Datos para entrenamiento/"
3103
3104     geometry = ["circle", "ellipse", "square", "rectangle",
3105     ↵   "triangle", "triangle_eq"]
3106     # geometry = ["test_ellipse"]
3107     # geometry = ["test_square"]
3108     # size_factor = [0.2, 0.4, 0.6, 0.8, 1.2, 1.4, 1.6, 1.8]
3109     size_factor = [1]
3110     # rotation_angle = [-30, -25, -20, -15, -10, -5, 5, 10, 15, 20,
3111     ↵   25, 30]
3112     rotation_angle = [1]
3113     position = [1, 1]
3114
3115     # sdf = loadmat("SDF_geometrias_data_augmentation/Test/" +
3116     ↵   geometry + "_1size_factor" + str(size_factor) +
3117     ↵   ".mat")['Z_matrix']
3118     # frc = loadmat("FRC_geometrias_data_augmentation/Test/" +
3119     ↵   geometry + "_1size_factor" + str(size_factor) +
3120     ↵   ".mat")['frc']
3121
3122     for geom in geometry:
3123         for size in size_factor:
3124             for rot in rotation_angle:
3125                 sdf =
3126                     loadmat("SDF_geometrias_data_augmentation/Test/" +
3127                     ↵   + geom + "_1size_factor" +
3128                     ↵   str(size) + "rot_angle" + str(rot) +
3129                     ↵   "pos_x" + str(position[0]) +
3130                     ↵   "pos_y" + str(position[1]) +
3131                     ↵   ".mat")['Z_matrix']
3132
3133                 frc =
3134                     loadmat("FRC_geometrias_data_augmentation/Test/" +
3135                     ↵   + geom + "_1size_factor" +

```

```

3119                         str(size) + "rot_angle" + str(rot) +
3120                         ↵   "pos_x" + str(position[0]) +
3121                         "pos_y" + str(position[1]) +
3122                         ↵   ".mat")['frc']

3123
3124     # plt.figure()
3125     # plt.imshow(sdf.T)
3126     # plt.figure()
3127     # plt.imshow(frc.T)

3128     x = np.zeros([1,2,172,79])
3129     x[0,0,:,:] = sdf
3130     x[0,1,:,:] = frc

3131     ti = time.time()
3132     out = model_prueba(torch.FloatTensor(x))
3133     tout = time.time()
3134     duration = tout - ti
3135     # path3 =
3136     ↵   "E:/NuevasGeometrias/Resultados_NuevasGeometrias/
3137     ↵   Resultados_first_sample/test_first_sample/"
3138     # s = 0
3139     # visualize2(out.cpu().detach().numpy(), s, geom,
3140     ↵   rot, size, 0, path3)

3141
3142     # directory = path1 + "tests_first_sample/"
3143     # if not os.path.exists(directory):
3144     #     os.makedirs(directory)

3145     path3 =
3146     ↵   "E:/NuevasGeometrias/Resultados_NuevasGeometrias/
3147     ↵   Resultados_first_sample/tests_first_sample/"

3148     s = 0
3149     visualize2(out.cpu().detach().numpy(), s, geom, rot,
3150     ↵   size, 1, path3)
3151     variable = 'vy'
3152     with open(path3 + "input_data_geometry_" +
3153     ↵   str(geom)

```

```

3149         + "_size_factor_" + str(size) +
3150         ↵ "_rotation_angle_" + str(rot) +
3151         ↵ "_variable_" + variable + ".pkl",
3152         ↵ "wb") as f:
3153     pickle.dump([out, sdf, frc], f)
3154
3155
3156
3157     cfd = pickle.load(open(path3 + "Test geometrias no
3158         ↵ vistas/dataY_367_1001.pkl", "rb"))
3159     cfd=cfd[:,[1,2,0],:,:]
3160     cfd = torch.tensor(np.moveaxis(cfd, [2,3], [3,2]))
3161     error = abs(cfd-out)
3162     visualize(cfd.cpu().detach().numpy(),
3163         ↵ out.cpu().detach().numpy(),
3164         ↵ error.cpu().detach().numpy(), s, "vx", 0,
3165         ↵ "5")
3166
3167     mean_error_vx =
3168         ↵ np.mean(error[:,0,:,:].cpu().detach().numpy())
3169     mean_error_vy =
3170         ↵ np.mean(error[:,1,:,:].cpu().detach().numpy())
3171     mean_error_p =
3172         ↵ np.mean(error[:,2,:,:].cpu().detach().numpy())
3173
3174     mean_value_cfd_vx =
3175         ↵ np.mean(cfd[:,0,:,:].cpu().detach().numpy())
3176     mean_value_cfd_vy =
3177         ↵ np.mean(cfd[:,1,:,:].cpu().detach().numpy())
3178     mean_value_cfd_p =
3179         ↵ np.mean(cfd[:,2,:,:].cpu().detach().numpy())
3180
3181
3182     mean_value_cnn_vx =
3183         ↵ np.mean(out[:,0,:,:].cpu().detach().numpy())
3184     mean_value_cnn_vy =
3185         ↵ np.mean(out[:,1,:,:].cpu().detach().numpy())
3186     mean_value_cnn_p =
3187         ↵ np.mean(out[:,2,:,:].cpu().detach().numpy())
3188
3189
3190     gen_data_distribution(cfd.detach().numpy(),
3191         ↵ out.detach().numpy())

```