HERRAMIENTA PARA LA VISUALIZACIÓN INTERACTIVA DE PROCESO INTERNOS EN REDES NEURONALES ARTIFICIALES

Gerard Guarin Vélez

Resumen— El Deep Learning es un campo con características idóneas para implementar nuevos tipos de representaciones enfocadas en diversos tipos de aprendizaje. Este Trabajo de Fin de Grado explora el campo del Deep Learning, desarrollando representaciones y visualizaciones de los procesos internos de las redes neuronales artificiales con el objetivo de abarcar una mayor variedad de tipos de aprendizaje. Para ello se desarrolla un software web con representaciones dinámicas de procesos típicos en el campo del Deep Learning, incluyendo inicializaciones de pesos, cambios de dimensiones en CNNs, activación de neuronas y ajuste de kernels. Estas visualizaciones, basadas en las características de una representación dinámica, facilitan una comprensión más profunda y práctica de los modelos, siendo valiosas para la educación e investigación en Machine Learning y Deep Learning.

Palabras clave—Deep Learning, Redes Neuronales Artificiales, Representaciones, Aprendizaje Enactivo, CNN, Perceptron full-connected, visualizaciones interactivas.

Abstract— Deep Learning is a field with ideal characteristics for implementing new types of representations focused on various types of learning. This Final Degree Project explores the field of Deep Learning, developing representations and visualizations of the internal processes of artificial neural networks with the aim of covering a greater variety of types of learning. For this, a web software is developed with dynamic representations of typical processes in the field of Deep Learning, including weight initializations, dimension changes in CNNs, activation of neurons and adjustment of kernels. These visualizations, based on the characteristics of a dynamic representation, facilitate a deeper and more practical understanding of the models, being valuable for education and research in Machine Learning and Deep Learning.

Index Terms— Deep Learning, Artificial Neural Networks, Representations, Enactive Learning, CNN, full-connected Perceptron, interactive visualizations.

1 Introducción - Contexto del trabajo

ESTE TFG (Trabajo Final de Grado) surge del deseo de comprender y aprender más sobre el campo de Deep Learning (aprendizaje profundo) es un subcampo del Machine Learning (aprendizaje automático) que utiliza redes neuronales artificiales compuestas por capas de unidades interconectadas para procesar y aprender de grandes cantidades de datos.

Otra de las motivaciones de este trabajo es la necesidad de visualizar el funcionamiento interno de modelos de Machine Learning para mejorar su comprensión. Uno de los mayores problemas a la hora de aprender sobre machine learning es la cantidad de parámetros y variaciones posibles que hay dentro de un modelo. Este problema deriva en una mayor complejidad en la comprensión y el aprendizaje de cómo afectan estas variaciones a un modelo y su resultado.

La visualización del funcionamiento interno de los modelos de machine learning no solo mejora la comprensión teórica, sino que también facilita la optimización y ajuste de estos modelos como se discute en *"Interpretable Machine Learning"[11] de Christoph Molnar* y en los artículos de *Chris Olah et al.[13]* y *Matthew D. Zeiler y Rob Fergus[21]*.

Poder visualizar y comprender los procesos internos de modelos de Deep Learning conlleva la necesidad de creación representaciones de estos.

Crear representaciones no es algo nuevo, se lleva haciendo desde hace siglos, podemos encontrar las primeras representaciones de gráficos en *The Commercial and Political Atla[15]* de Playfair, W. Sin embargo, con campos complejos, como el Deep Learning, se abre un mundo de posibles representaciones distintas a las clásicas basadas en

fórmulas o gráficos estáticos. A su vez, estas representaciones pueden estar enfocadas en distintos tipos de aprendizaje, como plantea Bret Victor en *Humane Representation of Thought: A Trail Map for the 21st Century[20]* con las representaciones dinámicas.

Hay muchas formas de clasificar los tipos de aprendizaje o comprensión, pero si tomamos en cuenta la clasificación de Jerome Bruner, que establece 3 tipos de comprensiones en su libro *Toward a theory of instruction[1]:* simbólica (símbolos, letras, números o palabras en una conversación), icónica (mediante imágenes, gráficos) y enactiva (mediante acción, imitación o manipulación). Las representaciones clásicas suelen enfocarse más en la comprensión simbólica o icónica, dejando de lado la comprensión enactiva.

2 OBJETIVOS

Desarrollar un software de representaciones y visualizaciones dinámicas, sobre diferentes procesos de Deep Learning, que abraquen más tipos de comprensión, como la enactiva, cumpliendo con las características que establece Bret Victor para considerarse dinámicas:

- **Computational**: Capaces de simular procesos.
- **Responsive**: Que respondan a estímulos, en este caso, interacciones.
- **Connected**: Capaces de intercambiar información.

Entre las visualizaciones se incluirán las siguientes:

- **Perceptron neural network**: visualización de los efectos de diferentes inicializaciones de pesos y funciones de activación sobre el aprendizaje en una red neuronal fully-connected o Perceptron.

- **Calculo geometría**: visualización de los cambios de dimensiones que sufre un input a través de diferentes operaciones que se usan típicamente en CNNs.

- **Receptive Field**: visualización del origen del output de una CNN.

- **Neuronas activadas**: visualización de activación de neuronas en una CNN, entrenada con MNIST, producto de un input creado por el usuario.

- **Kernels**: mostrar en tiempo real el ajuste de los kernels en el back propagation de una CNN.

3 ESTADO DEL ARTE

La exploración del arte se basó en la búsqueda de proyectos de visualización de redes neuronales y sus parámetros internos. También se exploró visualizaciones de grafos ya que mediante estos se pueden representar redes neuronales.

Se decidió utilizar Python, siguiendo el estándar de estilo *PEP 8 [14],* frente a otros lenguajes de programación debido a su popularidad en el campo del Deep Learning y a la cantidad de recursos disponibles para este lenguaje.

Como resultado de la exploración del estado del arte se analizaron los siguientes frameworks de python: Streamlit, GraVis, TensorBoard, Plotly, TensorSpace[10]. También analizo y se consideró rediseñar un *modulo de visualización[6]* desarrollado por Jianzheng Liu de manera que al recibir los pesos de una red neuronal se redibujara automáticamente, pero se rechazó debido a que es un módulo pensado para casos de usos poco complejos lo que aumentaría la carga de trabajo en el desarrollo de cada visualización.

Durante la explo*ración del estado del arte se encontraron proyectos como VS2N: Interactive Dynamic Visualization and Analysis Tool for Spiking Neural Networks[4]* y *Exploring Neural Networks with Streamlit: A Visual Learning Tool[18] ,* en los cuales se puede apreciar como los frameworks Plotly, Streamlit y TensorBoard son utilizados para desarrollar visualizaciones basadas en el uso de componentes que modifican los hiperparametros.

Además, existen proyectos como *[12] [9]* donde se usa GraVis para representar grafos. Teniendo en cuenta que una visualización dinámica es capaz de simular procesos, es decir, tiene una característica Computational, el incorporar métodos de

representaciones en forma de grafos es muy útil para la visualización de redes neuronales.

Después de valorar las diferentes opciones se decidió utilizar el framework *Plotly[17]* junto a *Dash[16].* Esta elección se debe a que Plotly y Dash se enfoca en visualizaciones gráficas web, incorpora módulos de representaciones de grafos y permiten la interacción directa sobre gráficos. Además, Plotly incorpora métodos de actualización automática de las visualizaciones, proporcionando la capacidad natural de visualizar los cambios producidos por la interacción del usuario en tiempo real. Estas características son idóneas para que las diferentes visualizaciones que se incluyan en el software cumplan con la característica dinámicas planteada en los objetivos: Computactonal, Responsive y Connected.

Por otro lado, mencionar que se consideró la opción de desarrollar el software como una aplicación web íntegramente, sin el uso de frameworks fronted de python, ya que esto permitiría una mayor personalización y libertad de desarrollo, pero tras analizar la visualización *'The Importance of Effective Initialization' de DeepLearning.AI[3]*, se determinó que el uso de Plotly sería preferible ya que su capacidad para facilitar la implementación de animaciones representa una ventaja significativa.

Durante el desarrollo de la visualización Calculo de Geometría se decidió modificar y adaptar un artefacto creado por Vincent Dumoulin y Francesco Visin[19] para generar animaciones GIF que representan el proceso de aplicación de un kernel al realizar una convolución.

4 PROPUESTA

Con el objetivo de facilitar la comprensión sobre el campo de Deep Learning, se desarrolla un software web con visualizaciones dinámicas.

Este software web tendrá un menú para explorar las diferentes visualizaciones disponibles. Cabe recalcar la intención de desarrollar una base escalable que permita a futuros TFG, o personas con interés en el campo, poder añadir sus visualizaciones e interacciones con diferentes modelos.

4.1 Perceptron neural network

Con esta visualización se podrá evaluar el proceso de aprendizaje de una red neuronal full-connected con diferentes parámetros. Para ello, la visualización permitirá escoger entre diferentes modos de inicialización de pesos y diferentes funciones de activación y mostrará el aprendizaje de una red neuronal full-connected en tiempo real. La arquitectura es dinámica, es decir, el usuario puede definir el número inputs, las capas ocultas y sus neuronas y el número de outputs.

La visualización permitirá seleccionar una neurona de cualquier capa oculta y observar las distribuciones de sus entradas y sus salidas, es decir, antes y después de la función de activación. También, permitirá cambiar el valor de los pesos de una neurona seleccionada.

Además, se podrá ver el gradiente que llega a una neurona seleccionada y la loss por iteración en el aprendizaje de la red neuronal creada.

4.2 Calculo geometría

Cuando se define una *CNN (Convolutional Neural Network)[2]* se debe tener en cuenta el tamaño del activation map resultante de un input ya que muchas veces se suele conectar a una red neuronal full-connected para su clasificación. Por esta razón, es importante ser capaz de comprender y calcular las dimensiones de los activation maps producidas por las convoluciones y otras operaciones típicas en una arquitectura CNN. Este concepto suele ser difícil de comprender únicamente con la formula correspondiente para calcular la resolución resultante de la aplicación de una convolución u otra operación.

Por otro lado, en una CNN se concatenan diferentes operaciones. Esto aumenta la dificultad de conocer las dimensiones del activation map resultante de una CNN y puede producir errores de longitud al vectorizar el activation map y pasarlo a una red neuronal fully-connected la cual espera un vector de entrada de una longitud específica para la clasificación.

A través de esta visualización se permite calcular el tamaño de la imagen de características

resultante de la concatenación de diferentes operaciones típicas en una CNN que alteran las dimensiones.

El usuario podrá escoger las dimensiones del input, las diferentes operaciones que le desea aplicar y el orden en el que lo desea. Las operaciones y sus parámetros se especifican en una tabla dinámica a la cual se le puede añadir filas para aplicar una operación a las salidas de la fila anterior.

Por otro lado, la visualización permite generar una animación donde se observa cómo se aplica un kernel a una input para generar un mapa de características o output. Se podrá generar la animación seleccionando una fila donde se aplique una operación que use kernel.

4.3 Receptive Field

En esta visualización se selecciona un píxel generado por cualquier capa de convolución y se marca sobre la imagen de entrada los pixeles que han sido convolucionados y han dado este píxel como resultado.

4.4 Neuronas activadas

En esta visualización se entrena una CNN con MNIST. El usuario podrá dibujar en un recuadro y la visualización mostrará las neuronas de la CNN resaltando las neuronas más decisivas a hora de clasificar el dibujo de entrada.

4.5 Kernels

La última visualización muestra cómo se ajustan en el backpropagation los kernels de una CNN al ser entrenada con el dataset MNIST similar a la *red neuronal convolucional planteada por Yann LeCun Leon Bottou Yoshua Bengio and Patrick Haffner [8].*

5 METODOLOGÍA

La metodología llevada a cabo es Agile. Esta metodología se basa en definir una serie de tareas (Sprint BackLog) para llevar a cabo durante un periodo de tiempo llamados sprints. Al final del período se hace una reunión de *stakeholders,* en este caso el tutor de TFG y el alumno que lo está llevando a cabo, para revisar el avance logrado durante el sprint, solventar dudas o problemas y si es necesario modificar la duración o el sprint backlog de los siguientes sprints.

La metodología Agile proporciona flexibilidad a la hora de cambiar los requisitos u objetivos de los proyectos donde se aplica y permite cambiar fechas de sprints o tareas de un sprint a otro si es necesario.

5.1 Planificación

El proyecto se dividió en 3 sprints principales: exploración del arte, desarrollo y redacción de este informe.

Durante la exploración del arte, dada la popularidad del uso del lenguaje de programación Python en el campo de machine learning, se exploró principalmente frameworks y sofwares desarrollados en este lenguaje.

El sprint de desarrollo abarca los sub-sprints de desarrollo de cada visualización y de la web app. Para el desarrollo de las visualizaciones se diseñaba e implementaba primero el frontend y a continuación el backend. En los sub-sprints de desarrollo también se solucionaban errores de subsprings anteriores.

6 DESARROLLO

6.1 Perceptron neural network

Como se planteaba en los objetivos, se ha desarrollado una visualización web la cual permite ver en tiempo real el proceso de aprendizaje de una red neuronal Perceptron o full-connected.

De igual manera, siguiendo la planificación primero se diseñó el frontend, *Ilustracion 1*.

Ilustración 1: Diseño inicial de la visualización Perceptron.

Esta visualización está compuesta por 3 scripts de Python:

1. **perceptron_view.py**: contiene la estructura el fornten de esta visualización.
2. **perceptron_data.py**: se encarga de almacenar toda la información necesaria para la visualización. También contiene funciones para dar el formato necesario a los diferentes parámetros de la red neuronal Perceptron de manera que puedan ser dibujados por perceptron_view.
3. **perceptron.py**: contiene la red neuronal full-connected Perceptron creada con Pytorch.

Gracias a los callbacks y dash components de Plotly representa la red neuronal Perceptron y permite interactuar con ella.

Los dash components que forman el layout de esta visualización son los siguientes: button, interval, cytoscape, radio ítems e histogram. El principal dash component es cytoscape ya que es el que representa en forma de grafo nuestra red neuronal Perceptron y es el componente con el que interactúan todos los demás. Por otro lado, gracias al componente Interval se puede realizar las visualizaciones en tiempo real ya que permite llamar a una función callback reiteradamente con un intervalo de tiempo especificado.

Cada dash component en Plotly tienen diferentes parámetros, pero los principales para esta visualización son los parámetros elements y stylesheet de cytoscape. El parámetro elements contiene un diccionario con los nodos y aristas. Los nodos representan las variables de entrada, las neuronas y las salidas del Perceptron. Las aristas representan los pesos asociados a las neuronas. El parámetro stylesheet contiene un diccionario de diccionarios. Estos diccionarios están formados por un elemento selector y un elemento style. El elemento selector funciona con el formato de los selectores en CSS y selecciona que elemento del parámetro elements de cytoscape recibe el estilo del diccionario correspondiente.

Los callbacks son funciones las cuales permiten el intercambio de información entre los componentes de la aplicación web. Además, dentro de los callbacks se obtiene los datos a visualizar de la red neuronal Perceptron y se formatean y guardan en Perceptron_data para poder ser dibujados.

A continuación, los callbacks y sus funciones:

- **button_action**: proporciona la lógica detrás de los botones Start y Stop los cuales permiten iniciar y detener la visualización y el entrenamiento del Perceptron. Se indica que el entrenamiento de la red neuronal está marcha con un borde verde alrededor de la representación en forma de grafo. Si el borde es amarillo, el entrenamiento ha sido detenido pero aún no ha finalizado. Si el borde es rojo, el Perceptron ya ha sido entrenado con todos los batch y la visualización ha finalizado.

- **reset_button**: permite reiniciar la visualización con los parámetros seleccionados previamente.

- **draw_edges**: cuando se llama a este callback, se entrena la red neuronal Perceptron con un batch. A continuación, se obtienen los pesos del Perceptron y se estandarizan y preparan para ser representados. Por otra parte, si se está haciendo el seguimiento de los resultados de las funciones de agregación y activación de alguna neurona, se crean histogramas para observar las distribuciones de los resultados de las dos funciones.

- **update_weights_option**: inicializa y dibuja los pesos del Perceptron con el tipo de inicialización seleccionada por el usuario.

- **neuron_tapped**: permite realizar un seguimiento del resultado de la agregación y la activación en una neurona seleccionada. A través de la creación de histogramas se puede observar las distribuciones de los datos, del último batch utilizado para entrenar, antes y después de la función de activación de la neurona. Además, genera formularios con los pesos de la neurona seleccionada

para que el usuario pueda modificarlos con un valor en específico.

- **button_custom**: dibuja los nuevos pesos modificados por el usuario.

- **steep_button**: permite entrenar al Perceptron de forma controlada. A diferencia del botón Start, el botón Steep realiza 1 entrenamiento con el siguiente batch, en caso de que aún queden batch por utilizar, y se detiene. El botón Start entrenará al Perceptron con todos los bacth disponibles a no ser que se detenga con el botón Stop.

Por último, se ha utilizado la librería Dash Boostrap Components para estructurar la posición de los elementos. Como resultado, se puede observar el frontend final de esta visualización en la *Ilustración 2.*

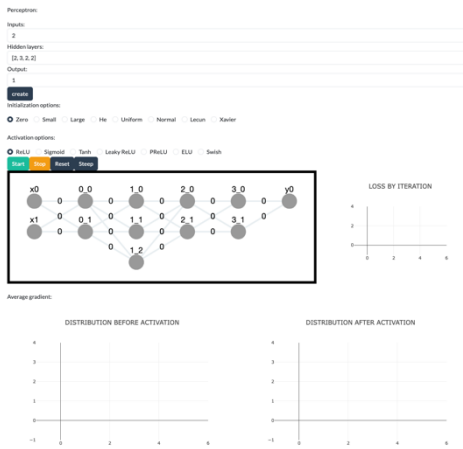

Ilustración 2: Frontend final de la visualización Perceptron.

En la fase del desarrollo del backend se ha implementado el script perceptron_data el cual contiene una clase de Python, PerceptronData, la cual formaliza toda la información del Perceptron. Entre sus funciones se encuentran:

- **buildElements** que construye los nodos, aristas y sus estilos. Los nodos y aristas son guardados en el atributo elements, los pesos de las aristas en el atributo edges_weights y los estilos en el atributo style_sheet. Esta función está diseñada de manera escalable gracias a las constantes PADDING las cuales permiten dibujar perceptrones de diferentes tamaños. Es decir, el código es capaz de dibujar la visualización sin importar si el perceptrón tiene más inputs, hidden layers o outputs.

- **update_style_sheet**: esta función es la que permite actualizar los pesos en la visualización y cambiar la tonalidad de la arista según su peso. Para ello, se obtiene el valor más distanciado de 0, es decir, se observa el peso más grande entre los valores absolutos de los nuevos pesos. Una vez se obtiene el máximo entre los absolutos, se divide este entre el número de colores definidos para visualizar la importancia de un peso. Una tonalidad más rojiza indica un peso más distante a 0 y como consecuencia, un peso más relevante. Con el resultado de la división se crean intervalos para cada tonalidad de rojo. Se detecta en que intervalo está contenido cada nuevo peso y al estilo de la arista correspondiente a este peso se le asigna la tonalidad de rojo pertinente. Además, la función controla si los pesos que se reciben son 0, caso poco probable pero no imposible.

- **generate_dummy_variables**: genera un dataset artificial para entrenar la red neuronal. Las variables X del dataset son generadas aleatoriamente a siguiendo una distribución normal y las variables objetivos Y son el producto de las variables X y pesos generados también aleatoriamente siguiendo una distribución normal.

- **standarize_weights**: se utiliza para formalizar los pesos del Perceptron y poder dibujarlos.

- **get_styles_list**: obtiene una lista con los estilosnecesarios para dibujar el Perceptron en el componente cytoscape.

Por otro lado, se desarrolló el script perceptron.py, el cual contiene es una red neuronal full-conected, creada con Pytorh, con un constructor parametrizado para permitir una inicialización de pesos, cambio dimensiones y selección de función de

activación dinámica. Entre las diferentes inicializaciones de pesos encontramos:

- **He(Kaiming)**: inicializa los pesos de manera que sigan una distribución normal (Gaussian) con una varianza escalada en función del tamaño de la capa anterior.
- **Uniforme**: inicializa los pesos con una distribución uniforme en un rango específico.
- **Normal**: inicializa los pesos con una distribución normal (Gaussian) con media 0 y varianza 1.
- **LeCun:** inicializa los pesos de acuerdo con una distribución normal con una varianza específica que depende del tamaño de la capa anterior.
- **Zero**: inicializa todos los pesos a 0.
- **Xavier:** es similar a la inicialización de LeCun, pero toma en cuenta el número de entradas y el número de salidas (fan-in y fan-out). Esta estrategia ayuda a mantener la varianza de los pesos a medida que se hace la propagación hacia adelante y hacia atrás, lo que puede ayudar a preservar el gradiente y evitar los problemas de desvanecimiento o explosión de gradiente.
- **Small:** inicializa los pesos usando la inicialización de Xavier, pero escalando los valores por un factor pequeño (0.01 en este caso).
- **Large**: inicializa los pesos usando la inicialización de Xavier, pero escalando los valores por un factor grande (100 en este caso).

Entre las opciones de funciones de activación se encuentra:

- **Sigmoid**: suaviza la salida de la nuerona a un rango entre 0 y 1. Puede sufrir del problema del desvanecimiento del gradiente

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + \exp(-x)}$$

- **Tanh (Tangente Hiperbólica)**: centraliza las salidas en torno a cero, lo que puede acelerar el entrenamiento.

$$\text{Tanh}(x) = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

- **ReLu (Rectified Linear Unit):** Puede producir a neuronas muertas, es decir, neuronas la salida de las nueronas siempre es 0 o un valor constante.

$$\text{ReLU}(x) = (x)^+ = \max(0, x)$$

- **Leaky ReLU**: variante de ReLU que incluye un pequeño gradiente para mitigar el problema de las neuronas muertas.

$$\text{LeakyReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases}$$

- **PReLU (Parametric ReLU)**: Similar a Leaky ReLU, pero el gradiente se aprende durante el entrenamiento.

$$\text{PReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax, & \text{otherwise} \end{cases}$$

- **ELU (Exponential Linear Unit)**: Puede tener valores negativos, lo cual permite que los resultados de las activaciones sean cercanos a cero.

$$\text{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha * (\exp(x) - 1), & \text{if } x \leq 0 \end{cases}$$

- **Swish**: Permite valores negativos y no sufre de neuronas muertas. Típicamente, y en este caso, se usa con beta = 1.

$$\text{swish}(x) = x\,\text{sigmoid}(\beta x)$$

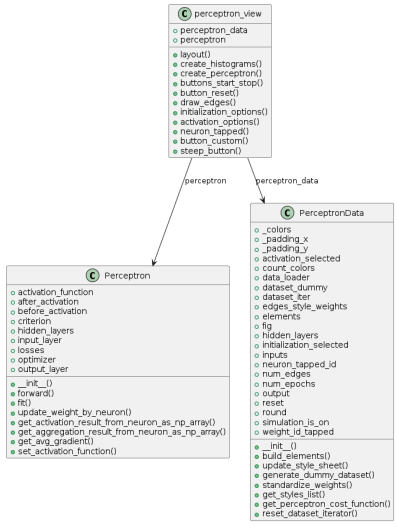

Ilustración 3: Diagrama clases visualización Perceptron.

6.2. Calculo geometría

Esta visualización permite calcular las dimensiones del mapa de activación resultante de una serie de operaciones como las que se podrían encontrar en una CNN. Para ello, se usa una tabla dinámica en la cual cada fila es una operación. Por cada fila se pueden definir los valores de los parámetros que utiliza la fila. Las filas reciben como entrada las dimensiones de la fila anterior.

Durante la implementación de esta visualización se decidió añadir la capacidad de seleccionar una fila con una operación que utilice kernel y generar una animación de extensión gif la cual represente como se aplica el kernel a la entrada y como se obtiene las dimensiones resultantes de es fila. Para esta nueva capacidad se modificó el módulo de generación de animaciones de *Vincent Dumoulin y Francesco Visin[19]* para que acepte parámetros no cuadrados, es decir, permitir la creación de animaciones con kernels rectangulares, paddings diferentes en el eje x e y junto a un stride y dilatación de diferente dimensión en cada eje. Por otro lado, también se modificó integrarse con el backend de esta visualización

Para la fase del frontend, se planteó el diseño de la *ilustración 4*.

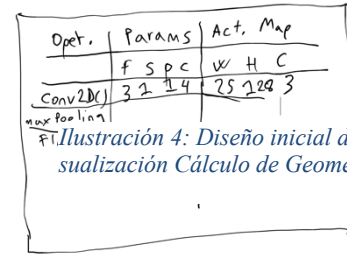

Ilustración 4: Diseño inicial de la visualización Cálculo de Geometría.

Ilustración 4:Diseño visualizacion CNN Geometry.

Una vez con el diseño inicial se procedió a implementar el script cnn_geometry.py para el frontend.

Para el layout de esta visualización se utiliza el dash components DataTable junto a elementos inputs y botones. Por otro lado, para implementar lógica a los elementos se utilizan los siguientes callbacks:

- **add_row**: implementa la lógica del botón con símbolo "+" el cual se utiliza para añadir una fila más en la tabla dinámica.
- **generate_animation**: esta función obtiene la animación del backend y la codifica en ASCII para mostrarla en el forntend.
- **compute_button**: esta función se encarga de calcular las dimensiones de cada activation map tras aplicar la operación seleccionada a la entrada o salida de la fila anterior en la tabla dinámica.

Además de los callbacks se ha desarrollado una función llamada checks_for_kernel_operations la cual recibe una fila de la tabla dinámica y controla que los parámetros utilizados para la operación cumplan con lo siguiente: el tamaño del kernel no puede ser 0 en ninguno de los ejes, el stride no puede ser 0, el stride no puede ser mayor a 1 en ninguno de los ejes si el tamaño del kernel no es mayor a 1 en las 2 dimensiones, x e y. En caso de que no se incumpla alguna de las condiciones se

muestra una alerta en la web app notificando la razón.

Las operaciones incluidas en esta visualización están basadas en funciones de Pytorch:

- **Conv2d:** La operación conv2d aplica un kernel a la entrada para extraer características mediante convoluciones. Para el cálculo de las dimensiones de salida utiliza los parámetros: stride, padding, dilation, output_channels.

- **Pooling:** Pooling representa operaciones cómo MaxPooling o AvgPooling. Utiliza los parámetros: stride, padding y ceil-mode.

- **Flatten:** La operación convierte una entrada multidimensional en un vector unidimensional. No utiliza ningún parámetro.

- **UpSampling2D:** aumenta las dimensiones de la imagen duplicando los píxeles según el parámetro factor_scale.

- **Conv2DTranspose:** realiza lo contrario de la convolución. Aumentar la dimensión espacial de la imagen y utiliza los parámetros: stride, padding, dilation, output_padding y output_channels.

Los parametros son los siguientes:

- **Stride**: define el número de píxeles por los cuales un kernel se mueve sobre la imagen de entrada en cada paso.

- **Padding:** agrega pixels o unidades alrededor de la entrada antes de aplicar un kernel.

- **Dilation:** insertar espacios entre los elementos o unidades del kernel.

- **Ceil mode:** define si redondea un número decimal hacia arriba o abajo.

- **Output Channels:** define el número de filtros aplicados durante la convolución, lo que determina la profundidad del mapa de características de salida.

- **Output Padding:** se utiliza en operaciones de convolución transpuesta para controlar las dimensiones espaciales de la salida. Agrega relleno a las dimensiones de salida para que coincidan con un tamaño deseado.

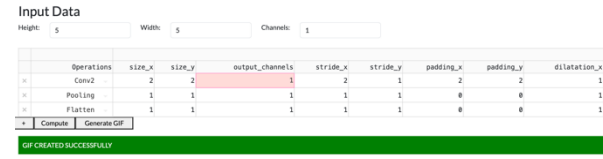

Ilustración 5: frontend de visualización Cálculo de Geometría.

En la fase del desarrollo del backend se adaptaron los scrips generate_gif.py y produce_figure.py de *Vincent Dumoulin y Francesco Visin[19]* para que funciones con tamaños de 2 dimensiones, x e y. También se modificó la plantilla arithmetic_figure.txt para que acepte parámetros de 2 dimensiones a la hora de especificar el padding, dilatation, kernel_size y output_size.
El proceso de generación de animaciones de extensión gif se basa en generar archivos latex, por cada frame de la animación final. Los frames son creados a partir de la plantilla arithmetic_figure.txt, con los parámetros correspondientes sustituidos, y la plantilla unit.txt la cual se utiliza para crear los cuadrados que representan cada unidad de entrada del input. Una vez se han creado los archivos latex, se transforman se crea un subproceso el cual ejecuta la comanda latex pdflatexpara convertir los archivos latex a formato pdf y los guarda en una carpeta llamada pdf. A continuación, se crea otro subproceso el cual usa la comanda convert de *ImageMagick[5]* para convertir los archivos pdf a formato png y guardarlos en una carpeta llamada png. Por último, se utiliza la comanda *gifsicle[7]* para crear la animación de formato gif con cada frame en formato png.

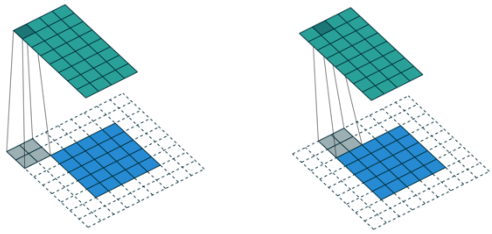

Ilustración 6: Frames creados para la visualización de la aplicacion de una convolución con kernel de tamaño 2x2 con stride 2 en el eje X a un input de tamaño 5x5 con 2x2 de padding. Esta operación genera una salida de 4x8.

Por otro lado, también se implementó la función empty_folders_png_pdf_gif para vaciar el contenido de las carpetas png, pdf y gif cada vez que se cree una animación.

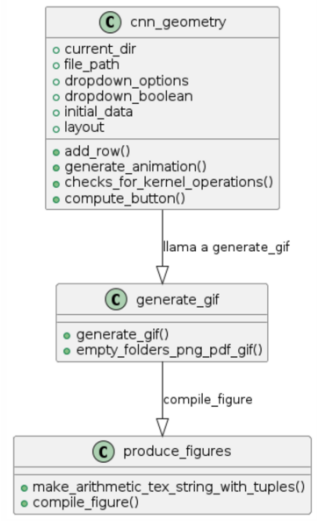

Ilustración 7: diagrama de clases de la visualización cnn_geometry.

6.3. App web

La app web que contiene las visualizaciones ha sido desarrollada en un script llamado app.py. Este script aprovecha la programación con paginación de Plotly. El frontend de las visualizaciones han sido registradas como una página con la función register_page de Plotly. El script app.py incorpora un botón que despliega un side-board donde se puede escoger entre todas las visualizaciones desarrolladas.

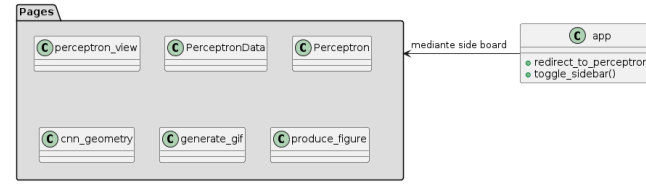

Ilustración 8: diagrama de clases de la app web.

7 RESULTADOS

Gracias a la visualización Perceptron, se puede observar cómo afectan la inicialización de los pesos. Una buena inicialización por norma general produce una salida con distribución normal, en la función de activación, como el caso de la inicialización Xavier, ilustración 9.

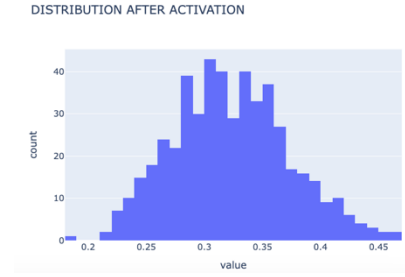

Ilustración 9: salida de funcion de activacion sigmoid con inicializacion Xavier en red neuronal con 10 inputs, 2 hidden layers de 10 neuronas y 1 salida.

Por otro lado, una inicialización con pesos muy grandes puede generar:

- **Sigmoid**: salida concentrada cerca de 0 o 1. Produce que las derivadas de la función sigmoid sean muy pequeñas, puede producir un desvanecimiento de gradientes
- **Tanh**: salida concentrada cerca de -1 o 1. Esto puede causar de igual manera a la sigmoid un problema de un posible desvanecimiento de gradientes.
- **ReLU**: salida mayormente positiva y muy dispersa. Esto provoca que los gradientes durante el backpropagation también sean grandes, lo que puede causar una explosión de gradientes

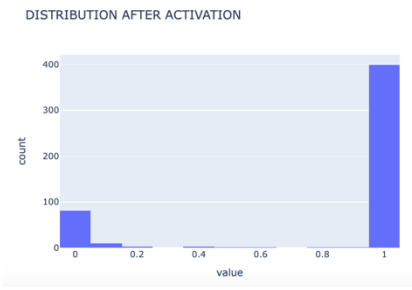

Ilustración 10: distribución de la salida de una función de activación sigmoid con inicializacion de pesos large en una red neuronal con 10 inputs, 2 hidden layers de 10 neuronas y 1 salida.

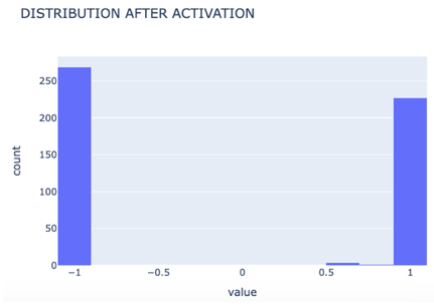

Ilustración 11: distribución de la salida de una función de activación tanh con inicializacion de pesos large en una red neuronal con 10 inputs, 2 hidden layers de 10 neuronas y 1 salida.

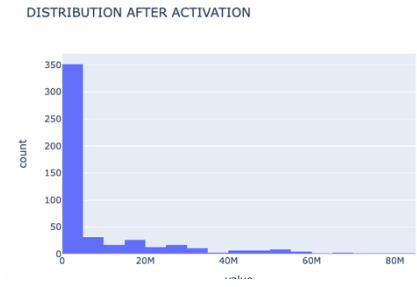

Ilustración 12: distribución de la salida de una función de activación ReLU con inicializacion de pesos large en una red neuronal con 10 inputs, 2 hidden layers de 10 neuronas y 1 salida.

Una inicialización de pesos pequeños genera:

- **ReLU**: salida puede ser muy pequeña y concentrada alrededor de cero. Los gradientes serán pequeños, lo que puede llevar a un entrenamiento muy lento.
- **Sigmoid**: salida concentrada alrededor de 0.5, donde la función tiene un comportamiento similar a una función lineal lo que puede llevar a limitar la capacidad del modelo para aprender patrones no lineales.
- **Tanh**: salida concentrada alrededor de 0. Esto produce un comportamiento muy similar al lineal y genera el mismo problema que en el caso de la sigmoid.

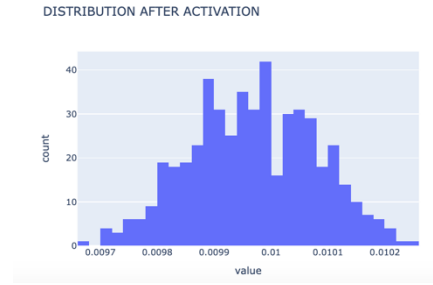

Ilustración 13: distribución de la salida de una función de activación ReLU con inicializacion de pesos small en una red neuronal con 10 inputs, 2 hidden layers de 10 neuronas y 1 salida.

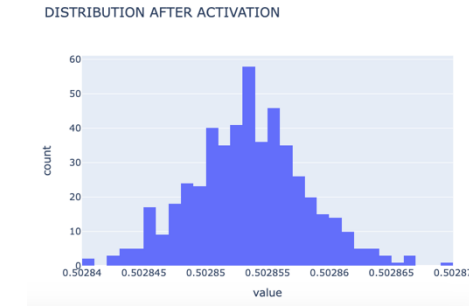

Ilustración 14: distribución de la salida de una función de activación sigmoid con inicializacion de pesos small en una red neuronal con 10 inputs, 2 hidden layers de 10 neuronas y 1 salida.

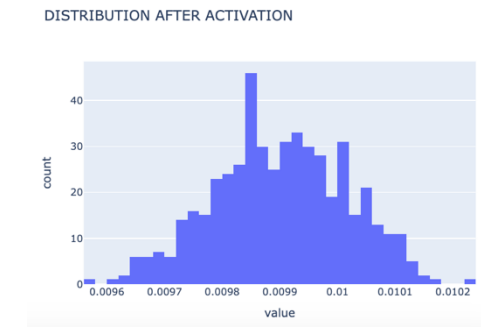

Ilustración 15: distribución de la salida de una función de activación Tanh con inicializacion de pesos small en una red neuronal con 10 inputs, 2 hidden layers de 10 neuronas y 1 salida.

Con la visualización del Cálculo de Geometría, se pueden observar las dimensiones que tendría el output después de cada operación que modifica la dimensión del input en toda una arquitectura CNN. Esto se logra simplemente especificando las operaciones y sus parámetros, lo cual es muy útil para comprender cómo afecta cada operación al input de una CNN específica que deseemos

diseñar. Además, gracias a la generación de animaciones, se puede mostrar claramente el porqué de las dimensiones resultantes de una operación con kernel.

8 CONCLUSIONES

Se han implementado casi la mitad de las visualizaciones planteadas junto la app web que las contiene. Además, las visualizaciones incorporadas cumplen con el enfoque de representaciones dinámicas:

- Computacionales: representan o simulan procesos reales en el campo del Deep Learning.
- Responsive: las representaciones cambian con la interacción del usuario, no son estáticas.
- Connected: gracia a las funciones callback de Plotly, existe un intercambio de información entre las representaciones, como el caso de la visualización del Perceptron donde al darle click a uno neurona los histogramas cambian y representan las distribuciones de sus funciones de agregación y activación.

Gracias a sus cualidades dinámicas, el software facilita distintos tipos de aprendizajes, como el enactivo, con la manipulación de las representaciones, y el aprendizaje icónico, con la representación gráfica en tiempo real del Perceptron o las animaciones de la aplicación de una operación con kernel a un input en una CNN.

La app web tiene la capacidad de ser ampliada gracias a utilizar frameworks como Dash junto a Plotly los cuales permiten la incorporación de nuevas visualizaciones como paginas a una web.

Aún que es cierto que estos frameworks facilitan el desarrollo web de las visualizaciones, cabe recalcar que pueden llegar a estar limitados a la hora de incorporar visualizaciones con otros frameworks. Por otro lado, como mejoras, destacaría:

- El desarrollo de un mejor estilo de interfaz de usuario, por ejemplo, mediante la incorporación de un CSS personalizado, lo cual es posible con Plotly.
- La incorporación de visualizaciones 3D, por ejemplo, en el caso de las CNN mostrar cómo se realizan operaciones como convoluciones 3D a distintos inputs.
- En el caso de la generación de animaciones en la visualización Cálculo Geometría, un indicador de carga de la animación

REFERENCIAS

[1] Bruner, J. (1974). *Toward a theory of instruction*. Harvard University Press.

[2] Convolutional neural network. (s/f). Wikipedia The Free Encyclopedia.

[3] Daniel Kunin, K. K. (2019). *Initializing neural networks*. DeepLearning.AI.

[4] Elbez, H., Benhaoua, M. K., Devienne, P., & Boulet, P. (2021). VS2N: Interactive Dynamic Visualization and Analysis Tool for Spiking Neural Networks. En *2021 International Conference on Content-Based Multimedia Indexing (CBMI)* (pp. 1-6). Lille, Francia. doi: 10.1109/CBMI50038.2021.9461916

[5] *ImageMagick – Mastering Digital Image Alchemy*. (n.d.). ImageMagick.

[6] jzliu-. (s/f). *Visualize neural network with or without weights*. GitHub.

[7] *kohler/gifsicle: Create, manipulate, and optimize GIF images and animations*. (n.d.). GitHub.

[8] Le Cun, Y., Bottou, L., Bengio, Y., & Haner, P. (s/f). Gradient-Based learning applied to document recognition. Stanford.edu. Recuperado el 10 de marzo de 2024.

[9] Lee, B. (s/f). An Interactive Visualisation for Your Graph Neural Network Explanations. Read Medium.

[10] Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T. Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, Pieter-Jan Kindermans. (13 de agosto de 2018). *iNNvestigate neural networks*!

[11] Molnar, C. (2020). *Interpretable Machine Learning*.

[12] Nowak, J., Eng, R. C., Matz, T., et al. (2021). A network-based framework for shape analysis enables accurate characterization of leaf epidermal cells. *Nat Commun*, 12, 458.

[13] Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., & Mordvintsev, A. (2018). *The building blocks of interpretability*. Distill, 3(3), e10.

[14] PEP 8 – Style guide for Python code | peps.python.org. (n.d.). *Python Enhancement Proposals (PEPs)*.

[15] Playfair, W. (1801). *The Commercial and Political Atlas: Representing, by Means of Stained Copper-plate Charts, the Progress of the Commerce, Revenues, Expenditure and Debts of England During the Whole of the Eighteenth Century*. T. Burton.

[16] *plotly/dash: Data Apps & Dashboards for Python. No JavaScript Required.* (n.d.). GitHub.

[17] *plotly/plotly.py: The interactive graphing library for Python This project now includes Plotly Express!* (n.d.). GitHub.

[18] Shi, A. A. K. (22 de octubre de 2023). *Exploring Neural Networks with Streamlit: A Visual Learning Tool*. Python in Plain English.

[19] *vdumoulin/conv_arithmetic: A technical report on convolution arithmetic in the context of deep learning*. (n.d.). GitHub.

[20] Victor, B. (2014). *Humane representation of thought: A trail map for the 21st century*. En *Proceedings of the Companion Publication of the 2014 ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity* (p. 5). Association for Computing Machinery.

[21] Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. En *Computer Vision – ECCV 2014* (pp. 818-833). Springer.