UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE QUINTANA ROO

DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

RECONSTRUCCIÓN ESTOCÁSTICA DE LA CAPA CATALÍTICA DE UNA CELDA DE COMBUSTIBLE

TESIS

PARA OBTENER EL GRADO DE

INGENIERO EN REDES

PRESENTA

Juan Carlos Domínguez Martínez

DIRECTOR DE TESIS

DR. JAIME SILVERIO ORTEGÓN AGUILAR

ASESORES

DR. GLISERIO ROMELI BARBOSA POOL

DR. ABIMAEL RODRIGUEZ SÁNCHEZ

DR. JAVIER VÁZQUEZ CASTILLO

DR. VICTOR MANUEL SÁNCHEZ HUERTA

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO

ÁREA DE TITULACIÓN

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO

DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

CHETUMAL QUINTANA ROO, MÉXICO, ABRIL DE 2022

UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE QUINTANA ROO

DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

TESIS TITULADA

"RECONSTRUCCIÓN ESTOCÁSTICA DE LA CAPA CATALÍTICA DE UNA CELDA DE COMBUSTIBLE"

ELABORADA POR

Juan Carlos Domínguez Martínez

BAJO SUPERVISIÓN DEL COMITÉ DEL PROGRAMA DE LICENCIATURA Y APROBADO COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE:

INGENIERO EN REDES

COMITÉ DE TESIS

DIRECTOR: _____

DR. JAIME SILVERIO ORTEGÓN AGUILAR

ASESOR: _____

DR. GLISERIO ROMELI BARBOSA POOL

ASESOR: _____

DR. ABIMAEL RODRIGUEZ SÁNCHEZ

ASESOR SUPLENTE: _____

DR. JAVIER VÁZQUEZ CASTILLO

ASESOR SUPLENTE: _____

DR. VICTOR MANUEL SÁNCHEZ HUERTA

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO
DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

CHETUMAL QUINTANA ROO, MÉXICO, ABRIL DE 2022

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO
ÁREA DE TITULACIÓN

CONTENIDO

RESUMEN

En los últimos años, la comunidad científica ha dirigido su interés a sistemas energéticos con enfoques renovables, en los que el hidrógeno se ha ido posicionando como un prospecto de combustible alternativo. Así es como la tecnología de celdas de combustible de membrana de intercambio de protones (PEMFC), por sus características de implementación, se proyecta a desempeñar un rol relevante tanto para la industria como para la sociedad del futuro.

Desde un punto de vista microestructural, debido a la composición de la capa catalítica en la PEMFC, puede ser catalogada como un material heterogéneo estocástico (MHE). Esta tesis forma parte de un proyecto de investigación de la Universidad Autónoma de Quintana Roo donde se propone el desarrollo e implementación de técnicas experimentales de manufactura y técnicas numéricas de simulación para determinar los coeficientes efectivos de transporte en electrodos de celdas de combustible.

En este trabajo, se presenta una propuesta de software inédito para usar modelos profundos de aprendizaje automático en la generación sintética de microestructuras estadísticamente similares a imágenes de microscopio tomadas de la capa catalítica de electrodos de celdas de combustible de intercambio protónico reales. Para ello se exponen los detalles de la metodología empleada para la obtención de datos y su posterior tratamiento, así como las herramientas de software utilizadas para el diseño e implementación del software propuesto y, finalmente, presentar un análisis de los resultados describiendo las aplicaciones del estudio buscando mejorar los sistemas energéticos basados en hidrógeno.

INTRODUCCIÓN

“Education isn’t something you can finish.” — Issac Asimov

Las necesidades energéticas de la era digital moderna presentan un gran reto para la sociedad. Con un paradigma de consumo dependiente en la explotación de fuentes combustibles fósiles no renovables, se ha derivado en problemáticas de carácter tanto ambiental como económico, político y social. Por ello, en las últimas décadas la comunidad científica ha sumado esfuerzos para incrementar el interés en la investigación y el desarrollo de sistemas energéticos con enfoques renovables y de características que representen un menor impacto para una situación cada vez más crítica en la conversación ambiental. Bajo este escenario, el hidrógeno se ha ido posicionando como un prospecto de combustible alternativo.

Su alta eficiencia y el gran potencial de uso en una amplia variedad de aplicaciones han posicionado al hidrógeno como una fuente alternativa de combustible ideal. Posee una alta densidad energética que puede ser producida y procesada sin liberar gases de efecto invernadero, ya que su obtención es mediante la disociación electrolítica del agua, se almacena y posteriormente se recupera mediante reacciones químicas entre el hidrógeno y el oxígeno. Este proceso ocurre en la estructura interna de una celda de combustible, generando electricidad, calor y agua como únicos productos en dichas reacciones. Así es como la tecnología de celdas de combustible de membrana de intercambio de protones (PEMFC), por sus características de implementación, se proyecta a desempeñar un rol relevante tanto para la industria como para la sociedad del futuro.

Desde un punto de vista microestructural, debido a la composición de la capa catalítica en la PEMFC puede ser catalogada como un material heterogéneo estocástico (MHE). Esto se debe a que sus fenómenos internos son significativamente afectados por las propiedades intrínsecas de las fases o elementos que la componen. Por ello, mediante el coeficiente de proporcionalidad o coeficiente efectivo de transporte (CET), que caracteriza todo el dominio del material, se puede realizar una evaluación de la eficiencia energética del MHE estudiado. Dicho coeficiente puede ser calculado mediante diferentes relaciones matemáticas. Los MHE están presentes en un amplio y diverso dominio de importantes aplicaciones ingenieriles, como pueden ser fibras, baterías, suelos, concretos y también en electrodos de celdas de combustible. A su vez, también son variados y abundantes entre los materiales orgánicos, como en huesos, sangre, madera, y demás tejidos o bioestructuras.

Esta tesis forma parte de un proyecto de investigación de la Universidad Autónoma de Quintana Roo donde se propone el desarrollo e implementación de técnicas experimentales de manufactura y técnicas numéricas de simulación para determinar los coeficientes efectivos de transporte en electrodos de celdas de combustible. Las técnicas y metodologías computacionales para la obtención de CETs en MHEs siguen un desarrollo muy específico según sus objetivos, por lo que en la literatura se describen aplicaciones de software hechas a la medida o *“in-house developed”*. Partiendo de esta idea, y contribuyendo a la

interdisciplinariedad, se desarrolló una aplicación en *Python* donde se evalúan las capacidades de modelos de aprendizaje automático para generar imágenes sintéticas intrínsecamente similares a las imágenes de microscopio obtenidas de celdas de combustible reales.

Este proyecto aún sigue en desarrollo, ya que se pretende obtener un software modular con opciones para configurar modelos, tipos de materiales u objetivos específicos para cada estudio, generando así para la comunidad científica una herramienta multifuncional para el análisis de materiales.

Antecedentes

En los últimos años, bajo la iniciativa del equipo de investigadores en la División de Ciencias, Ingeniería y Tecnología (DCIT) de la universidad, se ha contribuido y colaborado para el desarrollo de sistemas energéticos con enfoques renovables, aportando propuestas en metodologías de análisis tanto teóricos como prácticos, así como de manufactura industrial, basadas en tecnologías de hidrógeno como fuente principal de energía, específicamente, desarrollando prototipos de celdas de combustible [1]–[4].

Dichas metodologías han requerido un profundo trabajo de investigación conjunta en ciencias de materiales, sistemas energéticos y computación con lo que se han logrado una variedad de aportes y resultados, entre publicaciones, congresos y premios, reafirmando la relevancia del proyecto y confirmando la urgencia de un cambio en el paradigma energético.

Diversos proyectos han sido desarrollados en esta línea de investigación, principalmente; 1) CONACYT Redes Telemáticas 243725 periodo 2015-2015, 2) CONACYT Ciencia Básica 221988 periodo 2014-2017, 3) CONACYT SENER 254667 periodo 2017-2022. 4) CÁTEDRAS CONACYT 209 periodo 2016-2026.

En relación a los coeficientes efectivos de transporte, se han estudiado distintos parámetros, como la relación entre las resoluciones en las imágenes estudiadas [5], [6], así como la distribución geométrica de sus fases [7], [8]. También se han hecho propuestas y mejoras a las metodologías de estudio mediante análisis y optimizaciones, como con el *recocido simulado* en la etapa de reconstrucción [9]. Particularmente, también se propusieron máquinas de vector soporte (SVMs) como herramientas de clasificación automática para las fases en las imágenes fuente [10].

Inspirados por la idea de usar modelos automáticos en la optimización del proyecto, en esta tesis se proponen modelos de aprendizaje profundo como herramientas que permitan la obtención de muestras sintéticas similares a las reales con menor consumo de recursos de procesamiento. En este proyecto se presentan un par de modelos para esto: un modelo clasificador y otro generativo [11]. Dichos modelos son propuestas recientes en el área del aprendizaje profundo [12], en las que se han definido múltiples arquitecturas de implementación dependiendo de la aplicación que se requiera resolver por cada modelo [13].

Justificación

Con este trabajo se busca aportar una alternativa metodológica al estudio especializado en ciencia de los materiales realizado por el equipo de investigación: La simulación estocástica de imágenes provenientes del análisis microscópico de materiales heterogéneos estocásticos; desarrollando una herramienta de cómputo enfocada principalmente a procesar fotos de microscopio electrónico de barrido (SEM) tomadas de electrodos electroquímicos basados en carbón.

La metodología clásica para generar representaciones sintéticas digitales de las imágenes SEM presenta un problema de optimización que es tradicionalmente resuelto mediante métodos iterativos, como el *recocido simulado*. Sin embargo, estos procesos tienden a requerir altos tiempos de procesamiento en función tanto de los parámetros implementados, como del tamaño y la resolución de las imágenes, la complejidad de las funciones a optimizar y las capacidades del hardware para llevar a cabo el procesamiento. Esta tesis presenta el desarrollo e implementación de modelos profundos de generación automática como propuesta alternativa a dicho proceso.

La simulación computacional de las microestructuras, y el posterior estudio de los fenómenos de transporte que ocurren en ellas, permitirá al grupo de investigadores evaluar distintos coeficientes de proporcionalidad del material, lo que será más práctico y económico en comparación a realizar pruebas físicas directamente sobre los electrodos construidos. Con esto se busca producir prototipos de celdas más eficientes, previendo los CETs calculados en las simulaciones, así como corroborar las propiedades de los dispositivos ya desarrollados.

El software propuesto se presenta como una oportunidad de escalamiento a futuras posibilidades de estudio microestructural, no limitándose a ser un instrumento de estudio de celdas de combustible únicamente. Posteriores implementaciones de algoritmos o procesos de optimización, e incluso la utilización de librerías especializadas, son recomendables.

Objetivo General

Desarrollar un programa de cómputo para generar imágenes sintéticas estadísticamente similares, considerando la extracción de características intrínsecas en imágenes experimentales, validando mediante funciones de densidad probabilística, y haciendo uso de modelos de aprendizaje profundo.

Objetivos Particulares

- Desarrollar el software propuesto en *Python*
- Hacer uso de algoritmos de aprendizaje profundo
- Entrenar modelos de aprendizaje profundo y guardarlos para su reutilización
- Generar imágenes sintéticas representativas de las imágenes SEM de la capa catalítica de una PEMFC

Alcance

- El programa será desarrollado en una aplicación *Jupyter Notebook* con *Python 3*
- La librería de aprendizaje utilizada será *Keras*, con backend de *TensorFlow*
- Se usará *Google Colab* como plataforma de ejecución
- Las imágenes fuente son tomas de microscopio electrónico de barrido almacenadas en un directorio colaborativo de *Google Drive*

Estructura de la Tesis

Se presentó una introducción junto a la justificación del proyecto definiendo los objetivos y alcances del mismo. Posteriormente, a lo largo del capítulo siguiente se detallan los fundamentos teóricos de la metodología estudiada para el análisis de materiales, siguiendo con los fundamentos para la generación sintética mediante modelos de aprendizaje automático. En el segundo capítulo se hace una descripción detallada de las características del software: lenguaje, librerías, métodos, modelos e implementación. Los resultados obtenidos se presentan en el último capítulo, y finalmente se incluyen las conclusiones y el trabajo a futuro. También se presentan los códigos fuente principales a manera de apéndices.

CAPÍTULO 1: MARCO TEÓRICO

A continuación, se presentan los conceptos y técnicas que dieron origen al desarrollo de este proyecto. Primeramente, se describe la teoría energética de la que parte la investigación inicial. Posteriormente, se presentan los fundamentos del aprendizaje automático junto a las características principales de los modelos de aprendizaje relevantes para el desarrollo de la solución de software propuesta.

Fundamentos Energéticos

La metodología para el estudio de la capa catalítica en celdas de combustible de intercambio protónico ha sido una línea particular de desarrollo para el equipo de investigación de la DCIT en la Universidad Autónoma de Quintana Roo. El trabajo realizado busca extenderse al estudio de materiales con características similares a las celdas de combustible, también llamados materiales heterogéneos estocásticos.

Las etapas de desarrollo de dicha metodología son:

1. *Identificación heurística de la fase de estudio*: entendiendo como “fase” al elemento representativo de la imagen, por ejemplo: poros, metales en una matriz polimérica, esferas en una matriz dispersa, por mencionar algunos. Esta identificación permite separar la fase de estudio del resto de la imagen.
2. *Cuantificación estadística de la distribución estocástica*: la parametrización permite, entre otras cosas, identificar patrones en las imágenes, para determinar o cuantificar la diferencia estadística entre la secuencia de muestras.
3. *Reconstrucción estocástica*: consiste en generar una representación tridimensional de la imagen en una malla digital de volúmenes de control.
4. *Solución numérica de la ecuación de continuidad en la microestructura*: el análisis numérico permite determinar coeficientes efectivos de transporte y visualizar la continuidad de la materia mediante un análisis físico.

Los principios para el análisis de eficiencia en las celdas se encuentran fundamentados en funciones de correlación estadística, técnicas de optimización y simulación numérica mediante la solución de volúmenes de control finito [3], [14].

Celdas De Combustible (PEMFC)

Una celda de combustible es un dispositivo que es capaz de generar energía eléctrica mediante una reacción química interna inducida por las características físicas de sus componentes. Existe una variedad de celdas de combustible en desarrollo, entre las que destacan las de membrana de intercambio protónico, o PEMFC por sus siglas en inglés: *Proton Exchange Membrane Fuel Cell*.

Como se puede apreciar en la Fig. 1, una PEMFC está constituida por los siguientes componentes: placas bipolares (BP), capa difusora de gas (GDL), capa catalítica (CL) y membrana de intercambio protónico (PEM); donde se puede identificar que esta última es el componente más crítico para las reacciones que dan funcionamiento a la celda [1], [15], [16].

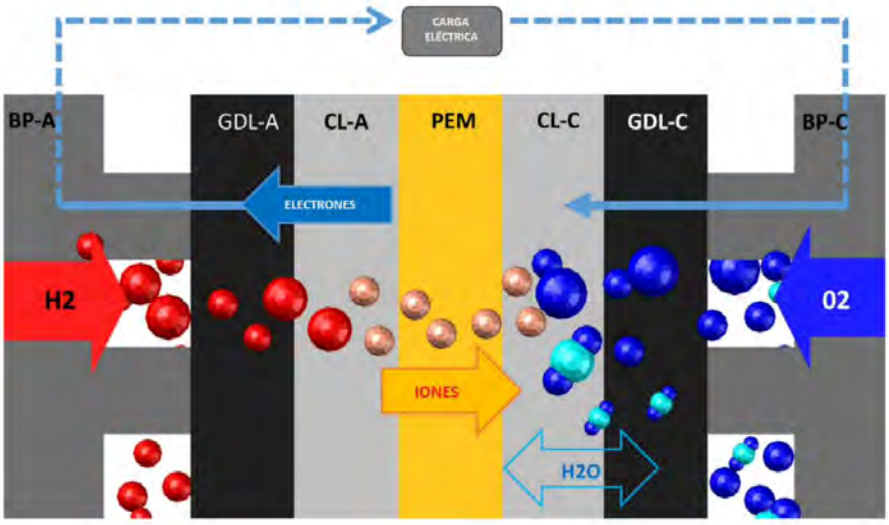

Figura 1. Modelo de una PEMFC.

Este tipo de celdas presentan las siguientes ventajas en su aplicación: baja temperatura de operación, alta eficiencia energética, un diseño compacto ideal para aplicaciones móviles y un funcionamiento libre de residuos contaminantes. Sin embargo, como tecnología aún en desarrollo, también presenta una serie de retos técnicos, como la reducción del costo de manufactura y el incremento en durabilidad y desempeño, que permitan una comercialización masiva.

Materiales Heterogéneos Estocásticos (MHE)

Un material heterogéneo puede ser definido como un medio que a niveles “macroscópicos” se puede estudiar como un solo elemento, pero a niveles “microscópicos” está constituido por distintas “fases”. En este sentido, entendemos como “fase” a un dominio identificable con propiedades particulares del resto de las otras fases que componen el medio (i.e. vacío y diversos materiales: sólidos, gaseosos o líquidos). Dichas fases afectan el comportamiento del material en sus fenómenos de transporte, es decir, su distribución define el coeficiente efectivo de transporte del material. Cuando las fases que componen al material heterogéneo no están ordenadas estos materiales suelen llamarse “Materiales Heterogéneos Estocásticos”.

En la Fig. 2 se presentan los distintos elementos, o fases, que componen la capa catalítica en una PEMFC, donde se aprecia su composición multifase de distribución aleatoria, definiéndose como un MHE [17].

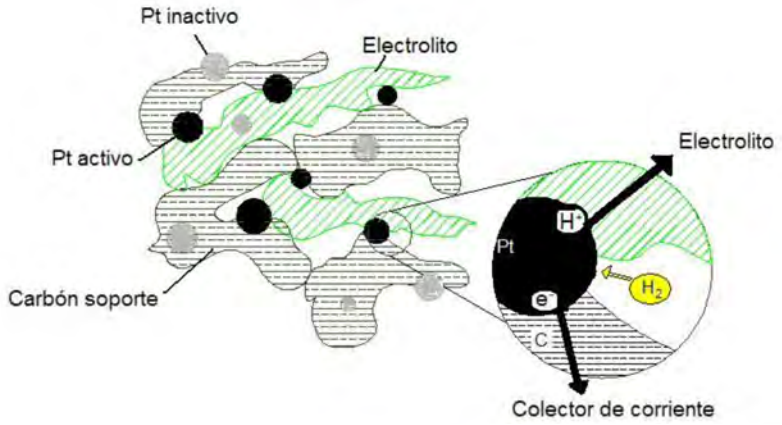

Figura 2. Esquema de las fases de la capa catalítica de una PEMFC.

Coeficientes Efectivos De Transporte (CET)

Las propiedades macroscópicas de un MHE son significativamente afectadas por las propiedades intrínsecas de sus fases, las fracciones volumétricas de su composición y la estructura de los elementos que lo componen. De esta forma, podemos definir al CET de un MHE como el coeficiente de proporcionalidad que caracteriza todo el dominio del mismo. El valor correcto del CET es indispensable para diseñar o dimensionar dispositivos y procesos energéticos.

La determinación de los CETs en un MHE digitalizado sintéticamente puede ser realizada mediante promedios de campos locales derivados de las teorías de transporte para el problema concerniente. Específicamente, cualquiera de las propiedades efectivas denotadas como conducción, sea eléctrica, protónica o de calor, son definidas por una relación lineal entre el promedio de un flujo local generalizado y el promedio de un potencial local o aplicado generalizado. En la Fig. 3 se ejemplifican distintos fenómenos de transporte relevantes para el funcionamiento de una PEMFC a través de la composición estructural de la capa catalítica.

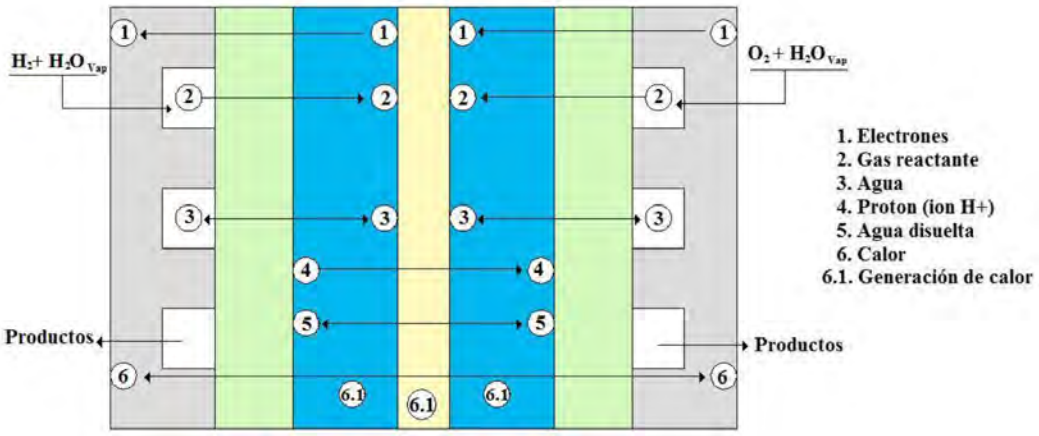

Figura 3. Mecanismos de transporte en una PEMFC.

Microscopio Electrónico De Barrido (SEM)

El microscopio electrónico de barrido es una herramienta utilizada para la observación de superficies de una manera detallada y meticulosa. El proceso de obtención de imágenes consiste en colocar sobre la muestra un haz focalizado de electrones que se interrelacionan con los átomos del material. El detector recoge información sobre los electrones secundarios que se han producido a raíz de la interacción inelástica de los electrones primarios (haz) con los electrones de las bandas de valencia o de conducción dependiendo del tipo de material.

En la Fig. 4 se presentan un par de muestras de las imágenes SEM procesadas, donde la primera es una imagen en escala de grises obtenida directamente del SEM y la segunda presenta un proceso de edición simétrica para ser evaluada mediante software.

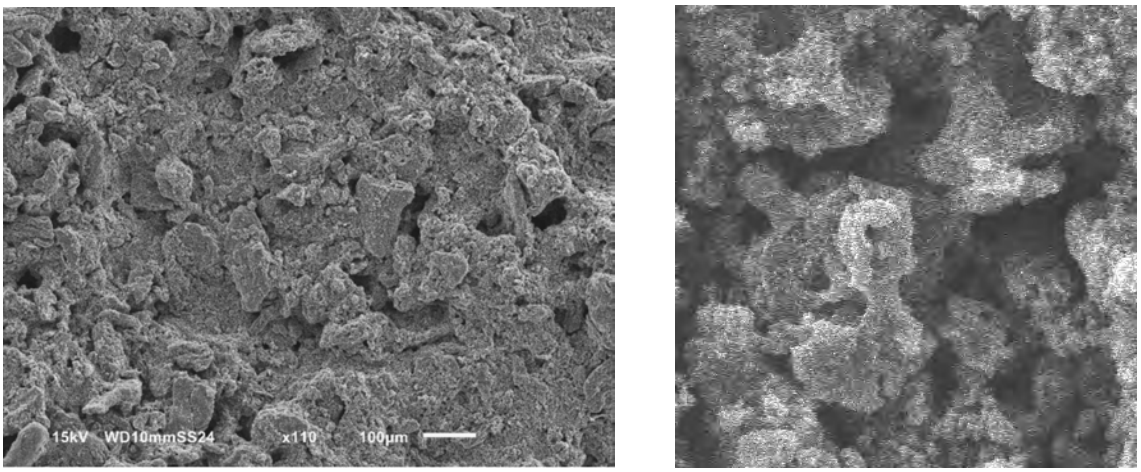

Figura 4. Imágenes SEM de una PEMFC.

Reconstrucción Estocástica

La reconstrucción de microestructuras en 3D de un material heterogéneo estocástico a partir de fotomicrografías en 2D es una técnica que ha sido desarrollada extensamente en las últimas décadas [9], [17]–[22]. Estos métodos son comúnmente referidos como “métodos de reconstrucción estocástica”.

En la Fig. 5 se describe un algoritmo general de reconstrucción, donde a partir de una imagen 2D se producen muestras volumétricas con distribución aleatoria de fases, pero de misma proporción, para posteriormente permutar pixeles hasta optimizar las funciones de correlación objetivo.

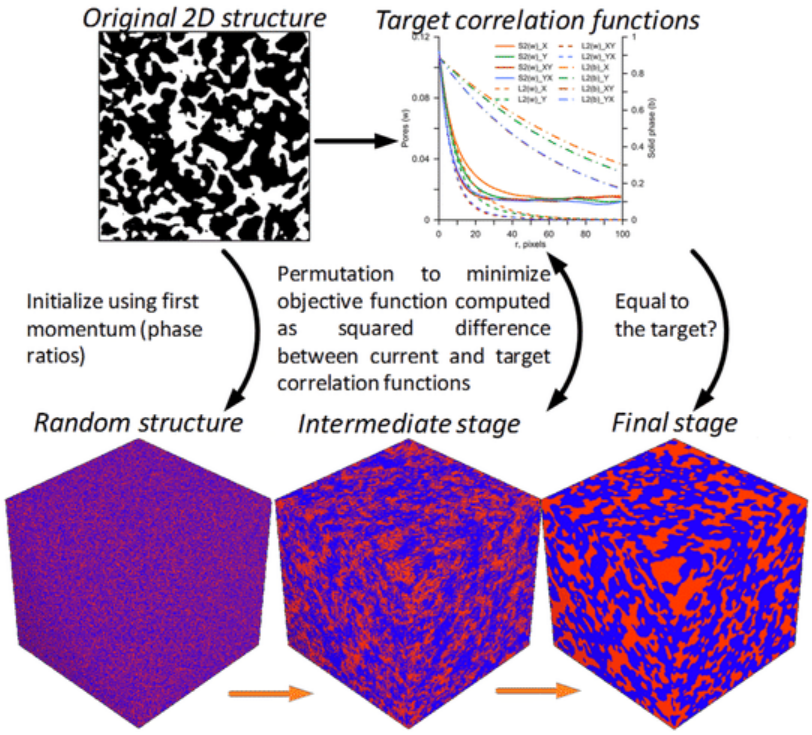

Figura 5. Diagrama de reconstrucción estocástica.

Funciones De Correlación

Los materiales heterogéneos estocásticos presentan un reto para el desarrollo de una teoría científica que permita describir con detalle su microestructura. Los avances recientes se han basado en distintas funciones de correlación estadísticas, como pueden ser fracciones volumétricas de las fases, cuantificación del área superficial o interfacial, orientación, distribuciones de tamaño, conectividad de las fases, etc. [18], [21].

Las funciones, aparte de tener valor descriptivo para el material estudiado, son tomadas como referencia para una posterior etapa de reconstrucción, permitiendo tener control de las características para las simulaciones sintéticas de materiales. En la Fig. 6 se muestra un ejemplo de caracterización por correlación.

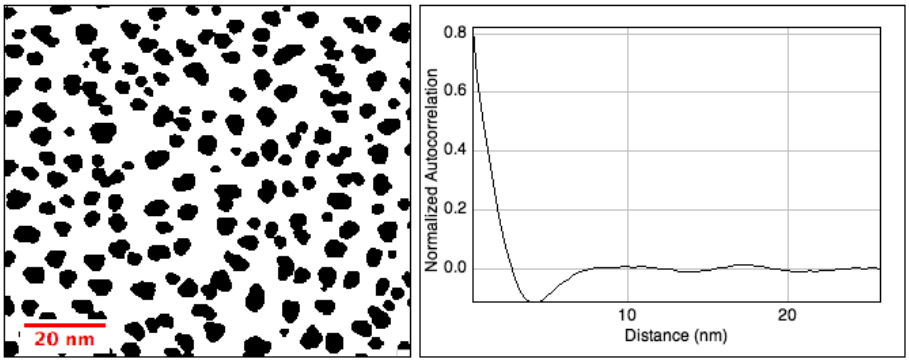

Figura 6. Función de correlación.

Volúmenes De Control Finitos

Finalmente, la técnica numérica utilizada para simular fenómenos de transporte de un material es conocida como método de volúmenes de control finitos, donde se evalúan ecuaciones de transporte clásicas y mediante procesos iterativos se resuelven potenciales locales para finalmente generar un promedio general [23], así se obtiene un diagrama como el de la Fig. 7.

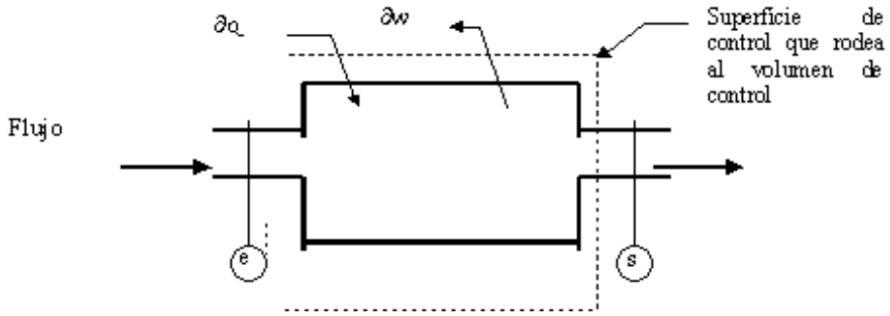

Figura 7. Diagrama de Volumen de control.

Aprendizaje Automático

Como alternativa de implementación a los algoritmos iterativos se propone el uso de modelos generativos de aprendizaje automático. Estos “generadores” se obtienen al realizar entrenamientos supervisados mediante imágenes SEM reales. Una vez entrenados, los modelos son capaces de producir de manera semicontrolada nuevas muestras sintéticas de materiales estocásticamente similares a las imágenes fuente.

Bajo el amplio contexto de la Inteligencia Artificial (*IA*), el aprendizaje automático, o *Machine Learning*, ha sido un área con gran desarrollo e impacto en la era moderna de las tecnologías de la información. Su estudio consiste en un conjunto de métodos computacionales que permiten a sistemas entrenarse con información y datos generalmente no procesados, y aprender o describir de manera automática las representaciones necesarias para el objetivo particular a resolver. En la Fig. 8 se muestra una comparativa entre el paradigma clásico de programación y el aplicado en el aprendizaje automático.

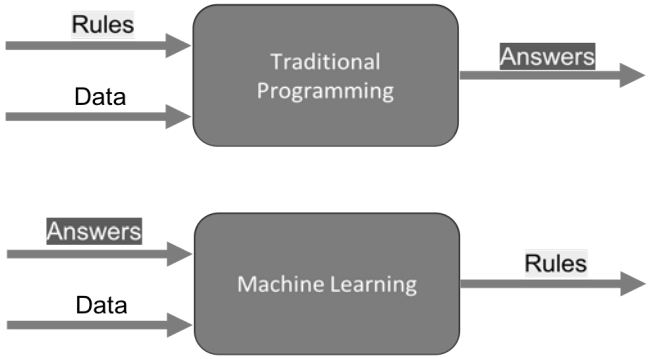

Figura 8. Paradigmas de programación.

Mientras la metodología tradicional presenta algoritmos y reglas para procesar la información y obtener siempre los mismos resultados, con *machine learning* se plantea alimentar modelos tanto con información de entrada como con las salidas esperadas, etiquetas, u otra retroalimentación, permitiendo al modelo "aprender" sus propias reglas de operación [24].

Bajo este nuevo paradigma, y mediante los modelos generadores propuestos, se busca sustituir el método tradicional de reconstrucción por optimización de funciones de correlación que, aunque se han demostrado sus capacidades en la literatura, suelen resultar insuficientes para una descripción estadística relevante, y que además requieren de un alto nivel de procesamiento para la producción de cada muestra. En cambio, con los modelos de aprendizaje profundo se pueden extraer características intrínsecas de las muestras a distintos niveles de abstracción según las capas de procesamiento utilizadas, y una vez entrenados con suficiente información representativa, la producción de muestras puede ser realizada de manera relativamente inmediata y con mínimos procesos adicionales de optimización.

Aprendizaje Profundo

El aprendizaje profundo, o *Deep Learning*, se refiere al desarrollo de modelos computacionales compuestos por múltiples capas de procesamiento. La "profundidad" de un modelo se refiere a la cantidad de capas que contiene. Esto permite describir estructuras altamente complejas o con grandes compilaciones de información y, mediante retropropagación o modificando los parámetros internos de la estructura del modelo, procesar la representación en cada capa a partir de la anterior [13]**.**

En cuanto al entrenamiento de los modelos de aprendizaje automático, se describen tres metodologías principales:

- *Supervisado*: Se refiere a los entrenamientos con parámetros de control disponibles para el proceso de aprendizaje.
- *No Supervisado*: Son modelos de naturaleza intuitiva que infieren la mejor ruta de aprendizaje sin ayuda de información para su validación.
- *Reforzado*: Este tipo de modelo se adapta a los cambios en el ambiente al que esté expuesto.

Este trabajo se enfoca al aprendizaje supervisado, por lo que cada una de las secciones siguientes se refieren a modelos especializados en dicha metodología.

Redes Neuronales

En el aprendizaje profundo se logran múltiples niveles de representación al combinar módulos, o capas, que transforman la información desde niveles muy simples hasta niveles más complejos [12]. El modelo básico en el aprendizaje profundo es la red neuronal y su arquitectura está constituida por un conjunto de módulos de procesamiento que pueden o no ser entrenables y que están apilados secuencialmente. Dichos módulos son llamados capas de procesamiento.

En la Fig. 9 se presenta un ejemplo sencillo de una arquitectura secuencial para una red neuronal, con una capa inicial de entrada que alimenta las capas ocultas internas de procesamiento, de las cuales la salida final es procesada en una capa de salida para de ella obtener un resultado. De manera similar a los MHEs, la estructura y/o configuración de los módulos internos de la red definen la funcionalidad de la misma.

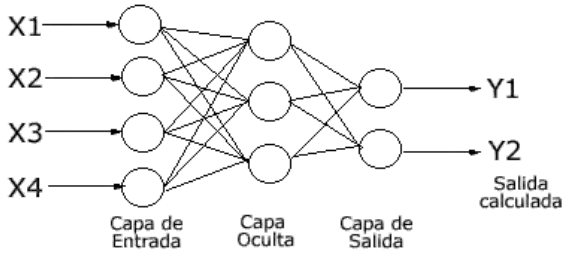

Figura 9. Diagrama de una Red Neuronal.

Entre las arquitecturas principales para redes neuronales secuenciales, según sus objetivos, se pueden mencionar los siguientes modelos:

- *Regresión*: Su objetivo es predecir un valor numérico según datos de entrada tras un entrenamiento de datos generalmente históricos.
- *Clasificación*: Buscan identificar la clase a la que pertenece el dato de entrada a partir de un entrenamiento sobre muestras etiquetadas con las clases a seleccionar.
- *Generación*: Se busca producir muestras sintéticas similares a la información de entrenamiento a partir de entradas controladas.

El proceso general de entrenamiento para una red neuronal consiste en ir actualizando los *pesos* dentro de cada capa entrenable mediante un *optimizador de aprendizaje* [25] a partir del error o *función de pérdida* configurada en relación a la diferencia entre el valor de salida o *predicción* contra el valor esperado durante un proceso iterativo o *épocas*. En la Fig. 10 se muestra una representación básica de una red neuronal y sus propiedades principales de funcionamiento.

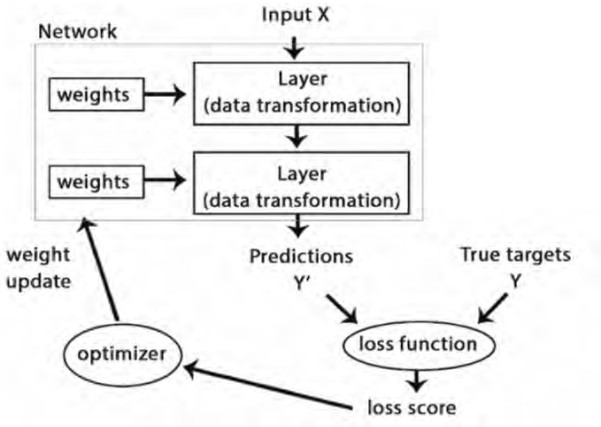

Figura 10. Entrenamiento de una Red Neuronal.

Perceptron

Como se ha mencionado, las redes neuronales están constituidas por capas, pero estas a su vez están definidas por una o más unidades de procesamiento, también llamadas neuronas. El modelo más básico de esta unidad se conoce como *perceptrón*.

En el diagrama de la Fig.11 se muestran los datos de entrada (*X*) que alimentan el perceptrón, los pesos (*W*) asignados a dichas entradas y su procesamiento, que consiste, en este caso, en realizar una sumatoria del producto de las entradas y los pesos, para posteriormente procesar el resultado mediante una función de activación, y así generar finalmente una valor de salida, que a su vez puede o no alimentar otros perceptrones, capas o modelos.

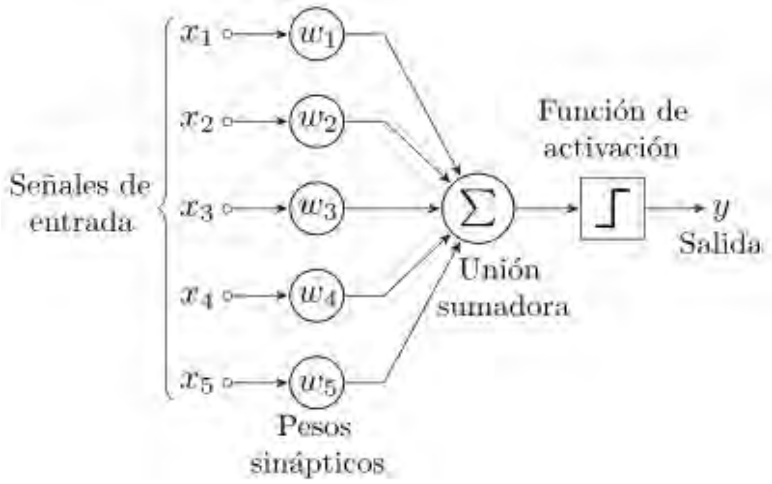

Figura 11. Diagrama del Perceptrón.

Capas

El acoplamiento de múltiples capas permite el diseño de redes profundas de aprendizaje complejas. Cada una procesa mapeos no lineales de múltiples entradas y salidas, transformando su propia entrada y modificando los pesos asignados para incrementar tanto la selectividad como la invarianza de la representación.

Entre las principales capas de procesamiento se pueden mencionar:

- *Capas Densas (Sencilla, Múltiple)*: Perceptrones básicos en cadena.
- *Capas Convolucionales (Normal, Transpuesta)*: Ideales para el procesamiento de matrices, aplican operaciones de convolución con matrices de perfiles predeterminados y son usadas en el procesamiento de señales e imágenes.
- *Capas Recursivas (LSTM, GRU)*: Cuentan con capacidad de memoria de estado debido a su naturaleza recursiva y son ampliamente utilizadas en el procesamiento de contexto y lenguaje natural.

Redes Generativas Adversarias (GAN)

Una red GAN (*Generative Adversarial Network*) es un moderno modelo de aprendizaje profundo generativo que al finalizar con su entrenamiento es capaz de generar nueva información sintética [24]**.** Otro tipo de modelo generativo es el Codificador Automático Variacional, o *VAE* por sus siglas en inglés, sin embargo, por ser de entrenamiento no supervisado se optó por no profundizar en este modelo, aunque presenta una arquitectura muy similar a la de una red GAN.

Por su parte, la estructura de una GAN se compone generalmente por dos modelos: un *generador* que captura la distribución de las muestras y un *discriminador* que calcula la probabilidad de que una muestra sea real o no. El entrenamiento del generador consiste en maximizar la probabilidad de que el discriminador cometa un error. Este algoritmo corresponde a un duelo minimax en teoría del juego [11].

En la Fig. 12 se presenta un diagrama del entrenamiento de una red GAN, el cual consiste en generar imágenes sintéticas con el *generador* a partir de ruido controlado y alimentarlas al *discriminador* para que las clasifique. Este proceso sigue una secuencia iterativa en la que el objetivo es lograr que el *generador* produzca imágenes que el *discriminador* entrenado sea incapaz de distinguir entre reales o falsas.

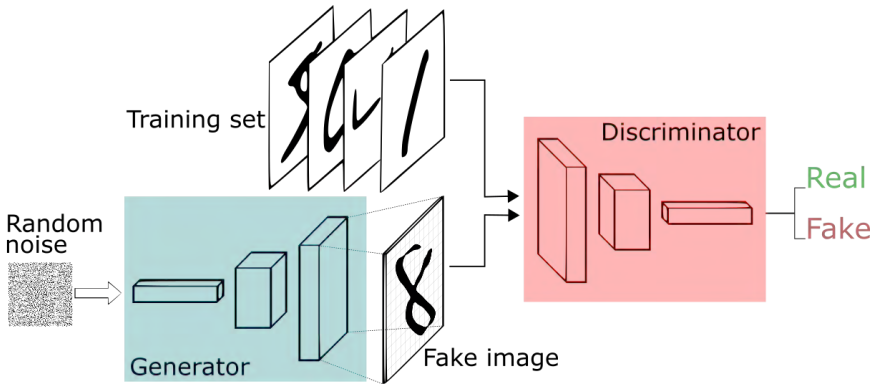

Figura 12. Modelo de una red GAN.

DCGAN

Las redes GAN son una reciente propuesta para el procesamiento de imágenes, donde las estructuras de sus elementos internos están basadas principalmente en el acoplamiento de múltiples capas convolucionales, por lo que son comúnmente referidas como DCGANs, o *Deep Convolutional Generative Adversarial Networks* [26]. Estas redes han demostrado ser muy poderosas y prometedoras para la generación de imágenes, muchas veces, indistinguibles entre reales y sintéticas.

A continuación, en la Fig. 13 se presentan algunos ejemplos de resultados mediante generación sintética con DCGANs.

Figura 13. Ejemplos de muestras producidas por DCGANs.

En una DCGAN encontramos principalmente capas convolucionales, las cuales permiten analizar los píxeles de las imágenes o en su caso generarlos. Sin embargo, estas capas también requieren procesamientos adicionales, como la normalización de las salidas mediante funciones de activación (*Tanh*, *Sigmoid*, *ReLU*, *LeakyReLU*, etc.), la reducción de dimensiones, también conocida como *pooling*, o la recuperación de bordes o *padding*. También se deben considerar las capas de entrada y salida para cada modelo, donde se definen las dimensiones y características de los datos de entrada y los resultados esperados. En la Fig. 14 se presenta un diagrama de una estructura DCGAN más detallada.

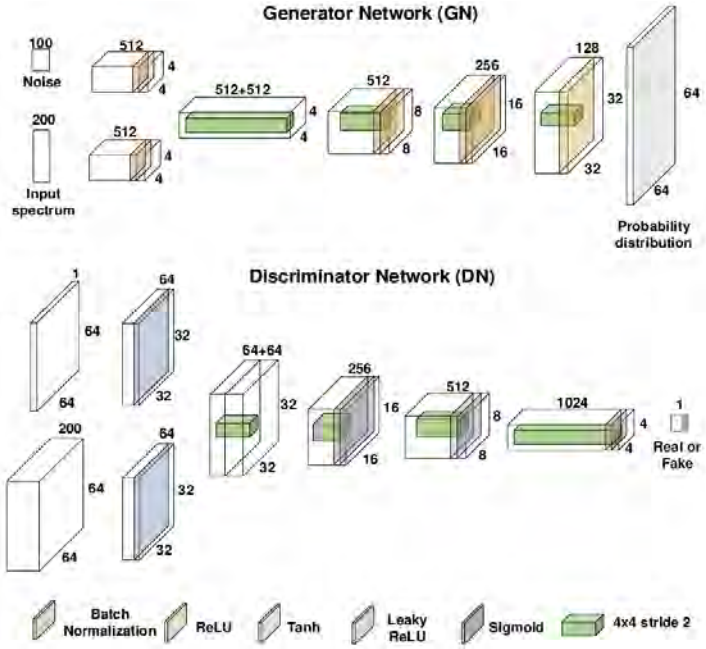

Figura 14. Arquitectura DCGAN.

Convoluciones

Una capa convolucional aplica la operación *convolución* mediante distintos filtros o *kernels* a las imágenes de entrada para resaltar sus características intrínsecas. Dicha operación procesa cada imagen con una máscara del mismo tamaño que el filtro y va recorriendo la imagen para que en cada una de las posiciones se realice un producto punto del filtro contra el contenido de la imagen enmascarada. También se utiliza la operación inversa a la convolución, que es la convolución transpuesta, también llamada *deconvolución*.

La Fig. 15 demuestra una operación de convolución donde la imagen de entrada es procesada por un filtro que permite resaltar las características de la imagen. Se debe tener en cuenta que durante esta operación la imagen resultante pierde el borde externo, debido al tamaño del kernel, aunque existen técnicas para recuperar parcialmente esta información, como la del *padding*.

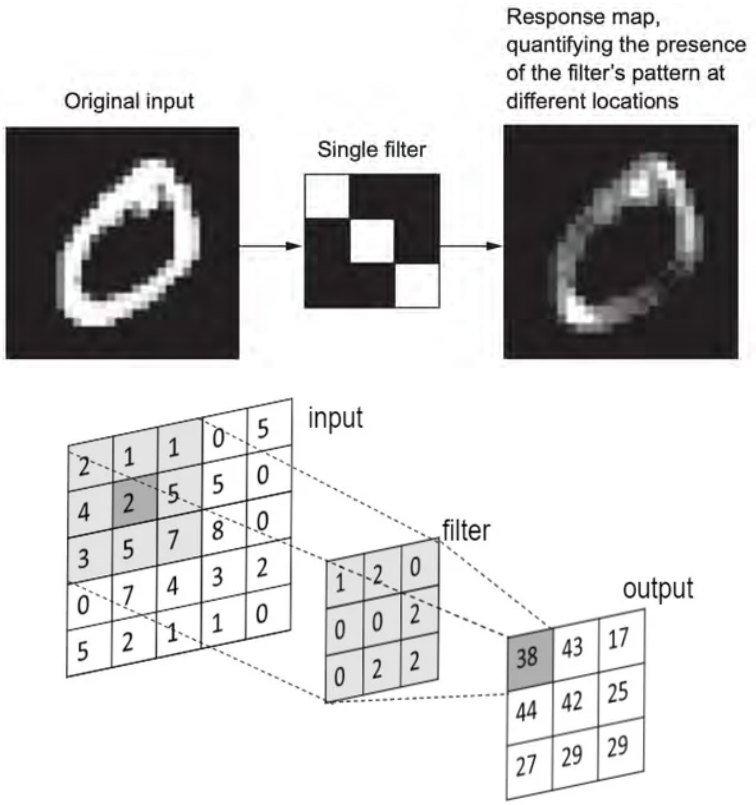

Figura 15. Operación "*Convolución*".

Con el proceso de filtrado mediante múltiples kernels, las capas convolucionales pueden resaltar diversas características específicas en la imagen procesada, así es como al acoplar múltiples capas se obtienen descripciones a diversos niveles de abstracción para las cualidades en la imagen. Como se puede observar en la Fig. 16, con cada elemento representativo que en conjunto caracteriza una etiqueta "*cat*".

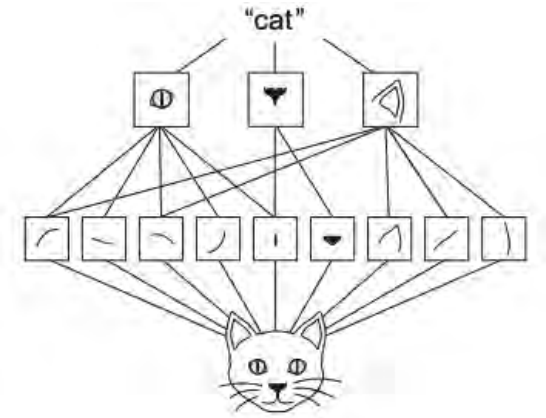

Figura 16. Abstracción convolucional.

Discriminador

En la arquitectura GAN, el *discriminador* es básicamente un modelo clasificatorio binario que describe la probabilidad de que la imagen procesada sea real o falsa. El diseño básico para un discriminador se presenta en la Fig. 17.

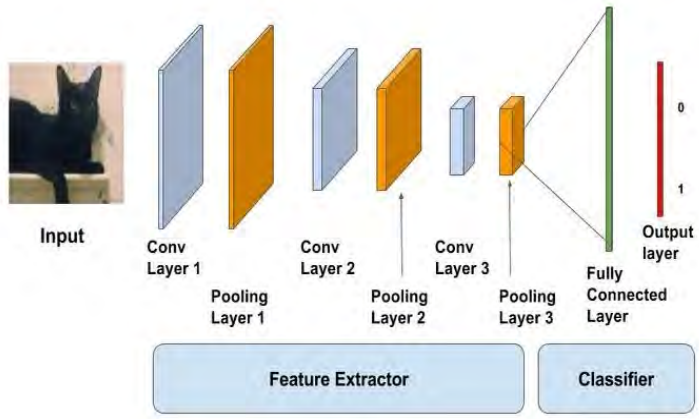

Figura 17. Discriminador.

El *discriminador* está compuesto por capas convolucionales que extraen las características intrínsecas de la imagen. Cada una es cada vez más pequeña, es decir, tiene dimensiones inferiores a la previa, lo que les permite obtener características más detalladas en cada paso. Esto se logra mediante el redimensionamiento de las salidas en cada capa, ya sea por *pooling* o *strides* internos, para finalmente generar una salida para un arreglo de capas densas o totalmente conectadas, donde se realiza la clasificación.

Generador

El modelo *generador* es el encargado de producir imágenes sintéticas durante y después del entrenamiento de la red GAN. Este modelo también está compuesto por diversas capas convolucionales que a partir de un vector de ruido semicontrolado son capaces de generar imágenes que eventualmente logren engañar al *discriminador*.

En el diagrama siguiente (Fig. 18) se presenta la estructura del *generador*, que, como el *discriminador*, también está basado en capas convolucionales, sin embargo, estas son de tipo transpuestas y poseen un orden de proceso con inicio lineal, mediante el vector de ruido de entrada, y por redimensionamiento convolucional se logra la dimensión de salida deseada para la imagen a generar. Estas capas también requieren procesos de activación, normalización y escalamiento similares al discriminador.

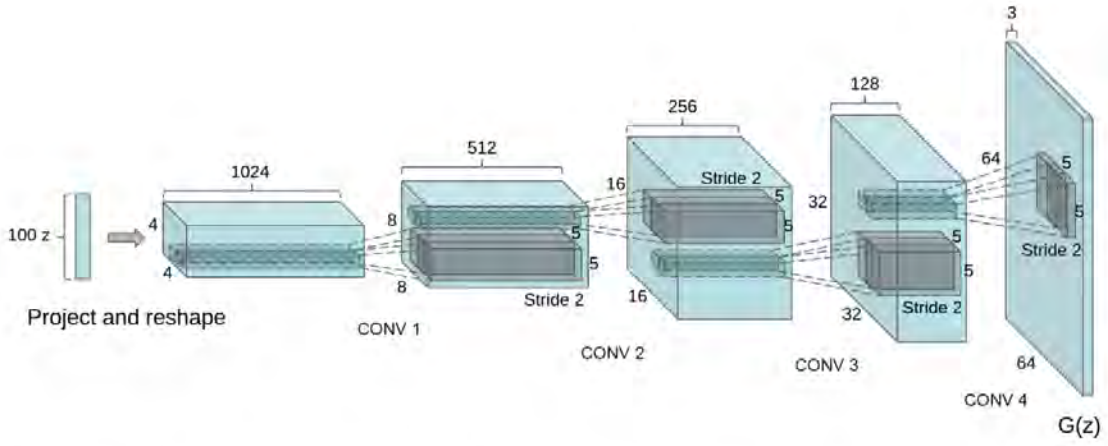

Figura 18. Generador.

CAPÍTULO 2: METODOLOGÍA

En este capítulo se hace una descripción de las tecnologías empleadas para el desarrollo del software propuesto. También se definen las herramientas de implementación y se abordan las características del diseño metodológico.

Herramientas de Software

Se desarrolló una aplicación de software en Python donde se modeló la metodología expuesta y se portó de tal forma que desde un navegador web se puedan evaluar cada una de las etapas del estudio. Y, a través de una interfaz de control, se presentan los métodos de procesamiento para las imágenes SEM y su posterior análisis mediante algoritmos de generación sintética. A continuación se presentan las principales tecnologías utilizadas para el procesamiento.

Lenguaje de programación (Python)

Python es un lenguaje de programación de código abierto de ejecución interpretada y de sintaxis expresiva, cuya simplicidad es comparable a la de un pseudocódigo. Su sencillez y legibilidad permiten desarrollar mediante metodología funcional u orientada a objetos indistintamente, y con un intérprete interactivo es posible probar código en tiempo real y sin necesidad de compilar. También es multiplataforma, lo cual permite portabilidad para virtualmente cualquier aplicación y al ser de código abierto cuenta con una gran comunidad de usuarios que aportan activamente a las capacidades del lenguaje mediante el desarrollo de librerías especializadas.

Como se puede apreciar en la Fig. 20, estas particularidades han hecho de Python el lenguaje recomendado por la comunidad científica en los últimos años [27], [28].

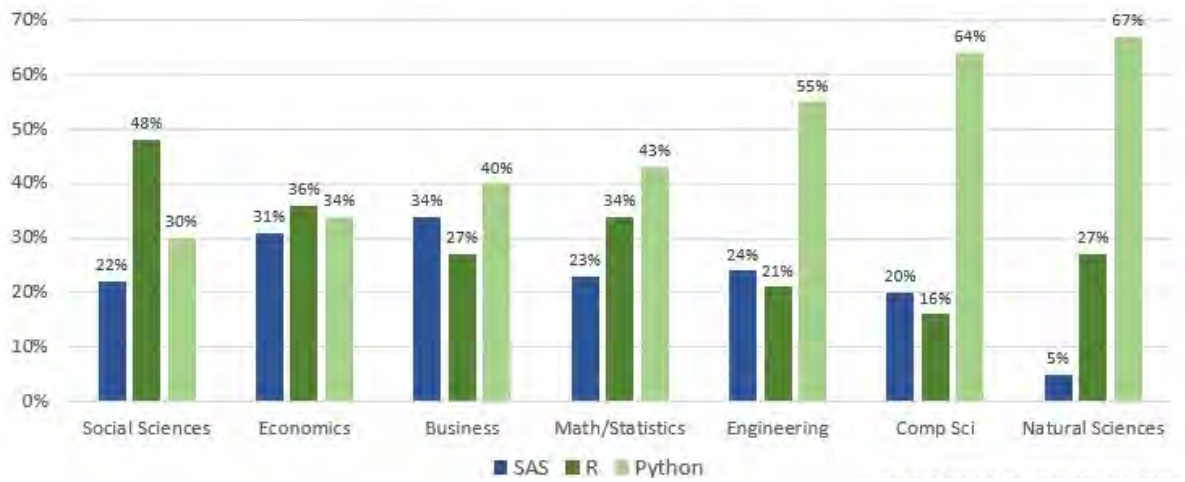

Figura 20. Diagrama de preferencias por sector [27].

Análisis de datos (*NumPy*, *IPython*, *matplotlib*, *pandas*)

En el ecosistema de desarrollo de Python existen un conjunto de poderosas librerías de código abierto que han sido de gran utilidad para la comunidad científica y que se han hecho parte indispensable en los proyectos de investigación. Las librerías permiten, entre otras cosas, procesar, manipular, limpiar, y demás acciones sobre los datos a procesar en flujos de trabajo de investigación de una manera controlada y relativamente simple [29].

- *Numpy* es la librería de herramientas matemáticas y manejo de arreglos predilecta [30].
- *IPython* es una librería gráfica que permite la visualización de información de manera interactiva [31].
- También *matplotlib* es una herramienta gráfica muy poderosa, ideal para generar reportes visuales de la información procesada [32].
- Finalmente, la librería *pandas* ofrece herramientas de control y manejo para grandes conjuntos de datos [33], [34].

Interfaz de implementación (Jupyter Notebook)

Jupyter-notebook es una herramienta en Python que funciona como interfaz web y que también es de código abierto. Brinda capacidades de virtualización para ambientes de desarrollo que pueden ser controlados y accedidos desde cualquier navegador. Estas "libretas" permiten describir etapas o elementos comunes en flujos de trabajo de investigación, como son la documentación, códigos o visualizaciones de datos. Estas características lo han posicionado como una robusta herramienta científica para compartir código e incluso procesamiento [35]. A continuación se presenta un ejemplo de la interfaz gráfica de un *Jupyter Notebook* (Fig. 21).

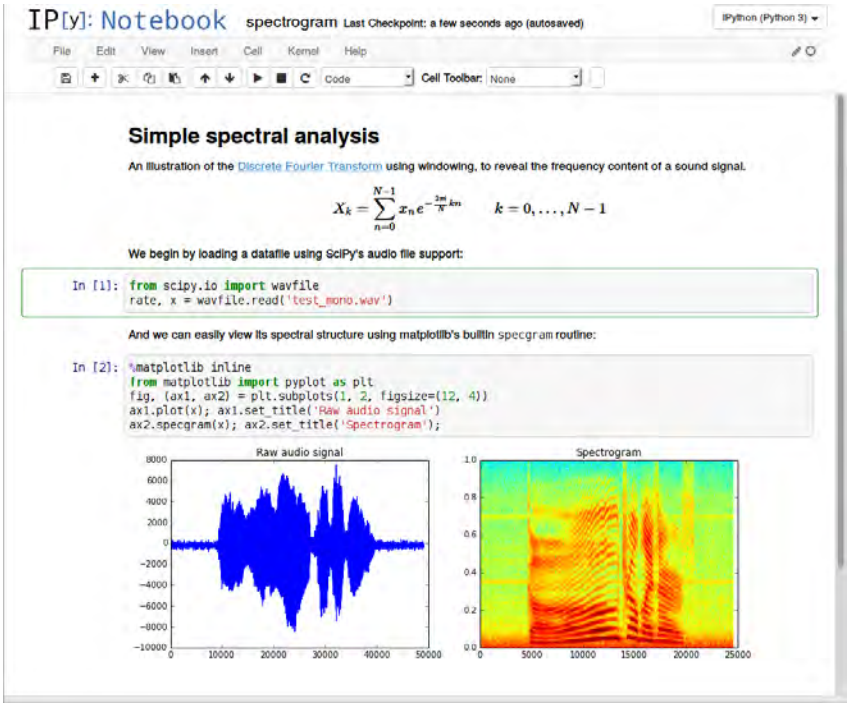

Figura 21. Diseño de un Jupyter Notebook.

Motor de aprendizaje

Para el desarrollo de los modelos de aprendizaje automático se utilizó la librería *Keras,* que permite utilizar la plataforma *TensorFlow* a manera de API (*Application Programming Interface*), lo cual hace posible desarrollar sistemas de aprendizaje automático personalizados de manera sencilla y organizada [36].

TensorFlow (TF)

TensorFlow es una plataforma de desarrollo de código abierto para el aprendizaje automático que funciona como una capa de infraestructura para programación diferenciable estática (compilable), con capacidades de ejecución operacionales a nivel CPU, GPU o TPU, cómputo de gradientes y debido a sus características estáticas de compilación también presenta una gran escalabilidad y la habilidad de exportar proyectos compilados a distintas plataformas, ya sean servidores, navegadores o dispositivos móviles [37].

La Fig. 22, presenta una comparación estimada del uso en las tecnologías más importantes de aprendizaje automático: TF y *PyTorch*. Aquí se puede apreciar la preferencia de la comunidad por estas tecnologías. *PyTorch*, a diferencia de TF, tiene un paradigma de desarrollo dinámico que, aunque brinda ciertos beneficios en la programación, también presenta limitantes en optimización y escalabilidad.

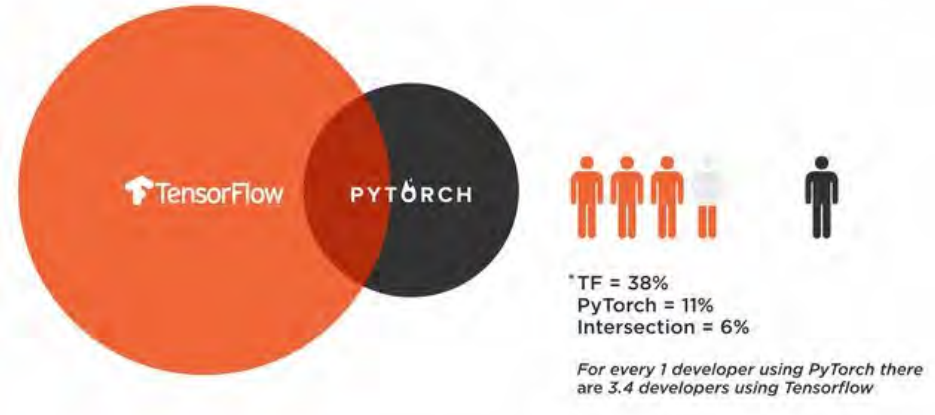

Figura 22. Diagrama de usuarios de TensorFlow y PyTorch.

Keras

Keras es una API de alto nivel que se utiliza como interfaz de programación y cuyo objetivo es el desarrollo de soluciones de aprendizaje automático bajo la plataforma TF. Esta librería provee abstracciones y elementos esenciales para la construcción, desarrollo e implementación de productos de aprendizaje automático [38].

En la siguiente tabla (Fig. 23) se observa la operación y ejecución entre Keras y TF, donde se aprecia que la primera presenta funciones y herramientas de más alto nivel de abstracción que facilitan el diseño de sistemas de aprendizaje automático mientras que con la segunda se realizan las operaciones de aprendizaje sobre el medio de ejecución.

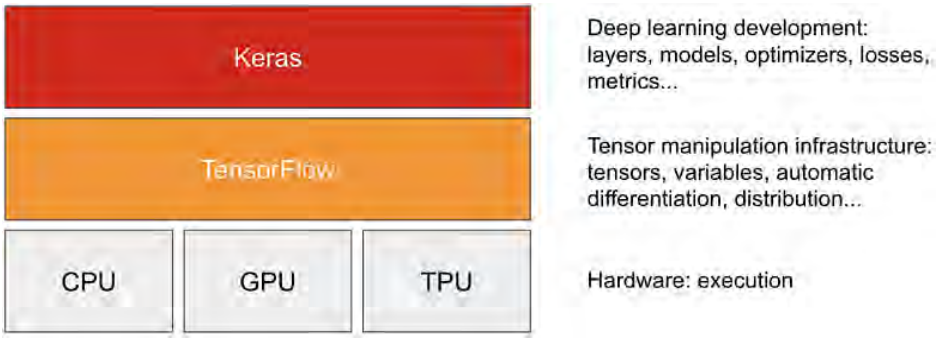

Figura 23. Diagrama de ejecución de *TensorFlow* y *Keras*.

Implementación

Una vez definidas las tecnologías y plataformas a utilizar, se propusieron las siguientes secciones de implementación:

- Definición de fuentes y medios de control de datos
- Clasificación semiautomática de imágenes para entrenamiento generativo
- Construcción y compilación de modelos de aprendizaje automático
- Pruebas de entrenamiento y evaluación de resultados
- Optimización y ajuste de modelos implementados

Para llevar a cabo dicha implementación se requieren múltiples elementos: las imágenes fuente, un generador que permita alimentar los modelos con las imágenes SEM de forma controlada, y finalmente, los dos modelos principales: un preclasificador, para seleccionar las imágenes fuente más adecuadas para el entrenamiento, y la red GAN para la producción sintética de muestras.

Imágenes Fuente

Para procesar las imágenes fuente se utilizó un "*DataFrame"* de la librería *pandas,* que puede ser exportado o guardado en formato CSV, como estructura de control de datos.

Así, se construye el *dataframe* inicial con la información necesaria para el proceso de las imágenes referenciando un directorio específico con los siguientes valores predeterminados:

- Identificador (*id*): Identificador único de imágenes. En este caso se utilizó el nombre del archivo correspondiente a cada imagen.
- Etiqueta (*label*): Clasificación asignada, entre "YES" o "NO", donde "YES" representa que es apta para entrenamiento.
- Ruta de acceso (*path*): Dirección específica de la imágen en su directorio.

Y, mediante un generador de flujos de datos (*ImageDataGenerator* de Keras [39]), que también ofrece ciertas capacidades de preprocesamiento, se alimentaron los modelos mediante flujos específicos a partir de los *dataframes* según el caso.

Selección semiautomática

Durante las pruebas iniciales de implementación resultaba complicado obtener resultados convincentes de imágenes sintéticas por lo que se diseñó una sección de preprocesamiento que consiste en seleccionar las imágenes fuente de entrenamiento mediante un modelo clasificador binario entrenado con imágenes manualmente etiquetadas como “aptas” para entrenar, contra ruidosas o irrelevantes.

Selección manual

Para la selección inicial de imágenes de validación se tomó un porcentaje del total de las imágenes fuente y se clasificaron manualmente. En la Fig. 24 se presentan un par de imágenes donde se aprecia que la imagen de la izquierda presenta ruido y carece de información o características importantes para entrenar, contrastando con la imagen de la derecha, la cual presenta múltiples elementos que a simple vista se aprecian relevantes para el entrenamiento del generador.

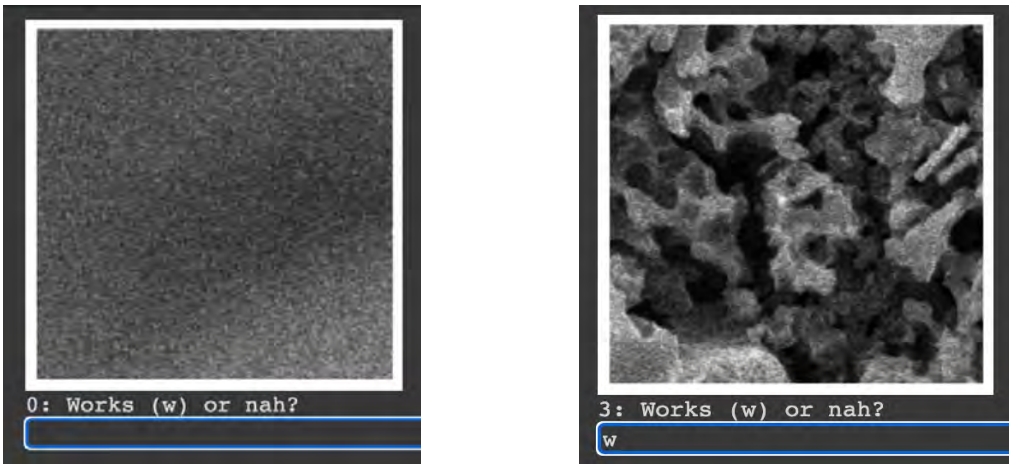

Figura 24. Capturas de pantalla del sistema de selección.

La clasificación manual se realizó mediante un *dataframe* donde se asignaron etiquetas de referencia (*lbl*) para que el modelo clasificador pueda diferenciar dichas imágenes.

Pre-Clasificador

Seguidamente, se definió un clasificador profundo convolucional para identificar las imágenes más representativas y con mayor calidad o resolución para el entrenamiento del generador GAN. Se optó por un modelo clasificador binario básico secuencial, constituido por un par de capas convolucionales con salidas promediadas que posteriormente alimentan a una capa densa de clasificación múltiple, que a su vez alimenta a otra capa densa pero de una neurona binaria que finalmente clasifica las imágenes [40].

En la Fig. 25 se presenta la arquitectura para el modelo clasificador convolucional.

```
Model: "classifier"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 320, 320, 1)]     0

 conv2d (Conv2D)             (None, 313, 313, 8)       520

 max_pooling2d (MaxPooling2D  (None, 156, 156, 8)      0
 )

 conv2d_1 (Conv2D)           (None, 149, 149, 16)      8208

 max_pooling2d_1 (MaxPooling  (None, 74, 74, 16)       0
 2D)

 flatten (Flatten)           (None, 87616)             0

 dense (Dense)               (None, 32)                2803744

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 2,812,505
Trainable params: 2,812,505
Non-trainable params: 0
_________________________________________________________________
```

Figura 25. *Classifier summary*.

Después de definir las capas internas es necesario compilar el modelo. Este paso requiere definir un optimizador (*adam*) y una medida de pérdida (*binary cross entropy*). Adicionalmente se pueden definir métricas más específicas a vigilar, como la precisión (*acc*). Todos estos elementos serán relevantes durante el entrenamiento del modelo.

Una vez compilado, el entrenamiento, o *fit*, se ejecuta con las imágenes que fueron previamente etiquetadas manualmente, para de manera automática etiquetar o predecir todas las imágenes fuente originales, minimizando la cantidad de imágenes con pocas características relevantes para utilizar en el entrenamiento de la red GAN. Cabe mencionar que para el entrenamiento se definió una función de control, o *callback*, para evitar el sobreentrenamiento, permitiendo reducir el costo de procesamiento tomando en cuenta que esta clasificación corresponde a una etapa de preprocesamiento para la red GAN.

Igualmente, mediante un *dataframe* se asignaron las etiquetas a las imágenes de acuerdo a la predicción obtenida por la red neuronal descrita en esta sección.

Red GAN

La red GAN se diseñó como un modelo de clase personalizable, a diferencia del *preclasificador*, con el objetivo de tener control sobre el comportamiento y arquitectura de los modelos internos, del generador y del discriminador, así como de los pasos a iterar en su entrenamiento (*train_step*) [41].

En la Fig. 26 se muestran los métodos básicos de la clase GAN personalizada donde se puede apreciar la inicialización de los modelos internos y el método *call* para generar nuevas imágenes desde vectores aleatorios de tamaño definido, o de espacio latente (*latent space*). También se definen las características de compilación, donde se asignan los optimizadores para los modelos internos, así como la función de error a evaluar durante la etapa de entrenamiento.

```python
class GAN(tf.keras.Model):
    """
    Custom GAN class model
    """

    def __init__(
        self,
        generator,
        discriminator,
        params: dict,
        name='GAN',
    ):
        super().__init__()
        self.latent_dim = params.get('LATENT_DIM')
        self.seed = seed
        self.discriminator = discriminator
        self.generator = generator

    def compile(self, d_optimizer, g_optimizer, loss_fn):
        self.loss_fn = loss_fn
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        super().compile()

    def call(self, inputs) -> List[tf.Tensor]:
        fakes = self.generator(
            get_random_vectors(
                tf.shape(inputs)[0], self.latent_dim
            )
        )
        predictions = [
            self.discriminator(inputs),
            self.discriminator(fakes)
        ]
        return [fakes, predictions]
```

Figura 26. Definición de la clase GAN.

Arquitectura

Los modelos internos fueron implementados mediante constructores que devuelven sistemas secuenciales de capas convolucionales basándose en las dimensiones de entrada y salida de las imágenes, y del vector de ruido a procesar, según sea el caso.

Discriminador

El discriminador definido para la red GAN es prácticamente un clasificador binario, muy similar al preclasificador descrito en la sección anterior, e incluso se podría implementar el mismo constructor para ambos modelos. Sin embargo, con el objetivo de tener un mejor control de los resultados obtenidos, y para poder evaluar el entrenamiento de la red, se optó por definir modelos específicos para cada caso.

Como se puede apreciar en la Fig. 27, el discriminador también cuenta con una capa de entrada donde se definen las dimensiones de la imagen a discriminar, seguida de una sección con sistemas de convolución: capas convolucionales 2D y sus respectivas funciones de activación, en este caso mediante *LeakyReLU,* y normalización del *batch*. Posteriormente, se asigna un sistema de capas densas, donde primero se procesa una capa de multiples neuronas para finalmente, mediante una capa densa de una sola neurona binaria, se realice la discriminación, similar al *preclasificador*.

```
Model: "discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 320, 320, 1)       65

 batch_normalization (BatchN  (None, 320, 320, 1)      4
 ormalization)

 leaky_re_lu (LeakyReLU)     (None, 320, 320, 1)       0

 conv2d_1 (Conv2D)           (None, 160, 160, 16)      1040

 batch_normalization_1 (Batc  (None, 160, 160, 16)     64
 hNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 160, 160, 16)      0

 conv2d_2 (Conv2D)           (None, 80, 80, 32)        8224

 batch_normalization_2 (Batc  (None, 80, 80, 32)       128
 hNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 80, 80, 32)        0

 conv2d_3 (Conv2D)           (None, 40, 40, 64)        8256

 batch_normalization_3 (Batc  (None, 40, 40, 64)       256
 hNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 40, 40, 64)        0

 conv2d_4 (Conv2D)           (None, 20, 20, 128)       32896

 batch_normalization_4 (Batc  (None, 20, 20, 128)      512
 hNormalization)

 leaky_re_lu_4 (LeakyReLU)   (None, 20, 20, 128)       0

 global_max_pooling2d (Globa  (None, 128)              0
 lMaxPooling2D)

 dense (Dense)               (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 68,086
Trainable params: 67,604
Non-trainable params: 482
```

Figura 27. Discriminador GAN.

Generador

El generador de la red GAN es el modelo más importante a configurar y de su correcto entrenamiento dependen los objetivos planteados en esta tesis, por lo que se buscó un balance entre sus propiedades y las del *discriminador*. Dicha estructura se describe en el *summary* de la Fig. 28.

```
Model: "generator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 6400)              416000

 dropout_1 (Dropout)         (None, 6400)              0

 reshape (Reshape)           (None, 10, 10, 64)        0

 conv2d_transpose (Conv2DTra  (None, 20, 20, 64)       16448
 nspose)

 batch_normalization_5 (Batc  (None, 20, 20, 64)       256
 hNormalization)

 leaky_re_lu_5 (LeakyReLU)   (None, 20, 20, 64)        0

 conv2d_transpose_1 (Conv2DT  (None, 40, 40, 64)       16448
 ranspose)

 batch_normalization_6 (Batc  (None, 40, 40, 64)       256
 hNormalization)

 leaky_re_lu_6 (LeakyReLU)   (None, 40, 40, 64)        0

 conv2d_transpose_2 (Conv2DT  (None, 80, 80, 64)       16448
 ranspose)

 batch_normalization_7 (Batc  (None, 80, 80, 64)       256
 hNormalization)

 leaky_re_lu_7 (LeakyReLU)   (None, 80, 80, 64)        0

 conv2d_transpose_3 (Conv2DT  (None, 160, 160, 64)     16448
 ranspose)

 batch_normalization_8 (Batc  (None, 160, 160, 64)     256
 hNormalization)

 leaky_re_lu_8 (LeakyReLU)   (None, 160, 160, 64)      0

 conv2d_transpose_4 (Conv2DT  (None, 320, 320, 64)     16448
 ranspose)

 batch_normalization_9 (Batc  (None, 320, 320, 64)     256
 hNormalization)

 leaky_re_lu_9 (LeakyReLU)   (None, 320, 320, 64)      0

 conv2d_transpose_5 (Conv2DT  (None, 320, 320, 1)      257
 ranspose)

=================================================================
Total params: 499,777
Trainable params: 499,137
Non-trainable params: 640
```

Figura 28. Generador GAN.

Así, después de probar distintas configuraciones, se definió la siguiente arquitectura: una capa de entrada de dimensión lineal que represente las características del vector de ruido a utilizar, un sistema de control inicial con un par de capas: una densa de múltiples unidades con *dropout*, para continuar con una capa de redimensionamiento que permita alimentar a las siguientes capas transpuestas convolucionales, donde cada una es emparejada con su respectiva función de activación y normalización, concluyendo con una capa final, también convolucional transpuesta pero de una única neurona y de activación por tangente hiperbólica (*tanh*), con la que se obtiene la imagen final generada.

Entrenamiento

El algoritmo de entrenamiento consiste en un proceso iterativo, que puede ser personalizado mediante la implementación de un método de clase "*train_step"*, donde se definen los pasos a seguir para modificar los pesos en las capas de la red durante cada iteración, lo que representa el proceso de aprendizaje que consta de las siguientes etapas: preparar datos de entrada para los modelos (imágenes reales, sintéticas y ruido), entrenar al discriminador y, finalmente, entrenar al generador. A continuación se presentan las secciones de código para las etapas mencionadas.

En la Fig. 29 se presenta el proceso inicial, donde se asignan las variables a utilizar durante el entrenamiento. Primeramente, se produce un vector de ruido para alimentar al generador, éste produce imágenes sintéticas que son concatenadas con las imágenes reales fuente, así como sus respectivas etiquetas, ya sean "reales" o "falsas".

```
def train_step(self, real_images):
    if isinstance(real_images, tuple):
        real_images = real_images[0]

    # Sample random points in the latent space
    batch_size = tf.shape(real_images)[0]
    random_latent_vectors = get_random_vectors(batch_size, self.latent_dim)

    # Decode them to fake images
    generated_images = self.generator(random_latent_vectors)

    # Combine them with real images
    combined_images = tf.concat(
        [generated_images, real_images],
        axis=0
    )

    # Assemble labels discriminating real from fake images
    labels = tf.concat(
        [tf.ones((batch_size, 1)), tf.zeros((batch_size, 1))],
        axis=0
    )

    # Add random noise to the labels - important trick!
    labels += 0.05 * tf.random.uniform(tf.shape(labels))
```

Figura 29. Primera etapa del TrainStep.

A continuación, con la información generada, se alimenta al discriminador para que genere predicciones, las cuales son evaluadas contra las etiquetas reales por la función de error definida y, mediante el optimizador asignado, se aplican operaciones con gradientes sobre los pesos internos para actualizarlos, o “entrenarlos”. Esta etapa del proceso se puede apreciar en la Fig. 30.

```
# Train the discriminator
with tf.GradientTape() as tape:
    predictions = self.discriminator(combined_images)
    d_loss = self.loss_fn(labels, predictions)
    grads = tape.gradient(
        d_loss,
        self.discriminator.trainable_variables
    )
    self.d_optimizer.apply_gradients(
        zip(grads, self.discriminator.trainable_variables)
    )
```

Figura 30. Entrenamiento del discriminador.

El siguiente paso consiste en generar un nuevo vector de ruido para que el generador produzca imágenes, pero en este caso las etiquetas a utilizar por el discriminador son todas iguales, o “reales”, aumentando las probabilidades del generador para engañar al discriminador entrenado. Seguidamente, con la función de error se evalúan las predicciones que se obtengan del discriminador contra las etiquetas falsas y mediante el optimizador asignado se actualizan los pesos del generador, concluyendo la iteración. Este paso se presenta en la Fig. 31.

```
# Sample random points in the latent space (again)
random_latent_vectors = get_random_vectors(batch_size, self.latent_dim)

# Assemble labels that say "all real images"
misleading_labels = tf.zeros((batch_size, 1))

# Train the generator
# (note that we should *not* update the weights of the discriminator)
with tf.GradientTape() as tape:
    generated_images = self.generator(random_latent_vectors)
    predictions = self.discriminator(generated_images)
    g_loss = self.loss_fn(misleading_labels, predictions)
    grads = tape.gradient(g_loss, self.generator.trainable_variables)
    self.g_optimizer.apply_gradients(
        zip(grads, self.generator.trainable_variables)
    )

return {"d_loss": d_loss, "g_loss": g_loss}
```

Figura 31. Entrenamiento del generador.

CAPÍTULO 3: RESULTADOS

En este capítulo se desarrolla una presentación de los resultados obtenidos en las distintas etapas de la solución propuesta, así como del escenario de implementación y los parámetros definidos para los entrenamientos.

Escenario

Como se mencionó en la sección de implementación, el proyecto fué realizado en un entorno de desarrollo *Jupyter*, y los resultados presentados fueron ejecutados bajo la plataforma web de máquinas virtuales de *Google Colab* [42]. Sin embargo, al hacer uso de una cuenta gratuita los recursos no están garantizados ni son ilimitados y, en ocasiones, los límites de uso fluctúan, como la capacidad de GPUs por sesión y el tiempo de asignación de las mismas, con un máximo de *12 horas* [43].

También, la plataforma permitió tomar como fuente de datos un directorio colaborativo en *Google Drive*, donde se almacenaron las imágenes originales tomadas mediante el microscopio SEM en una estructura de directorios catalogados por muestras, obtenidas por el equipo de investigación de la universidad para el análisis de microestructuras en celdas de combustible y al cual se nos brindó acceso para experimentación.

Preclasificador

Partiendo de la configuración original propuesta en la literatura [40], y considerando tanto las limitaciones de hardware como el hecho de que este modelo se presenta como un etapa de preprocesamiento, se implementó un clasificador acorde al tamaño promedio de las imágenes a procesar.

Así, considerando los resultados y la estabilidad de los entrenamientos evaluados, que se analizan más adelante, así como los tiempos ágiles de ejecución (el entrenamiento ronda los *5 minutos* en Colab), las características definidas previamente en la sección de implementación de este modelo se consideraron suficientes para el objetivo planteado de clasificación.

La arquitectura definida en la Fig. 32 fué entrenada con imágenes fuente de una dimensión original de *350 x 350 px* que se redimensionan a *320 x 320 px* para ajustarlas a un tamaño acorde a las capas convolucionales internas. Dichas imágenes alimentan al modelo a partir de un *ImageDataGenerator* que las pre-procesa y entrega en *batches* de *32* imágenes, siendo un total mínimo aproximado de *200* imágenes manualmente etiquetadas y divididas en dos subgrupos, entrenamiento y validación, con una proporción de *5:1*.

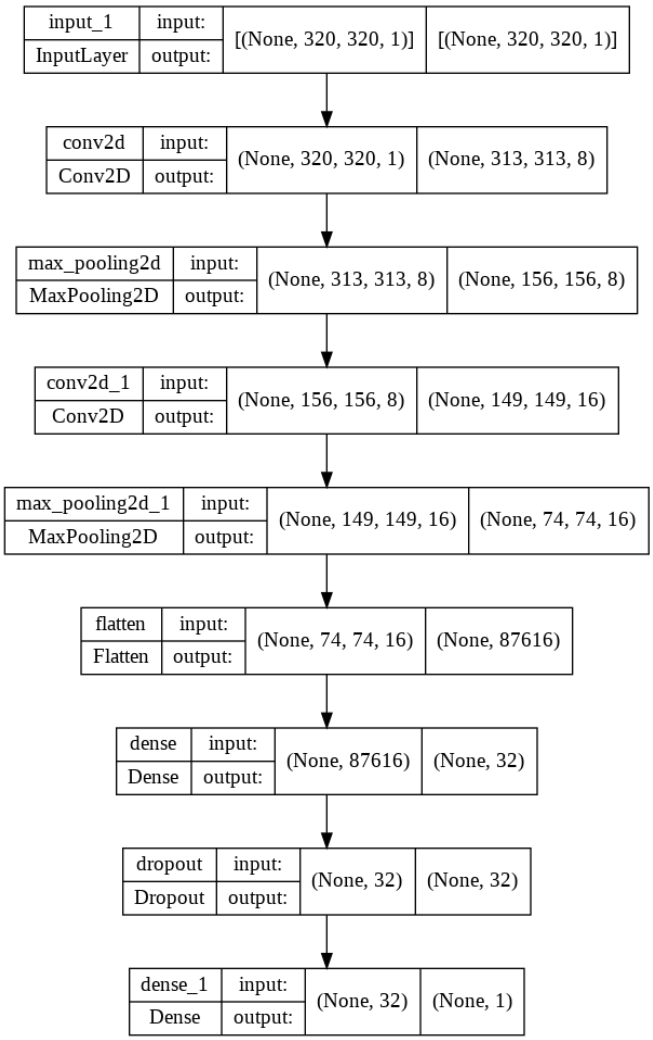

Figura 32. Capas internas de arquitectura del Pre-clasificador.

Entrenamiento

El entrenamiento de la red se programó para ejecutarse por *50* épocas (*epochs)* cuyas iteraciones se detienen mediante un *callback* de control al no presentar mejoras sustanciales, esto sucede en promedio alrededor de la época *30*. Igualmente, la función de *callback* almacena tanto un *checkpoint* del entrenamiento, como una versión del modelo más relevante considerando la precisión resultante de la etapa de validación por cada época.

En la Fig. 33 se presenta un ejemplo de las curvas de aprendizaje obtenidas tras la ejecución del *fit* sobre imágenes fuente en las etapas de entrenamiento y validación donde se presenta un comportamiento común para un modelo clasificador: al transcurrir las primeras épocas (*epochs*), la precisión (*acc*) es baja, menor a *0.5*, y progresivamente aumenta para acercarse a la unidad máxima, y en paralelo, los resultados de pérdida (*loss*) inician con valores extremos altos para acercarse a un mínimo local.

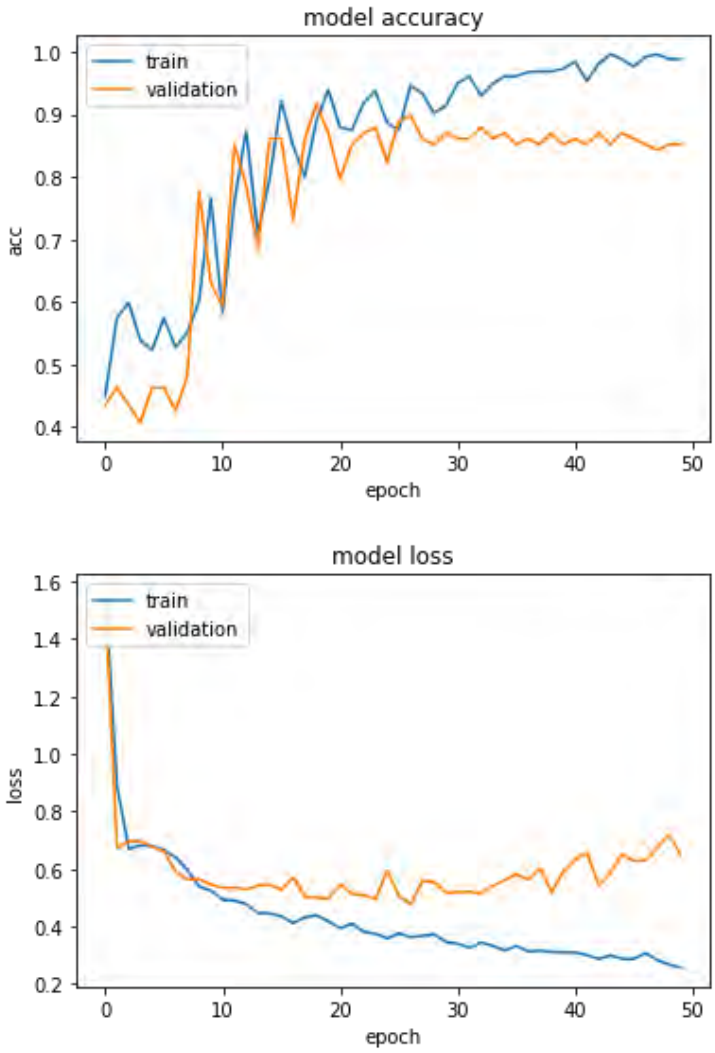

Figura 33. Curvas de aprendizaje del Pre-clasificador.

Resultados

Una vez concluido el entrenamiento, el *preclasificador* es utilizado para limpiar el lote total de imágenes fuente, de aproximadamente *4000* unidades, mediante la función *predict*, lo que requiere entre *20-30* minutos de ejecución sobre las GPUs de Colab. Dichos resultados son interpretados y almacenados en un *dataframe* para recuperarlos en formato CSV y reutilizarlos.

A continuación, en la Fig 34, se presentan ejemplos visuales de resultados obtenidos a partir de este procesamiento.

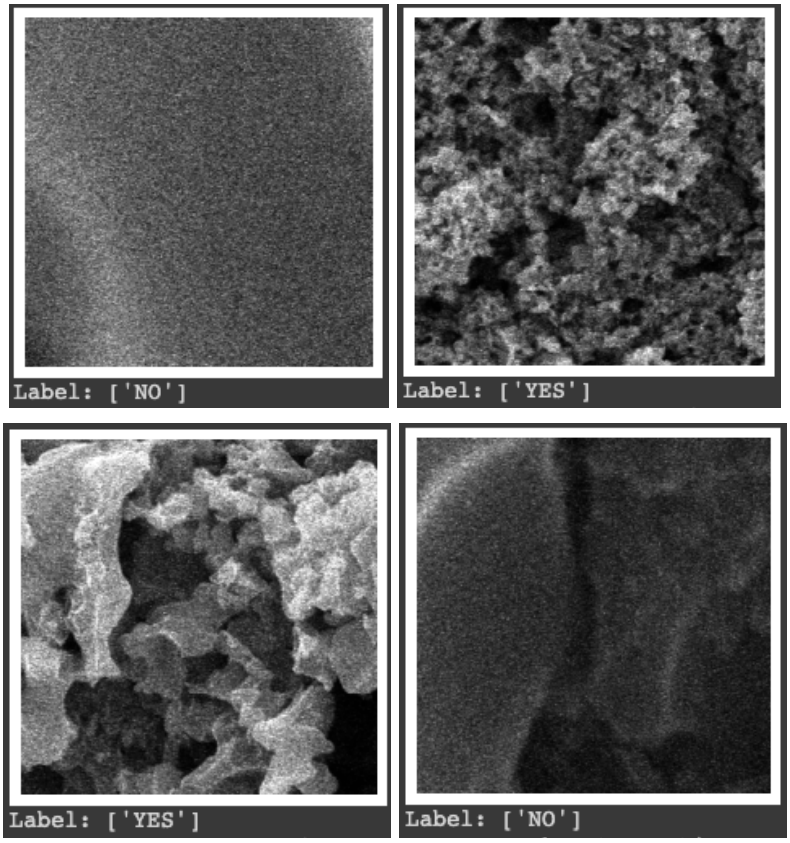

Figura 34. Resultados del etiquetado (*label*) del Preclasificador.
YES: se utilizará para entrenar la DCGAN, *NO*: no es apta para entrenamiento.

Como se aprecia en las imágenes anteriores, el *preclasificador* realiza un etiquetado valioso al filtrar las imágenes fuente ideales para el entrenamiento de la red GAN de las que presentan exceso de ruido o pocas características relevantes, automatizando un proceso manual para miles de imágenes.

Red GAN

De manera similar, para la DCGAN se evaluaron diversas características y comportamientos esperados a partir de un planteamiento inicial basado en la literatura [26], [41]. Y de la misma manera, mediante las dimensiones de las imágenes de entrada, se propusieron las características de las capas y neuronas descritas en la sección de implementación.

Las imágenes fuente preclasificadas también son de tamaño *320 x 320 px*, están en escala de grises y son procesadas en lotes (*batch*) de *32* imágenes. Del conjunto original se obtienen aproximadamente *2000* imágenes útiles después del filtrado del preclasificador; a ellas se les aplica un proceso de normalización para manejar valores en un rango de *[-1, 1]*, al igual que en la mayoría de redes neuronales para procesamiento de imágenes.

Se definió para el espacio latente (*latent space*) del vector de entrada que servirá para la generación de cada nueva imagen una dimensión de *64* elementos, sin embargo, es posible usar un espacio superior para obtener resultados más precisos. También se utilizó una función de error tipo *binary cross-entropy* y, para un balance entre la rapidez en el entrenamiento del discriminador y el generador, se designaron optimizadores independientes de aprendizaje, con tasas de aprendizaje de *0.0001* para el discriminador y de *0.0003* para el generador, ambos tipo *adam* (*adaptive momentum*).

Entrenamiento

El tiempo de ejecución aproximado para *400* épocas fue de *4* horas bajo el escenario establecido, siendo así más estable con estas configuraciones, como se presenta en las curvas de aprendizaje de la Fig. 35.

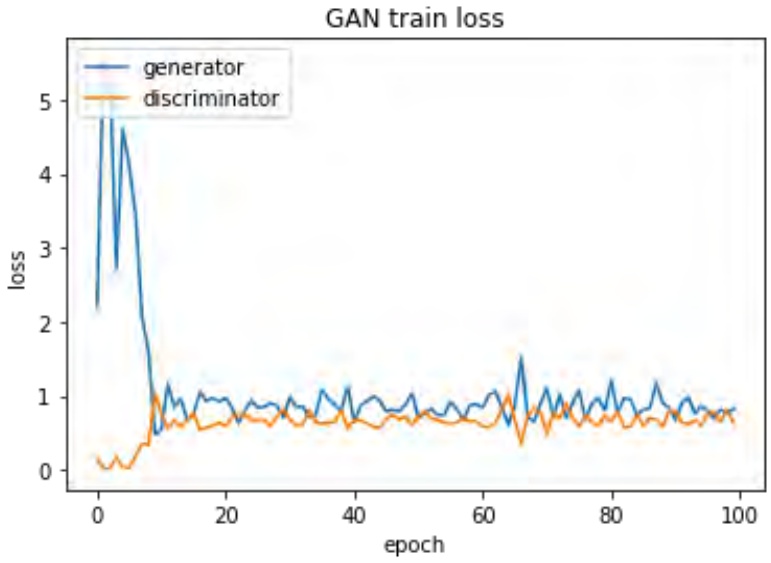

Figura 35. Curvas de aprendizaje (*loss*) de la DCGAN.

Las fluctuaciones que se presentan en las gráficas se deben al duelo *minimax* entre el discriminador y el generador, donde ambos modelos evolucionan en paralelo. Esto es un comportamiento esperado durante el inicio del entrenamiento, sin embargo, conforme los modelos van completando épocas, las curvas se estabilizan.

Durante este proceso también se registra el progreso de las imágenes sintéticas del modelo generador, así como los *checkpoints* del entrenamiento y los *CSVs* utilizados en las sesiones para un posterior almacenamiento, análisis y recuperación.

En la Fig. 36 se presentan los resultados obtenidos durante un entrenamiento de *100 epochs* de duración donde cada imágen representa el estado del generador durante cierta época, y donde todas las imágenes presentadas parten de la misma fuente de ruido (*seed*).

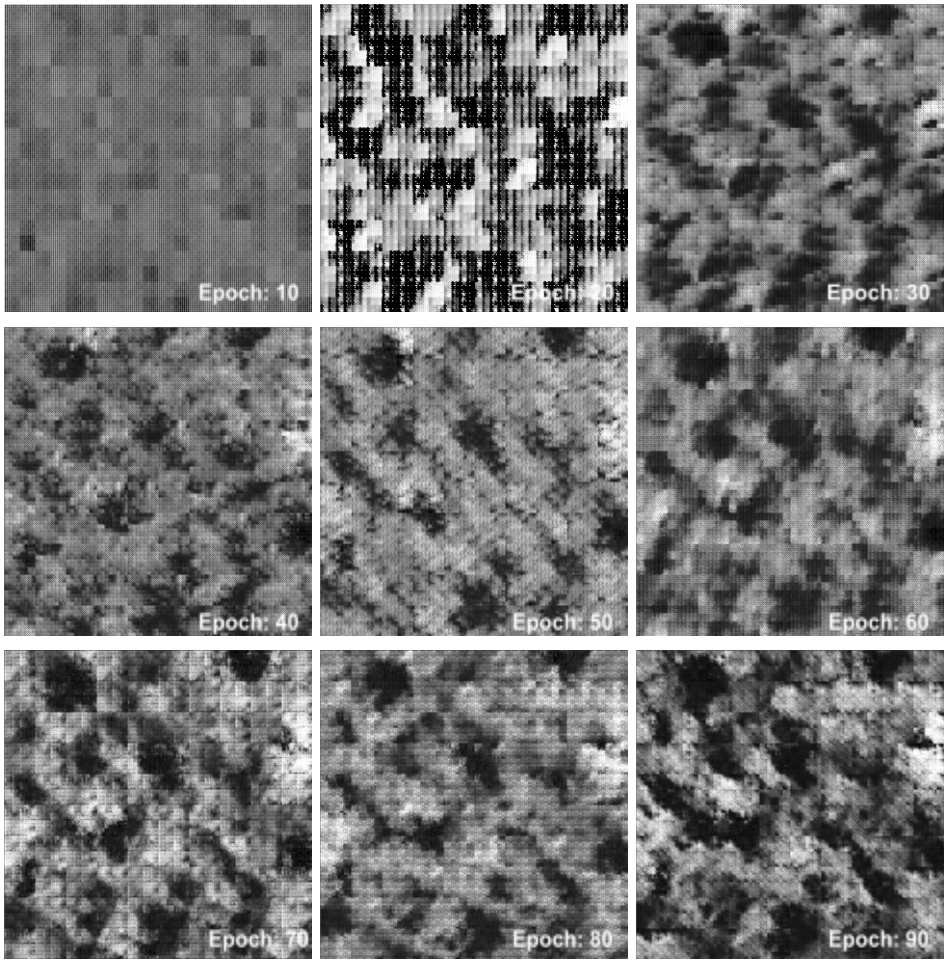

Figura 36. Progreso de resultados GAN (*100 epochs*).

Resultados

Con el generador de la GAN entrenado se pueden producir lotes de imágenes sintéticas a partir de vectores de ruido aleatorios, donde un batch de mil imágenes es generado en cuestión de minutos en la plataforma planteada. Las imágenes presentadas en la Fig. 37 corresponden a muestras sintéticas generadas por distintos modelos GAN entrenados durante *100* épocas.

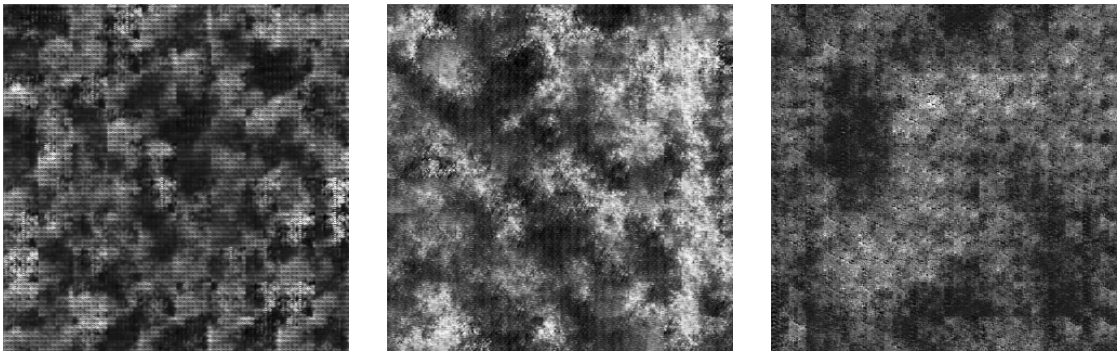

Figura 37. Muestras GAN.

Posteriormente, las imágenes resultantes se procesan mediante las funciones de correlación definidas en un módulo personalizado escrito en *C*, cuyos resultados son almacenados en un archivo *CSV*, y graficados para analizar los resultados sintéticos del generador de la red. Este procedimiento también se realiza con las imágenes fuente que sirvieron de entrenamiento para realizar comparaciones.

En la Fig.38 se presenta una primera comparativa estadística individual calculada bajo las funciones de correlación de dos puntos (*F2P*) y de camino lineal (*FLP*), con distinción según sus fases (*0s* y *1s*).

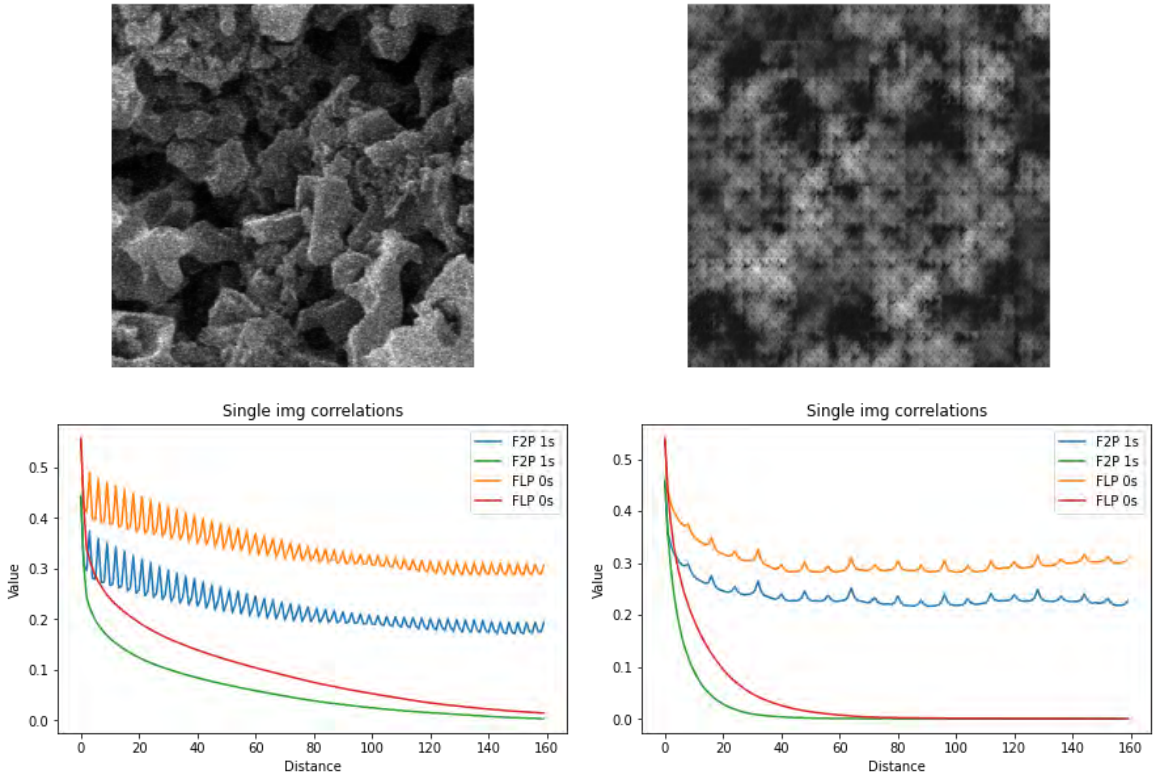

Figura 38. Resultados de caracterización en muestras individuales.

Las fluctuaciones que se aprecian en la función F2P que aparecen en la gráfica de los resultados de caracterización son un efecto producido por la binarización en las imágenes durante el procesamiento en *C*, donde se generan ciertos artefactos de ruido en las estadísticas individuales.

Estos artefactos se suavizan al realizar los cálculos promedio, como se presenta a continuación: las siguientes figuras (Fig. 39 y 40) corresponden a la caracterización estadística de mil muestras por tipo.

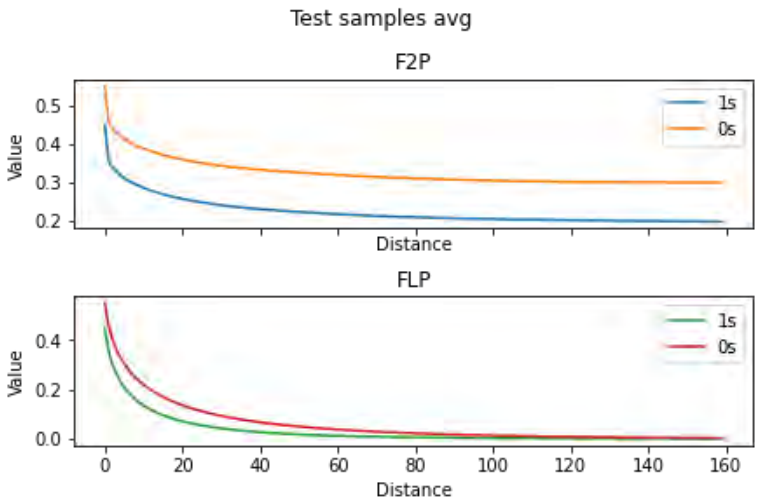

Figura 39. Resultados de caracterización para muestras reales.

Como se aprecia en los resultados del proceso de caracterización anterior (Fig. 39), las muestras reales presentan curvas similares para ambas funciones de correlación evaluadas, donde también se aprecia un mayor valor promedio en las fases en *0*.

Este comportamiento, así como las pendientes en las curvas generadas, son características estadísticas que también se aprecian en los resultados promediados de las muestras sintéticas, como se aprecia en la Fig. 30.

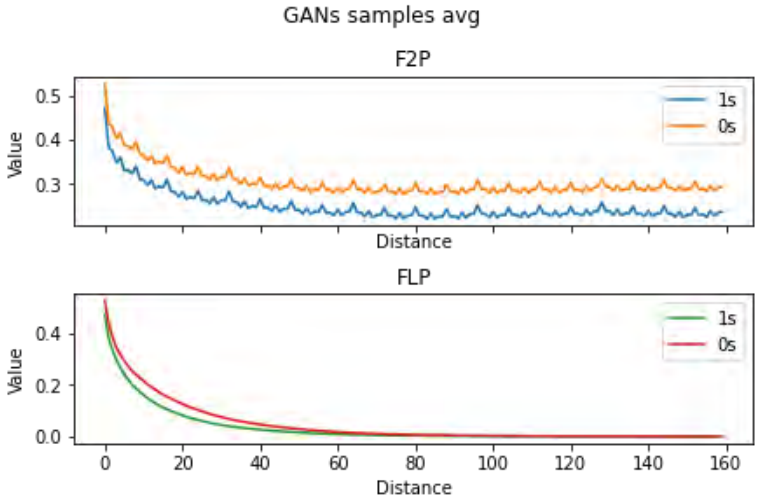

Figura 40. Resultados de caracterización para muestras GAN.

Sin embargo, para la función *F2P* también se aprecian picos de “ruido” que, en parte, son resultado de un efecto generado por el proceso de deconvolución que depende del espacio latente utilizado, o el tamaño del vector de ruido de entrada para la generación, que como se mencionó anteriormente es de *64* unidades.

CONCLUSIONES

La aplicación de modelos de aprendizaje automático en dicha metodología se presentó como alternativa a procesos iterativos clásicos y los resultados obtenidos se muestran prometedores.

Entre los resultados se puede destacar que los modelos son capaces de generar imágenes relativamente similares estadísticamente con entrenamientos relativamente cortos y además, una vez entrenados, producen imágenes sintéticas prácticamente de manera instantánea, siendo una alternativa más rápida a la metodología original.

También se logró demostrar la capacidad de abstracción de las redes neuronales convolucionales y la importancia de dicha operación en la arquitectura de los modelos, tanto clasificadores como generadores.

El análisis expuesto a lo largo de esta tésis pretende contribuir a la metodología de desarrollo de electrodos porosos basados en hidrógeno. Consecuentemente, el mejoramiento de estos materiales permitirá diseñar sistemas energéticos de mayor eficiencia y eficacia en su operación. Por ello, el software propuesto presenta un aporte para el desarrollo de dichos sistemas, siendo también un producto computacional innovador de alcances escalables para la ciencia de los materiales.

Trabajo Futuro

En las distintas etapas de implementación de la solución presentada se pueden identificar diversas oportunidades de desarrollo.

A continuación se presentan algunas tareas iniciales.

- En la sección del Preclasificador, se pretende desarrollar un método de entrenamiento “activo”, donde la selección manual pueda ser realizada durante la ejecución de los pasos de entrenamiento. Esto sería posible al implementar el modelo como una clase que permita personalizar su *train_step*, como se realiza en la DCGAN.
- De igual manera, en el entrenamiento de la DCGAN es posible implementar un ajuste dinámico de las tasas de aprendizaje de los optimizadores. Esto permitirá tener entrenamientos con mayor velocidad de convergencia y evitar eventos de sobreentrenamiento.
- Otro requerimiento presentado por parte del equipo de investigación es tener la posibilidad de evaluar una imágen de entrada original durante el proceso de generación de batches sintéticos, y de todas las posibles imágenes se realice un filtrado de las estadísticamente más similares a la imágen referencia mediante los coeficientes de correlación.

- Finalmente, el objetivo principal de este proyecto es obtener volúmenes representativos mediante un modelo generador (red neuronal), para lograr esto se requiere un análisis y diseño de una DCGAN más compleja. Entre las posibles opciones de solución se plantea evaluar el uso de capas convolucionales 3D.

BIBLIOGRAFÍA

[1] Colleen Spiegel, *Designing and Building Fuel Cells*. McGraw-Hill Professional, 2007. doi: 10.1036/9780071510639.

[2] C. Pacheco, R. Barbosa, A. Rodriguez, G. Oskam, M. Ruiz-Gómez, y B. Escobar, “Numerical Simulation to Determine the Effect of Topological Entropy on the Effective Transport Coefficient of Unidirectional Composites”, *Crystals*, vol. 10, núm. 6, Art. núm. 6, jun. 2020, doi: 10.3390/cryst10060423.

[3] R. Barbosa, J. Andaverde, B. Escobar, y U. Cano, “Stochastic reconstruction and a scaling method to determine effective transport coefficients of a proton exchange membrane fuel cell catalyst layer”, *J. Power Sources*, vol. 3, núm. 196, pp. 1248–1257, 2011, doi: 10.1016/j.jpowsour.2010.08.033.

[4] R. Barbosa, B. Escobar, U. Cano, J. Ortegon, y V. M. Sanchez, “Multiscale relationship of electronic and ionic conduction efficiency in a PEMFC catalyst layer”, *Int. J. Hydrog. Energy*, vol. 42, núm. 41, pp. 19399–19407, 2016, doi: 10.1016/j.ijhydene.2016.04.071.

[5] A. Rodriguez *et al.*, “Effect of An Image Resolution Change on the Effective Transport Coefficient of Heterogeneous Materials”, *Materials*, vol. 12, núm. 22, Art. núm. 22, ene. 2019, doi: 10.3390/ma12223757.

[6] R. Ledesma-Alonso, R. Barbosa, y J. Ortegón, “Effect of the image resolution on the statistical descriptors of heterogeneous media”, *Phys. Rev. E*, vol. 97, núm. 2, p. 023304, feb. 2018, doi: 10.1103/PhysRevE.97.023304.

[7] A. Rodriguez, R. Pool, J. Ortegon, B. Escobar, y R. Barbosa, “Effect of the Agglomerate Geometry on the Effective Electrical Conductivity of a Porous Electrode”, *Membranes*, vol. 11, núm. 5, Art. núm. 5, may 2021, doi: 10.3390/membranes11050357.

[8] R. Barbosa, B. Escobar, A. Rodríguez, y J. Andaverde, “A study on the conduction efficiency of solid materials that evolves from a particulate system to an overlapping discs agglomerate”, *Powder Technol.*, vol. 391, pp. 569–583, oct. 2021, doi: 10.1016/j.powtec.2021.06.041.

[9] B. Escobar *et al.*, “Simulated annealing and finite volume method to study the microstructure isotropy effect on the effective transport coefficient of a 2D unidirectional composite”, *Mater. Today Commun.*, vol. 24, p. 101343, sep. 2020, doi: 10.1016/j.mtcomm.2020.101343.

[10] J. Ortegon, R. Ledesma-Alonso, R. Barbosa, J. Vázquez Castillo, y A. Castillo Atoche, “Material phase classification by means of Support Vector Machines”, *Comput. Mater. Sci.*, vol. 148, pp. 336–342, jun. 2018, doi: 10.1016/j.commatsci.2018.02.054.

[11] I. J. Goodfellow *et al.*, “Generative Adversarial Networks”, *ArXiv14062661 Cs Stat*, jun. 2014, [En línea]. Disponible en: http://arxiv.org/abs/1406.2661

[12] Y. LeCun, Y. Bengio, y G. Hinton, “Deep learning”, *Nature*, vol. 521, núm. 7553, pp. 436–444, may 2015, doi: 10.1038/nature14539.

[13] Ian Goodfellow and Yoshua Bengio and Aaron Courville, *Deep Learning*. MIT Press, 2016. [En línea]. Disponible en: http://www.deeplearningbook.org

[14] Gliserio Romeli Barbosa Pool, “Estudio teórico-experimental de la capa catalítica y su influencia en los fenómenos de transporte en una PEMFC”, Universidad Nacional Autónoma de México, 2012.

[15] V. Mehta y J. S. Cooper, “Review and analysis of PEM fuel cell design and manufacturing”, *J. Power Sources*, vol. 114, núm. 1, pp. 32–53, feb. 2003, doi: 10.1016/S0378-7753(02)00542-6.

[16] R. Barbosa y B. Escobar Morales, “Electrochemical and Microstructural Analysis of a Modified Gas Diffusion Layer for a PEM Water Electrolyzer”, *Int. J. Electrochem. Sci.*, vol.

15, may 2020, doi: 10.20964/2020.06.12.
[17] S. Torquato, “Theory of Random Heterogeneous Materials”, en *Handbook of Materials Modeling: Methods*, S. Yip, Ed. Dordrecht: Springer Netherlands, 2005, pp. 1333–1357. doi: 10.1007/978-1-4020-3286-8_66.
[18] C. Manwart, S. Torquato, y R. Hilfer, “Stochastic reconstruction of sandstones”, *Phys. Rev. E*, vol. 62, núm. 1, pp. 893–899, jul. 2000, doi: 10.1103/PhysRevE.62.893.
[19] E.-Y. Guo, N. Chawla, T. Jing, S. Torquato, y Y. Jiao, “Accurate modeling and reconstruction of three-dimensional percolating filamentary microstructures from two-dimensional micrographs via dilation-erosion method”, *Mater. Charact.*, vol. 89, pp. 33–42, mar. 2014, doi: 10.1016/j.matchar.2013.12.011.
[20] S. Y. Chung y T. S. Han, “Reconstruction of random two-phase polycrystalline solids using low-order probability functions and evaluation of mechanical behavior”, *Comput. Mater. Sci.*, vol. 49, núm. 4, pp. 705–719, oct. 2010, doi: 10.1016/j.commatsci.2010.06.014.
[21] M. Baniassadi, S. Ahzi, H. Garmestani, D. Ruch, y Y. Remond, “New approximate solution for N-point correlation functions for heterogeneous materials”, *J. Mech. Phys. Solids*, vol. 60, núm. 1, pp. 104–119, ene. 2012, doi: 10.1016/j.jmps.2011.09.009.
[22] R. Bostanabad, A. T. Bui, W. Xie, D. W. Apley, y W. Chen, “Stochastic microstructure characterization and reconstruction via supervised learning”, *Acta Mater.*, vol. 103, pp. 89–102, ene. 2016, doi: 10.1016/j.actamat.2015.09.044.
[23] “An Introduction to Computational Fluid Dynamics - H. Versteeg - 9780131274983 - Mechanical Engineering - Thermal Science (120)”. https://www.pearson.ch/HigherEducation/Pearson/EAN/9780131274983/An-Introduction-to-Computational-Fluid-Dynamics
[24] F. Chollet, *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018.
[25] D. P. Kingma y J. Ba, “Adam: A Method for Stochastic Optimization”, *ArXiv14126980 Cs*, ene. 2017, [En línea]. Disponible en: http://arxiv.org/abs/1412.6980
[26] “Deep Convolutional Generative Adversarial Network | TensorFlow Core”, *TensorFlow*. https://www.tensorflow.org/tutorials/generative/dcgan
[27] “The State of the Octoverse”, *The State of the Octoverse*. https://octoverse.github.com/
[28] T. E. Oliphant, “Python for Scientific Computing”, *Comput. Sci. Eng.*, vol. 9, núm. 3, pp. 10–20, may 2007, doi: 10.1109/MCSE.2007.58.
[29] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O’Reilly Media, Inc., 2012.
[30] C. R. Harris *et al.*, “Array programming with NumPy”, *Nature*, vol. 585, núm. 7825, pp. 357–362, sep. 2020, doi: 10.1038/s41586-020-2649-2.
[31] F. Perez y B. E. Granger, “IPython: A System for Interactive Scientific Computing”, *Comput. Sci. Eng.*, vol. 9, núm. 3, pp. 21–29, 2007, doi: 10.1109/MCSE.2007.53.
[32] J. D. Hunter, “Matplotlib: A 2D Graphics Environment”, *Comput. Sci. Eng.*, vol. 9, núm. 3, pp. 90–95, may 2007, doi: 10.1109/MCSE.2007.55.
[33] W. McKinney, “Data Structures for Statistical Computing in Python”, Austin, Texas, 2010, pp. 56–61. doi: 10.25080/Majora-92bf1922-00a.
[34] J. Reback *et al.*, *pandas-dev/pandas: Pandas 1.3.2*. Zenodo, 2021. doi: 10.5281/zenodo.5203279.
[35] B. M. Randles, I. V. Pasquetto, M. S. Golshan, y C. L. Borgman, “Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study”, en *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, jun. 2017, pp. 1–2. doi: 10.1109/JCDL.2017.7991618.
[36] “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition [Book]”. https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/
[37] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed

Systems", p. 19.
[38] F. Chollet and others, "Keras", 2015. https://keras.io/
[39] "ImageDataGenerator", *TensorFlow*. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
[40] "Image classification | TensorFlow Core", *TensorFlow*. https://www.tensorflow.org/tutorials/images/classification
[41] "Customize what happens in Model.fit | TensorFlow Core", *TensorFlow*. https://www.tensorflow.org/guide/keras/customizing_what_happens_in_fit
[42] E. Bisong, "Google Colaboratory", en *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, E. Bisong, Ed. Berkeley, CA: Apress, 2019, pp. 59–64. doi: 10.1007/978-1-4842-4470-8_7.
[43] "Google Colab". https://research.google.com/colaboratory/faq.html