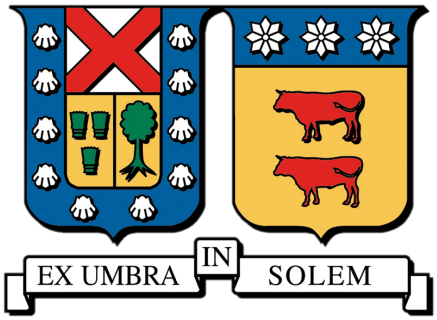
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INGENIERÍA MECÁNICA

VALPARAÍSO-CHILE



Utilización de Redes Neuronales Convolucionales para Resolver la Ecuación de Poisson

Eduardo Matías Hasbun Contreras

Tesis de grado para optar al Grado Académico de Magíster en Ciencias de la Ingeniería Mecánica y al Título de Ingeniero Civil Mecánico

Profesor Guía: PhD. Christopher Cooper

Profesor Correferente: PhD. Ignacio Muga, PhD. Joaquín Mura

Agosto-2025

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☒ Tesis de Postgrado;

Título del trabajo: Utilización de Redes Neuronales Convolucionales para Resolver la Ecuación de Poisson

Nombre del candidato(a): Eduardo Matías Hasbún Contrereas

Carrera / Grado: Magíster en Ciencias de la Ingeniería Civil Mecánica / Ingeniería Civil Mecánica

Campus: Casa Central Valparaiso **;** **Departamento**: Ingeniería Mecánica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Christopher Cooper, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 13-08-2025 ; **Firma:** 

Estudiante o Candidato(a):

Fecha: 12-08-2025 ; **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que han apoyado durante el desarrollo de este trabajo, partiendo por mi papá Jorge, mi hermana Josefa y mi hermano Juan, quienes me aportaron con su apoyo incondicional en todo momento, además de sus enseñanzas y consejos que me ayudaron a crecer como persona ser quien soy hoy en día.

También quiero agradecer a mis tíos Marcia Y Cristian quienes me ayudaron a llevar un mejor pasar durante mi estadía en la ciudad y en la universidad.

Del mismo modo, expreso mi gratitud a mis amigos de toda la vida, Diego y María, con quienes crecí y compartí grandes momentos. Tambien a mis amigos de la universidad —Martín, Matías y Catalina—, por su apoyo en los momentos complejos y por acompañarme en cada logro académico.

Finalmente, quiero agradecer a mis profesor Christopher, por su confianza y conocimiento entregado durante el desarrollo de este trabajo.

Resumen

Las redes neuronales profundas han demostrado ser una herramienta poderosa para resolver problemas de ecuaciones diferenciales parciales (EDPs), permitiendo aproximar soluciones a diferentes escenarios donde los métodos tradicionales presentan grandes desafíos. Sin embargo su principal limitación es el alto costo computacional asociado a su entrenamiento y uso, junto a su baja capacidad de generalización.

Con el fin solventar este inconveniente, en este trabajo se propone un nuevo enfoque para resolver la ecuación de Poisson mediante el uso de redes neuronales convolucionales (CNNs) con el fin de generar un método más eficiente que el utilizado por las redes neuronales informadas por física (PINNs). El modelo propuesto consiste en entrenar una red CNN para un dominio con condiciones de frontera específicos el cual pueda ser utilizado para resolver diferentes casos de la ecuación de Poisson dentro del dominio establecido.

Los resultados obtenidos muestran que el modelo propuesto es capaz de aproximar la solución de la ecuación de Poisson para diferentes escenarios de dominios bidimensionales y tridimensionales, junto con condiciones de frontera variadas e incluso con dominios que incluyen interfaces con un buen grado de generalización, alcanzando errores promedios menores al 5 % en la mayoría de los casos.

Sin embargo, el modelo presenta limitaciones en cuanto a la generalización de dominios con geometrías complejas, en donde se observó que no es capaz de generalizar adecuadamente casos tridimensionales con interfaces si no se cuenta con datos de entrenamiento previamente resueltos, ademas de presentar una clara deficiencia en aproximar cargas puntuales, llegando a errores puntuales de sobre el 90 % en algunos casos.

Esto deja abierto un camino para futuras investigaciones, donde se invita a explorar el uso de redes neuronales convolucionales para resolver diferentes problemas de EDPs, así como la posibilidad de mejorar el modelo propuesto.

Palabras claves: Ecuación de Poisson, PINNs, Redes Neuronales Convolucionales

Abstract

Deep neural networks have proven to be a powerful tool for solving partial differential equations (PDEs), allowing for the approximation of solutions to different scenarios where traditional methods face significant challenges. However, their main limitation is the hight computational cost associated with their training and usage, along with their poor generalization capabilities.

To solve this inconvenience, this work proposes a new approach to solve the Poisson equation using convolutional neural networks (CNNs) to create a more efficient method than that used by physics-informed neural networks (PINNs). The proposed model consists of training a CNN for a domain with specific boundary conditions that can be used to solve different cases of the Poisson equation within the established domain.

The results obtained show that the proposed model is capable of approximating the solution of the Poisson equation for different scenarios of two-dimensional and three-dimensional domains, along with varied boundary conditions and even domains that include interfaces, achieving average errors below 5 % in most cases.

However, the model has limitations regarding the generalization of domains with complex geometries, where it was observed that it is not able to adequately generalize three-dimensional cases with interfaces unless previously solved training data is available. Additionally, it shows a clear deficiency in approximating point loads, resulting in point errors exceeding 90 % in some cases.

This opens a path for future research, inviting exploration of the use of convolutional neural networks to solve different PDE problems, as well as the possibility of improving the proposed model.

Keywords: Poisson Equation, PINNs, Convolutional Neural Networks

Índice general

Índice de figuras

Índice de cuadros

List of Algoritmos

Listings

Capítulo 1

Introducción

La ecuación de Poisson constituye un pilar de la modelación física y la ingeniería: describe potenciales electrostáticos, campos de presión en fluidos incompresibles, difusión de calor y fenómenos gravitacionales, entre otros. Su carácter elíptico implica que la solución depende de todo el dominio, lo que acarrea elevados costos computacionales en problemas de gran escala o cuando se requieren evaluaciones paramétricas múltiples. Los métodos numéricos clásicos—diferencias/elementos finitos y esquemas analíticos—siguen siendo la referencia en precisión, pero su costo crece rápidamente con la resolución y deben reejecutarse por completo ante cualquier modificación de las condiciones de contorno o de los parámetros físicos.

El avance de la capacidad computacional ha impulsado el empleo de redes neuronales en ciencias e ingeniería. Entre ellas destacan las *Physics-Informed Neural Networks* (PINN) [7], que incorporan las leyes físicas como parte de la función de pérdida, aprendiendo la solución de ecuaciones diferenciales parciales (EDP) sin necesidad de grandes conjuntos de datos. No obstante, al igual que los esquemas numéricos tradicionales, las PINN convencionales requieren un reentrenamiento completo cuando cambian las condiciones de contorno o los parámetros físicos, lo que limita su eficiencia práctica.

Para superar estas limitaciones se ha propuesto combinar PINN con arquitecturas convolucionales (CNN) [3], con el objeto de entrenar *una única red* capaz de resolver múltiples configuraciones del problema de Poisson. Esta estrategia pretende combinar la capacidad de generalización de las PINN con la rapidez de inferencia de las CNN, que han demostrado gran eficacia en visión por computador y procesamiento de imágenes, pero que aún se exploran de forma incipiente en el contexto de EDPs elípticas.

Sin embargo, existen pocos estudios en esta area y no se han realizado alguno que apliquen este

enfoque a dominios con geometrías complejas, condiciones de contorno mixtas o interfaces con permitividades discontinuas. Por lo tanto, se necesita un modelo que combine la flexibilidad física de las PINN con la eficiencia computacional de las CNN, ofreciendo una alternativa a los métodos numéricos tradicionales y a las PINN puramente fully–connected.

En este trabajo se presenta un análisis exhaustivo de un modelo PINN basado en CNN para resolver la ecuación de Poisson en distintos casos de estudio. La metodología se valida en dominios 2D y 3D que incluyen condiciones de contorno mixtas e permitividades discontinuas. Asimismo, se discuten la importancia de la regularización, la selección de pesos en la función de pérdida y la alineación de cargas puntuales con la malla, proporcionando pautas prácticas para extender el enfoque a geometrías más complejas y a EDPs no lineales.

Todo el código implementado en este trabajo se encuentra disponible en el repositorio de GitHub [8], facilitando el uso de este modelo a nuevos usuarios y contribuyendo al avance de la investigación en este campo.

1.1. Objetivo General

En este estudio se busca diseñar, implementar y validar un modelo de *Physics-Informed Neural Network* (PINN) basado en redes neuronales convolucionales (CNN) que resuelva la ecuación de Poisson en dominios bidimensionales y tridimensionales con condiciones de contorno mixtas e interfaces de permitividad discontinua, alcanzando precisiones razonables para aplicaciones prácticas y que permita la generalización para resolver múltiples configuraciones del problema sin necesidad de reentrenamiento.

1.2. Objetivos Específicos

- Implementar el modelo CNN–PINN y cuantificar su capacidad resolutiva y de generalización frente a variaciones en geometría, fuentes y condiciones de borde.
- Explorar distintas arquitecturas (*receptive field*, capas y escalas) y combinaciones de términos en la función de pérdida para optimizar la precisión y la estabilidad.
- Comparar el rendimiento del modelo con soluciones analíticas.
- Analizar la sensibilidad del modelo a los hiperparámetros de entrenamiento (tasa de aprendizaje, número de épocas, pesos de la funciones de pérdida) e investigar técnicas de regularización para mejorar la convergencia y la precisión.
- Estudiar el efecto de la resolución de la malla.
- Proponer lineamientos para extender el enfoque a EDPs elípticas no lineales y a dominios con topologías más complejas.
- Facilitar el uso del modelo a nuevos usuarios mediante la publicación del código en un repositorio de GitHub, incluyendo documentación y ejemplos de uso.

Capítulo 2

Marco Teórico

2.1. El Perceptrón

El perceptrón es la unidad más básica y fundamental del campo de redes neuronales artificiales. Introducido por Rosenblatt en 1958 [9], el perceptrón fue concebido como un modelo matemático inspirado en la forma en que las neuronas biológicas procesan la información. A pesar de su simplicidad, el perceptrón permitió sentar las bases para el desarrollo de redes neuronales complejas, utilizadas hoy en día en una amplia gama de aplicaciones.

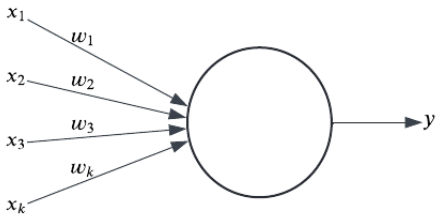


Figura 2.1: Esquema de un perceptrón

En esencia, el perceptrón es un clasificador lineal el cual recibe entradas, realiza una serie de operaciones aritméticas y produce una salida, la cual intenta predecir a qué clase pertenece la entrada. La operación básica del perceptrón puede describirse matemáticamente mediante la siguiente ecuación:

$$y = f(\sum_{i=1}^{n} w_i x_i + b) \tag{2.1}$$

Donde y es la salida del perceptrón, x_i son las entradas y w_i los pesos asociados a cada una de estas, b se conoce como término de sesgo (bias) y f es una función de activación que introduce no linealidad al modelo.

El término de sesgo, es un parámetro adicional que se introduce en el perceptrón para permitir que la función de activación se desplace a lo largo del eje de entrada. Este desplazamiento es crucial, ya que permite que el perceptrón tome decisiones más flexibles. En términos simples, el bias actúa como un valor constante que se suma al total ponderado de las entradas antes de que la función de activación procese la información. Sin este valor, la única forma de ajustar la salida de la red sería mediante los pesos de las entradas, lo cual puede no ser suficiente para capturar relaciones complejas de algunos datos

2.2. Redes Neuronales Artificiales

Las redes neuronales artificiales (ANNs) corresponde a una agrupación de funciones no lineales las cuales van operando sobre un conjunto de parámetros θ (conocidos como hiperparámetros), específicamente, se refiere a un conjunto de perceptrones los cuales forman capas y que operan sobre los pesos y sesgos de los mismos. Existen distintas arquitecturas que se pueden formar dependiendo de como se agrupen las neuronas y su cantidad de parámetros.

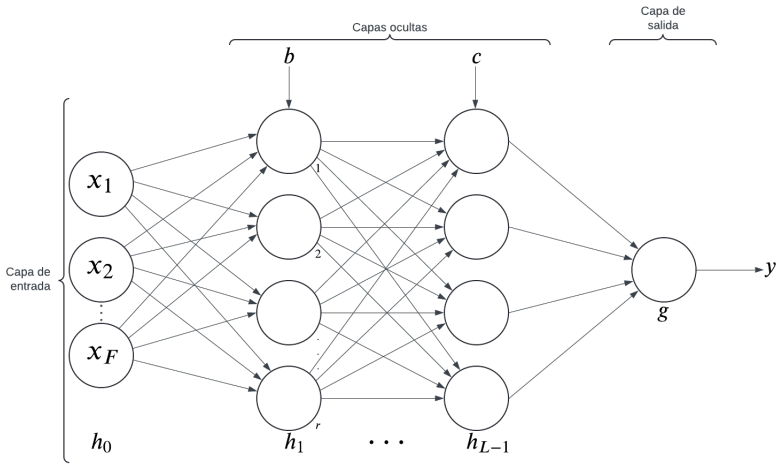


Figura 2.2: Diagrama de una Red Neuronal Feed Forward

En esencia, lo que busca una red neuronal es resolver u_θ por medio de la red neuronal $\mathcal{N}$ para aproximar la función u

$$u \approx u_\theta = \mathcal{N}(x;\theta) \tag{2.2}$$

La unidad base de una red neuronal artificial es el perceptron previamente explicado, estos se agrupan formando capas, las cuales se encargan de procesar la información en la red. Cada ANN consta de al menos 3 capas de neuronas: una capa de entrada, una o más capas ocultas y una capa de salida. Cada neurona está conectada con todas las neuronas de la siguiente capa a través de conexiones ponderadas.

- Capa de entrada: La capa de entrada es la que recibe las señales externas que alimentan la red. Cada neurona de esta capa corresponde a una característica de un conjunto de datos.
- Capas ocultas: Estas están compuestas por un conjunto de perceptrones y procesan la información. El número de neuronas y capas se conoce como hiperparametro y esta asociado a la arquitectura de la red.
- Capa de salida: Esta produce el resultado final de la red.

En general, una ANN con L capas ocultas se puede escribir como el siguiente conjunto de ecuaciones matemáticas:

$$\begin{aligned} h^{(i)} &= f^{(i)}(h^{(i-1)}W^{(i)} + b^{(i)}) \\ y &= g(h^{(L)}U + c) \\ h^{(0)} &= x \end{aligned} \tag{2.3}$$

En donde x es un vector con los parámetros de entrada de dimension (F), W es una matriz con los pesos de los perceptrones de dimension (F, r), b es un vector con los *bias* de dimension (r) y h es un vector de dimension (r).

Para que una ANN reproduzca una función esperada es sometida a un proceso de entrenamiento o aprendizaje. Para esto se plantea una función de perdida $\mathcal{L}$ la cual determina que tan alejado esta el resultado predicho por la red del real. De esta manera, para entrenar la red se busca minimizar el valor de la función de perdida variando los parámetros θ^* hasta encontrar los óptimos.

$$\theta^* = \underset{\theta}{argmin}\ \mathcal{L}(\theta) \tag{2.4}$$

Este proceso se suele realizar mediante métodos de optimización tales como descenso de gradiente, ADAM o BFGS y la función de perdida puede variar dependiendo del contexto en el

cual se este utilizando la red neuronal.

Cabe mencionar que las redes neuronales artificiales pueden ser utilizadas en diversos contextos tales como: procesamiento de texto, análisis de imágenes y videos, campos de la medicina, resolución de ecuaciones diferenciales parciales (PDE's por sus siglas en inglés), entre otros.

2.2.1. Universal Approximation Theorem

El *Universal Approximation Theorem* (UAT) establece que:

Sea f continua $[0,1]^k \to [0,1]$, para todo $\epsilon > 0$ existen $W,\ b,\ U$ tal que:

$$F(x) = sig(xW + b)U$$
$$|f(x) - F(x)| < \epsilon \ \forall x \in [0,1]^k$$

Este teorema es de gran peso para las redes neuronales, ya que implica que independiente de que función se busca entrenar, existe una ANN lo suficientemente grande que sea capaz de reproducirla. A pesar de que este teorema demuestra que ANN puede representar una función, no esta garantizado que lo haga, esto debido a que el entrenamiento puede fallar, específicamente de dos formas distintas: Primero, el algoritmo de optimización utilizado para el entrenamiento puede no ser capaz de encontrar el valor de los parámetros que corresponde a la función deseada. Segundo, el algoritmo de entrenamiento podría elegir la función incorrecta debido al sobreajuste [1].

2.3. Funciones de activación

Las funciones de activación son componentes esenciales del funcionamiento del perceptrón, y por ende, de las redes neuronales artificiales. Estas funciones determinan si una neurona debe o no activarse, basándose en el valor de la suma ponderada de las entradas y el sesgo. Originalmente, se utilizaba la función escalón, la cual tenia una salida binaria (0 o 1). Sin embargo, con el avance de las redes neuronales, se han desarrollado funciones de activación más sofisticadas que permiten a las redes aprender relaciones no lineales complejas. Algunas de estas funciones son [10]:

- **Funciones Sigmoide:** Esta es una función que genera una curva en forma de "S" la cual ha sido de gran importancia dentro de la historia de redes neuronales artificiales. Una de sus motivaciones son los patrones de activación observados en neurociencia. A continuación

se detallan las funciones más importantes de esta categoría:

- Sigmoid: La ecuación es la siguiente:

$$z \to \frac{1}{1 + e^{-z}} \tag{2.5}$$

  Esta función se inspira en el concepto biológico de "grado de activación neuronal" y tiene un rango de salida continuo entre (0, 1), a diferencia de la función escalón. Puede considerarse una versión suavizada de esta última.

- Arctan: La ecuación es la siguiente:

$$z \to arctan(z) \tag{2.6}$$

  A diferencia de la función anterior, esta tiene una salida entre (-$\pi/2$, $\pi/2$) y no tiene una motivación biológica. Dentro de las características principales de esta función, esta su simetría respecto al origen, lo cual la hace conveniente desde una perspectiva matemática y algorítmica, ademas es diferenciable en todo su dominio

- Tanh: Esta función se define como:

$$z \to tanh(z) \tag{2.7}$$

  La tangente hiperbólica puede interpretarse como una versión escalada y desplazada de la función sigmoide. Ademas, se ha comprobado que con esta función resulta una optimización mas sencilla con descenso de gradiente estocástico. [11]

- Softsign: La ecuación es la siguiente:

$$z \to \frac{1}{1 + |z|} \tag{2.8}$$

  Igual que las funciones vistas previamente, esta esta centrada en el origen. La ventaja de esta, es su simplicidad computacional, al no tener términos expotenciales, hace que el costo computacional de su evaluación sea bajo.

- **Funciones lineales por partes:** Estas funciones están compuestas por segmentos lineales. Similar a las funciones Sigmoide, estas se motivaron en sus inicios por observaciones neurobiologicas. Hoy en día, estas son las mas populares para las redes neuronales, esto debido a su bajo costo computacional tanto de la misma función como de sus derivadas.

Otra propiedad interesante de estas funciones es que son homogéneamente no negativas, lo cual ha sido util para la teoría y metodología del Deep Learning [12]

- Lineal: Esta es la mas sencilla de las funciones y se define como:

$$z \to z \tag{2.9}$$

  Se aprecia que la función deja la entrada intacta, debido a esto, por lo general esta se utiliza unicamente en algunas capas de la red neuronal (por ejemplo, en la capa de salida). Ademas, esta suele ser utilizada para analizar y probar algoritmos de optimización.

- ReLu: La ecuación es:

$$z \to max(0, z) \tag{2.10}$$

  Esta función consta de dos partes, la primera es su parte positiva la cual es equivalente a la función lineal y su parte constante con valor cero. Similar a la función lineal, esta tiene un bajo costo computacional tanto la función misma como su derivada. Esta tiene un problema conocido como "dying relu phenomenon", el cual sucede cuando muchos nodos de una red neuronal están inactivos durante largos periodos del entrenamiento lo cual dificulta el aprendizaje de modelos complejos. A pesar de esta dificultad, actualmente esta es la función más popular de las redes neuronales. de las redes neuronales artificiales.

- Leakyrelu: Dato un parámetro $a \in [0, \infty[$, esta función se define como

$$z \to max(0, z) + min(0, az) \tag{2.11}$$

  El objetivo de esta función es replicar el comportamiento de ReLU, evitando el problema conocido como "dying ReLU phenomenon" [13]. Sin embargo, aunque el método es prometedor, seleccionar un valor adecuado para el parámetro a es desafiante, ya que no existe una metodología establecida para hacerlo.

- **Aprendizaje de funciones de activación:** Por lo general, las funciones de activación se mantienen constantes durante el entrenamiento de una red neuronal, pero existen trabajos en los cuales se ha propuesto que durante el entrenamiento de la red se vaya seleccionando una función de un set predeterminado [14].

2.4. Función de Perdida

La función de perdida es la componente que mide la discrepancia entre el resultado predicho por la red y la solución real. Esto sirve como guía durante el proceso de aprendizaje para ajustar los parámetros mediante un proceso de optimización con le objetivo de minimizar esta diferencia. Para esto, la función de perdida se evalúa en *batches* (conjuntos de datos de entrenamiento) para asi encontrar los parámetros de red θ óptimos.

En ciertos casos, la función de perdida puede estar compuesta en varios términos los cuales se ponderan por pesos w_k para regular la influencia de cada uno en el entrenamiento:

$$\mathcal{L}(\theta; S) = \sum_{k=1}^{n} w_k \mathcal{L}_k(\theta; S_k) \tag{2.12}$$

En donde $\mathcal{L}$ corresponde a la funcion de perdida, $\mathcal{L}_k$ el termino k de la funcion de perdida, S_k, es el *batch* asociado a la funcion de perdida y w_k es el peso asociado a la misma. El *batch* de entrenamiento S corresponde a la unión de los subconjuntos S_K

$$S = \bigcup_{j} S_J \tag{2.13}$$

Existen distintos tipos de funcion de pérdida, a continuación se presentan los más utilizados:

- Perdida de entropía cruzada: Mide la diferencia entre la distribución de probabilidad predicha y la distribución real. Esta se utiliza principalmente en problemas de clasificación [15]. Se define como:

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{n} y_i log(\hat{y}_i) \tag{2.14}$$

- Error cuadrático medio: Mide el promedio cuadrado de los errores y es utilizado en problemas de regresión [16]. Su formula es:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.15}$$

- Error absoluto medio: Mide el promedio absoluto de los errores y se utiliza en casos similares a MSE, con la ventaja de que este no amplifica los errores al no tener un termino

cuadrado. Su formula es:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.16}$$

- Perdida Huber: Es una combinación entre *MSE* y *MEA*, en donde se utiliza la función absoluta cuando los errores son grandes y cuadrática cuando son pequeños. Su formula es:

$$\mathcal{L}_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{si } |y_i - \hat{y}_i| \leq \delta \\ \delta|y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{si } |y_i - \hat{y}_i| > \delta \end{cases} \tag{2.17}$$

  Donde δ es un umbral de transición entre ambos modelos.

2.5. Métodos de optimización

Los métodos de optimización son algoritmos iterativos empleados para ajustar los parámetros de la red (θ) con el fin de minimizar la función de pérdida $\mathcal{L}$. Existen dos categorías principales de algoritmos: los métodos basados en gradiente y los métodos con tasas de aprendizaje adaptativas. Además, es habitual incorporar técnicas estocásticas y de momento para evitar caer en mínimos locales y mejorar la eficiencia del modelo. [1]

A continuación se presentan los modelos más populares:

2.5.1. Descenso de gradiente

Este es el modelo más simple y consiste en actualizar los parámetros de red θ un pequeño paso en la dirección contraria a la derivada de la función de pérdida.

$$\theta_k + 1 = \theta_k - \lambda\nabla_k\mathcal{L}(\theta_k; S) \tag{2.18}$$

En donde λ es la tasa de aprendizaje, el cual define de que tamaño será el paso de la iteración.

Debido a lo simple del método, se presentan varios problemas tales como caer en mínimos locales.

2.5.2. Descenso de gradiente estocástico (SGD)

Este es uno de los modelos más utilizados en la actualidad. Es muy similar al modelo de descenso por gradiente, pero con la particularidad de que utiliza una estimación estocástica del gradiente

para actualizar los parámetros θ. Esto reduce la probabilidad de que el modelo quede atrapado en mínimos locales. En el contexto del Machine Learning, esto se implementa muestreando un pequeño grupo de m muestras del conjunto de entrenamiento y calculando su gradiente, como se muestra en el algoritmo 2.1.

Algoritmo 2.1 Algoritmo de Descenso Estocástico por Gradiente (SGD)

Require: Tasa de aprendizaje global λ
Require: Parámetros iniciales θ
  while No se cumple el criterio de parada **do**
    Muestrear un minibatch de m ejemplos del conjunto de entrenamiento $\{x_1,\ x_2,\ ...,\ x_m\}$
    Establecer $g = 0$
    for $i = 0$ hasta m **do**
      Calcular el gradiente: $g_{k+1} = g_k + \nabla_\theta \mathcal{L}(\theta_k : S)/m$
    end for
    Aplicar la actualización: $\theta_{k+1} = \theta_k - \lambda g_{k+1}$
  end while

Cabe destacar que para utilizar este algoritmo de forma eficiente, es crucial disminuir la tasa de aprendizaje durante el entrenamiento si se busca converger a un mínimo. Esto debido a que el modelo introduce *ruido* al gradiente, por lo cual, este no será 0 aunque se encuentre en el mínimo.

2.5.3. Descenso de gradiente estocástico con momento

A pesar de la gran eficacia de SGD, el modelo puede hacer que el aprendizaje sea lento, especialmente en situaciones donde el gradiente es pequeño.

El modelo de momentum [17] busca acelerar el proceso de entrenamiento en situaciones donde el gradiente es pequeño y constante. Este método se inspira en un fenómeno físico: imagina una pelota colocada en una superficie con una ligera inclinación y sin fricción. Al principio, una fuerza débil empuja la pelota cuesta abajo, haciendo que se mueva lentamente. Sin embargo, con el tiempo, la pelota gana velocidad y acelera. El algoritmo de momentum está diseñado para incorporar este efecto de aceleración en el proceso de descenso de gradiente.

Formalmente, se introduce la variable que cumple la función de la velocidad que se acumula en el gradiente en la ecuación 2.19.

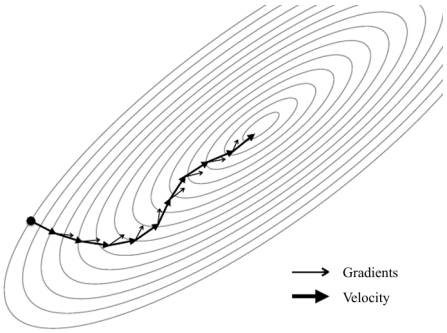


Figura 2.3: Esquema de momento [1]

$$\begin{aligned} v_{k+1} &= \alpha v + \lambda \nabla_k \frac{1}{m} \sum_{t=1}^{m} \mathcal{L} \\ \theta_{k+1} &= \theta_k - v \end{aligned} \tag{2.19}$$

El algoritmo 2.2 muestra como funciona este modelo dentro de una iteración.

Algoritmo 2.2 Algoritmo de SGD + momento

Require: Tasa de aprendizaje global λ, parámetro de momento α
Require: Parámetros iniciales θ, velocidad inicial v
  while No se cumple el criterio de parada **do**
    Muestrear un minibatch de m ejemplos del conjunto de entrenamiento $\{x_1,\ x_2,\ ...,\ x_m\}$
    Establecer $g = 0$
    for $i = 0$ hasta m **do**
      Calcular el gradiente: $g_{k+1} = g_k + \nabla_\theta \mathcal{L}(\theta_k : S)$
    end for
    Calcular la actualización de la velocidad: $v_{k+1} = \alpha v - \lambda g_{k+1}$
    Aplicar la actualización: $\theta_{k+1} = \theta_k - v_{k+1}$
  end while

2.5.4. RMSprop

Este algoritmo ajusta de manera individual las tasas de aprendizaje de todos los parámetros del modelo, escalándolas de forma inversamente proporcional a la suma acumulada de las derivadas parciales al cuadrado, que se encuentran ponderadas por un factor de amortiguación, a lo largo de todas las iteraciones del entrenamiento [18]. Este modelo viene representado por el algoritmo 2.3

Algoritmo 2.3 Algoritmo de RMSprop

Require: Tasa de aprendizaje global λ, tasa de decaimiento ρ
Require: Parámetros iniciales θ
  Inicializar la variable de acumulación $r = 0$
  while No se cumple el criterio de parada **do**
    Muestrear un minibatch de m ejemplos del conjunto de entrenamiento $\{x_1,\ x_2,\ ...,\ x_m\}$
    Establecer $g = 0$
    for $i = 0$ hasta m **do**
      Calcular el gradiente: $g_{k+1} = g_k + \nabla_\theta \mathcal{L}(\theta_k : S)$
    end for
    Acumular el gradiente: $r_{k+1} = \rho r_k + (1-\rho) g_{k+1}^2$
    Calcular actualización de parámetros: $\nabla\theta_k = -\frac{\lambda}{\sqrt{r}} \cdot g_{k+1}$ $\triangleright$ $\frac{1}{\sqrt{r}}$ aplicado a cada elemento
    Aplicar la actualización: $\theta_{k+1} = \theta_k - \nabla\theta_k$
  end while

2.5.5. Adam

Este modelo combina los principios de RMSprop y momentum. Su algoritmo calcula una tasa de aprendizaje adaptativa para cada parámetro, incorporando correcciones basadas en el primer y segundo momento de los gradientes, con el fin de ajustar el escalamiento de manera más eficiente. Este procedimiento se presenta en el algoritmo 2.4.

Algoritmo 2.4 Algoritmo de Adam

Require: Tasa de aprendizaje global λ, tasa de decaimiento ρ, coeficiente de momento α
Require: Parámetros iniciales θ, velocidad inicial v
  Inicializar la variable de acumulación $r = 0$
  while No se cumple el criterio de parada **do**
    Muestrear un minibatch de m ejemplos del conjunto de entrenamiento $\{x_1,\ x_2,\ ...,\ x_m\}$
    Establecer $g = 0$
    for $i = 0$ hasta m **do**
      Calcular el gradiente: $g_{k+1} = g_k + \nabla_\theta \mathcal{L}(\theta_k : S)$
    end for
    Acumular el gradiente: $r_{k+1} = \rho r_k + (1-\rho) g_{k+1}^2$
    Calcular actualización de velocidad: $v_{k+1} = \alpha v - \frac{\lambda}{\sqrt{r}} \cdot g$ $\triangleright$ $\frac{1}{\sqrt{r}}$ aplicado a cada elemento
    Aplicar actualización: $\theta_{k+1} = \theta_k - v_{k+1}$
  end while

2.6. Back propagation

Como se vio en la sección anterior, para entrenar la red neuronal se debe calcular las derivadas de la función de perdida $\mathcal{L}$ con respecto a los hiperparametros θ, por esta razón, es importante generar un algoritmo eficiente que realice este proceso. La técnica más utilizada para lograr

este objetivo es *back propagation*, el cual corresponde a un caso especial de la diferenciación automática [19], método reconocido por su eficiencia y bajo costo computacional. La idea básica de este algoritmo es utilizar la regla de la cadena para calcular la derivada parcial de la función de pérdida J con respecto a cualquier parámetro θ. Esto se logra a través de los valores intermedios que conectan θ con la función de perdida J. Para entender de mejor forma el algoritmo, a continuación se detallara el mismo considerando una red de la siguiente forma:

$$\begin{aligned} a_i^{(k)} &= b_i^{(k)} + \sum_{j=1}^{L} W_{ij}^{(k)} h_j^{(k-1)} \\ h_i^{(k)} &= f(a^{(k)})v \end{aligned} \tag{2.20}$$

Para esto, primero se debe realizar *forward propagation*, el cual es pasar una entrada x por la red y calcular la función de perdida de la salida de la misma. Tal como se detalla en el algoritmo 2.5.

Algoritmo 2.5 Algoritmo de *Forward propagation*

Require: Conjunto de datos x, objetivo y, tasa de aprendizaje λ
  $h_0 = x$
  for $k = 1$ hasta L **do**
    $a^{(k)} = b^{(k)} + W^{(k)} h^{(k-1)}$
    $h^{(k)} = f(a^{(k)})$
  end for
  $\hat{y} = h^{(L)}$
  $J = \mathcal{L}(\hat{y}; y) \cdot \lambda$

Una vez realizado el *forward propagation*, es necesario conocer cuanto contribuye cada parámetro del set θ a la función de perdida, por lo cual se calculan las derivadas para luego ajustarlas según el método de optimización seleccionado. Para esto, se aplica el algoritmo de *back propagation*, detallado en 2.6

2.7. Arquitecturas

2.7.1. Multi Layer Perceptron

Las arquitecturas de *Multi Layer Perceptron* (MLPs) consisten en conectar L capas de r_j perceptores de forma tal, que todos los perceptores de la capa L_i quedan conectados con todos los perceptrones de la capa L_{i+1}. De esta forma, cada perceptron r_j realiza la siguiente transfor-

Algoritmo 2.6 Algoritmo de *Back propagation*

Después del cálculo hacia adelante, calcular el gradiente en la capa de salida:
$g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} \mathcal{L}(\hat{y}; y) \cdot \lambda$
for $k = L$ hasta 1 **do**
  $g \leftarrow \nabla_{a^{(k)}} J = g \cdot f'(a^{(k)})$
  $\nabla_{b^{(k)}} J = g$
  $\nabla_{W^{(k)}} J = g h^{(k-1)}$
  $g \leftarrow \nabla_{h^{(k-1)}} J = W^{(k)} g$
end for

mación asociada a la eq. 2.21

$$h_j^i = f_j^i\left(\sum_{k=1}^{r^{(i-1)}} w_k^{(i-1)} h_k^{(i-1)} + b_k^{(i-1)}\right) \tag{2.21}$$

Un esquema de la arquitectura de MLP se observa en la Fig. 2.4

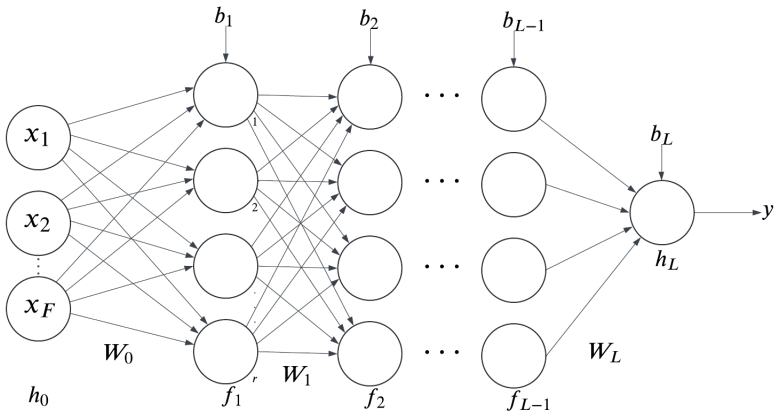


Figura 2.4: Esquema de una MLP

En donde la entrada de la red es el vector x y la salida es el vector y. Ademas, los parámetros internos de la red se pueden agrupar en el conjunto de parámetros θ como:

$$\theta = \{W_1, b_1, W_2, b_2, ..., W_L, b_L\} \tag{2.22}$$

El detalle del *forward propagation* de la red se muestra a continuación:

$$
\begin{aligned}
h_i &= f_i(W_i h_{i-1} + b_i) \quad 1 \leq i \leq L-1 \\
y &= f_L(W_L h_L + b_L) \\
h_0 &= x
\end{aligned}
\tag{2.23}
$$

Esta arquitectura es una de las más sencillas de ANNs, esto permite que se pueda utilizar en una gran variedad de aplicaciones. Ademas, esta arquitectura sirve como base para otras arquitecturas más complejas.

2.7.2. Residual Neural Network

La arquitectura de *residual neural network* (ResNet) [20] nace bajo la necesidad de utilizar redes con arquitecturas de MLP en problemas complejos, lo cual lleva a usar un número elevado de capas y se produce un fenómeno conocido como *vanishing gradient* [21] [22], el cual consiste en una cantidad de parámetros θ mayor a los necesarios para obtener la solución del problema planteado, por lo cual, en el entrenamiento las primeras capas de la red se quedan sin entrenar.

Para evitar este fenómeno, la arquitectura se construye con conexiones residuales, lo que permite pasar información directamente entre capas, como se muestra en el esquema de la Fig. 2.5

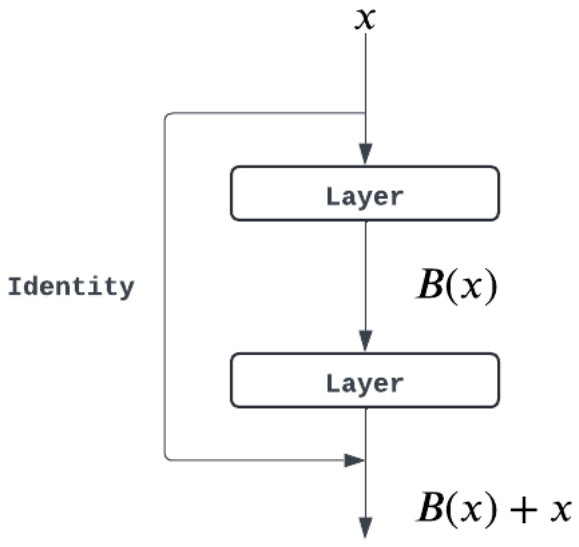


Figura 2.5: Diagrama de una red ResNet

A modo de ejemplo, se detallará como aplicar estas conexiones tomado como referencia una

arquitectura MLP. En este caso, se consideraran bloques de dos capas cada uno, como se muestra a continuación:

$$\begin{aligned} h_0 &= x \\ g_i &= f_i(w_i h_{i-1} + b_i) \\ h_i &= h_{i-1} + f_i(W_i g_i + b_i) \\ y &= f_L(W_L h_L + b_L) \end{aligned} \tag{2.24}$$

Lo que hace este modelo, es crear más "caminos" para calcular el gradiente de algún parámetro de la red durante el proceso de *back propagation*, por lo cual el entrenamiento llega de forma mas eficiente a los parámetros de las primeras capas ocultas.

2.7.3. Convolutional Neural Networks

Las redes neuronales convolucionales (CNNs) son un tipo de arquitectura derivada de las redes neuronales multicapa (MLPs), diseñada específicamente para reconocer, identificar y clasificar objetos en imágenes [23].

En la actualidad, las CNNs se utilizan ampliamente en tareas de identificación, análisis y segmentación de imágenes, ademas de otras aplicaciones que involucran el procesamiento de datos en 2D y 3D. Su efectividad radica en su arquitectura, optimizada para detectar características clave en las entradas, lo que las hace especialmente adecuadas para estos tipos de problemas. En Fig. 2.6, se observa un esquema de la arquitectura de una CNN, la cual opera sobre operadores lineales W_j, y operadores no lineales predefinidos [24].

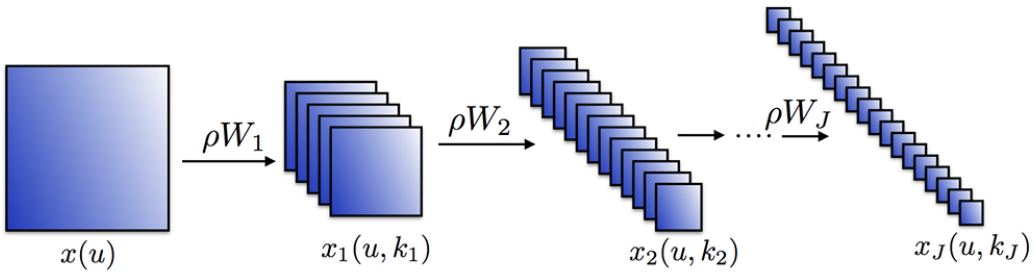


Figura 2.6: Esquema de una red neuronal convolucional [2]

Una red convolucional recibe una señal de entrada en dos dimensiones $x(u)$, en una capa interna, denotada como $x_j(u^{k_j})$, la profundidad j esta indexada por la misma variable de traslación u,

aunque usualmente se aplica un muestreo reducido y por un indice de canal k_j. Para calcular la capa x_j, se considera la capa x_{j-1} sobre la cual se aplica un operador lineal W_j, seguido por uno no lineal ρ

$$x_j = \rho(W_j x_{j-1}) \tag{2.25}$$

En la arquitectura de una CNN, W_j representa los pesos de las convoluciones, mientras que ρ denota la función de activación, que suele ser ReLU (2.10) o sigmoid (2.5). Es útil interpretar W_j como un conjunto de filtros convolucionales apilados. De esta manera, cada capa se puede visualizar como un mapa de características generado por estos filtros. Cada capa se construye a partir de la suma de las convoluciones aplicadas sobre la salida de la capa anterior:

$$x_j(u, k_j) = \rho(\sum_k (x_{j-1}(., k) * W_{j,k_j}(., k))(u)) \tag{2.26}$$

Donde $*$ corresponde al operador de convolución:

$$(f * g)(x) = \sum_{u=-\infty}^{+\infty} f(u)g(x - u) \tag{2.27}$$

El problema de optimización definido por una CNN es altamente no convexo. La no convexidad implica que la función de pérdida asociada al entrenamiento de la red tiene una superficie compleja con múltiples mínimos locales, picos y puntos de silla. En otras palabras, no posee una forma simple, como una parábola, donde el mínimo global pueda identificarse fácilmente. Esto se debe a la alta dimensionalidad del espacio de parámetros y a las interacciones no lineales entre las capas de la red, lo que dificulta que el optimizador alcance la solución óptima global. Para enfrentar esta complejidad, se utiliza el descenso de gradiente estocástico [25] para entrenar los parámetros de red W_j. Este método, al calcular el gradiente en subconjuntos de datos (mini-lotes), introduce aleatoriedad, lo que permite explorar de manera más eficiente el espacio de soluciones y superar mínimos locales y puntos críticos.

Una arquitectura de CNN se puede subdividir en las siguientes capas:

1. **Capa convolucional:** La capa convolucional es el núcleo de las arquitecturas de las redes neuronales convolucionales (CNN). Esta capa se compone de un conjunto de filtros, también conocidos como kernels, que operan sobre los datos de entrada. Cada kernel se caracteriza por tres parámetros: ancho (w), alto (h) y pesos ($W_{i,j}$). Mientras que el ancho

y el alto se definen al diseñar la arquitectura y permanecen fijos, los pesos se inicializan de forma aleatoria y se ajustan durante el proceso de entrenamiento.

Los filtros son matrices de tamaño $(h \times w)$ que se desplazan horizontal y verticalmente a lo largo de la información de entrada, realizando la operación de convolución entre los valores de la región actual de la entrada (x_j) y los pesos del kernel $(W_{i,j})$. Esta operación se ilustra en la Fig. 2.7.

Figura 2.7: Representación visual de la operación de convolución de los Kernels

Cabe mencionar que el tamaño de la salida de una convolución viene dada por la siguiente formula:

$$O = 1 + \frac{(N - 2P - K)}{S} \tag{2.28}$$

En donde, O representa el tamaño de la salida, N el tamaño de los datos de entrada, y K las dimensiones del filtro. S y P se refieren al stride y al padding, respectivamente, conceptos que se explicarán a continuación.

Existen otros conceptos clave que caracterizan la arquitectura de las CNNs:

- **Pesos compartidos:** Este concepto distingue a las CNNs de otras arquitecturas como las redes completamente conectadas (*fully connected*). Está directamente relacionado con el uso de kernels, ya que en una CNN se utiliza el mismo kernel, con sus respectivos pesos, para procesar toda la información de entrada. A diferencia de las redes completamente conectadas, donde cada neurona está conectada a la siguiente con un peso específico, en las CNNs se comparte un conjunto reducido de pesos a lo largo de toda la entrada. Esto reduce significativamente el número de parámetros de entrenamiento, disminuyendo así el costo computacional.
- **Stride:** El *stride* se refiere al avance del kernel a través de la información de entrada, es decir, cuántos píxeles se desplaza el filtro entre cada operación de convolución. Este parámetro suele ajustarse según el tamaño del kernel. Si se utilizan filtros de gran dimensión, es posible que muchos nodos se solapen entre capas, lo que podría causar problemas durante el entrenamiento. Como referencia, en la Fig. 2.7 se muestra un stride de 1.
- **Padding:** Una de las principales desventajas de las CNNs es la posible pérdida de detalle en los bordes de los datos de entrada, ya que estos se procesan una sola vez y solo con los bordes del filtro, lo que puede llevar a que no se consideren adecuadamente durante el entrenamiento. Para solucionar este inconveniente, se utiliza el *padding*, que consiste en extender los bordes de la entrada con ceros. Esto permite que el kernel se posicione de manera más centrada en los bordes, mejorando la captura de la información en esas áreas.

2. **Capa de Pooling:** La capa de polling, también conocida como capa *down-sampling* se utiliza para reducir la dimensión de la salida en comparación con la entrada, manteniendo al mismo tiempo la información más relevante. En esta capa, se aplica una operación de *pooling*, de la cual existen tres tipos: *max polling, min polling* y *avg pooling*. De estos, *max pooling* es el mas utilizado.

   La esencia de esta operación consiste en dividir la información de entrada en regiones rectangulares predefinidas y, dentro de cada región, se selecciona un valor representativo. En el caso de *max pooling*, la salida corresponde al valor máximo de la región; en *min*

pooling, el valor mínimo; y en *avg pooling*, se calcula el promedio de todos los valores involucrados.

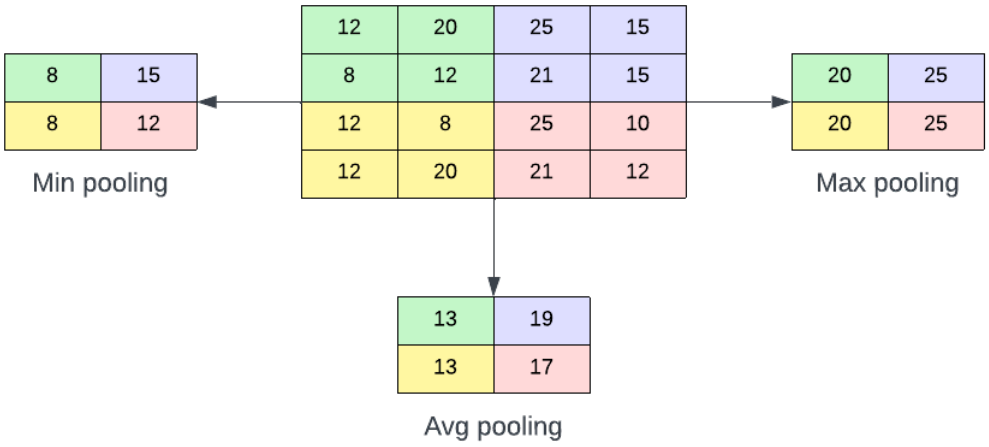


Figura 2.8: Esquema visual de pooling

En la Fig. 2.8, se muestra una representación de los distintos tipos de *pooling*. En este ejemplo, se utilizó una región de 2x2 que se desplaza con un *stride* de 2.

3. **Capa de función de activación:** Esta es la capa que sigue a al convolución. Como se detalló en secciones anteriores, las funciones de activación cumplen el rol de cambiar el valor de la salida acorde a ciertos parámetros, ademas le entrega una no linealidad al modelo, la cual es fundamental para el modelo para que este sea capaz de resolver problemas complejos.

   Dentro de las funciones de activación, la más popular para la arquitectura de CNN es Relu, esto debido a que este tipo de función es mas sencilla comparada a otras como sigmoide y tanh lo que da ventajas para reducir costos computacional. Ademas, funciones como las ya mencionadas suelen tener problemas con *back propagation* por el ya mencionado *vanishing gradient*. Finalmente, Relu genera una representación más dispersa porque un gradiente cero produce un valor completamente nulo, caso contrario para sigmoide tanh, lo que puede ser contraproducente durante el entrenamiento.

4. **Capa fully connected:** Como se vio en secciones anteriores, una capa *fully connected* representa que cada neurona de la capa i esta conectada con todas las neuronas de la capa $i+1$. Esta capa no se encuentra en todas las arquitecturas de CNN debido a que aumenta drásticamente el costo computacional de esta, sino que por lo general, se encuentra en los problemas relacionados a clasificación.

2.7.3.1. Campo Receptivo

El campo receptivo (*receptive field*) se define como la región de la entrada original que influye en una neurona particular de una capa en la red. En otras palabras, es el área de la imagen que "ve" una neurona y que afecta directamente su activación en la red.

En una CNN, las capas convolucionales aplican filtros (o kernels) que procesan pequeñas áreas locales de la entrada. Inicialmente, en la primera capa convolucional, el tamaño del campo receptivo de una neurona es equivalente al tamaño del filtro utilizado. Por ejemplo, si el filtro es de 3x3 píxeles, el campo receptivo inicial será de 3x3. Sin embargo, a medida que la información se propaga a través de capas sucesivas, el campo receptivo de las neuronas en capas más profundas se expande, abarcando regiones más grandes de la imagen original. Este crecimiento ocurre porque cada neurona en una capa posterior recopila información procesada por varias neuronas de capas anteriores, las cuales, a su vez, han sido influenciadas por diferentes partes de la imagen.

En el caso de redes con múltiples ramas convolucionales, como el caso de las arquitecturas MSNet (Fig. 3.1), el campo receptivo total (RF) se puede calcular como la suma de los campos receptivos individuales de cada rama (RF_b). Esto se expresa en la siguiente ecuación:

$$RF = \sum_{b=0}^{n_b-1} RF_b \tag{2.29}$$

Donde n_b representa el número total de ramas. El campo receptivo para cada rama (RF_b) esta definido de la siguiente manera:

$$RF_b = \begin{cases} 1 + d_b(ks-1)2^b & \text{si } b = 0 \\ d_b(ks-1)2^b & \text{si } b \neq 0 \end{cases} \tag{2.30}$$

En esta fórmula, k_s representa el tamaño del kernel de las capa convolucionales que se supone constante para todas las capas de la red, d_b indica el factor de dilatación asociado a la rama b y el término 2^b refleja el crecimiento exponencial del campo receptivo a medida que la red se profundiza.

El campo receptivo es crucial en el diseño de CNN, ya que su tamaño determina la cantidad de información que una neurona puede procesar de la imagen original. Un campo receptivo muy pequeño en las capas profundas podría limitar la capacidad de la red para captar patrones

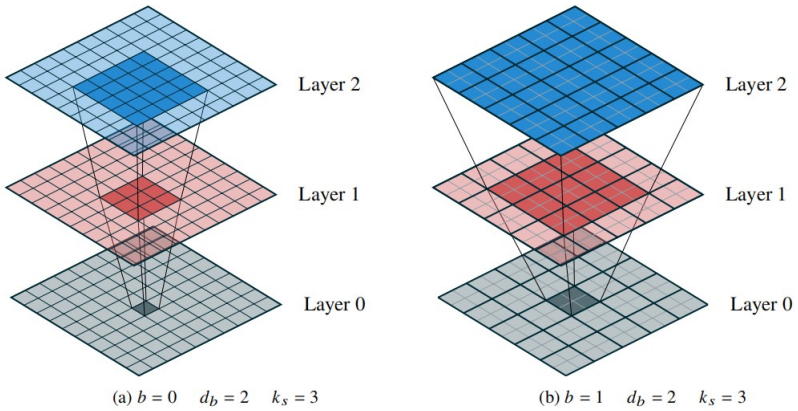


Figura 2.9: Visualización del campo receptivo en la propagación de información [3]

globales, mientras que un campo receptivo demasiado grande en las capas superficiales podría diluir la especificidad de las características locales que la red debe aprender. Por lo tanto, la correcta configuración del campo receptivo a través del uso adecuado de filtros, stride y capas de pooling es esencial para garantizar que la red capture tanto detalles locales como patrones globales.

2.7.3.2. Regularización de una CNN

Cuando se entrena un modelo de red neuronal convolucional (CNN), uno de los principales desafíos es encontrar el equilibrio adecuado entre la capacidad del modelo y la complejidad de los datos. En este contexto, los conceptos de subajuste (underfitting) y sobreajuste (overfitting) son fundamentales para entender el rendimiento del modelo.

El sobreajuste ocurre cuando el modelo logra ajustar perfectamente los datos de entrenamiento, incluyendo el ruido y las variaciones irrelevantes. Aunque esto puede parecer positivo a primera vista, en realidad significa que el modelo no generaliza bien y falla al aplicarse a datos nuevos o de prueba. En CNN, esto suele suceder cuando se utilizan redes excesivamente profundas o complejas, o cuando no se aplican técnicas de regularización adecuadas. En contraste, el subajuste se presenta cuando el modelo es incapaz de capturar los patrones subyacentes en los datos de entrenamiento. Esto da lugar a un rendimiento deficiente tanto en el conjunto de entrenamiento como en el de prueba. En CNN, esto puede deberse a una arquitectura muy simple, insuficiente número de capas o filtros, o a un entrenamiento incompleto.

La Fig.2.10 ilustra gráficamente los tres escenarios, en donde la primera gráfica (underfitting) el modelo es demasiado simple y no logra ajustarse a las tendencias principales. En la gráfica central (balanced) se alcanza un ajuste adecuado, donde el modelo capta la estructura de los

datos omitiendo el ruido de los datos. En la última, el modelo es demasiado complejo lo que genera que se adapte excesivamente a los datos, perdiendo la capacidad de generalización.

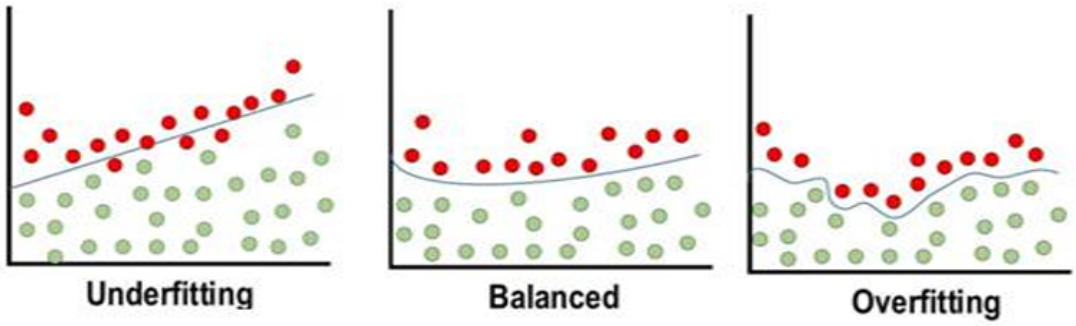


Figura 2.10: Regularización de una CNN [4]

Existen diversos métodos para mitigar tanto el sobreajuste como el subajuste. A continuación, se detallan algunos de los más utilizados:

- **Dropout:** Este método consiste en desactivar de manera aleatoria ciertas neuronas durante el proceso de entrenamiento. Como resultado, estas neuronas no participan en las fases de *forward propagation* ni *backpropagation*, lo que ayuda a evitar la dependencia excesiva del modelo en un conjunto reducido de neuronas [26].
- **Drop-weight:** Este método funciona de manera similar al *dropout*, pero en lugar de desactivar neuronas, se eliminan aleatoriamente las conexiones (pesos) entre ellas durante el entrenamiento.
- **Batch-normalization:** Este método normaliza las activaciones de una capa de la red para que tengan una media cercana a 0 y una varianza cercana a 1 antes de ser transmitidas a la siguiente capa [27]. La normalización ocurre en pequeños lotes (*batches*) de datos y puede describirse en los siguientes pasos:

  1. Cálculo de la media y la varianza del *batch* que pasa por una capa de la red.
  2. Normalización de los datos del *batch* utilizando los valores obtenidos.
  3. Aplicación de una escala y un desplazamiento (entrenables) para que el modelo conserve su capacidad de representación del problema.

  Las ventajas del uso del método incluyen:

  - Acelera el entrenamiento al permitir el uso de tasas de aprendizaje más altas, lo que mejora la convergencia del modelo.

- Mejora la estabilidad del modelo, mitigando el problema del *vanishing gradient*.
- Reduce la sensibilidad a la inicialización de los pesos, evitando problemas relacionados con una mala elección de los mismos.

2.7.3.3. Tipos de arquitecturas de CNN

Las redes neuronales convolucionales (CNN) pueden subdividirse en tres categorías principales, las cuales se diferencian según la forma en que se construye su arquitectura y la función específica que cumplen:

1. **Clasificación:** Este tipo de arquitectura está diseñada principalmente para tareas de clasificación de imágenes, donde el objetivo es asignar una etiqueta o clase a una imagen dada, como se observa en Fig. 2.11. Estas arquitecturas son fundamentales en aplicaciones de inteligencia artificial que requieren identificar y clasificar objetos dentro de imágenes, como la recuperación de información visual, el seguimiento de objetivos en tiempo real, y el análisis médico para el diagnóstico de enfermedades a partir de imágenes. Entre las arquitecturas más reconocidas de este tipo se encuentran VGG-16 [28] y ResNet [20].

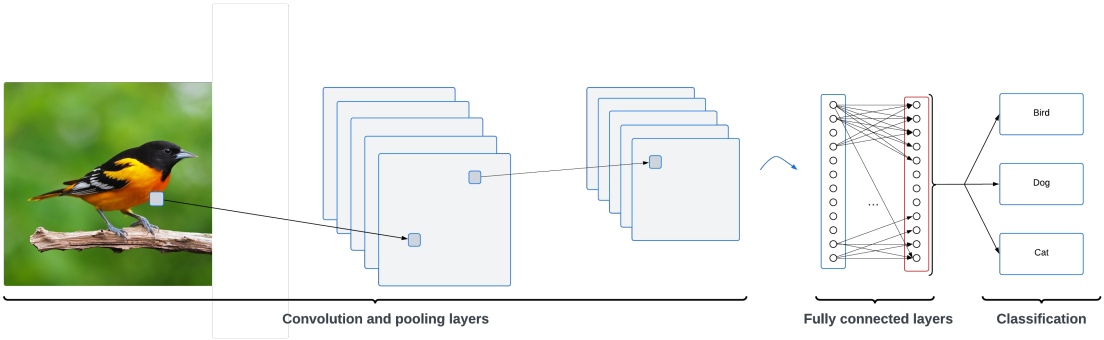


Figura 2.11: Esquema de arquitectura de clasificación de una CNN

2. **Detección:** Las arquitecturas de detección se centran en identificar la ubicación de objetos específicos dentro de una imagen, además de clasificarlos. Esto implica no solo predecir qué objeto está presente, sino también su posición mediante el uso de *bounding boxes* (cajas delimitadoras). Este enfoque es crucial en aplicaciones como la conducción autónoma, vigilancia, y la detección de objetos en tiempo real. Ejemplos conocidos de arquitecturas de detección son YOLO (You Only Look Once) [29] y Faster R-CNN [30], que destacan por su capacidad de realizar detección rápida y precisa.te tipo de arquitectura esta diseñada principalmente para las tareas de clasificación de objetos en imágenes y su localización en esta misma.

3. **Segmentación:** Las arquitecturas de segmentación buscan clasificar cada píxel de una imagen en una categoría específica, lo que resulta en una representación mucho más detallada que la detección por cajas delimitadoras. La segmentación es especialmente útil en tareas como la segmentación de imágenes médicas, la visión por computadora para vehículos autónomos y la segmentación semántica en imágenes de alta resolución. Arquitecturas populares en este campo incluyen UNet [31], MSNet [32] y Mask R-CNN [33], las cuales son ampliamente utilizadas por su capacidad de realizar segmentación precisa y detallada. Cabe mencionar que este tipo de arquitecturas suelen utilizar un modelo conocido como *encoder-decoder*, el cual se detallará en las próximas secciones.

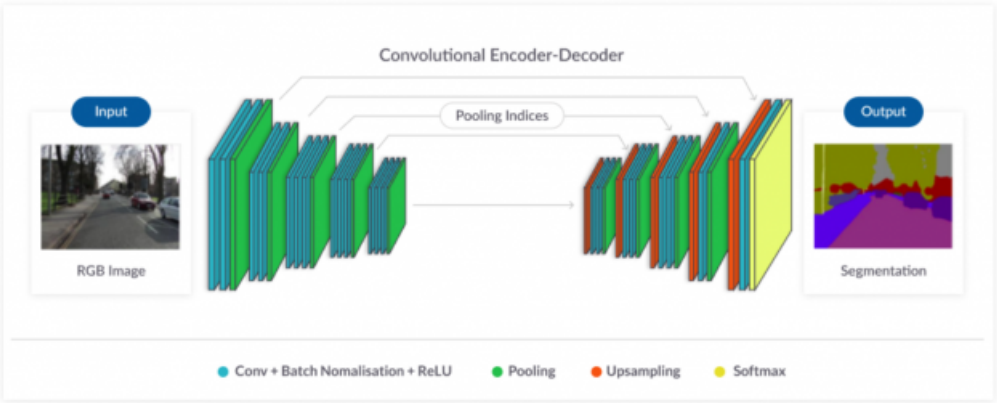


Figura 2.12: Esquema de arquitectura de segmentación de una CNN [5]

2.7.3.4. Encoder-Decoder

La arquitectura Encoder-Decoder es ampliamente utilizada en tareas de segmentación de imágenes y otras aplicaciones de visión por computadora. Se compone de dos partes principales [34]:

1. **Encoder:** Es la primera parte de la red, encargada de recibir la entrada y procesarla mediante operaciones de convolución y *pooling*, previamente detalladas. El objetivo de esta etapa es extraer las características más relevantes de la entrada, comprimiendo la información en un espacio de menor dimensionalidad. Es decir, el *encoder* reduce la dimensionalidad de la entrada al capturar su esencia sin perder información crucial.

2. **Decoder:** La segunda etapa de la arquitectura recibe la representación comprimida del *encoder* y la expande hasta alcanzar una dimensionalidad específica. Durante este proceso, la información es procesada para resolver un problema particular, como la segmentación de una imagen. El *decoder* utiliza principalmente la convolución transpuesta, un operador

que, al igual que la convolución normal, emplea un *kernel*. Sin embargo, a diferencia de la convolución estándar, que reduce la dimensionalidad, la convolución transpuesta incrementa el tamaño de la representación, permitiendo la reconstrucción de la imagen original o la generación de una salida a mayor escala.

Este enfoque permite que la red aprenda a mapear entradas complejas a salidas detalladas, manteniendo la estructura espacial necesaria para tareas de precisión como la segmentación.

2.8. Physics Informed Neural Networks

Las redes *Physics Informed Neural Networks* PINNs, son un tipo de redes neuronales artificiales diseñada para resolver ecuaciones diferenciales parciales (PDEs) mediante la incorporación de principios físicos directamente en el entrenamiento del modelo. Mientras una arquitectura de red tradicional se entrena mediante datos de entrada y salida específicos [7, 35], PINNs incorpora conocimiento de las leyes físicas características del problema en la función de perdida, lo que permite resolver problemas en donde los datos de entrenamiento son escasos o caros de obtener. Este método ha ganado bastante popularidad en estos últimos años, siendo utilizado para resolver problemas en diversas areas de la ingeniería y la física, tales como dinámica de fluidos, mecánica de sólidos, transferencia de calor, y electromagnetismo [36, 37, 38, 3], además de trabajos teóricos en los que se estudia la convergencia y errores del método [39, 40, 41, 42]. A pesar de los grandes avances de estos últimos años, al ser un tema nuevo aun queda mucho por investigar para lograr una estandarización de este modelo [43]. A continuación, se detallará en que consiste el método de PIINs.

Generalmente, se considera que una PDEs tiene la siguiente forma [44]:

$$u_t + \mathcal{N}[u] = 0,\ t \in\ [0,t],\ x\ \in\ \Omega \tag{2.31}$$

con las condiciones iniciales y de contorno de:

$$\begin{aligned} \mathcal{D}[u] &= f(x),\ x \in \Omega \\ \mathcal{B}[u] &= g(x),\ x \in \partial\Omega \end{aligned} \tag{2.32}$$

En donde $\mathcal{N}[\cdot]$ es un operador lineal o no lineal, $\mathcal{B}[\cdot]$ es el operador de borde que puede corresponder a Dirichlet, Neumann, Robin o condición de borde periódico. Adicionalmente, u describe

la solución desconocida de la PDE.

De forma general, se procede a buscar una representación de la solución real del problema $u(t, x)$ utilizando una red neuronal para asi obtener una aproximación $u_\theta(t, x)$, en donde θ, como se vio en secciones anteriores, denota todos los hiperparametros de la red (pesos y sesgos). Esto permite definir el residual de una PDE como:

$$\mathcal{R}_\theta(t, x) = \frac{\nabla u_\theta}{\nabla t}(t_r, x_r) + \mathcal{N}[u_\theta](t_r, x_r) \tag{2.33}$$

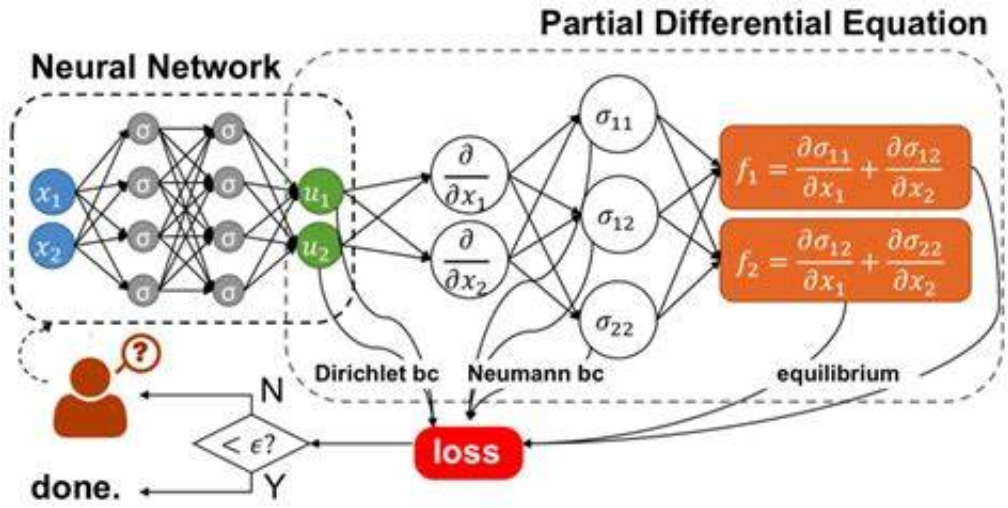


Figura 2.13: Esquema de método PINNs [45]

El método de PINNs se puede diferenciar en distintos modelos: primero tenemos los relacionados a la arquitectura de la red neuronal, en donde el mas utilizado es MLP [44], aunque existe literatura del uso de PINNs con arquitecturas de CNN [46]. Por el otro lado, se puede caracterizar el modelo según como este adopta la función de perdida $\mathcal{L}$ y la integra en la red neuronal. Algunos de estos modelos se detallan a continuación:

- **Métodos basados en puntos de colocación:** Utilizan los residuales de la ecuación 2.32 en puntos del dominio para formar una función de pérdida. Este es uno de los mas utilizados debido a su sencilla implementación y buenos resultados [7, 35]. Adicionalmente, se han realizado avances con la descomposición de dominios para la resolución de problemas de dominios complejos (por ejemplo con una interfaz y dos medios con propiedades físicas distintas) utilizando una subred neuronal por cada subdominio. Entre estos se encuentra: *Conservative Physics Informed Neural Networks* (CPINNs) [47], *Extended Physics Informed Neural Networks* (XPINNs) [48], *Augmented Physics Informed Neural Networks* (APINNs) [49] y *Distributed Physics Informed Neural Networks* (DPINNs) [50].

- **Métodos basados en la formulación variacional:** Utilizan una formulación variacional de la ecuación 2.32 para formar una función de perdida. Este método dio origen a una rama de PINNS llamada *Variational Physics Informed Neural Networks* (VPINNs) [45] y *Deep Ritz Method* [51]
- **Métodos basados en la formulación integral:** Utilizan la formulación integral en la frontera de la ecuación 2.32 para formar la función de pérdida. Gracias a este modelo se formo *Boundary Integral Neural Networks* (BINet) [52].
- **Métodos basados en el aprendizaje de operadores:** Este método consiste en generar un modelo de red neuronal como un aproximador de operadores no lineales con el objetivo de resolver PDEs. Una arquitectura clasica de este modelo es *Deep Operator Network* (DeepONet) [53]

2.8.1. Métodos Basados en Puntos de Colocación

Este método es uno de los más comunes para PINNS debido a su simplicidad. Este consiste en utilizar el residual de la ecuación diferencial directamente en la función a minimizar.

Con esto, la función de perdida puede ser descrita como:

$$\mathcal{L}(\theta) = w_{\partial\Omega}\mathcal{L}_{\partial\Omega}(\theta) + w_{\Omega}\mathcal{L}_{\Omega}(\theta) + w_{data}\mathcal{L}_{data}(\theta) \tag{2.34}$$

En donde:

$$\begin{aligned}
\mathcal{L}_{\Omega}(\theta) &= \frac{1}{N_{\Omega}} \sum_{x_i \in \Omega} ||\mathcal{D}u_{\theta}(x_i) - f(x_i)||^2 \\
\mathcal{L}_{\partial\Omega}(\theta) &= \frac{1}{N_{\partial\Omega}} \sum_{x_i \in \partial\Omega} ||\mathcal{B}u_{\theta}(x_i) - g(x_i)||^2 \\
\mathcal{L}_{data}(\theta) &= \frac{1}{N_{data}} \sum_{x_i \in data} ||u_{\theta}(x_i) - u_{data}(x_i)||^2
\end{aligned} \tag{2.35}$$

En el conjunto de ecuaciones 2.35, $\mathcal{L}_{\Omega}$ corresponde al componente que calcula la perdida en el dominio de la ecuacion diferencial, $\mathcal{L}_{\partial\Omega}$ es el término asociado a las condiciones de borde y $\mathcal{L}_{data}$ es un termino que se incluye para considerar los datos disponibles u_{data} característicos del problema. Cabe mencionar que este último término no es estrictamente necesario para la

convergencia del modelo pero permite añadir de forma directa información experimental del problema. Ademas, en la ecuación 2.34, se observan los términos $w_{\partial\Omega}$, w_{Ω}, w_{data} los cuales corresponden a los pesos de cada función de perdida. Estos son utilizados para modificar la dominancia de cada función de perdida en el problema, asi se puede establecer que parámetros generan más importancia durante el entrenamiento. Cabe mencionar que una buena elección de estos es fundamental para la convergencia del modelo y que a pesar de que por lo general son fijos, existe la posibilidad de que puedan ser aprendidos durante el entrenamiento [54].

2.8.1.1. Descomposición de Dominios

Los métodos basados en puntos de colocación permiten una descomposición de dominios en regiones, a las cuales se les asocia diferentes redes neuronales para representar la solución en cada subdominio. De forma esquemática para dos dominios se tiene lo siguiente:

$$u \approx u_\theta = \begin{cases} \mathcal{N}_1(x, \theta^1);\ x \in \Omega_1 \\ \mathcal{N}_2(x, \theta^2);\ x \in \Omega_2 \end{cases} \tag{2.36}$$

Cada red neuronal $\mathcal{N}_1$ y $\mathcal{N}_2$ proporciona una aproximación de la solución u_θ^1 y u_θ^2 de sus respectivos subdominios Ω_1 y Ω_2. Cada red tiene sus propios conjuntos de hiperparametros θ^1 y θ^2 que se aprenden simultáneamente durante el entrenamiento en conjunto. Es importante destacar que a la función de perdida 2.34 se le añade el término $\mathcal{L}_\Gamma$ que corresponde a la condición de contorno en la interfaz. Por lo tanto, la función de pérdida para un caso con j subdominios se expresa como:

$$\mathcal{L}^j(\theta) = w_{\partial\Omega}\mathcal{L}^j_{\partial\Omega}(\theta) + w_\Omega \mathcal{L}^j_\Omega(\theta) + w_{data}\mathcal{L}^j_{data}(\theta) + w_\Gamma \mathcal{L}^j_\Gamma(\theta) \tag{2.37}$$

En donde la el término de $\mathcal{L}_\Gamma$ se descompone como:

$$\mathcal{L}^j_\Gamma(\theta) = \frac{1}{N_\Gamma} \sum_{x_i \in \Gamma} ||\mathcal{C}_j u^j_\theta(x_i) - \overline{\mathcal{C}u_\theta}(x_i)||^2 \tag{2.38}$$

En este término se aplica el operador $\mathcal{C}$ sobre u_θ en la interfaz de todos los subdominios. Este operador puede entregar continuidad en la función, su gradiente o su residual, esto acorde a las características propias del problema a resolver. La solución de cada region en su interfaz puede aproximarse al promedio entre las redes que comparten esa interfaz $\overline{\mathcal{C}u_\theta}$.

Este método tiene la ventaja de que puede ser aplicado en problemas más complejos que requieran la separación entre dos o mas medios aplicando una red especialmente entrenada para cada region sin la necesidad de aumentar drásticamente el costo computacional.

2.8.2. Physics Informed Neural Networks en base a Redes Convolucionales

Como se menciono en anteriormente, las arquitecturas clásicas para el método de PINNS suelen ser las *fully connected* lo cual puede representar una dificultad para problemas a gran escala debido a que aumenta considerablemente el costo computacional. Una de las alternativas que ha surgido ha sido utilizar como base las redes neuronales convolucionales para formar la arquitectura de PINNs. Esto trae grandes ventajas ya que permite el aprendizaje de *extremo a extremo* basado en imágenes y pueden generar directamente campos de solución sobre todo el dominio en lugar de producir salidas puntuales [55]. Específicamente, el campo de solución $u(x, \mu(x))$ de un dominio rectangular puede ser aproximado como:

$$u(\mathcal{X}, \mu(\mathcal{X})) = u^{cnn}(\mathcal{X}, \mu(\mathcal{X}); \Gamma) \tag{2.39}$$

Donde $\mathcal{X}$ representa un conjunto de puntos de tamaño n_g, distribuidos de manera uniforme, y $\Gamma = \gamma^{l}{}_{l=1}^{n_l}$ es el conjunto de filtros entrenables de la red. En la entrada de la red, además de la información de la malla, se incluyen los parámetros correspondientes, denotados como $\mu(\mathcal{X})$. Para obtener la representación de la solución, se aplican múltiples capas convolucionales utilizando los filtros Γ y un operador no lineal $\phi(\cdot)$ (en este caso, las funciones de activación previamente mencionadas). La siguiente ecuación describe la operación en la capa l de la red:

$$g^l(x) = \phi((g^l * \gamma^l)(x)), \ x \in \mathcal{X}^l \tag{2.40}$$

En donde $*$ representa la operación de la convolución vista en la ecuación 2.27

Este tipo de arquitecturas suelen entrenarse utilizando datos de entrenamiento, por lo cual la función de perdida consistiría en minimizar el siguiente conjunto de datos: $\{u_i; u_i^{data}\}_{i=1}^{n_d}$

$$\min_{\Gamma} = \sum_{i=1}^{n_d} ||u^{cnn}(\mathcal{X}, \mu_i; \Gamma) - u_i^{data}(\mathcal{X})||_{\Omega_p} \tag{2.41}$$

Para un correcto entrenamiento de la red se necesita un conjunto de datos u^{data} muy grande para evitar problemas de ajuste, lo cual suele ser muy costoso de obtener, por lo cual de

forma alternativa se incluyen funciones de perdida en relación a las ecuaciones que gobiernan el problema físico y sus condiciones de borde:

$$\begin{aligned} \min_{\Gamma} &= \sum_{i=1}^{n_d} ||\mathcal{F}(u^{cnn}(\mathcal{X}, \mu_i; \Gamma), \nabla u^{cnn}(\mathcal{X}, \mu_i; \Gamma), \nabla^2 u^{cnn}(\mathcal{X}, \mu_i; \Gamma), ...)||_{\Omega_p} \\ \min_{\Gamma} &= \sum_{i=1}^{n_d} ||\mathcal{B}(u^{cnn}(\mathcal{X}, \mu_i; \Gamma), \nabla u^{cnn}(\mathcal{X}, \mu_i; \Gamma), \nabla^2 u^{cnn}(\mathcal{X}, \mu_i; \Gamma), ...)||_{\partial\Omega_p} \end{aligned} \tag{2.42}$$

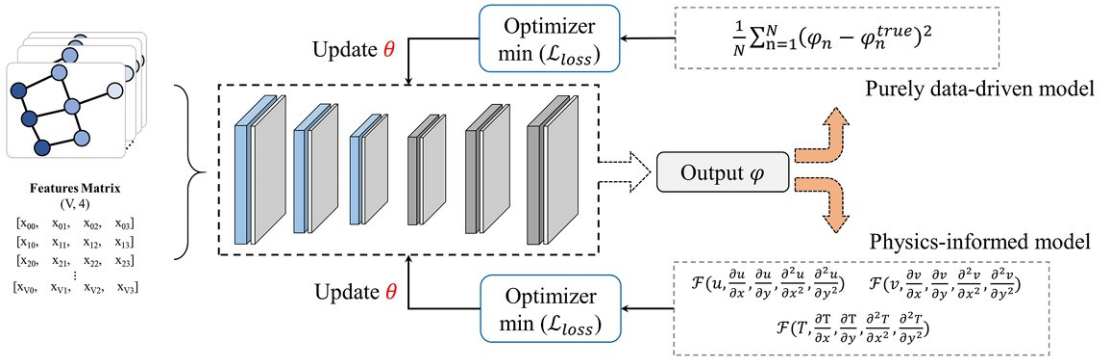


Figura 2.14: Esquema de PINNs con arquitectura de CNN [6]

Este método ha demostrado resultados prometedores para resolver EDPs sin la necesidad de grandes conjuntos de datos de entrenamiento [3, 6, 56], generando modelos que, una vez entrenados, pueden resolver diversas situaciones que compartan las mismas condiciones de dominio y de frontera utilizadas en el entrenamiento [3]. Además, existen trabajos que extienden este modelo para su aplicación en mallados no uniformes, lo cual permite abordar problemas de mayor complejidad [57]

2.9. Ecuación de Poisson

La ecuación de Poisson es una ecuación diferencial parcial elíptica ampliamente utilizada para describir diversos fenómenos físicos en campos como la difusión, electrostática, la gravitación, la mecánica de fluidos, y la teoría del potencial, entre otros. Esta establece una relación entre un potencial escalar y una fuente distribuidora y su forma general es:

$$-\Delta\phi = \sigma(X) \tag{2.43}$$

En donde Δ representa el operador Laplaciano descrito por la ecuación 2.44, ϕ es la incognita que describe el potencial y $\sigma(X)$ es el término fuente o distribución de cargas de dicho potencial.

$$\Delta f = \nabla^2 = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2} \tag{2.44}$$

Esta ecuación se manifiesta como un modelo fundamental en múltiples fenómenos físicos, por ejemplo, en el contexto de la difusión, describe el comportamiento estacionario de una cantidad escalar $\phi(x)$, como la concentración de un soluto, en presencia de una fuente externa $S(x)$, dando lugar a la forma $\nabla^2\phi = -S(x)/k$, donde k representa la difusividad del medio. En electrostática, a partir de la formulación diferencial de la Ley de Gauss y considerando un campo magnético estacionario, se obtiene la relación $\nabla^2\phi = -\rho/\epsilon_0$, con ρ denotando la densidad volumétrica de carga eléctrica. De modo análogo, en el ámbito gravitacional, el potencial generado por una distribución de masa $\rho(x)$ se rige por la ecuación $\nabla^2\phi = 4\pi G\rho(x)$, derivada de la ley de Gauss para la gravitación. Estos tres ejemplos evidencian el carácter unificador de la ecuación de Poisson en la modelación de sistemas físicos gobernados por interacciones a distancia.

Existen diversos métodos para resolver la ecuación de Poisson dependiendo del dominio y las condiciones de contorno. Estos métodos pueden clasificarse en dos grupos principales: métodos analíticos y métodos numéricos. Los métodos analíticos son aquellos que buscan una solución exacta a la ecuación, mientras que los métodos numéricos son aproximaciones a la solución de la ecuación utilizando técnicas computacionales. Los principales métodos analíticos incluyen la separación de variables, transformada de Fourier y la función de Green. Obtener los resultados de estos métodos suele representar un gran desafío, especialmente para dominios de geometrías complejas y condiciones de borde no homogéneas. Por otro lado, los métodos numéricos principales son diferencias finitas, elementos finitos y volúmenes finitos. Estos métodos son más flexibles y su implementación es más sencilla, aunque representan un mayor costo computacional ademas de estar sometidos a problemas de convergencia y estabilidad para casos complejos.

2.9.1. Solución integral de la ecuación de Poisson mediante la función de Green

El teorema de Green establece que, para cualquier par de funciones suaves ϕ y ψ, la identidad es la siguiente [58]:

$$\int_V (\phi \nabla^2 \psi - \psi \nabla^2 \phi) d^3x = \oint_S \left[\phi \frac{\partial \psi}{\partial n} - \psi \frac{\partial \phi}{\partial n} \right] da \tag{2.45}$$

Por otro lado, el teorema de la divergencia nos dice que:

$$\int_V \nabla \cdot A, d^3x = \oint_S A \cdot n, da \tag{2.46}$$

Utilizando estos dos teoremas, se puede demostrar la siguiente identidad:

$$\Phi(x) = \frac{1}{4\pi\epsilon_0} \int_V \rho(x') G(x, x') d^3x' + \frac{1}{4\pi} \oint_S \left[G(x, x') \frac{\partial \Phi}{\partial n'} - \Phi(x') \frac{\partial G(x, x')}{\partial n'} \right] da' \tag{2.47}$$

La cual será utilizada para regularizar el entrenamiento de la red neuronal. Para mayor detalle de la derivación, revisar anexo A.

2.9.2. Ecuación de Poisson para dominios con regiones

Algunos problemas físicos requieren resolver potenciales para dominios que presentan regiones con diferentes propiedades físicas (por ejemplo, diferentes permitividades para el caso de electrostática), por lo cual es necesario considerar las interacciones que existen entre las subregiones.

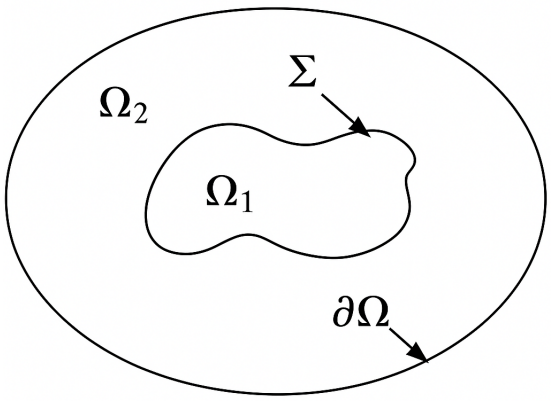


Figura 2.15: Esquema de un dominio con dos regiones con distintas propiedades físicas

A modo de ejemplo, se considerara un problema de electrostática con un dominio con dos regiones las cuales tienen distintas permitividad, denotadas como ϵ_1 y ϵ_2. El problema general

se puede plantear como:

$$\begin{aligned} -\nabla^2\phi_1(x) &= \frac{\rho_1}{\epsilon_1}; x \in \Omega_1 \\ -\nabla^2\phi_2(x) &= \frac{\rho_2}{\epsilon_2}; x \in \Omega_2 \end{aligned} \tag{2.48}$$

Para que existan una solución coherente, es necesario considerar las interacciones entre ambas regiones en la interfaz (Σ). Estas condiciones pueden ser derivadas de la ley de Gauss e indica lo siguiente:

1. Continuidad del potencial: El potencias debe ser continuo a lo largo de la interfaz, es decir:

$$\phi_1(x) = \phi_2(x); x \in \Sigma \tag{2.49}$$

2. Continuidad en el desplazamiento eléctrico: La derivada normal del potencial debe ser continua en la interfaz, es decir:

$$\epsilon_1 \left.\frac{\partial\phi_1}{\partial n}\right|_\Sigma = \epsilon_2 \left.\frac{\partial\phi_2}{\partial n}\right|_\Sigma \tag{2.50}$$

Capítulo 3

Metodología

3.1. Utilización de PINNs basadas en CNN para resolver la ecuación de Poisson

En esta sección se describe la construcción de un modelo de PINNs que emplea una arquitectura basada en CNN, capaz de reproducir las soluciones a la ecuación de Poisson para distintos dominios. Este enfoque se inspira en el trabajo desarrollado por Lionel Cheng et al. [3]. El objetivo es diseñar una red neuronal $\mathcal{N}$ que reciba información sobre el término del lado derecho de la ecuación de Poisson, asociado al caso R_{in} y que genere una salida ϕ_{out}, la cual es una aproximación de la solución real ϕ_{target}:

$$\phi_{target} \approx \phi_{out} = \mathcal{N}(R_{in}) \tag{3.1}$$

Para este modelo, es necesario definir parámetros relacionados con la geometría del dominio, específicamente sus dimensiones L_i y la cantidad de nodos a considerar en cada dirección n_i. Esto se debe a que, al ser una arquitectura de CNN (inicialmente diseñada para procesar imágenes), se requiere trabajar con mallas discretas en lugar de datos puntuales como en las MLP.

En general, el problema a resolver se plantea como:

$$\begin{aligned} -\nabla^2\phi &= f(R) \quad \text{en } \Omega \\ \phi &= g(R) \quad \text{en } \partial\Omega \end{aligned} \tag{3.2}$$

3.2. Función de Pérdida

Para el entrenamiento de la red, se pueden utilizar dos tipos de funciones de pérdida para los puntos del interior del dominio Ω, mientras que para el contorno se emplea una función de pérdida asociada a las condiciones de frontera. De manera general, la función de pérdida se puede escribir como:

$$\mathcal{L}(\theta) = w_{in}\mathcal{L}in(\theta) + w_{bc}\mathcal{L}_{bc}(\theta) \tag{3.3}$$

donde $\mathcal{L}_{in}$ y w_{in} representan, respectivamente, la función de pérdida y el peso asociado para los puntos internos del dominio. De manera análoga $\mathcal{L}_{bc}$ y w_{bc}, corresponden a la función de pérdida y el peso para la frontera.

En las redes CNN, se utiliza comúnmente una función de pérdida denominada *inside loss*, la cual evalúa nodo a nodo la diferencia entre la salida del modelo y una solución de referencia previamente conocida. Esta comparación requiere contar de antemano con los datos objetivo ϕ_{target}, que pueden provenir de soluciones numéricas, resultados experimentales o de una solución analítica, si está disponible. En el contexto del presente trabajo, esta función de pérdida viene definida por la Ecuación 3.5, donde se compara la salida de la red ϕ_{out} con la solución de referencia ϕ_{target}, lo cual puede interpretarse como una medida del residuo del modelo con respecto a la ecuación de Poisson discretizada. Sin embargo, en problemas de resolución de PDEs no siempre se dispone de conjuntos de ejemplos de gran tamaño, por lo que se propone usar la *Laplacian loss*, que utiliza el residual de la ecuación 3.2. Para las fronteras del dominio, se introducen la *Dirichlet loss* (cuando se conoce el valor del potencial en el borde) y la *Neumann loss* (en caso de que se conozca la derivada normal en el borde). A continuación, se presentan las funciones de pérdida para el caso tridimensional:

- *Laplacian Loss:*

$$\mathcal{L}_L(\phi_{out}) = \frac{L_x^2 L_y^2 L_z^2}{b_s(n_x-1)(n_y-1)(n_z-1)} \sum_{b,k,j,i} \left(\nabla^2 \phi_{out}^{b,k,j,i} + R_{in}^{b,k,j,i}\right)^2 \tag{3.4}$$

- *Inside Loss:*

$$\mathcal{L}_I(\phi_{out}, \phi_{target}) = \frac{1}{b_s(n_x-1)(n_y-1)(n_z-1)} \sum_{b,k,j,i} \left(\phi_{out}^{b,k,j,i} - \phi_{target}^{b,k,j,i}\right)^2 \tag{3.5}$$

- *Dirichlet Loss:*

$$\mathcal{L}_D(\phi_{out}, \phi_{bc}) = \frac{1}{b_s(2n_x + 2n_y + 2n_z - 4)} \sum_{b,k,j,i} \left(\phi_{out}^{b,k,j,i} - \phi_{bc}^{b,k,j,i}\right)^2 \tag{3.6}$$

- *Neumann Loss:*

$$\mathcal{L}_N(\phi_{out}) = \frac{1}{b_s(2n_x + 2n_y + 2n_z - 4)} \sum_{b,k,j,i} \left(\nabla\phi_{out}^{b,k,j,i} \cdot \hat{n}\right)^2 \tag{3.7}$$

En estas expresiones b_s es el *batch size*, n_x, n_y, n_z son el número de nodos en cada dirección, y *b, i, j, k* representan los índices para batch y nodos en *x, y, z* respectivamente.

Cabe destacar que se pueden emplear diversas combinaciones de estas funciones de pérdida, por ejemplo, *Laplacian - Dirichlet loss, Inside - Dirichlet loss, Laplacian - Neumann loss* o *Inside - Neumann loss*. Además, es posible asignar diferentes fronteras del dominio a condiciones de borde de tipo Neumann o Dirichlet. Para un correcto entrenamiento de la red, se agregan los pesos correspondientes a cada función de pérdida, los cuales permiten la ponderación de cada componente de la función de pérdida total para agregar o reducir la importancia de cada una de ellas en el entrenamiento de la red.

3.3. Arquitecturas de Red

La elección de la arquitectura CNN adecuada es fundamental para la resolución de este problema, ya que debe procesar cada *píxel* (o nodo, en este caso) con cuidado. Asimismo, la naturaleza de la ecuación de Poisson hace que el escalado espacial sea importante en la red. Con estas consideraciones, se seleccionan las arquitecturas UNet [31] y MSNet [32], ampliamente utilizadas en tareas de segmentación.

3.3.1. Arquitectura MSNet

MSNet se propuso originalmente para el análisis y la predicción de secuencias de video [32], y destaca por procesar grandes cantidades de datos sin divergir. Ha sido utilizada previamente en la resolución de PDEs en flujos de plasma, flujos turbulentos y ondas acústicas [3, 59, 60]. En general, se describe como un flujo paralelo de n_s canales, donde cada canal procesa una escala distinta de la información. Por lo general, se utilizan 3 canales:

- **High scale**: Procesa la información de entrada original y se optimiza para capturar detalles

finos.

- **Medium scale**: Procesa la información tras un proceso de *downsampling* moderado, capturando detalles de nivel medio.
- **Low scale**: Procesa la información después de un *downsampling* más intenso, capturando las características más generales.

Cada canal realiza convoluciones y aplica funciones de activación. La Fig. 3.1 muestra un diagrama de la arquitectura con $n_s = 3$ Se observa que el canal superior condensa la información a un cuarto de la resolución original, el canal medio a la mitad y el inferior emplea la información completa.

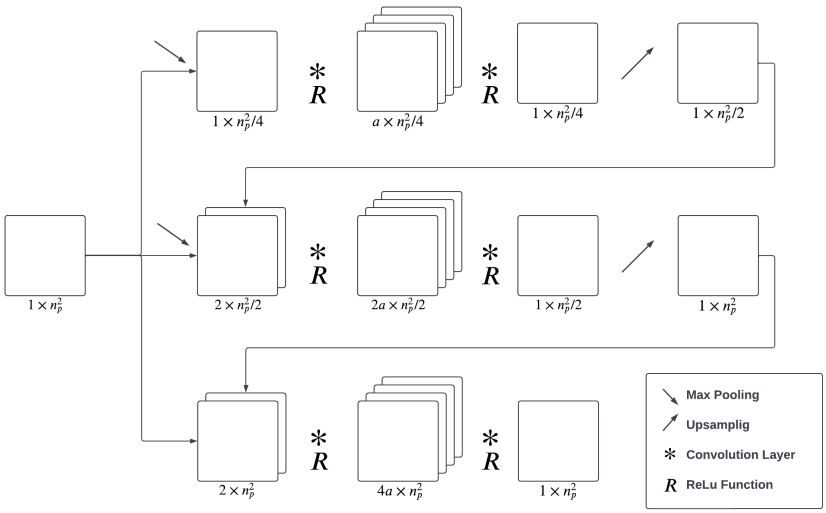


Figura 3.1: Diagrama de una arquitectura MSNet con $n_s = 3$

3.3.2. Arquitectura UNet

UNet se introdujo para la segmentación de imágenes en biomedicina [31], pero ha sido ampliamente adoptada por la comunidad de ML-CFD [61, 62]. Su nombre proviene de la forma en "U" de la arquitectura, que facilita la captura efectiva tanto del ontexto global como de los detalles espaciales. Similar a MSNet, consta de n_s etapas que reducen o aumentan la resolución de la información.

Pertenece a la familia *encoder-decoder*. El *encoder* comprime la información entrante para extraer el contexto global, mientras que el *decoder* recupera progresivamente los detalles que se perdieron durante la compresión. Además, se añaden *skip connections* entre capas homólogas del *encoder* y el *decoder*, lo que mejora sustancialmente la capacidad de recuperación de detalles

finos.

La Fig. 3.2 muestra un ejemplo de UNet con $n_s = 3$. Puede verse como un valle con cuatro partes:

- Descenso (*encoder*): Se comprime gradualmente la información.
- Fondo del valle (*bottleneck*): Punto de mayor compresión.
- Ascenso (*decoder*): Se recupera la resolución y los detalles perdidos.
- Puentes (*skip connections*): Vínculos entre capas del *encoder* y del *decoder*.

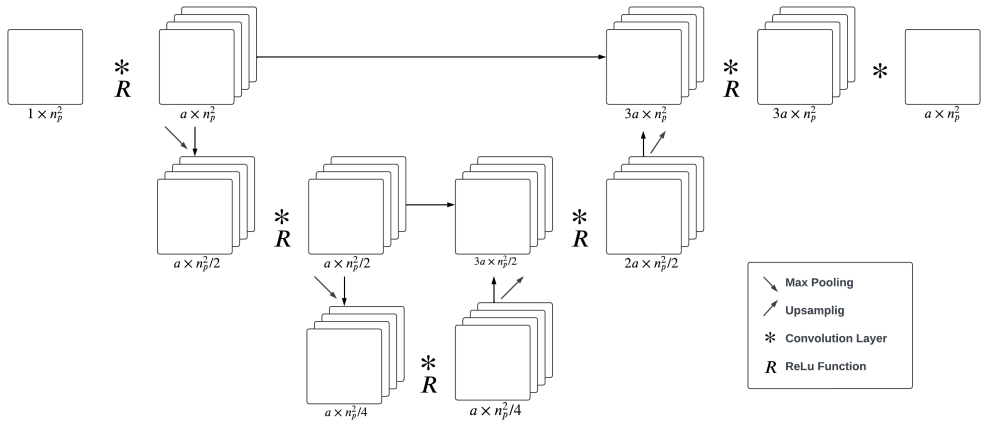


Figura 3.2: Diagrama de una arquitectura UNet con $n_s = 3$

La eficacia de ambas arquitecturas depende no solo de su estructura, sino también de la selección y el ajuste de hiperparámetros, tales como el tamaño del kernel, número de capas, número de convoluciones por etapa, etc. El ajuste se realiza generalmente por experimentación, buscando un equilibrio entre la capacidad de la red y la eficiencia computacional.

Un ejemplo de aruitectura que demostró ser efectiva en la resolución de problemas de Poisson es la denominada como *UNet4 rf200 ks3*, la cual consta de 200 filtros en la primera capa, un tamaño de kernel de 3x3 y 4 escalas.

3.4. Normalización

Para entrenar una red neuronal es importante que los valores de entrada y salida tengan magnitudes similares [63]. Por ejemplo, en el caso de predicción de imágenes, se suelen normalizar los datos en el rango [0, 1] [64].

En el presente trabajo, no se conoce *a priori* el orden de magnitud del potencial a resolver,

pero sí el de la información de entrada. Por ello, se normaliza la entrada a la red de la siguiente manera:

$$\phi_{out} = \hat{\mathcal{R}}_{\rangle\backslash} \text{ donde } \hat{R}_{in} = R_{in} \times \left| \frac{\phi_{out}}{R_{in}} \right|_{max} \tag{3.8}$$

Puesto que el valor máximo del potencial es desconocido, se propone emplear una aproximación analítica para obtener una cota razonable. La solución general de la ecuación de Poisson con condiciones de frontera de Dirichlet está relacionada con la función de Green G:

$$\phi(\mathbf{x}) = \frac{1}{4\pi\epsilon_0} \int \rho(x')G(x,x')dV' + \frac{1}{4\pi} \int \left(G\frac{\partial\phi}{\partial n'} - \phi\frac{\partial G}{\partial n'} \right) dS' \tag{3.9}$$

Asumiendo un dominio cuadrado L^2 con fronteras de Dirichlet iguales a 0, la función de Green es:

$$G(x,y,x',y') = \frac{16}{\pi L_x L_y} \sum_{n=1}^{+\infty} \sum_{m=1}^{+\infty} \frac{sin(\frac{n\pi x}{L_x})sin(\frac{n\pi x'}{L_x})sin(\frac{n\pi y}{L_y})sin(\frac{n\pi y'}{L_y})}{\frac{n^2}{L_x^2} + \frac{m^2}{L_y^2}} \tag{3.10}$$

Sustituyendo en la solución analítica:

$$\begin{aligned} \phi(x,y) = \sum_{n=1}^{+\infty} \sum_{m=1}^{+\infty} & \left[\frac{4}{L_x L_y} \int_{x',y'} sin\left(\frac{n\pi x'}{L_x}\right) sin\left(\frac{m\pi y'}{L_y}\right) R(x',y')dx'dy' \right] \\ & \times \frac{sin\left(\frac{n\pi x}{L_x}\right) sin\left(\frac{m\pi y}{L_y}\right)}{\pi^2 \left(\frac{n^2}{L_x^2} + \frac{m^2}{L_y}\right)} \end{aligned} \tag{3.11}$$

Para obtener una relación aproximada de ϕ respecto a R, se considera un potencial constante y solo el primer término de la sumatoria. Usando la propiedad de las funciones seno entre [-1, 1], se llega a:

$$\left| \frac{\phi}{R} \right|_{max} \leq \frac{1}{\left(\frac{\pi^2}{4}\right)\left(\frac{1}{L_x^2} + \frac{1}{L_y^2}\right)} \tag{3.12}$$

Definiendo $\alpha \leq 1$, se propone:

$$\left|\frac{\phi}{R}\right|_{max} = \frac{\alpha}{\left(\frac{\pi^2}{4}\right)\left(\frac{1}{L_x^2}+\frac{1}{L_y^2}\right)} \tag{3.13}$$

Para el caso de un dominio en tres dimensiones, considerando que la función de Green es:

$$\begin{aligned} G(x,x') =& \frac{32}{\pi L_x L_y L_z} \times \\ & \sum_{l,mn=1}^{+\infty} \frac{sin\left(\frac{l\pi x}{L_x}\right) sin\left(\frac{l\pi x'}{L_x}\right) sin\left(\frac{l\pi y}{L_y}\right) sin\left(\frac{l\pi y'}{L_y}\right) sin\left(\frac{l\pi z}{L_z}\right) sin\left(\frac{l\pi z'}{L_z}\right)}{\frac{l^2}{L_x^2}+\frac{m^2}{L_y^2}+\frac{n^2}{L_z^2}} \end{aligned} \tag{3.14}$$

Se establece la siguiente relación:

$$\left|\frac{\phi}{R}\right|_{max} = \frac{\alpha}{\left(\frac{\pi^2}{8}\right)^2\left(\frac{1}{L_x^2}+\frac{1}{L_y^2}+\frac{1}{L_z^2}\right)} \tag{3.15}$$

Durante el entrenamiento, esta relación (llamada *ratio max*) se utiliza para normalizar los datos de entrenamiento. Para esto, antes de pasar los datos por el modelo, se normalizan multiplicando por el factor, y al finalizar, se divide por el mismo factor para obtener la solución desnormalizada.

3.5. Datos para el entrenamiento

Para entrenar el modelo, se requiere un conjunto de datos que, en general, puede ser costoso de generar si se buscan soluciones exactas de la ecuación de Poisson. Por ello, se propone la técnica utilizada por Ali Ozbay [65], que consiste en generar distribuciones aleatorias en [-1, 1], sobre una malla uniforme, para luego interpolarlas mediante splines cúbicos y así obtener campos aleatorios suaves. Un ejemplo se ilustra en la figura siguiente:

Para evitar el *overfitting*, se recomienda emplear un gran número de ejemplos aleatorios y suficiente dispersión de los puntos de colocación iniciales. Con pocos puntos de colocación, la red podría memorizar patrones de entrenamiento, perjudicando su capacidad de generalización.

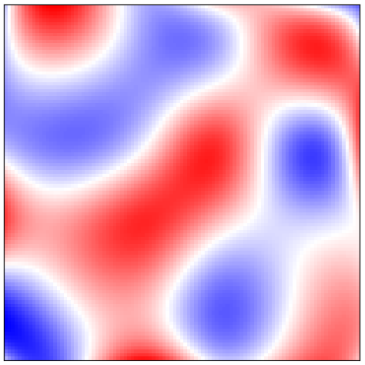


Figura 3.3: Campo de valores aleatorios

3.6. Ecuación de Poisson con Interfaces

Entre los casos abordados, se contempla la ecuación de Poisson en dominios que presentan regiones con diferentes propiedades físicas [66, 67]. Por ejemplo, en electrostática podrían existir regiones con distinta permitividadϵ.

La ecuación a resolver es:

$$\nabla \cdot (\epsilon((x)\nabla\phi(\mathbf{x}))) = f(\mathbf{x}) \tag{3.16}$$

Al discretizar con diferencias finitas (o volúmenes finitos), la dificultad surge cuando ϵ discontinuidades, pues el término $\nabla \cdot (\epsilon\nabla\phi)$ incluye la variación tanto de ϕ como de ϵ.

3.6.1. Caso 1D

El problema de permitividad variable puede generar dificultades de convergencia en la red, especialmente cuando se presentan grandes variaciones en este parámetro físico. Para enfrentar este desafío, se propone una técnica de discretización diseñada para manejar discontinuidades en la permitividad ϵ. Esta técnica se fundamenta en el caso unidimensional sobre el eje x y luego se generaliza a dimensiones superiores.

En el nodo i, la discretización del operador $\nabla \cdot (\epsilon\nabla\phi)$ se expresa como:

$$(\nabla \cdot (\epsilon\nabla\phi))_i \approx \frac{1}{\Delta x}\left((\epsilon\nabla\phi)_{i+\frac{1}{2}} - (\epsilon\nabla\phi)_{i-\frac{1}{2}}\right) \tag{3.17}$$

El flujo en la superficie $i + \frac{1}{2}$, se define como:

$$(\epsilon \nabla \phi)_{i+\frac{1}{2}} = \epsilon_{i+\frac{1}{2}} \left(\frac{\phi_{i+1} - \phi_i}{\Delta x} \right) \tag{3.18}$$

Asumiendo ϕ lineal entre nodos i e $i+1$ y que ϵ cambia abruptamente en $i + \frac{1}{2}$:

$$\phi(x) = \phi(x)_i + \left(\frac{\phi_{i+1} - \phi_i}{\Delta x} \right) (x - x_i) \tag{3.19}$$

Con $\Delta x = x_{i+1} - x_i$, Además, se establece:

$$\epsilon(x) = \begin{cases} \epsilon_i & \text{para } x_i \leq x < x_{i+\frac{1}{2}} \\ \epsilon_{i+1} & \text{para } x_{i+\frac{1}{2}} \leq x \leq x_{i+1} \end{cases} \tag{3.20}$$

Tal que $\phi(x)$ es lineal entre los nodos i e $i+1$:

$$\frac{d\phi}{dx} = \frac{\phi_{i+1} - \phi_i}{\Delta x} \tag{3.21}$$

Considerando los flujos en ambas superficies adyacentes al punto, se obtiene:

$$\begin{aligned} J_i &= -\epsilon_i \left(\frac{\phi_{i+\frac{1}{2}} - \phi_i}{\Delta x/2} \right) \\ J_{i+1} &= -\epsilon_{i+1} \left(\frac{\phi_i - \phi_{i+\frac{1}{2}}}{\Delta x/2} \right) \end{aligned} \tag{3.22}$$

Aplicando condición de continuidad $J_i = J_{i+1} = J$ en el nodo $i + \frac{1}{2}$, se obtiene:

$$\epsilon_i \left(\phi_{i+\frac{1}{2}} - \phi_i \right) = \epsilon_{i+1} \left(\phi_{i+1} - \phi_{i+\frac{1}{2}} \right) \tag{3.23}$$

Despejando el potencial $\phi_{1+\frac{1}{2}}$:

$$\phi_{i+\frac{1}{2}} = \frac{\epsilon_{i+1}\phi_{i+1} + \epsilon_i \phi_i}{\epsilon_i + \epsilon_{i+1}} \tag{3.24}$$

Reemplazando este término en el flujo J:

$$J = -\epsilon_i \left(\frac{2\epsilon_{i+1}(\phi_{i+1} - \phi_i)}{\Delta x(\epsilon_i + \epsilon_{i+1})} \right) = -e_{eff} \left(\frac{\phi_{i+1} - \phi_i}{\Delta x} \right) \tag{3.25}$$

En donde e_{eff} es la permitividad efectiva, concepto que se utilizará para definir la función de pérdida de la red y definida como:

$$\epsilon_{eff} = \frac{2\epsilon_i \epsilon_{i+1}}{(\epsilon_i \epsilon_{i+1})} \tag{3.26}$$

3.6.2. Configuración del problema

Esta investigación se centró en la resolución de la ecuación de Poisson para un dominio con dos regiones con diferente permitividad.

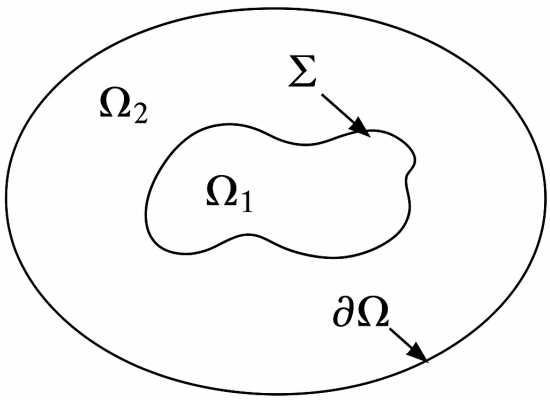


Figura 3.4: Dominio con dos regiones con interface Σ

Sea Ω_1 la región con permitividad ϵ_1 y Ω_2 la región con permitividad ϵ_2 y Σ la interfaz entre ambas regiones. La ecuación de Poisson a resolver es:

$$
\begin{aligned}
-\nabla^2\phi_1(\mathbf{x}) &= \frac{\rho_1}{\epsilon_1} \text{ en } \mathbf{x} \in \Omega_1 \\
-\nabla^2\phi_2(\mathbf{x}) &= \frac{\rho_2}{\epsilon_2} \text{ en } \mathbf{x} \in \Omega_2 \\
\phi &= g(\mathbf{x}) \text{ en } \partial\Omega \\
\phi_1 &= \phi_2 \text{ en } \Sigma \\
\epsilon_1 \cdot \nabla\phi_1 &= \epsilon_2 \cdot \nabla\phi_2 \text{ en } \Sigma
\end{aligned}
\tag{3.27}
$$

Para resolverlo, se utiliza descomposición de dominios: se definen submodelos de red, uno por cada región Ω_1 y Ω_2. Así la solución total ϕ se compone como:

$$
\phi = \begin{cases} u_\theta^1 = \mathcal{N}_1(\mathbf{x}, \theta^1) \text{ en } \mathbf{x} \in \Omega_1 \\ u_\theta^2 = \mathcal{N}_2(\mathbf{x}, \theta^2) \text{ en } \mathbf{x} \in \Omega_2 \end{cases}
\tag{3.28}
$$

En esta notación, el término u_θ^i representa la solución entregada por la red neuronal $\mathcal{N}_i$ del subdominio Ω_i compuesta por los hiperparametros θ_i. Para entrenar la red, se puede formular una función de pérdida a partir de la descomposición de dominios definida en 2.8.1.1, en donde $\theta = \theta_1 \cup \theta_2$.

$$
\mathcal{L}(\theta) = w_{\Omega_1}\mathcal{L}_{\Omega_1}(\theta_1) + w_{\Omega_2}\mathcal{L}_{\Omega_2}(\theta_2) + w_{\partial\Omega}\mathcal{L}_{\partial\Omega}(\theta_2) + w_\Sigma\mathcal{L}_\Sigma(\theta_1, \theta_2)
\tag{3.29}
$$

Cabe destacar que es importante que cada subdominio incluya el borde de interfaz para así determinar el error en estos puntos.

3.6.3. Función de perdida

Para el entrenamiento de este modelo es necesario agregar una nueva componente a la función de perdida, el cual aplicara el residual de las condiciones de interfaz a la red. Con esto en consideración, se puede escribir que la función de perdida para el modelo:

$$
\mathcal{L}(\theta) = w_{in}\mathcal{L}_{in}(\theta) + w_{bc}\mathcal{L}_{bc}(\theta) + w_{ib}\mathcal{L}_{ib}(\theta)
\tag{3.30}
$$

Siendo $\mathcal{L}_{ib}$ la función de perdida de la interfaz y w_{ib} su respectivo peso.

Para construir la función de perdida de interfaz, se tienen que considerar las condiciones de

continuidad y de derivada de la misma. Además, es necesario redefinir *Laplacial loss*, ya que para este caso, la permitividad no es constante, por lo cual se debe ajustar este parámetro físico en la ecuación.

- *Laplacian Loss:*

$$\mathcal{L}_L(\phi_{out}) = \frac{L_x^2 L_y^2 L_z^2}{bs(n_x-1)(n_y-1)(n_z-1)} \sum_{b,k,j,i} \left(\nabla(\epsilon_{eff}^{k,j,i} \cdot \nabla \phi_{out}^{b,k,j,i}) + R_{in} \right) \tag{3.31}$$

  En donde ϵ_{eff} es la permitividad efectiva definida en la ecuación 3.26.

- *Interface Loss:*

$$\begin{aligned} \mathcal{L}_I(\phi_{out_1}, \phi_{out_2}) = \frac{L_x^2 L_y^2 L_z^2}{bs(n_x-1)(n_y-1)(n_z-1)} \sum_{b,\Sigma} & \left(\phi_1^{b,\Sigma} - \phi_2^{b,\Sigma} \right)^2 + \\ & \left(\epsilon_{eff} \cdot \nabla \phi_1^{b,\Sigma} - \epsilon_{eff} \cdot \nabla \phi_2^{b,\Sigma} \right)^2 \end{aligned} \tag{3.32}$$

  En donde Σ corresponde a los puntos de colocación asociados a la interfaz entre regiones de diferente ϵ.

Cabe mencionar que para casos en los que se cuenta con data experimental o analítica, se puede utilizar la función de perdida *Inside Loss* en lugar de *Interface Loss*.

$$\mathcal{L}(\theta) = w_{in}\mathcal{L}(\theta) + w_{bc}\mathcal{L}_{bc} \tag{3.33}$$

En donde $\mathcal{L}_{in}$ corresponde a la función de perdida definida en la ecuación 3.5. Este cambio se debe a que los datos objetivos ya contienen información sobre la interfaz, por lo que se vuelve redundante para el modelo contener ambas funciones de perdida.

3.6.4. Arquitectura de Red

Inspirado en el modelo de XPINNs, se generan n submodelos que corresponden a la cantidad de regiones que tiene el dominio. Para ello, lo primero que se debe realizar es generar la malla regular que define el dominio completo y determinar los nodos que corresponden a las fronteras entre regiones. A continuación, se asignan los submodelos a cada región. Es decir, al proporcionar información del dominio al modelo, existirán varios submodelos, cada uno con sus hiperparámetros θ_i, encargados de procesar dicha información. Es importante destacar que, para este método, los nodos de interfaz deben pertenecer a ambos modelos asociados a las regiones

adyacentes. Esto es crucial para calcular correctamente la función de pérdida en la interfaz.

Con este método el entrenamiento de los distintos subdominos se realiza en paralelo, utilizando una de las combinaciones de funciones de perdida descritas previamente. Cabe mencionar que se realiza una única optimización para ambas redes. Una vez entrenado el modelo, este puede ser utilizado, lo que entregara información separada asociada a las distintas regiones del dominio, la cual puede acoplarse nuevamente para formar el dominio completo.

Para este trabajo se utilizaron submodelos idénticos entre si, de modo que todos tengan la misma capacidad computacional para poder representar cada región del problema dado, aunque esto se puede modificar para entregar a cada submodelo una cantidad de diferente de hiperparámetros acorde a la importancia o complejidad asociada a una region en específico.

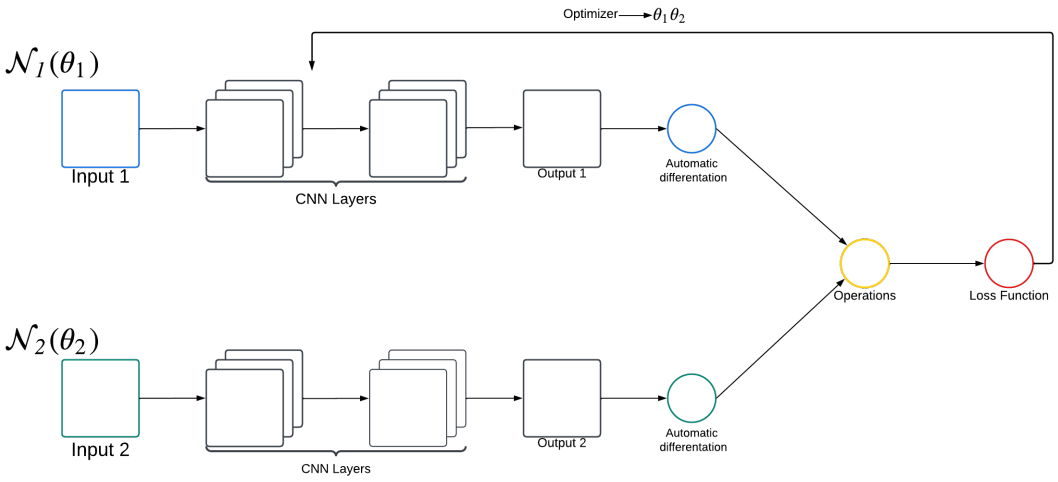


Figura 3.5: Esquema de dos submodelos para un dominio con interfaz

3.7. Validación de Resultados

Para validar los resultados obtenidos por la red neuronal existen diferentes parámetros que se pueden considerar. Uno de ellos es medir la función de pérdida durante el entrenamiento de la red. Esto va a permitir determinar si se están minimizando los residuales del Laplaciano y las condiciones de borde ademas de que es un buen indicador de cuando la red ha convergido. Sin embargo, este no puede ser el único parámetro a considerar, es necesario contar con una métrica adicional que permita evaluar la precisión del modelo. Una opción es calcular el error relativo entre la solución obtenida por la red y la solución analítica o numérica. Este error se puede definir como:

$$\text{Error Relativo} = \frac{\|\phi_{out} - \phi_{target}\|_2}{\|\phi_{target}\|_2} \tag{3.34}$$

donde $\|\cdot\|_2$ representa la norma L_2.

Otra métrica útil es el coeficiente de determinación R^2, que mide la proporción de la varianza en la variable dependiente que es predecible a partir de la variable independiente. Se define como:

$$R^2 = 1 - \frac{\sum_i (\phi^i_{target} - \phi^i_{out})^2}{\sum_i (\phi^i_{target} - \bar{\phi}_{target})^2} \tag{3.35}$$

donde $\bar{\phi}_{target}$ es el valor medio de ϕ_{target}.

Estas métricas permiten evaluar de manera cuantitativa la precisión del modelo y su capacidad de generalización a nuevos casos.

3.8. Código Implementado

Se desarrolló el código **PoissonSolverCNN** en Python [68], con el fin de facilitar la resolución de la ecuación de Poisson en dominios 2D y 3D, con o sin interfaces. El repositorio completo está disponible en [8]. Se distinguen tres etapas principales, configuradas mediante archivos `.yml`:

- Generación de datos: Se definen las dimensiones del dominio, la cantidad de nodos y la cantidad de puntos de colocación para generar datos aleatorios, u opcionalmente se ingresa una solución analítica o datos experimentales en un archivo `.npy`.
- Entrenamiento: Se elige la arquitectura (MSNet o UNet), las funciones de pérdida y sus pesos, y se especifica si existe interfaz. El modelo resultante se guarda como `.pth`.
- Resolución: Se carga el modelo entrenado, se le entrega un nuevo caso a resolver y se obtiene

Todas las funciones y clases utilizadas en el código se encuentran en archivos separados. A continuación, se presentan los algoritmos de los principales códigos:

Como se visualiza en los algoritmos, el código se entrena un modelo de red neuronal convolucional que es capaz de predecir campos en diferentes casos que comparten condiciones de borde y dominio. Es importante destacar que en todo el proceso de entrenamiento y resolución, los parámetros acorde al dominio y a la red deben coincidir para evitar errores en la resolución del

Algoritmo 3.1 Algoritmo de para generar datos de entrenamiento

Require: Archivo `.yaml` con dominio, número de nodos, puntos de colocación y datos n.
  for $i = 0$ hasta n **do**
    Generar la malla regular del dominio.
    Asignar a los puntos de colocación datos aleatorios.
    Interpolar cubicamente el campo de datos.
  end for
  Guardar datos en archivo `.npy`.

Algoritmo 3.2 Algoritmo de entrenamiento

Require: Archivo `.yaml` con arquitectura, funciones de perdida y pesos.
Require: Archivo `.npy` con datos de entrenamiento.
  Cargar datos de entrenamiento.
  Generar modelo con arquitectura definida e hiperparámetros iniciales aleatorios.
  for $i = 0$ hasta n **do**
    Calcular salida de la red $\phi_{out} = \mathcal{N}(R_{in})$
    Calcular $\partial_n \phi_{out}, \nabla \phi_{out}, \nabla^2 \phi_{out}$ con diferencia automática.
    Calcular función de perdida $\mathcal{L}(\phi, R_{in})$
    Actualizar los hiperparametros del modelo $\theta_k \to \theta_{k+1}$ con método de optimización
  end for
  Guardar modelo en archivo `.pth`.

Algoritmo 3.3 Algoritmo de resolución

Require: Archivo `.pth` con modelo entrenado.
Require: Archivo `.yaml` con problema a resolver.
  Cargar modelo entrenado.
  Generar dominio con problema a resolver.
  Calcular salida de la red $\phi_{out} = \mathcal{N}(R_{in})$
  Archivo `.npy` ϕ_{out}

problema.

El uso del código es bastante sencillo, un ejemplo se observa a continuación:

Código 3.1: Ejemplo de uso básico del codigo PoissonSolverCNN

```
cd ./folder/to/dataset
python random_data.py -c dataset.yml
cd ./folder/to/train
python train.py -c train.yml
cd ./folder/to/solve
python solve.py -c solve.yml
```

Para correr el código, es necesario modificar los archivos `.yml` de cada directorio para mantener acorde los parámetros del dominio y de la red, ademas de especificar rutas de directorio como de los datos del entrenamiento o el modelo entrenado.

El codigo utiliza las siguientes librerias:

- Pytorch [69]: Corresponde a una interfaz para programar y optimizar redes neuronales. Esta librería permite customizar la red y la función de pérdida dependiendo del problema. Además viene con el algoritmo de diferenciación automática y de back propagation.
- Scipy [70]:Se utiliza esta librería por los algoritmos y expresiones de computación científica que tiene implementado.
- Matplotlib [71]: Se utiliza para la visualización de los resultados obtenidos.
- Numpy [72]: Se utiliza para la manipulación de arreglos y matrices.

```
PoissonSolverCNN/
└── 2D/ ..................................... Clases para resolver problemas en 2D
    ├── dataset ......................... Clases para generar datos de entrenamiento
    │   ├── generated
    │   ├── dataset.yml
    │   └── random_data.py
    └── training ..................................... Clases para entrenar el modelo
        ├── trained_models
        ├── models.py
        ├── operators.py
        ├── train.yml
        ├── train_interface.yml
        └── train.py
```

```
    ├── train_interface.py
    ├── solver ..........................................Clases para resolver casos
    │   ├── solver.yml
    │   ├── solver_interface.yml
    │   ├── solver.py
    │   └── solver_interface.py
├── 3D/ .....................................Clases para resolver problemas en 3D
    ├── dataset ......Clases para generar datos de entrenamiento en 3 dimensiones
    │   ├── generated
    │   ├── dataset.yml
    │   └── random_data.py
    ├── training .................Clases para entrenar el modelo en 3 dimensiones
    │   ├── trained_models
    │   ├── models.py
    │   ├── operators.py
    │   ├── train.yml
    │   ├── train_interface.yml
    │   ├── train.py
    │   └── train_interface.py
    ├── solver .......................Clases para resolver casos en 3 dimensiones
    │   ├── solver.yml
    │   ├── solver_interface.yml
    │   ├── solver.py
    │   └── solver_interface.py
└── Archs ....................................Clases con arquitecturas de red
```

Figura 3.6: Estructura de archivos para el código implementado

Capítulo 4

Resultados y Análisis

En esta sección se presentan los resultados obtenidos en la implementación del modelo propuesto para diferentes arquitecturas y problemas sobre la ecuación de Poisson.

4.1. Pruebas realizadas

A continuación, se entrega un resumen de las pruebas realizadas para la validación del modelo propuesto.

- Variación de arquitectura: Para estas pruebas, se entrenaron varias arquitecturas conservando los hiperparámetros para luego comparar los desempeños. El proposito de esta prueba fue identificar la arquitectura más adecuada para resolver distintos casos de la ecuación de Poisson, tanto en 2D como en 3D. Los ensayos abarcaron problemas de cargas puntuales en el dominio y funciones fuente suaves acompañadas de condiciones de frontera no homogéneas. Se consideraron dos tipos de arquitecturas:
  - Arquitecturas de tipo UNet, las cuales se caracterizan por su estructura de codificación y decodificación.
  - Arquitecturas de tipo MSNet, que se basan en la idea de múltiples escalas para capturar mejor las características del campo.

  A estas se les agregaron variaciones en el número de escalas y el campo receptivo.
- Variación de hiperparámetros: Tomando como punto de partida las arquitecturas con mejor desempeño en la prueba anterior, se exploró el efecto de ajustar sus hiperparámetros. Asimismo, se entrenaron redes que inicialmente arrojaron resultados pobres, con la expectativa de que una calibración adecuada mejorara su rendimiento. Al igual que antes, los experimentos se replicaron en 2D y 3D, cubriendo casos con cargas puntuales y funciones fuente suaves. Los hiperparámetros que se ajustaron incluyen:

  - Tasa de aprendizaje.
  - Cantidad de epochs.
  - Tamaño del kernel.
  - Normalización de los datos.
  - Pesos de las funciones de perdida.
- Variación en los dominios: Estas pruebas evalúan la sensibilidad del modelo frente a cambios en la definición del dominio y en la densidad de la malla. Para ello, se entrenó una arquitectura con los hiperparámetros óptimos sobre distintas combinaciones de dominio–malla. Primero se trabajó en 2D con el fin de identificar el tamaño de malla mínimo que garantice la convergencia. Este resultado sirve de referencia para las simulaciones en 3D, donde se busca emplear la malla más pequeña posible y así reducir el elevado costo computacional.
- Dominios con interfaces: El objetivo es entrenar al modelo en problemas que incluyen discontinuidades materiales. Para ello se ensayaron varias arquitecturas y conjuntos de hiperparámetros, junto con tres esquemas de entrenamiento:
  - Pérdida combinada: *laplacian loss* + *interface loss* + *dirichlet loss*.
  - Pérdida combinada: *inside loss* + *dirichlet loss*.
  - Pérdida combinada: *laplacian loss* + *inside loss* + *dirichlet loss*.

  Estas pruebas permiten evaluar la capacidad del modelo para resolver configuraciones con interfaces y, al mismo tiempo, identificar la combinación de hiperparámetros que asegure un entrenamiento estable y preciso.

Por simplicidad, se trabajará con la nomenclatura de UNetX-ksY-rfZ para las arquitecturas de tipo UNet y MSNetX-ksY-rfZ para las arquitecturas de tipo MSNet, en donde X representa el numero de escalas, Y representa el Tamaño del kernel, el cual es constante para todas las capas en este caso, y Z representa el campo receptivo de la red.

4.2. Carga Puntual Centrada con Condiciones de Borde Homogéneas en 2D

Como primera validación de resultados, se buscó resolver el caso de una carga puntual centrada en un dominio de dos dimensiones con condiciones de borde homogéneas iguales a cero. Este modelo es util debido a su sencillez y que se conoce la solución analítica del caso, presentada en la siguiente ecuación:

$$\phi(r) = \sum_{i=1}^{N} -\frac{q_i}{2\pi\epsilon} \ln\left(|r - r_i|\right) \tag{4.1}$$

En donde N es el número de cargas, q_i es el valor de la carga i, $|r - r_i|$ es la distancia entre la posición de la carga y el punto de evaluación r y r_0 es una constante arbitraria.
En las siguientes secciones, se resolverá el problema planteado para diferentes casos con el objetivo de validar el modelo propuesto, dentro de estos, se realizaran estudios con respecto a las arquitecturas planteadas en secciones anteriores, variando los hiperparámetros y parámetros del entrenamiento, esto con el fin de estudiar el alcance del modelo, su capacidad de generalización, que parámetros son importantes y los ideales para el entrenamiento del modelo para luego aplicar estos resultados en problemas más complejos.

4.3. Estudio del Modelo y del Entrenamiento con Cargas Puntuales

En esta sección se estudia cómo distintos aspectos de la arquitectura del modelo y de los parámetros de entrenamiento influyen en el desempeño general de la red para resolver la ecuación de Poisson.

4.3.1. Estudio de Arquitecturas

4.3.1.1. Profundidad de la Red

Como primera prueba, se evaluó el efecto de la profundidad de la red en el desempeño del modelo, a través del entrenamiento de diferentes arquitecturas bajo condiciones y parámetros uniformes, los cuales se detallan a continuación:

- Cantidad de nodos: 101 x 101 en un dominio de 5 x 5 unidades de longitud.
- Cantidad de epochs: 250.
- Tasa de aprendizaje: 0.0001.
- Normalización: $\alpha = 0.1$.
- Pesos de las funciones de perdida: *laplacian loss* = 8 x 10^5 y *dirichlet loss* = 8 x 10^1.
- Tamaño del kernel: 3 x 3.
- Tamaño del batch: 64.

En cuanto a las arquitecturas, se entrenaron variantes de MSNet y UNet, manteniendo constante el número de escalas en 4, pero variando el campo receptivo. A continuación, se presentan las

arquitecturas entrenadas:

Arquitectura	**Campo Receptivo**
MSNet	50
MSNet	100
MSNet	200
UNet	50
UNet	100
UNet	200
UNet	300
UNet	400

Para definir la calidad de los resultados obtenidos, se utilizó como referencia la solución analítica para el caso de una carga puntual centrada en el dominio, con magnitud de 18 unidades, la cual se presenta a continuación:

$$\phi(r) = -\frac{18}{2\pi} \ln(|r|) \tag{4.2}$$

La Figura 4.1 muestra la solución analítica al caso de una carga puntual centrada en el dominio.

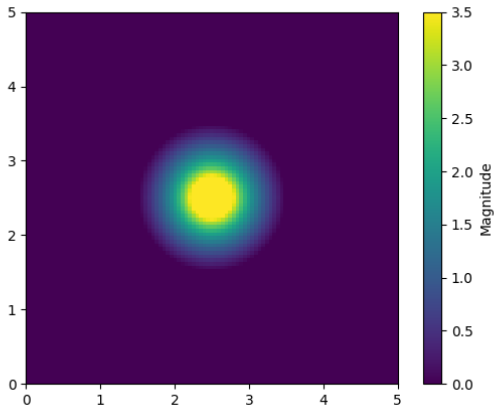


Figura 4.1: Solución analítica al caso de una carga puntual centrada en el dominio

A continuación, se presentan los resultados obtenidos para cada una de las arquitecturas entrenadas:

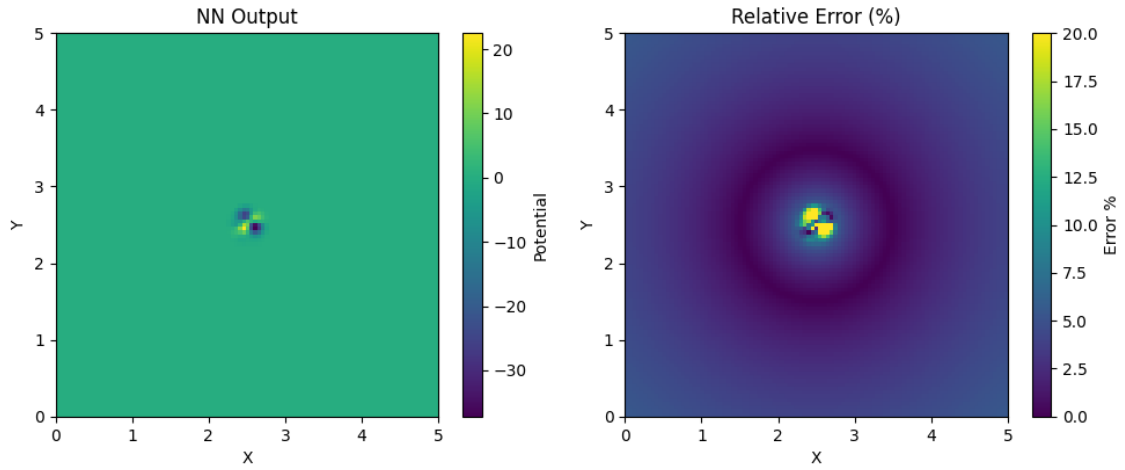


Figura 4.2: Resultados obtenidos para MSNet4-rf50

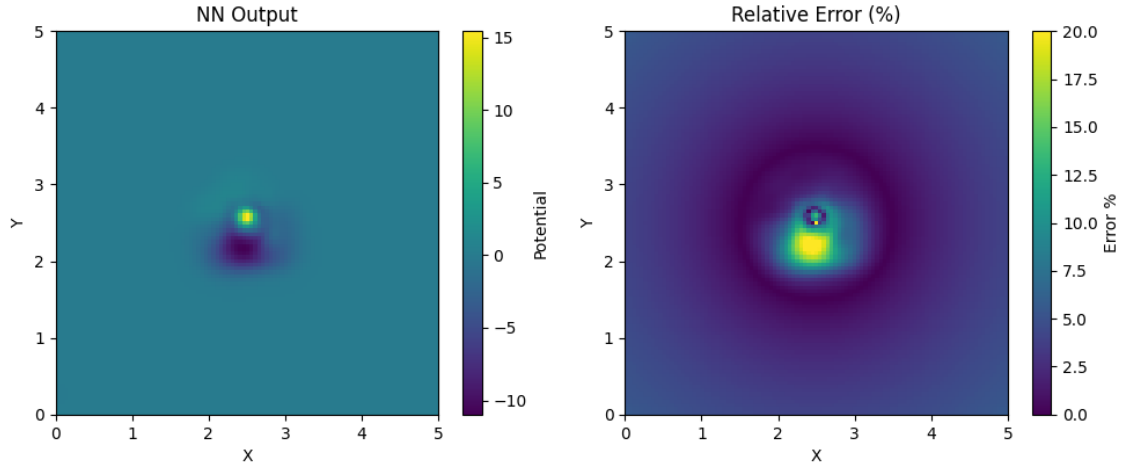


Figura 4.3: Resultados obtenidos para MSNet4-rf100

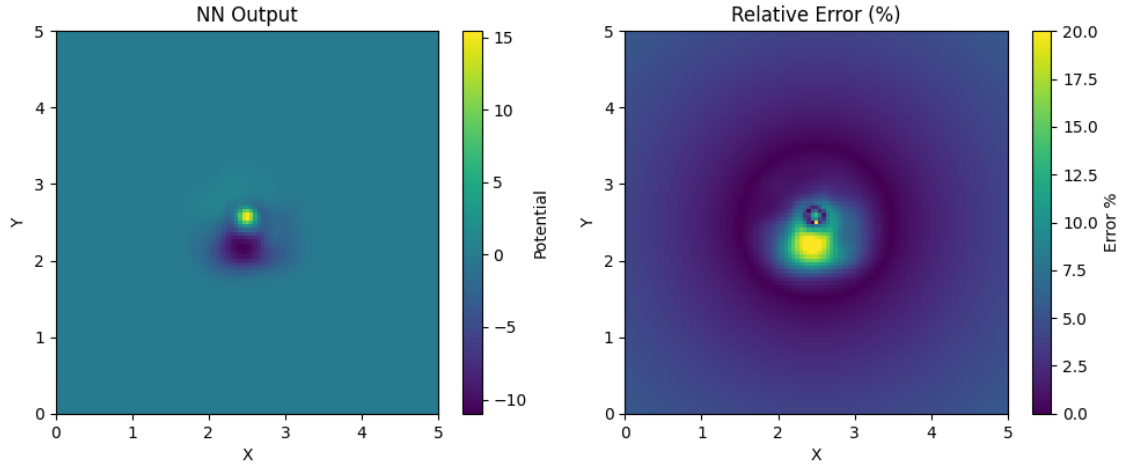


Figura 4.4: Resultados obtenidos para MSNet4-rf200

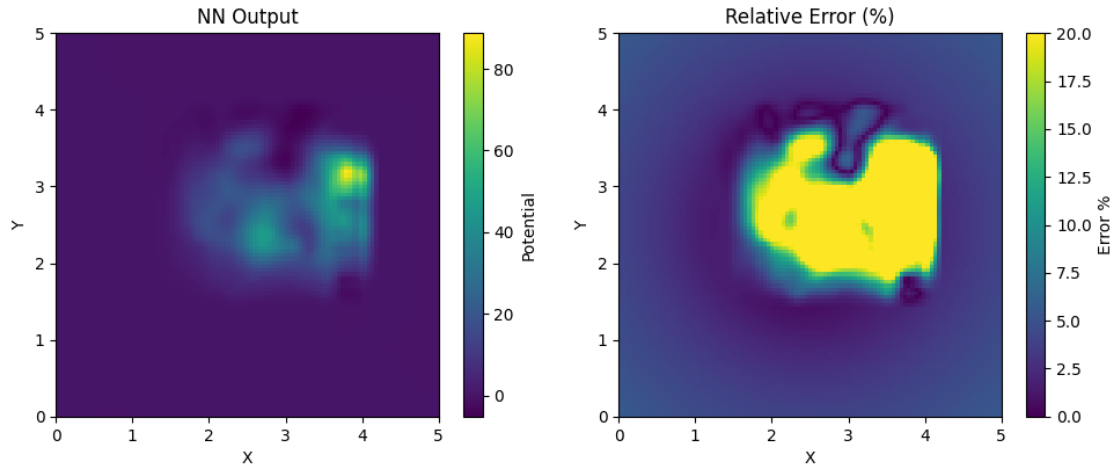


Figura 4.5: Resultados obtenidos para UNet4-ks3-rf50

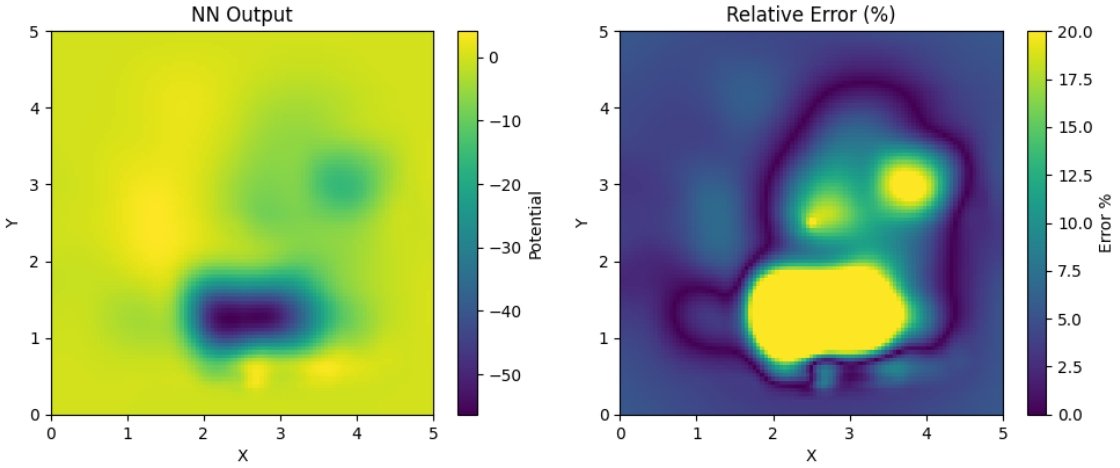


Figura 4.6: Resultados obtenidos para UNet4-ks3-rf100

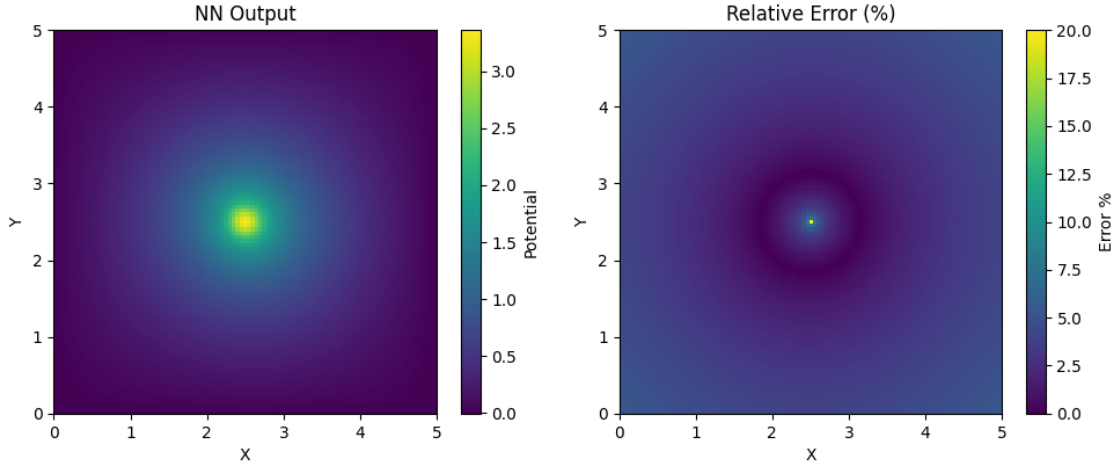


Figura 4.7: Resultados obtenidos para UNet4-ks3-rf200

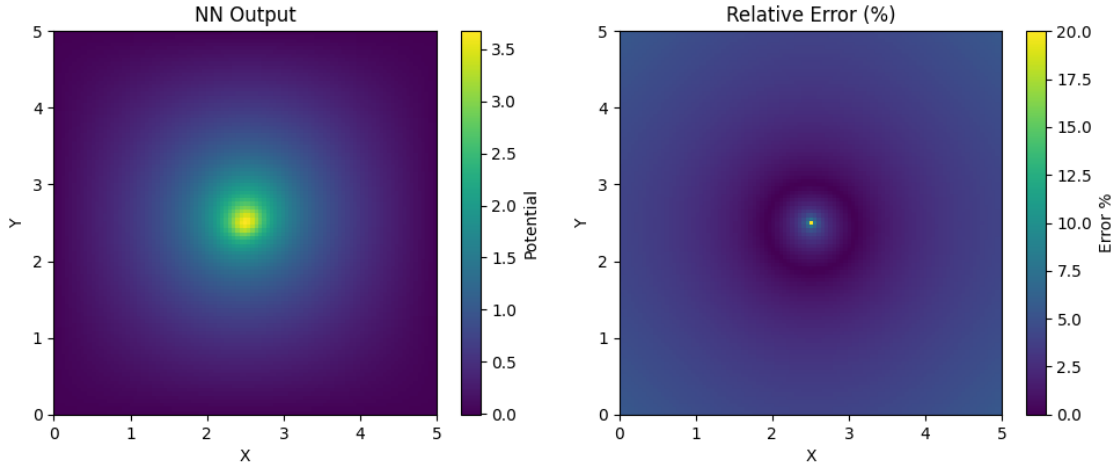


Figura 4.8: Resultados obtenidos para UNet4-ks3-rf300

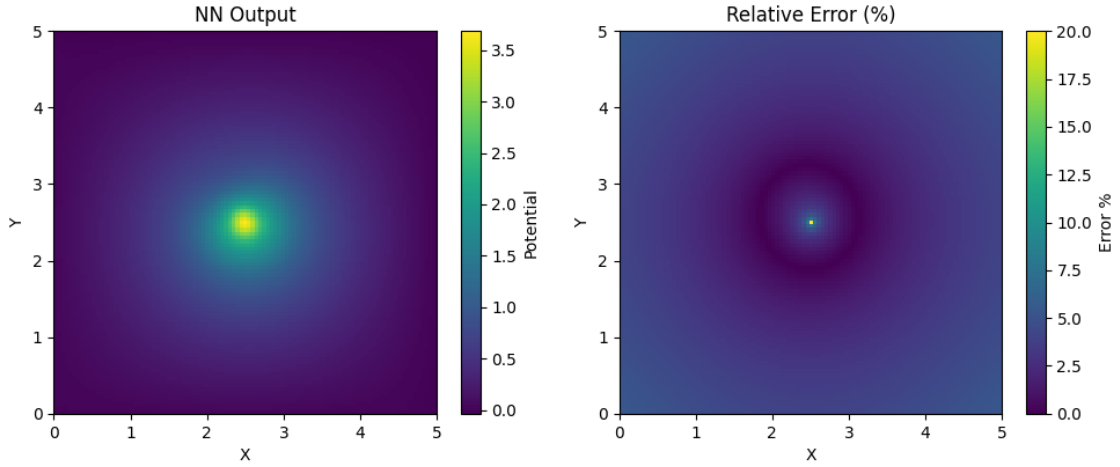


Figura 4.9: Resultados obtenidos para UNet4-ks3-rf400

Cuadro 4.1: Errores promedios, máximos y varianza obtenidos para las diferentes arquitecturas

Caso	**Error Máx. (%)**	**Error Prom. (%)**	**Varianza R^2**
MSNet4_rf50	72.85	3.02	-1.3467
MSNet4_rf200	72.43	3.44	-2.9486
Unet4_rf50	270.79	14.36	-182.2912
Unet4_rf100	169.87	15.12	-163.9440
Unet4_rf200	89.81	3.58	-1.4360
Unet4_rf300	88.87	3.83	-1.6911
Unet4_rf400	88.83	3.52	-1.3648

Los resultados obtenidos muestran que las arquitecturas MSNet (Figuras 4.2 a 4.4)tienen un desempeño deficiente independientemente de su profundidad. Esto podría deberse a que, a diferencia de las redes UNet, MSNet procesa la información en caminos paralelos sin una integración efectiva entre niveles de resolución, lo que dificulta la conservación de las propiedades físicas del problema. A pesar de este resultado, se planifican pruebas adicionales con la arquitectura MSNet4-rf200 —la que mostró el resultado más cercano a la solución analítica—, variando el número de escalas y algunos hiperparámetros para determinar si es posible mejorar su desempeño.

Por otra parte, las aruitecturas UNet alcanzan un buen desempeño a partir de un campo receptivo de 200, es decir, para la arquitectura UNet4-ks3-rf200, la cual logra aproximar el campo potencial de la solución analítica como se muestra en la Figura 4.7. En este resultado, se aprecia que el error se concentra en la zona de la carga puntual, lo cual se puede explicar por la naturaleza del problema, donde la solución analítica presenta una singularidad en el punto de la carga, lo que dificulta la aproximación del modelo en esa región. Sin embargo, el error promedio y máximo de la arquitectura UNet4-ks3-rf200 es 3.58 % y 89.81 % respectivamente, lo que es un buen resultado considerando que el error máximo de la arquitectura es un punto singular en el espacio, mientras que el error promedio es aceptable. Para confirmar esta hipótesis, se recalculó el error máximo omitiendo el nodo central asociado a la carga puntual, obteniéndose un error máximo reducido de 4.52 %, lo que evidencia que la mayor parte del dominio está bien resuelta por la red y que la singularidad localizada es la principal responsable del error extremo.

La Tabla 4.1 muestra que el error promedio y máximo de las arquitecturas UNet con campos receptivos de 300 y 400 son similares, lo que indica que el campo receptivo no es un factor determinante en el desempeño del modelo una vez que se alcanza una profundidad adecuada. Esto es favorable ya que permite entrenar con campos receptivos más pequeños, lo que reduce el costo computacional del entrenamiento.

4.3.1.2. Numero de escalas

Para esta prueba, se busca evaluar cómo influye el número de escalas en el desempeño del modelo, así como determinar si las arquitecturas de tipo MSNet son capaces de representar adecuadamente la solución al aumentar dicho parámetro. Con este objetivo, se entrenaron los siguientes modelos, manteniendo constante los hiperparámetros de la prueba anterior:

Cuadro 4.2: Arquitecturas utilizadas para el estudio del número de escalas

Arquitectura	**Numero de Escalas**	**Campo Receptivo**
MSNet	5	200
UNet	5	100
UNet	5	200
UNet	6	200

A continuación, se presentan los resultados obtenidos para cada una de las arquitecturas entrenadas:

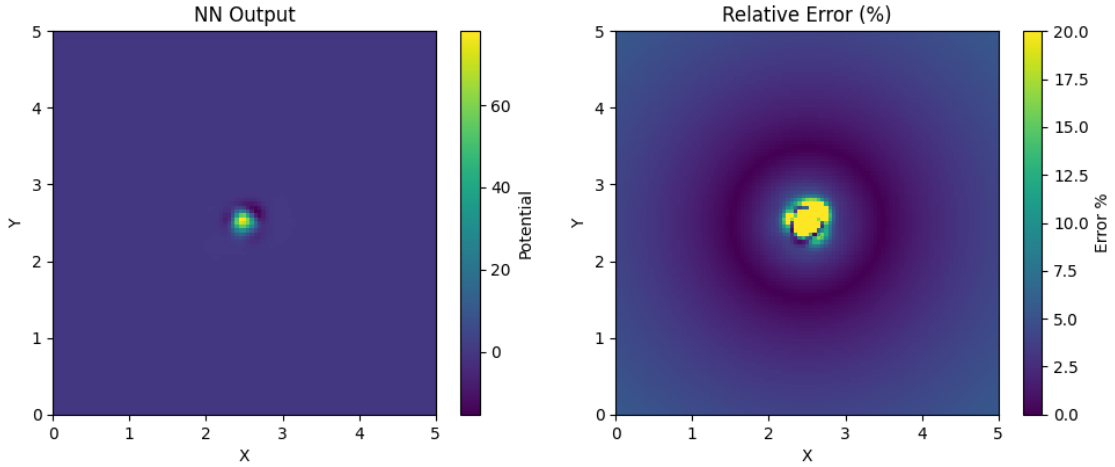


Figura 4.10: Resultados obtenidos para MSNet5-ks3-rf200

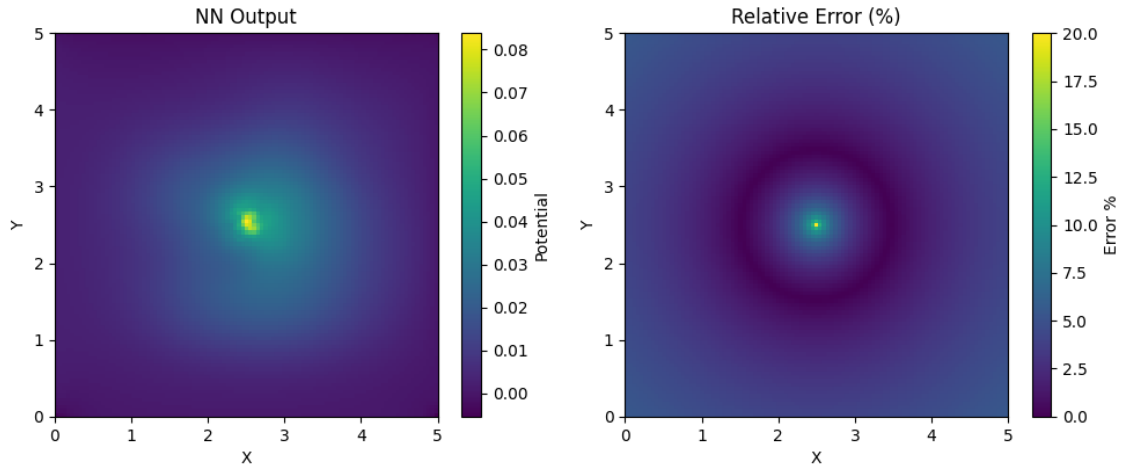


Figura 4.11: Resultados obtenidos para UNet5-ks3-rf100

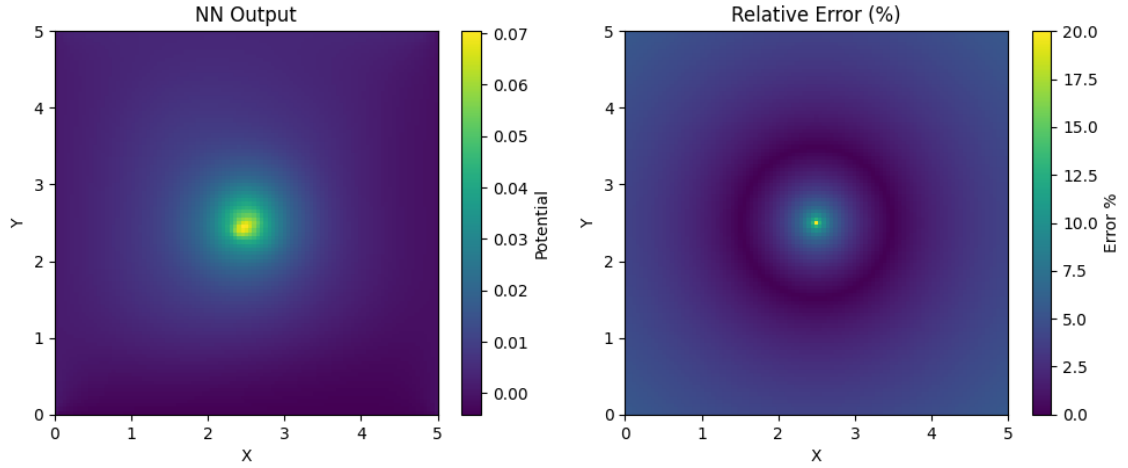


Figura 4.12: Resultados obtenidos para UNet5-ks3-rf200

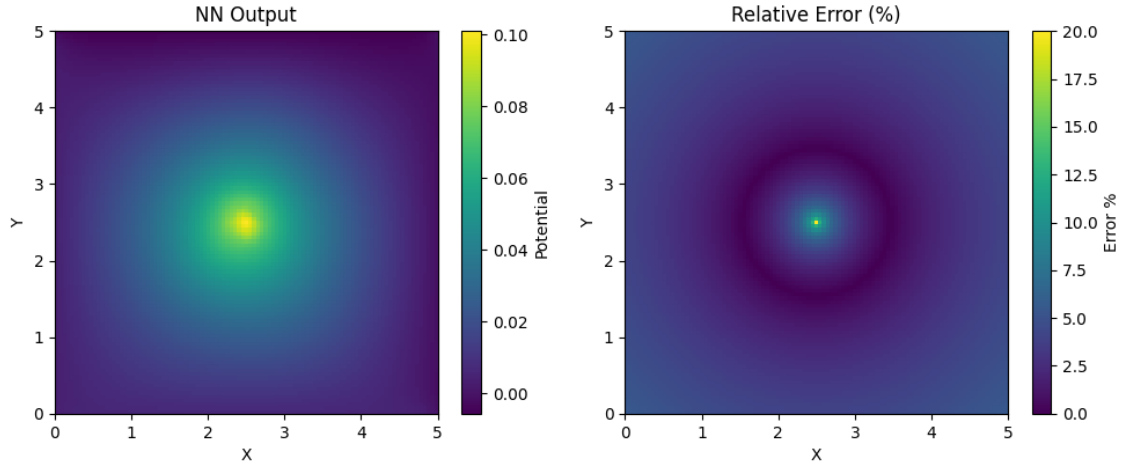


Figura 4.13: Resultados obtenidos para UNet6-ks3-rf200

Cuadro 4.3: Tabla de errores para diferentes escalas de arquitectura

Caso	**Error Máx. (%)**	**Error Prom. (%)**	**R^2**
MSNet5_ks3_rf200	225.12	3.39	-9.4758
Unet5_ks3_rf100	60.83	6.03	-8.2389
Unet5_ks3_rf200	54.37	5.97	-5.6317
Unet6_ks3_rf200	82.61	17.51	-76.5044

En la Figura 4.10 se observa que el modelo MSNet con mayor número de escalas y alta profundidad logra aproximarse a la solución analítica, sin embargo, su desempeño continúa siendo considerablemente inferior al de las arquitecturas UNet, en particular, esta alcanza un error máximo del 225 %, lo que es más del doble que el obtenido con UNet. Además, la varianza R^2 del error es significativamente mayor (9.47 % en MSNet frente a 1.34 % en UNet), lo que indica una mayor dispersión del error y la presencia de más puntos con desviaciones elevadas. Sumado a lo anterior, el mayor tamaño de la arquitectura MSNet implica tiempos de entrenamiento notablemente superiores, se descarta el uso de esta red para futuras pruebas relacionadas con cargas puntuales debido a su pobre desempeño y alto costo computacional.

Con respecto a las arquitecturas UNet, se observa en la Tabla 4.3 que el aumento en el número de escalas reduce el error máximo de campo predichos, aunque a su vez, también se incrementa el error promedio y la varianza R^2. Esto sugiere que, si bien el modelo con mayor número de escalas logra representar de manera aproximada el campo potencial —tal como se aprecia en la Figura 4.12 a 4.13—, el aumento de complejidad no se traduce en una mejora del desempeño del modelo, razón por la cual por la cual se recomienda utilizar una arquitectura UNet4-ks3-rf200 para este tipo de problemas de forma general.

En síntesis, se concluye que para este tipo de problemas, las arquitecturas UNet son mas efectivas que las MSNet, las cuales logras predecir el campo potencial de manera adecuada cuando esta tiene un campo receptivo de 200 y 4 escalas. Además, se observa que el número de escalas no es un factor determinante en el desempeño del modelo, al igual que el aumento del campo receptivo, por lo que se recomienda utilizar una arquitectura UNet4-ks3-rf200 para este tipo de problemas, ya que es la que logra el mejor desempeño con un costo computacional razonable.

4.3.1.3. Variación en el tamaño del kernel

Para esta prueba, se busca analizar cómo el tamaño del kernel influye en el desempeño del modelo. Con este objetivo, se entrenaron las arquitecturas UNet4-ks3-rf200 y UNet4-ks3-rf300, variando únicamente el tamaño del kernel entre 5 x 5 y 7 x 7 unidades. Los hiperparámetros

empleados en el entrenamiento son los utilizados en la Sección 4.3.1.1 a excepción del tamaño del kernel. A continuación, se presentan los resultados obtenidos:

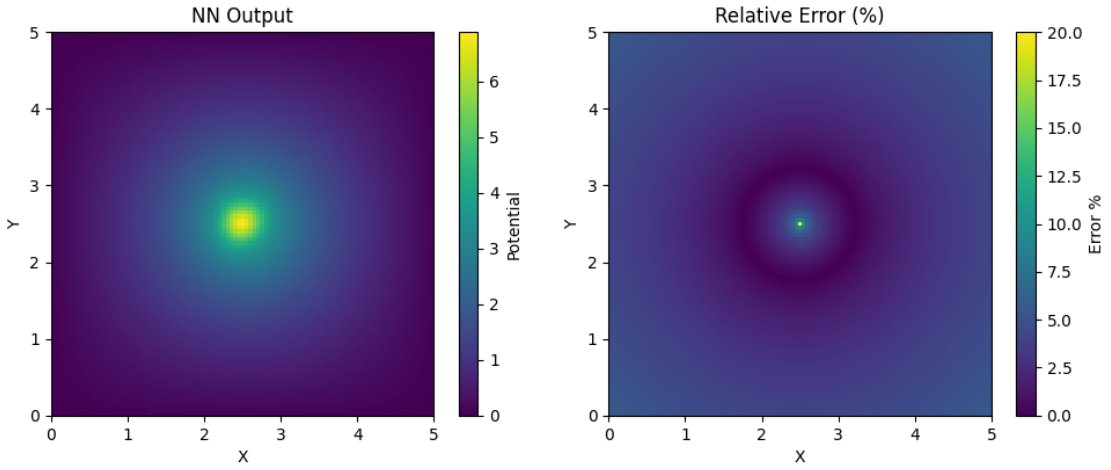


Figura 4.14: Resultados obtenidos para UNet4-ks5-rf200

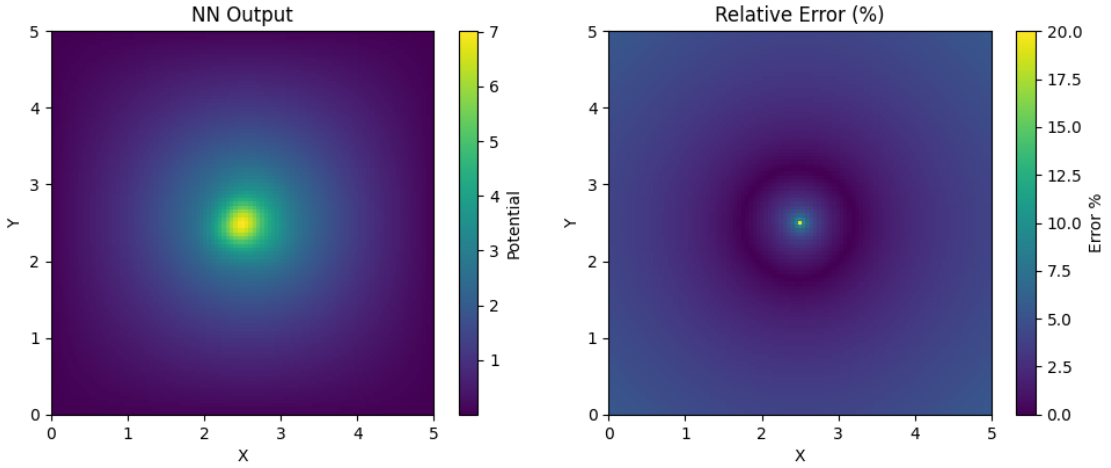


Figura 4.15: Resultados obtenidos para UNet4-ks7-rf200

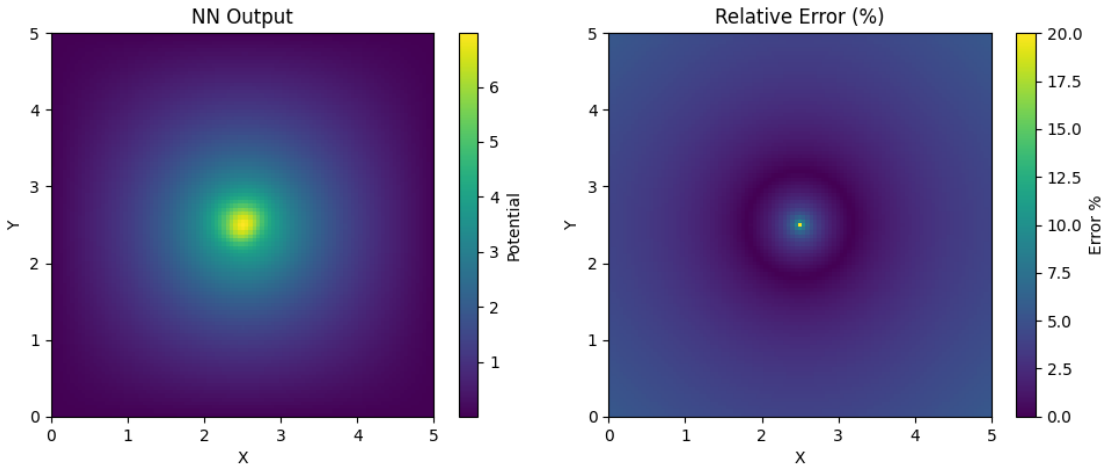


Figura 4.16: Resultados obtenidos para UNet4-ks5-rf300

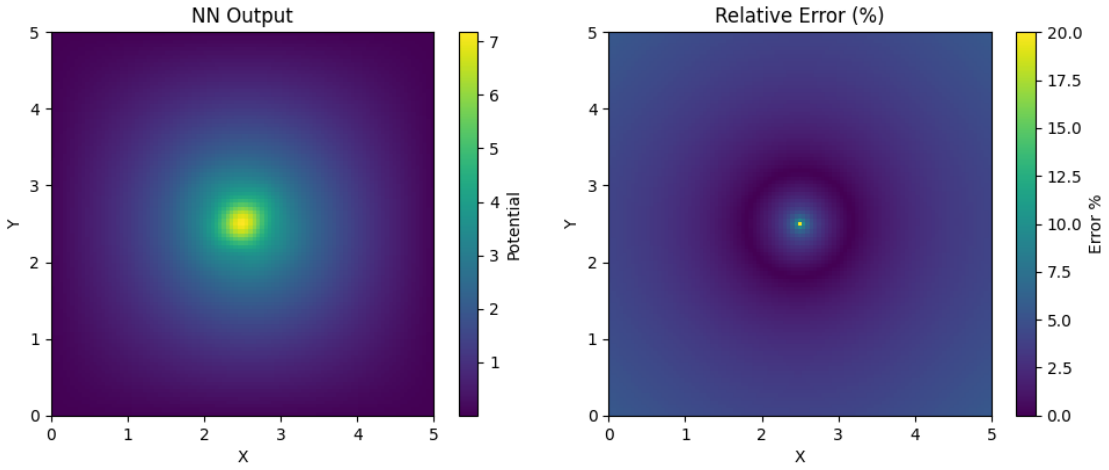


Figura 4.17: Resultados obtenidos para UNet4-ks7-rf300

De los resultados obtenidos, se observa que el tamaño del kernel no tiene un impacto significativo en la predicción del campo potencial, esto se evidencia en las Figuras 4.14-4.17, donde se aprecia que el modelo logra representar adecuadamente el campo, manteniendo un comportamiento similar al de la arquitectura UNet4-ks3-rf200. Por otra parte, la Tabla 4.4 muestra que los errores promedio se mantienen en rangos comparables a los del caso UNet con campo receptivo de 200, aunque se registra un leve aumento en el error máximo, asimismo, se observa una ligera disminución en la varianza R^2,

En consecuencia, si bien el tamaño del kernel introduce una variación marginal en el rendimiento del modelo, esta no es lo suficientemente relevante como para justificar un cambio. Además, el costo computacional no presenta diferencias significativas. Por lo tanto, se recomienda utilizar un tamaño de kernel de 3 x 3 unidades.

Cuadro 4.4: Errores promedios, máximos y varianza obtenidos para las diferentes arquitecturas

Arquitectura	**Error Máx. (%)**	**Error Prom. (%)**	**Varianza R^2**
Unet4-ks3-rf200	89.81	3.58	-1.4360
Unet4-ks5-rf200	96.24	3.12	-1.0690
Unet4-ks7-rf200	96.17	3.14	-1.0859
Unet4-ks3-rf300	88.87	3.83	-1.6911
Unet4-ks5-rf300	96.19	3.16	-1.1021
Unet4-ks7-rf300	96.09	3.17	-1.1079

4.3.2. Estudio de parámetros de entrenamiento

4.3.2.1. Cantidad de epochs

Este estudio tiene como objetivo evaluar la capacidad resolutiva del modelo a medida que se incrementa la cantidad de epochs de entrenamiento, así como identificar el punto en que el aprendizaje comienza a estancarse. Este análisis es especialmente relevante para futuras implementaciones, ya que permite determinar el número óptimo de iteraciones, evitando tiempos de entrenamiento innecesarios y posibles efectos negativos asociados al sobreentrenamiento.
Para ello, se entrenó la arquitectura UNet4-ks3-rf200 utilizando los mismos parámetros de entrenamiento que en la Sección 4.3.1.1 con excepción del número de epochs, y se registraron los resultados cada 10 epochs, los cuales serán analizados en conjunto con las curvas de las funciones de pérdida. A continuación, se presentan los resultados obtenidos:

Cuadro 4.5: Evolución del error y R^2 para Unet4-ks3-rf200 con diferentes epochs de entrenamiento

Caso	**Cantidad de epochs**	**Error Máx. (%)**	**Error Prom. (%)**	**R^2**
Caso 1	10	90.43	3.07	-1.0047
Caso 2	20	91.97	2.91	-0.8662
Caso 3	30	92.87	2.86	-0.8313
Caso 4	40	93.66	2.83	-0.8121
Caso 5	50	93.44	2.84	-0.8163
Caso 6	60	93.10	2.80	-0.7849
Caso 7	70	92.60	2.85	-0.8297
Caso 8	80	92.30	2.89	-0.8637
Caso 9	90	92.04	2.91	-0.8804
Caso 10	100	92.53	2.45	-0.8726

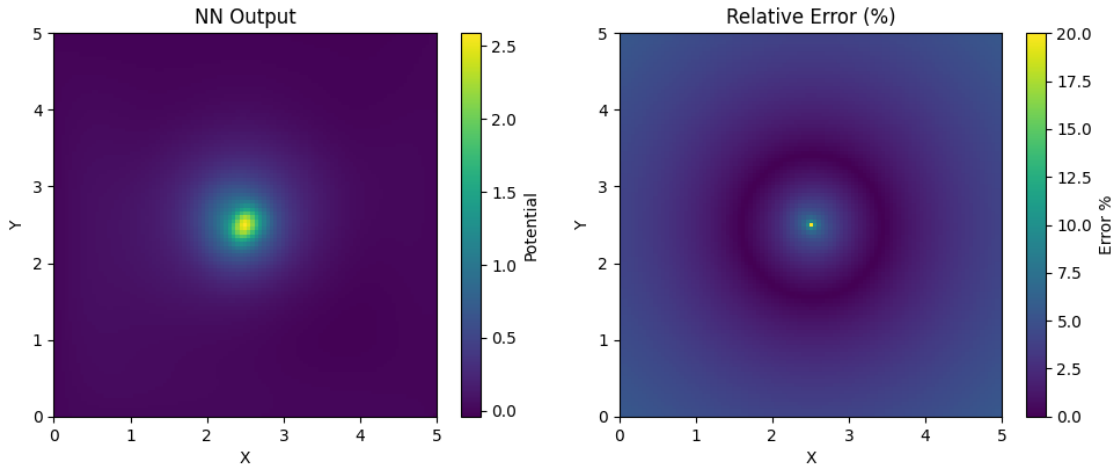


Figura 4.18: Resultados obtenidos para UNet4-ks3-rf200 con 10 epochs de entrenamiento

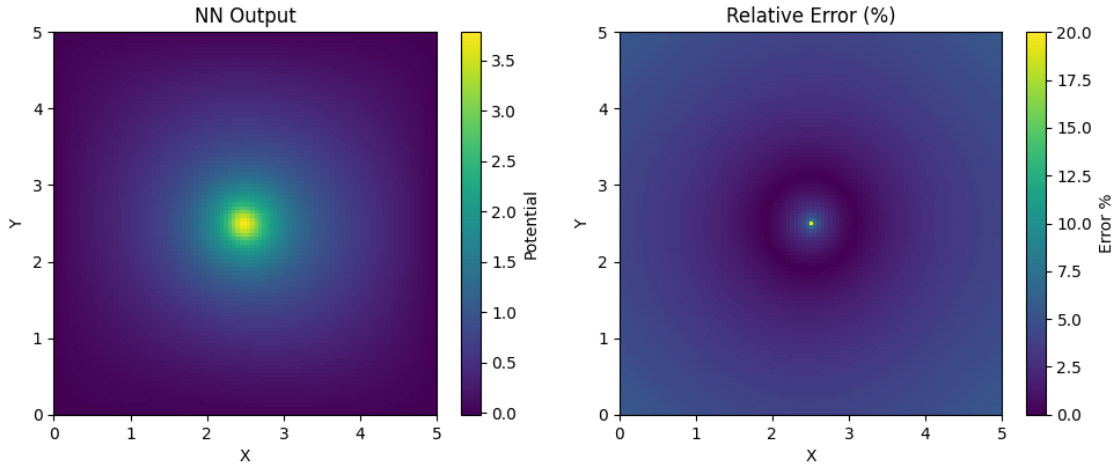


Figura 4.19: Resultados obtenidos para UNet4-ks3-rf200 con 100 epochs de entrenamiento

A partir de los resultados obtenidos, se observa en la Tabla 4.5 que los errores promedio y máximo, junto con la varianza R^2, presentan una tendencia decreciente a medida que aumenta el número de epochs de entrenamiento, hasta alcanzar un mínimo en los 70 epochs, con valores de 2.80 %, 93.10 % y 0.7849, respectivamente. Sin embargo, al continuar el entrenamiento más allá de este punto, se evidencia un aumento en los tres indicadores, lo que sugiere posibles signos de sobreentrenamiento. Otra explicación posible es que el modelo ha alcanzado su punto de convergencia y ya no es capaz de mejorar su desempeño, lo que lo lleva a explorar regiones del espacio de soluciones con mayor error.

De la Figura 4.20, se aprecia que el modelo tiene una tendencia a estabilizarse en torno a las 30.000 iteraciones (lo que equivale a aproximadamente 25 epochs), lo que coincide con lo observado previamente en la Tabla 4.5. Por esta razón, se recomienda entrenar el modelo con un mínimo de 30 epochs, ya que a partir de este punto el modelo se estabiliza, y un máximo de

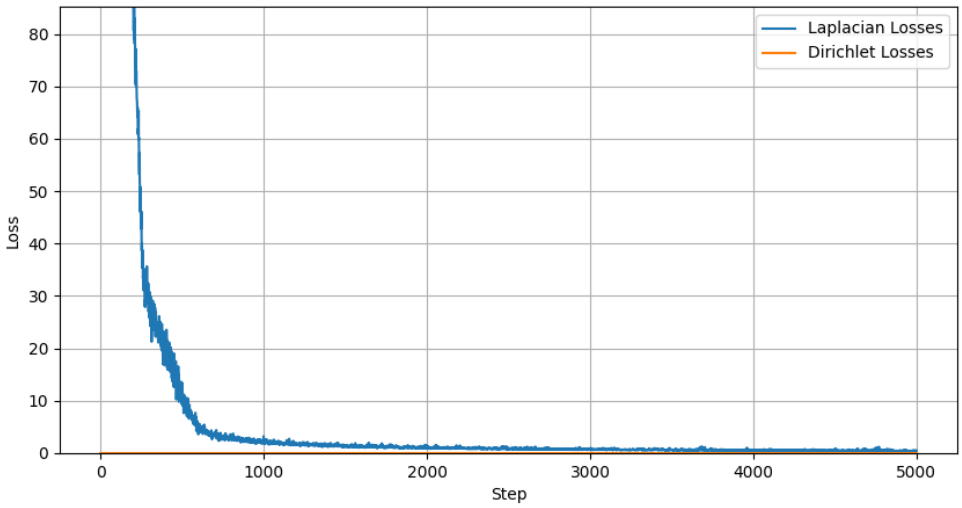


Figura 4.20: Funciones de perdida obtenidas para UNet4-ks3-rf200 con 100 epochs de entrenamiento

70 epochs, ya que a partir de este punto los resultados comienzan a empeorar.
De forma general, se puede afirmar que el modelo propuesto no requiere de un gran número de epochs para converger, lo que es una ventaja significativa en términos de tiempo de entrenamiento y recursos computacionales. Ademas, se observa que gráfica las funciones de perdida permite identificar el punto de convergencia del modelo y esta puede ser utilizada como recurso para determinar el número óptimo de epochs de entrenamiento.

4.3.2.2. Tasa de aprendizaje

Para el siguiente estudio, se busca determinar como la tasa de aprendizaje afecta en el desempeño del modelo, especialmente en la convergencia del mismo, ademas de determinar si se puede mejorar la convergencia para poder obtener resultados empleando menores costos computacionales. Para esto, se entrenó la arquitectura UNet4-ks3-rf200 con diferentes tasas de aprendizaje, variando entre 1 e^{-5} y 0.1, manteniendo constante el resto de los hiperparámetros utilizados en la Sección 4.3.1.1. A continuación, se presentan los resultados obtenidos:

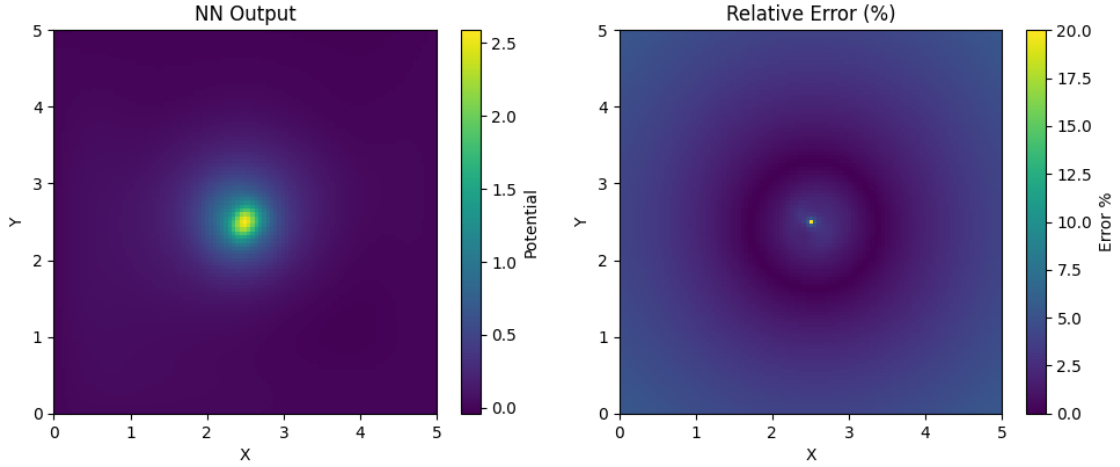


Figura 4.21: UNet4-ks3-rf200 con tasa de aprendizaje de 1 e^{-5}

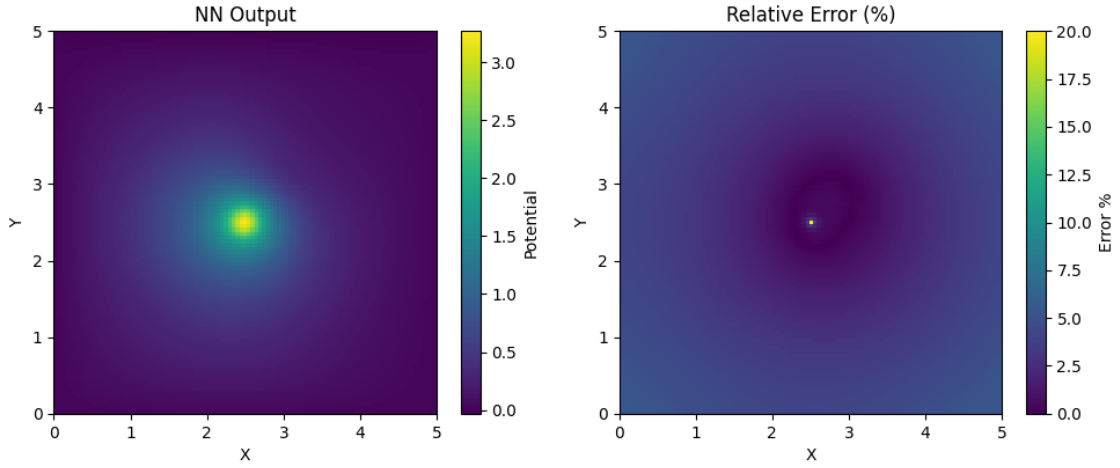


Figura 4.22: UNet4-ks3-rf200 con tasa de aprendizaje de 1 e^{-4}

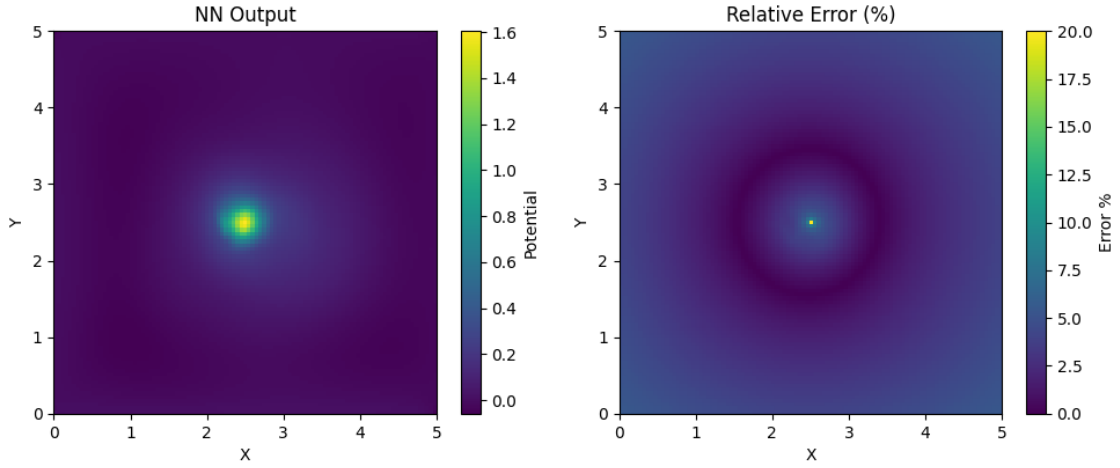


Figura 4.23: UNet4-ks3-rf200 con tasa de aprendizaje de 1 e^{-3}

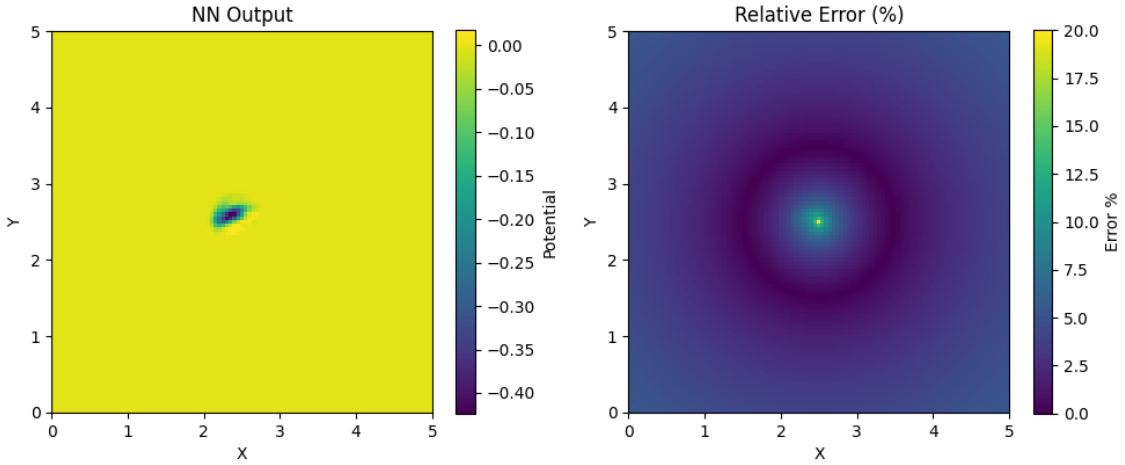


Figura 4.24: UNet4-ks3-rf200 con tasa de aprendizaje de 0.01

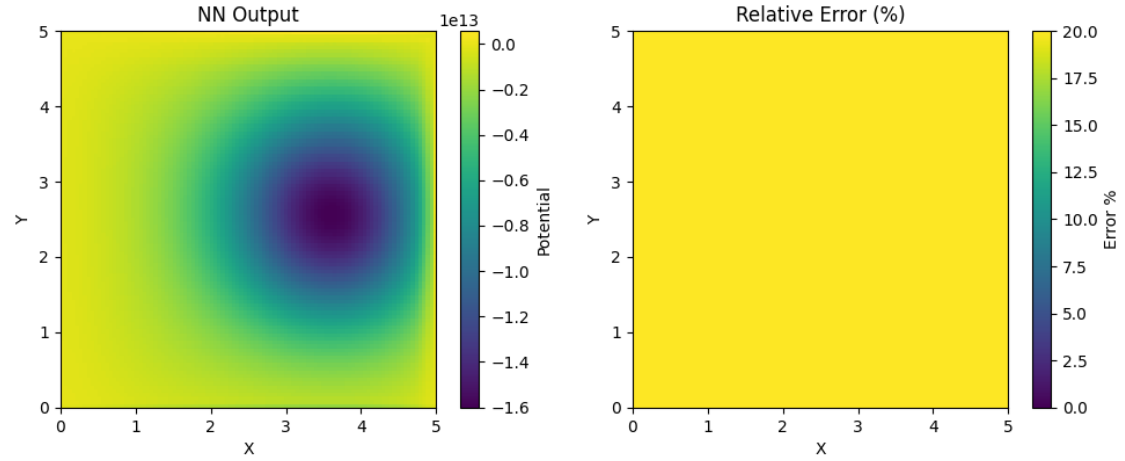


Figura 4.25: UNet4-ks3-rf200 con tasa de aprendizaje de 0.1

Como se observa en las Figuras 4.21 a 4.25, el modelo es sensible a la tasa de aprendizaje, mostrando que esta tiene un impacto significativo en la convergencia y estabilidad del mismo. En particular, se aprecia que tasas de aprendizaje de bajo valor (1 x 10^{-5}, 1 x 10^{-4} y 1 x 10^{-3}) permiten que el modelo converja correctamente y predicen el campo de manera correcta, mientras que a partir de una tasa de 0.01, el entrenamiento comienza a divergir, generando inestabilidad en el dominio, generando un campo potencial que no es capaz de representar adecuadamente la solución analítica, como se observa en las Figuras 4.24 y 4.25.

Además, la Figura 4.26 muestra que no existen diferencias sustanciales en la convergencia entre las tasas de1 x 10^{-5} y 1 x 10^{-3}, lo que sugiere que es preferible optar por una tasa de aprendizaje baja para evitar inestabilidad durante el entrenamiento.

Por estos motivos, se recomienda utilizar una tasa de aprendizaje de 1 x 10^{-5} para el entrenamiento del modelo, ya que permite una convergencia estable, con un margen de error para

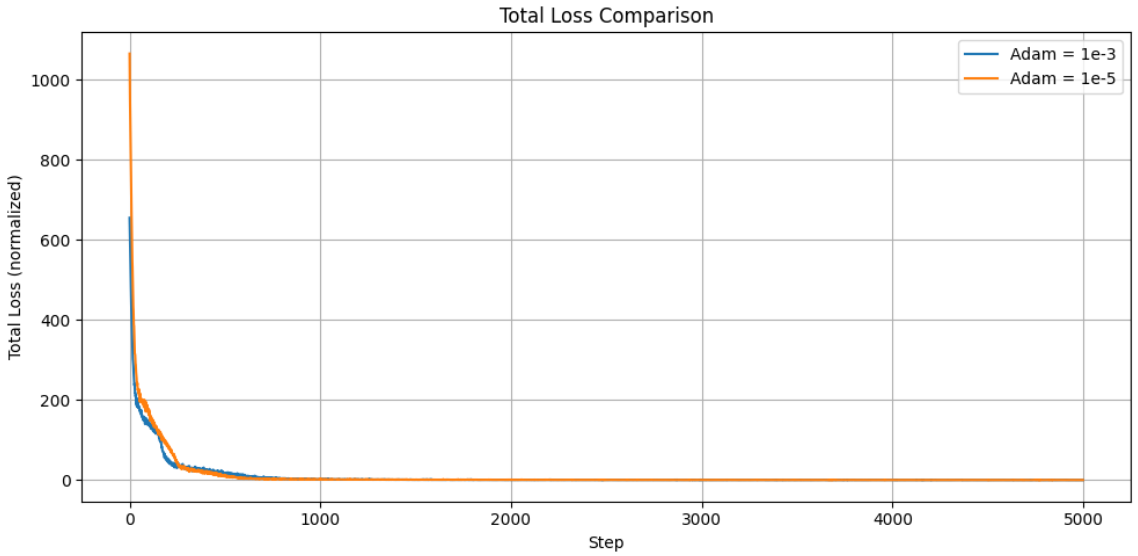


Figura 4.26: Funciones de perdida obtenidas para UNet4-ks3-rf200 con diferentes tasas de aprendizaje

evitar problemas de inestabilidad, ademas de ser un valor que permite un entrenamiento rápido y eficiente.

4.3.2.3. Normalización

En este estudio se pretende evaluar, por un lado, la necesidad de normalizar los datos y, por otro, el efecto del parámetro α en el desempeño del modelo. Para ello se entrenó inicialmente la arquitectura UNet4-ks3-rf200 sin aplicar normalización. Posteriormente, se repitió el entrenamiento con datos normalizados, variando α en tres niveles: 0.01, 0.5 y 1.0, siendo este último el valor máximo teórico admitido por el esquema de normalización. Los conjuntos de datos utilizados para todos los experimentos fueron:

- Cantidad de nodos: 101 x 101 en un dominio de 5 x 5 unidades de longitud.
- Cantidad de epochs: 40.
- Tasa de aprendizaje: 1 x 10^{-5}.
- Pesos de las funciones de perdida: *laplacian loss* = 8 x 10^5 y *dirichlet loss* = 8 x 10^1.
- Tamaño del kernel: 3 x 3.
- Tamaño del batch: 64.

A continuación, se presentan los resultados obtenidos para cada una de las configuraciones entrenadas:

Los resultados, mostrados en la Figura 4.27, confirman que la normalización es indispensable para lograr la convergencia del modelo: sin ella, la red no converge y el campo potencial re-

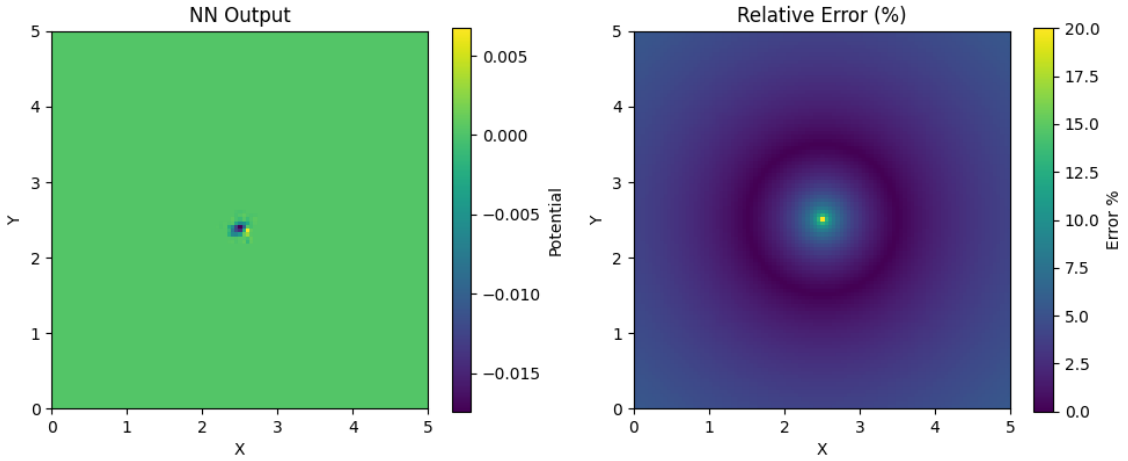


Figura 4.27: Resultados obtenidos para UNet4-ks3-rf200 sin normalización

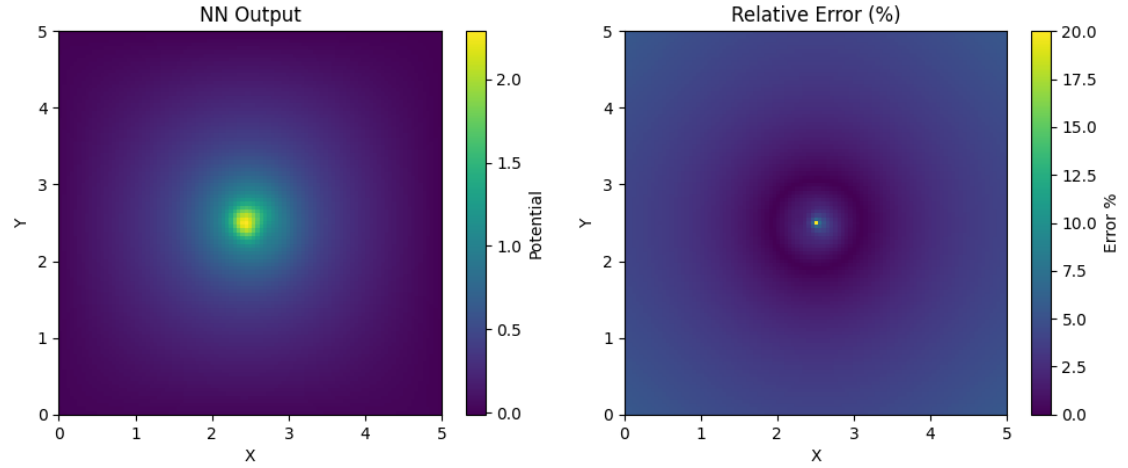


Figura 4.28: Resultados obtenidos para UNet4-ks3-rf200 con $\alpha = 0.01$

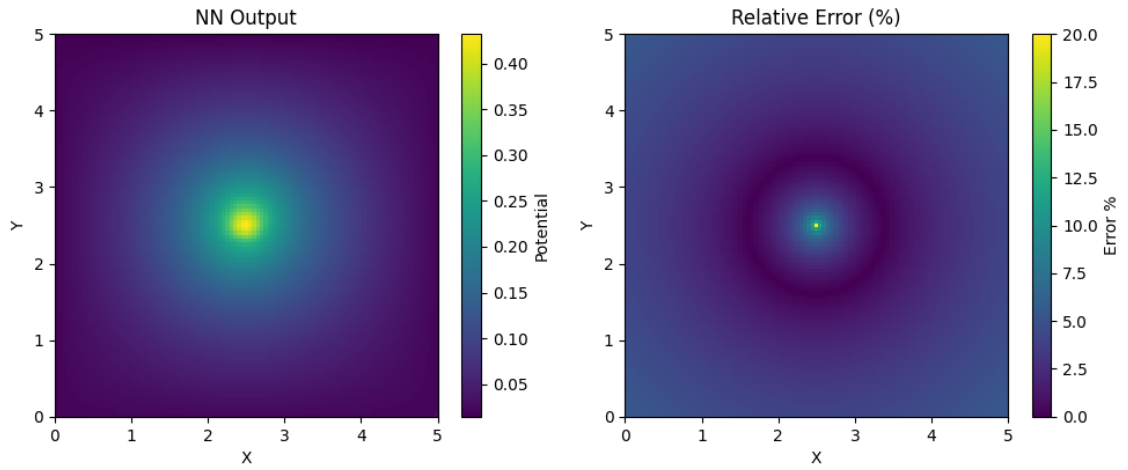


Figura 4.29: Resultados obtenidos para UNet4-ks3-rf200 con $\alpha = 0.5$

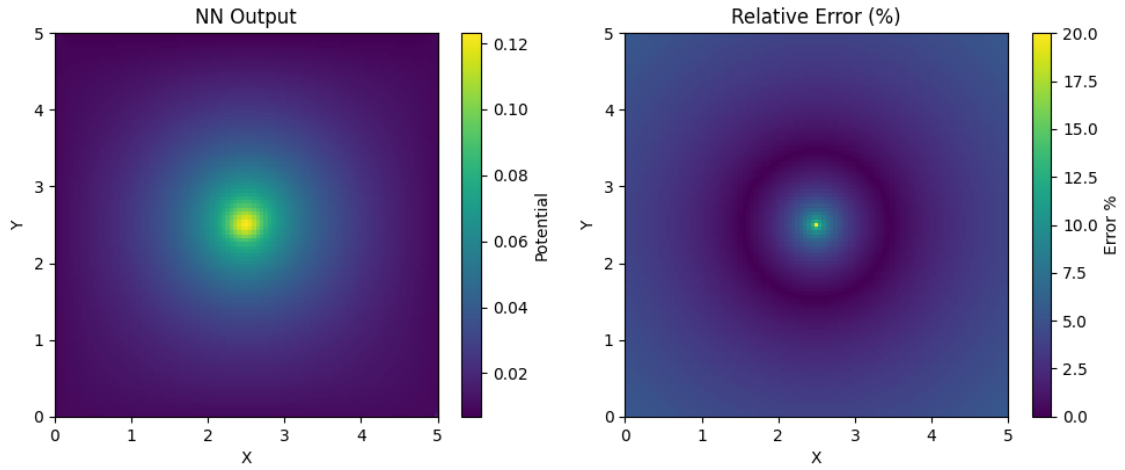


Figura 4.30: Resultados obtenidos para UNet4-ks3-rf200 con $\alpha = 1.0$

Cuadro 4.6: Comparación del error y R^2 Para diferentes valores de α

Caso	**Error Máx. (%)**	**Error Prom. (%)**	$\mathbf{R^2}$
Unet 200 sin normalizar	99.25	2.98	-0.9777
unet 200 $\alpha = 0.01$	103.19	3.02	-1.1008
unet 200 $\alpha = 0.5$	101.02	2.82	-0.8446
unet 200 $\alpha = 1.0$	99.51	3.16	-1.1805

sultante es inestable. Una vez normalizados los datos, el modelo reproduce correctamente el campo potencial para todos los valores de α evaluados, con errores promedio y máximos prácticamente idénticos. Esto permite seleccionar un valor de α que no comprometa el rendimiento independientemente del problema. Por consiguiente, en los trabajos posteriores se mantendrá $\alpha = 0.1$, tal como se adoptó en las secciones anteriores.

4.3.2.4. Pesos en la función de pérdida

Este estudio evalúa la influencia de los pesos de cada término en la función de pérdida y busca una relación óptima para el aprendizaje del modelo. Para ello se entrenó la arquitectura UNet4-ks3-rf200 variando solo el peso del término Laplaciano, $w_{\text{laplacian}}$, mientras se fijó el peso de las condiciones de borde en $w_{\text{bc}} = 1$. El valor de $w_{\text{laplacian}}$ se incrementó de 1 a 10^5 en saltos de una orden de magnitud, manteniendo intactos todos los demás hiperparámetros descritos en la Sección 4.3.2.3.

Los campos de potencial generados para cada configuración se incluyen en el Anexo C; los resultados correspondientes se presentan a continuación.

A partir de los resultados presentados en la Tabla 4.7, se infiere que el peso del término Laplaciano no tiene un impacto significativo en los errores globales. Sin embargo, al analizar los

Cuadro 4.7: Comparación del error y R^2 ajustando el peso `laplacian weight (lw)`

Caso	**Peso Laplaciano**	**Error Máx. (%)**	**Error Prom. (%)**	**R^2**
Caso 1	1 x 10^0	98.81	2.97	-0.9685
Caso 2	1 x 10^1	98.77	3.01	-0.9969
Caso 3	1 x 10^2	99.30	2.95	-0.9557
Caso 4	1 x 10^3	97.78	3.24	-1.1905
Caso 5	1 x 10^4	96.09	3.38	-1.2790
Caso 6	1 x 10^5	98.72	2.98	-0.9562

campos de potencial obtenidos, se observa que el modelo solo converge adecuadamente en los casos con pesos mayores a 1 x 10^4 (véanse las figuras C.5 y C.6), en contraste, para valores menores, se presentan problemas de estabilidad en las soluciones. Esto se explica por la influencia dominante del término asociado a las condiciones de frontera en la función de pérdida cuando el peso del Laplaciano es bajo. Al aumentar dicho peso, el término del Laplaciano comienza a prevalecer, guiando correctamente la optimización y permitiendo una mejor convergencia.
Cabe destacar que no se evidencian diferencias relevantes entre los resultados obtenidos con pesos superiores a 1 x 10^4, lo que sugiere que el modelo es robusto ante variaciones dentro de ese rango, siempre que el peso del Laplaciano sea suficientemente elevado.
Es importante mencionar que, en este caso, se utilizaron condiciones de frontera homogéneas e iguales a cero, lo cual justifica que el término Laplaciano deba tener mayor influencia. Sin embargo, en escenarios futuros con condiciones de borde no homogéneas, se anticipa que será necesario reducir el peso del Laplaciano para que el modelo pueda interpretar correctamente la información de frontera y su efecto en el campo de potencial.

4.3.3. Estudio del dominio de entrenamiento

En las siguientes pruebas se analiza cómo influyen el tamaño del dominio y la cantidad de nodos en el desempeño del modelo. Para ello, se entrenó la arquitectura UNet4-ks3-rf200 considerando distintas combinaciones de tamaño de dominio y resolución espacial. El problema considerado es el mismo que en las secciones anteriores: una carga puntual centrada en el dominio, con condiciones de borde homogéneas igual a cero.

4.3.3.1. Variación en tamaño del dominio

En el presente estudio, se evalúa si la definición del tamaño del dominio tiene un impacto significativo en el desempeño del modelo. Para ello, se entrenó la arquitectura UNet4-ks3-rf200 utilizando dominios de 0.1, 1 y 100 unidades de longitud, manteniendo constantes todos los

demás parámetros de entrenamiento descritos en la Sección 4.3.1.1. A continuación, se presentan los resultados obtenidos:

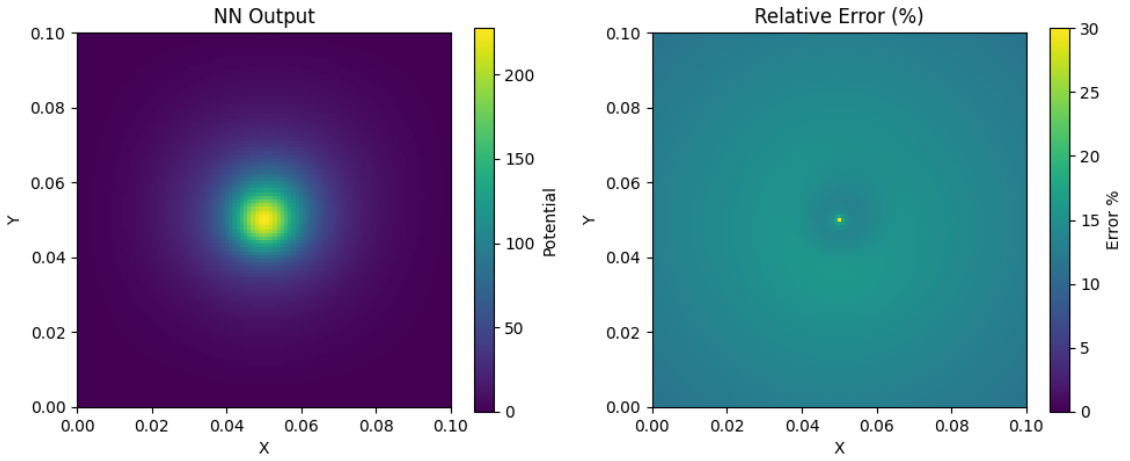


Figura 4.31: Resultados obtenidos para UNet4-ks3-rf200 con dominio de 0.1 unidades de longitud

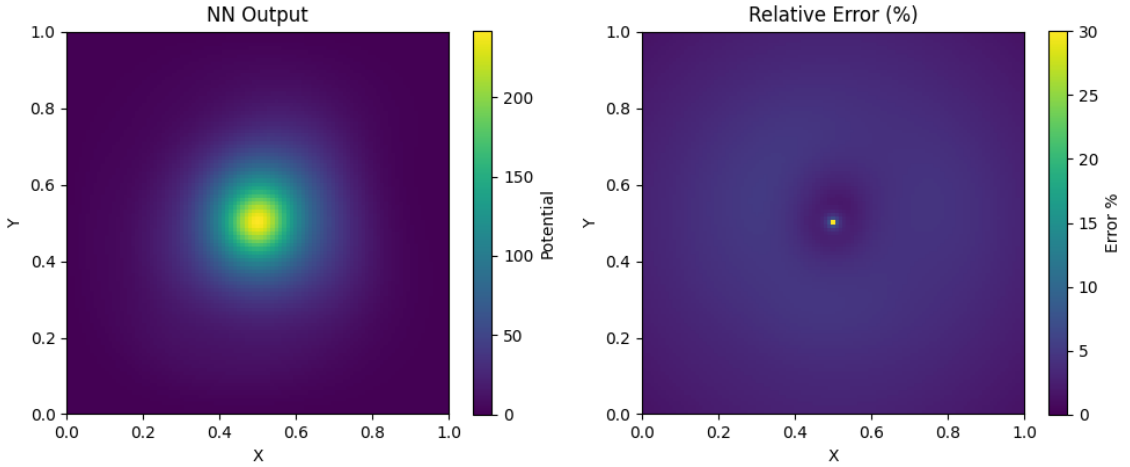


Figura 4.32: Resultados obtenidos para UNet4-ks3-rf200 con dominio de 1 unidad de longitud

Cuadro 4.8: Comparación del error y R^2 con diferente tamaño de dominio

Caso	**Tamaño de Dominio (unidades)**	**Error Máx. (%)**	**Error Prom. (%)**	**R^2**
Caso 1	0.1	87.58	13.77	-33.1165
Caso 2	1.0	85.33	2.16	-0.8052
Caso 3	100.0	99.58	15.48	-38.5621

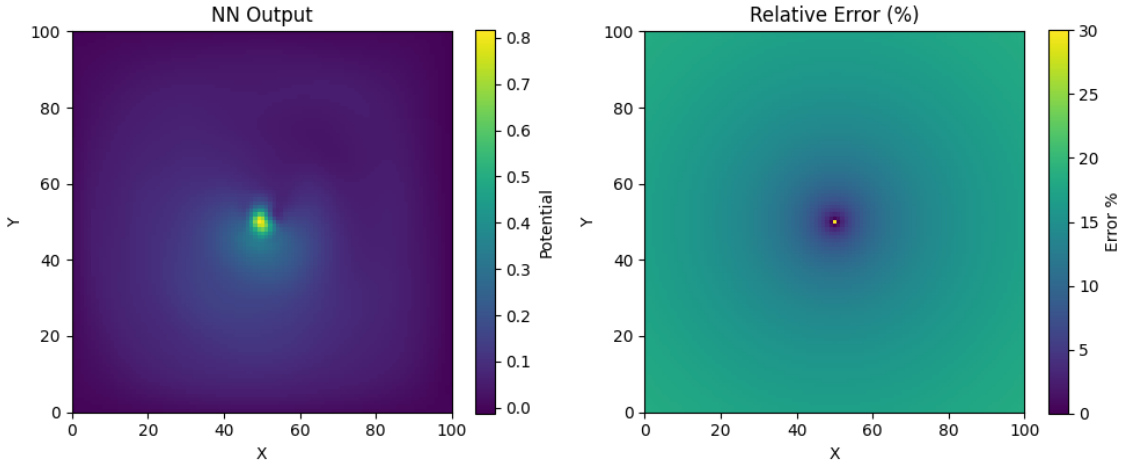


Figura 4.33: Resultados obtenidos para UNet4-ks3-rf200 con dominio de 100 unidades de longitud

Los resultados obtenidos en la Tabla 4.8 muestran que el tamaño del dominio tiene un impacto significativo en el desempeño del modelo. En particular, para un dominio de 0.1 unidades de longitud (Figura 4.31), el modelo logra converger, pero presenta un error promedio de 13.77 % y un error máximo de 87.58 %, lo que indica una gran deficiencia en la predicción del campo potencial. Caso similar ocurre para un dominio de 100 unidades (Figura 4.33), donde el modelo no logra predecir la solución del potencial, generando un campo inestable ademas de un error promedio de 15.48 % y un error máximo de 99.58 %. En contraste, cuando se utiliza un dominio de 1 unidad de longitud (Figura 4.32), el modelo logra converger correctamente, con un error promedio de 2.16 % y un error máximo de 85.33 %, lo que indica una predicción más precisa del campo potencial y se acopla a los resultados obtenidos en secciones anteriores.
Con los resultados observados, se recomienda entrenar el modelo utilizando dominios de tamaño cercano a 1 unidad de longitud —lo cual puede lograrse mediante escalamiento o normalización—, en coherencia con la estrategia de normalización de datos aplicada en las secciones anteriores.

4.3.3.2. Variación en nodos de entrenamiento

Para las siguientes pruebas, se tiene como objetivo determinar si el número de nodos de malla afecta el desempeño del modelo. Para ello, se entrenó la arquitectura UNet4-ks3-rf200 con diferentes cantidades de nodos, variando entre 61 y 141 nodos, manteniendo constante los demás parámetros de entrenamiento utilizados en la Sección 4.3.1.1.
Este rango de nodos se eligió para evaluar el impacto de la resolución espacial en la capacidad del modelo para aprender y generalizar, considerando que un número excesivo de nodos eleva

considerablemente el costo computacional y el tiempo de entrenamiento, especialmente para problemas en casos de tres dimensiones, por lo que es importante determinar el número mínimo de nodos necesario para lograr una convergencia adecuada del modelo. Los campos de potencial obtenidos para cada configuración se presentan en el Anexo D, y a continuación se muestran los resultados correspondientes:

Cuadro 4.9: Comparación del error y R^2 con diferente número de nodos

Caso	**# Nodos**	**Error Máx. (%)**	**Error Prom. (%)**	**R^2**
Caso 1	61	94.72	3.36	-0.9021
Caso 2	71	85.13	3.87	-0.6479
Caso 3	81	92.94	2.90	-0.6748
Caso 4	91	94.23	3.31	-1.1357
Caso 5	111	90.92	3.57	-1.4623
Caso 6	121	103.70	2.67	-1.1529
Caso 7	131	91.22	3.25	-1.3877
Caso 8	141	91.35	3.38	-0.8842

Los resultados presentados en la Tabla 4.9 muestran que el número de nodos en la malla no afecta de manera significativa el desempeño global del modelo, ya que tanto los errores como el coeficiente de determinación R^2 se mantienen dentro de rangos similares en todas las configuraciones evaluadas.
Sin embargo, al examinar los campos de potencial —particularmente en la Figura D.1— se evidencia que el modelo no converge adecuadamente con 61 nodos, generando un campo potencial no uniforme. A partir de 71 nodos, en cambio, el modelo predice correctamente el campo, sin observarse diferencias significativas ni en la distribución del potencial ni en los errores asociados. Por lo tanto, se concluye que el modelo es capaz de converger adecuadamente con un número mínimo de nodos de 71, lo que permite reducir el costo computacional y el tiempo de entrenamiento, sin comprometer la calidad de la solución.

4.3.4. Generalización en el modelo

Para el siguiente estudio, se busca determinar la capacidad de generalización del modelo. Para lo cual, se utilizará la arquitectura Unet4-ks3-rf200, previamente entrenada, para resolver diferentes casos de problemas de carga puntual, variando la cantidad, amplitud y localización de estas. A continuación se presentan los parámetros de entrenamiento utilizados:

- Arquitectura: Unet4-ks3-rf200
- Cantidad de nodos: 101 x 101
- Cantidad de epochs: 50

- Tasa de aprendizaje: 0.001
- Normalización: $\alpha = 0.1$
- Pesos de las funciones de perdida: *laplacian loss* = 1 x 10^5 y *dirichlet loss* = 1.0
- Rango de ampltud de cargas puntuales: [-1, 1]

4.3.4.1. Cargas puntuales en diferentes localizaciones

Para este estudio, se busca determinar si el modelo mantiene su capacidad de resolución cuando la carga puntual no se encuentra centrada, para esto, se utilizara el modelo para resolver una carga puntual localizada en diferentes puntos del dominio y variando su amplitud de manera aleatoria. A continuación, se presentan los resultados obtenidos, mientras que los campos de potencial generados se incluyen en el Anexo E.

Cuadro 4.10: Errores promedios y máximos para cargas puntuales en diferentes localizaciones

Caso	**Posición** (x_1, y_1)	**Amplitud** A_1	**Error Máx. (%)**	**Error Prom. (%)**	$\mathbf{R^2}$
Caso 1	(2.23, 4.08)	13.22	55.86	28.72	-2.2884
Caso 2	(0.71, 4.59)	10.92	66.91	31.51	-3.5776
Caso 3	(3.76, 0.86)	12.83	63.15	26.70	-2.4716
Caso 4	(1.93, 4.38)	3.51	68.49	45.71	-6.2008
Caso 5	(3.92, 2.25)	9.32	56.16	27.50	-2.7354
Caso 6	(2.74, 0.58)	-3.18	109.57	10.75	0.2239
Caso 7	(3.08, 3.64)	14.83	56.69	25.68	-2.0696
Caso 8	(4.57, 4.23)	0.64	171.96	143.91	-60.6859
Caso 9	(1.16, 2.35)	13.35	67.18	19.91	-2.2338
Caso 10	(0.98, 3.31)	-8.69	98.17	17.07	-0.4579
Caso 11	(2.25, 4.10)	13.22	93.18	4.43	-1.9909
Caso 12	(0.70, 4.60)	10.92	93.97	5.67	-3.1819
Caso 13	(3.75, 0.85)	12.83	93.38	4.79	-2.1807
Caso 14	(1.95, 4.40)	3.51	90.45	7.06	-5.4660
Caso 15	(3.90, 2.25)	9.32	92.47	4.66	-2.3568

Los resultados obtenidos en la Tabla 4.10 muestran que el modelo es capaz de resolver cargas puntuales ubicadas en diferentes posiciones del dominio, sin embargo, se observan comportamientos inesperados en algunos casos. Para los primeros 10 casos, se nota que se obtienen valores de error promedio mayores de lo esperado, mientras que los errores máximos son significativamente menores, tras un análisis, se observa que esto se debe a que la posición de la carga puntual no se encuentra ubicada en nodo de malla, lo que podría explicar este comportamiento anómalo por dos motivos: el primero es que al colocar la Gausseana fuera de un nodo, esta se reparte dentro de los cuatro nodos más cercanos, lo que "reparte" la cumbre de la misma y disminuye su amplitud entre un 10 % y 15 % (debido a las dimensiones de celda utilizados),

lo que genera un error promedio mayor. El segundo motivo viene dado por el factor de que la solución analítica del problema es calculada de forma continua en el dominio, lo que conlleva que el punto de mayor error (justo ubicada en el centro de la carga) no se considera dentro del calculo del error, por lo que el error máximo es menor al esperado. Adicionalmente, se observa que para el caso 8 el modelo presenta un error promedio y máximo muy elevados, lo cual se puede deber a que la carga puntual se encuentra localizada muy cerca del borde del dominio, lo que genera que exista un ruido en el campo por la los hiperparámetros entrenados utilizando las condiciones de borde. Para confirmar esta hipótesis, se dará solución a distintas cargas puntuales con amplitud aleatoria ubicadas a lo largo del borde del dominio y si analizaran sus errores. Cabe mencionar que los campos de potencial generados muestran buena estabilidad, lo que es un buen indicador de la capacidad de generalización del modelo.

Cuadro 4.11: Errores promedios y máximos para cargas puntuales ubicadas a lo largo del borde del dominio

Caso	**Posición** (x_1, y_1)	**Amplitud** A_1	**Error Máx. (%)**	**Error Prom. (%)**	$\mathbf{R^2}$
Caso 16	(0.22, 4.93)	10.92	78.96	40.65	-6.0187
Caso 17	(0.49, 0.38)	13.58	69.15	34.58	-3.7283
Caso 18	(4.73, 0.19)	17.80	77.26	37.63	-4.5370
Caso 19	(4.91, 4.72)	-3.18	118.30	16.01	-0.4248
Caso 20	(0.03, 4.91)	8.95	83.53	40.36	-7.5263
Caso 21	(0.18, 4.99)	16.79	86.16	38.41	-5.8921
Caso 22	(4.60, 4.73)	-8.69	100.70	22.75	-1.6345
Caso 23	(4.84, 0.37)	19.03	77.46	35.21	-4.2960
Caso 24	(0.19, 0.23)	-4.32	107.10	18.50	-0.7742
Caso 25	(0.24, 0.11)	10.09	81.87	36.35	-5.8858

En la Tabla 4.11 se presentan los resultados obtenidos al evaluar el modelo con cargas puntuales ubicadas a lo largo del borde del dominio. Si bien los errores máximos y promedio se mantienen dentro de un rango aceptable, la inspección visual de los campos de potencial generados —como se muestra en la Figura 4.34— revela una marcada inestabilidad, particularmente en las proximidades de las cargas. Esto evidencia que el modelo no logra representar de manera adecuada las cargas puntuales situadas cerca del borde del dominio.

De este modo, se puede afirmar que el modelo representa correctamente las cargas puntuales, aunque se recomienda que estas se encuentren ubicadas en nodos de malla y en las lejanías del borde (al menos 10 veces el valor de σ) para evitar problemas de estabilidad en el campo y en los errores.

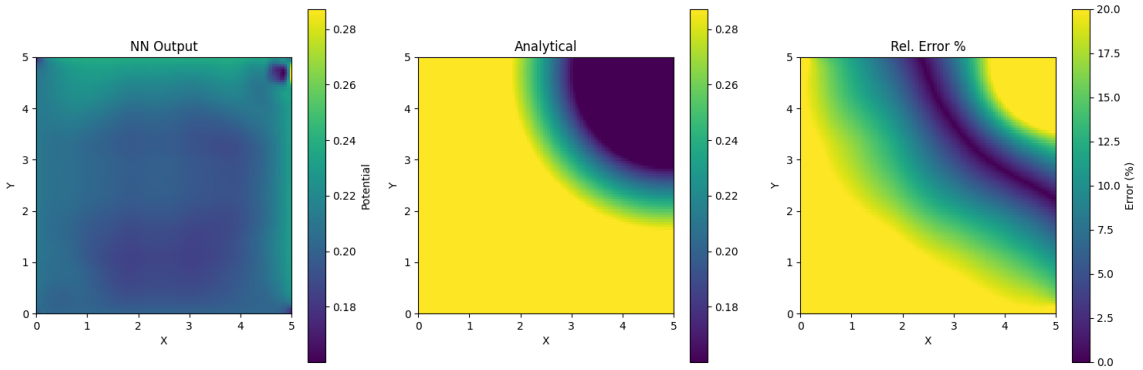


Figura 4.34: Resultados obtenidos para cargas puntuales ubicadas a lo largo del borde del dominio

4.3.4.2. Casos con multiples cargas puntuales

En los casos siguientes se analiza el desempeño del modelo al predecir campos de potencial generados por múltiples cargas distribuidas dentro del dominio, variando tanto su amplitud como su localización. Para ello, se resuelven distintos escenarios aleatorios, siguiendo las recomendaciones establecidas en la sección anterior, considerando configuraciones con un número de cargas que varía entre 2 y 10. Los campos de potencial obtenidos para cada caso se presentan en el Anexo F, y a continuación se detallan los resultados cuantitativos correspondientes:
A partir de los resultados obtenidos, se observa que el modelo logra resolver adecuadamente los distintos casos con múltiples cargas puntuales, manteniendo los errores promedio y máximos dentro de un orden aceptable. No obstante, se identifican ciertos casos anómalos, como los casos 11 (Fig. F.11), 16 (Fig. F.16) y 20 (Fig. F.20), en los cuales los errores promedio son significativamente mayores en comparación con el resto; 12.78 %, 13.24 % y 20.07 % respectivamente. El análisis de los campos de potencial generados, revela que dichos casos corresponden a configuraciones con cargas puntuales de alta amplitud, ubicadas a grandes distancias entre sí dentro del dominio. Esta disposición puede inducir la presencia de múltiples singularidades separadas espacialmente, lo cual puede comprometer la estabilidad del modelo y explicar el incremento en los errores promedio.
Para validar esta hipótesis, se propone estudiar casos con cargas puntuales ubicadas muy próximas entre sí, y posteriormente aumentar gradualmente la distancia entre ellas, evaluando el impacto en el error. Además, se analizará cómo influye la amplitud de las cargas en función de su separación. En este experimento, se fijará la coordenada vertical de ambas cargas en 2.5 unidades, variando únicamente su posición horizontal para controlar la distancia relativa.
Los resultados presentados en la Tabla 4.13 evidencian que, a medida que dos cargas puntuales de gran magnitud se separan dentro del dominio, el error promedio tiende a incrementarse. Esto

Cuadro 4.12: Errores de predicción por número de cargas

Caso	**# Cargas**	**Error Máx. (%)**	**Error Prom. (%)**	$\mathbf{R^2}$
Caso 1	2	101.64	1.52	-0.1925
Caso 2	2	92.26	6.91	-4.3307
Caso 3	3	93.58	2.79	-1.0688
Caso 4	3	90.76	5.38	-0.5424
Caso 5	3	90.07	9.86	-2.8407
Caso 6	4	92.97	6.05	-3.3347
Caso 7	5	100.43	1.75	-0.0236
Caso 8	5	92.24	2.46	0.0970
Caso 9	5	90.36	5.45	-0.9872
Caso 10	6	92.95	3.08	-1.2397
Caso 11	6	91.61	12.78	-3.1663
Caso 12	7	93.44	4.30	-2.7283
Caso 13	7	92.04	9.69	-2.7029
Caso 14	8	92.35	10.87	-4.8255
Caso 15	8	91.48	3.13	-0.4263
Caso 16	8	90.80	13.24	-4.1862
Caso 17	9	92.95	3.93	0.1328
Caso 18	9	92.04	7.21	-2.7568
Caso 19	10	90.87	3.64	0.0796
Caso 20	10	88.79	20.07	-6.7223

se observa desde el caso 21, donde las cargas se encuentran a una distancia de 0.5 unidades y el error promedio alcanza un 6.99 %, hasta el caso 26, con una separación de 3.5 unidades y un error promedio de 12.49 %. Complementariamente, se aprecia que los errores promedio y máximos son considerablemente mayores cuando las cargas presentan una amplitud reducida. En los casos 27 a 30, los errores promedio superan el 10 %, mientras que los errores máximos alcanzan valores por sobre el 100 %. Este comportamiento sugiere que el modelo presenta dificultades para representar cargas de baja amplitud, posiblemente debido a una limitada capacidad de la red para aprender las características sutiles asociadas a estas configuraciones.

Por otro lado, en los casos 31 a 34, donde se consideran cargas de igual magnitud pero con signos opuestos, se observa una reducción en los errores promedio y una menor sensibilidad respecto a la distancia entre las cargas. Esto indica que el modelo logra representar con mayor eficacia configuraciones con simetría de signo, posiblemente debido a una mejor cancelación de efectos singulares en el campo.

En síntesis, los resultados muestran que el desempeño del modelo se ve afectado significativamente por la magnitud, separación y polaridad relativa de las cargas puntuales. Mientras las cargas de gran magnitud y cercanas entre sí son representadas con mayor precisión, aquellas de baja amplitud o altamente separadas generan mayor inestabilidad. En contraste, la presencia

Cuadro 4.13: Errores de predicción para pares de cargas gaussianas

Caso	**Amp. 1**	**Amp. 2**	**$x_0$1**	**$x_0$2**	**Error Máx. (%)**	**Error Prom. (%)**	R^2
Caso 21	10	10	2.0	2.5	90.52	6.99	-1.9066
Caso 22	10	10	2.0	3.0	91.29	7.25	-2.3472
Caso 23	10	10	2.0	3.5	91.61	8.54	-2.9444
Caso 24	10	10	1.5	3.5	91.98	9.80	-3.8116
Caso 25	10	10	1.0	3.5	92.18	10.15	-4.8404
Caso 26	10	10	1.0	4.0	92.26	12.49	-6.1306
Caso 27	1	1	2.0	2.5	79.27	16.87	-13.9718
Caso 28	1	1	2.0	2.0	87.28	8.74	-11.8297
Caso 29	1	1	1.5	3.0	80.75	17.90	-19.0952
Caso 30	1	1	1.0	3.0	81.04	18.38	-22.1691
Caso 31	1	-1	2.0	2.5	111.15	11.91	-39.2932
Caso 32	1	-1	2.0	3.0	108.64	11.45	-19.4106
Caso 33	1	-1	2.0	3.5	107.87	11.09	-11.8693
Caso 34	1	-1	1.5	3.5	107.36	11.10	-8.6702
Caso 35	10	-10	1.5	3.5	99.78	2.53	0.1208
Caso 36	10	-10	1.5	3.0	100.17	2.27	-0.0001
Caso 37	10	-10	2.0	3.0	100.67	1.76	-0.1137
Caso 38	10	-10	2.5	3.0	102.36	1.58	-0.6018

de cargas opuestas mejora la estabilidad del campo, lo que sugiere una capacidad del modelo para capturar simetrías estructurales del problema.

4.4. Dominios 2D con interfaces

Para validar los resultados obtenidos en los casos bidimensionales con interfaces, se desarrolló una solución analítica correspondiente a un dominio con una interfaz circular y una carga puntual ubicada en el origen. El objetivo es evaluar la capacidad del modelo para resolver problemas en presencia de discontinuidades en la permitividad del medio, lo cual introduce un desafío adicional en la resolución de la ecuación de Poisson.

Como primer paso, se obtuvo la solución de la ecuación de Laplace en dos dimensiones utilizando coordenadas polares (ver detalles de la derivación en el Anexo [A]). A partir de esta formulación, se deduce la expresión del potencial electrostático para un dominio circular con una interfaz concéntrica y una carga puntual centrada, la cual se expresa como:

$$\begin{aligned}\phi_1 &= \frac{q}{2\pi\epsilon_1} ln\left(\frac{R}{r}\right) - \frac{q}{2\pi\epsilon_2} ln(R) \\ \phi_2 &= \frac{-q}{2\pi\epsilon_2} ln(r)\end{aligned} \tag{4.3}$$

En donde R es el radio de la interfaz, q es el valor de la carga, ϵ_1 y ϵ_2 son la permitividad de la región interior y exterior respectivamente. Para más detalles de la derivación revisar Anexo [B]. En las siguientes secciones, se resolver el problema de una carga puntual centrada en un dominio con interfaces considerando diferentes casos, variando parámetros como la amplitud de la carga, el radio de la interfaz y la permitividad de las regiones para determinar la capacidad del modelo de resolver este tipo de problemas.

4.4.1. Estudio de arquitecturas

Como primer estudio, se busca determinar que arquitecturas son capaces de resolver este tipo de problemas mas complejos que los anteriores. Para esto, se entrenaron diferentes arquitecturas con los siguientes parámetros de entrenamiento:

- Arquitectura: UNet4-ks3-rf200, UNet4-ks3-rf300, UNet4-ks3-rf400.
- Cantidad de nodos: 101 x 101 en un dominio de 1 x 1 unidades de longitud.
- Cantidad de epochs: 30.
- Tasa de aprendizaje: 1 x 10^{-5}.
- Normalización: $\alpha = 0.1$.
- Pesos de las funciones de perdida *laplacian loss* = 1.0 x 10^5, *dirichlet loss* = 1.0 y *interface loss* = 1.0 x 10^5.

El caso a resolver sera el siguiente, el cual se presenta en la Figura 4.35:

- Carga puntual centrada en el origen con valor $q = 18.0$.
- Radio de la interfaz $R = 0.3$.
- Permitividad de la región interior $\epsilon_1 = 1.0$.
- Permitividad de la región exterior $\epsilon_2 = 80.0$.

A continuación, se presentan los resultados obtenidos.

Cuadro 4.14: Comparación de arquitecturas por errores y varianza R^2

Arquitectura	**Error Máx. (%)**	**Error Prom. (%)**	**R^2**
UNet4-ks3-rf200	15.61	1.98	0.8572
UNet4-ks3-rf300	37.78	2.24	0.8109
UNet4-ks3-rf400	60.76	2.14	0.7481

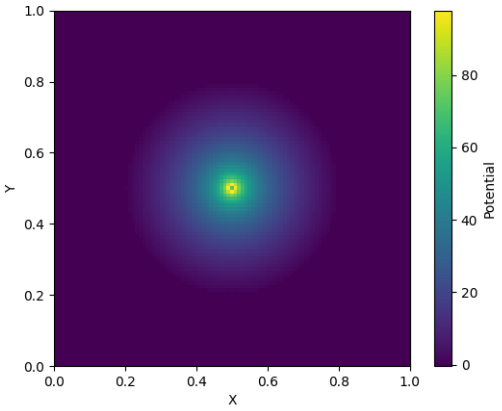


Figura 4.35: Caso de estudio: Carga puntual centrada con interfaz circular

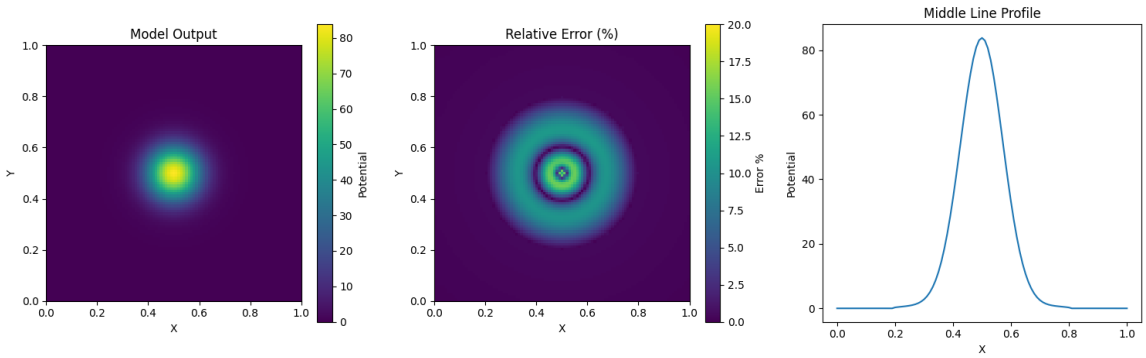


Figura 4.36: Resultados obtenidos para UNet4-ks3-rf200 con interfaz

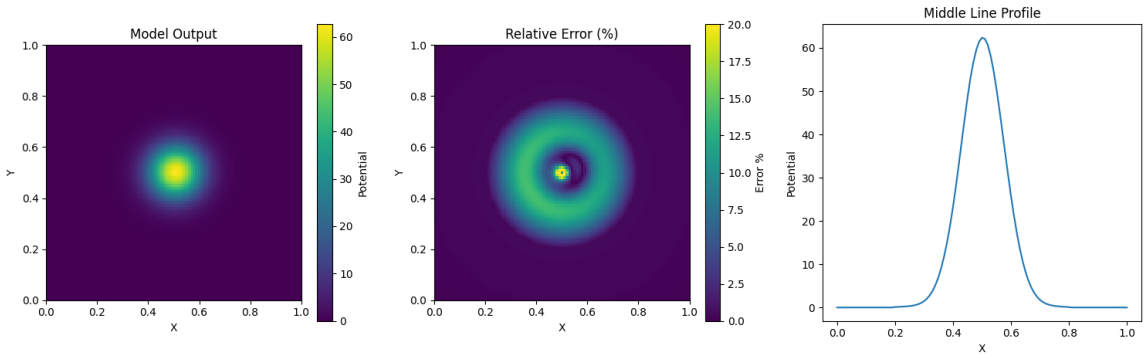


Figura 4.37: Resultados obtenidos para UNet4-ks3-rf300 con interfaz

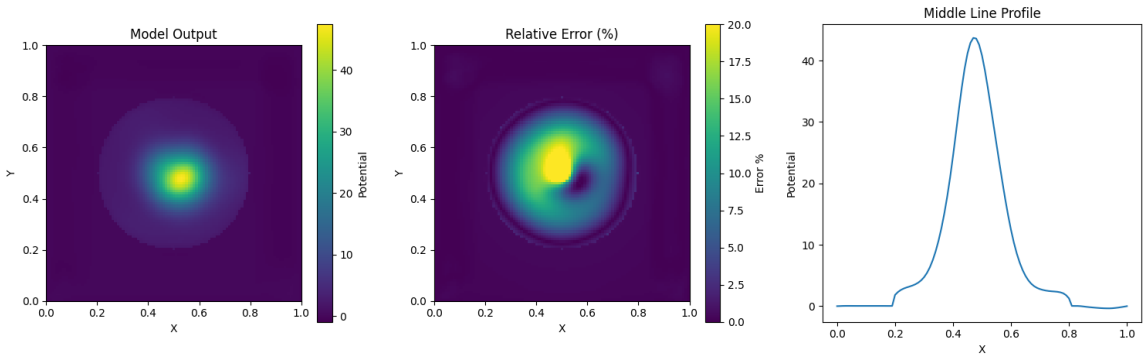


Figura 4.38: Resultados obtenidos para UNet4-ks3-rf400 con interfaz

Los resultados muestran que, para la configuración UNet4-ks3-rf200, el modelo resuelve satisfactoriamente el problema propuesto, alcanzando un error promedio de 1.98 % y un error máximo de 15.61 %, ambos valores mejoran lo reportado en las secciones anteriores. Este desempeño se explica por la arquitectura bifurcada empleada. La primera sub-red convolucional se especializa en capturar el pronunciado gradiente que genera la carga puntual, reproduciendo con mayor fidelidad el comportamiento tipo delta de Dirac en la región interna del dominio, donde antes se concentraba la mayor parte del error. En paralelo, la segunda sub-red genera un campo de potencial suave en la región externa, evitando que los efectos de suavizado alteren la solución cerca de la carga. La combinación de ambas contribuciones permite al modelo representar simultáneamente las singularidades locales y la variación suave global, reduciendo de forma notable los errores globales y locales.

Para la configuración UNet4-ks3-rf300, se observa que el modelo logra una solución aceptable, con un error promedio de 2.24 % y un error máximo de 37.78 %. No obstante, el campo de potencial resultante presenta una asimetría significativa, lo que indica una falta de estabilidad en la solución. Esta inestabilidad explica el aumento considerable del error máximo en comparación con el caso anterior. El problema se acentúa en la configuración UNet4-ks3-rf400, donde, si bien el error promedio disminuye levemente a 2.14 %, el error máximo se incrementa de forma considerable alcanzando un 60.76 %. Además, el campo de potencial generado es marcadamente inestable, como se evidencia en la Figura 4.38. Estos resultados sugieren que para este tipo de problemas, redes con mayor cantidad de parámetros genera problemas de estabilidad, posiblemente debido a el modelo trabaja en espacios de menor dimension (menor número de nodos) y por lo tanto, la cantidad de parámetros es excesiva para el problema a resolver, generando un sobreajuste de los datos de predicción.

De esta manera, se concluye que la arquitectura UNet4-ks3-rf200 es la más adecuada para resolver problemas con interfaces en 2D, ya que logra un equilibrio entre la capacidad de modelar

singularidades locales y la variación suave del campo, ademas se observa que la elección de parámetros para el entrenamiento es la indicada, ya que la red predice correctamente cada region del dominio, sin generar problemas de estabilidad en el campo de potencial.

4.5. Condiciones de Borde No Homogéneas en 2D

En las pruebas siguientes se evalúa la capacidad del modelo para representar soluciones con condiciones de borde no homogéneas. Para ello, se considera un caso en el que el campo escalar se define de manera explícita, lo que permite calcular de forma directa el término fuente al aplicar el operador Laplaciano sobre la función conocida. La función de potencial utilizada en este experimento es la siguiente:

$$\begin{aligned} \phi &= x^3 + y^3 \\ \nabla^2\phi &= 6x + 6y \\ f &= 6x + 6y \end{aligned} \tag{4.4}$$

A partir de esta función, se determinan tanto los valores del campo como las condiciones de borde necesarias para el entrenamiento. El modelo se entrena utilizando una combinación de funciones de pérdida: *Laplacian loss*, que evalúa el cumplimiento de la ecuación en el dominio, y *Dirichlet loss*, aplicada sobre los nodos de frontera definidos por la solución analítica.
Adicionalmente, este caso permite analizar la capacidad del modelo para aproximar funciones suaves, lo cual resulta relevante dado que en las pruebas anteriores se evidenciaron dificultades al representar campos generados por cargas puntuales, debido a la presencia de singularidades con valores extremos. Evaluar el desempeño del modelo ante funciones suaves constituye, por tanto, un paso clave para entender sus limitaciones y fortalezas en diferentes regímenes de solución

4.5.1. Normalización del las Condiciones de Borde

Como primera prueba, se busca determinar si la normalización de las condiciones de borde es necesaria para el correcto funcionamiento del modelo. En la sección anterior, las redes fueron entrenadas sin aplicar normalización en los valores de frontera; sin embargo, dado que en este caso las condiciones de borde no son homogéneas (es decir, no están igualadas a cero), surge la interrogante de si dicha normalización podría mejorar la estabilidad y precisión del entrenamiento.

Para abordar esta cuestión, se entrenó una red bajo dos escenarios: con y sin normalización de las condiciones de borde. A continuación, se detallan las condiciones utilizadas en ambos entrenamientos:

- Arquitectura: UNet4-ks3-rf200.
- Cantidad de nodos: 101 x 101 en un dominio de 5 x 5 unidades de longitud.
- Cantidad de epochs: 30.
- Tasa de aprendizaje: 1 x 10^{-5}.
- Normalización: $\alpha = 0.1$.
- Pesos de las funciones de perdida *laplacian loss* = 8.0 x 10^5 y *dirichlet loss* = 8.0

A continuación, se presentan los resultados obtenidos.

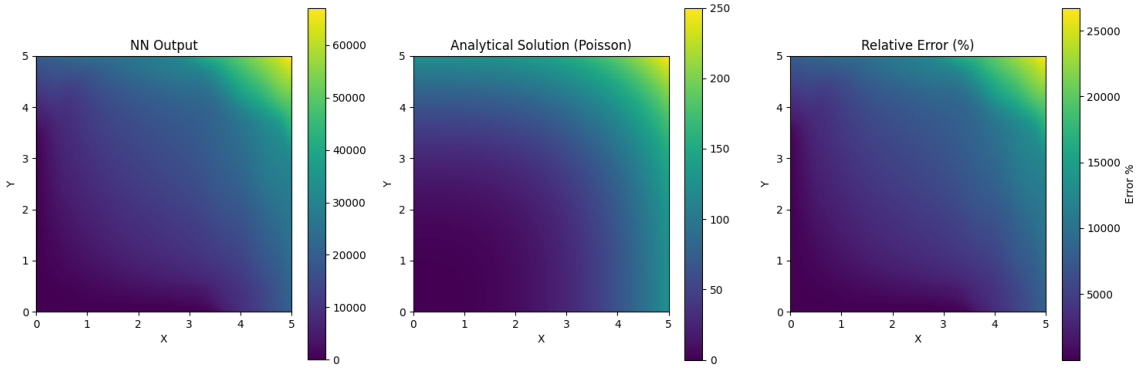


Figura 4.39: Resultados obtenidos para UNet4-ks3-rf200 con condiciones de borde no normalizadas

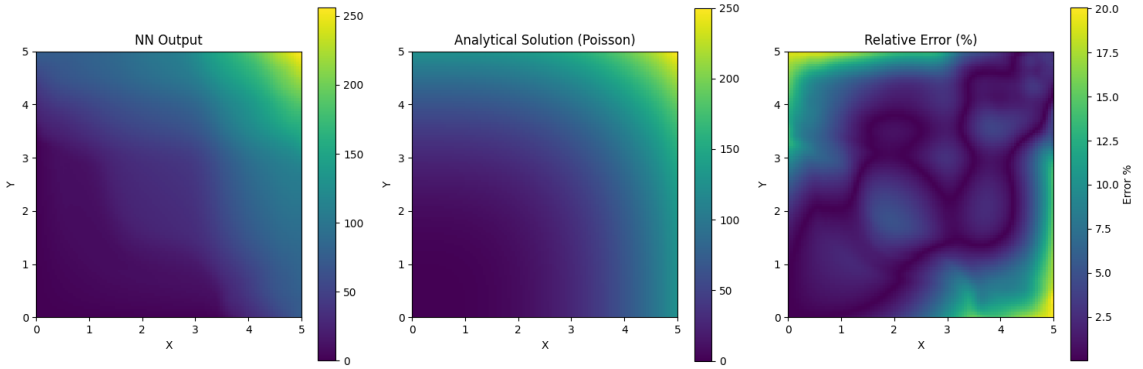


Figura 4.40: Resultados obtenidos para UNet4-ks3-rf200 con condiciones de borde normalizadas

De los resultados obtenidos, se observa que el modelo entrenado con condiciones de borde normalizadas representadas en la Figura 4.40 logra converger a la solución esperada, con un error promedio de 3.96 % y un error máximo de 17.61 %. En contraste, el modelo entrenado sin normalizar las condiciones de borde, como se muestra en la Figura 4.39, representa el campo

Cuadro 4.15: Comparación del efecto de normalización en la predicción

Caso	**Error Máx. (%)**	**Error Prom. (%)**	$\mathbf{R^2}$
Con normalizar	17.61	3.96	0.9322
Sin normalizar	6284.28	1460.65	-8068.9112

del potencial pero la magnitud de los valores se aleja significativamente de la solución esperada, con un error promedio de 1460.65 % y un error máximo de 6284.28 %, lo cual indica que el modelo necesita de una normalización de las condiciones de borde para poder converger a la solución esperada situaciones que no se tengan condiciones de frontera homogéneas.
Es relevante destacar que el error máximo se concentra principalmente en los bordes del dominio, mientras que la región central del campo presenta errores bajos y una distribución estable. Este comportamiento sugiere que el modelo ha logrado aprender y representar adecuadamente la función suave en el interior del dominio, aunque persisten dificultades para reproducir con precisión las condiciones de frontera no homogéneas. A pesar de estas limitaciones en la zona de frontera, se observa una mejora significativa en el desempeño del modelo al tratar con funciones suaves. En particular, el error máximo se reduce considerablemente en comparación con los casos analizados previamente, donde la presencia de cargas puntuales introducía singularidades que afectaban negativamente la precisión del modelo.

4.5.2. Arquitecturas

A continuación, se realiza un comparativo de diferentes arquitecturas para evaluar el desempeño de estas en este nuevo tipo de problemas. En este caso, se entrenaron diversas redes manteniendo los párametros de entrenamiento de la sección anterior, con el fin de determinar cuál arquitectura es más adecuada para resolver problemas con condiciones de borde no homogéneas. A continuación, se presentan los resultados obtenidos para cada caso, los campos generados por cada modelo se presentan en el Anexo G:

Cuadro 4.16: Comparación de arquitecturas por errores y varianza R^2

Caso	**Arquitectura**	**Error Máx. (%)**	**Error Prom. (%)**	$\mathbf{R^2}$
Caso 1	Unet4 - rf100	19.45	6.78	0.8315
Caso 2	UNet5 - rf200	19.35	5.47	0.8710
Caso 3	UNet4 - rf300	21.50	7.62	0.7744
Caso 4	UNet5 - rf300	15.56	6.80	0.8388
Caso 5	UNet4 - rf400	27.57	9.52	0.6327
Caso 6	UNet5 - rf400	26.63	6.29	0.8155
Caso 7	MSNet4 - rf200	49.02	14.57	0.2187

De los resultados presentados en la Tabla 4.16, se puede observar que la arquitectura MSNet4-ks3-rf200 presenta un desempeño significativamente inferior en comparación con las variantes UNet, obteniendo errores promedio y máximos mucho mayores, además de una varianza R^2 muy baja, lo que indica que el campo generado presenta tasa de errores altos en todo el dominio. Esto se confirma al observar la Figura G.7, donde se aprecia un campo de potencial inestable, con valores que se alejan considerablemente de la solución esperada. Por esta razón, se descarta su uso en estudios posteriores.
Entre las variantes UNet, se aprecia que la que obtuvo mejores resultados fue la UNet5-ks3-rf200 (Fig. G.2), con un error promedio de 5.47 % y un error máximo de 19.35 %, seguida de cerca por la UNet4-ks3-rf200 (Fig. G.1), que obtuvo un error promedio de 6.78 % y un error máximo de 19.45 %. Esto indica que, similar a lo observado en los casos con cargas puntuales, incrementar el tamaño de la red no necesariamente mejora la precisión del modelo. La razón de esto podría ser que el problema no requiere una red tan compleja para capturar las características del campo, y al tener una mayor cantidad de hiperparámetros, la red puede tender a sobreajustar los datos de entrenamiento generando ruido en los datos que se procesan.
Es destacable que la arquitectura UNet4-ks3-rf200 presenta los errores más altos localizados en dos esquinas del dominio, mientras que para las otras configuraciones —Figs. G.3 a G.6—, los errores máximos se concentran en una region próxima al centro del dominio.

4.5.3. Estudio de Parámetros de Funciones de Borde

Este estudio evalúa la influencia de los pesos de cada término de la función de pérdida en la capacidad del modelo para resolver problemas con condiciones de borde no homogéneas. Para ello, se entrenó una red neuronal variando los pesos asignados a las pérdidas de Dirichlet y del operador de Laplace. La metodología consistió en entrenar distintos casos, manteniendo constante uno de los pesos mientras se variaba el otro, a fin de aislar su efecto en el desempeño del modelo. Los parámetros de entrenamiento empleados en cada experimento fueron los siguientes:

- Arquitectura: UNet4-ks3-rf200.
- Cantidad de nodos: 101 x 101 en un dominio de 5 x 5 unidades de longitud.
- Cantidad de epochs: 30.
- Tasa de aprendizaje: 1 x 10^{-5}.
- Normalización: $\alpha = 0.1$.
- Pesos de las funciones de perdida *laplacian loss* y *dirichlet loss*.
- Rango de pesos de funciones de perdida: [1.0, 1.0 x 10^5]

A continuación, se presentan los resultados obtenidos para cada caso, los campos generados por

cada modelo se presentan en el Anexo G.

Cuadro 4.17: Comparación de errores y varianza R^2 según pesos de funciones de pérdida

Caso	**Peso usado**	**Error Máx. (%)**	**Error Prom. (%)**	**R^2**
Caso 8	LW: 1 x 10^1	62.74	8.50	0.6498
Caso 9	LW: 1 x 10^2	83.39	16.95	-0.0383
Caso 10	LW: 1 x 10^3	99.95	25.61	-1.5869
Caso 11	LW: 1 x 10^4	99.97	25.15	-1.5309
Caso 12	LW: 1 x 10^5	99.97	25.72	-1.6011
Caso 13	BW: 1 x 10^0	23.70	4.27	0.9131
Caso 14	BW: 1 x 10^1	22.89	8.14	0.7704
Caso 15	BW: 1 x 10^2	31.26	7.42	0.7318
Caso 16	BW: 1 x 10^3	23.93	5.86	0.8412
Caso 17	BW: 1 x 10^4	20.25	4.54	0.9071
Caso 18	BW: 1 x 10^5	28.72	4.70	0.8742

Los resultados presentados en la Tabla 4.17 evidencian que el modelo requiere una adecuada elección de los pesos en la función de pérdida para poder resolver correctamente problemas con condiciones de borde no homogéneas. En particular, cuando el peso del término Laplaciano es elevado, el modelo tiende a generar campos de potencial inestables, como se observa en las Figuras G.8 a G.12. Esto ocurre ya que la red prioriza la minimización del error asociado al término Laplaciano, descuidando la influencia del termino de las condiciones de frontera. En contraste, cuando los pesos de ambos términos en la función de pérdida son iguales, el modelo logra generar campos de potencial estables, con errores y valores de varianza R^2 aceptables, tal como se aprecia en la Figura G.13. Además, se observa que incrementar el peso del término correspondiente a las condiciones de Dirichlet no mejora significativamente la estabilidad del campo. Por lo tanto, con base en los resultados obtenidos, se recomienda utilizar pesos iguales para todos los términos de la función de pérdida al enfrentar condiciones de borde no homogéneas.

4.6. Extensión a 3D

En base a los resultados y a lo aprendido en los casos bidimensionales, se busca extender el modelo a tres dimensiones. Para esto, se resolverán los problemas equivalentes de los casos en dos dimensiones considerando que los parámetros de entrenamiento, arquitecturas y funciones de perdida fueron las óptimas para estudiar el caso en 3D.

4.6.1. Condiciones de Borde No Homogéneas

En la sección presente, se tiene como objetivo determinar el comportamiento del modelo al resolver problemas con condiciones de borde no homogéneas y funciones fuente suaves en 3D. Para esto, se utilizará la siguiente función para determinar las condiciones de borde y a su vez, el potencial del campo:

$$\begin{aligned} \phi &= x^3 + y^3 + z^3 \\ \nabla^2\phi &= 6x + 6y + 6z \\ f &= 6x + 6y + 6z \end{aligned} \tag{4.5}$$

Los parámetros de entrenamiento utilizados fueron los siguientes:

- Arquitectura: UNet4-ks3-rf200.
- Cantidad de nodos: 71 x 71 x 71 en un dominio de 5 x 5 x 5 unidades de longitud.
- Cantidad de epochs: 30.
- Tasa de aprendizaje: 1 x 10^{-5}.
- Normalización: $\alpha = 0.1$.
- Pesos de las funciones de perdida *laplacian loss* = 1.0, *dirichlet loss* = 1.0 x 10^5.

De los resultados que se observan en la Figura 4.41, se confirma la capacidad predictiva del modelo para el caso con condiciones de borde no homogéneas en tres dimensiones. El error promedio se mantiene en 4.66 %, mientras que el error máximo es de 23.20 %; ambos valores son coherentes con el caso de condiciones de borde no homogéneas en 2D. Asimismo, el coeficiente de determinación $R^2 = 0.8706$ revela una baja dispersión de los errores. Ademas se aprecia que la red reproduce un campo de potencial suave y consistente en la mayor parte del dominio. El mapa de errores puntual muestra que las desviaciones se concentran en una única zona próxima al contorno, específicamente en la esquina inferior derecha del corte en z, donde se aprecia el máximo error en un punto, el cual disminuye rápidamente. Este comportamiento es similar al caso en 2D, donde el modelo también presenta los errores máximos en los bordes del dominio. En resumen, se afirma que el modelo es capaz de resolver problemas con condiciones de borde no homogéneas en tres dimensiones de forma similar a como lo hace en dos dimensiones. Presentando errores del mismo orden de magnitud y con un comportamiento similar en cuanto a la distribución del mismo.

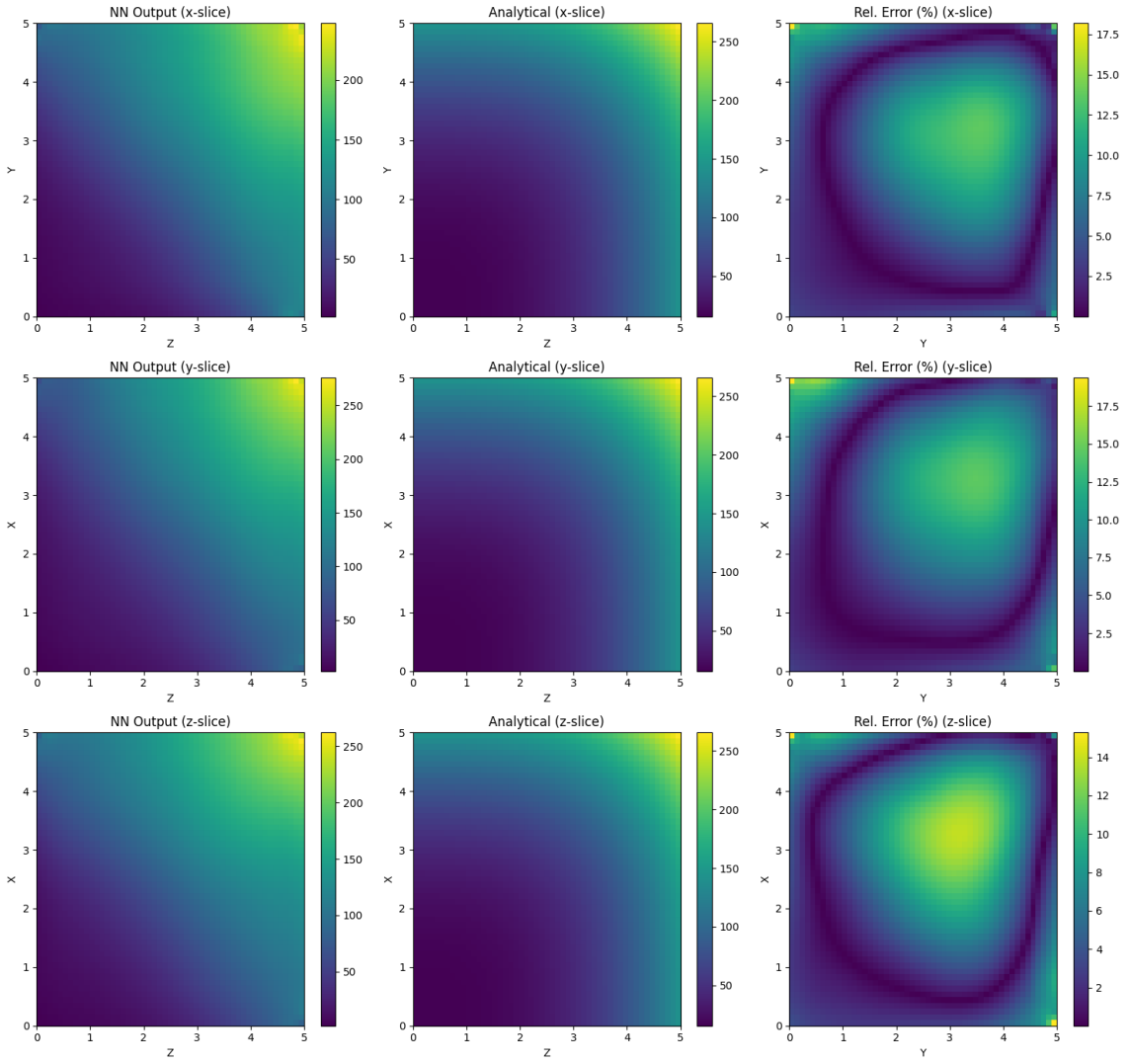


Figura 4.41: Resultados obtenidos para UNet4-ks3-rf200 en 3D

4.6.2. Carga Puntual Centrada con Condiciones de Borde Homogéneas

Para esta primera prueba en 3D, se busca determinar si el modelo es capaz de resolver los problemas de cargas puntuales en un dominio de tres dimensiones. Con este objetivo, se considera la solución de cargas puntuales colocadas en un dominio, la cual es la siguiente:

$$\phi(r) = \sum_{i=1}^{N} \frac{q_i}{4\pi\epsilon} \frac{1}{|r - r_i|} \tag{4.6}$$

En donde q_i es la carga puntual, ϵ es la permitividad del medio y r_i es la posición de la carga. Los parámetros de entrenamiento utilizados fueron los siguientes:

- Arquitectura: UNet4-ks3-rf200.
- Cantidad de nodos: 71 x 71 x 71 en un dominio de 5 x 5 x 5 unidades de longitud.
- Cantidad de epochs: 30.
- Tasa de aprendizaje: 1 x 10^{-5}.
- Normalización: $\alpha = 0.1$.
- Pesos de las funciones de perdida *laplacian loss* = 1.0 x 10^5 y
- Permitividad = 1.0. *dirichlet loss* = 1.0.

A continuación, se presentan los resultados obtenidos para diferentes casos de cargas ubicadas en el dominio, variando la cantidad de cargas, su amplitud y su posición. Los campos generados por cada modelo se presentan en el Anexo H.

Cuadro 4.18: Resultados obtenidos para cargas puntuales en 3D

Caso	**Cargas**	**Error Máx. (%)**	**Error Prom. (%)**	$\mathbf{R^2}$
Caso 1	1	90.35	16.43	0.5910
Caso 2	2	92.32	17.75	0.6347
Caso 3	3	96.45	17.97	0.5877
Caso 4	4	96.62	18.12	0.6392
Caso 5	5	96.51	18.53	0.5916
Caso 6	6	97.64	20.17	0.7671
Caso 7	7	98.19	20.31	1.0238
Caso 8	8	99.41	21.69	0.8031
Caso 9	9	100.14	21.71	0.7638
Caso 10	10	101.07	21.82	0.7042

De los resultados obtenidos, se observa que el modelo presenta un desempeño desfavorable al resolver problemas de cargas puntuales en 3D. En la Tabla 4.21 se aprecia que el error máximo conserva el mismo orden de magnitud observado en 2D, mientras que el error promedio es significativamente mayor. El análisis de las figuras revela zonas con errores elevados próximas a las fronteras del dominio (véase la Figura H.1). Esta discrepancia podría atribuirse a que la predicción del modelo se anula con demasiada rapidez en los bordes, mientras que en la solución analítica el potencial tiende a infinito.

Para comprobar esta hipótesis se evaluó el modelo en un dominio de mayor tamaño, con la expectativa de obtener un potencial más estable. En este caso, se consideró un dominio de 12 x 12 x 12 unidades de longitud, manteniendo las demás condiciones constantes.

En la Figura 4.42 se observa que el modelo reproduce el potencial con mayor estabilidad: el error promedio disminuye al 4.23 % y el error máximo se mantiene en 96.34 %. Estos resultados sugieren que el modelo puede resolver adecuadamente el problema de cargas puntuales en 3D, siempre que el dominio sea lo suficientemente grande para capturar la atenuación del potencial.

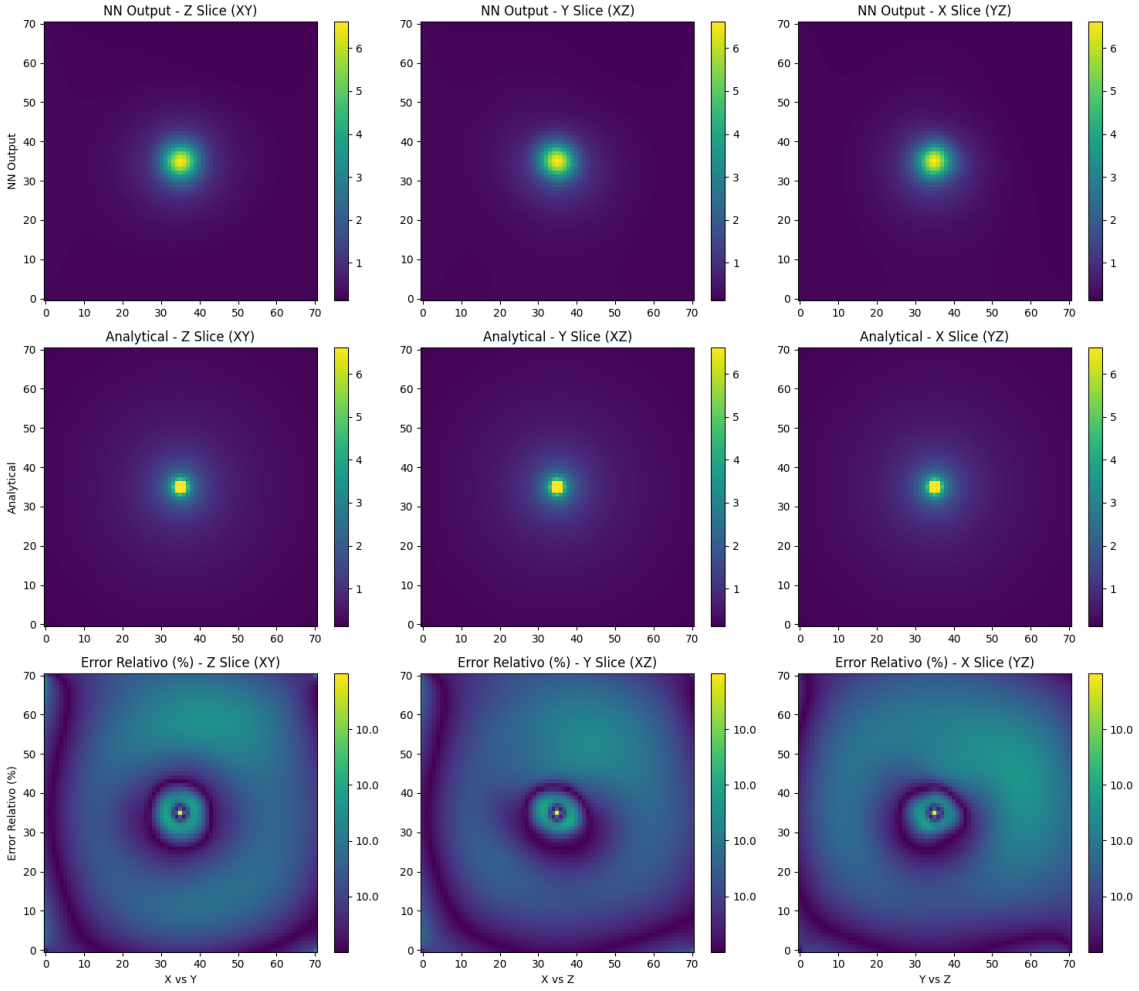


Figura 4.42: Resultados obtenidos para una carga puntual centrada en un dominio de 12 x 12 x 12 unidades de longitud

Este comportamiento concuerda con la naturaleza del potencial de una carga puntual en tres dimensiones, que decrece como $1/r$, mientras que en dos dimensiones lo hace como 1/ln(r).

4.6.3. Dominios 3D con Interfaces

Para la validación de resultados del caso en 3 dimensiones con interfaz se utilizó la teoría de los armónicos esféricos y la expansión multipolar para resolver la ecuación de Poisson en un sistema con geometría esférica. La ecuación principal es una solución aproximada del potencial $\phi(r)$, derivada de la expansión en series de armónicos esféricos:

$$\phi(r,\theta) = \sum_{n=0}^{N} \sum_{m=-n}^{n} \left(A_{mn} r^{n} + B_{nm} r^{-(n+1)}\right) Y_{n}^{m}(\phi,\theta) \tag{4.7}$$

En donde:

- $Y_n^m(\phi, \theta)$ son los armónicos esféricos
- A_{mn} y B_{mn} son los coeficientes de la expansión que dependen de las cargas q_i y de las constantes dieléctricas ϵ_i.
- $r,\ \theta,\ \phi$ son las coordenadas esféricas del punto de evaluación

El cálculo de los coeficientes A_{mn} y B_{mn} incorpora las condiciones de continuidad del potencial y del desplazamiento eléctrico en la interfaz dieléctrica (radio R):

$$\begin{aligned} A_{mn} &= \frac{E_{mn}}{4\pi} \cdot \frac{2n+1}{e^{-kR}((\epsilon_1 - \epsilon_2))nK_n(kR) + \epsilon_2(2n+1)K_{n+1}(kR)} \\ B_{mn} &= \frac{1}{R^{2n+1}} \left(e^{-kR} K_n(kR) A_{nm} - \frac{E_{mn}}{4\pi\epsilon_1} \right) \end{aligned} \tag{4.8}$$

Siendo E_{mn} los coeficientes de la expansión multipolar de la carga y K_n la función de Bessel modificada, definidos como:

$$\begin{aligned} E_{mn} &= \sum_k q_k r_k^n Y_n^m(\theta_k, \phi_k) \\ K_n(kR) &= \sum_{s=0}^{n} \frac{2^s n!(2n-s)!}{s!(2n)!(n-s)!} x^s \end{aligned} \tag{4.9}$$

Para la validación de resultados, se resolvió numéricamente esta expansión para obtener el potencial dentro del dominio. Adicionalmente, se considera la solución regularizada del potencial en coordenadas esféricas, dada por el siguiente sistema:

$$\begin{aligned} &\nabla^2 \psi^{(m)}(\mathbf{x}) = 0 \quad \mathbf{x} \in \Omega_m \\ &-\nabla^2 \phi^{(w)}(\mathbf{x}) + k_w^2 \phi^{(w)}(\mathbf{x}) = 0 \quad \mathbf{x} \in \Omega_w \\ &\psi^{(m)}(\mathbf{x}) + g_c(\mathbf{x}) = \phi^{(m)}(\mathbf{x}) \quad \mathbf{x} \in \Gamma \\ &\epsilon_m(\partial_n \psi^{(m)}(\mathbf{x}) + \partial_n g_c(\mathbf{x})) = \epsilon_w \partial_n \phi^{(w)}(\mathbf{x}) \quad \mathbf{x} \in \Gamma \\ &\phi^{(m)}(\mathbf{x} \to 0) = 0 \end{aligned} \tag{4.10}$$

En este sistema:

- ψ representa el potencial regularizado.
- g_c es el potencial de Coulomb.
- Ω_m corresponde al dominio interno de la molécula.

- Ω_w es la región del solvente.
- Γ es la interfaz entre ambas regiones.
- ϵ_m y ϵ_w son las permitividades del medio molecular y del solvente, respectivamente.

Esta formulación surge del problema de Poisson-Boltzmann aplicado a sistemas con cargas en una molécula inmersa en un solvente. La razón de emplear la versión regularizada es que produce una solución más suave, lo cual facilita el aprendizaje del campo por parte del modelo y evita la aparición de picos de error, como se evidenció en la sección anterior.

Para abordar este problema, se entrenó el modelo UNet4-ks3-rf200 utilizando los siguientes parámetros:

- Arquitectura: UNet4-ks3-rf300.
- Cantidad de nodos: 71 x 71 x 71 en un dominio de 5 x 5 x 5 unidades de longitud.
- Cantidad de epochs: 30.
- Tasa de aprendizaje: 1 x 10^{-5}.
- Normalización: $\alpha = 0.1$.
- Permitividad: $\epsilon_1 = 1.0$ y $\epsilon_2 = 80.0$.
- Radio de la interfaz: $R = 1.0$.

Dado que este caso cuenta con una solución analítica del potencial, es posible generar fácilmente los datos de entrenamiento, por lo cual, en una primera etapa, se entrena el modelo empleando únicamente la función de pérdida de interfaz *interface loss*, con el objetivo de evaluar su capacidad para ajustarse a este tipo de problemas. Posteriormente, se entrena el modelo incorporando tanto la función de pérdida *(laplacian loss)* como la función de pérdida de *dirichlet loss*, según corresponda al sistema de ecuaciones considerado —ya sea la formulación no regularizada o la regularizada, respectivamente.

4.6.3.1. Caso No Regularizado

En el primer caso, se entreno la red considerando la solución no regularizada del potencial, buscando replicar los resultados obtenidos para el caso en dos dimensiones y utilizando estos mismos como base para este nuevo desafió. Para esto, se utilizaron los siguientes pesos en las funciones de pérdida:

- *interface loss* = 1.0 x 10^5.
- *laplacian loss* = 1.0 x 10^5.
- *dirichlet loss* = 1.0.

A continuación, se presentan los resultados obtenidos para el difernetes casos de cargas puntuales ubicadas en el dominio, variando la cantidad de cargas, su amplitud y su posición. Los campos generados por cada modelo se presentan en el Anexo I:

Cuadro 4.19: Resultados obtenidos para cargas puntuales en 3D con interfaz no regularizada

Caso	**Cargas**	**Error Máx. (%)**	**Error Prom. (%)**	$\mathbf{R^2}$
Caso 1	1	16.80	2.58	0.9492
Caso 2	1	15.08	2.39	1.3959
Caso 3	2	17.38	1.93	0.9115
Caso 4	3	17.96	1.91	1.1796
Caso 5	4	18.18	1.92	0.9461

Los resultados obtenidos en la Tabla 4.19 demuestran que el modelo es capaz de resolver problemas con cargas puntuales en 3D, logrando un error promedio de 2.58 % y un error máximo de 16.80 % en el caso con una sola carga. Estos valores son comparables a los obtenidos en 2D, lo que sugiere que el modelo mantiene un desempeño similar al resolver problemas con cargas puntuales en tres dimensiones, manteniendo una capacidad predictiva adecuada.
Sin embargo, una inspección a los campos de potencial generados (Figuras I.1 al I.5) revela que el modelo presenta un grado de inestabilidad en algunos casos, especialmente en las condiciones de borde y de interfaz, zonas en donde el error es considerablemente mayor que en el resto del dominio. Esto se puede deber a diversos factores como la complejidad del problema, arquitectura utilizada, falta de entrenamiento o pobre elección de parámetros de entrenamiento. Un hipótesis plausible es la falta de nodos en el dominio, lo que podría estar limitando la capacidad del modelo para capturar adecuadamente las variaciones del campo de potencial, especialmente en las zonas cercanas a la interfaz y a las condiciones de borde. Lamentablemente, debido a las limitaciones de hardware, no fue posible aumentar la cantidad de nodos en el dominio, lo que podría haber mejorado la capacidad del modelo para resolver este tipo de problemas.
En resumen, se concluye que el modelo es capaz de resolver problemas con cargas puntuales aunque presenta gran margen de mejora, especialmente en las zonas cercanas a la interfaz y a las condiciones de borde. Se espera que el uso de una formulación regularizada del potencial mejore la capacidad del modelo para resolver este tipo de problemas debido a la suavidad de la formulación.

4.6.3.2. Caso Regularizado

Para estudiar el desempeño del modelo en el caso de la ecuación regularizada, se realizaron dos entrenamientos distintos, con variaciones en las funciones de pérdida utilizadas. Se buscó evaluar si la suavidad de esta formulación facilita el aprendizaje del modelo.

1. **Entrenamiento 1 — *inside loss*.**
   El modelo se entrenó exclusivamente con la función de pérdida *inside loss*, empleando 10.000 pares de datos entrada-objetivo generados a partir de la solución analítica. El

propósito fue verificar si la arquitectura puede reproducir el potencial regularizado penalizando únicamente el error en la región interior del dominio.

2. **Entrenamiento 2 — pérdidas combinadas.**
   El segundo experimento utilizó una combinación de *laplacian loss* (adaptada al nuevo sistema de ecuaciones), *dirichlet loss* e *interface loss*. Esta configuración busca evaluar la robustez del modelo frente a formulaciones más generales y, posteriormente, transferir el aprendizaje a problemas donde no se dispone de una solución analítica del potencial.

Cuadro 4.20: Configuraciones de cargas puntuales evaluadas en 3D.

Caso	**# Cargas**	**Amplitud**	**Ubicación**
Caso 1	1	[+1.0]	(0.1, 0.1, 0.0)
Caso 2	2	[+5.0, +6.0]	(0.4, 0.0, 0.3), (−0.3, −0.2, −0.1)
Caso 3	3	[+2.0, +1.0, +3.0]	(0.1, 0.3, 0.2), (−0.2, 0.4, −0.1), (0.0, −0.4, 0.0)
Caso 4	2	[+8.0, +6.0]	(−0.6, 0.1, 0.0), (0.2, −0.3, 0.5)
Caso 5	3	[+1.0, +2.0, +5.0]	(0.4, −0.1, 0.5), (−0.3, 0.0, −0.4), (0.0, 0.5, 0.1)

Cuadro 4.21: Errores obtenidos con *inside loss* en 3D.

Caso	**Error Máx. (%)**	**Error Prom. (%)**	R^2
Caso 1	40.90	0.53	0.9987
Caso 2	56.05	1.02	0.8395
Caso 3	43.66	1.77	0.9687
Caso 4	50.41	1.53	0.9639
Caso 5	51.88	1.74	0.9574

Los resultados de la Tabla 4.21 demuestran que la red reproduce el potencial regularizado con elevada precisión: el error promedio se mantiene por debajo del 2 % en todos los casos y el coeficiente de determinación R^2 supera el 0.95 en la mayoría de ellos. Como era previsible, los errores máximos se concentran en las proximidades de la interfaz, donde el gradiente del potencial cambia abruptamente debido a la discontinuidad en la permitividad dieléctrica. Las distribuciones de potencial predichas se encuentran en el Anexo J.

Siguiendo los hiperparámetros indicados en la Sección 4.6.3.2, se llevó a cabo el Entrenamiento 2 y se evaluó inicialmente el Caso 1. La Figura 4.43 presenta la distribución de potencial obtenida tras utilizar la combinación de pérdidas.

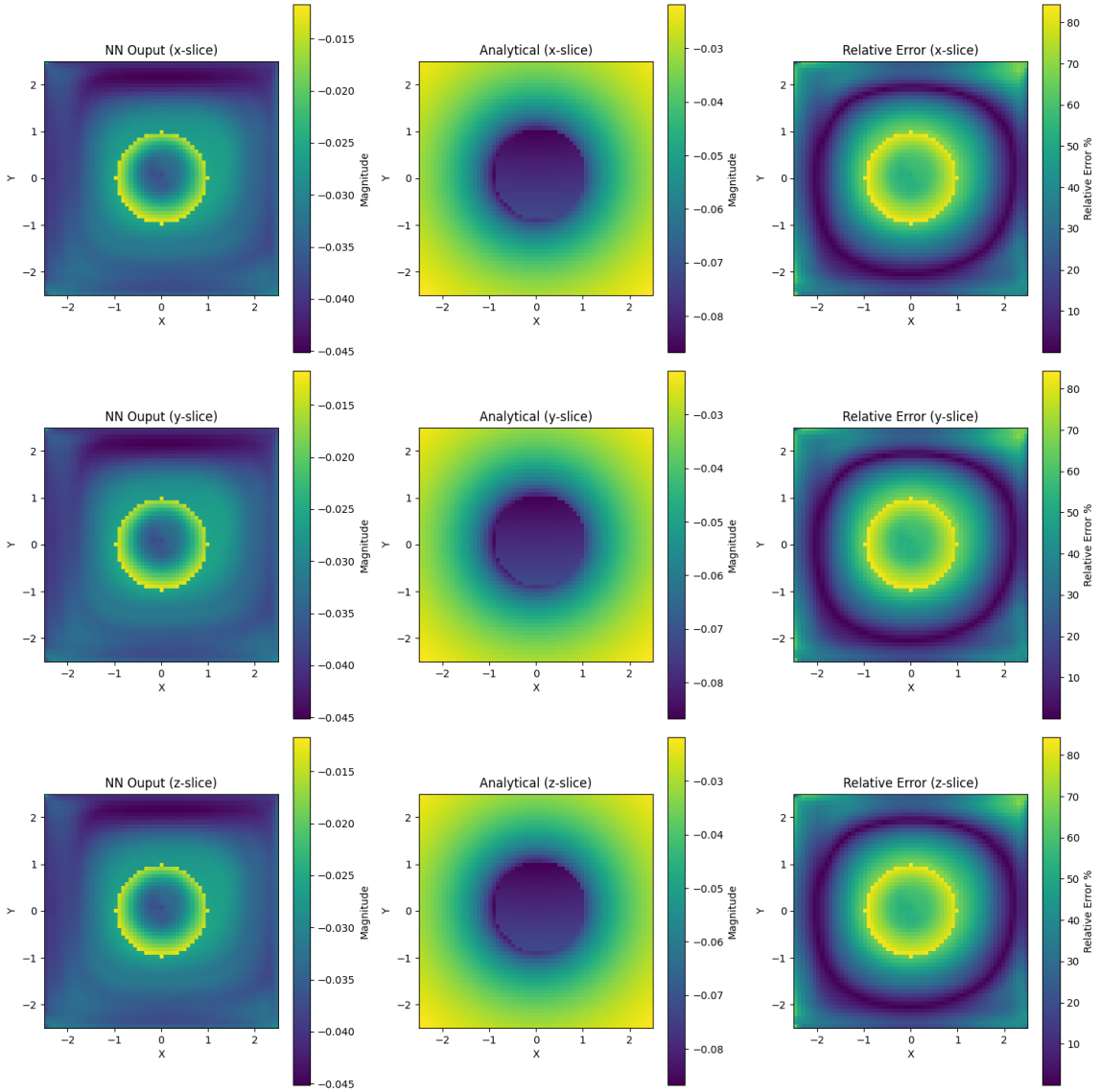


Figura 4.43: Distribución de potencial predicha tras el Entrenamiento

En este escenario, el modelo no logró representar adecuadamente el potencial: el error máximo alcanzó el 127.23 % y el error promedio el 60.24 %. La red tiende a anular el campo en todo el dominio, lo que revela que el término laplaciano domina la función de pérdida, arrastrando la solución hacia un mínimo trivial.

En síntesis, se demuestra que el modelo es capaz de reproducir con éxito el potencial regularizado en 3D cuando se dispone de una solución analítica. Sin embargo, el pobre desempeño del caso que se entrena utilizando la función de perdida *laplacian loss* pone de manifiesto la necesidad de refinar la combinación de ecuaciones o del mismo modelo para abordar problemas carentes de solución explícita.

Capítulo 5

Conclusiones

En el presente trabajo se implementó y aplicó un modelo PINNs con arquitectura convolucional tipo UNet para resolver la ecuación de Poisson. La implementación detallada permitió abordar casos bidimensionales y tridimensionales, incorporar condiciones de frontera no homogéneas e incluir dominios con interfaces que presentan parámetros físicos discontinuos. Las soluciones numéricas se contrastaron con las correspondientes soluciones analíticas, obteniéndose una concordancia robusta: los errores relativos promedio se situaron entre 3 % y 5 %. Aun cuando los errores máximos superaron el 90 %, se verificó que éstos se localizan exclusivamente en los nodos que contienen cargas puntuales o en puntos aislados de las fronteras no homogéneas; al excluirlos, los errores máximos descienden por debajo del 6 %.

Las pruebas demuestran que un modelo entrenado puede resolver múltiples configuraciones de la ecuación de Poisson siempre que se mantengan constantes las dimensiones del dominio y las condiciones de borde. Esto confirma la versatilidad y eficiencia del enfoque: una sola red neuronal es capaz de generalizar a un abanico de escenarios similares sin necesidad de reentrenamiento exhaustivo. Durante la implementación se evidencio que:

- La regularización adecuada de la red es esencial para la convergencia y la estabilidad numérica.
- La arquitectura UNet con campos receptivos intermedios ofrece el mejor equilibrio entre precisión y costo computacional.
- Se requieren al menos 71 nodos por dimensión para garantizar la convergencia; incrementar significativamente la densidad de nodos no mejora proporcionalmente la precisión, salvo en presencia de cargas puntuales, las cuales deben alinearse con nodos de la malla.
- La normalización de los datos de entrada y salida es crítica para evitar inestabilidades durante el entrenamiento.
- La selección adecuada de los pesos en la función de pérdida es crucial; se recomienda

ajustarlos según la física del problema y utilizar tasas de aprendizaje bajas para evitar oscilaciones sin penalizar excesivamente el tiempo de entrenamiento.

Entre los desafíos más relevantes destaca la selección de los pesos de cada término en la función de pérdida. Se recomienda afinarlos según la física del problema y adoptar tasas de aprendizaje bajas para evitar oscilaciones de la solución sin penalizar de forma notable el tiempo de entrenamiento.

En problemas 3D con interfaces regularizadas, el uso de una pérdida interna (*inside loss*) permite capturar discontinuidades en los parámetros físicos. Futuros trabajos podrían explorar el uso exclusivo de pérdidas de Laplaciano (*laplacian loss*), lo que eliminaría la necesidad de datos de referencia y simplificaría la implementación, facilitando el tratamiento de geometrías más complejas.

En síntesis, este estudio valida la viabilidad de las CNN-PINNs como herramienta eficiente para resolver la ecuación de Poisson y sienta una metodología replicable para otras ecuaciones diferenciales parciales. Pese a los desafíos identificados, los resultados son prometedores y abren el camino a investigaciones más amplias en física computacional basada en redes neuronales. El código elaborado está disponible en el repositorio de GitHub del autor [8], el cual permite al usuario resolver la ecuación de Poisson para diferentes escenarios.

Bibliografía

[1] I. Goodfellow, *Deep learning*. MIT press, 2016, vol. 196.

[2] S. Mallat, "Understanding deep convolutional networks," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150203, 2016.

[3] L. Cheng, E. A. Illarramendi, G. Bogopolsky, M. Bauerheim, and B. Cuenot, "Using neural networks to solve the 2d poisson equation for electric field computation in plasma fluid simulations," *arXiv preprint arXiv:2109.13076*, 2021.

[4] M. M. Taye, "Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions," *Computation*, vol. 11, no. 3, p. 52, 2023.

[5] T. Autonomous, "How rnns are used in computer vision applications," 2024, accessed: 2024-12-05. [Online]. Available: https://www.thinkautonomous.ai/blog/rnns-in-computer-vision/

[6] J.-Z. Peng, N. Aubry, Y.-B. Li, M. Mei, Z.-H. Chen, and W.-T. Wu, "Physics-informed graph convolutional neural network for modeling geometry-adaptive steady-state natural convection," *International Journal of Heat and Mass Transfer*, vol. 216, p. 124593, 2023.

[7] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[8] E. Hasbún, "Poisson solver cnn," 2024, accessed: 2024-12-25. [Online]. Available: https://github.com/EduardoHasbun/poissonSolverCNN

[9] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[10] J. Lederer, "Activation functions in artificial neural networks: A systematic overview," *arXiv preprint arXiv:2101.09957*, 2021.

[11] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[12] M. Hebiri, J. Lederer, and M. Taheri, "Layer sparsity in neural networks," *Journal of Statistical Planning and Inference*, vol. 234, p. 106195, 2025.

[13] A. L. Maas, A. Y. Hannun, A. Y. Ng *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1. Atlanta, GA, 2013, p. 3.

[14] Y. Liu and X. Yao, "Evolutionary design of artificial neural networks with different nodes," in *Proceedings of IEEE international conference on evolutionary computation.* IEEE, 1996, pp. 670–675.

[15] A. Mao, M. Mohri, and Y. Zhong, "Cross-entropy loss functions: Theoretical analysis and applications," in *International conference on Machine learning.* PMLR, 2023, pp. 23 803–23 828.

[16] P. J. Bickel and K. A. Doksum, *Mathematical statistics: basic ideas and selected topics, volumes I-II package.* Chapman and Hall/CRC, 2015.

[17] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *Ussr computational mathematics and mathematical physics*, vol. 4, no. 5, pp. 1–17, 1964.

[18] Y. Dauphin, H. De Vries, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization," *Advances in neural information processing systems*, vol. 28, 2015.

[19] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of machine learning research*, vol. 18, no. 153, pp. 1–43, 2018.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[21] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[22] A. Rehmer and A. Kroll, "On the vanishing and exploding gradient problem in gated recurrent units," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 1243–1248, 2020.

[23] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.

[24] S. Mallat, "Understanding deep convolutional networks," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150203, 2016.

[25] J. Koushik, "Understanding convolutional neural networks," *arXiv preprint arXiv:1605.09081*, 2016.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[27] P. Luo, X. Wang, W. Shao, and Z. Peng, "Towards understanding regularization in batch normalization," *arXiv preprint arXiv:1809.00846*, 2018.

[28] K. Simonyan, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[30] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2016.

[31] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 2015, pp. 234–241.

[32] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," *arXiv preprint arXiv:1511.05440*, 2015.

[33] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[34] J. C. Ye and W. K. Sung, "Understanding geometry of encoder-decoder CNNs," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 7064–7073. [Online]. Available: https://proceedings.mlr.press/v97/ye19a.html

[35] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, 2021.

[36] Y. Sun, Q. Sun, and K. Qin, "Physics-based deep learning for flow problems," *Energies*, vol. 14, no. 22, p. 7760, 2021.

[37] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (pinns) for fluid mechanics: A review," *Acta Mechanica Sinica*, vol. 37, no. 12, pp. 1727–1738, 2021.

[38] D. W. Abueidda, Q. Lu, and S. Koric, "Meshless physics-informed deep learning method

for three-dimensional solid mechanics," *International Journal for Numerical Methods in Engineering*, vol. 122, no. 23, pp. 7182–7201, 2021.

[39] Y. Shin, J. Darbon, and G. E. Karniadakis, "On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes," *arXiv preprint arXiv:2004.01806*, 2020.

[40] S. Wu, A. Zhu, Y. Tang, and B. Lu, "Convergence of physics-informed neural networks applied to linear second-order elliptic interface problems," *arXiv preprint arXiv:2203.03407*, 2022.

[41] T. De Ryck, A. D. Jagtap, and S. Mishra, "Error estimates for physics-informed neural networks approximating the navier–stokes equations," *IMA Journal of Numerical Analysis*, vol. 44, no. 1, pp. 83–119, 2024.

[42] S. Wang, X. Yu, and P. Perdikaris, "When and why pinns fail to train: A neural tangent kernel perspective," *Journal of Computational Physics*, vol. 449, p. 110768, 2022.

[43] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney, "Characterizing possible failure modes in physics-informed neural networks," *Advances in neural information processing systems*, vol. 34, pp. 26 548–26 560, 2021.

[44] S. Wang, S. Sankaran, H. Wang, and P. Perdikaris, "An expert's guide to training physics-informed neural networks," *arXiv preprint arXiv:2308.08468*, 2023.

[45] E. Kharazmi, Z. Zhang, and G. E. Karniadakis, "Variational physics-informed neural networks for solving partial differential equations," *arXiv preprint arXiv:1912.00873*, 2019.

[46] X. Xiao, Y. Zhou, H. Wang, and X. Yang, "A novel cnn-based poisson solver for fluid simulation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 3, pp. 1454–1465, 2020.

[47] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 365, p. 113028, 2020.

[48] A. D. Jagtap and G. E. Karniadakis, "Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations," *Communications in Computational Physics*, vol. 28, no. 5, 2020.

[49] Z. Hu, A. D. Jagtap, G. E. Karniadakis, and K. Kawaguchi, "Augmented physics-informed neural networks (apinns): A gating network-based soft domain decomposition methodology," *Engineering Applications of Artificial Intelligence*, vol. 126, p. 107183, 2023.

[50] V. Dwivedi, N. Parashar, and B. Srinivasan, "Distributed physics informed neural net-

work for data-efficient solution to partial differential equations," *arXiv preprint arXiv:1907.08967*, 2019.

[51] B. Yu *et al.*, "The deep ritz method: a deep learning-based numerical algorithm for solving variational problems," *Communications in Mathematics and Statistics*, vol. 6, no. 1, pp. 1–12, 2018.

[52] G. Lin, P. Hu, F. Chen, X. Chen, J. Chen, J. Wang, and Z. Shi, "Binet: Learning to solve partial differential equations with boundary integral networks," *arXiv preprint arXiv:2110.00352*, 2021.

[53] L. Lu, P. Jin, and G. E. Karniadakis, "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," *arXiv preprint arXiv:1910.03193*, 2019.

[54] S. Li and X. Feng, "Dynamic weight strategy of physics-informed neural networks for the 2d navier–stokes equations," *Entropy*, vol. 24, no. 9, p. 1254, 2022.

[55] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, "Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data," *Journal of Computational Physics*, vol. 394, pp. 56–81, 2019.

[56] Z. Fang, "A high-efficient hybrid physics-informed neural networks based on convolutional neural network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 10, pp. 5514–5526, 2021.

[57] H. Gao, L. Sun, and J.-X. Wang, "Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain," *Journal of Computational Physics*, vol. 428, p. 110079, 2021.

[58] J. D. Jackson, *Classical electrodynamics*. John Wiley & Sons, 2021.

[59] K. Fukami, K. Fukagata, and K. Taira, "Super-resolution reconstruction of turbulent flows with machine learning," *Journal of Fluid Mechanics*, vol. 870, pp. 106–120, 2019.

[60] A. Alguacil, M. Bauerheim, M. C. Jacob, and S. Moreau, "Predicting the propagation of acoustic waves using deep convolutional neural networks," *Journal of Sound and Vibration*, vol. 512, p. 116285, 2021.

[61] N. Thuerey, K. Weißenow, L. Prantl, and X. Hu, "Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows," *AIAA Journal*, vol. 58, no. 1, pp. 25–36, 2020.

[62] C. J. Lapeyre, A. Misdariis, N. Cazard, D. Veynante, and T. Poinsot, "Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates," *Combustion and Flame*, vol. 203, pp. 255–264, 2019.

[63] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[64] F. Chollet, *Deep learning with Python*. Simon and Schuster, 2021.

[65] A. G. Özbay, A. Hamzehloo, S. Laizet, P. Tzirakis, G. Rizos, and B. Schuller, "Poisson cnn: Convolutional neural networks for the solution of the poisson equation on a cartesian mesh," *Data-Centric Engineering*, vol. 2, p. e6, 2021.

[66] T. LIN, "Numerical interfaces in finite difference methods for hyperbolic equations with discontinuous coefficients," *Journal of Computational Acoustics*, vol. 1, no. 02, pp. 151–184, 1993.

[67] C. R. Rao, X. Shi, and Y. Wu, "Approximation of the expected value of the harmonic mean and some applications," *Proceedings of the National Academy of Sciences*, vol. 111, no. 44, pp. 15 681–15 686, 2014.

[68] G. Van Rossum and F. L. Drake, *Introduction to python 3: python documentation manual part 1*. CreateSpace, 2009.

[69] E. Stevens, L. Antiga, and T. Viehmann, *Deep learning with PyTorch*. Manning Publications, 2020.

[70] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, "Scipy 1.0: fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[71] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 03, pp. 90–95, 2007.

[72] T. E. Oliphant *et al.*, *Guide to numpy*. Trelgol Publishing USA, 2006, vol. 1.

Anexos

A. Derivación Ecuación de la solución de Poisson con teorema de Green

Partiendo de los teoremas de Green 2.45 y de la divergencia 2.46 se debe considera una función ψ particular, la cual es:

$$\frac{1}{R} = \frac{1}{|x - x'|} \tag{A.1}$$

donde x es el punto de referencia y x' es la variable de integración. Al tomar $\phi = -\rho/\epsilon_0$, y utilizando la ecuación 2.46, se obtiene:

$$\nabla^2 \left(\frac{1}{R}\right) = -4\pi\delta(x - x') \tag{A.2}$$

De esta manera, la ecuación 2.45 puede reescribirse como:

$$\int_V \left[-4\pi\phi(x')\delta(x - x') + \frac{1}{\epsilon_0 R}\rho(x')\right] d^3x' = \oint_S \left[\phi \frac{\partial}{\partial n'}\left(\frac{1}{R}\right) - \frac{1}{R}\frac{\partial \phi}{\partial n'}\right] da' \tag{A.3}$$

Si el punto x se encuentra dentro del volumen V, se obtiene:

$$\phi(x) = \frac{1}{4\pi\epsilon_0}\int_V \frac{\rho(x')}{R} d^3x' + \frac{1}{4\pi}\oint_S \left[\frac{1}{R}\frac{\partial \phi}{\partial n'} - \phi\frac{\partial}{\partial n'}\left(\frac{1}{R}\right)\right] da' \tag{A.4}$$

La función de Green en la ecuación A.1 satisface el teorema de la divergencia 2.46. En general, se puede escribir:

$$\nabla'^2 G(x, x') = -4\pi\delta(x - x') \tag{A.5}$$

donde:

$$G(x, x') = \frac{1}{|x - x'|} + F(x, x') \tag{A.6}$$

Aquí, F es una función que satisface la ecuación de Laplace dentro del volumen V:

$$\nabla'^2 F(x, x') = 0 \tag{A.7}$$

Usando el teorema de Green 2.45, tomando $\phi = \Phi$ y $\psi = G(x, x')$, junto con la propiedad de la ecuación A.5, se puede generalizar la solución de A.4 como:

$$\Phi(x) = \frac{1}{4\pi\epsilon_0}\int_V \rho(x')G(x,x')d^3x' + \frac{1}{4\pi}\oint_S \left[G(x,x')\frac{\partial\Phi}{\partial n'} - \Phi(x')\frac{\partial G(x,x')}{\partial n'}\right]da' \tag{A.8}$$

A. Derivación Laplaciano Coordenadas Polares

Partiendo de la ecuación de Laplace en coordenadas cartesianas y la transformación a coordenadas polares:

$$\begin{aligned} \nabla^2 u &= \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \\ x &= r\cos(\theta) \\ y &= r\sin(\theta) \end{aligned} \tag{A.9}$$

Se aplica la regla de la cadena para $\partial u/\partial x$:

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial r}\frac{\partial r}{\partial x} + \frac{\partial u}{\partial \theta}\frac{\partial \theta}{\partial x} \tag{A.10}$$

Sabiendo que $r = \sqrt{x^2 = y^2}$ y que $\theta = \tan^{-1}(y/x)$:

$$\begin{aligned} \frac{\partial r}{\partial x} &= \frac{1}{2\sqrt{x^2+y^2}}2x = \frac{x}{r} = \cos(\theta) \\ \frac{\partial \theta}{\partial x} &= \frac{1}{1+(\frac{y}{x})^2}\frac{-y}{x^2} = \frac{-y}{x^2+y^2} = \frac{-y}{r^2} = \frac{-\sin\theta}{r} \end{aligned} \tag{A.11}$$

Sustituyendo en la ecuación (A.10):

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial r}\cos(\theta) - \frac{\partial u}{\partial \theta}\frac{\sin(\theta)}{r} \tag{A.12}$$

Aplicando la misma regla de la cadena para $\partial u/\partial y$:

$$\begin{aligned}\frac{\partial r}{\partial y} &= \frac{1}{2\sqrt{x^2+y^2}}2y = \frac{y}{r} = \sin(\theta)\\ \frac{\partial \theta}{\partial y} &= \frac{1}{1+(\frac{y}{x})^2}\frac{1}{x} = \frac{x}{x^2+y^2} = \frac{x}{r^2} = \frac{\cos\theta}{r}\\ \frac{\partial u}{\partial y} &= \frac{\partial u}{\partial r}\sin(\theta) + \frac{\partial u}{\partial \theta}\frac{\cos(\theta)}{r}\end{aligned} \tag{A.13}$$

Aplicando la regla de la cadena para $\partial^2 u/\partial x^2$:

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right) = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial r}\cos(\theta) - \frac{\partial u}{\partial \theta}\frac{\sin(\theta)}{r}\right) \tag{A.14}$$

Aplicando nuevamente regla de la cadena se obtiene:

$$\begin{aligned}\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial r}\cos(\theta)\right) &= \left(\frac{\partial}{\partial x}\frac{\partial u}{\partial r}\cos(\theta) + \frac{\partial u}{\partial r}\frac{\partial}{\partial x}\cos(\theta)\right) = \frac{\partial^2 u}{\partial r^2}\cos^2(\theta) + \frac{\partial u}{\partial r}\frac{\sin^2(\theta)}{r}\\ \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial \theta}\frac{\sin(\theta)}{r}\right) &= -\left(\frac{\partial^2 u}{\partial \theta^2}\frac{\partial \theta}{\partial x}\frac{\sin(\theta)}{r}\right) - \frac{1}{r}\frac{\partial u}{\partial \theta}\frac{\partial}{\partial x}\sin(\theta) - \frac{\partial u}{\partial \theta}\sin(\theta)\frac{\partial}{\partial x}\frac{1}{r} =\\ &\quad \frac{\partial^2 u}{\partial \theta^2}\frac{\sin^2(\theta)}{r^2} + \frac{\partial u}{\partial \theta}\frac{2\sin(\theta)\cos(\theta)}{r^2}\end{aligned} \tag{A.15}$$

Por lo tanto, la derivada de segundo en x es:

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial r^2}\cos^2(\theta) + \frac{\partial u}{\partial r}\frac{\sin^2(\theta)}{r} + \frac{\partial^2 u}{\partial \theta^2}\frac{\sin^2(\theta)}{r^2} + \frac{\partial u}{\partial \theta}\frac{2\sin(\theta)\cos(\theta)}{r^2} \tag{A.16}$$

De forma similar, se obtiene la derivada de segundo orden en y:

$$\frac{\partial^2 u}{\partial y^2} = \frac{\partial^2 u}{\partial r^2}\sin^2(\theta) + \frac{\partial u}{\partial r}\frac{\cos^2(\theta)}{r} + \frac{\partial^2 u}{\partial \theta^2}\frac{\cos^2(\theta)}{r^2} - \frac{\partial u}{\partial \theta}\frac{2\sin(\theta)\cos(\theta)}{r^2} \tag{A.17}$$

Juntando ambos resultados y simplificando, se obtiene:

$$\nabla^2 u = \frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} \tag{A.18}$$

B. Derivación Ecuación de Poisson carga puntual centrada en 2D con Interfaz

Partiendo de la configuración del problema que es:

$$\begin{aligned} \nabla^2\phi_1 &= -\frac{q}{\epsilon_1}\delta(r) \quad \text{en} \quad r < R \\ \nabla^2\phi_2 &= 0 \quad \text{en} \quad r > R \end{aligned} \tag{B.19}$$

En donde, la solución para el caso axisimétrico es:

$$\begin{aligned} \phi_1 &= \frac{-q}{2\pi\epsilon_1}\ln(r) + C \quad \text{en} \quad r < R \\ \phi_2 &= \frac{A}{r} + B \quad \text{en} \quad r > R \end{aligned} \tag{B.20}$$

Utilizando condiciones de contorno en la interfaz $r = R$:

- Continuidad en la derivada:

$$\begin{aligned} \epsilon_1\frac{\partial\phi_1}{\partial r}|_{r=R} &= \epsilon_2\frac{\partial\phi_2}{\partial r}|_{r=R} \\ \epsilon_1\frac{-q}{2\pi\epsilon_1 R} &= \epsilon_2\frac{A}{R} \\ A &= -\frac{q}{2\pi\epsilon_2} \end{aligned} \tag{B.21}$$

- Considerando que cuando $r \to \infty$, la region interna se puede considerar como una carga puntual, por lo cual el termino B debe ser cero, se obtiene:

$$\phi_2 = A\ln(r) \tag{B.22}$$

- Continuidad en el potencial:

$$\begin{aligned} \phi_1|_{r=R} &= \phi_2|_{r=R} \\ A\ln(R) &= \frac{-q}{2\pi\epsilon_1}\ln(R) + C \\ C &= \frac{q}{2\pi\epsilon_1}\ln(R) - \frac{q}{2\pi\epsilon_2}\ln(R) \end{aligned} \tag{B.23}$$

Con lo cual, la solución general es:

$$\begin{aligned}\phi_1 &= \frac{q}{2\pi\epsilon_1}\ln\left(\frac{R}{r}\right) - \frac{q}{2\pi\epsilon_2}\ln(R) \quad \text{en} \quad r < R \\ \phi_2 &= \frac{-q}{2\pi\epsilon_2}\ln(r) \quad \text{en} \quad r > R\end{aligned} \tag{B.24}$$

C. Resultados de Casos 2D Funciones de Perdida

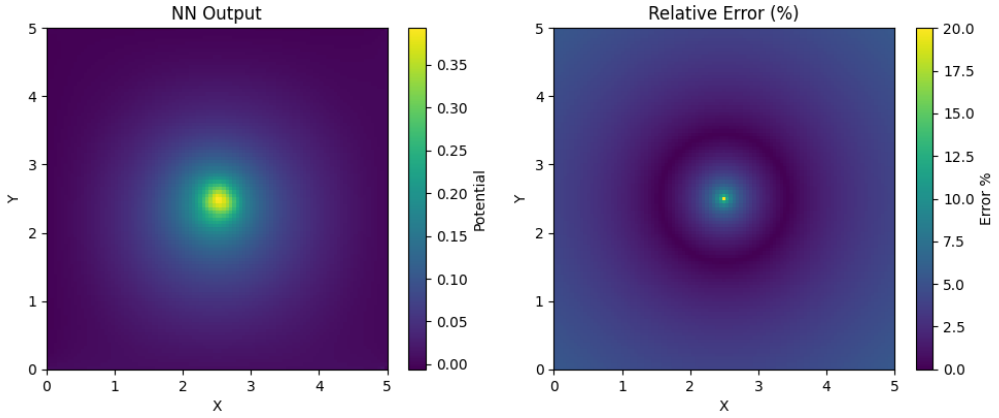


Figura C.1: Resultados del Caso 1 Funciones de Perdida

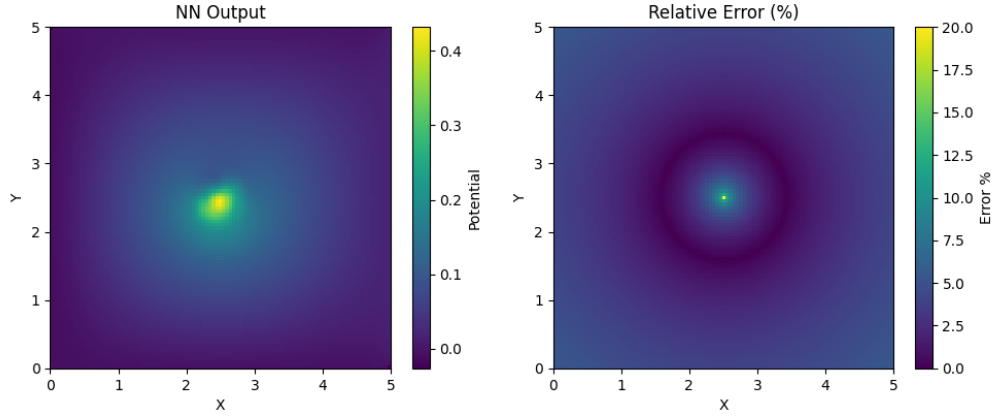


Figura C.2: Resultados del Caso 2 Funciones de Perdida

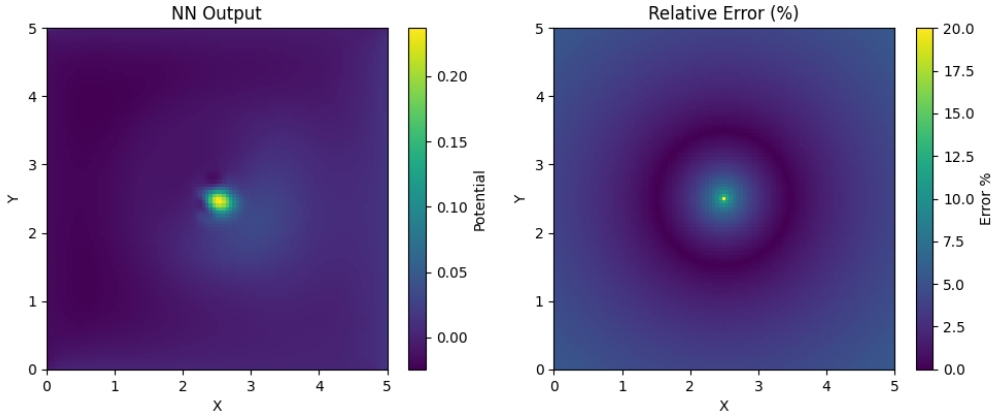


Figura C.3: Resultados del Caso 3 Funciones de Perdida

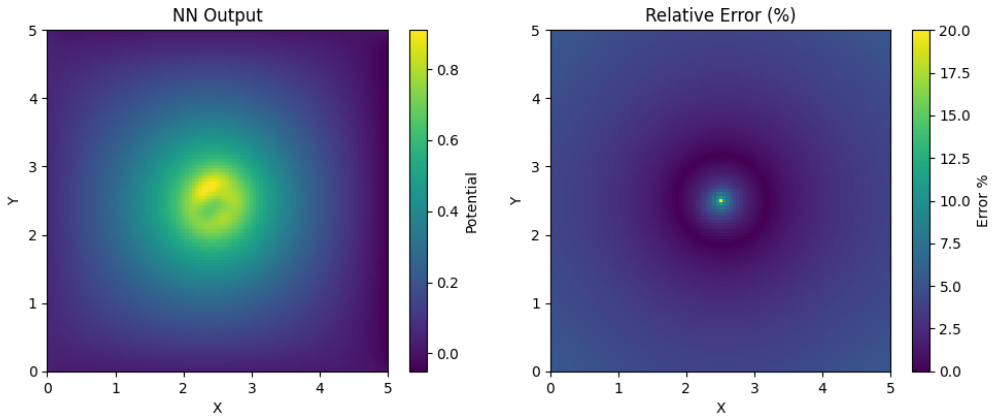


Figura C.4: Resultados del Caso 4 Funciones de Perdida

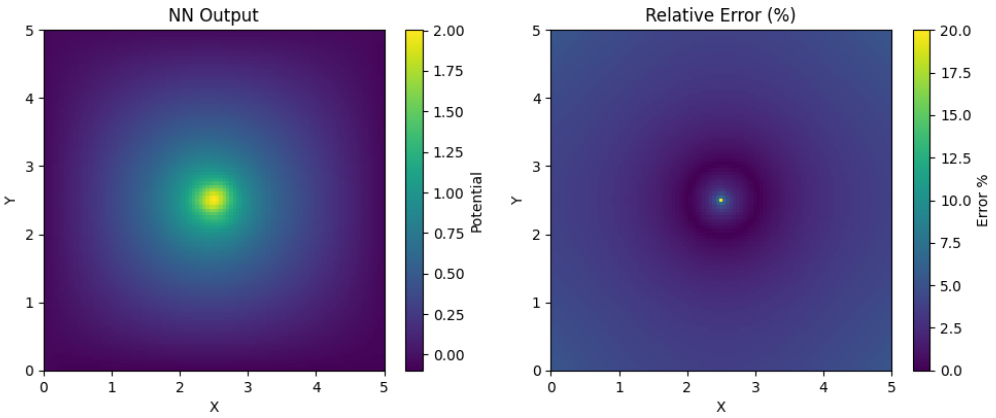


Figura C.5: Resultados del Caso 5 Funciones de Perdida

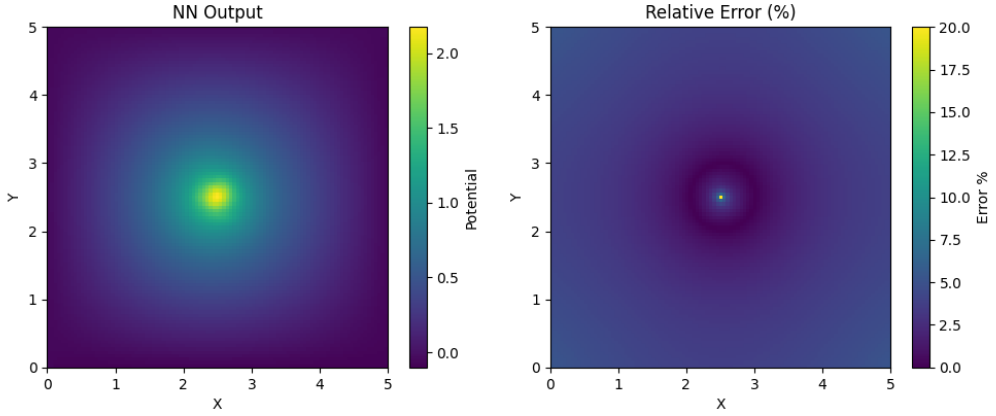


Figura C.6: Resultados del Caso 6 Funciones de Perdida

D. Resultados de Casos 2D Variación de Número de nodos

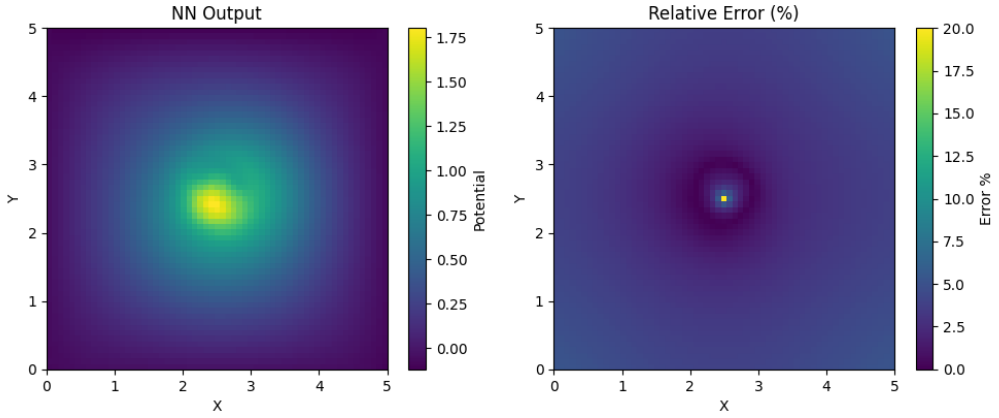


Figura D.1: Resultados del Caso 1 Variación de Número de Nodos

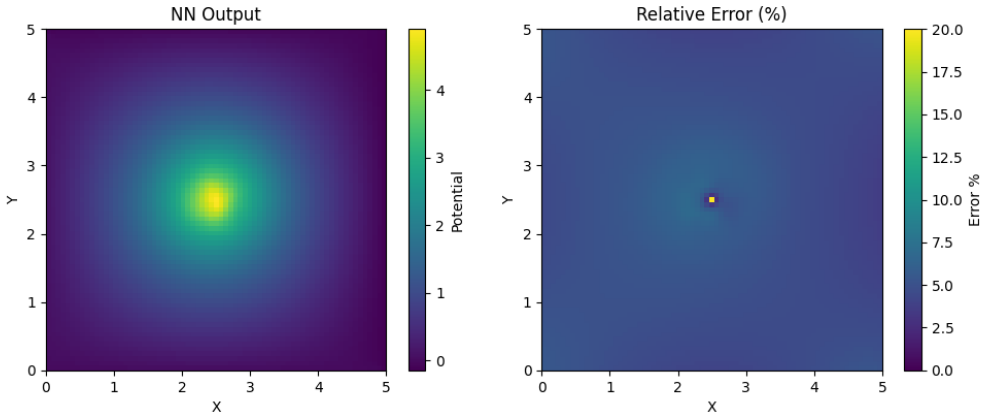


Figura D.2: Resultados del Caso 2 Variación de Número de Nodos

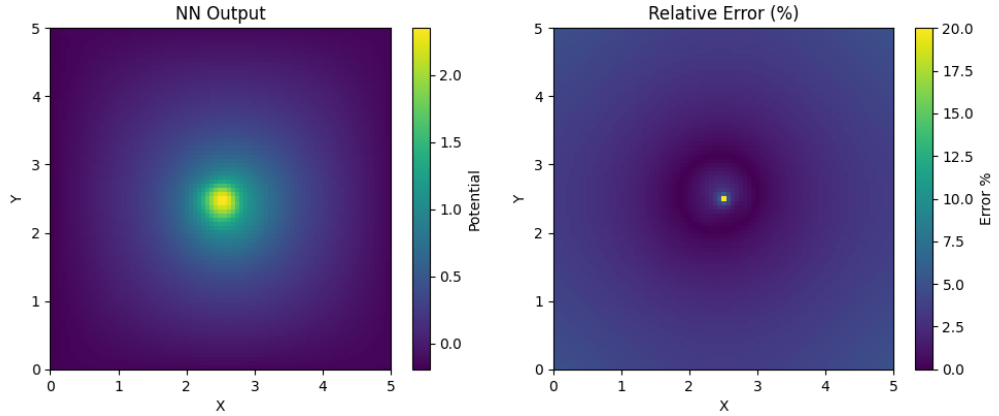


Figura D.3: Resultados del Caso 3 Variación de Número de Nodos

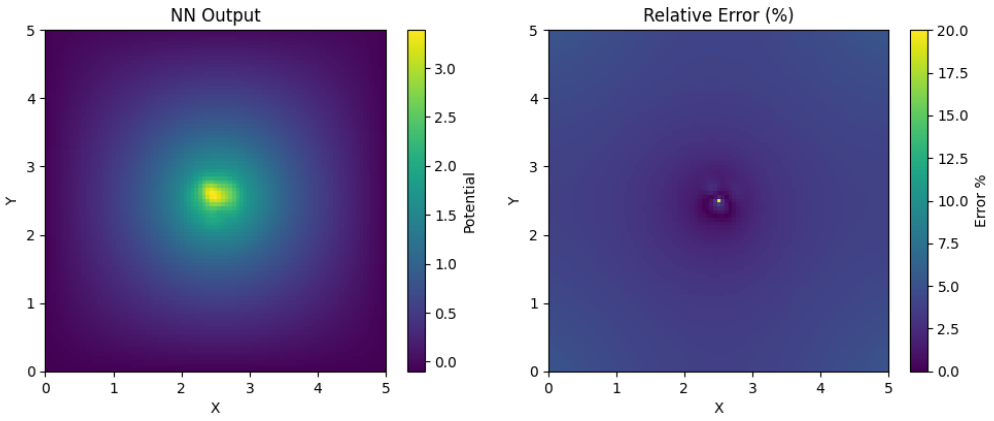


Figura D.4: Resultados del Caso 4 Variación de Número de Nodos

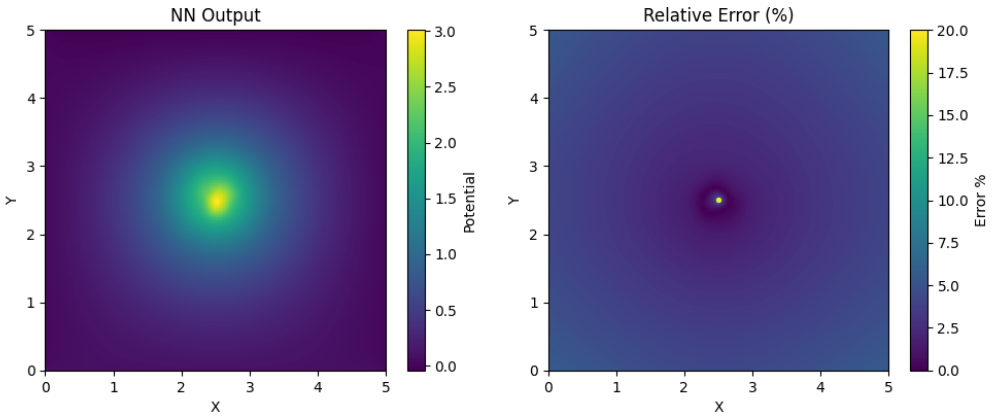


Figura D.5: Resultados del Caso 5 Variación de Número de Nodos

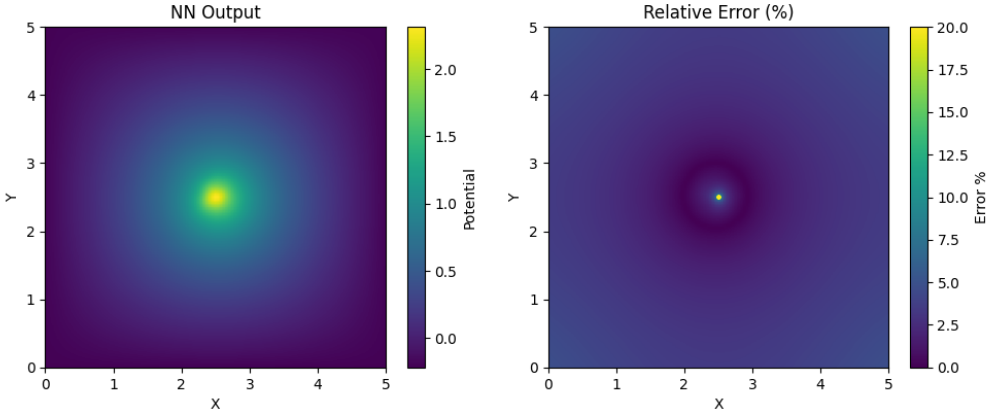


Figura D.6: Resultados del Caso 6 Variación de Número de Nodos

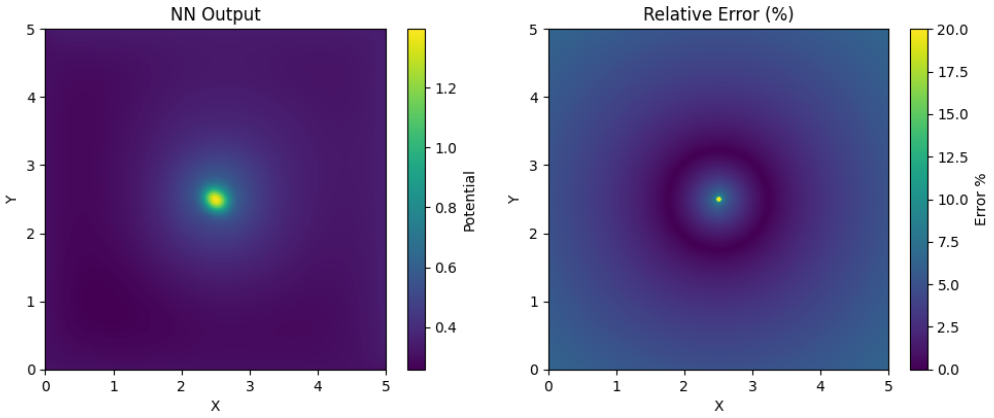


Figura D.7: Resultados del Caso 7 Variación de Número de Nodos

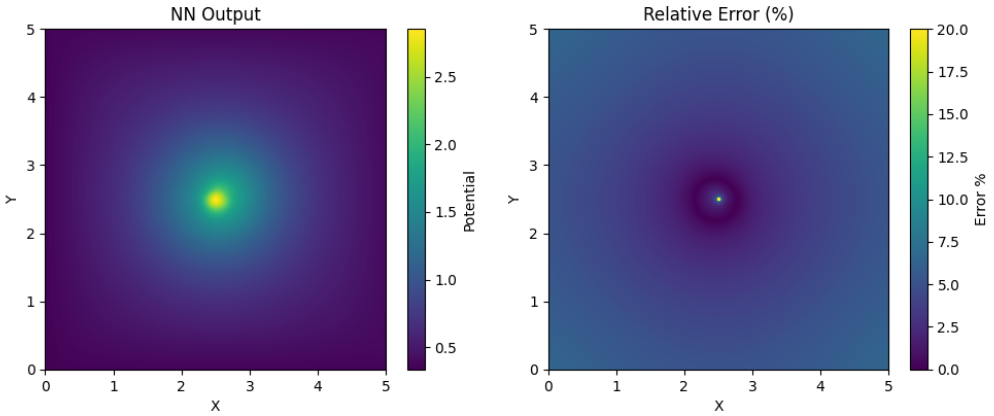


Figura D.8: Resultados del Caso 8 Variación de Número de Nodos

E. Resultados de Casos 2D Cargas puntuales Aleatorias

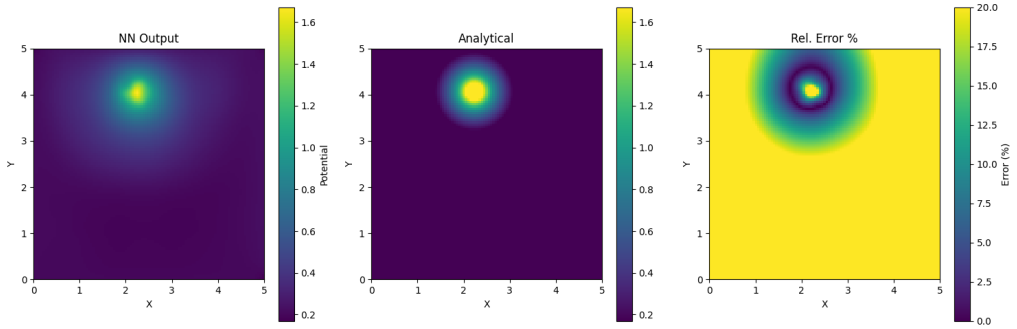


Figura E.1: Resultados del Caso 1 Carga Puntual Aleatoria

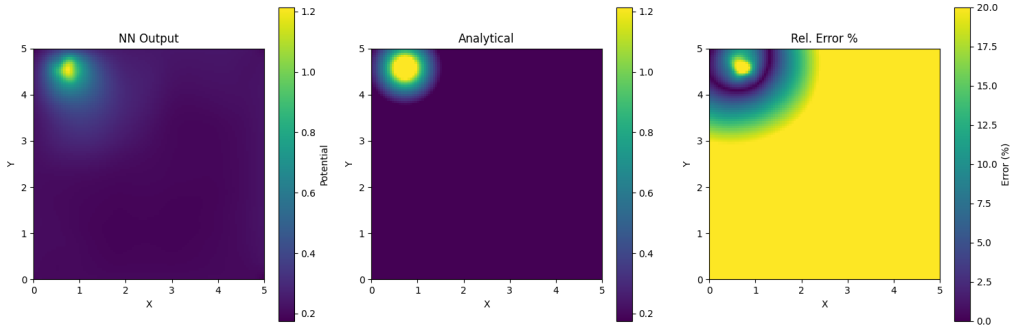


Figura E.2: Resultados del Caso 2 Carga Puntual Aleatoria

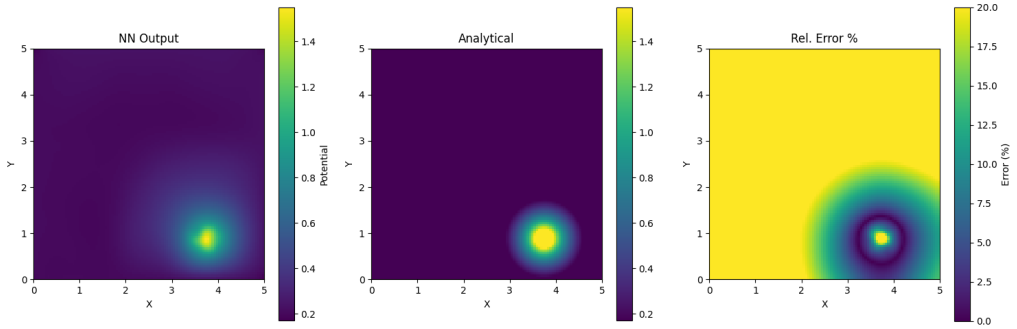


Figura E.3: Resultados del Caso 3 Carga Puntual Aleatoria

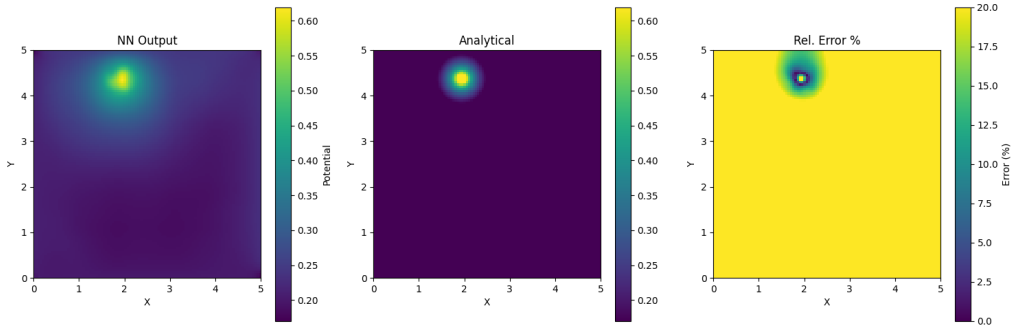


Figura E.4: Resultados del Caso 4 Carga Puntual Aleatoria

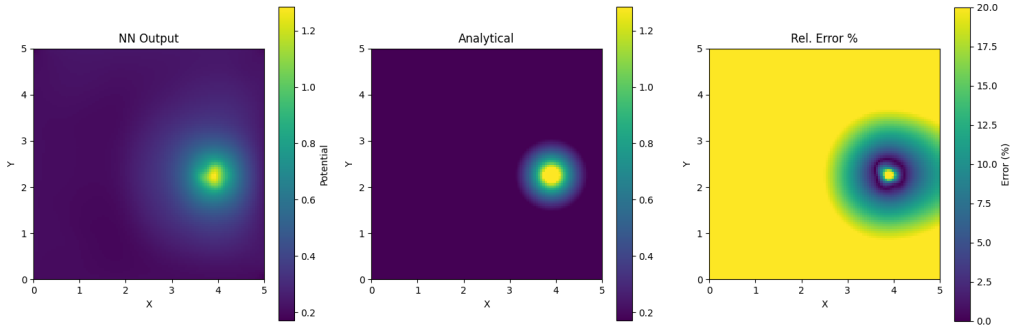


Figura E.5: Resultados del Caso 5 Carga Puntual Aleatoria

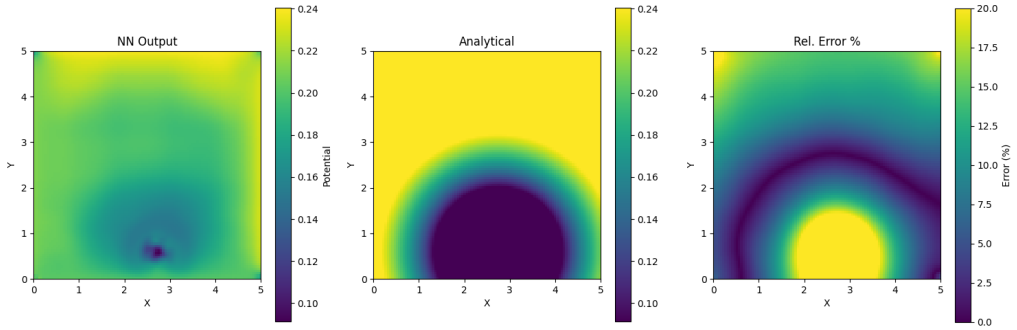


Figura E.6: Resultados del Caso 6 Carga Puntual Aleatoria

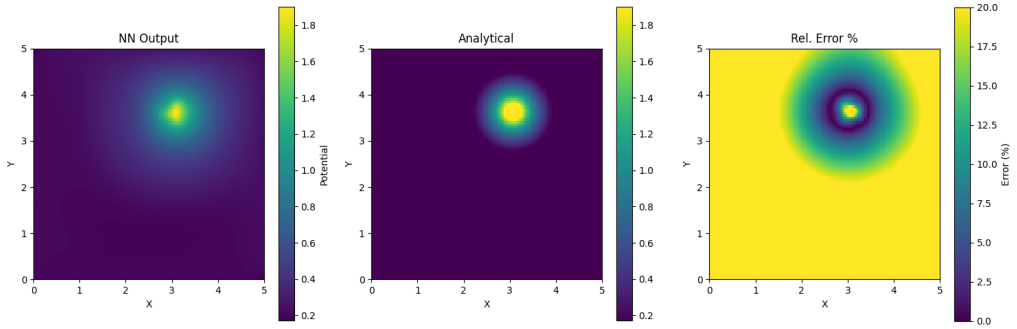


Figura E.7: Resultados del Caso 7 Carga Puntual Aleatoria

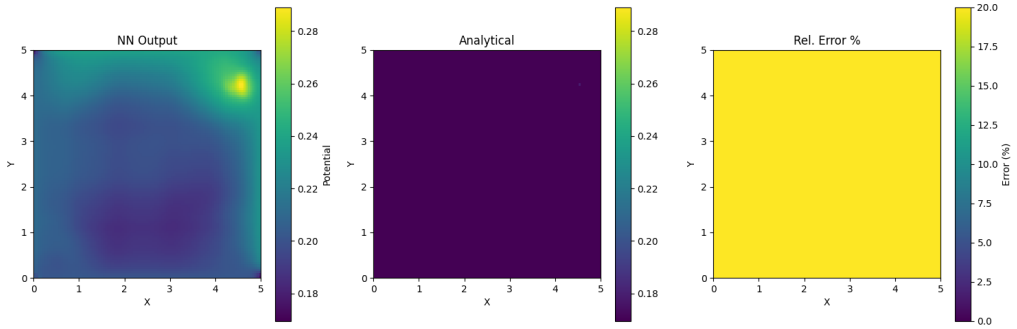


Figura E.8: Resultados del Caso 8 Carga Puntual Aleatoria

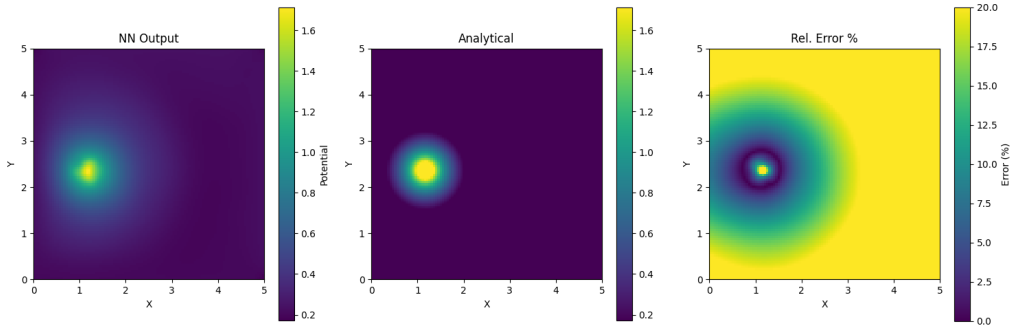


Figura E.9: Resultados del Caso 9 Carga Puntual Aleatoria

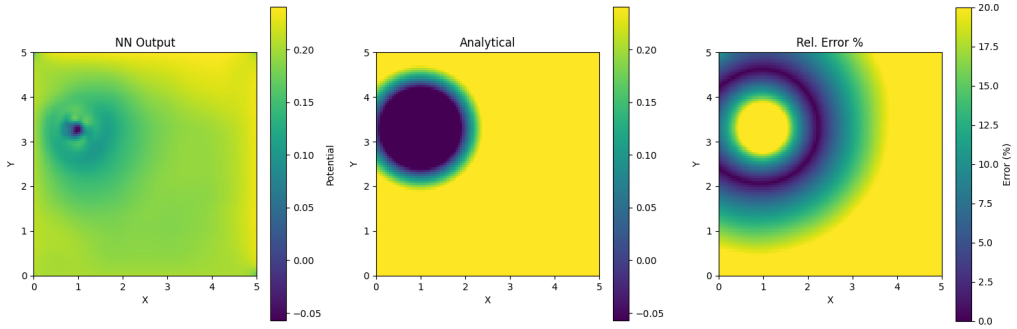


Figura E.10: Resultados del Caso 10 Carga Puntual Aleatoria

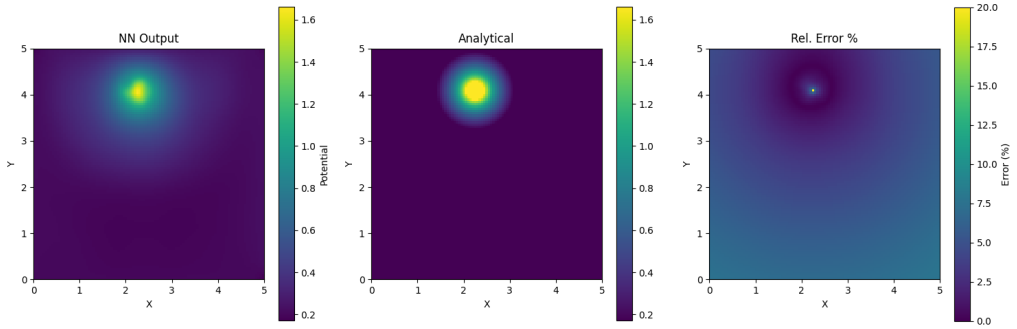


Figura E.11: Resultados del Caso 11 Carga Puntual Aleatoria

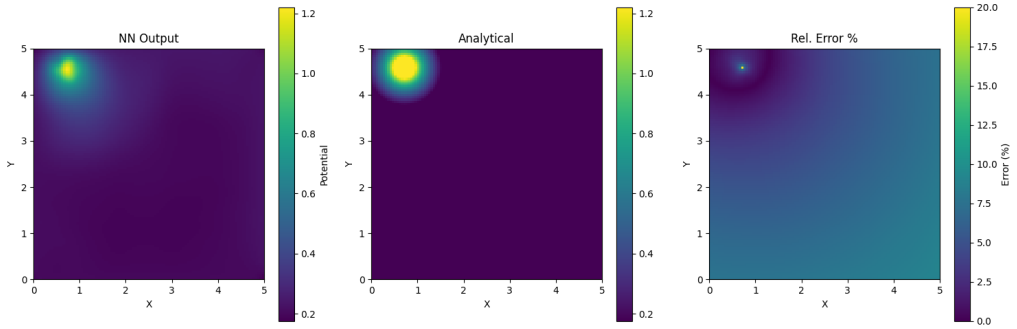


Figura E.12: Resultados del Caso 12 Carga Puntual Aleatoria

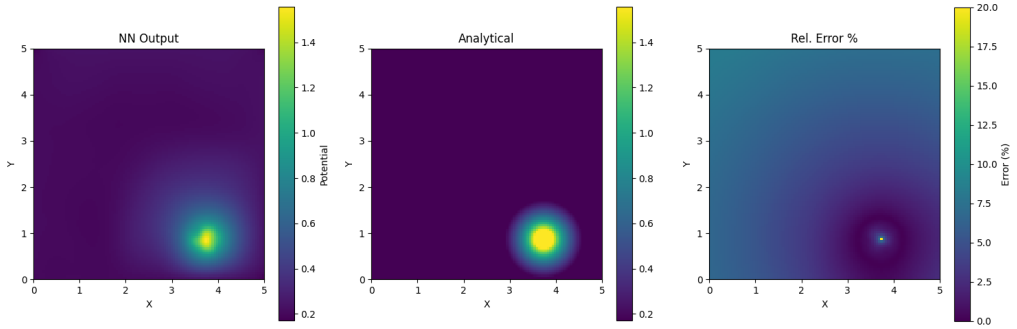


Figura E.13: Resultados del Caso 13 Carga Puntual Aleatoria

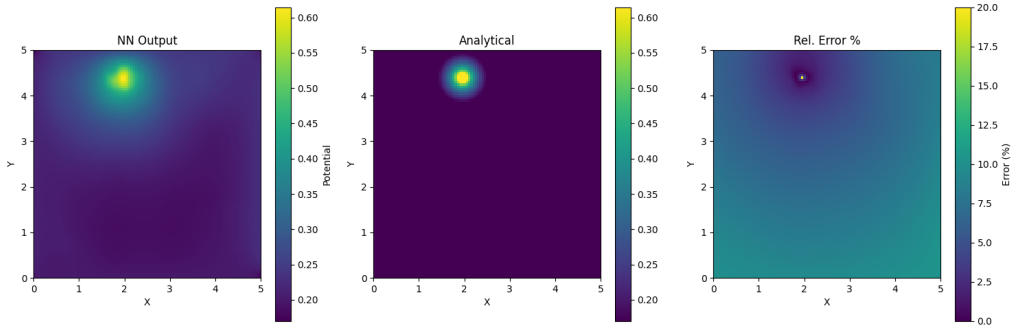


Figura E.14: Resultados del Caso 14 Carga Puntual Aleatoria

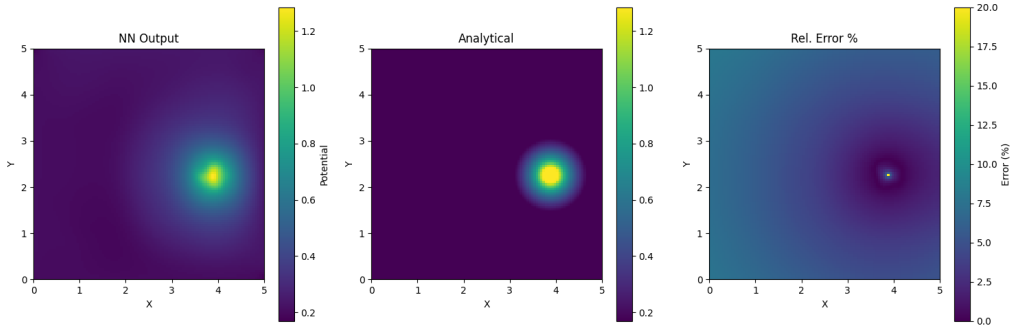


Figura E.15: Resultados del Caso 15 Carga Puntual Aleatoria

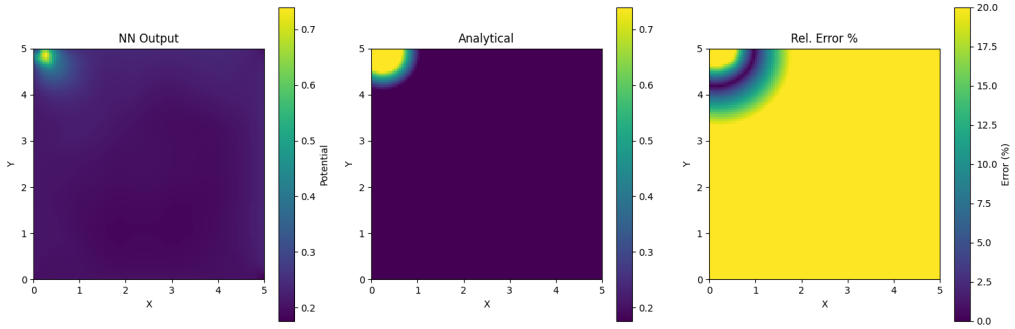


Figura E.16: Resultados del Caso 16 Carga Puntual Aleatoria

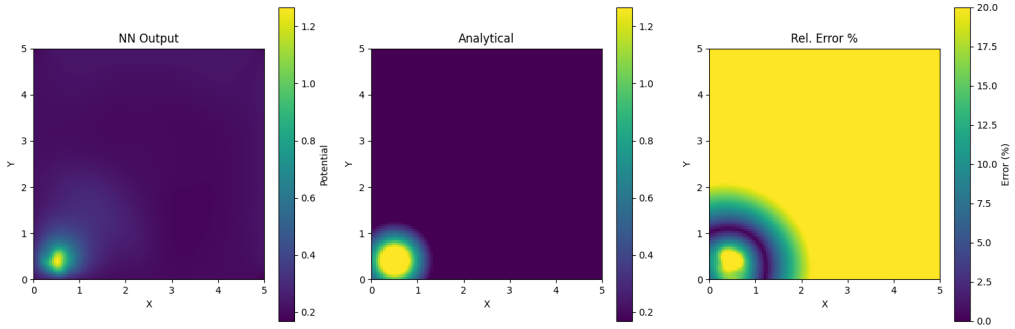


Figura E.17: Resultados del Caso 17 Carga Puntual Aleatoria

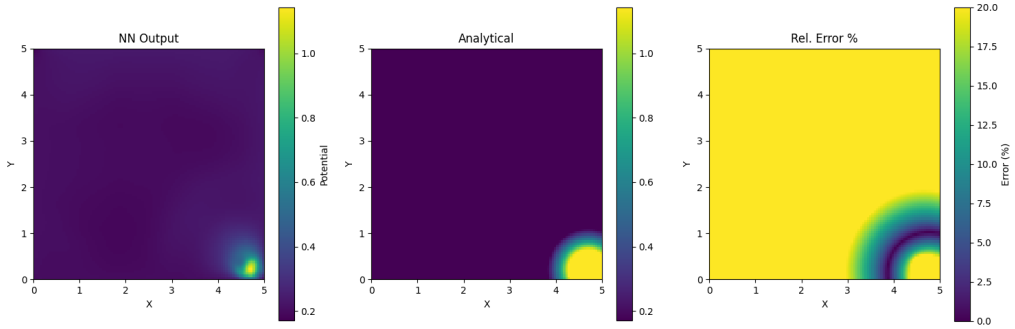


Figura E.18: Resultados del Caso 18 Carga Puntual Aleatoria

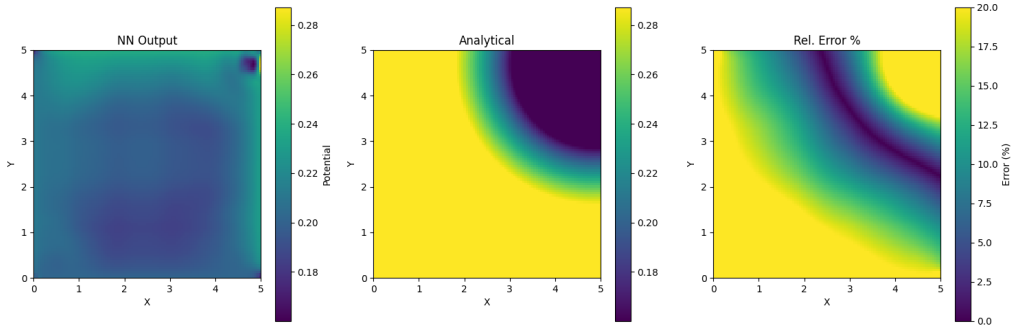


Figura E.19: Resultados del Caso 19 Carga Puntual Aleatoria

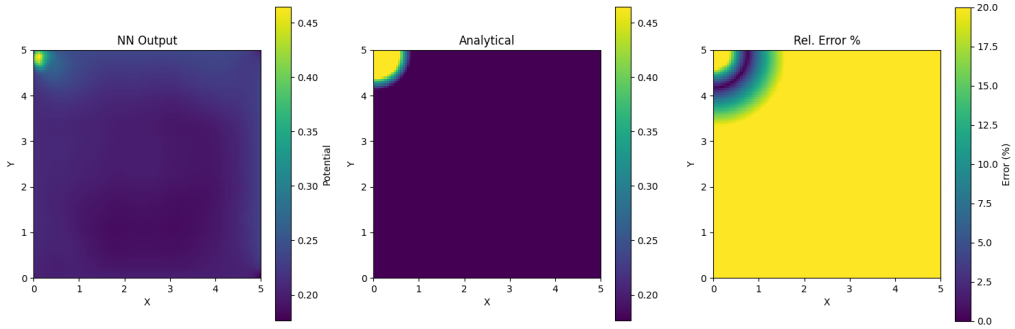


Figura E.20: Resultados del Caso 20 Carga Puntual Aleatoria

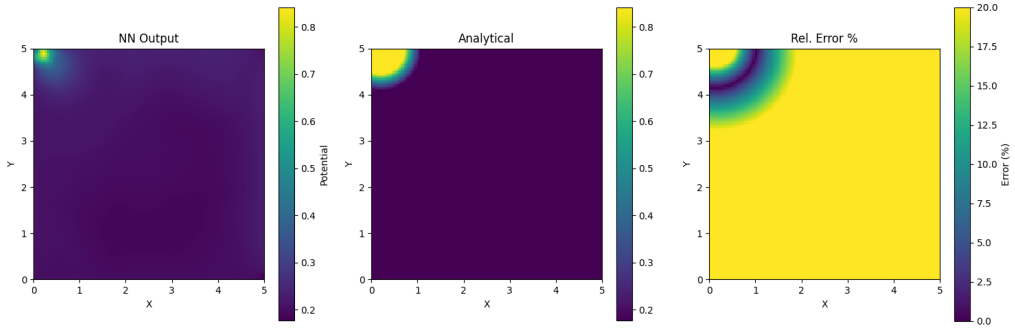


Figura E.21: Resultados del Caso 21 Carga Puntual Aleatoria

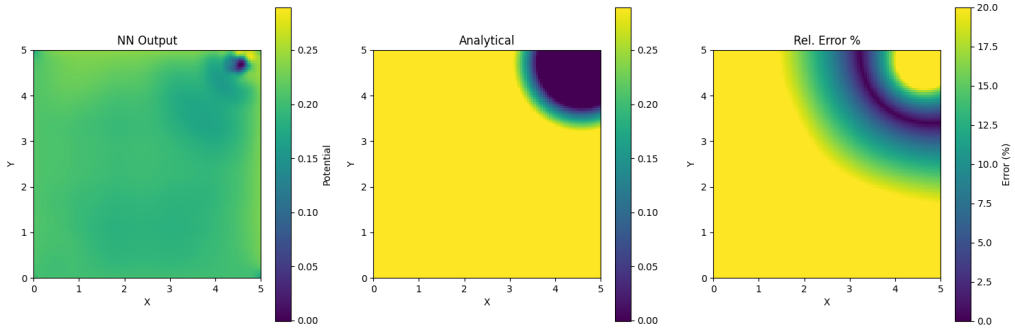


Figura E.22: Resultados del Caso 22 Carga Puntual Aleatoria

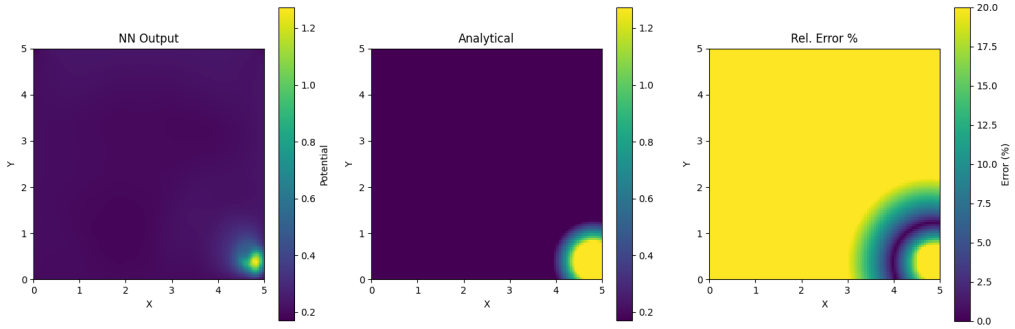


Figura E.23: Resultados del Caso 23 Carga Puntual Aleatoria

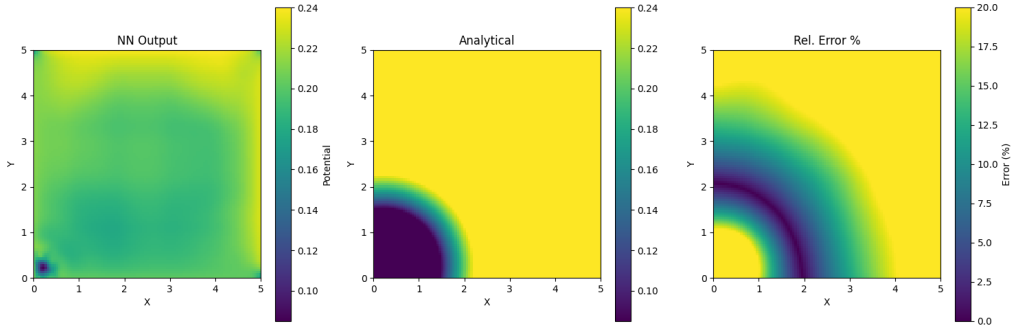


Figura E.24: Resultados del Caso 24 Carga Puntual Aleatoria

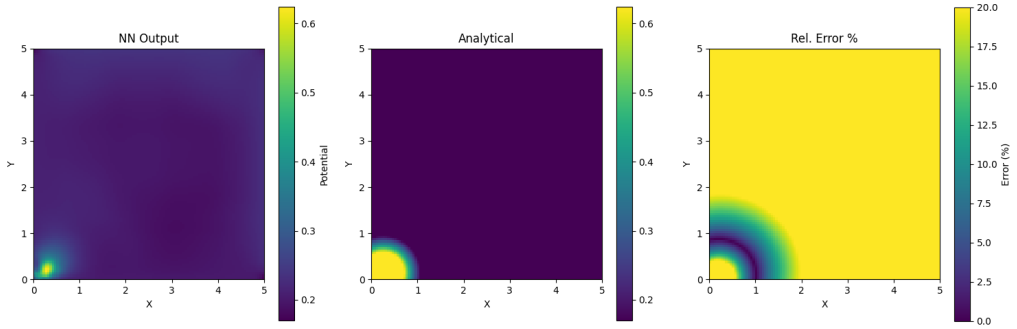


Figura E.25: Resultados del Caso 25 Carga Puntual Aleatoria

F. Resultados de Casos 2D Cargas Puntuales Distribuidas Aleatorias

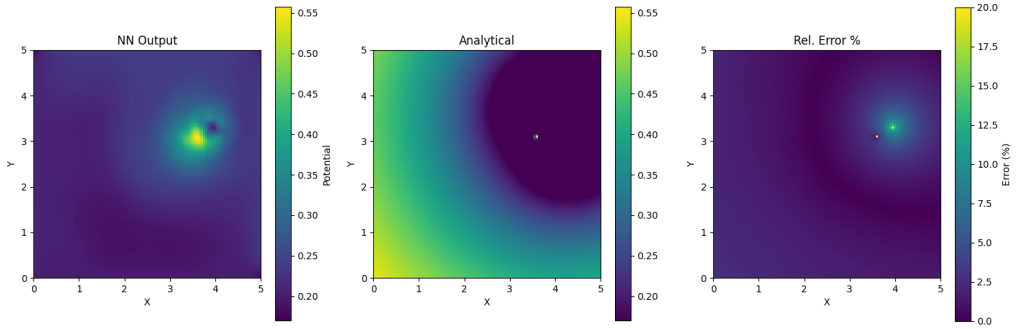


Figura F.1: Resultados del Caso 1 para cargas aleatorias

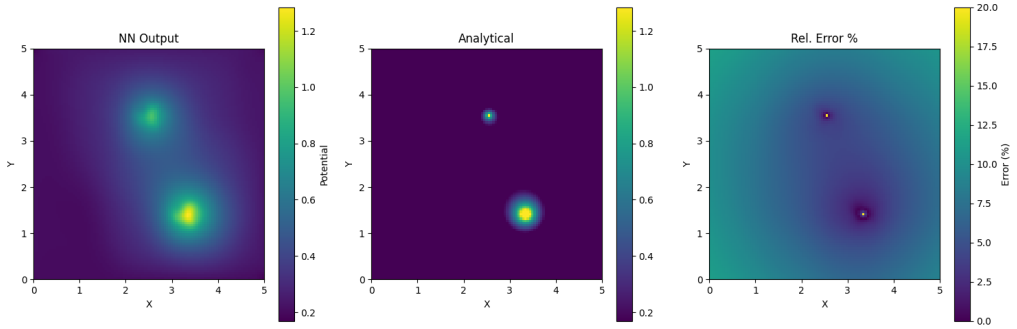


Figura F.2: Resultados del Caso 2 para cargas aleatorias

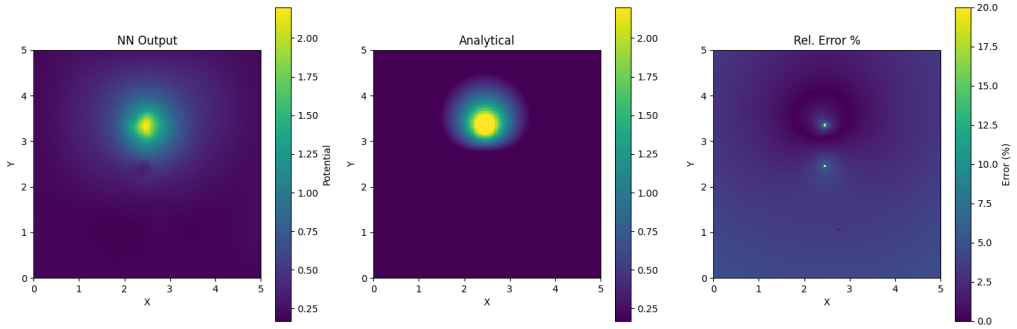


Figura F.3: Resultados del Caso 3 para cargas aleatorias

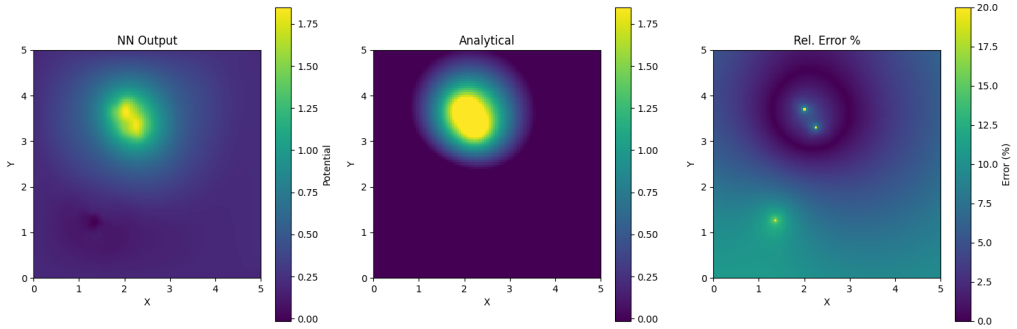


Figura F.4: Resultados del Caso 4 para cargas aleatorias

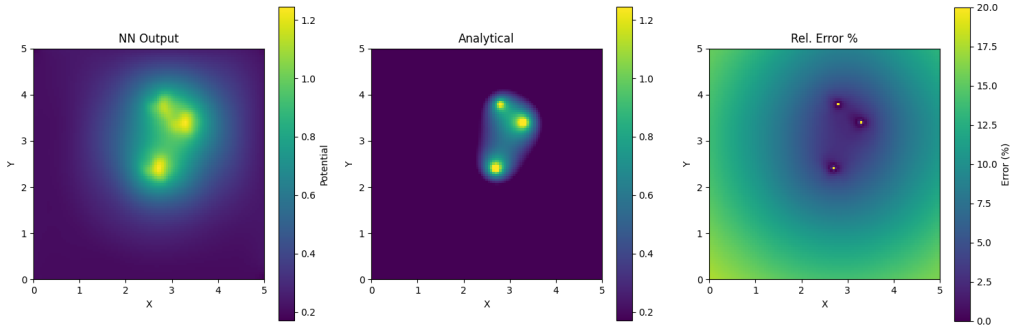


Figura F.5: Resultados del Caso 5 para cargas aleatorias

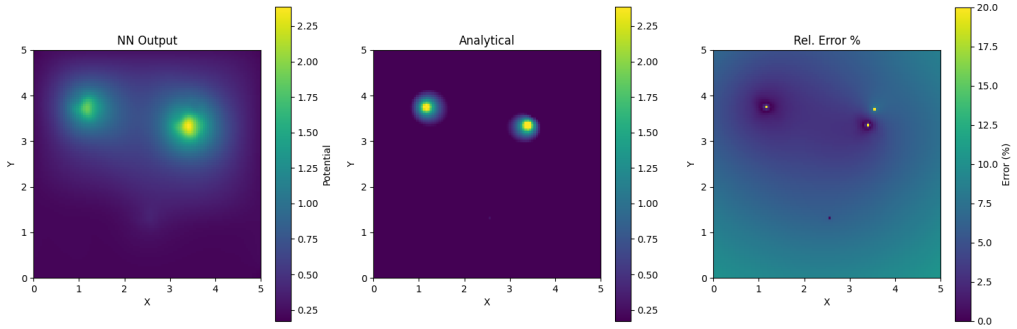


Figura F.6: Resultados del Caso 6 para cargas aleatorias

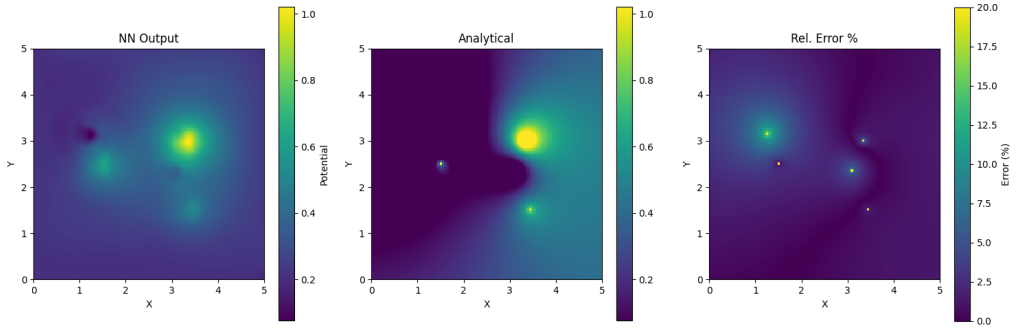


Figura F.7: Resultados del Caso 7 para cargas aleatorias

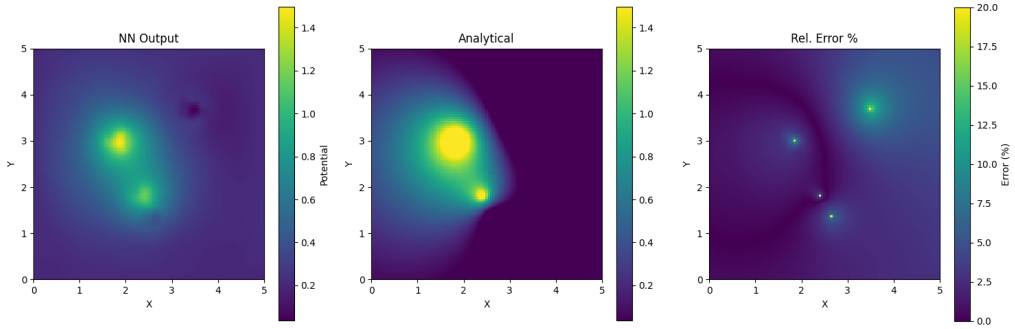


Figura F.8: Resultados del Caso 8 para cargas aleatorias

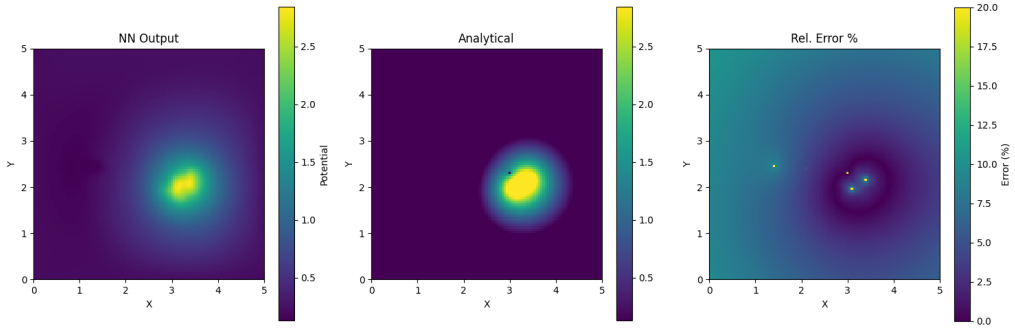


Figura F.9: Resultados del Caso 9 para cargas aleatorias

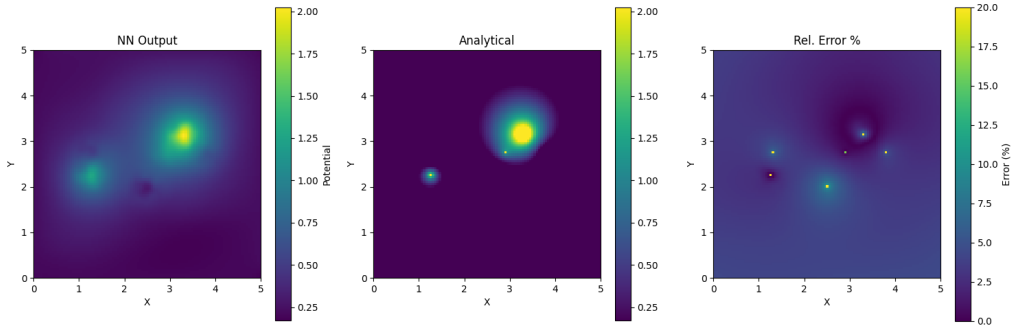


Figura F.10: Resultados del Caso 10 para cargas aleatorias

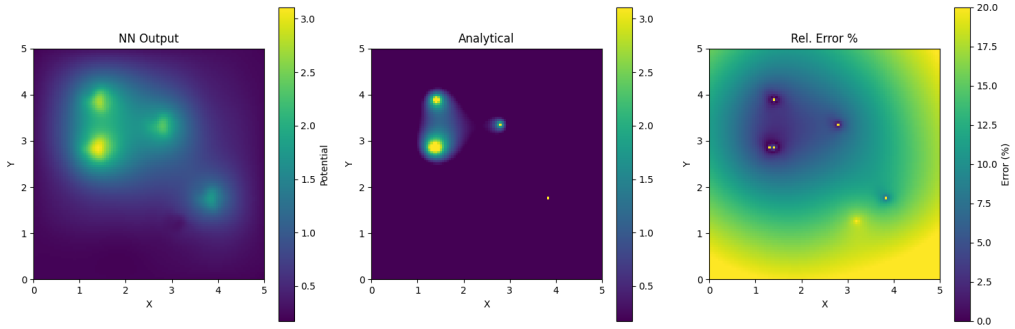


Figura F.11: Resultados del Caso 11 para cargas aleatorias

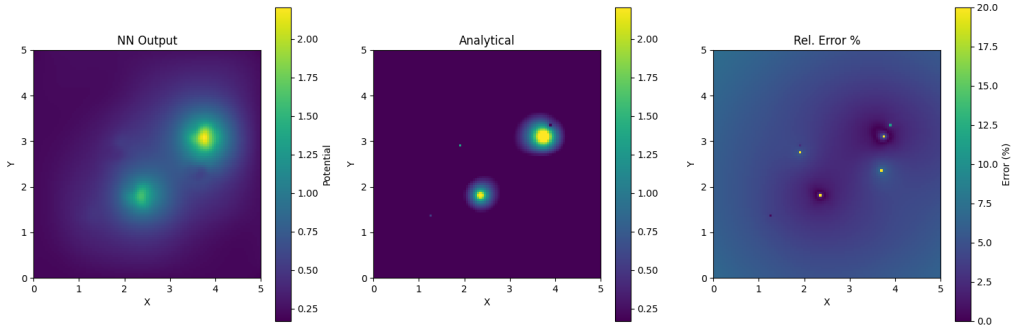


Figura F.12: Resultados del Caso 12 para cargas aleatorias

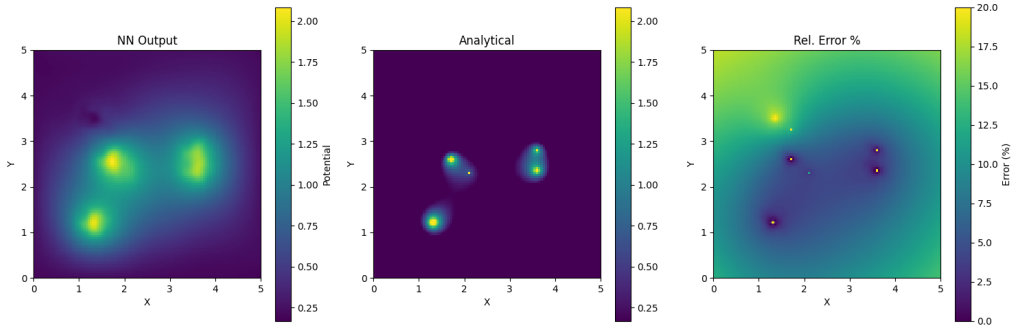


Figura F.13: Resultados del Caso 13 para cargas aleatorias

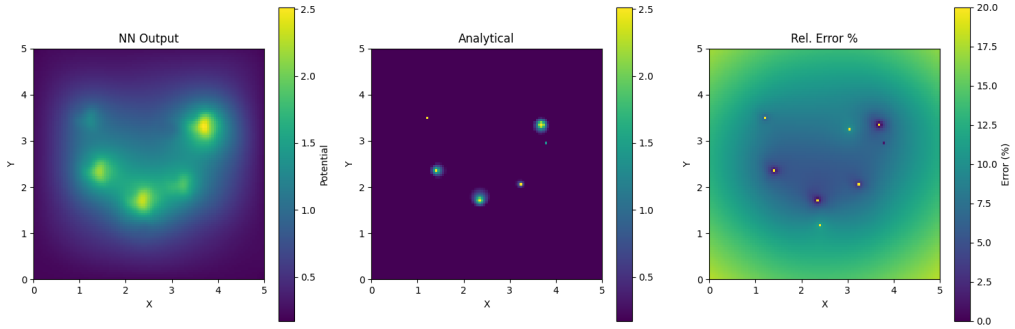


Figura F.14: Resultados del Caso 14 para cargas aleatorias

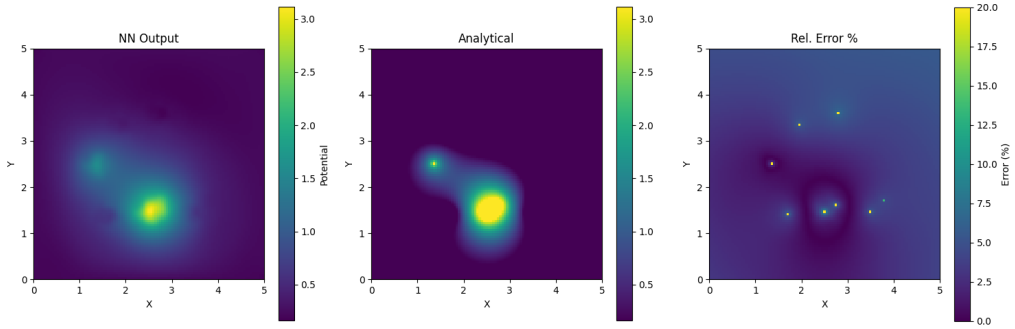


Figura F.15: Resultados del Caso 15 para cargas aleatorias

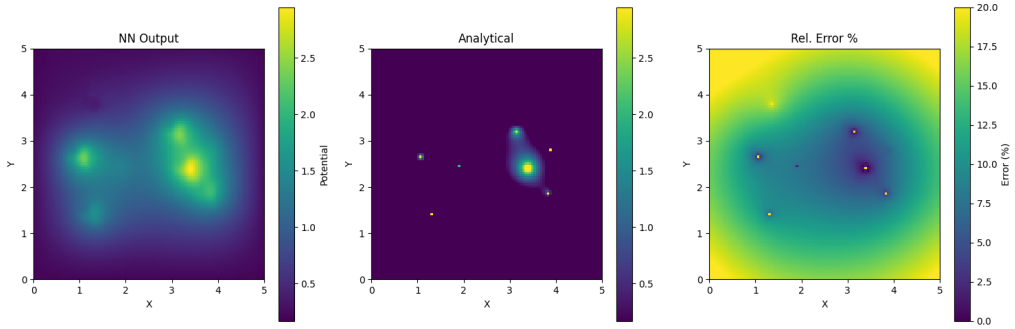


Figura F.16: Resultados del Caso 16 para cargas aleatorias

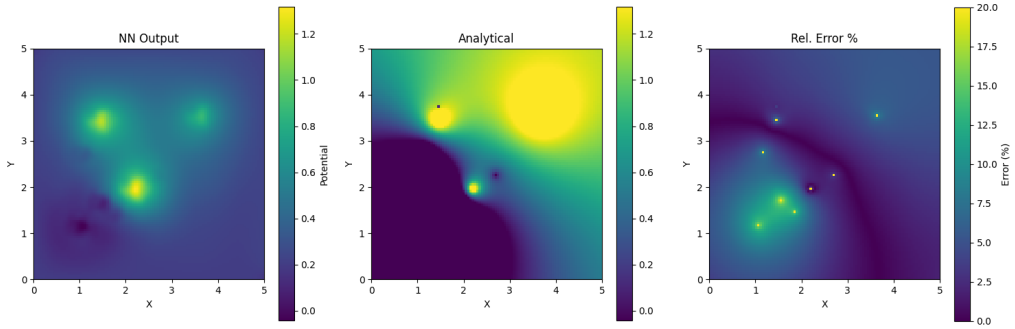


Figura F.17: Resultados del Caso 17 para cargas aleatorias

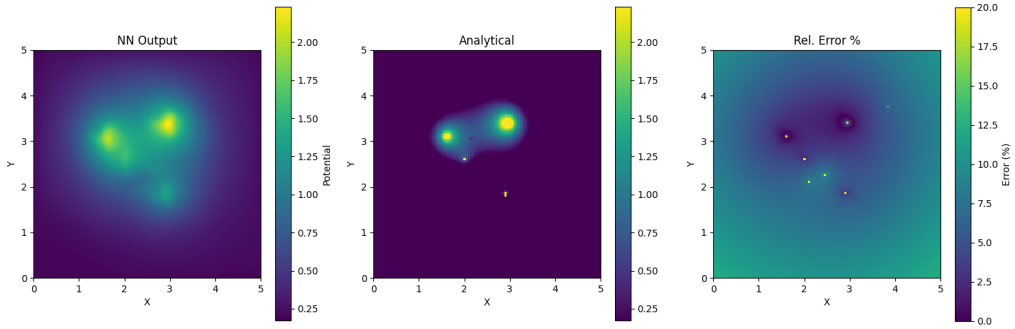


Figura F.18: Resultados del Caso 18 para cargas aleatorias

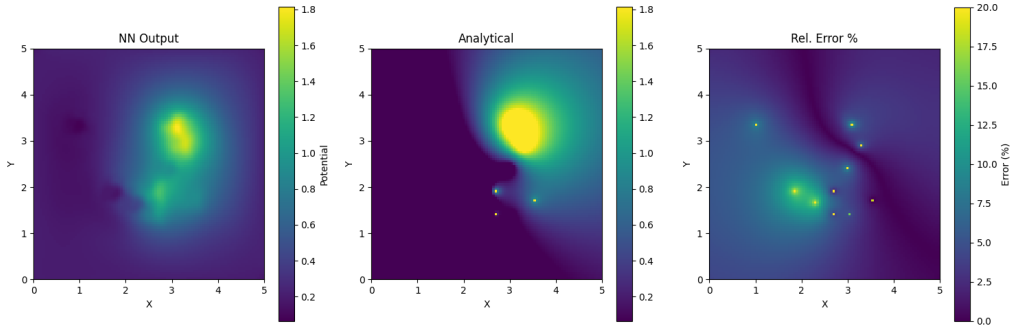


Figura F.19: Resultados del Caso 19 para cargas aleatorias

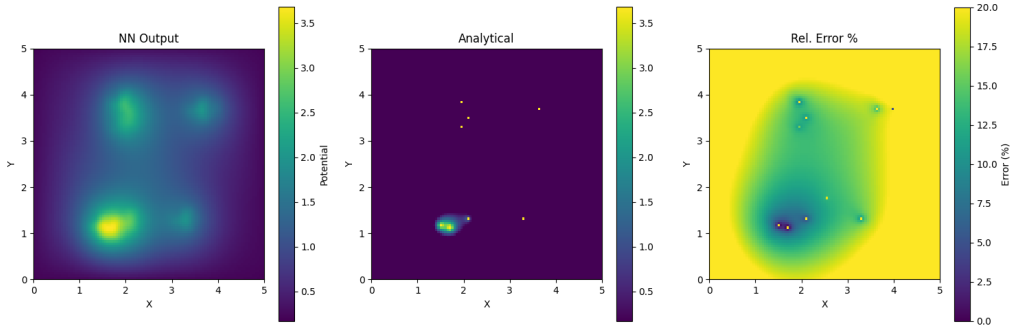


Figura F.20: Resultados del Caso 20 para cargas aleatorias

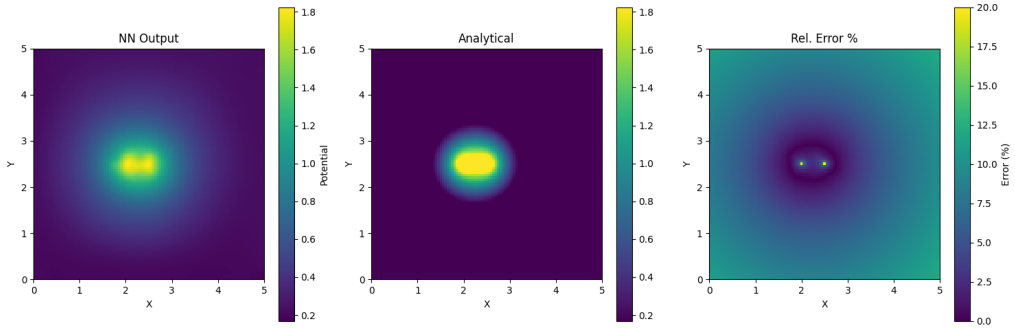


Figura F.21: Resultados del Caso 21 para cargas aleatorias

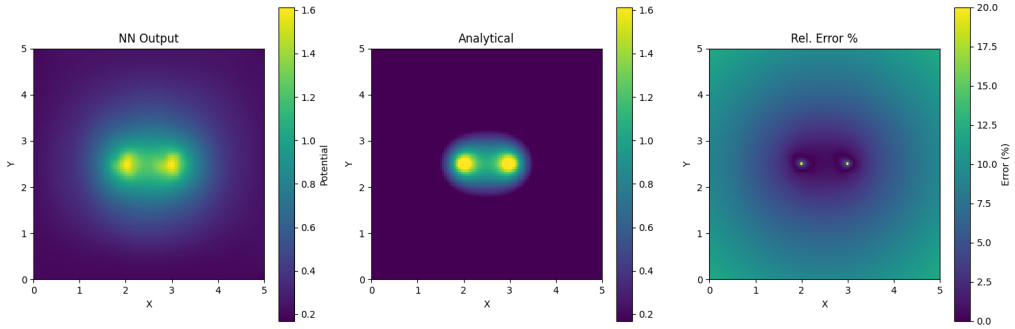


Figura F.22: Resultados del Caso 22 para cargas aleatorias

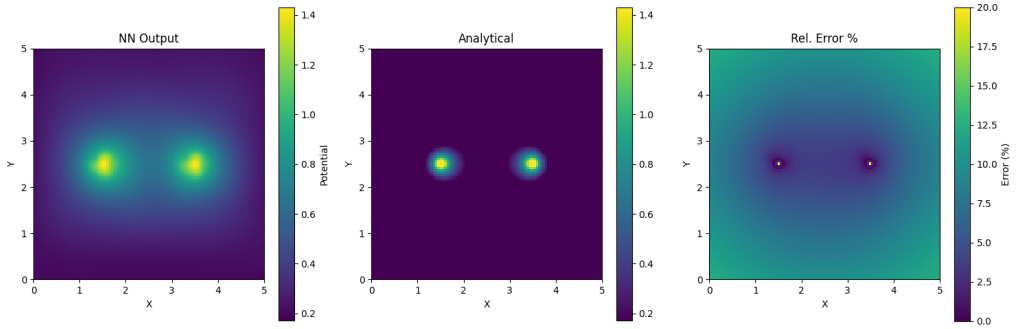


Figura F.23: Resultados del Caso 23 para cargas aleatorias

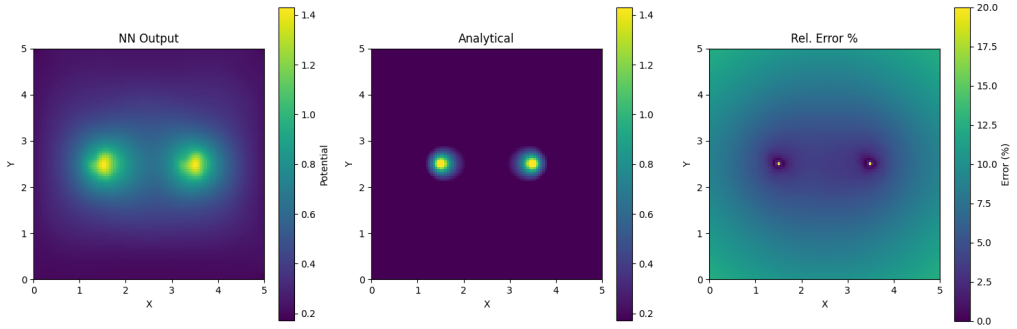


Figura F.24: Resultados del Caso 24 para cargas aleatorias

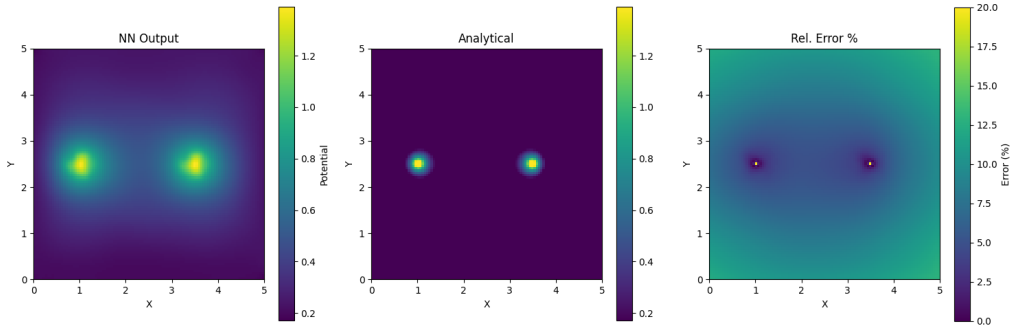


Figura F.25: Resultados del Caso 25 para cargas aleatorias

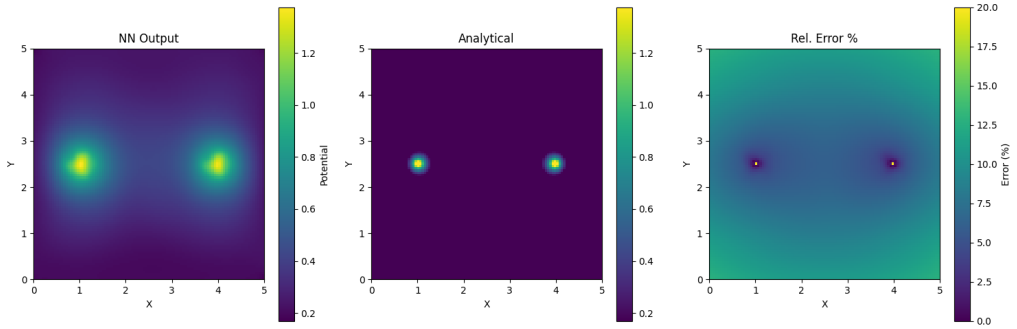


Figura F.26: Resultados del Caso 26 para cargas aleatorias

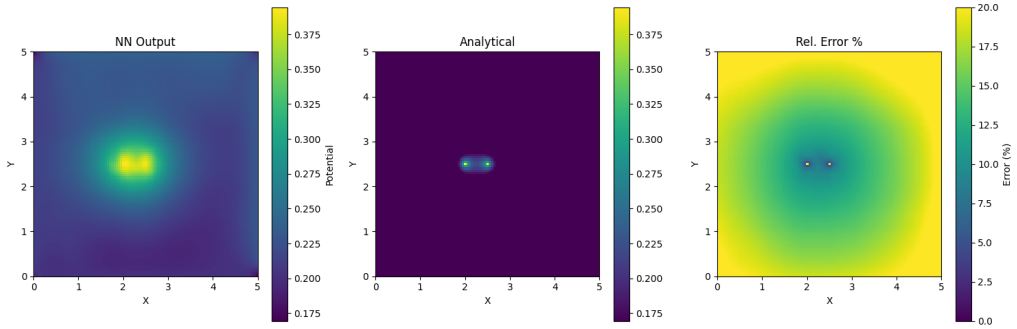


Figura F.27: Resultados del Caso 27 para cargas aleatorias

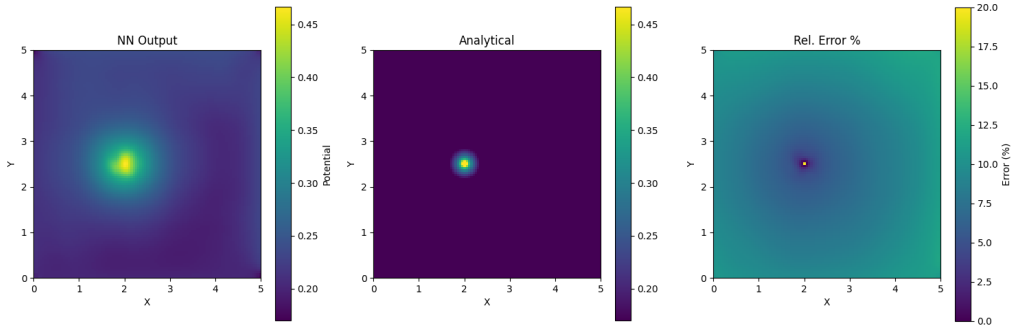


Figura F.28: Resultados del Caso 28 para cargas aleatorias

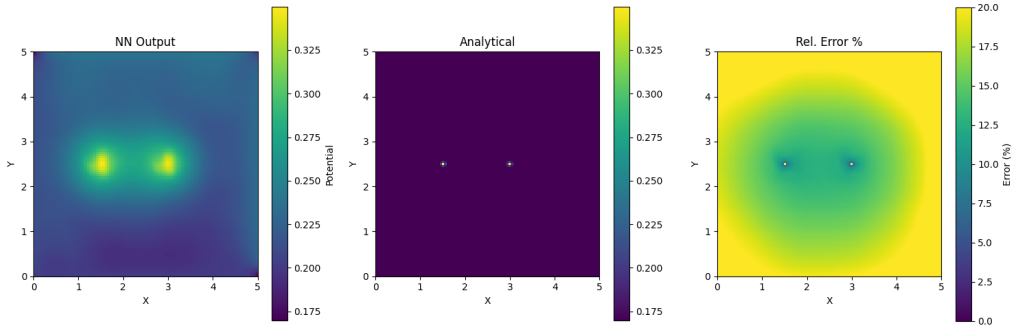


Figura F.29: Resultados del Caso 29 para cargas aleatorias

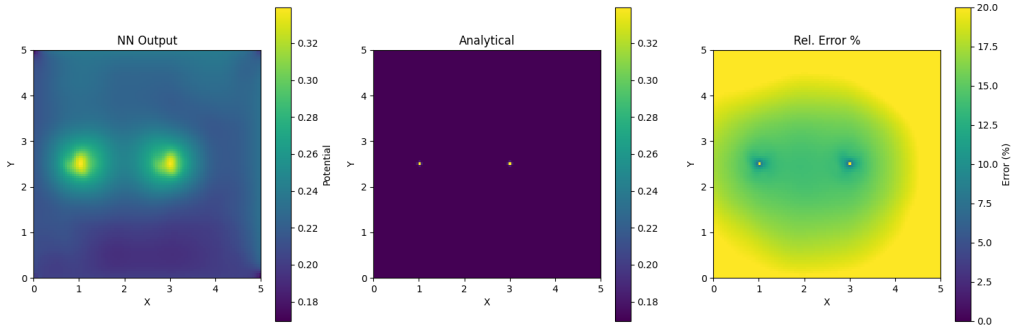


Figura F.30: Resultados del Caso 30 para cargas aleatorias

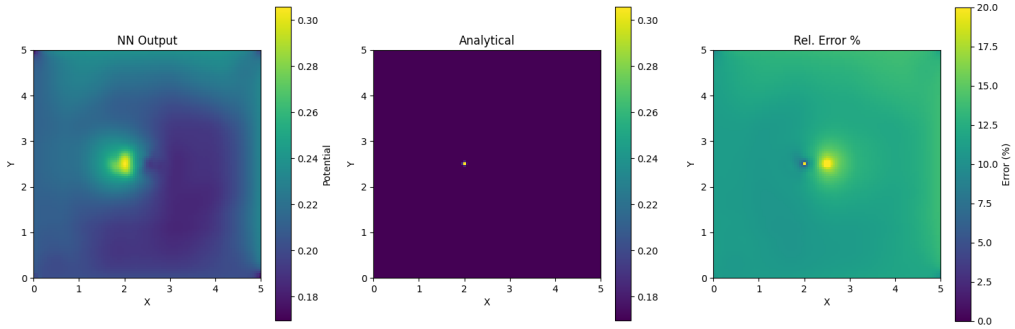


Figura F.31: Resultados del Caso 31 para cargas aleatorias

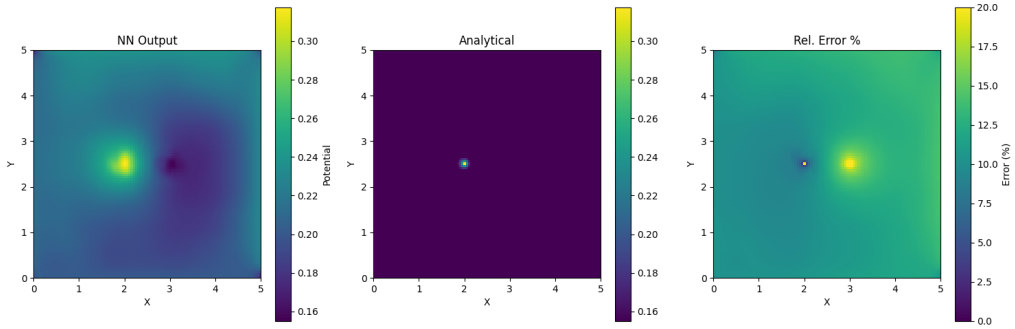


Figura F.32: Resultados del Caso 32 para cargas aleatorias

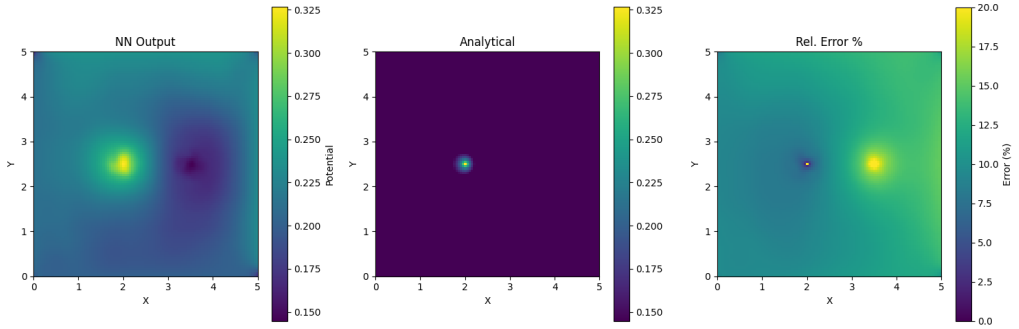


Figura F.33: Resultados del Caso 33 para cargas aleatorias

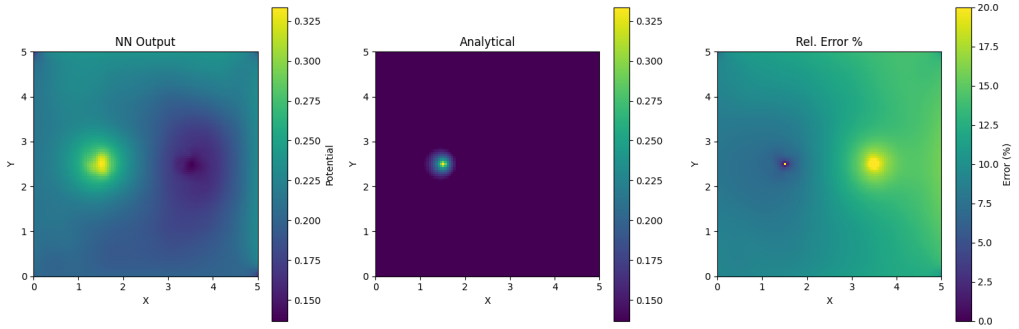


Figura F.34: Resultados del Caso 34 para cargas aleatorias

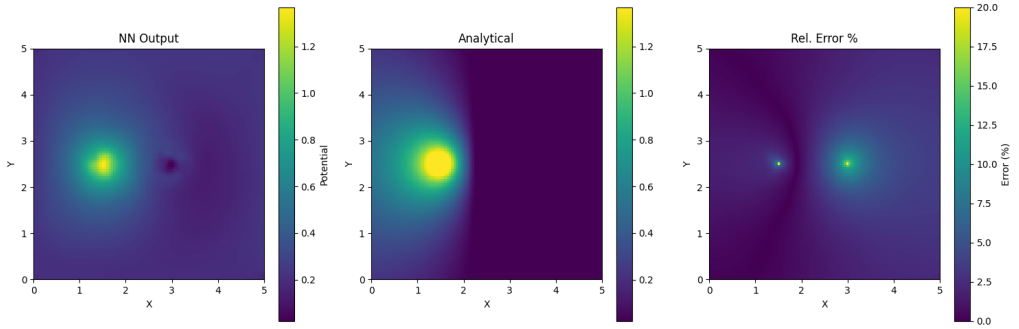


Figura F.35: Resultados del Caso 35 para cargas aleatorias

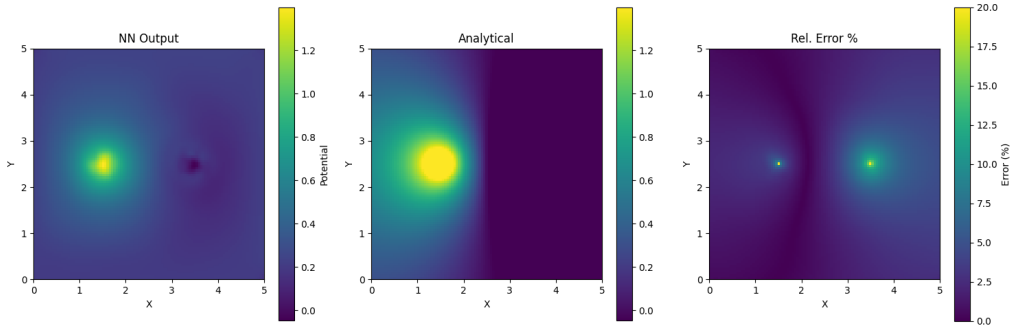


Figura F.36: Resultados del Caso 36 para cargas aleatorias

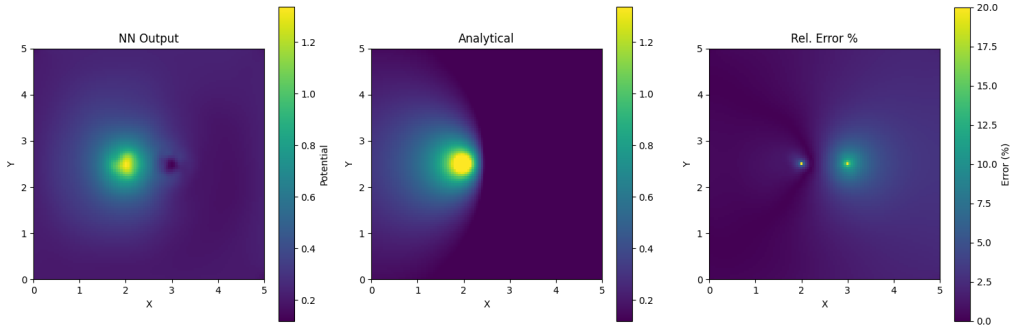


Figura F.37: Resultados del Caso 37 para cargas aleatorias

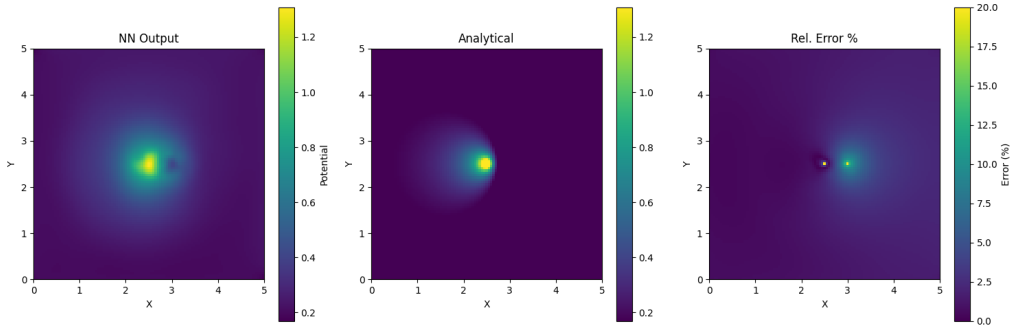


Figura F.38: Resultados del Caso 38 para cargas aleatorias

G. Resultados de Casos 2D Cargas Distribuidas Condiciones de Contorno No Homogéneas

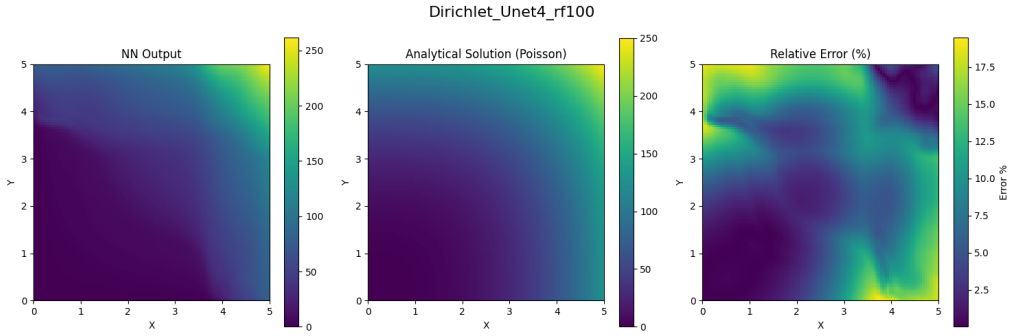


Figura G.1: Resultados del Caso 1 Condiciones de Contorno No Homogéneas

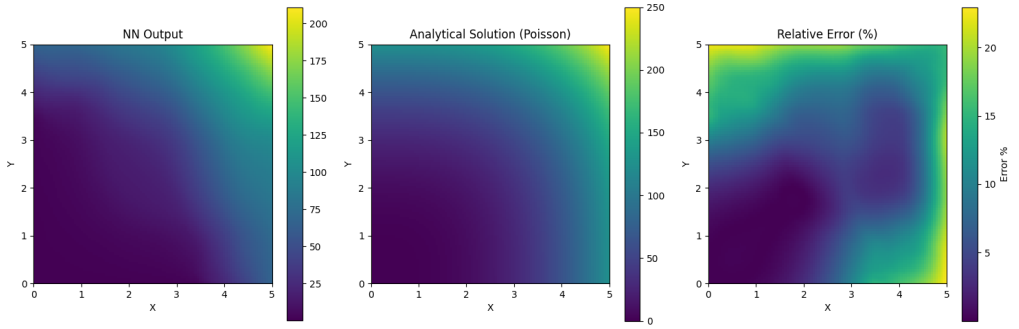


Figura G.2: Resultados del Caso 2 Condiciones de Contorno No Homogéneas

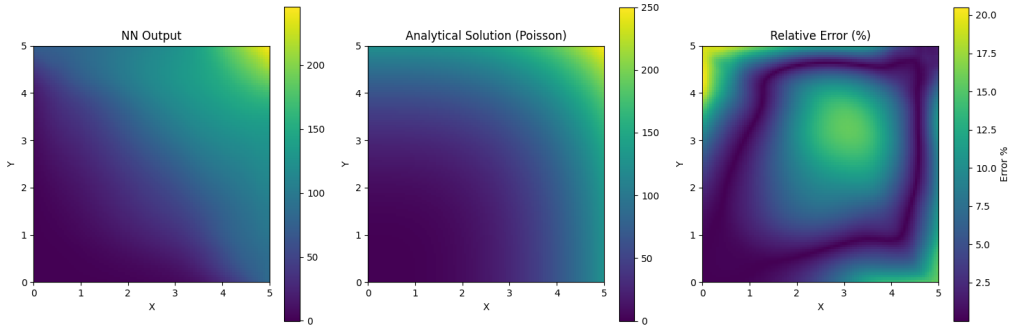


Figura G.3: Resultados del Caso 3 Condiciones de Contorno No Homogéneas

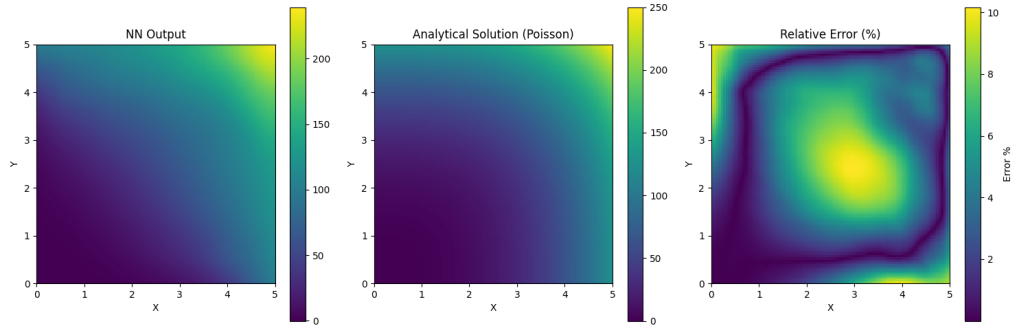


Figura G.4: Resultados del Caso 4 Condiciones de Contorno No Homogéneas

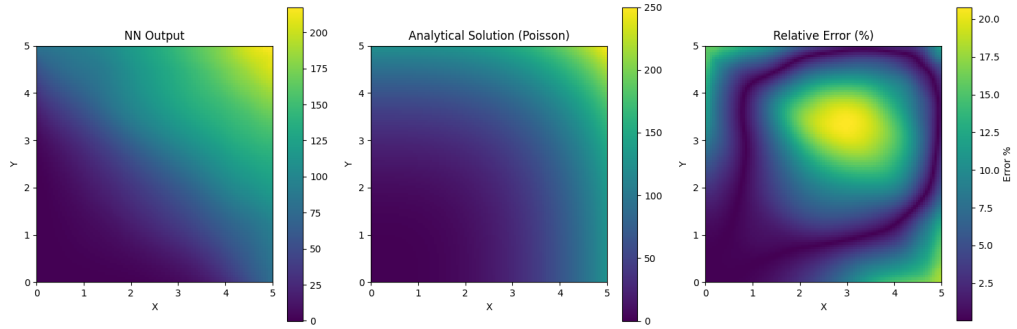


Figura G.5: Resultados del Caso 5 Condiciones de Contorno No Homogéneas

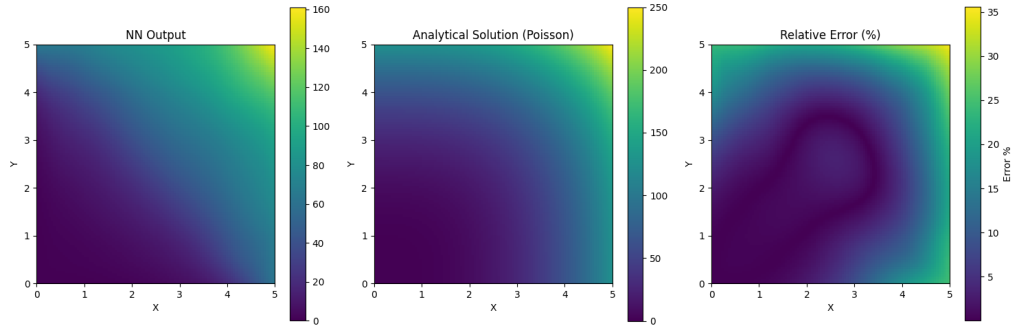


Figura G.6: Resultados del Caso 6 Condiciones de Contorno No Homogéneas

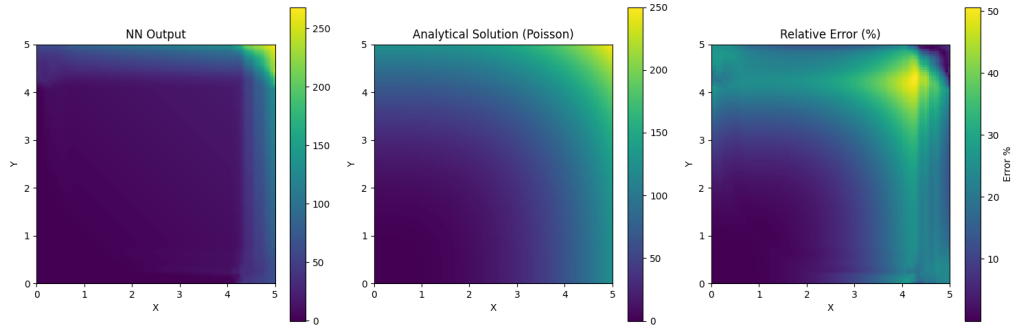


Figura G.7: Resultados del Caso 7 Condiciones de Contorno No Homogéneas

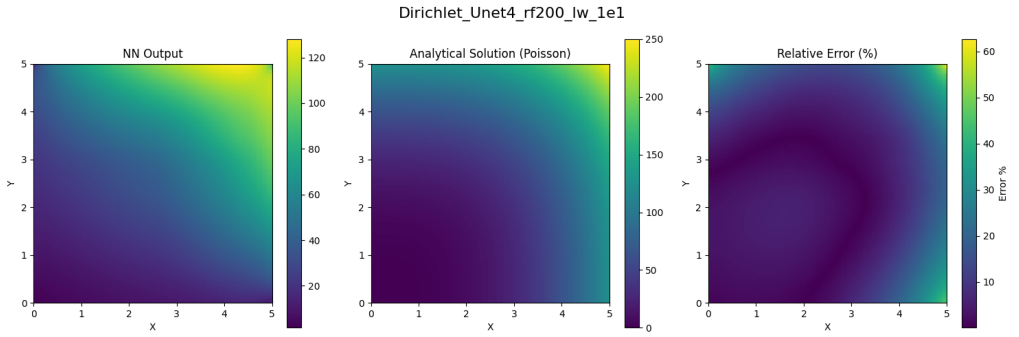


Figura G.8: Resultados del Caso 8 Condiciones de Contorno No Homogéneas

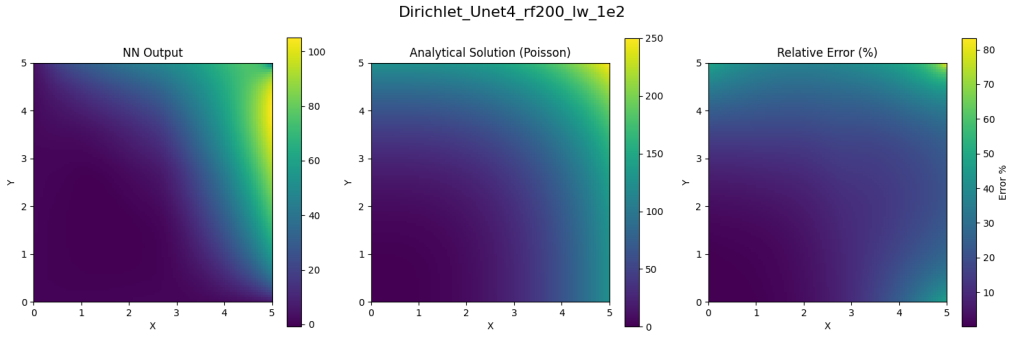


Figura G.9: Resultados del Caso 9 Condiciones de Contorno No Homogéneas

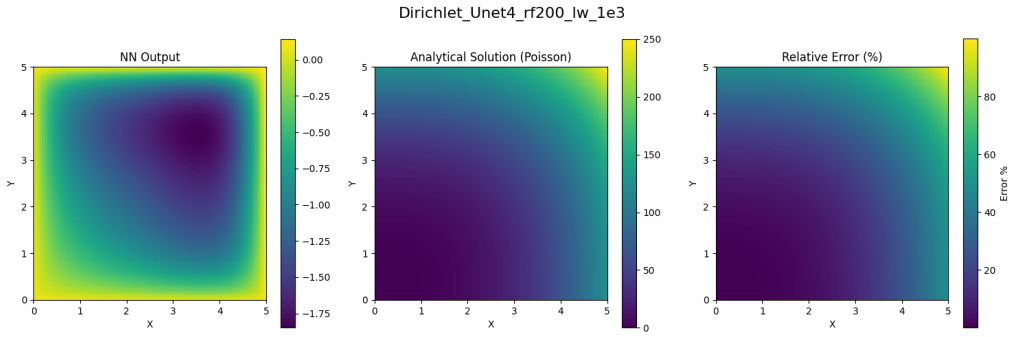


Figura G.10: Resultados del Caso 10 Condiciones de Contorno No Homogéneas

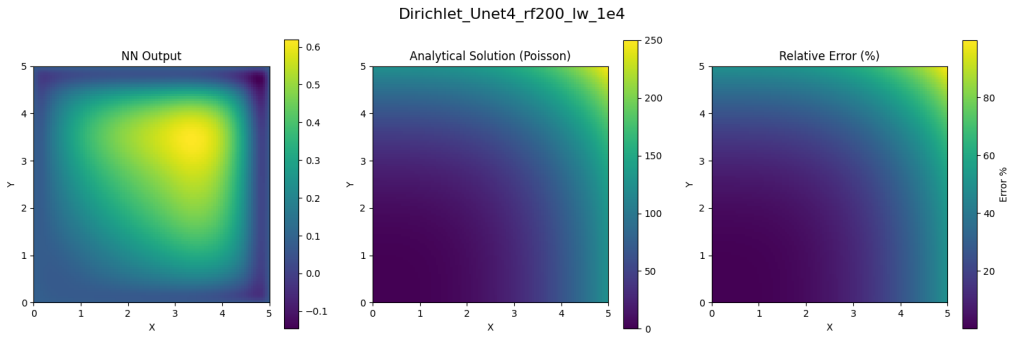


Figura G.11: Resultados del Caso 11 Condiciones de Contorno No Homogéneas

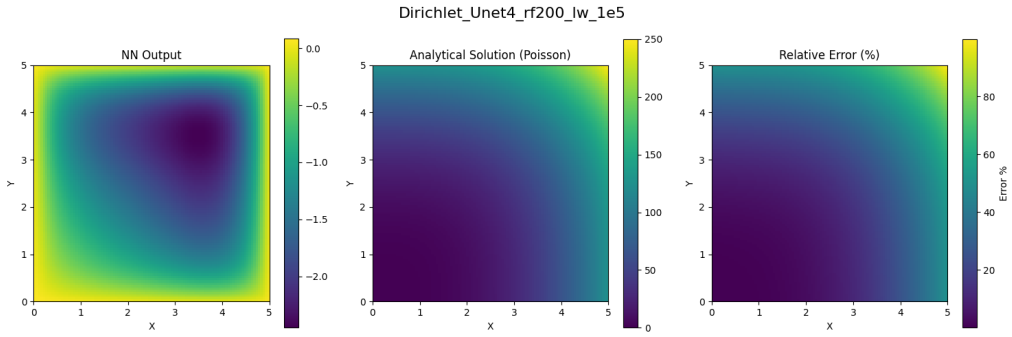


Figura G.12: Resultados del Caso 12 Condiciones de Contorno No Homogéneas

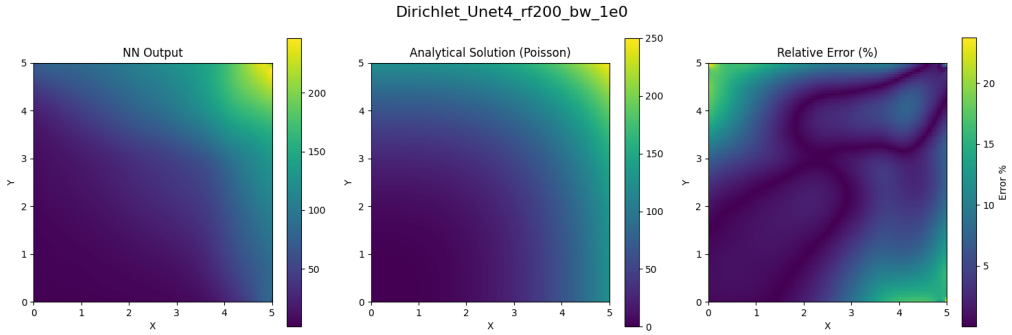


Figura G.13: Resultados del Caso 13 Condiciones de Contorno No Homogéneas

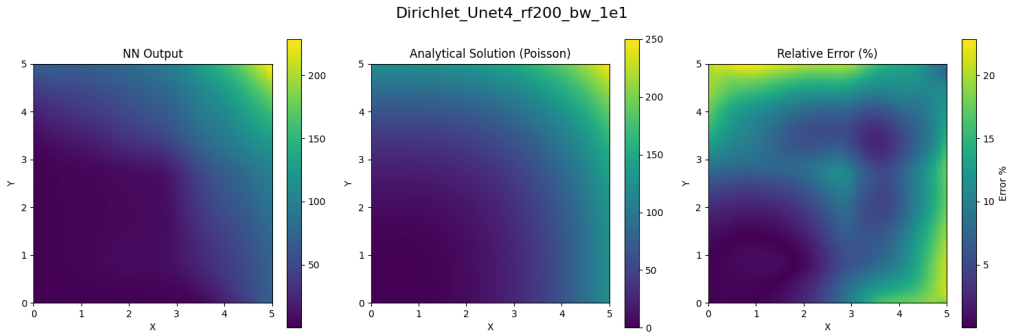


Figura G.14: Resultados del Caso 14 Condiciones de Contorno No Homogéneas

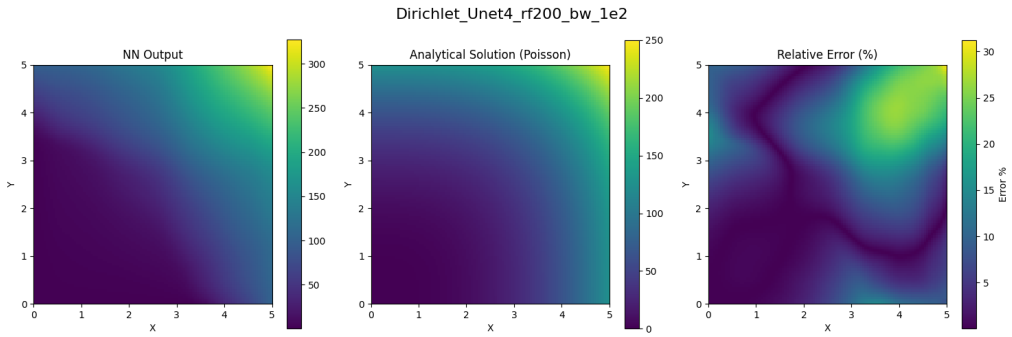


Figura G.15: Resultados del Caso 15 Condiciones de Contorno No Homogéneas

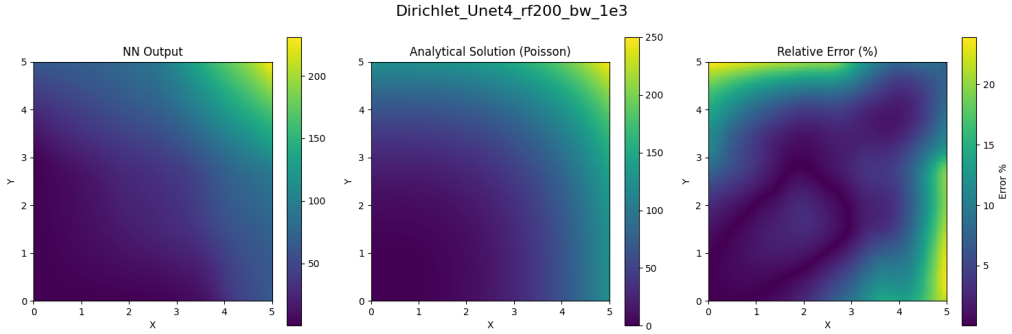


Figura G.16: Resultados del Caso 16 Condiciones de Contorno No Homogéneas

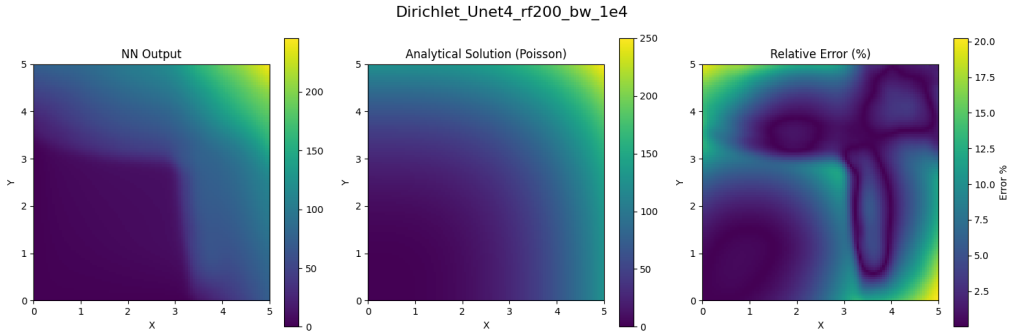


Figura G.17: Resultados del Caso 17 Condiciones de Contorno No Homogéneas

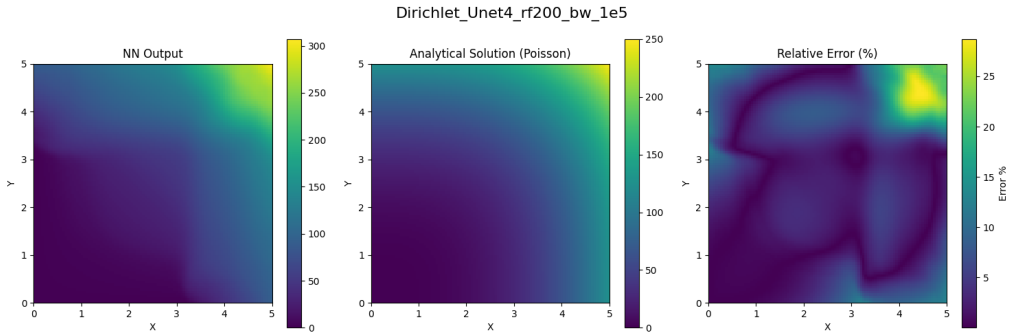


Figura G.18: Resultados del Caso 18 Condiciones de Contorno No Homogéneas

H. Resultados de Casos 3D Cargas Puntuales

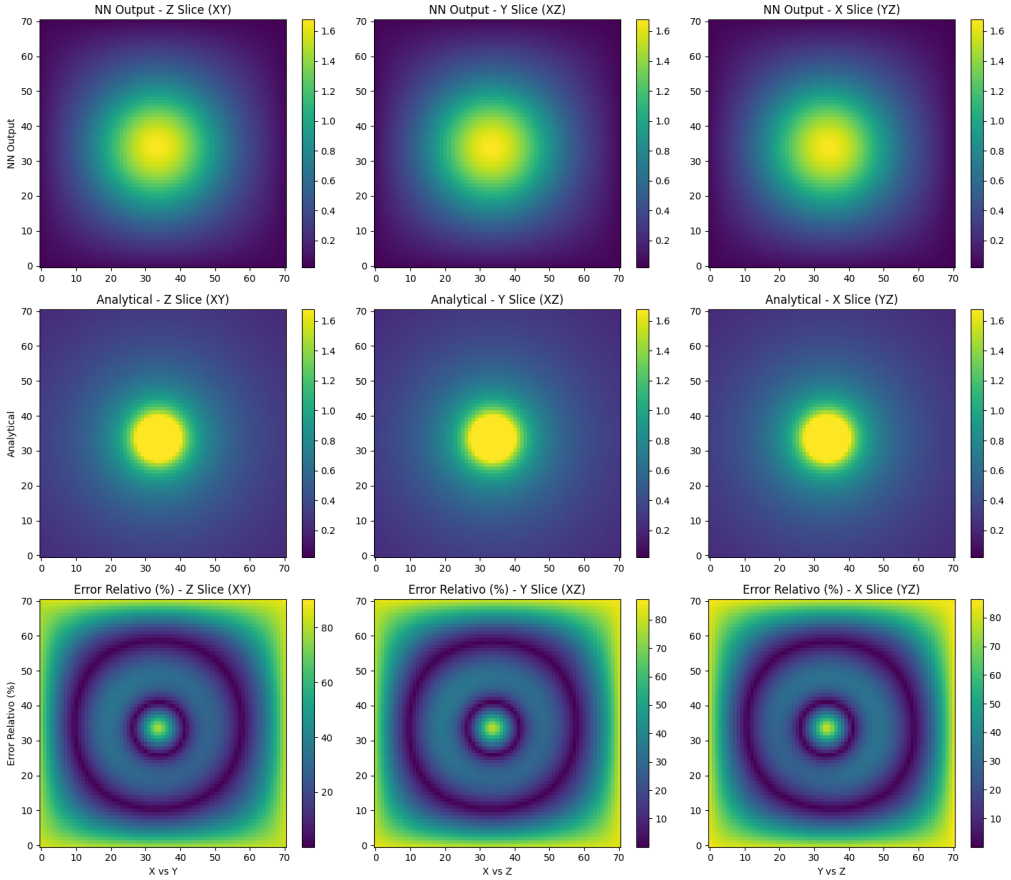


Figura H.1: Resultados del Caso 1 Carga Puntual 3D

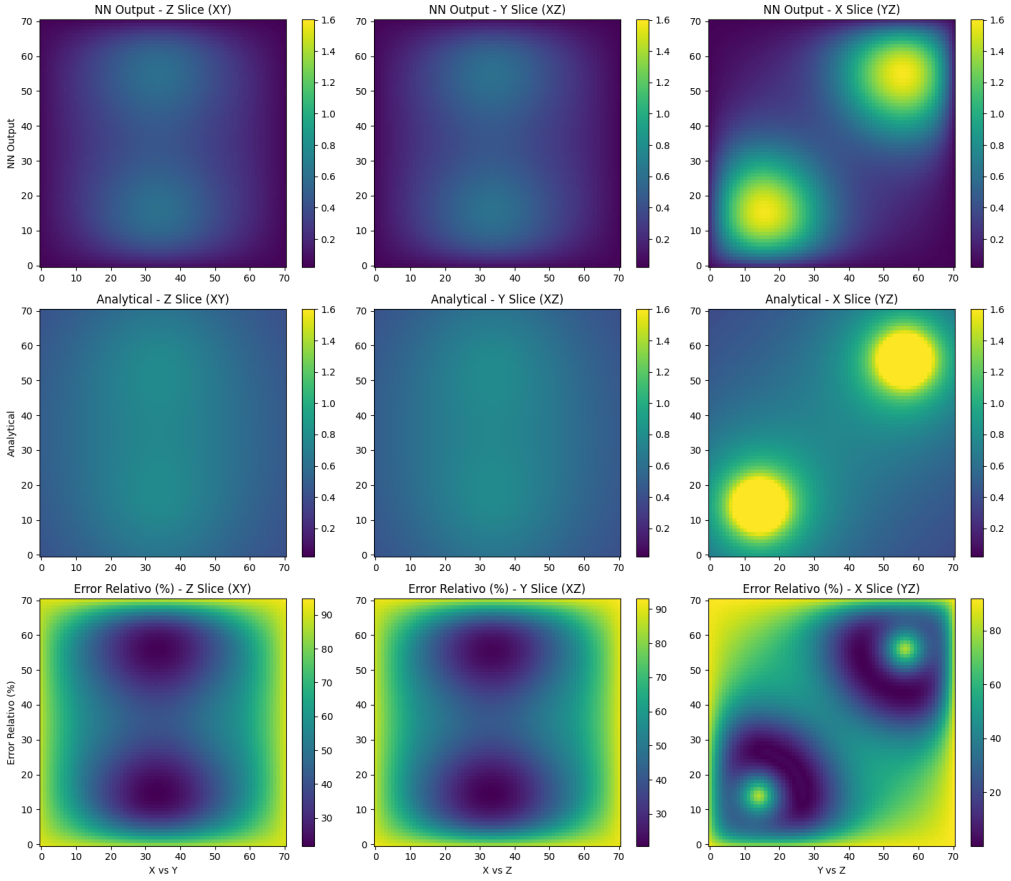


Figura H.2: Resultados del Caso 2 Carga Puntual 3D

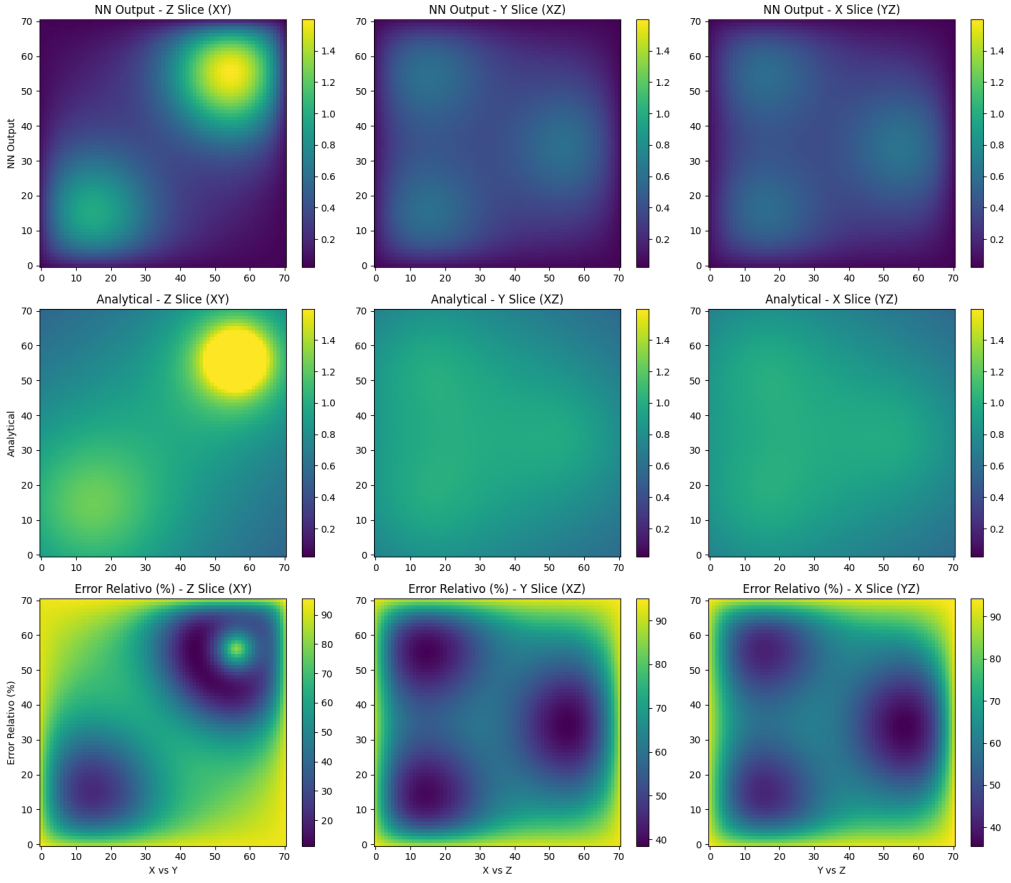


Figura H.3: Resultados del Caso 3 Carga Puntual 3D

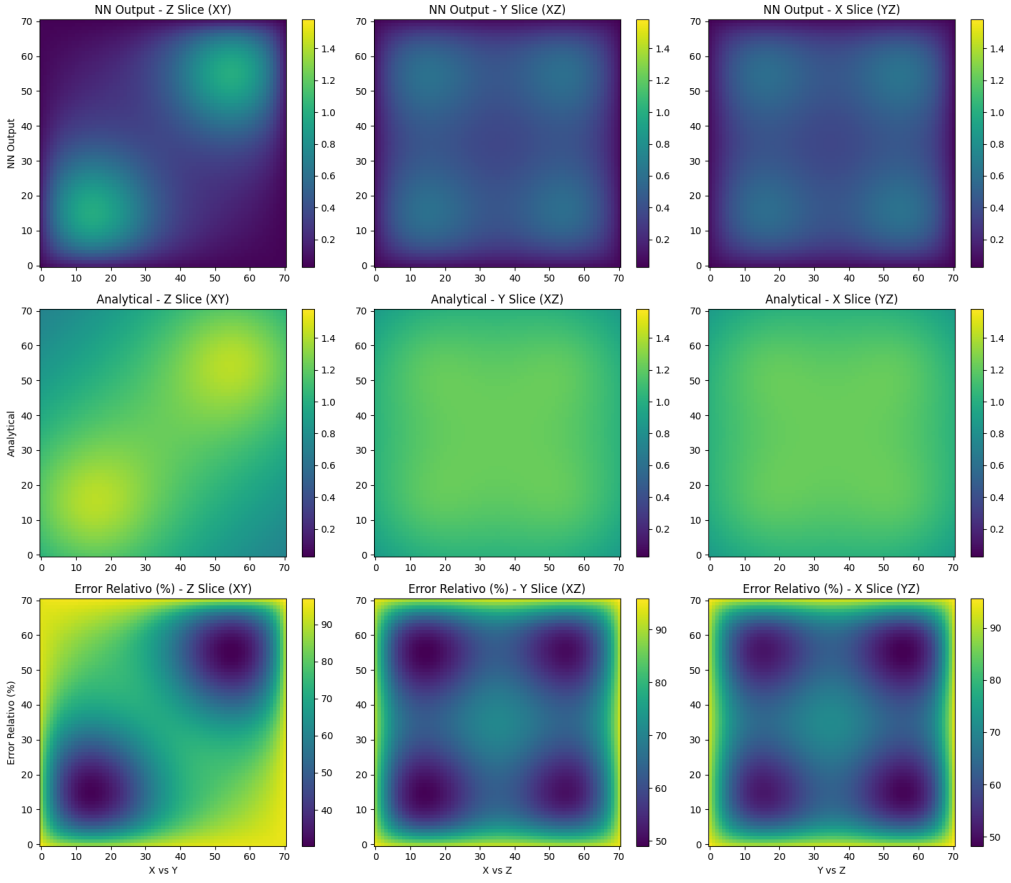


Figura H.4: Resultados del Caso 4 Carga Puntual 3D

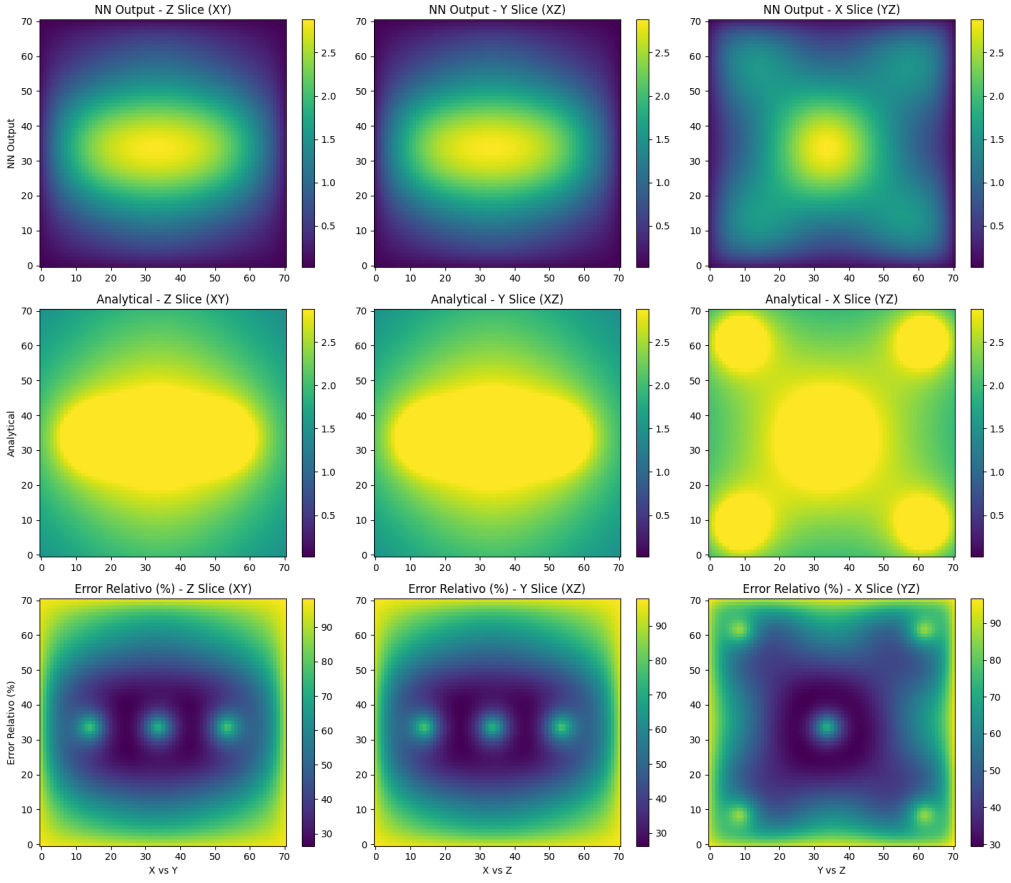


Figura H.5: Resultados del Caso 5 Carga Puntual 3D

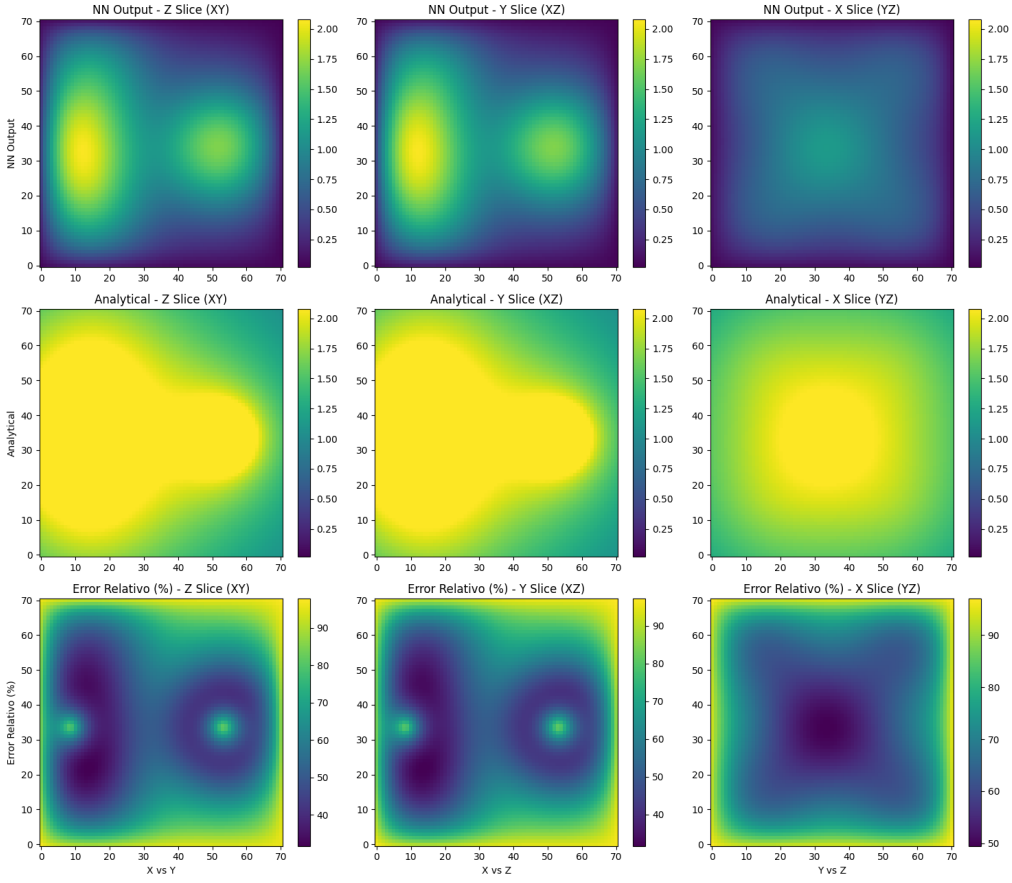


Figura H.6: Resultados del Caso 6 Carga Puntual 3D

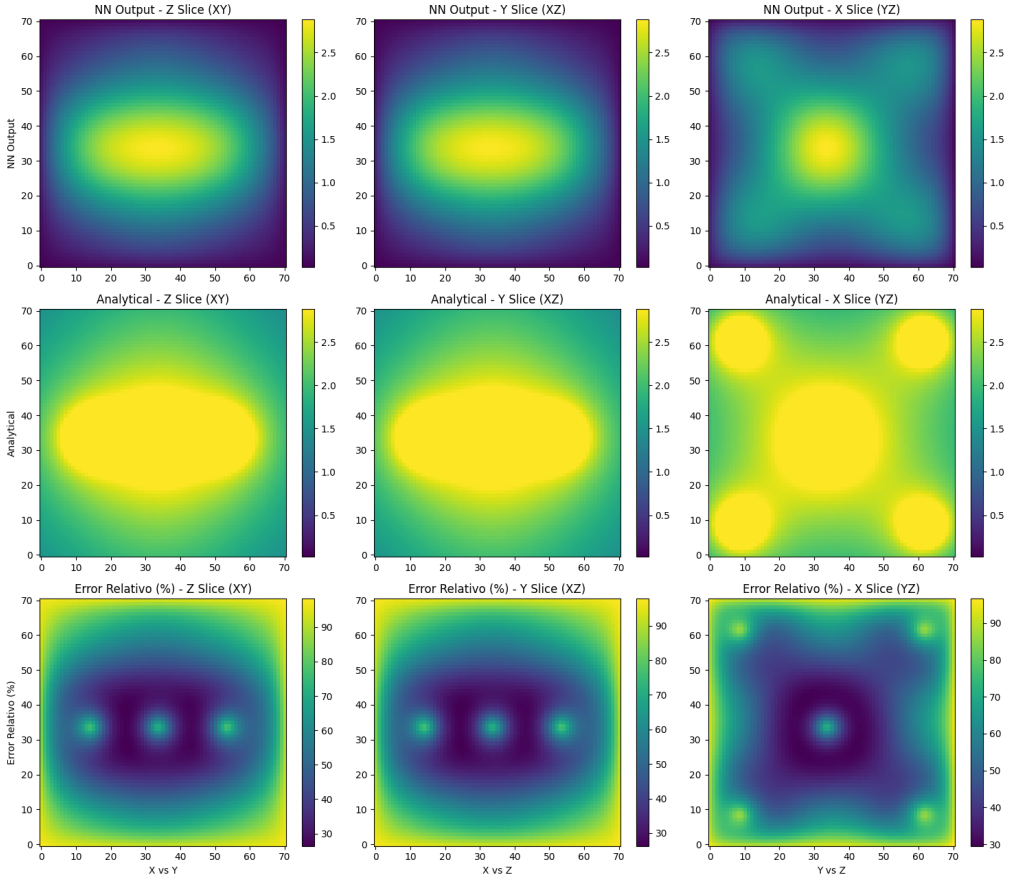


Figura H.7: Resultados del Caso 7 Carga Puntual 3D

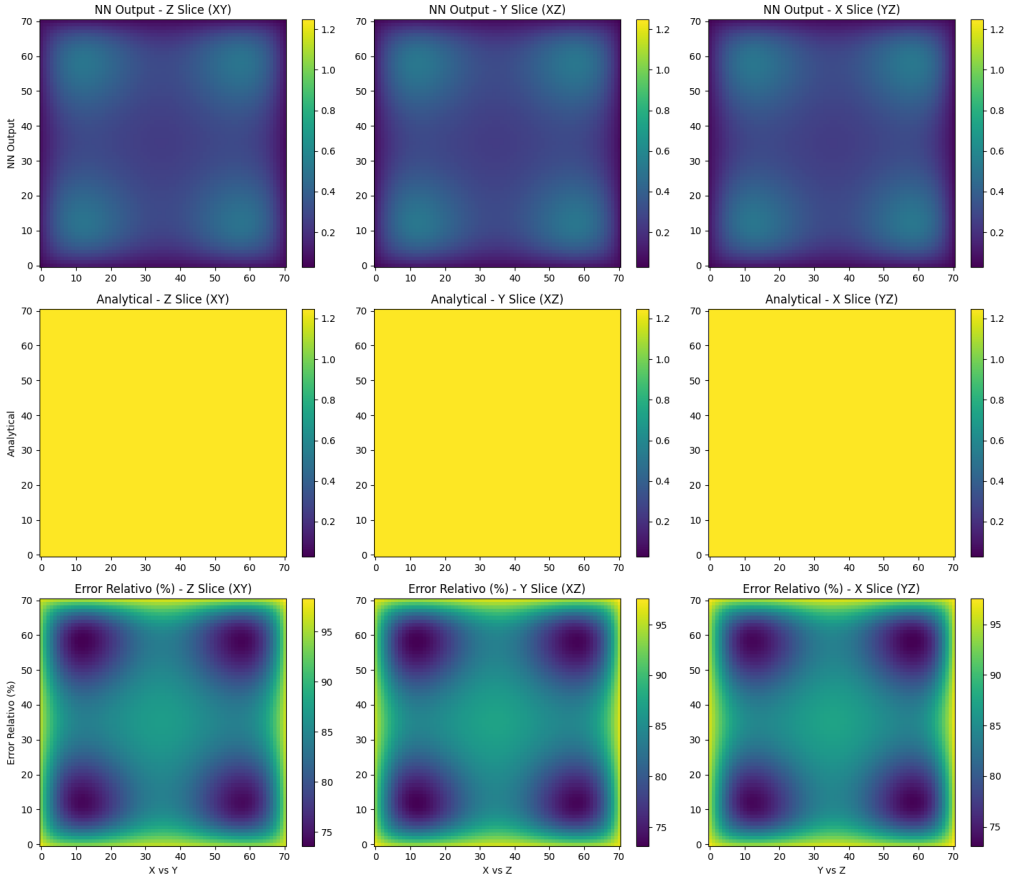


Figura H.8: Resultados del Caso 8 Carga Puntual 3D

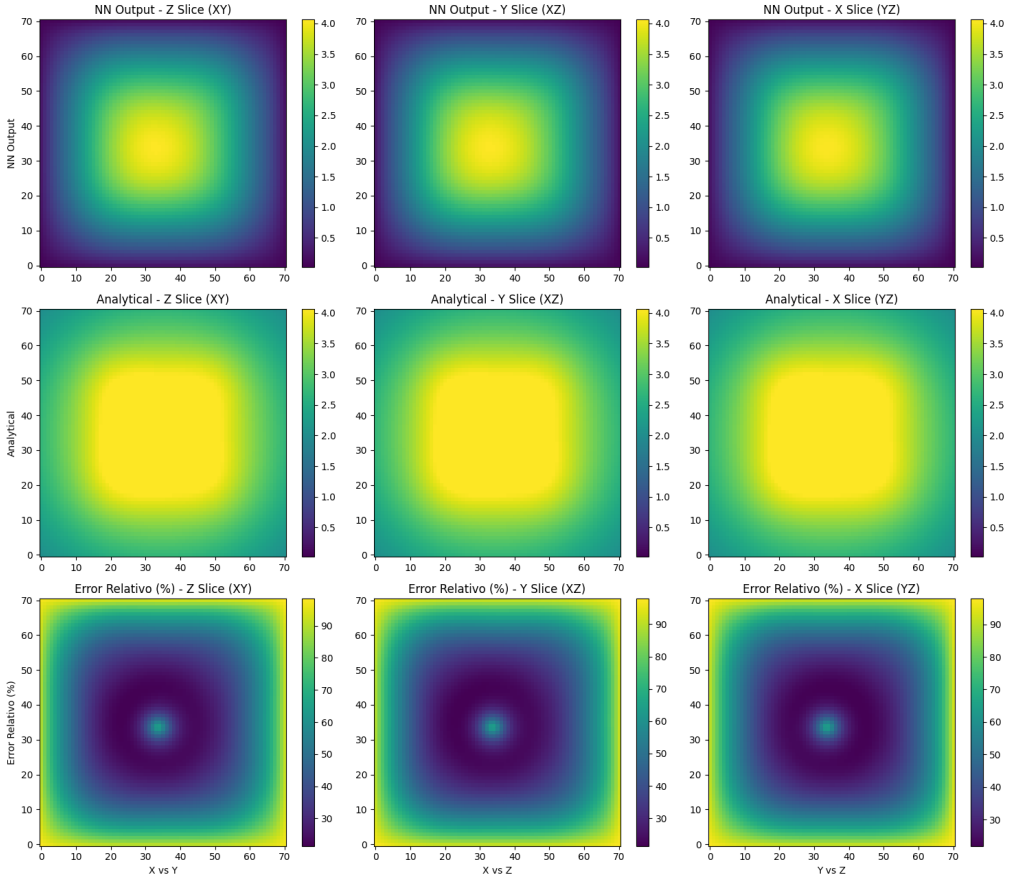


Figura H.9: Resultados del Caso 9 Carga Puntual 3D

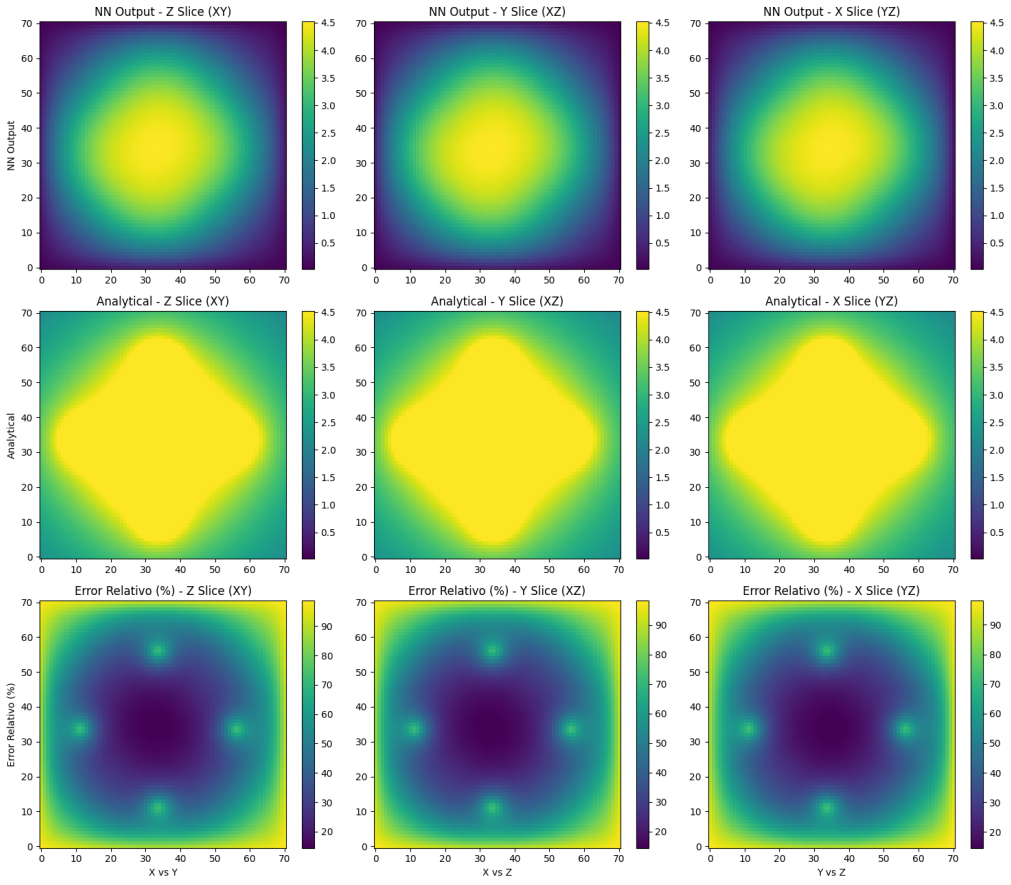


Figura H.10: Resultados del Caso 10 Carga Puntual 3D

I. Resultados de Casos 3D Interfaces No regularizado

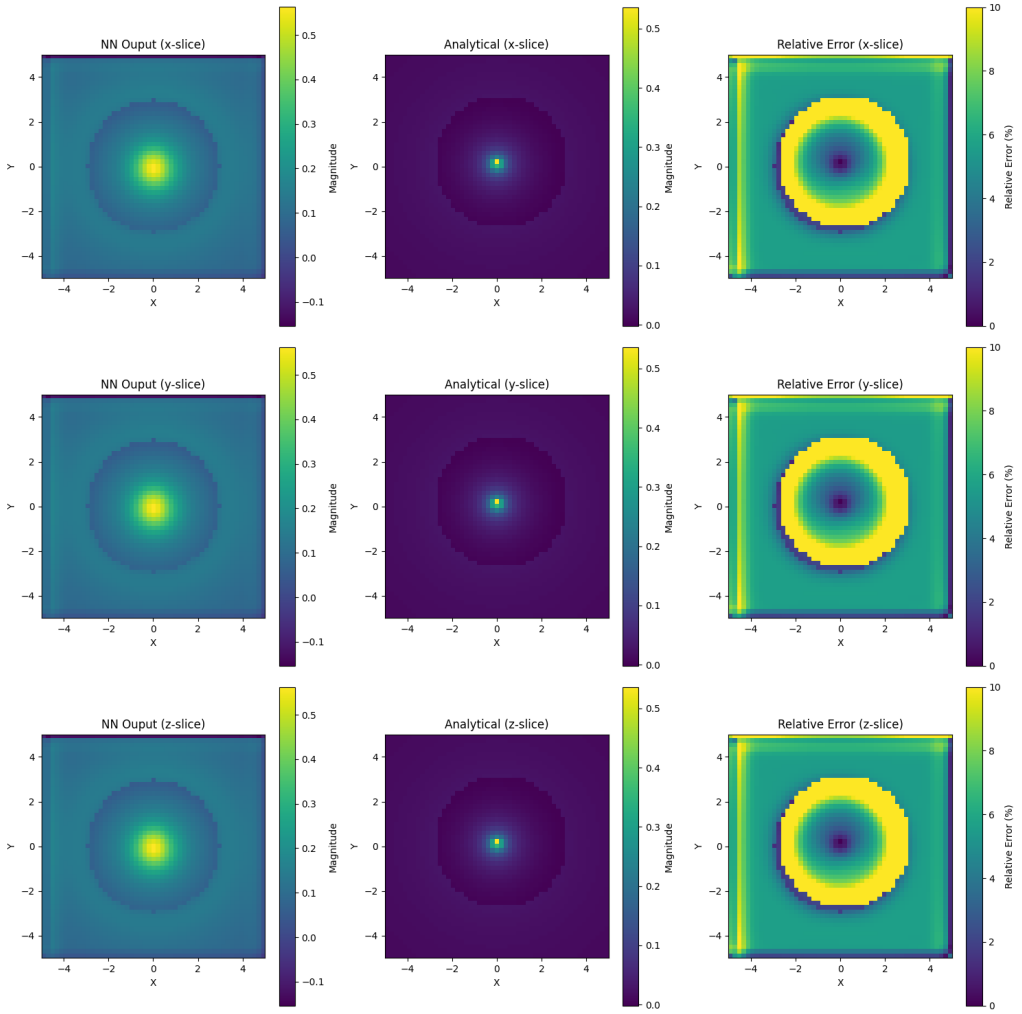


Figura I.1: Resultados del Caso 1 Interfaces 3D No Regularizado

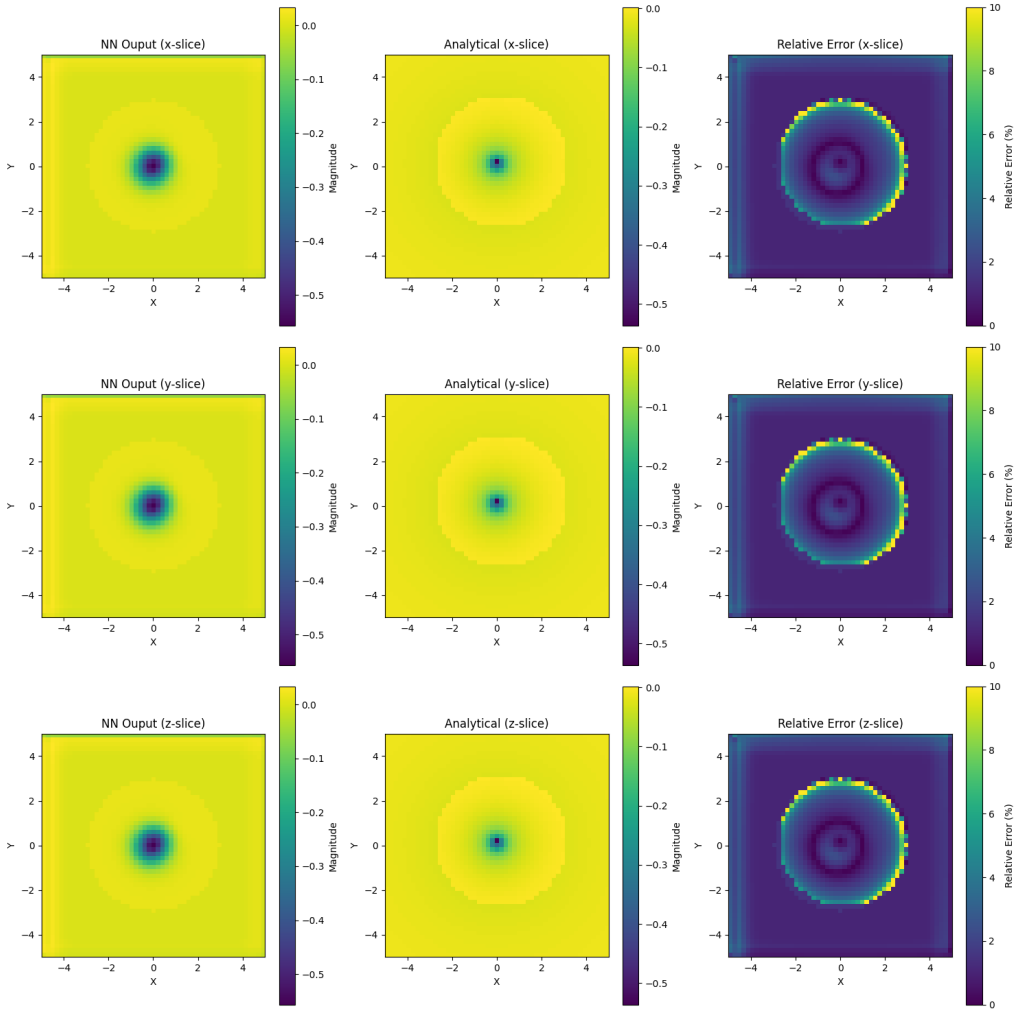


Figura I.2: Resultados del Caso 2 Interfaces 3D No Regularizado

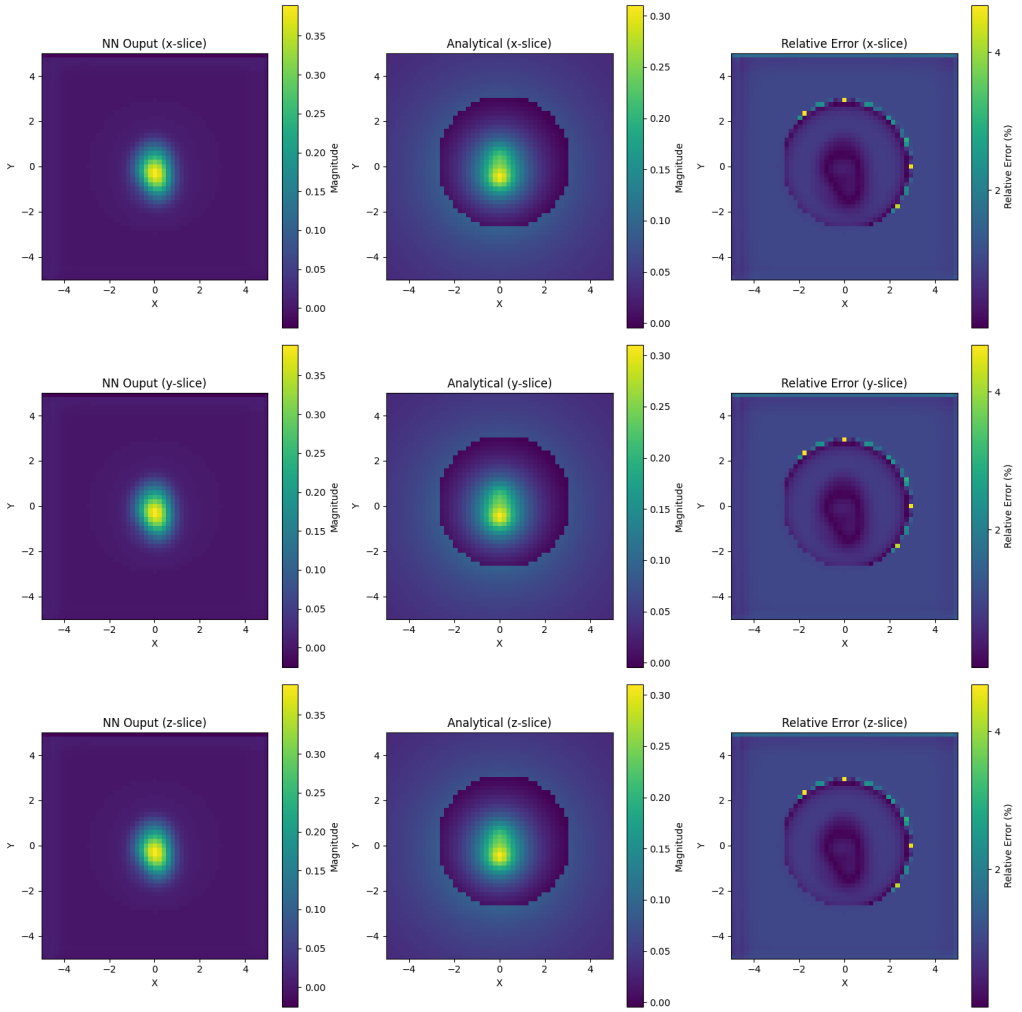


Figura I.3: Resultados del Caso 3 Interfaces 3D No Regularizado

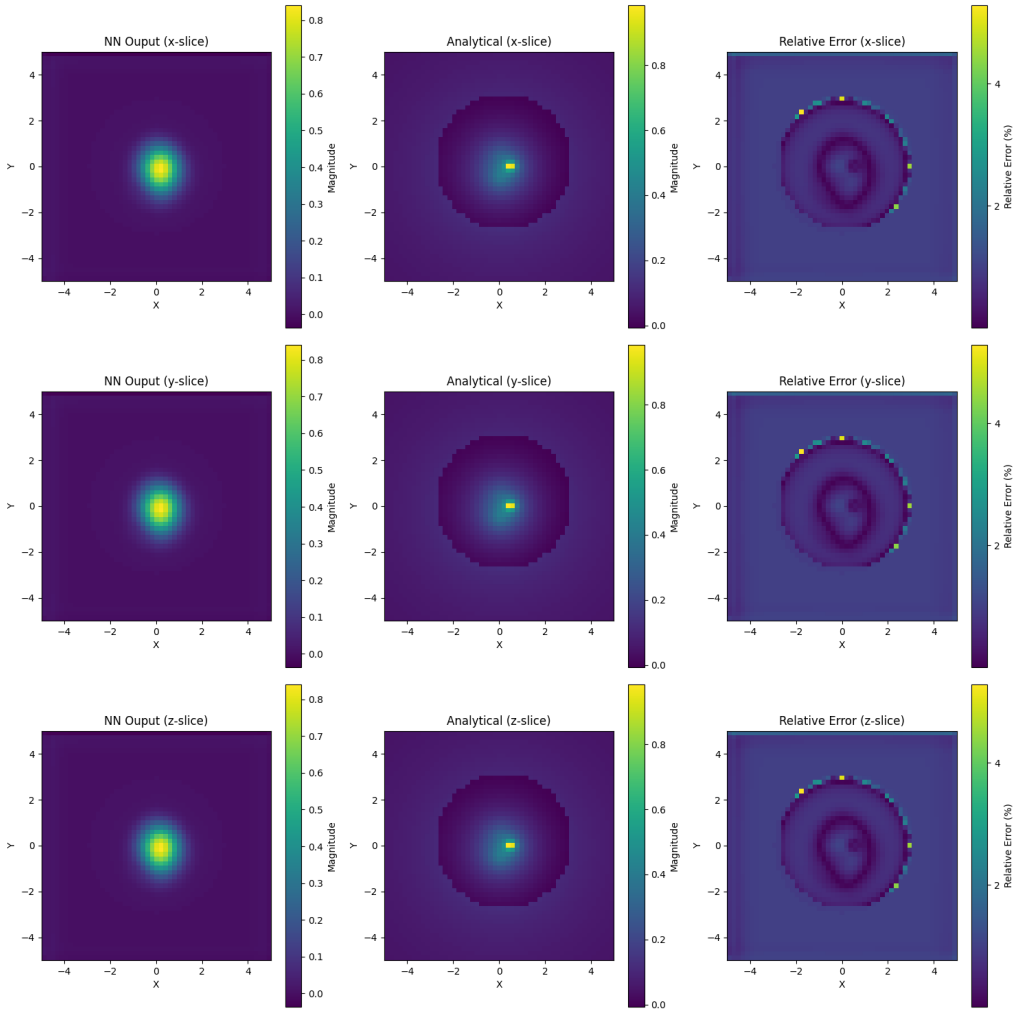


Figura I.4: Resultados del Caso 4 Interfaces 3D No Regularizado

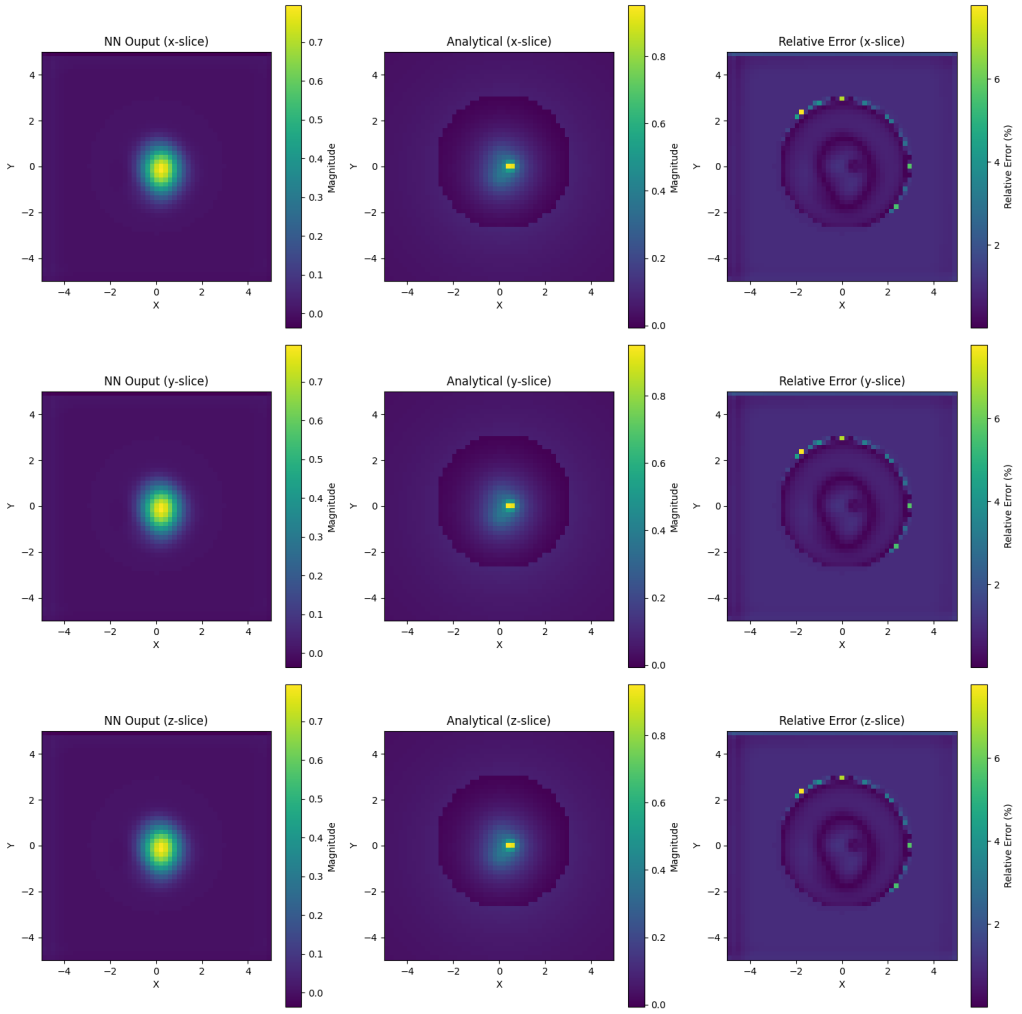


Figura I.5: Resultados del Caso 5 Interfaces 3D No Regularizado

J. Resultados de Casos 3D Interfaces Regularizadas

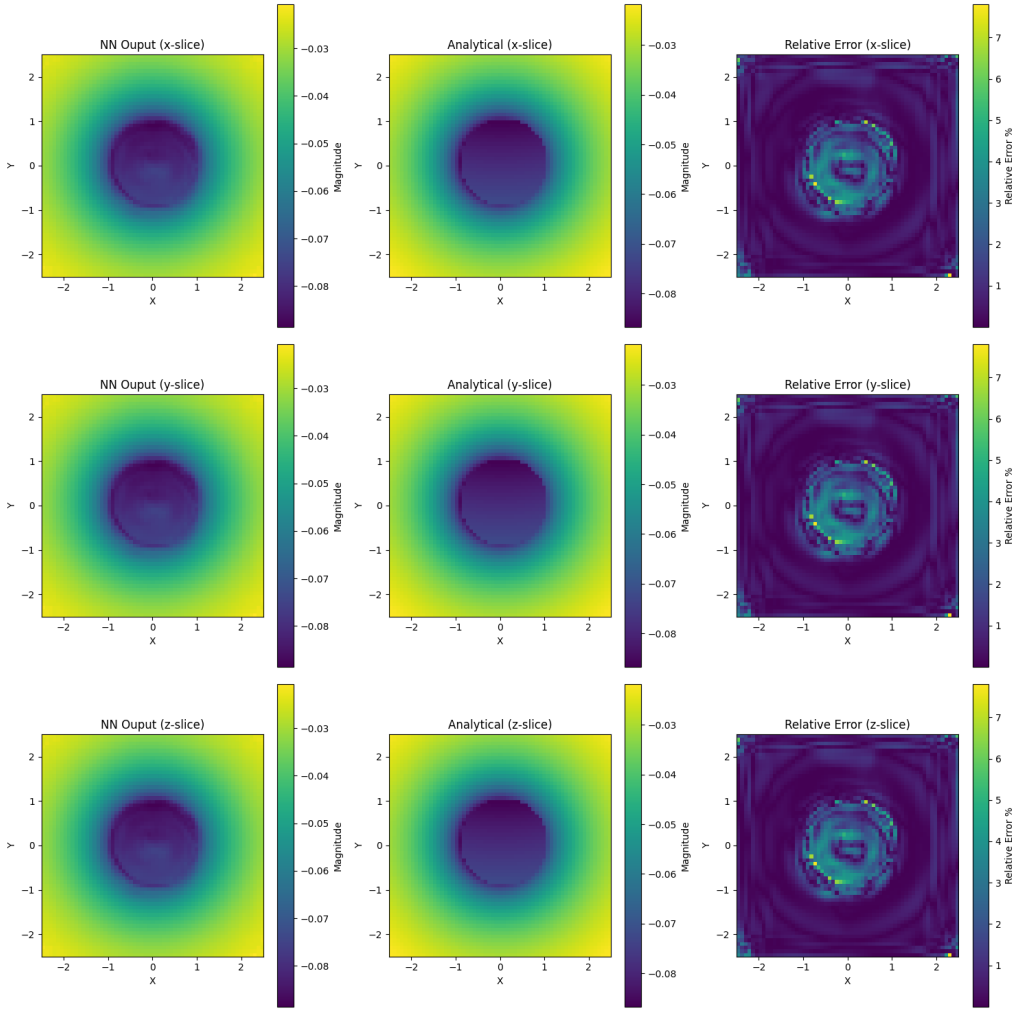


Figura J.1: Resultados del Caso 1 Interfaces 3D

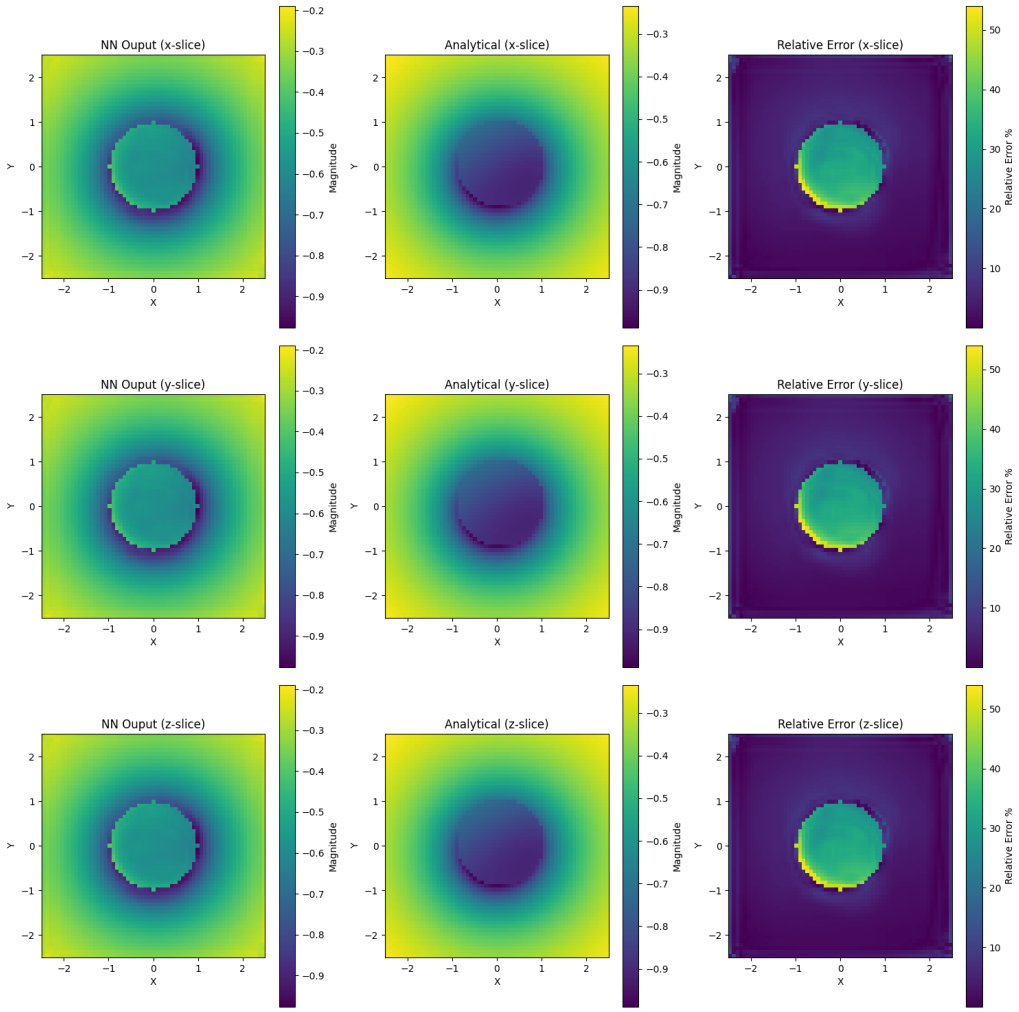


Figura J.2: Resultados del Caso 2 Interfaces 3D

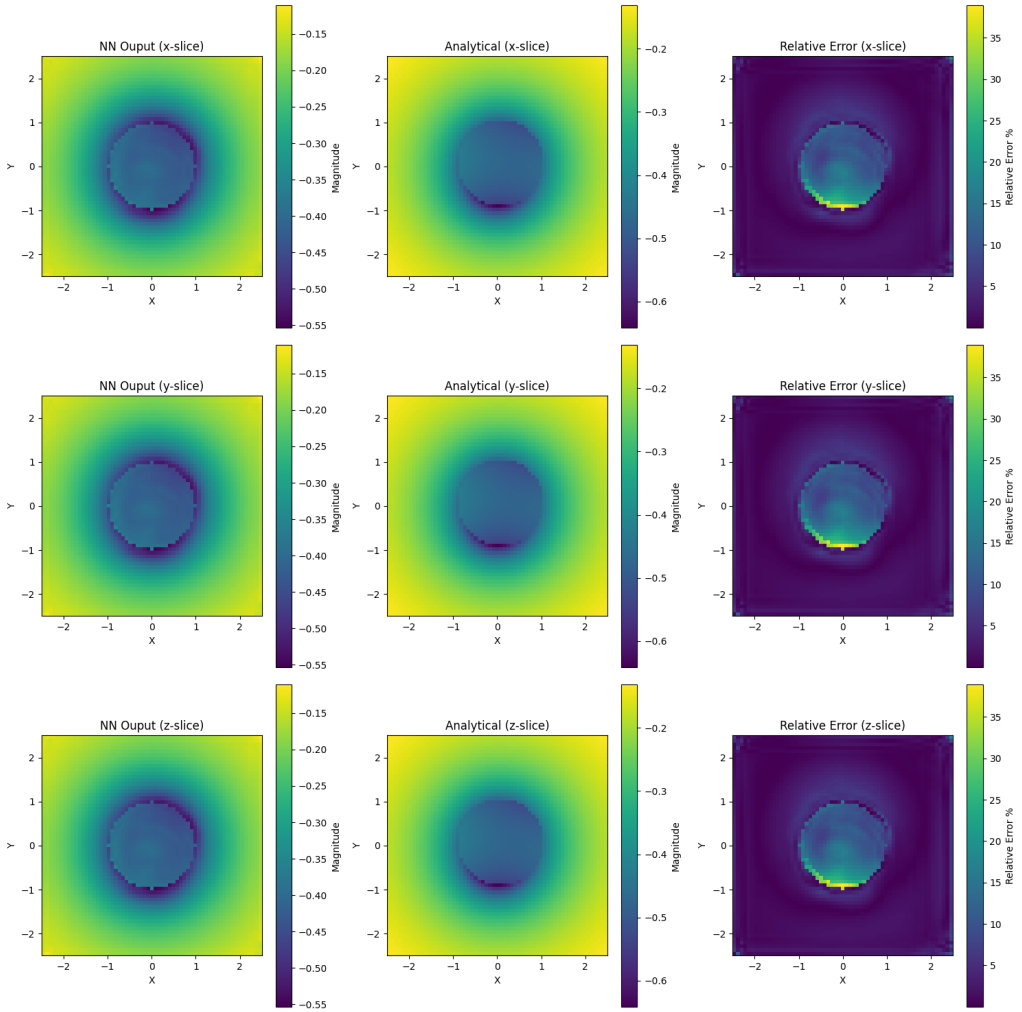


Figura J.3: Resultados del Caso 3 Interfaces 3D

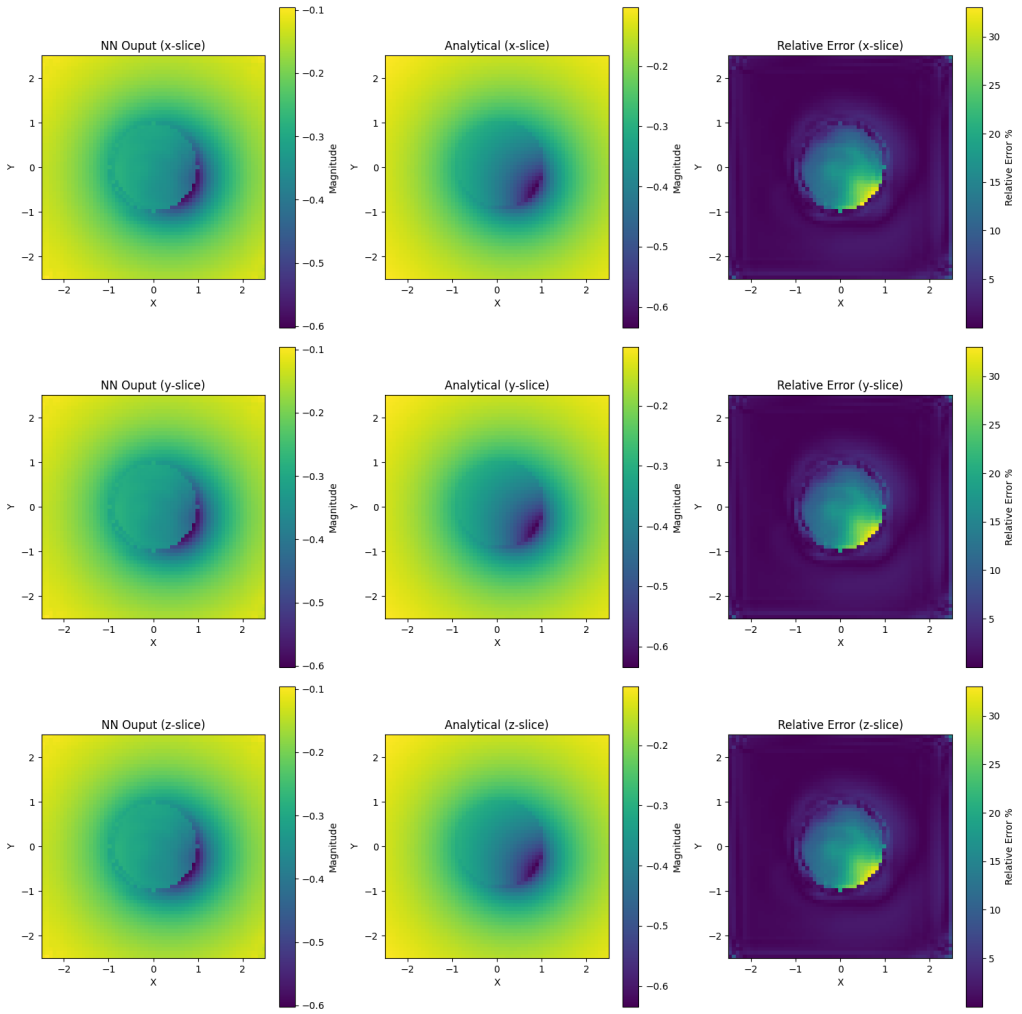


Figura J.4: Resultados del Caso 4 Interfaces 3D

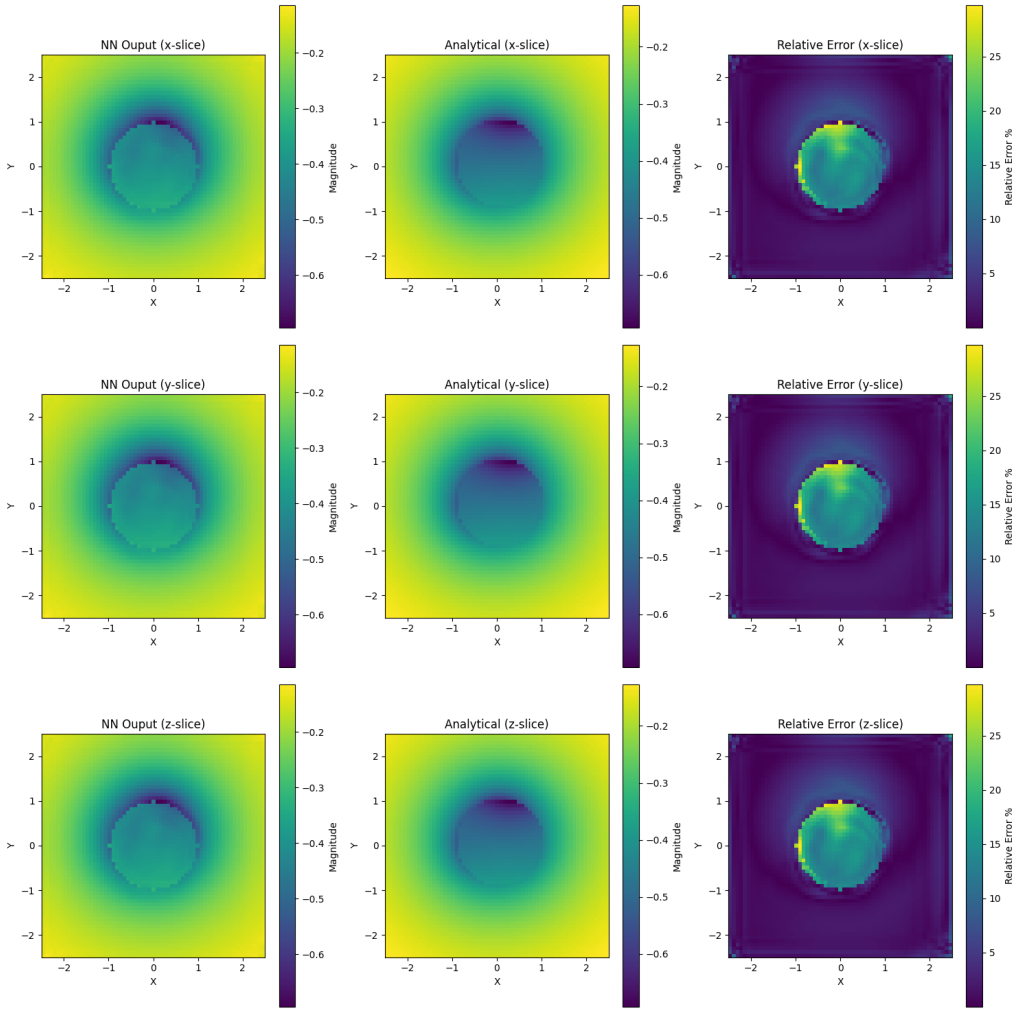


Figura J.5: Resultados del Caso 5 Interfaces 3D