COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

INVESTIGACIÓN DE LA IMPLEMENTACIÓN DE REDES NEURONALES EN LA OPTIMIZACIÓN TOPOLÓGICA EN 2-D

Autor: María Plata Martínez

Director: Thomas Mair

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Investigation of 2D Topology Optimization Using Neural Networks

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2022/2023. es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: María Plata Martínez, Fecha: 15/ 06/2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Thomas Mair Fecha: 21/06 /2023

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

INVESTIGACIÓN DE LA IMPLEMENTACIÓN DE REDES NEURONALES EN LA OPTIMIZACIÓN TOPOLÓGICA EN 2-D

Autor: María Plata Martínez

Director: Thomas Mair

Madrid

Acknowledgements

This master's thesis was done in cooperation with the Technical University of Munich (TUM) and the Pontifical University of Comillas (ICAI) in the field of Mechanical Engineering and Artificial Intelligence. My passion for AI made me choose this subject even though my knowledge in programming neural networks started from scratch. I would like to express my deepest gratitude to all those who have supported me throughout my journey in completing this master's thesis.

First, I would like to give my sincere thanks to Prof. Dr.-Ing. Michael F. Zäh, who made this work possible for me at the Institute of Machine Tools and Manufacturing Technology (iwb) at TUM. Moreover, I would like to thank Thomas Mair for guiding my journey and supervising my research. The constructive feedback helped me achieve my goals for this thesis. I would like also to thank Prof. Dr.-Ing. José Ignacio Linares from ICAI for providing a conducive learning environment.

I would like to thank min alldeles egna Henrik, not only for the constant motivation and support, but also for trying to follow every step of this thesis. Of course, I would like to thank as well my family for the constant support during my entire university phase.

Finally, I would like to thank as well my friends from both, Spain and Germany. Even though some of you are very far from Munich, you all made this phase unforgettable.

Resumen

Abstract

La Optimización Topológica (Topology Optimization, TO) es un método empleado para optimizar la masa de una estructura sin alterar su comportamiento final. Sin embargo el elevado uso de memoria al implementar las funciones objetivo y las restricciones definidas en el método de elementos finitos (Finite Element Method, FEM) ralentiza la obtención de resultados. Hoy en día, la implementación de la Inteligencia Artificial (IA) está reduciendo la carga de trabajo. Por este motivo, se está utilizando para acelerar el tiempo computacional en la mayoría de los campos de estudio.

El objetivo de este trabajo de fin de máster es desarrollar un método con la IA para reducir el tiempo en el cual una estructura es optimizada, comparada con el método tradicional de (FEM). En primer lugar, se programa un código para converger todas las simulaciones TO en Hyperworks. Esto proporciona el banco de datos para entrenar las redes neuronales. En segundo lugar, una Red Adversarial Generativa (Generative Adversarial Network, GAN) para generar figuras de estructuras optimizadas se entrena y se valida. Finalmente, dos casos diferentes de Redes Neural Convolucionales (onvolutional Neural Network,CNN) son programados para evaluar la posición de la carga en la estructura optimizada. La convergencia de la red neuronal es estudiada y validada.

Los resultados de la generación de imágenes después del entrenamiento de GAN y CNN mostraron que el costo computacional podría reducirse considerablemente. En este trabajo de fin de master, las imágenes de estructuras optimizadas pueden generarse 17 000 veces más rápido. Sin embargo, la calidad de las imágenes generadas se ve ligeramente afectada. Los posibles ajustes para mejorar aún más los métodos y aumentar la convergencia de las redes neuronales se proporcionan en la tesis para futuras investigaciones.

PALABRAS CLAVE: Optimizacion topológica, Inteligencia Artificial, Red Adversarial Generativa y Redes Neural Convolucional.

Introducción

La capacidad de un ordenador para reconocer una imagen, clasificar un objeto o analizar un patrón, es posible hoy en día con la inteligencia artificial (IA). Si bien los humanos son la forma más segura de evaluar un entorno, las máquinas están desarrollando cálculos de alto rendimiento para analizar el alrededor con un bajo margen de error. A diferencia de las mentes humanas, un ordenador puede combinar múltiples operaciones y resolver sistemas complejos de ecuaciones continuamente en pocos milisegundos por operación.

La optimización de topología (TO) se utiliza, dentro del método de elementos finitos (FEM), para obtener optimizar estructuras. Debido a los algoritmos programados para optimizar el diseño del material, el tiempo de simulacion es elevado. Sin embargo, este puede ser reducido al implementar el uso de IA.

Objectives

Explotando la capacidad multitarea de los ordenadores y el potencial de la IA, este trabajo de fin de master fusiona los temas de la IA y TO. Este trabajo desarrolla un nuevo método para generar automáticamente estructuras optimizadas sujetas a diferentes condiciones de contorno. El objetivo de esta tesis de maestría fue optimizar la generacion de figuras con TO mediante la implementación de redes neuronales. Primero, se calculó una base de datos de figuras TO, de vigas empotradas sujetas a diferentes condiciones de contorno. Después, se entrenó y validó una Red Adversaria Generativa (GAN) para generar automáticamente el banco de datos. Finalmente, una Red Neural Convolucional (CNN) determinó la posición de la carga, aplicada en la viga en voladizo, analizando su forma. La estructura de la tesis se puede observar en la figura 1.

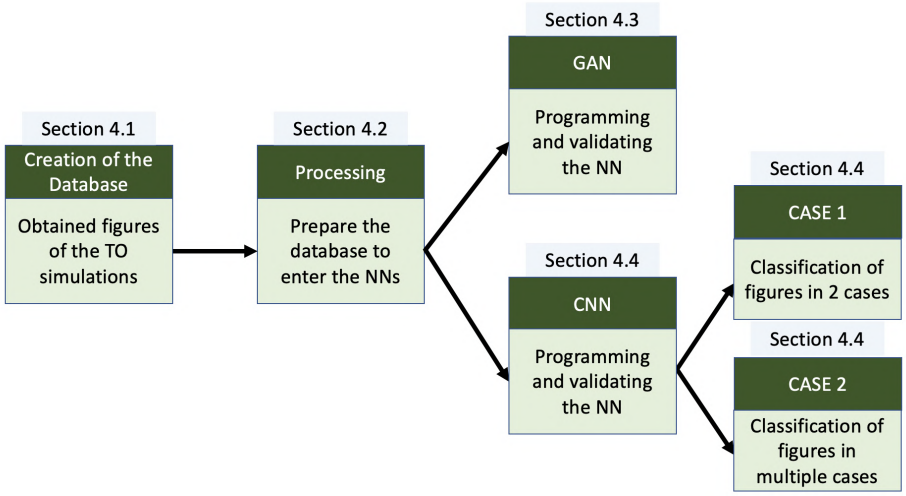


Figure 1: Plan ejecutado para la implementación del concepto junto con la metodología

Teoría y Metodología

Optimización topologica

TO es un método que tiene como objetivo optimizar la asignación de material en una estructura para minimizar la masa. Este método asegura, sin modificar el comportamiento final, maximizar las propiedades internas, como la rigidez. Esto se consigue aplicando el método SIMP, también llamado método de densidad. Este, asigna la distribución de densidad en el espacio diseñado entre 0, o vacío, y 1, o sólido. Los valores dentro de este rango pueden considerarse como material con cavidades homogéneas como se menciona en el artículo de ROZVANY (2009).

En esta sección, se programaron dos scripts TCL en Hyperworks y Hypermesh (ALTAIR ENGINEERING, INC. 2017). El primer código, automatizó el proceso de construcción del modelo y la ejecución de simulaciones. El segundo, procesó las capturas de pantalla necesarias para la creación del banco de datos. El modelo se puede observar en la figura 2.

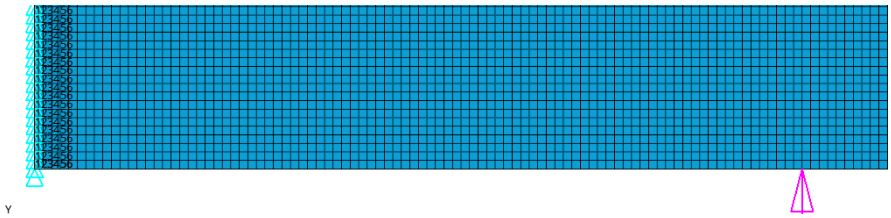


Figure 2: Ejemplo del modelo de la viga en voladizo con la carga aplicada en $x = 450, y = 0$

Redes neuronales

Las redes neuronales artificiales sirve para mapear una relación altamente no lineal entre variables y objetivos aprendiendo a través de datos proporcionados para ser entrenadas (WEIDMAN 2019). Las redes neuronales están compuestos por pesos, sesgos y funciones de activación. Los pesos o w_x , al ser x un número natural (**N**), varía el valor de entrada multiplicándolo, el sesgo o b_x lo modifica mediante la suma de un valor y por último, las funciones de activación, lo cambian aplicando una función. Un ejemplo de estructura de una redes neuronales se proporciona en la figura 3.

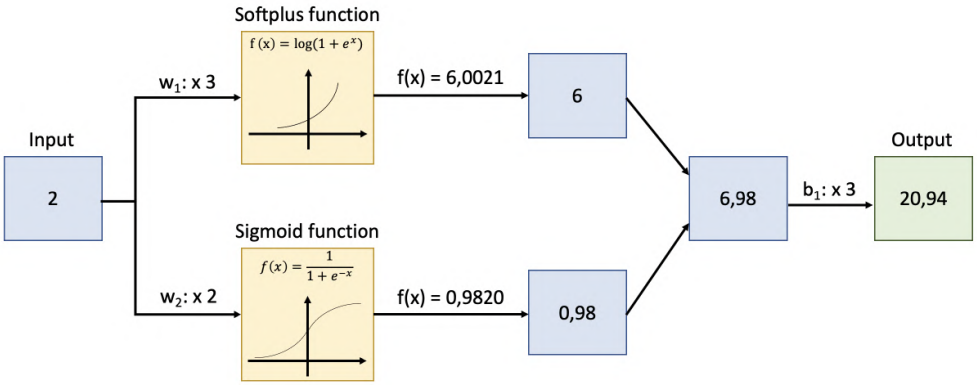
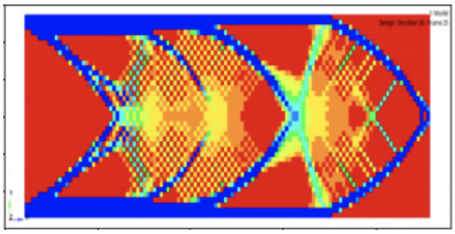


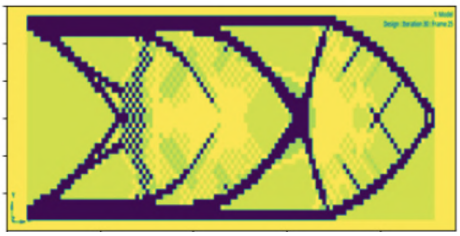
Figure 3: Ejemplo completo de una red neuronal (basada en WEIDMAN (2019))

Transformando una Imagen en un Tensor

Una figura, está compuesta por píxeles, y cada píxel está compuesto por un vector de información de tres canales que determina los valores de intensidad de rojo, verde y azul, de cero a 255. Esta cantidad de información es elevada al ser procesada por una red neuronal. Por lo tanto, los vectores de tres canales, de las figuras TO, se combinaron convirtiéndose en una imagen de un canal para preservar todos los detalles. La figura 4 muestra la diferencia entre la imagen obtenida en Hyperworks, y la que se ingresó en la NN.



(a) Imagen con 3 canales. Fuerza posicionada en x:500 y:0



(b) Imagen con 1 canal. Fuerza posicionada en x:500 y:0

Figure 4: Ejemplo de procesamiento de una imagen de 3 canales en una imagen de un canal

Red adversaria generativa, GAN

La Red adversaria generativa o GAN es un modelo generativo presentado por GOODFELLOW et al. (2014). Esta se divide en un generador y un discriminador. El discriminador es una máquina binaria que trata de identificar si la imagen obtenida es real, es decir, del conjunto de datos proporcionado o falsa, es decir, creada en el generador. El generador se encarga de crear una nueva imagen, lo más parecida posible, a la del dataset real para engañar al discriminador. La estructura de la red se muestra en la figura 5

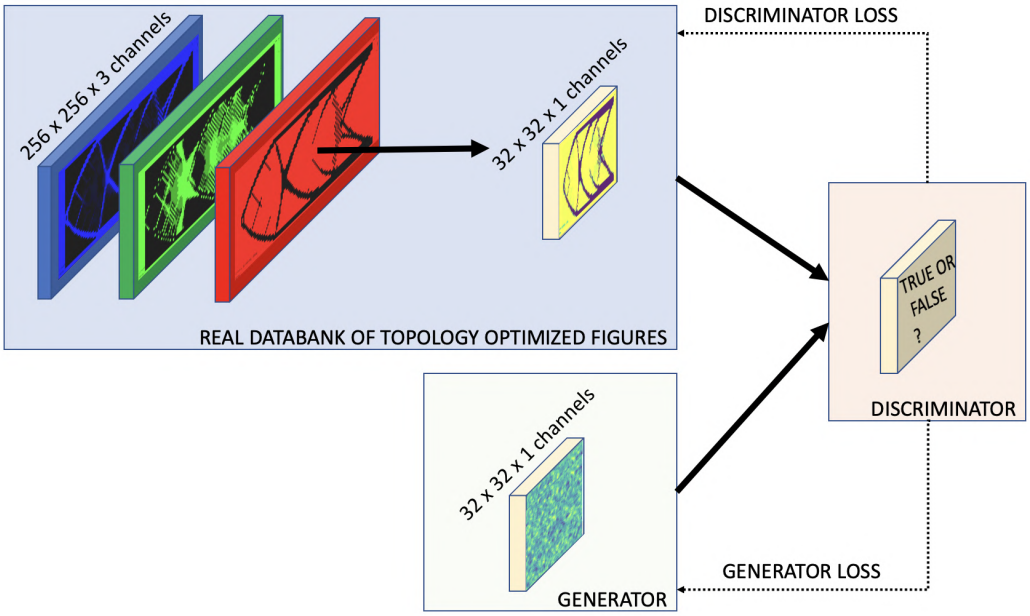


Figure 5: Scheme of the GAN programmed

La GAN programada en este trabajo de fin de máster empleó el banco de datos, de las imágenes TO, para ser entrenado. Tanto el generador como el discriminador fueron entrenados por separado. El discriminador utilizó cuatro bloques de capas para convertir una imagen, es decir, una matriz de números, en un parámetro booleano. El generador, por su parte, implementó dos bloques diferentes para transformar un vector aleatorio de números en una nueva figura. Estos vectores aleatorios, cuando se trazan como una figura, se conocen como ruido, y dentro de las iteraciones, el generador aprende patrones para modificarlo y, en consecuencia, crear una figura (RASHID and LANGENAU 2020).

Red neuronal convolucional, CNN

La Red neuronal convolucional, o CNN, es capaz de reconocer diferentes patrones para distinguir diferentes imágenes. Esta red neuronal es capaz de transformar una imagen en una matriz de números, y sucesivamente en un vector, para procesar la información y posteriormente, clasificarla. La estructura de las múltiples CNN programadas en este proyecto se proporciona en la figura 6 (SEWAK et al. 2018).

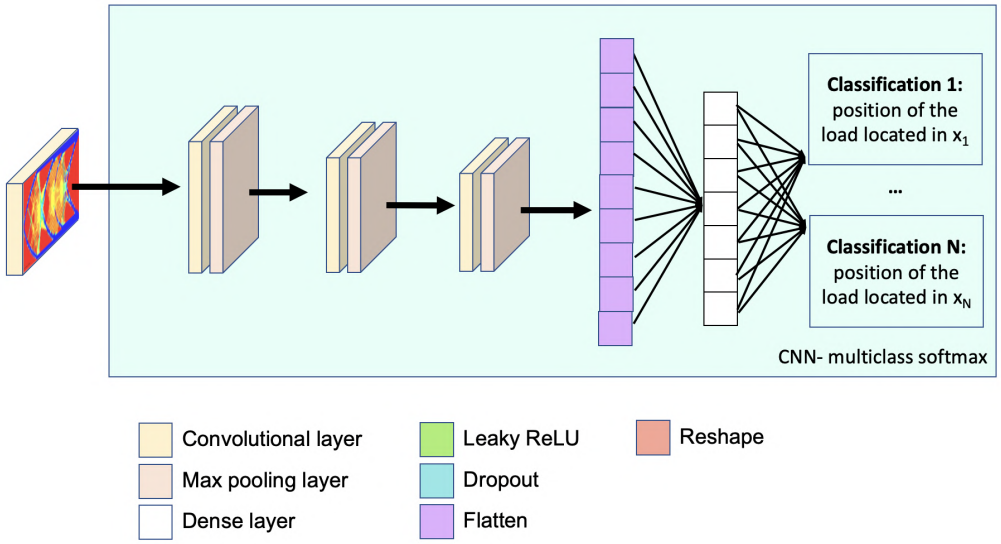


Figure 6: Estructura interna de las capas de la CNN con 3 clasificaciones

Las estructuras principales de una CNN son una capa convolucional, una capa de agrupación y, por último, una capa totalmente conectada. Estas capas se encargan de simplificar la complejidad de la figura para obtener el resultado deseado. Un ejemplo, de la simplificación de una imagen se puede observar en la figura 7. Para que la red neuronal la procese, un filtro, recorre esta calculando el producto escalar entre el filtro y los píxeles de la figura (SEWAK et al. 2018). Al implementar este método, las figuras fueron analizadas para ser clasificadas en múltiples categorías.

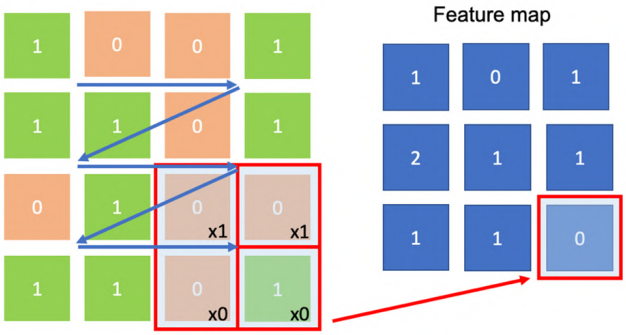
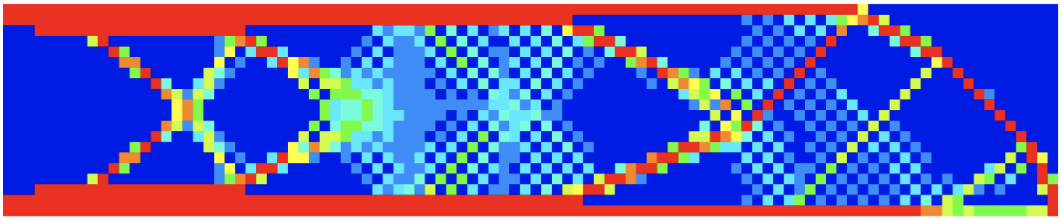


Figure 7: Desplazamiento del filtro para crear el mapa destacado (basado en SEWAK et al. (2018))

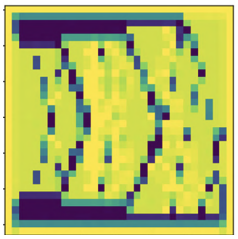
Resultados

Creación y proceso del banco de datos

Una base de datos de 1911 simulaciones fue creada con Hyperworks para entrenar tanto la GAN, como la CNN. Cada uno de los resultados de TO convergió después de 30 iteraciones cumpliendo las funciones objetivo y las restricciones. Un ejemplo de las figuras del banco de datos creadas para cada una de las secciones se proporciona en la figura 8.



(a) Simulación TO con la carga aplicada en x:500 y:0



(b) Imagen 8a adaptada para entrar en la GAN

(c) Imagen 8a adaptada para entrar en la CNN

Figure 8: Ejemplo del banco de datos utilizado en este trabajo de fin de master

Las redes neuronales necesitan procesar las imágenes en formato cuadrado. Debido a la complejidad del GAN, el tamaño de las figuras asignado, se basó en el conjunto de datos del MNIST. De esta manera, el tamaño final de la imagen se redujo de $100 \times 500 \times 3$ a un tamaño de $32 \times 32 \times 1$ píxeles por imagen. La CNN fue entrenada con imágenes de mayor calidad, $224 \times 224 \times 1$, para poder aprender más patrones y, por lo tanto, realizar una mejor clasificación. Como se puede apreciar el diseño inicial es demasiado rectangular, y por ello, las calidad de las imagenes, procesada por las redes neuronales, se ve reducida.

Resultados de la Generación de Imágenes de la GAN

Despues de programar la primera GAN, se variaron los parámetros hasta lograr la convergencia del modelo. Las modificados necesarias fueron las tasas de aprendizaje, tanto del generador como del discriminador, y el algoritmo de optimización. Una vez compilado el modelo para 11 000 epochs, el error se estabilizó obteniendo un valor entre 0,4 y 0,6 tanto para el generador como para el discriminador, siendo el generador ligeramente inferior al discriminador como se puede observar en la figura 9.

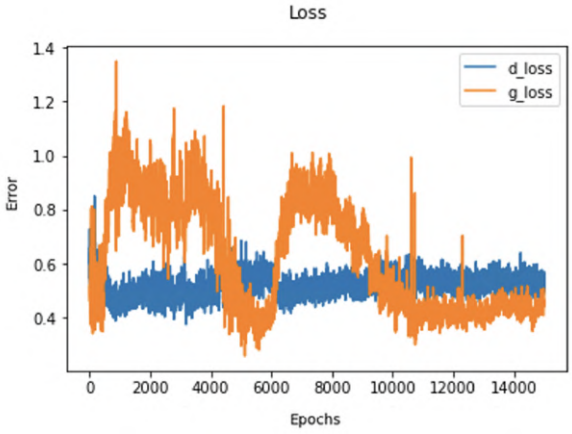


Figure 9: Convergencia final de funciones de pérdida para el generador y el discriminador

Las imagenes finales obtenidas se pueden observar en la figura 22. El tiempo necesario para generar una figura con este GAN fue de 35 ms. El tiempo computacional utilizado para ejecutar una simulación TO en Hyperworks fue de diez minutos. Esto significa que las figuras pudieron generarse 17 000 veces más rápido en la GAN que cuando se computaron en la metodología SIMP convencional. Además, se puede concluir que el banco de datos proporcionado se barajó y proporcionó correctamente a la GAN. Esto se puede asumir ya que la generación de imágenes es completamente aleatoria y todas ellas son diferentes. La menor calidad de algunas de las imagenes producidas por la GAN podría deberse al error en la convergencia del modelo en la figura 9.

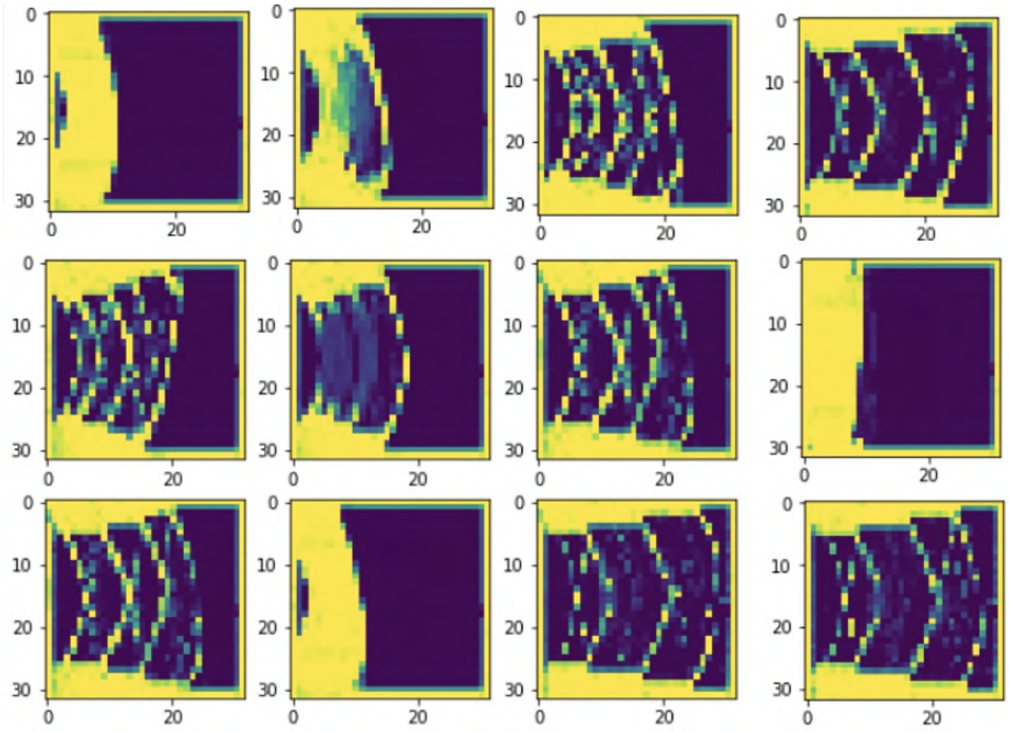


Figure 10: Ejemplos de cifras generadas por la GAN.

Resultados de la Clasificación del Banco de Datos de la CNN

La CNN programada para ordenar las cifras en dos categorías diferentes funcionó. El modelo convergió después de tres epochs de 42 pasos cada una, como se puede observar en la figura 11.

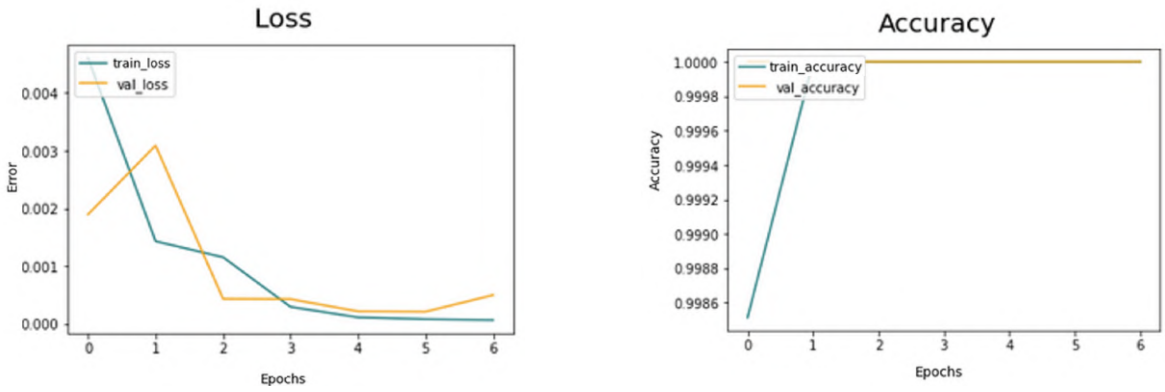


Figure 11: Convergencia obtenida clasificando 2 tipos de figuras con la CNN

La convergencia de la CNN en tres tipos de imágenes mostró que, mientras que la pérdida como la precisión del conjunto de entrenamiento convergían, la validación no lo hacía. Por lo tanto, se realizaron las siguientes implementaciones. Primero, el model se simplificó eliminando bloques de arquitectura. En segundo lugar, se programaron las técnicas de regularización para modificar el banco de datos. Luego, se redujeron los filtros utilizados en las capas convolucionales. Posteriormente, se minimizó el tamaño de la capa densa. Finalmente se aumentó el tamaño del lote. Todas estas medidas supusieron una mejora de la CNN. Los resultados finales de convergencia se pueden observar en la figura 12.

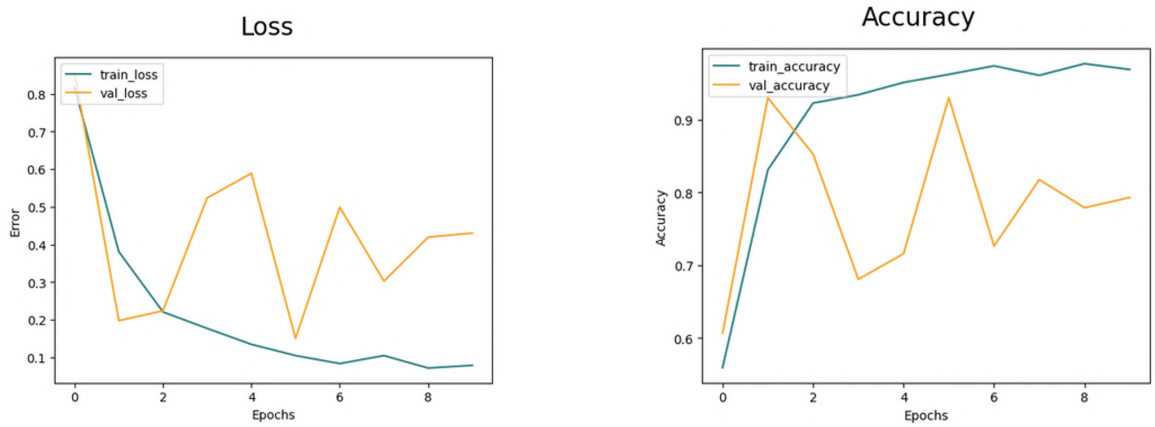


Figure 12: Mejor convergencia obtenida al modificar la CNN utilizada para clasificar en 3 grupos diferentes

Summary

ABSTRACT

Topology Optimization (TO) is a method to optimize the mass of a structure without losing its internal properties or changing its final behaviour. However, the computing time is elevated due to the objective functions and constraints defined in the Finite Element Method (FEM). For this reason, the reduction of computational cost is a matter of interest. The implementation of Artificial Intelligence (AI) is speeding up processes and reducing the workload. As well as computer software has been employed to solve complex systems of equations and run simulations, AI is being used to accelerate the software's computational time in most fields of study. For this reason, this master's thesis merges the concepts of TO with AI.

The aim of this master's thesis is to develop a method to generate TO structures with less computational cost with AI, compared to the FEM. There are three main goals addressed. First, a successful code is programmed to converge all the TO simulations in Hyperworks. This provides the needed databank to train the neural networks. Secondly, a Generative Adversarial Network (GAN) is trained and validated to generate TO figures. Finally, two different cases of Convolutional Neural Network (CNN) are programmed to evaluate the position of the load in the optimized figure. The convergence of the neural network is studied and maximized.

The results of the image generation after the training of the GAN and CNN showed that the computational cost could be greatly reduced. In this master's thesis the figures could be generated 17 000 times faster, however with loss of picture quality. Possible adjustments to further improve the methods and increasing convergence are provided in the thesis for future work.

KEY WORDS: Topology Optimization, Artificial Intelligence, neural networks, Generative Adversarial Network, Convolutional Neural Network.

Introduction

The ability for a computer to recognize an image, classify an object or recognize a pattern is possible with artificial intelligence (AI). Even though humans are the most reliable way to evaluate an environment, computers are developing high-performance calculations to analyze the surroundings with a low error margin. Unlike human minds, a computer can combine multiple operations and solve complex systems of equations continuously within a few milliseconds per operation.

For example, Finite Element Method (FEM) is among the most demanding operations a computer can perform. Topology optimization (TO) can be used within FEM to obtain an opti-

mized shape in a simulation but the computational time is considerable. In other words, the computer performs algorithmic models to optimize material layout increasing the time used per simulation. However, this time employed in the TO simulations can be reduced when implementing the use of AI.

Objectives

Exploiting the multi-task capacity of a computer and the potential of AI, this master's thesis merges the topics of AI and TO. A new method is developed to automatically generate optimized structures subjected to different boundary conditions. The aim of this master's thesis is to perform time-efficient TO of a beam model by implementing neural networks. First, a databank of TO figures, of cantilever beams subject to different boundary conditions, is computed to train the neural networks(NN). Then, a Generative Adversarial Network (GAN) is trained and validated to automatically generate the databank. Finally, a Convolutional Neural Network (CNN) determines the position of the load applied in the cantilever beam by analyzing its shape. The structure of the thesis can be observed in the figure 13.

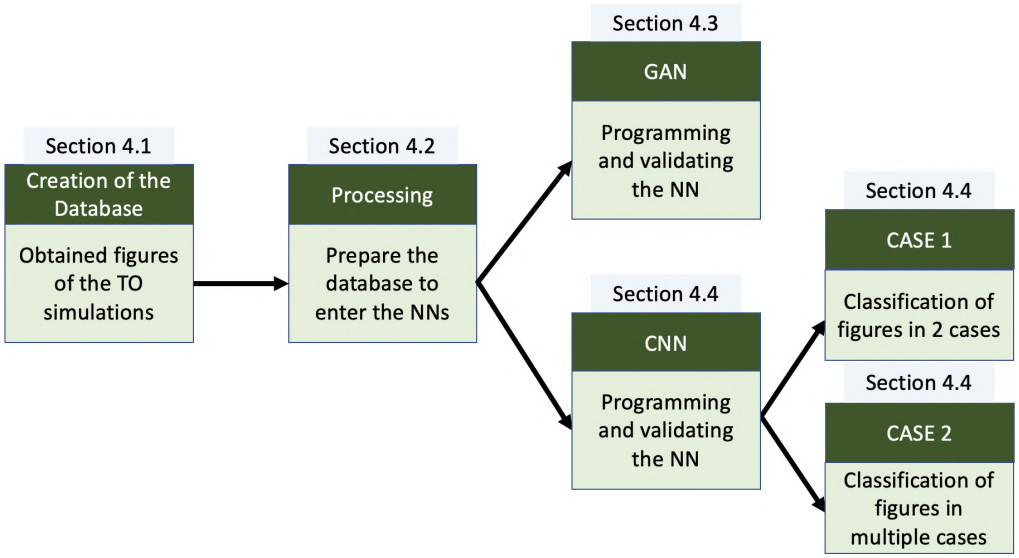


Figure 13: Project plan for the implementation of the concept, together with the methodology to be applied.

Theory and Method

Topology Optimization

TO is a method that aims to optimize material allocation in a structure to achieve minimum mass. This method ensures, without modifying external factors, such as constraints, maximizing internal properties, such as stiffness, so that the final behavior of the structure remains the same. TO enables the design to achieve minimal mass and maximum stiffness using the Solid Isotropic Material with Penalization Method (SIMP) (SIVA RAMA KRISHNA et al. 2017). The SIMP method, also called the density method, identifies the density distribution in the

designed space between 0, or void, and 1, or solid. The values inside this range can be considered as distributed material with homogeneous cavities as mentioned in the paper from ROZVANY (2009).

In this section, two TCL scripts are programmed in Hyperworks and Hypermesh (ALTAIR ENGINEERING, INC. 2017). The first code automatizes the process of building the model and running simulations. The second, captures the needed screenshots for the databank. The model can be observed in the figure 14.

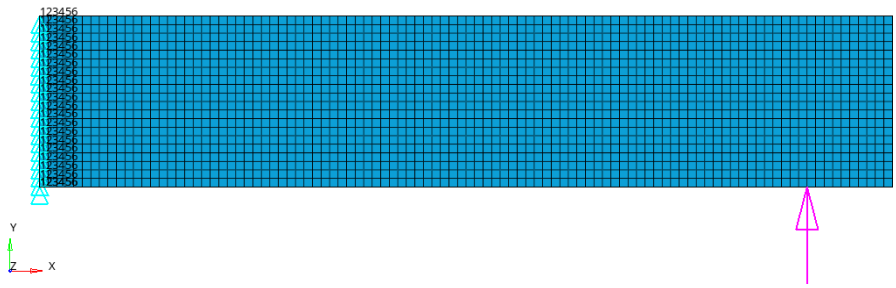


Figure 14: Example of the simulation with the fixed conditions on the left side of the cantilever beam and the load applied at $x = 450, y = 0$

Neural Networks

The NN are able to map a highly non-linear relationship between the variables and the objectives by learning through the provided data. The NN are composed by weights, bias and activation functions. The weights or w_x , being x a natural number ($\mathbf{N}$), varies the input by multiplying it. The bias or b_x varies the input with the addition of a value. Lastly, the activation functions varies it by applying a function. An example of structure of a NN is provided in the figure 15 (WEIDMAN 2019).

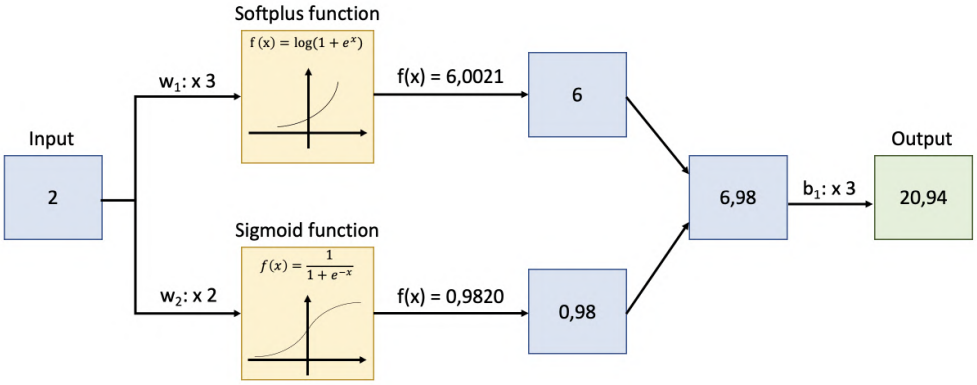


Figure 15: Complete example of a neural network (based on WEIDMAN (2019))

Transforming a picture into a Tensor

One figure, is composed by pixels, and each pixel is composed by a three channel information vector that determines the red, green and blue intensity values, from zero to 255. This

quantity of information is high for a NN to process it. Therefore, the three channel vectors, from the TO figures, were combined turning into a one channel picture to preserve all the details. The figure 16 shows the difference between the image obtain in Hyperworks, and the one to enter in the NN.

(a) 3 Channels picture. Load in x:500 y:0

(b) 1 Channel picture. Load in x:500 y:0

Figure 16: Example of processing a 3 channel picture into a one channel picture

Generative Adversarial Network, GAN

GAN is a generative model introduced by GOODFELLOW et al. (2014). This NN is divided into a generator and a discriminator. The discriminator is a binary machine that tries to identify whether the obtained image is real, i.e. from the real dataset or fake i.e. created in the generator. The generator is responsible for creating a new image, as similar as possible, to the one in the real dataset to fool the discriminator. The structure of the network is shown in the figure 17 (RASHID and LANGENAU 2020).

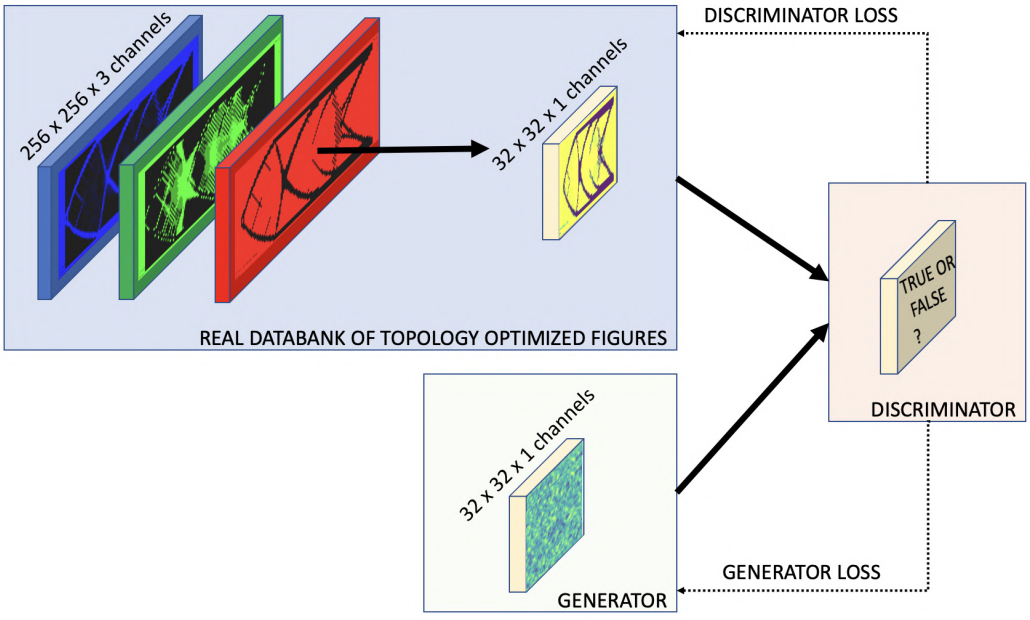


Figure 17: Scheme of the GAN programmed

The GAN programmed in this master's thesis employs the databank, from the TO pictures, to be trained. Both, the generator and the discriminator were trained separately. The discriminator used fours blocks of layers to turn an image, i.e., a matrix of numbers, into a boolean parameter. The generator implemented two different blocks to transform a random

vector of numbers into a new figure. This random vectors, when plotted as a figure, are known as noise, and withing the iterations, the generator learns patterns to modify it, and consequently, create a figure (RASHID and LANGENAU 2020).

Convolutional Neural Network

The CNN is capable to recognize different patterns and relationships performing great attainments in image recognition. This NN is be able to transform an image into a matrix of numbers, and successively into a vector, to process the information. The structure of the multiple CNN programmed in this project is provided in the figure 18 (SEWAK et al. 2018).

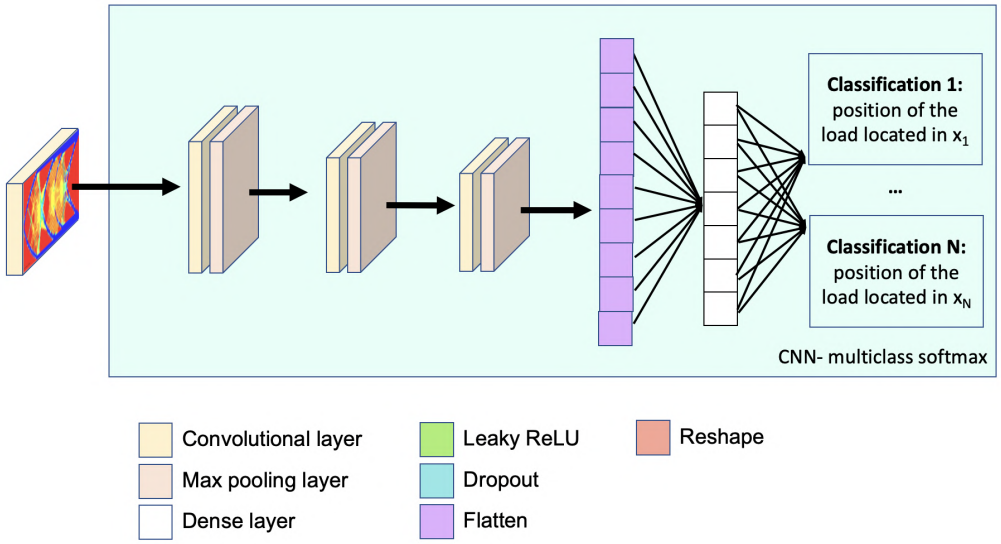


Figure 18: Internal structure of the layers of the CNN with 3 classification

The main structures of a CNN are a convolutional layer, a pooling layer and lastly a fully connected layer. This layers are in charge of simplifying the complexity of the figure to obtain the desired output. An example, of the simplification of the figure can be observed in the figure 2.13. In order for the NN to process an image, a filter, goes along an image calculating the dot product with the pixels of the figure (SEWAK et al. 2018). By implementing this method, the figures were analyzed to be classified into multiple categories.

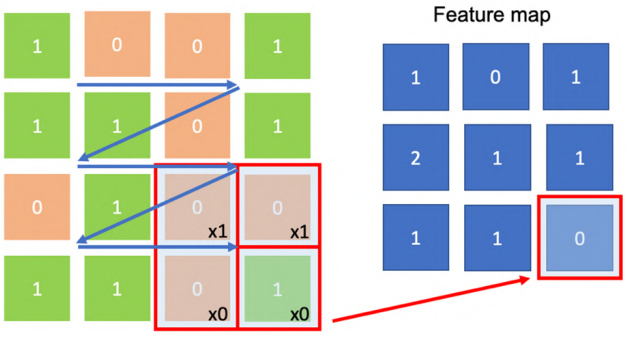
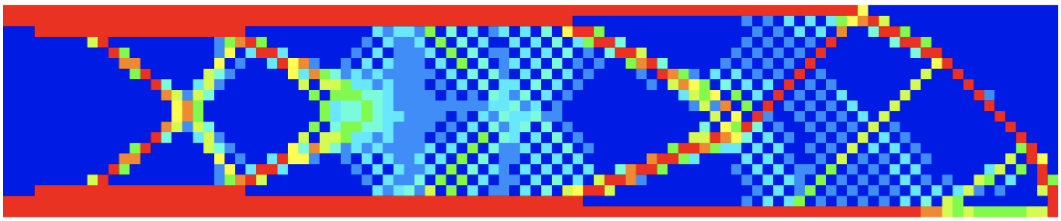


Figure 19: Displacement of the filter to create the featured map (based on SEWAK et al. (2018))

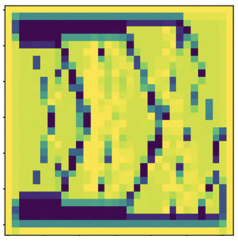
Results

Creating and processing the databank

A database of 1911 simulations was created to train both, the GAN and the CNN. Each of the TO figures converged after 30 iterations achieving the objective functions and the constraints established. An example of the databank figures created for each of the sections is provided in the figure 20.



(a) TO simulation with the load applied in x:500 y:0



(b) Figure 20a adapted for GAN

(c) Figure 20a adapted for CNN

Figure 20: Example of the databank used in this master thesis

As it can be observed, due to the complexity of the GAN, the size of the figures was based in the MNIST dataset. By implementing that, the final size of the picture was reduced from $100 \times 500 \times 3$ to a size of $32 \times 32 \times 1$ pixel picture. The CNN was trained with higher quality pictures to be able to learn more patterns, and therefore, perform a better classification. For this reason, the pictures on the left column from the figure 5.5, were resized into a quadratic shape with a size of $224 \times 224 \times 1$ pixels to train the CNN.

The structure of the cantilever beam is characterized by its substantial length in comparison to its width. When analyzing the dataset employed for training both, the GAN and the CNN, it was observed that the reshaping of images resulted in a significant degradation of the information. Therefore, a different option, such as changing the model, or dividing the beam into multiple squares, could be implemented to improve this databank.

Results of the GAN's Image Generation

After programming the GAN, the parameters were varied until achieving the convergence. The parameters modified were the learning rates, of both the generator and the discriminator, and the optimization algorithm. Once the model compiled for 11 000 epochs, it stabilized

obtaining an error between 0.4 and 0.6 for both the generator and the discriminator,being the generator slightly lower than the discriminator as it can be observed in the figure 21.

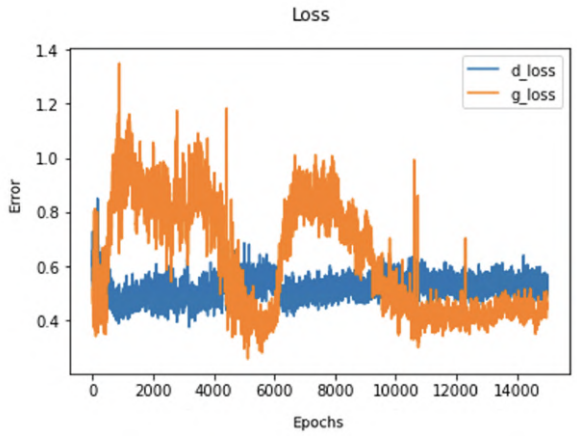


Figure 21: Final convergence of loss functions for the generator and discriminator

The final obtained figures can be observed in the figure 22. The time needed to generate a figure with this GAN was 35 ms. The computational time used to run a TO simulation in Hyperworks was ten minutes. This means that the figures were able to be generated 17 000 times faster in the GAN than when computing it in the conventional SIMP methodology. Moreover, the provided databank was correctly shuffled and provided to the GAN. This can be assumed as the generation of images is completely random and all of them are different. The lower quality of some of the GAN's output figures could be caused by the error in the model convergence in figure 21.

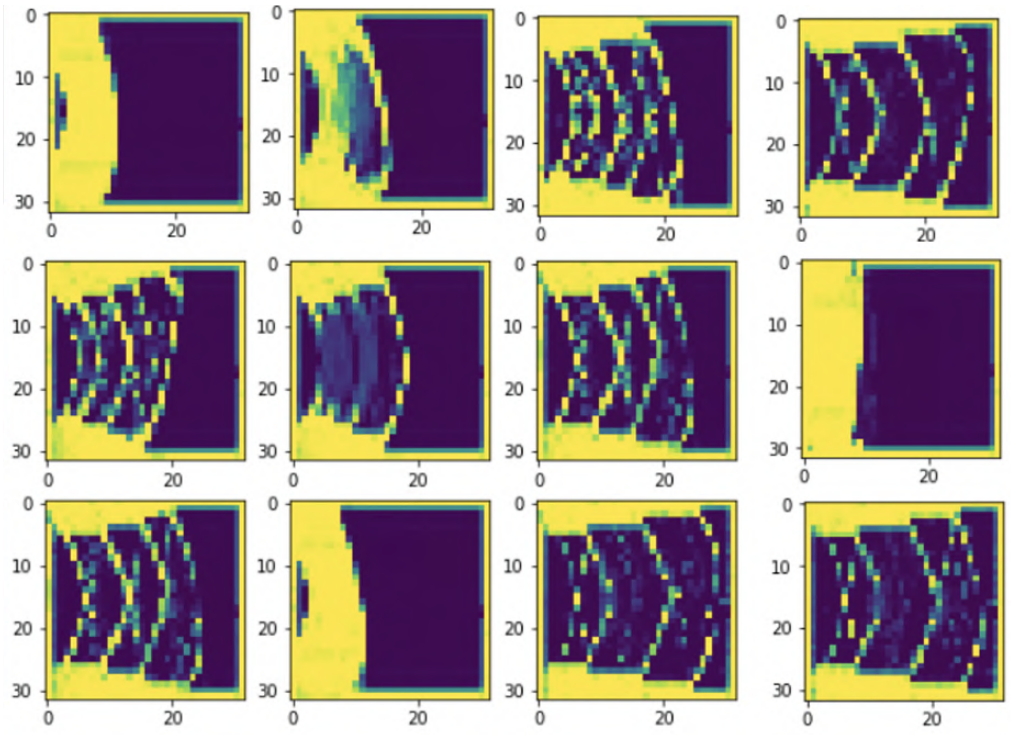


Figure 22: Examples of generated figures by the GAN.

Results of the CNN's Classification of the Databank

The CNN programmed to sort the figures in two different categories worked. The model converged after three epochs of 42 steps each as it can be observed in figure 23

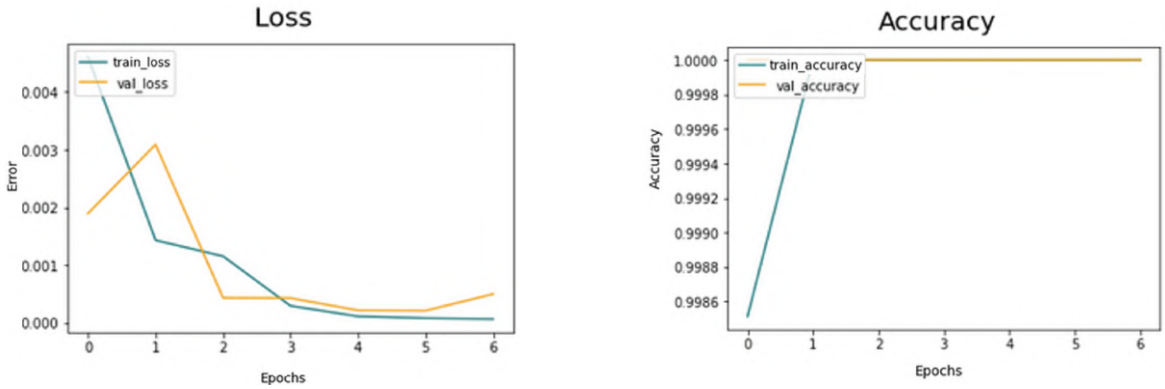


Figure 23: Convergence obtained classifying 2 clases with the CNN

The convergence of the CNN for three categories showed that while both the loss and the accuracy from the training set were converging, the validation was not. Therefore, the following implementations were performed. First the architecture was simplified by deleting architecture blocks. Second, the regularization techniques to modify the databank were programmed. Then, the filters used in the convolutional layers were reduced. Afterwards, the size of the dense layer was minimized. Finally the size of the batch was increased. All these measures performed an improvement of the model. The final convergence results can be observed in the figure 24

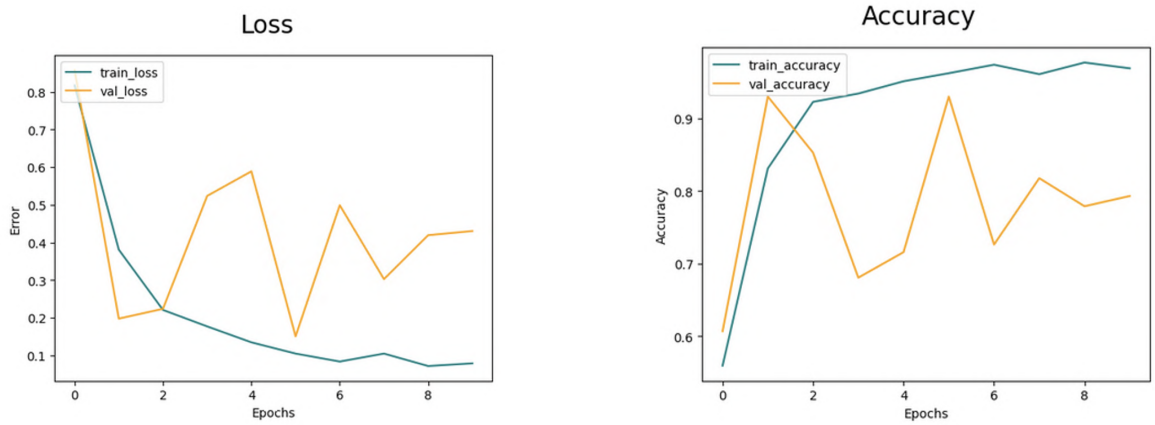


Figure 24: Best convergence when modifying the CNN used to classify into 3 different groups

Contents

List of Formula Symbols

Latin letters

Symbol	Unit	Description
b	[–]	Bias from Neural Network
C	[Nm]	Compliance or Strain Energy
e	[–]	Euler Number
f	[–]	Desired Volume fraction
F	[N]	External Force
K	[N/m]	Stiffness matrix
k_o	[N/m]	Element Stiffness
N	[–]	Number of elements to be discretized
p	[–]	Penalization Power
U	[m]	Global Displacement
u_e	[m]	Element Displacement
V	[m^3]	Total Material Volume
V^*	[m^3]	Desired Volume Constraint
v_e	[m^3]	Elemental volume
V_o	[m^3]	Design domain Volume
X	[–]	Vector of relative Densities
x	[–]	Vector of elemental relative densities
x_{min}	[–]	Vector of minimum relative Densities
x_e	[–]	Eth design variables
w	[–]	Weight from Neural Network
z	[–]	Output of the Neural Network

Greek letters

Symbol	Unit	Description
ρ	[kg/m^3]	Density
σ	[–]	Sigmoid function

List of Abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
CGAN	Conditional Generative Adversarial Network
CNN	Convolutional Neural Network
Conv2D	Convolutional layer in two dimensions
CWGAN	Conditional Wasserstein Generative Adversarial Network
DL	Deep Learning
DNN	Deep Neural Network
FEM	Finite Element Method
GAN	Generative Adversarial Network
iwb	Institute for Machine Tools and Industrial Management
MNIST	Modified National Institute of Standards and Technology
NN	Neural Network
PBF-LB/M	Powder Bed fusion of metal using a laser beam
RMSprop	Root Mean Squared Propagation
SIMP	Solid Isotropic Material with Penalization Method
TO	Topology Optimization
WGAN	Wasserstein Generative Adversarial Network

List of Figures

List of Tables

Chapter 1

Introduction

Artificial Intelligence (AI) implementation has rapidly increased in recent years and is being introduced in most fields of study. One of the latest applications of AI is ChatGPT, which is an AI designed to generate automatic answers to all conceivable questions (OPENAI 2022). These automatic responses are quickly generated after analyzing a database of information. AI can be used for image analysis in an efficient matter and this thesis will analyze if this could be implemented in structural analysis.

Modern construction simulation software is allowing engineers use computers to perform comprehensive structural analyses. Topology Optimization (TO) is an approach where the material used in structural elements is minimized without modifying its final behaviour. Although TO is an efficient structural optimization tool, it still requires a high computational power, reaching computational time of hours or even days. AI Programming can be employed to generate and recognize structural elements through image analysis, so that the computation speed can be increased and the computational time decreased.

1.1 Motivation

At the Institute for Machine Tools and Industrial Management (*iwb*)at the Technical University of Munich, Powder Bed fusion of metal using a laser beam PBF-LB/M (DIN EN ISO/ASTM 52900) is used to create structures for experiments. Material consumption is closely linked to the cost of these projects and its reduction is necessary for economic, environmental, and sustainable purposes.

In order to optimize the material in a structure, Topology Optimization (TO) can be used within Finite Element Method (FEM) to obtain an optimized shape in a simulation without changing its final behavior. However, TO is among the most demanding operations a computer can perform due to the algorithmic models computed to optimize the material layout. Nevertheless, this time spent on TO simulations can be reduced by implementing the use of AI. In this sense, combining the learning process of a computer with the methodology from TO, a new computing method can be developed to optimize solutions of a defined structure in a faster method.

This project is linked to the Sustainable Development Goals (UNITED NATIONS and DEVELOPMENT 2015) , which were proposed in 2015 to eradicate poverty, protect the planet and ensure prosperity for all as part of a new sustainable development agenda. The Sustainable Development Goals consist of seventeen targets and are part of the 2030 Agenda for

Sustainable Development. It is essential that these goals are fully achieved by 2030..

This thesis aims to achieve objective number nine, *Industry innovation and infrastructure* (UNITED NATIONS and DEVELOPMENT 2015) and number twelve, *Responsible, consumption and production* (UNITED NATIONS and DEVELOPMENT 2015). The implementation of TO will reduce the use of unnecessary material in structures, thus achieving sustainable production and establishing sustainable consumption and production practices, which are crucial for preserving the means of subsistence for both present and future generations. In addition, the use of AI will provide the user with solutions that are computed in less time, thus requiring less energy required to achieve the same results. Moreover, the implementation of TO and AI imply an upgrade of the technological capabilities used in the construction sector, encouraging innovation and motivation for the scientific research.

1.2 Aim and Objectives

This master's thesis performs a qualitative study to increase the speed of generation of images. For this reason, modeling AI with neural networks (NN) is implemented to perform three different objectives. First, the database of optimized structures is processed so that the NN can recognize the images faster. Then, the NN is trained, with pictures from the previous step, to learn how to generate them. Finally, testing the NN allows the AI create images faster as with the TO conventional method. All this process is be conducted by the implementation of a Generative Adversarial Network (GAN).

To analyze the obtained images from GAN, another NN is implemented. In this case, with the help of image recognition, this new NN is in charge of processing the new created picture from the GAN and analyze the boundary conditions of the generated structure. This process is conducted by the implementation of a Convolutional Neural Network (CNN).

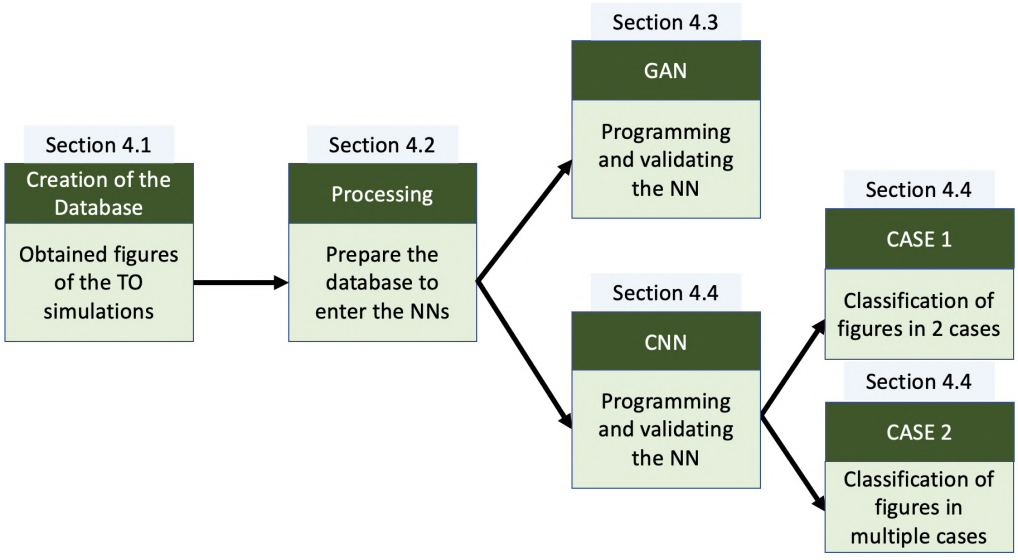


Figure 1.1: Project plan for the implementation of the concept, together with the methodology to be applied.

In order to achieve these purposes, the databank of optimized structures is generated with FEM. For this, two automated scripts are in charge of creating a computing the simulations. The first one sets up the boundary conditions and runs the simulations. The second, analyzes

the results and captures the pictures with the required labels. All the objectives achieved can be observed in figure 1.1

1.3 Structure of the work

Chapter 2 contains the basic principles related with TO and NN. First, the basics of TO are analyzed. Secondly, an overview of the NN is provided for a comprehension of each of the parameters. A summary of the functioning including general formulas and processes are included for a better understanding of the thesis. Chapter 3 contains the state of art and science. An outline of the implementations performed until today is detailed explained. Moreover the research field, needed for this master thesis, is collected.

The methodology is written in Chapter 4. First, the process of obtaining the databank is provided. Here the incorporation of the AI code is shown explaining the performed process until achieving the desired outputs. The analysis of the accuracy and precision is performed until achieving the qualitative goal.

Finally, the Chapter 5 provides the results achieved in this master thesis with its respective discussion, and the Chapter 6 contains the conclusions and the future work to continue this research.

Chapter 2

Basic Principles

2.1 Topology Optimization (TO) Theory

TO is a method that aims to optimize material allocation in a structure to achieve minimum mass. This method ensures, without modifying external factors such as constraints, maximizing internal properties such as stiffness so that the final behavior of the structure remains the same (SIVA RAMA KRISHNA et al. 2017). Parameters such as inner boundaries, outer shapes, and the number of inner cavities are simultaneously optimized with a specific optimization target driving the optimization procedure (RAWAT and SHEN 2019b).

TO enables the design to achieve minimal mass and maximum stiffness using the Solid Isotropic Material with Penalization Method (SIMP) (SIVA RAMA KRISHNA et al. 2017). The SIMP method, also called the density method, identifies the density distribution in the designed space between 0, or void, and 1, or solid. The values inside this range can be considered as distributed material with homogeneous cavities as mentioned in the paper from ROZVANY (2009). This chapter discusses the mathematical formulae and the iterative flowchart employed for creating the databank.

2.1.1 Methodology of Topology Optimization

The mathematics of TO can be expressed with a constrained optimization problem. Each of the finite elements programmed in the Finite Element Method (FEM) simulation, needs to be considered as a design variable in which the density will be optimized. For this reason, the density ρ is selected as the design variable (ZUO and XIE 2015). The function to be minimized is the compliance (or strain energy) and the constraints restricting the problem are the volume fraction and the density distribution. This approach can be expressed as an optimization problem like shown in the equation from 2.1 to 2.4 (ZUO and XIE 2015).

$$\text{minimize } x\text{: } C(x) = F^T U = U^T K U \tag{2.1}$$

$$\text{subjected to: } X = \{x_e\}, x_e = 1 \quad or \quad x_{min} \quad \forall e = 1, ..., \mathbf{N} \tag{2.2}$$

$$KU = F, \tag{2.3}$$

$$V(x) = \sum_X x_e v_e = V^* \tag{2.4}$$

In order to minimize the $C(x)$, compliance or strain energy, the formula of virtual works that relates external forces F with global displacements U is applied. K is the stiffness matrix. Moreover, as constrains, a volume fraction is employed dividing the total material volume V, V^* as the desired volume constraint and v_e as elemental volume. X is defined as vector of the elemental relative densities, x_e is the eth design variables which values are between 1 and zero, for solid and void respectively, x_min is a vector of minimum relative densities (ZUO and XIE 2015).

2.1.2 Solid Isotropic Material with Penalization (SIMP) Method

The SIMP method, also called the density method, identifies the density distribution in the designed space between 0, for void, and 1, for solid. The values inside this range, are considered intermediate densities which can be interpreted as a mesostructured material with holes as covered by ROZVANY (2009). ROZVANY et al. (1992) developed the SIMP method with the aim of achieving better performances when optimizing the density distribution of a structure. To accomplish that, they followed the flowchart represented in the figure below.

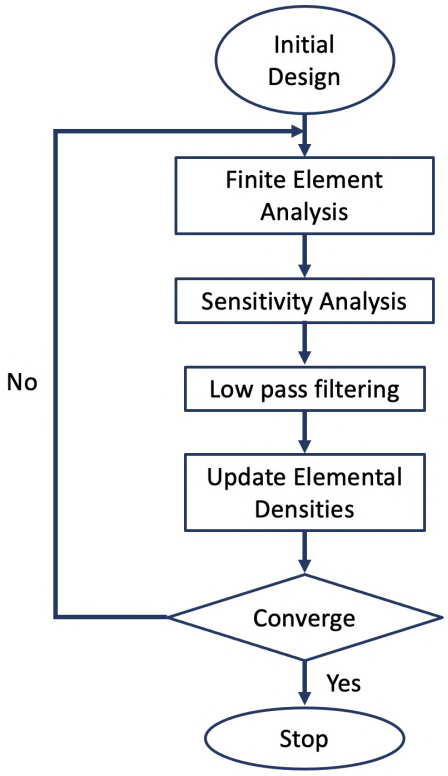


Figure 2.1: SIMP method Flowchart. Source: ROZVANY et al. (1992)

Once the initial design is established, the process is as follows. First, the design domain is meshed to be able to perform the numerical analysis in each of the finite elements. This is known as domain discretization. Second, each of the finite elements is assigned to the corresponding density distribution. Then, one or multiple objective functions are introduced to determine the amount material that should be used in each voxel. Afterwards, each of the densities will get actualized according to the constrains detailed (ROZVANY et al. 1992).

Finally, a structural analysis will be performed to evaluate the behavior of the structure such as stiffness, deformation or distribution of internal stresses. An iteration of the last three

steps will be performed until the structure is optimized (ROZVANY et al. 1992). To represent mathematically the SIMP method, some changes are made regarding the mathematical formulae from 2.1 to 2.4 in TO. The following math equations show the TO when SIMP method is applied regarding to RAWAT and SHEN (2019b).

$$\text{minimize } x\text{: } C(x) = U^T K U = \sum_{e=1}^{N} (x_e)^p u_e k_o u_e \tag{2.5}$$

$$\text{subjected to: } \frac{V(x)}{V_0} = f, \tag{2.6}$$

$$KU = F, \tag{2.7}$$

$$0 < x_{min} \leq x \leq 1 \tag{2.8}$$

As before, the $C(x)$ represents the compliance or strain energy, the F, the external forces and the U, the global displacements U. K is the stiffness matrix. Additionally, as constrains, a desired volume fraction f is employed dividing the total material volume V by the design domain volume V_o. x is defined as vector of the elemental relative densities, x_e is the eth design variables which values are between 1 and zero, for solid and void respectively and $x_m in$ is a vector of minimum relative densities (RAWAT and SHEN 2019b).

In this case, the minimum amount of density distribution is between one and a value $x_{\min}$ so that singularities are avoided. The N is the number of elements of the discretized domain design and p is the penalization power. The values u_e and k_o stands for the element displacement and stiffness (RAWAT and SHEN 2019b).

2.2 Principles of Neural Network Implementation

The name *neural network* (NN) comes from the biological term of real neuron from animal nervous systems. A neuron is the unit of the brain, or the nervous system, in charge of receiving a signal from the external world or another neuron, processing the information, and transmitting a signal to the following neuron or generating an output. The neurons in artificial neural networks are able to map a highly non-linear relationship between the variables and the objectives by learning through the provided data (WEIDMAN 2019).

In this chapter, the structure and the process to program a NN is explained. Moreover a distinction between GAN and the CNN is clarified. With this chapter a better understanding of the AI section of the master thesis is achieved.

2.2.1 Structure of a Neural Network

The main components of a NN are the weights, the bias and the activation functions. Adjusting these parameters will make the NN be able to transform an input to coincide with a desired output after crossing the NN. The weights or w_x, being x a natural number (**N**), are estimated numbers that, combined with mathematical processes, i.e., multiplication, varies the input to obtain an output. Alternatively, the bias varies the result by adding a value to the estimated previous result. The following picture shows an example of a NN with the

calculation of an output after using different weights represented as w_1 and w_2 and one bias represented as b_1 in the figure 2.2 (WEIDMAN 2019).

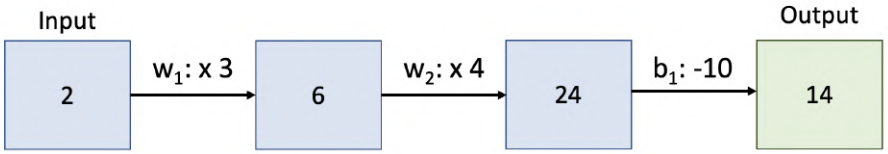


Figure 2.2: Example of the structure of a NN: weights and bias (based on WEIDMAN (2019))

After modifying the input parameters with the weights or bias, they can be modified as well by Activation Functions. These provide an output $f(x)$ after modifying the input x . In the figure 2.3, two examples of calculating an output, with the *Softplus* and *Sigmoid* as activation function, are shown (WEIDMAN 2019).

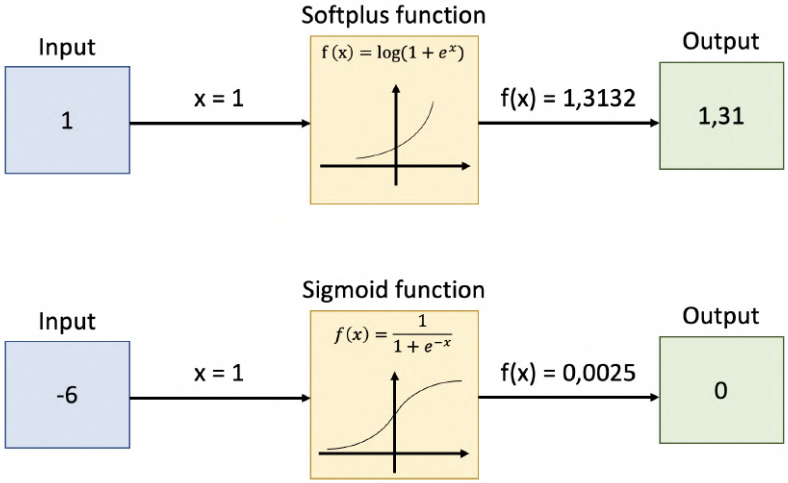


Figure 2.3: Example of Softplus (above) and Sigmoid (below) activation functions (based on WEIDMAN (2019))

When writing the code to program the NN, the only parameters that can be modified in the code are the activation functions. The weights and bias are modified automatically when running the epochs, or learning periods, by comparing the desired output with the obtained one as it was explained before. The picture 2.4 shows an example of a complete NN. Summarizing, once the desired activation functions are programmed and the simulation runs for the first time, the first output is calculated from the first input established. Then, the calculated output and the desired output are compared with an optimization algorithm, which will be explained in the following section, and the corresponding weights and bias are recalculated. This method is known as *backpropagation*. Once the estimated output is the desired output for any input, the NN has converged and there is no need of performing more iterations (WEIDMAN 2019).

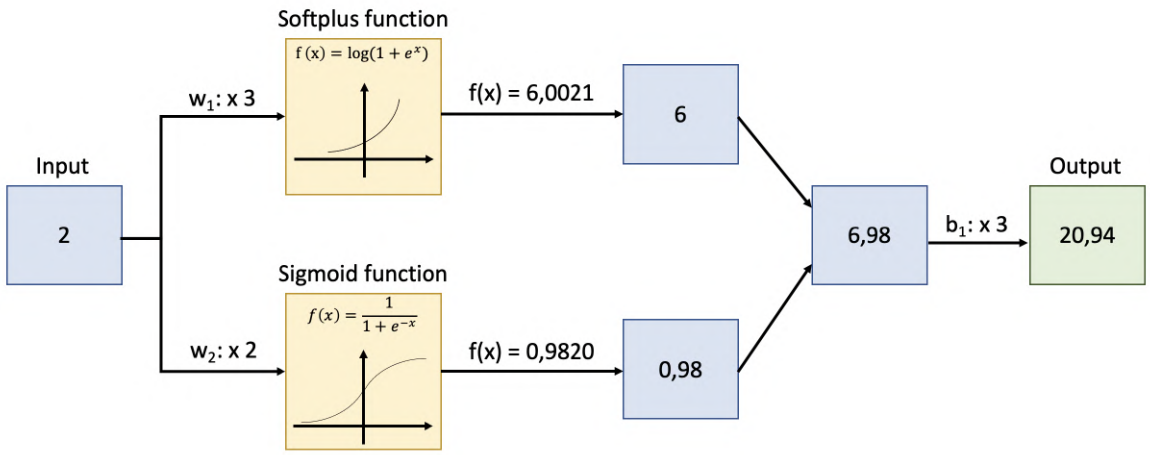


Figure 2.4: Complete example of a neural network (based on WEIDMAN (2019))

2.2.2 Optimization Algorithms

In order to perform backpropagation, an optimization algorithm stochastic gradient descent needs to be applied. The most simple one is the Stochastic Gradient Descent as it is the most popular first-order method to solve stochastic optimization problems (LEE et al. 2017a). However, the complexity of analyzing a picture is higher due to the number of parameters to be computed. For this reason, and based on the results obtained in LEE et al. (2017a) that can be observed in figure 2.5, an Adam optimizer will be used for analyzing pictures in the programmed model. Adam stands for Adaptative Moment Estimation and it is a method for efficient stochastic optimization that only requires first-order gradients with less memory requirement (KINGMA and BA 2014).

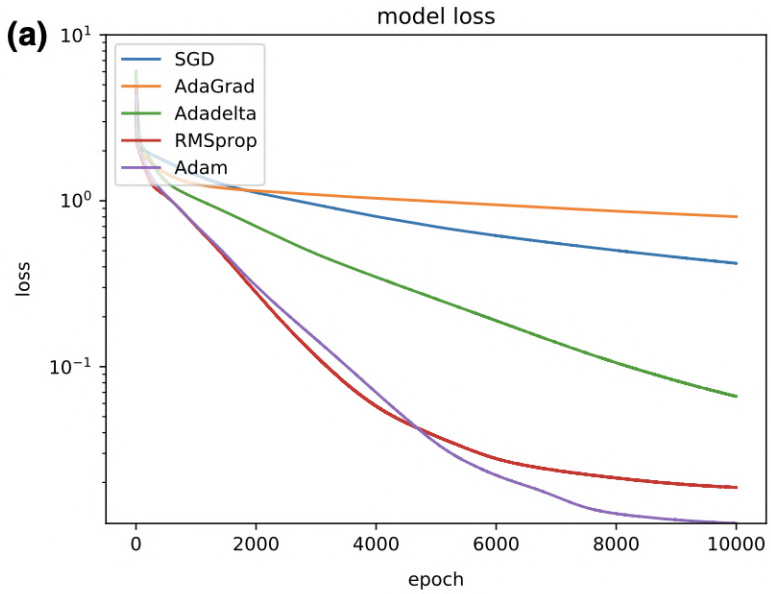


Figure 2.5: Comparisons among various gradient descent optimization algorithms. Source: LEE et al. (2017a)

2.2.3 Loss and Accuracy

In Deep Learning (DL), the NN will always try to minimize the loss to achieve the desired target. But also, in image classification, the classes are classified based on probability. For this reason, the likelihood of assigning the picture to the corresponding class is maximized. With the aim of achieving this, the needed loss to optimize the NN would be defined as *categorical crossentropy*. Once the model is compiling, two different values of loss will be computed. As it will be discuss in figure 3.1, in the chapter 3, each of these losses, known as *loss* and *val_loss*, belong to the train data set and the validation data set respectively (WEIDMAN 2019).

The *val_loss* will be used in this thesis to avoid overfitting the model, i.e. when the model excessively fits the training data. In that case, a decreasing tendency will be observed in the parameter *loss* and a remaining position of the parameter *val_loss* either increases or remains the same. In addition, the value of accuracy will be used to measure the amount of data that the NN is able to classify. Following the same methodology as with the loss procedure, two values of accuracy will be measured, one for the train dataset, and another for the validation dataset. To sum up, the neural networks programmed in this master thesis will try to achieve minimum loss and maximum accuracy, avoiding overfitting (WEIDMAN 2019).

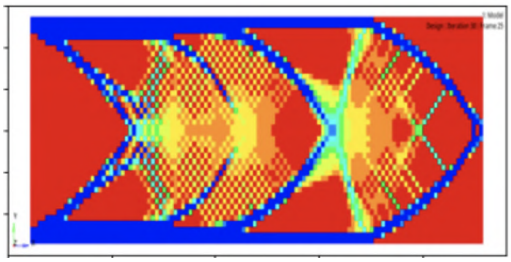
2.2.4 Transforming a Picture into a Tensor

Computers does not process a picture as humans do. While the human brain can distinguish between colors, depth or shadows, computers can only provide numbers to generate an image. The quality of a picture can be defined by the amount of pixels or picture elements, i.e. the smallest discrete unit of a digital image. Each pixel can be composed by a three channel information that determines the red, green, blue and intensity of it. This is known as the RGB values and the figure 2.6 will provide an example of different colors represented with the RGB channels (WARR 2019).

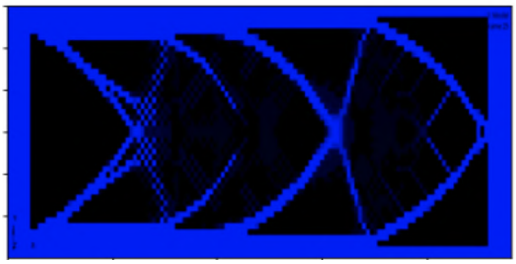
RED	GREEN	BLUE	COLOR
255	0	0	
255	224	32	
0	32	255	
0	192	0	

Figure 2.6: Colors described with RGB channels. (based on WARR (2019))

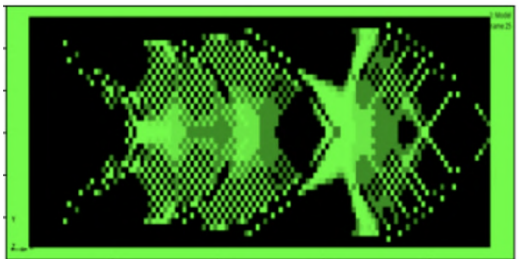
These three channels are stored as a numeric vector. If a picture has a size of 1200×2400 pixel size, the amount of information stored will be equal to $1200 \times 2400 \times 3$ channels $= 8,640.000$ cells containing different numbers that must be processed. The figure 2.7 provides an example of a picture with $8,640.000$ cells of information and its decomposition into the three channels of information. On the upper left corner, the original image of the simulation of the topology optimization is represented. On the following subfigures, the decomposition of the pictures in the blue channel in the subfigure 2.7b, the green channel in the subfigure 2.7c, and the red one in the subfigure 2.7d are shown (WARR 2019).



(a) 3 Channels original picture



(b) Blue channel



(c) Green channel

(d) Red channel

Figure 2.7: Example of the 3 different channels of an image. Figure processed in python from an example of a TO structure.

2.2.5 Last Pre-Processing Step for using Images in the Neural Network

Finally, the last sequence that needs to be performed before start programming the NNs will be explained in the following five steps. First, the images are normalized. This means that instead of a one channel picture which values are in between 0 and 255, the values will be between 0 and 1. Second, the images will be shuffled. This is important in order to make the NN learn randomly so that it learns from all the figures at the same time, i.e. overfitting is avoided. Third, the data will be saved in batches. This part is necessary such the NN needs to receive batches of pictures in order to learn parameters. The size of the batch needs to be related to the size of the dataset. Afterwards, the command *cache* will be employed to store the information of each batch in the RAM for training the NN. Finally, the likelihood bottlenecking will be mitigated with the command *prefetch* (WARR 2019).

2.2.6 Generative Adversarial Networks (GAN)

GAN is a generative model introduced by GOODFELLOW et al. (2014). This NN is divided into a generator and a discriminator, which must learn simultaneously. The discriminator is a binary machine that tries to identify whether the obtained image is real, i.e. from the real dataset or fake i.e. created in the generator. The generator is responsible for creating a new image, as similar as possible, to the one in the real dataset to fool the discriminator. The structure of the network is shown in the figure 2.8 (RASHID and LANGENAU 2020).

Programming a GAN consists in two different training processes and as a result, the learning method should be alternated between training the generator and the discriminator. Only either the generator or the discriminator can be modified during the complete epoch keeping the other one constant (RASHID and LANGENAU 2020).

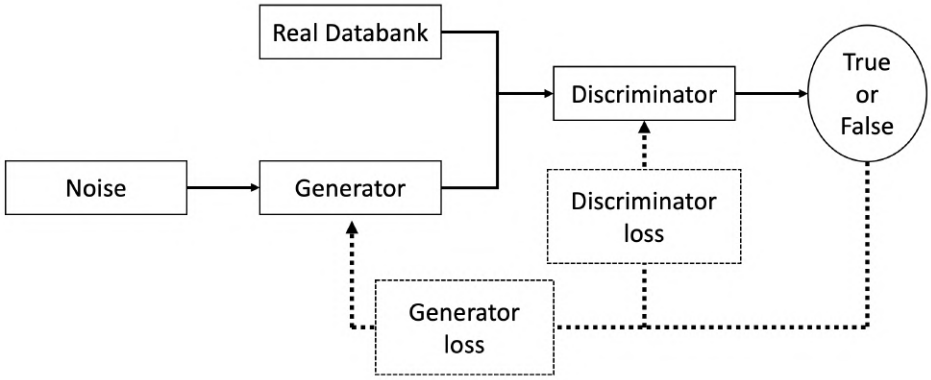


Figure 2.8: Structure of a GAN (based on RASHID and LANGENAU (2020))

This training process should be parallel because otherwise, two undesired situations could happen. If the discriminator is learning too fast, it will not let the generator learn the necessary patterns to create its own images and the generator will not have any pattern to learn how to create an image. For this reason, no image will be generated. On the other hand, if the generator is fooling the discriminator every time, it will learn that the necessary information to generate a picture, is to provide the discriminator with random data. For this reason, only noise pictures will be generated (RASHID and LANGENAU 2020).

These random data pictures are known as noise images, i.e. each pixel contains a random value between zero and one. An example of noise pictures created in this thesis is in the figure 2.9 (RASHID and LANGENAU 2020).

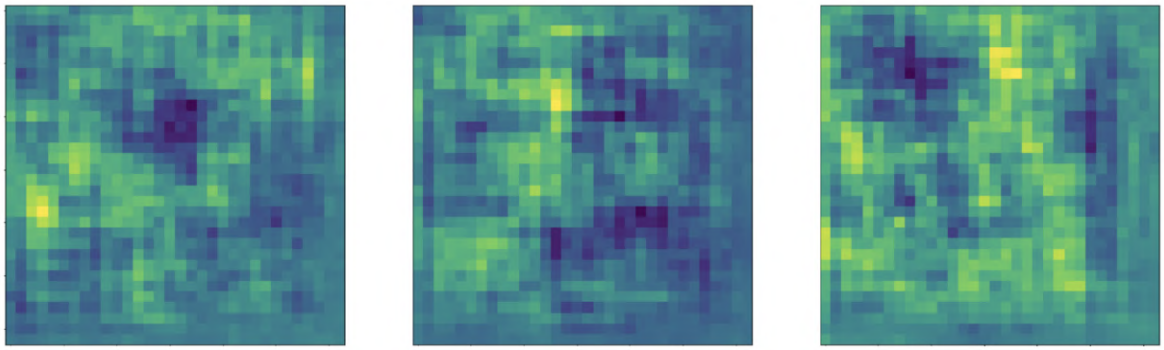


Figure 2.9: Three examples of noise generated in this master's thesis(based on (RASHID and LANGENAU 2020))

2.2.7 Training the Discriminator

The discriminator is a binary machine that obtains samples of pictures from two different sources, the real data bank and the generator. Every time the discriminator receives a picture, it needs to analyze the veracity of the picture by assigning a *True* if the discriminator interprets that the picture comes from the real data bank or a *False* if it assumes that the picture comes from the generator. For this reason, two possible options can be considered whether it is classified correctly or incorrectly and therefore feedback will be transmitted with backpropagation to both, the generator and the discriminator (RASHID and LANGENAU 2020).

The discriminator is related to both, the generator and the discriminator loss, as it can be observed in the figure 2.8. After assessing the veracity of a picture, the discriminator loss penalizes the discriminator for missclassifying a picture from the data bank as false or an image from the generator as true. In this case, the discriminator needs to update all its parameters, i.e. weights and bias, to evaluate the next picture and learn the possible patterns that made the previous picture not good classified (RASHID and LANGENAU 2020).

2.2.8 Training Generator

The most suitable input to produce a picture is random noise, i.e. the examples in the figure 2.9. Once this first random-noise picture is created, the generator varies the patterns and gives the picture to the discriminator until it fools it for the first time. In other words, the generator will change the numbers of the one-channel provided picture until the discriminator believes that it is a real picture from the data bank (RASHID and LANGENAU 2020).

As well as the discriminator loss was providing with information to the discriminator to improve its evaluating methods, the generator loss provides an insight of the characteristics of the picture that makes the discriminator be fooled. This insight information is known as gradient and it is used to change the weights and bias of the NN that improve the training (RASHID and LANGENAU 2020).

2.2.9 Simultaneous Learning

At the beginning, the discriminator has easy time evaluating the origin of the pictures obtained because of the difference between a picture from the databank and a random noise picture, as it can be observed in figure 2.10 (RASHID and LANGENAU 2020).

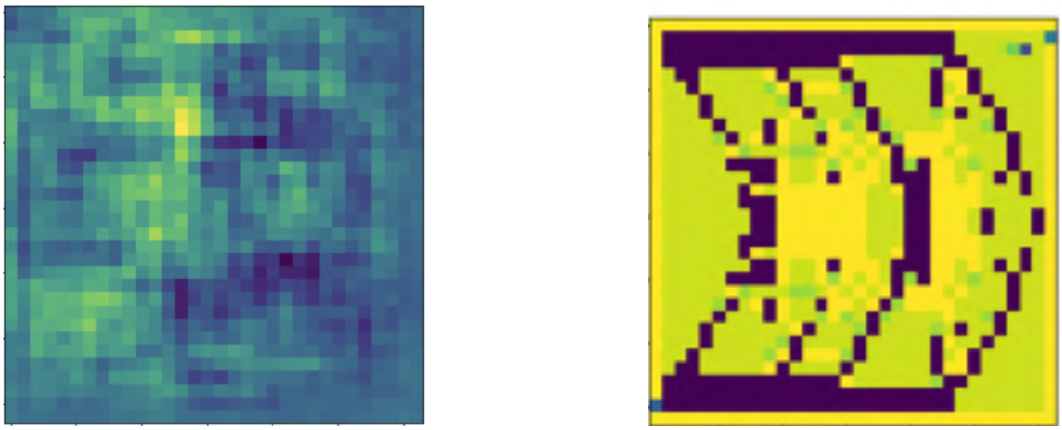


Figure 2.10: Comparison in the first iteration of a picture provided by the generator (left) and one provided by the real databank (right). (based on RASHID and LANGENAU (2020))

For this reason, two methods are applied to make easier for the generator the start of the learning path. The first one is assigning learning rates and the second is applying methods of data augmentation (RASHID and LANGENAU 2020). .

First, when sending information with backpropagation to both the generator and the discriminator, a learning rate can be assigned. Making use of this benefit, a learning rate much

smaller will be assigned to the discriminator. This way, with each iteration, the generator will get more information, than the discriminator, to learn further parameters and create the pictures faster. Second, some of the images from the databank will be modified to make the discriminator confused when analyzing them. The way to achieve this, is by making some databank look like random noise. In other words, adding noise to the pictures (RASHID and LANGENAU 2020).

Nevertheless, these two methods are iterative and there is no fixed number for the learning rate or fixed noise in order to generate the images. A big implementation of these techniques will make the generator learn too fast, creating noise pictures as valid pictures and little implementation will make the discriminator not let the generator learn (RASHID and LANGENAU 2020).

2.2.10 Convolutional Neural Network (CNN)

Deep Learning (DL) is a subsection from machine learning that allows computers produce responses, processing information like a human brain. Moreover, they are able to achieve high level of veracity when examining figures (HEATON 2018).DL uses a deep neural network (DNN) in which a NN is multilayered (TAKAHASHI et al. 2019). The CNN is, within the DNN, capable to recognize different patterns and relationships performing great attainments in image recognition. With this tool, a computer will be able to transform an image into a matrix of numbers and successively into a vector to process the information. An example of the structure of a CNN is provided in the figure 2.11 (SEWAK et al. 2018).

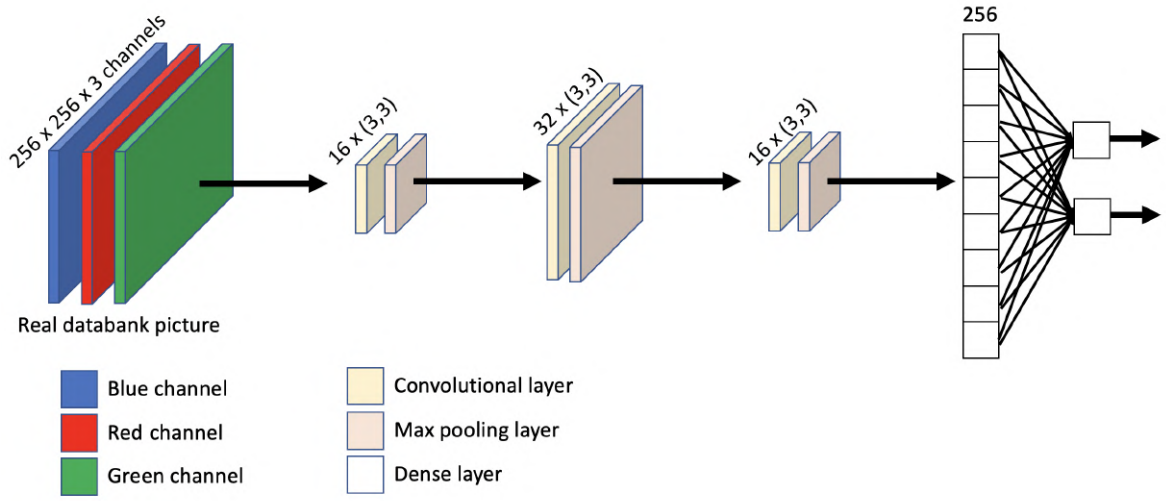


Figure 2.11: Structure of a CNN (based on SEWAK et al. (2018))

Structure and Theory of the CNN

The CNN is divided into three different layers. First, a convolutional layer, then a pooling layer and lastly a fully connected layer. The convolutional layer is the nucleus of the CNN. It oversees using a filter, also called kernel, of size $n \times n$ to analyze the image of size $m \times m$, being $m > n$, by moving one pixel at one time to analyze the structure, properties and analogies. Every time the filter is moving one pixel position in one direction, a dot product is calculated in between the picture and the filter as the figure 2.12 shows (SEWAK et al. 2018).

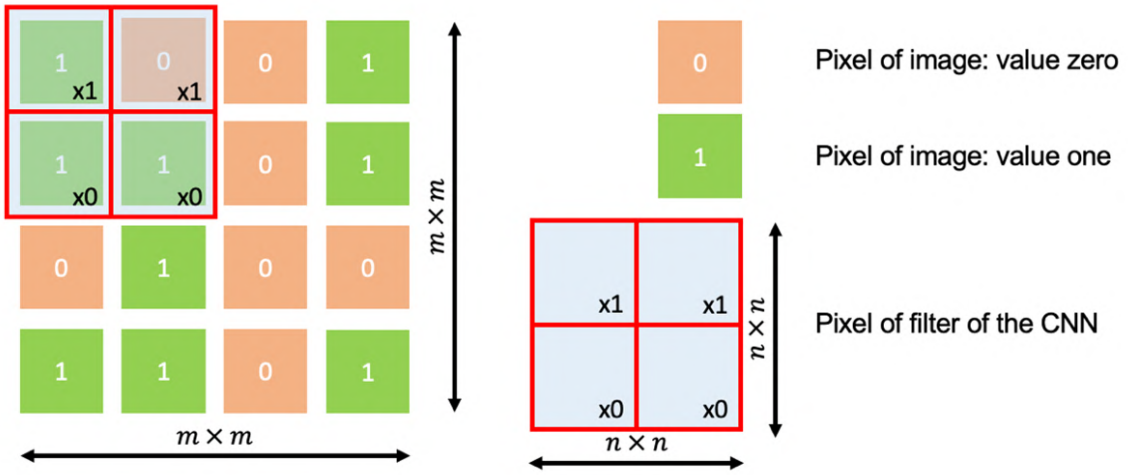


Figure 2.12: Image and filter processed by a CNN. (based on SEWAK et al. (2018))

For example, to calculate the dot product of the pixels selected in the picture 2.13, the math is as follows:

$$0 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 + 1 \cdot 0 = 0$$

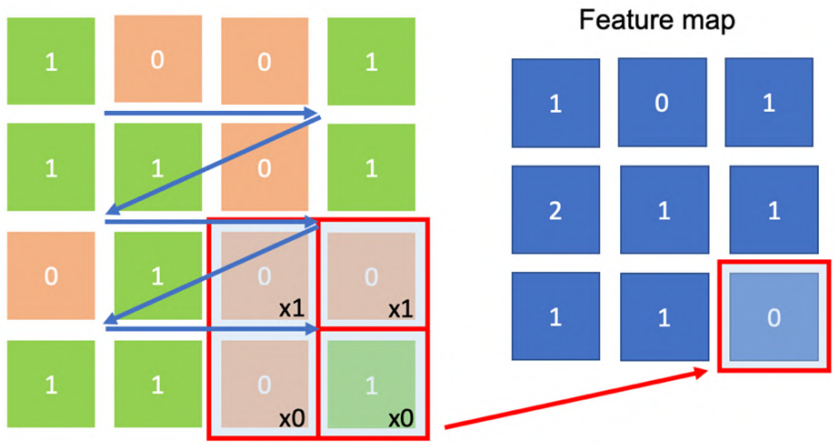


Figure 2.13: Displacement of the filter to create the featured map (based on SEWAK et al. (2018))

Once the filter has covered the entire image and calculated the respectively dot products, it creates a feature map with the respectively results. This method allows a picture be turned into numerical matrices that can be interpreted by a computer (SEWAK et al. 2018).

The pooling layer behaves as the convolutional layer so that it reduces the amount of input parameters. This layer simplifies the operations by establishing relationships, reducing the complexity of the CNN and improving its efficiency. In the figure 2.14 an example of max pooling layer that will be used in this thesis is employed. Lastly, the fully connected layer, links the information from every layer so that the NN can use forward and backward propagation to test the data and establish parameters (SEWAK et al. 2018).

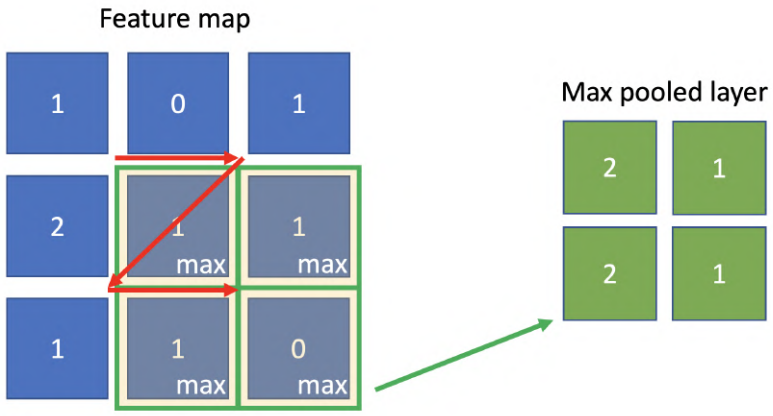


Figure 2.14: Feature map with a max pooled layer.(based on SEWAK et al. (2018))

To sum up the previous steps, the original picture analyzed as 3 channel matrix has crossed different filters. First, a convolutional filter to obtain a feature map and then a pooled filter to obtain a max pooled layer. Once the amount of parameters is sufficient, and no more filters need to be applied, the data is reorganized so that everything can be connected in a NN with weights and bias is it was explained in section 2.2.1. This is known as fully connected layer. The figure 2.15 provides an example of a fully connected layer (WEIDMAN 2019).

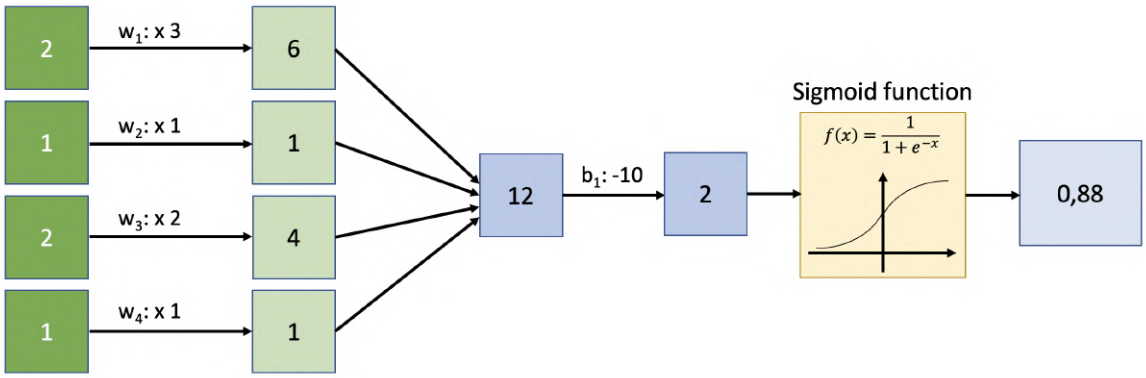


Figure 2.15: Fully connected layer. (based on WEIDMAN (2019))

The final layer used to obtain the desired outputs is the dense layer and it is directly correlated with the number of outputs. For example a NN with two outputs would be as provided in the figure 2.16 (WEIDMAN 2019).

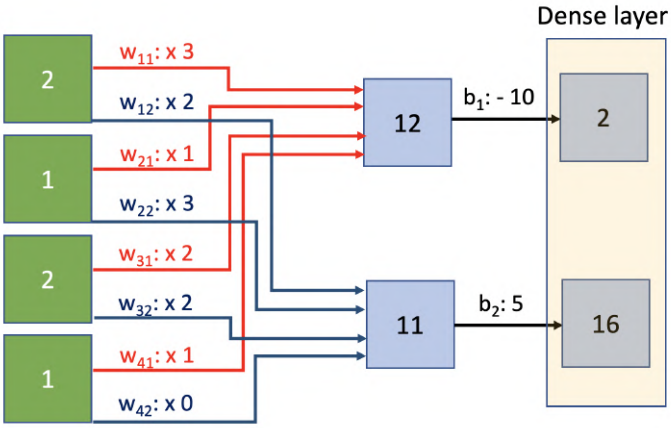


Figure 2.16: Neural network with two outputs (based on WEIDMAN (2019))

Chapter 3

State of the Art

The aim of this chapter is to summarize previous works and bring together the research done to date to obtain a complete overview. The state of art will be divided into three different sections. The section 3.1 will introduce paper works where NN started being used in structural systems to compute complex systems. The section 3.2, will present previous analysis where TO started being combined with both, GAN and CNN. Finally, the section 3.4 will show the importance and the need of action to implement these techniques and keep researching in this field.

3.1 Neural Networks and Structural Engineering Systems

Establishing a relationship between Deep Learning (DL) and structural engineering has always being a matter of interest. The behavior of 1-D structures can be easily hand written and computed, but when referring to 2-D or 3-D structures, the size of the matrices and the amount of computations considerably increase. For this this reason, multiple investigations have been used to implement DL techniques.

For example, LEE et al. (2017a) performed an analysis in a ten bar truss structure analyzing the different activation functions and optimizers to evaluate the displacements of the configuration. An analysis of different settings of the NN showing both, the error in the train and test performances, was analyzed and compared. The figure 3.1 shows the results obtained when training and testing the NN.

Optimizer	SGD		AdaGrad		Adadelta		RMSprop		Adam	
Activation function	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test
Sigmoid	0.093	0.202	0.032	0.190	0.127	0.155	0.015	0.245	0.011	0.289
Tanh	0.055	0.364	0.047	0.336	0.106	0.267	0.053	0.403	0.025	0.630
Softplus	0.049	0.420	0.019	0.178	0.072	0.134	0.014	0.183	0.008	0.235
ReLU	0.058	0.690	0.118	0.835	0.099	0.374	0.032	0.338	0.100	0.502

Table 3.1: Studio made by LEE et al. (2017a) to analyze the mean square error in both training and test sets analyzing five optimizers and four activation functions

After analyzing the influence of the use of AI in the basics of structural engineer, huge advantages were observed in the computational time. For example, both LEE et al. (2017b) and WANG et al. (2020) evaluated displacements in structures and internal stresses in crack

propagation respectively achieving great improvements regarding computational cost. Thereafter, new computer tools with higher computational time, such as TO, started being in the spotlight.

3.2 Using TO with AI

With the aim of merging the AI with TO, DL was selected as a considered tool to be implemented. This way, an optimized solution while satisfying all the requirements from the TO constraint optimization problem would be achieved faster.

At the beginning, papers assessed the new topic of mixing TO with NN as an image segmentation task such as SOSNOVIK and OSELEDETS (2017). In this case, techniques such as pixel-wise image labeling was implemented so that the issue was solved with high efficiency accelerating the method.

Afterwards, CHANDRASEKHAR and SURESH (2021) proved that TO could be programmed using NN by implementing activation functions to represent the Solid Isotropic Material with Penalization (SIMP) used to optimize the density distribution in a structure. They validated a framework of structures in 2D and 3D in which properties from neural networks such as activation functions, backpropagation or implicit filtering were analyzed and studied until the method was validated. An example of the 2-D and 3-D optimized structures created with the NN can be observed in the figure 3.1 and 3.2 respectively.

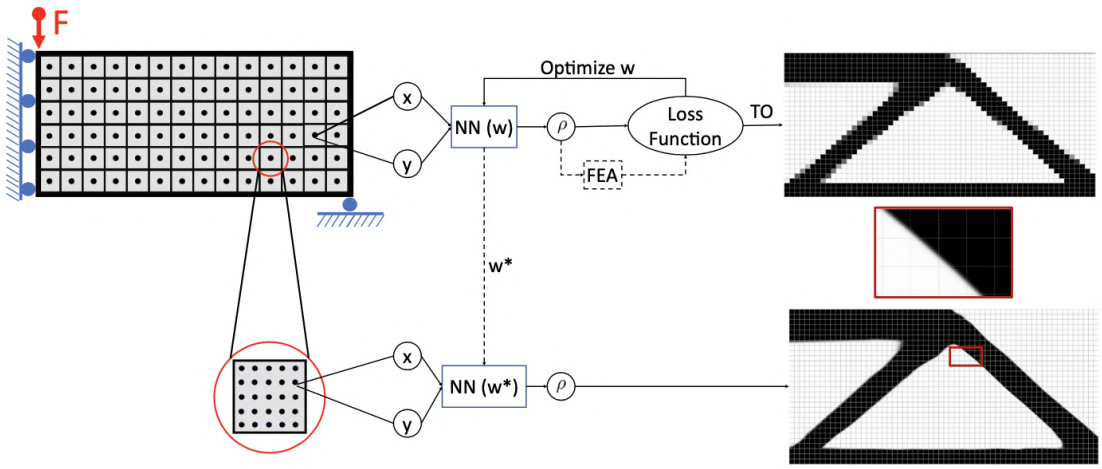


Figure 3.1: Implementation of NN in a 2D structure performed by CHANDRASEKHAR and SURESH (2021)

For the purpose of analyzing the NNs that will be implemented, this section will provide with different examples of previous compilation of TO using GAN and CNN. First, the achievements of implementing GAN will be described and following, the improvements using CNN. This two fields have being potentially investigated enhancing great performances when being implemented. At the end of this section the last techniques combining both GAN and CNN for obtaining TO structures will be presented.

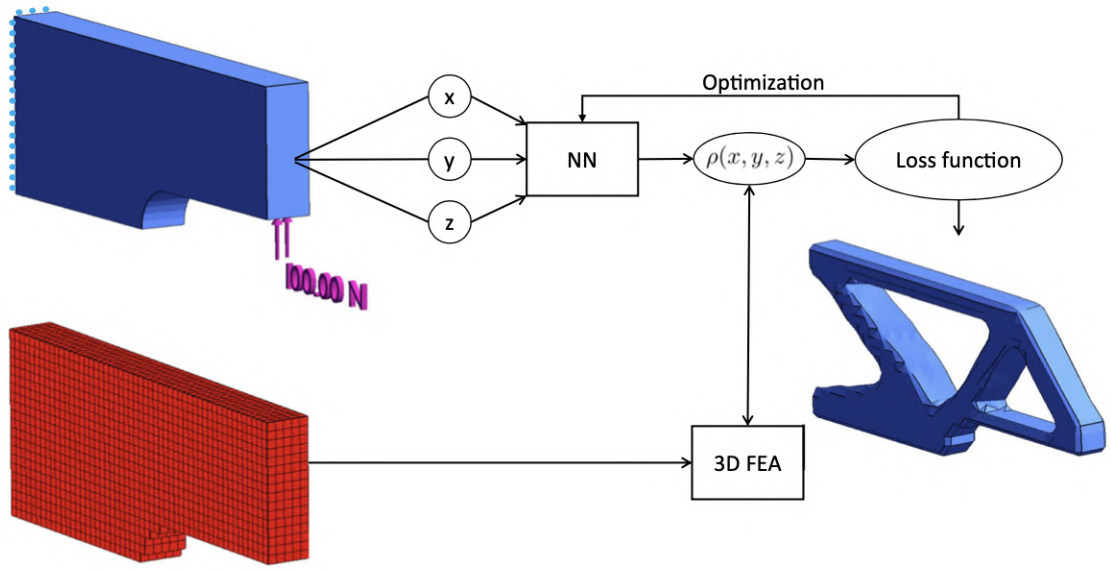


Figure 3.2: Implementation of NN in a 3D structure performed by CHANDRASEKHAR and SURESH (2021)

3.2.1 Implementation of GAN in TO

Among all the different methods employed to automatize TO with NNs , GAN is one of the most popular ones. As explained before, this learning method employs two different NNs to learn parameters and relationships in the images to be generated. Moreover, it has the ability to generate new images or optimal structures applied to design problems achieving all the predefined constrains (RAWAT and SHEN 2019b). For this reason, the research related with GAN is nowadays on the rise.

This master thesis will use the simple GAN. However, nowadays the implementations of GAN are being developed and different types of GAN are being implemented. Among others, RAWAT and SHEN (2019a) implemented a different type of GAN named Conditional Wasserstein Generative Adversarial Networks (CWGAN). In this case, the difference consists of an extra equality constraint to be satisfied to generate the image. An example of the TO figures obtained with the CWGAN, compared with the conventional algorithm, is provided in the figure 3.3 That being said, it can be assumed that the field of programming GAN combined with TO is very wide and therefore, multiple research can be performed.

3.2.2 Implementation of CNN in TO

The CNNs are commonly known for being used in classification and regression techniques. Therefore, this structures are unable to produce an output without analyzing a database (RAWAT and SHEN 2019a). For this reason, they are commonly used to analyze images previously generated, for example with GAN, as RAWAT and SHEN (2019a) did.

As it was explained in the section 2.2, both the generator and discriminator in a GAN are CNN. Therefore, all the examples provided in the section 3.2.1 are examples of CNN as well. However, in this subsection more examples identified as fully CNN will be provided.

(a) TO figures obtained with the normal algorithm by RAWAT and SHEN (2019a)

(b) TO figures obtained with the CWGAN by RAWAT and SHEN (2019a)

Figure 3.3: Comparison of results from the paper RAWAT and SHEN (2019a)

ZHANG et al. (2019) implemented CNN by using the technique U-Net to achieve better results with the NN. By training the NN with a databank of SIMP simulations, the NN was able to reproduce an optimized figure when receiving a tensorflow with the neccessary parameters. All these simulations were performed using encoding and decoding blocks. An example of the results obtained can be observed in the figure 3.4

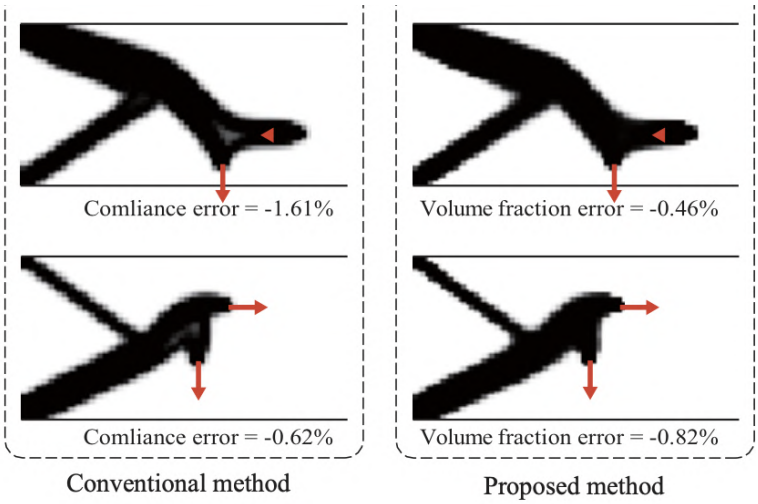


Figure 3.4: Results obtained when applying the U-Net in CNN by ZHANG et al. (2019)

Another example of the use of CNN was provided by XIANG et al. (2022). This paper shows the process of optimizing 3D structures without implementing a recursive process. This work employed a SIMP database of generated structures from random volume fractions and boundary conditions. As well as the previous papers, the computational cost was considerably minimized achieving good designs.

3.2.3 Combination of GAN and CNN

Finally, there is a need to include the importance of combining different NNs. As this master thesis will be implementing GAN and CNN together, this section will show the importance of combining different NN to achieve better qualities.

RAWAT and SHEN (2019b) combined both GAN and CNN to reproduce TO structures. The

flowchart to combine both techniques is provided in the figure 3.5. In this case, a new type of GAN, known as Wasserstein Generative Adversarial Network (WGAN), was implemented instead of a GAN to achieve better results.

The aim of the work of RAWAT and SHEN (2019b) is to map the structures created by the WGAN classifying each of the WGAN outputs with the matching inputs. The results of this paper provided great achievements in computational cost achieving the generation of pictures in nano-seconds. However, the creation of this data was slightly noisy due to the size of the mesh selected as it can be seen in the subfigure 3.6b. In order to improve it, a filter was implemented to obtain the desire figure. An example of the results generated by this structure can be observed in the figure 3.6

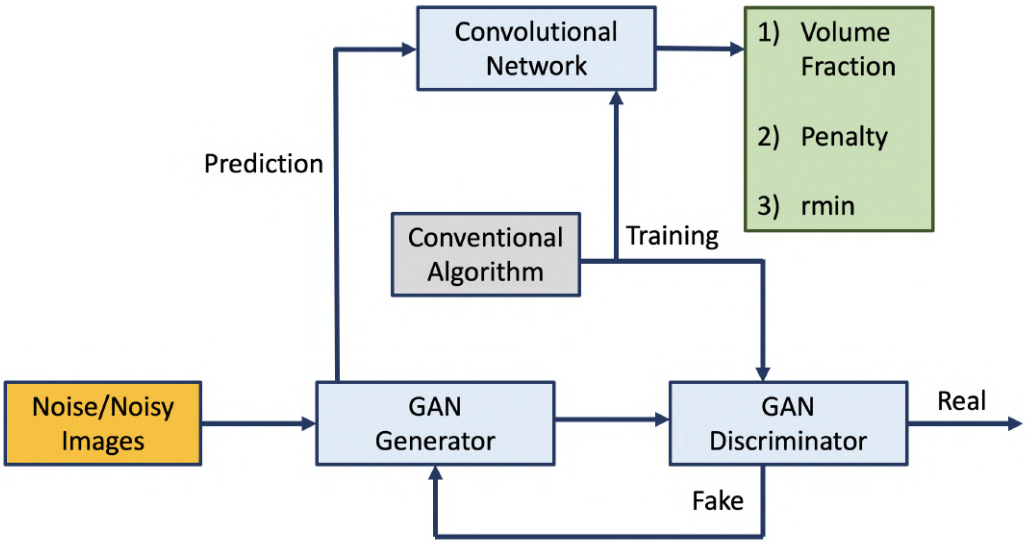
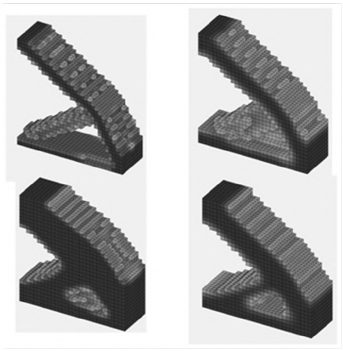
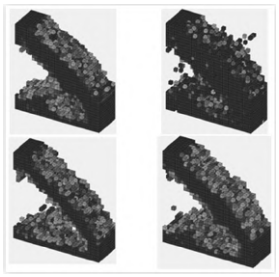


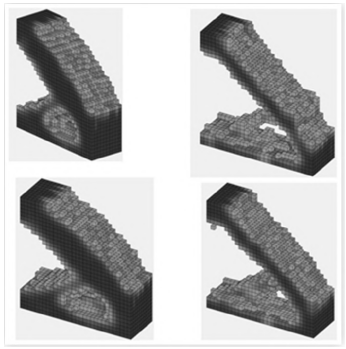
Figure 3.5: Flowchat implemented by RAWAT and SHEN (2019b) to use CNN and WGAN together with TO



(a) TO figures obtained with the normal algorithm by RAWAT and SHEN (2019b)



(b) TO figures obtained with the WGAN by RAWAT and SHEN (2019b)



(c) Post processed figures of the optimal structures from WGAN by RAWAT and SHEN (2019b)

Figure 3.6: Presentation of results RAWAT and SHEN (2019b)

3.3 Overfitting in Neural Networks

During this master thesis, the problem of overfitting appeared during the training and validation of the NNs. For this reason, the possible solutions to minimize this problem, that have been implemented in previous works, are mentioned in this section.

Overfitting can show up in different formats in both the GAN and the CNN. In the case of the GAN, it becomes visible that the NN is not converging, by obtaining a big loss in the generator while the discriminators loss converges to zero. Regarding the CNN, the accuracy of the validation dataset decreases or remains with a low value when computing the NN. For this reason, three methods to reduce this issue, have been studied and analyzed (YING 2019).

First, the work of YING (2019) shows different solutions to reduce overfitting in NN. Among others, this paper analyzes the influence of the treatment of the database related to minimization of overfitting. This paper analyzes that methods of data augmentation such as, adding noise to pictures, processing the existing data to generate new one, adcquire more data will reduce considerably the overfitting of the NN.

Second, the layer *Dropout* is a method that has been implemented to avoid overfitting. This layer is obtained from tensorflow in keras. The paper work of SRIVASTAVA et al. (2014) performed different experiments computing different NN and evaluated that the error obtained when using the layer Dropout was minimized. This can be observed in the figure 3.7. Furthermore, TUSHAR et al. (2017) analyzed the implementation of Dropout confirming that "the Dropout forces neurons of neural network to regularize, resulting in reduced overfitting" Due to these two reasons, the layer Droupout will be implemented in this master thesis to minimize overfitting.

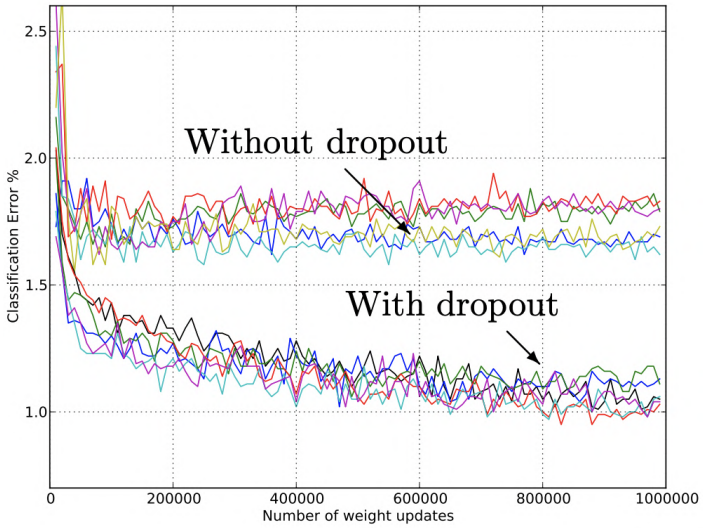


Figure 3.7: Results obtained for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units in NN. Source: SRIVASTAVA et al. (2014)

Besides the advantage of minimizing the overfitting, a disadvantage of using the dropout layer is the the computational cost. The time of training the NN will be affected in an average in between 2 or 3 times slower when implementing Dropout (SRIVASTAVA et al. 2014). For this reason, a layer to neutralize this effect of slowing down the computations can be employed. As TUSHAR et al. (2017) analyzed, the use of Batch normalization reduces the computational time when training a NN by improving the gradients used to compile the parameters of the NN. This layer, combined with Dropout, ensures a better performance of the

NN minimizing the overfitting. For this reason, the use of Dropuout and Batch normalization is a matter of interest that will be studied to overcome Overfitting. The figure 3.8 provides the results of the improvements achieved when implementing both Dropout and Batch Normalization (TUSHAR et al. 2017).

Model	Accuracy	Epoch
CNN	97.00%	85
CNN with Dropout	98.00%	135
CNN with Batchnorm and Dropout	98.50%	72

Figure 3.8: Results obtained when using Dropout combined with Batch Normalization in NN by TUSHAR et al. (2017)

3.4 Intended Added Value

The summed up knowledge from all the previous works commented in this state of art is the basis to make the needed decisions to program the NNs in this master thesis. As shown before, the field of NN is very wide and there are still several topics to be covered by implementing connections in the different types of NNs. There is a need for further research in this field, and for this reason, this master thesis tries to cover the following areas:

- Create an adequate databank to train the NNs
- Develop a method to process the dataset
- Program and adjust the NNs to obtain the desired outputs
- Train and evaluate the method

After covering these points, the main expected advantages can be summed up in three main goals. First, achieving a faster computation for the creation of TO structures. Second, performing a generation of valid images that can be used instead of the ones obtained with the conventional TO algorithm. Finally, ensure a good classification of the generated pictures.

Chapter 4

Method

This chapter will be in charge of explaining the necessary steps employed to establish the entire methodology of this master thesis. For this reason, it will be divided into four different modules. First, section 4.1 will explain how the database was created in order to program the neural networks. This section includes the definition of the model and the necessary TCL scripts to automatically create the model, run it and take the necessary screenshots to create the databank. Second, the section 4.2 will explain how, implementing python, the databank created in the previous section can be processed. This chapter includes the reduction of complexity of the images and the implemented techniques before using the dataset in the NN. Finally, section 4.3, and 4.4 will explain the steps followed to both program NN to achieve the objectives of generating new sets of figures and analyzing them.

4.1 Creation of the Database to Program the Neural Networks

The programming language TCL was implemented in the both Hyperworks and Hypermesh software (ALTAIR ENGINEERING, INC. 2017) to create the databank. The methodology to obtain this database was created by implementing a loop, in charge of changing the position of the load in a cantilever beam, to obtain different configurations of TO structures. Subsequently, the steps implemented for the creation of the database are described below.

4.1.1 Defining the Model and the Boundary Conditions

The model studied in this master thesis was a cantilever beam subjected to different boundary conditions. The dimensions of this cantilever beam were a hundred meters time five hundred meters, as shown in the figure 4.2. The material selected was steel and consequently, the properties were defined as follows in the table 4.1.

Young Modulus	210 GPa
Poison ratio	$0,3$

Table 4.1: Properties used to model the cantilever beam.

The grid size selected to mesh the simulations was five meters. The mesh dimension was chosen under two conditions. First, a qualitative analysis was performed. It ensured that the

post TO density distribution of the structure was not being altered when varying the mesh size. In other words, when decreasing the size of the mesh in the simulation, the density distribution remained the same. Then in order to evaluate the position of the load along the Y axis, each voxel of the mesh should correspond to one pixel of the final generated picture. This was implemented to avoid combination of two different colors in one pixel. Considering that the final size of the picture would be reduced from $100 \times 500$to a size of 32×32 pixel picture, as it is explained in section 4.2.2, the size of the mesh should be a minimum of size of 3.125. Consequently, each voxel of the mesh was represented with one color, and when reducing the picture, the proportion of pixels and colors should be preserved. Due to this, a mesh with size of 5×5 meters was computed to obtain the databank. An example of the final configuration to be implemented is provided in the figure 4.1.

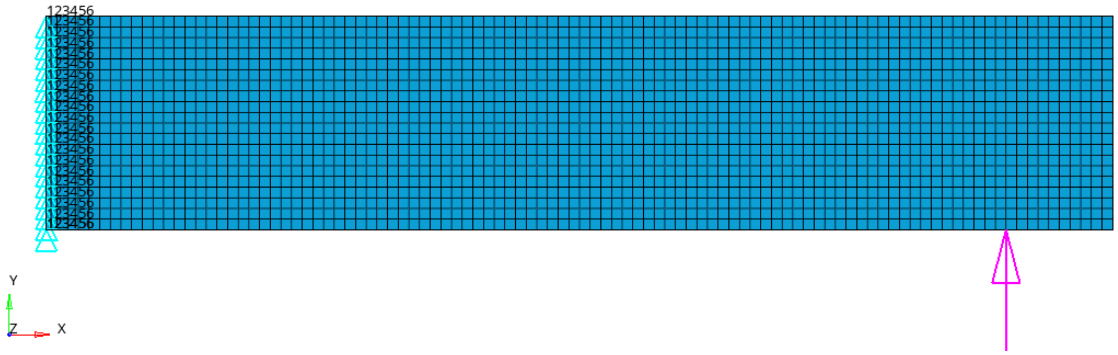


Figure 4.1: Example of the simulation with the fixed conditions on the left side of the cantilever beam and the load applied at $x = 450, y = 0$

All simulations were cantilever beams fixed in the three degrees of freedom on the left side of the structure, i.e. the left side is attached to the wall. The boundary condition that differed each of the simulations was the position of the load. Leveraging the arrangement of the mesh, each simulation was displaying a vertical load every five meters comprehending the x axis, between fifty and five hundred meters, and the y axis, between zero and one hundred meters. That summed up to a total of 1911 simulations that were computed. The figure 4.2 shows the final selected mesh, with each of the white points indicating the position where the load was applied, to obtain the databank.

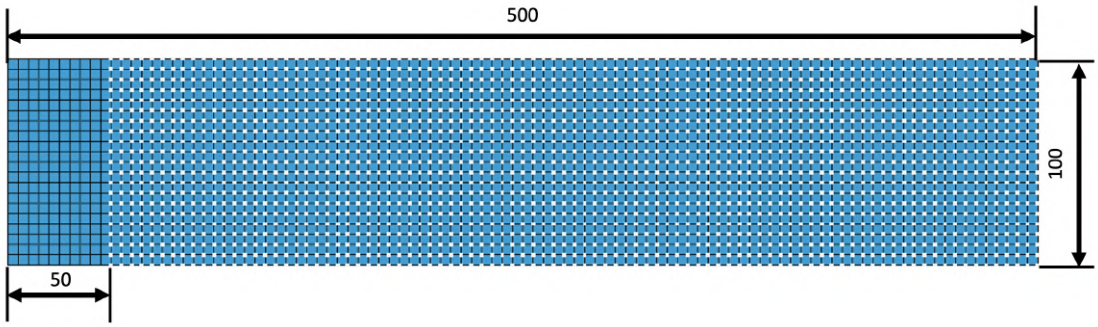


Figure 4.2: Selected mesh with the white points where the load was positioned for each of the simulations

Once the materials and the boundary conditions were defined, the needed properties for the TO were implemented. The variables *compliance* and *volume fraction* were created as *response variables* in Hyperworks. First, the *volume fraction* was set up as constraint, as in equation 2.6 with an upper boundary of 0.3 per voxel. Afterwards, the variable *compliance* was set up to

be minimized in an objective function, as declared in equation 2.5. The rest of the equations, implicit in the theory, were applied when running the solver *OptiStruct* in Hyperworks.

4.1.2 TCL Script to Build and Run the Simulations in Hypermesh

This section was created employing the TCL console in the command window from Hyperworks and the Altair User Guide. Each of the commands selected in Hyperworks were recorded in the tab *Command Window*. A loop was generated by modifying these commands so that the creation of simulations was changing the position of the load in each of the simulations automatically.

This script was compiled together with the compilation script described in the figure 4.3. This process was necessary for the program to process the information and compile the simulations. If two different scripts were used separately to first build the model and then run the simulations,the program was not compiling due to internal errors. For this reason the scripts were combined. The new script would build, run, wait for completion and then delete the model to start a new cycle. With the aim of creating an automatic compilation of the simulations, an script programmed in TCL languages was employed. Due to the unexpected lack of information on the internet, the necessary coding lines, to run the simulation, are provided and briefly explained.

```
*createstringarray 1 "CONNECTORS_SKIP "
*feoutputwithdata "C:/Program
Files/Altair/2021.2/hwdesktop/templates/feoutput/optistruct/optistruct"
"(Folder where the simulations will be saved)/(Name of the
simulation).fem " 1 0 2 1 1
exec "C:/Program Files/Altair/2021.2/hwsolvers/scripts/optistruct"
"(Folder where the simulations will be saved)/(Name of the
simulation).fem " -len 16000 -nt 12
set status -wait
set process autopurge
```

Figure 4.3: TCL code used for the compilation of the simulations in the command window for Hyperworks

First the command **createstringarray 1* prepared the simulation to be exported. Then, **feoutputwithdata*, followed by the directory of the Hyperworks solver output in quotation marks, was exporting the simulation to the directory. Subsequently, and again in quotation marks, the path where the files will be saved, followed by the name of the file plus dot *fem*, was provided. Finally, the command *exec* and *set* were in charge of running the simulations and waiting for the completion of the process.

The code provided in the figure 4.3 was compiled in the command window from Hyperworks. This script is only working with a student or professional license for Hyperworks. As mentioned in the previous section, the code for building and running the simulations must be compiled together in the same script. The concluded reason is because for each of the simulations, the command *autopurge* needs to be followed by the reset of the simulation by deleting the current model and creating the new one.

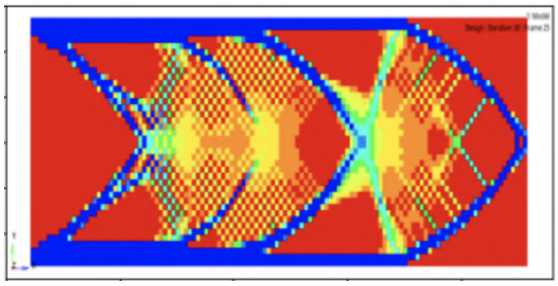
4.1.3 TCL Script to Take the Screenshots

Once all the simulations were compiled, the process of taking screenshots, and saving them, was conducted in Hypermesh. In this section, a different script was employed such as, the commands used in Hyperview are different than in Hyperworks. The purpose of this script was to prepare the images in a dataset to be processed by NNs. For this reason, the pictures were saved in different folders according to the position of the load in the X and Y axis.

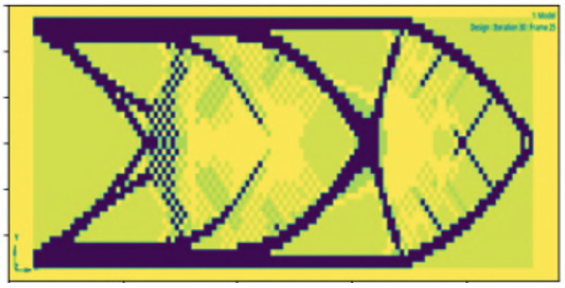
4.2 Processing the Pictures Obtained for the Databank

As described in section 2.2.4, NNs cannot process a picture if they are not decomposed into numbers. For this reason, the second part of the methodology will explain the process of turning the pictures into the necessary information that a NN can process.

4.2.1 Creating the Databank and Reducing the Complexity of the Images



(a) 3 Channels picture. Load in x:500 y:0



(b) 1 Channel picture. Load in x:500 y:0

```
array([[[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],
```

(c) Matrix of the 3 channel subfigure 4.4a. Size $(1200, 2400, 3)$

```
array([[255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
```

(d) Matrix of the 1 channel subfigure 4.4b. Size $(1200, 2400, 1)$

Figure 4.4: Example of processing a 3 channel picture into a one channel picture with its respective matrices

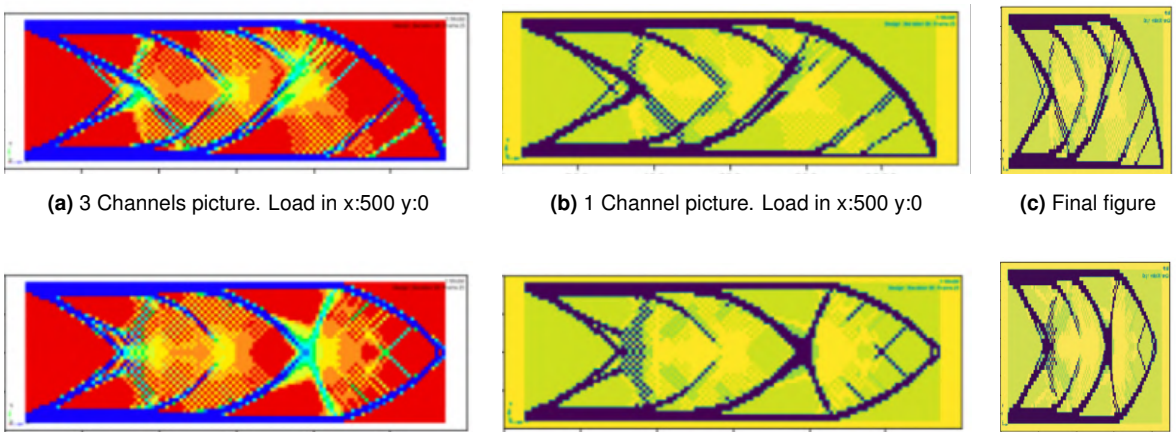
Once all the pictures were saved in the different folders needed for the classification, as mentioned in the section 4.1.3, the command *Tensor Flow Data Set* or *tfds* was implemented to create the database. This command was in charge of creating a databank by saving pictures

as a three channel vector, together with its label. The label was assigned to each of saved pictures depending on in which folder they were previously saved. This sorting method made possible for the CNN to classify the pictures. However, training NNs is a laborious process and, due to the magnitude of training images and the volume of information per picture, the complexity of the figures needed to be reduced.

As it can be observed in the figure 2.7, if only one channel of the three channel picture, i.e., either red, green or blue, was taken as picture to train the NN, information could be lost. For this reason, a combination of the three channels was turned into a one channel picture to preserve all the details. The figure 4.4 will provide the comparison between the original picture and the one channel picture with their corresponding matrices. As it can be observed in the subfigure 4.4c, each of the pixels is represented by a three dimensional array. For example, the first value $[255, 255, 255]$ corresponds to the pixel in the upper left corner of the image 4.4a. These three numbers represent the color white.

When analyzing the transformation into a one channel picture information, each of the three vector array was represented as a one vector array. For example, the color white in the subfigure 4.4a or the vector $[255, 255, 255]$ in the subfigure 4.4b now corresponded to the color yellow in the subfigure 4.4b and the number $[255]$ in the matrix in the subfigure 4.4d. The information from subfigure 4.4c, equivalent to the figure 4.4d, is an example of the simplification of information for the two first rows of pixels of the subfigure 4.4a.

Additionally, the pictures needed to be resized, as mentioned before, into a squared shape for the NN to process the information faster. In the figure 4.5, the process of transforming an original figure into one channel picture, and then resizing it, is provided with two different examples.



(a) 3 Channels picture. Load in x:500 y:0 **(b)** 1 Channel picture. Load in x:500 y:0 **(c)** Final figure

(d) 3 Channels picture. Load in x:500 y:50 **(e)** 1 Channel picture. Load in x:500 y:50 **(f)** Final figure

Figure 4.5: Example of image processing

The format of the pictures provided in the last column of the figure 4.5 was used to train the NNs. The size selected for the CNN pictures was 224×224 pixels with one channel of information. The size for the GAN databank was 32×32 pixels with one channel of information. This meant that the amount of data processed by the CNN is equal to $224 \times 224 \times 1$ channel equals to 50176 cells and by the GAN, $32 \times 32 \times 1$ equals to 1024. This reduction from the original eight million units from the 3 channels picture would drastically reduce the computational time used to train the NN due to the minimization of information.

This method was programmed in python using the command *map*. This function was in charge of calling a function that was able to normalize the figure, resize the data into a

squared shape, and turn it into one channel information picture. The figure 4.6 provides an example of the arrays obtained when processing the dataset for the first time, when computing the command *tfds*, and the desired format to train the NN.

```
{'image': array([[[254, 255, 251],
         [254, 255, 253],
         [252, 255, 255],
         ...,
         [254, 255, 253],
         [255, 255, 253],
         [255, 255, 250]],

        [[251, 255, 255],
         [254, 255, 255],
         [253, 255, 254],
         ...,
         [254, 254, 254],
         [255, 255, 251],
         [255, 255, 250]],

        [[252, 255, 251],
         [252, 255, 255],
         [255, 254, 252],
         ...,
         [255, 255, 253],
         [254, 255, 253],
         [255, 255, 250]]], dtype=uint8),
 'label': 0}
```

(a) Picture processed with *tfds*

```
(array([[[0.99607843, 0.99607843, 0.9764706 , ...,
         0.99607843],
        [0.99215686, 0.99607843, 0.99607843, ...,
         0.99607843],
        [1.        , 0.99607843, 0.99215686, ...,
         0.99215686],
        ...,
        [0.99607843, 0.9254902 , 0.9882353 , ...,
         0.99215686],
        [0.99607843, 0.83137256, 0.99607843, ...,
         0.99215686],
        [0.99607843, 0.93333334, 0.9764706 , ...,
         1.        ]], dtype=float32),
 1)
```

(b) Picture ready to enter the NN

Figure 4.6: Transformation of the picture to be ready to train the NN

4.2.2 Dataset Created for Training the GAN

As analyzed in the section 2.2.6, the figures created by a GAN have quadratic shape. Therefore, the databank of pictures used to train the GAN must have a squared format. The size selected for the creation of the dataset of the TO cantilever beams was based on the images

from the MNIST (Modified National Institute of Standards and Technology) database. This set is composed of 70,000 figures, of size 28×28, of handwritten digits from 0 to 9 (LECUN and CORTES 2010). The pictures of the TO of the cantilever beam were selected with a slightly higher size , i.e., 32×32. Therefore, the final size of the picture was reduced from $100 \times 500 \times 3$ to a size of $32 \times 32 \times 1$ pixels.

4.2.3 Dataset Created for Training the CNN

As previously stated, the CNN uses the information provided by the databank and analyzes patterns in order to sort the figures. For this reason, there must be a balance between training the NN with good quality figures and not exceeding the size of information to make the training process intricate.

With the aim of analyzing the generated figures from the GAN in the CNN, if there is a big difference between the data used to train the CNN and the generated figures provided the GAN, the CNN might not provide accurate results. Therefore, there should not be a big difference between both figures. Consequently, the size of the figures to train the CNN was selected as $224 \times 224 \times 3$.

4.2.4 Last Preparations of the Datasets

In order to prepare the database for the NNs, two different methods were used for the two NNs respectively. The dataset was normalized, shuffled and processed by the cache for both of the NNs. These steps were necessary for the NNs to analyze the information randomly and avoid future overfitting.

However, when creating the size of the batches, the number of pictures included in each batch differed due to the amount of figures in each label. Specifically for the GAN, the batches were made with 128 pictures each. In the CNN, the number of pictures per batch differed. As it will be shown in the section 5.5, several attempts were calculated using different batch sizes to make the case converge. Additionally, the command *likelihood* was implemented in GAN to avoid bottlenecking.

4.3 Programming the GAN

This section explains the methodology to create a GAN in this master thesis. This NN was programmed to be able to learn the patterns of the images from the databank, and therefore, generate its own figures based on learning paths. As explained before, the goal of the GAN was to generate TO structures, appliyng the SIMP methodology, in fewer time as with the conventional method. The methodology of RENOTTE (2022) was used as a guide to program the GAN.

4.3.1 Structure of the GAN

The general structure of the GAN implemented can be observed in the figure 4.7. Once the figures from the database were adapted from the three channel vector to a one channel vector, the generator and the discriminator were programmed. This section explains the main steps to establish a balance between generator and discriminator in the GAN, including computational parameters and the optimization algorithms.

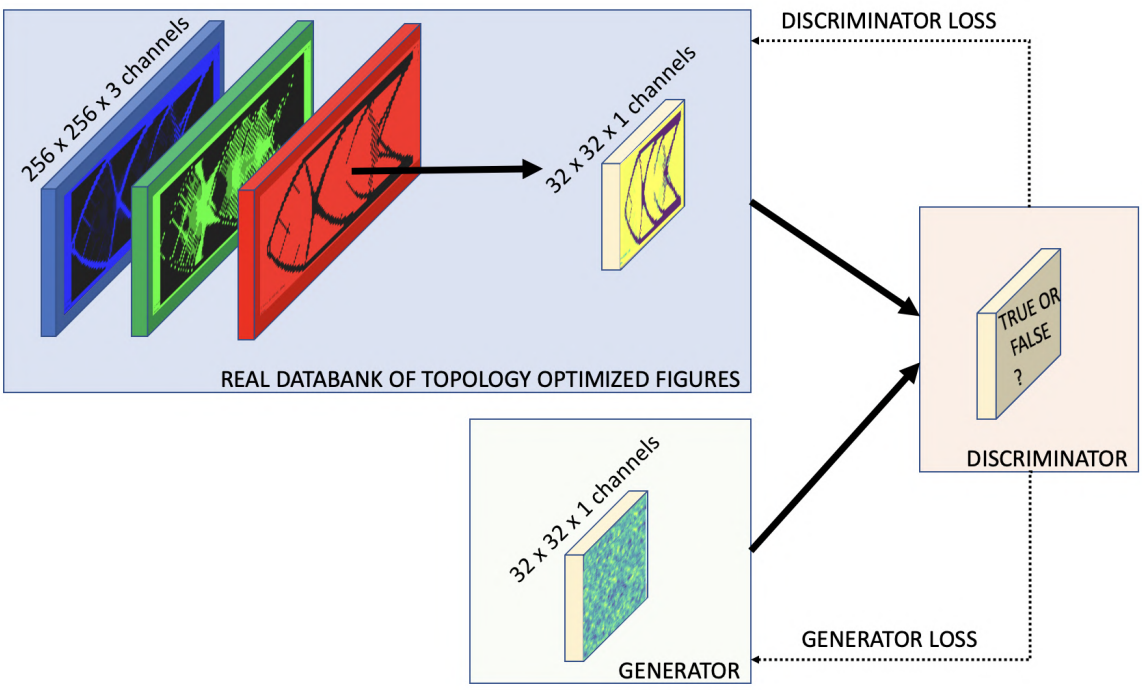


Figure 4.7: Scheme of the GAN programmed

As mentioned, the mission of the GAN is to train a generator by getting feedback from the discriminator to be able to generate its own figures. In order to achieve this, both generator and discriminator were programmed separately with different architectures and parameters.

Before establishing both structures in the GAN, a first estimation of the trainable parameters for the generator was performed based on the database provided. In this case, two thousand images, of 32×32 pixel, were employed to train the GAN. By implementing this databank, two million trainable parameters needed to be programmed to train the generator. Based on the performance and overfitting when computing the GAN, the number of parameters was varied as it will be explained in the results.

As mentioned in the section 4.3 it is important to facilitate the generator learning methods to be able to generate new figures. For this reason, as it can be observed in the figure A, the quantity of parameters provided to the discriminator would be selected as half as the generator i.e., one million trainable parameters. This means that the generator had as twice parameters as the discriminator to adapt and implement learning paths faster to fool the discriminator. In other words, this measure allowed the generator get more feedback to obtain learning paths faster than the discriminator.

Moreover, the tables A.1 and A.2 in the Appendix A, show that the last layer of each of the NNs corresponded to the needed output. The outcome of the generator is the final developed

picture, as it can be seen in the right side of the figure 4.9. This was achieved with the last 2D convolutional layer in charge of generating the picture of 32×32 pixels. This can be observed in the figure A.1 in the output shape from the *conv2d_45* layer. The result of the discriminator is the prediction of the picture to be true or false as if can be observed on the right side of the figure 4.8. This was programmed with a dense layer that saved the prediction in one unit cell is the output shape of the *dense_9* layer as the figure A.2 shows.

Afterwards, in order to create the blocks to achieve the desired number of parameters, the necessary layers to be used in the generator and discriminator were chosen among the ones explained in the theory, in the subsection 2.2.10, and in the state of art, in the section 3.3. The layers chosen were the convolutional 2D, the max pooling, the dense, the flatten, the reshape, the dropout and the batch normalization. As a result, the first draft used the necessary convolutional, upsampling and dense layers with its respective filters until obtaining its desired output. Following that, the other necessary ones such as layer of dropout, batch normalization were added to converge the NN. The following sections provides the detailed information of the distribution of the layers and the parameters implemented in both the generator and discriminator.

4.3.2 Training the Internal Structures of the Discriminator

The structure needed for the discriminator can be seen in the figure 4.8. This structure was in charge of turning an image, i.e., a matrix of information as in the subfigure 4.4d, into a boolean parameter to indicate whether the provided picture belonged to the databank of TO structures or was generated in the discriminator from noise. In order to do so, four blocks of the GAN layers were implemented. Each of this blocks analyzed the figure by passing the figure through a convolutional layer, a Leaky ReLU activation function and a Dropout layer. The function of each of the layers is detailed in the subsection 2.2.10. Moreover, an additional layer, named *Flatten*, positioned all the information processed in the figures in a one dimension vector to compute the final result and compute if the picture was true, i.e. belonged to the databank, or fake, i.e. generated from noise.

4.3.3 Training the Internal Structures of the Generator

The code programmed for the generator utilizes the inverse process as the one used in the discriminator. In this case, the generator must, from a random vector of numbers between one and zero, generate a new figure. With this purpose, an array was relocated in a matrix shape like the subfigure 4.4d to be processed by different layers and therefore create the final image.

In this case, and due to the complexity of the NN, the vector of numbers went through two different blocks. First, the Reshape block converted the arbitrary array into a matrix form with its corresponding Leaky ReLU activation function. Then, seven blocks containing a convolutional layer were implemented to generate the figure. The structure of the generator programmed can be observed in the figure 4.9.

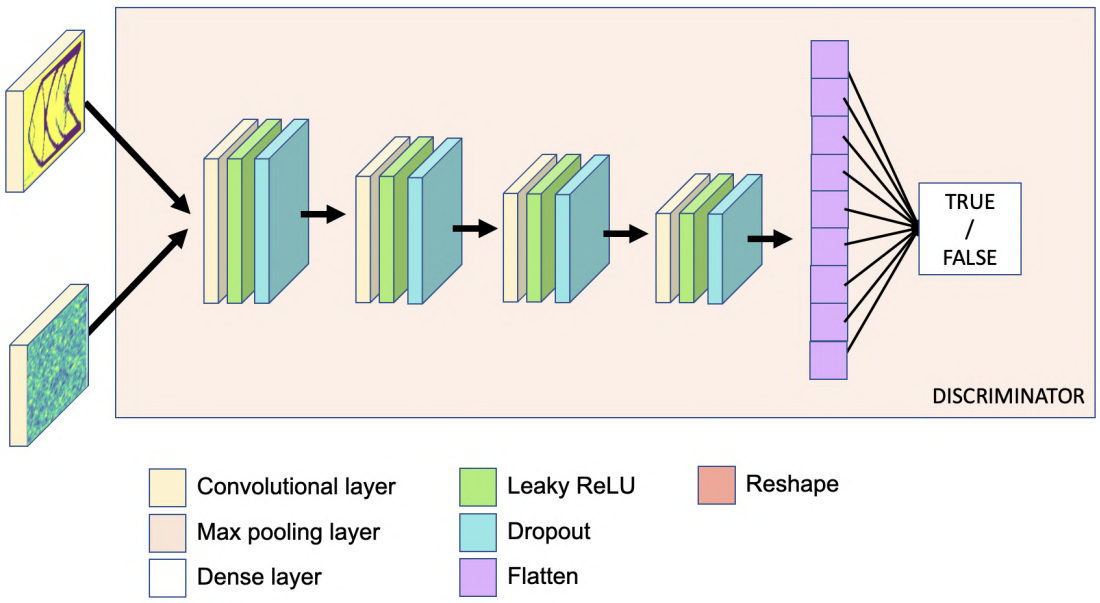


Figure 4.8: Internal structure of the layers of the discriminator in the GAN

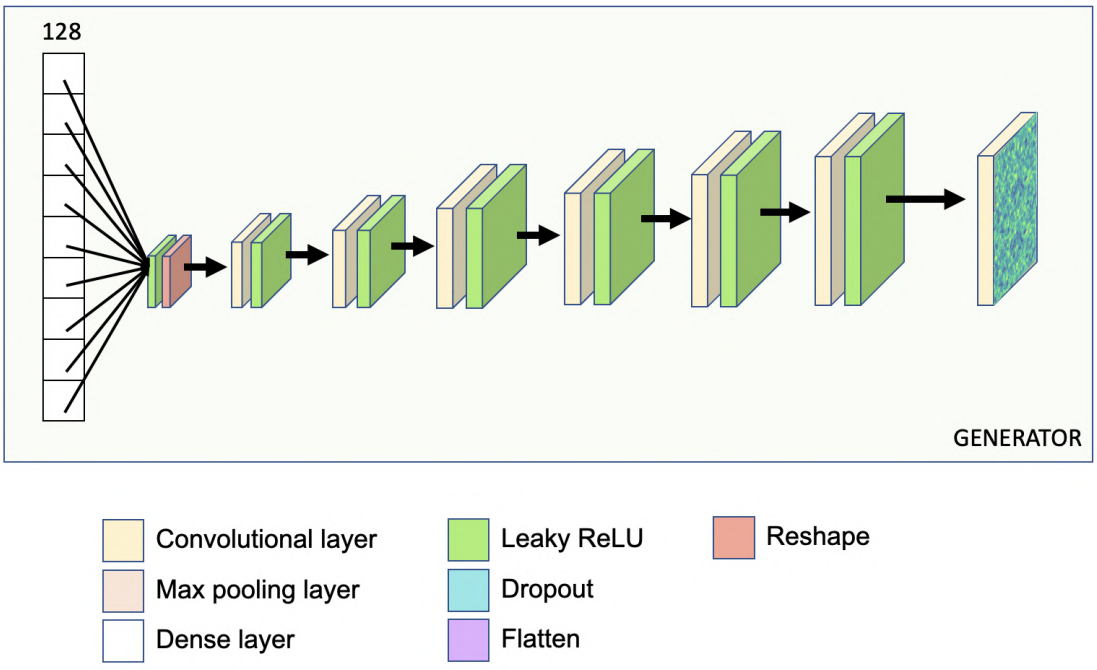


Figure 4.9: Internal structure of the layers of the generator in the GAN

4.3.4 Selecting Optimization Algorithms

As mentioned in the section 2.2.2, the optimizer selected to process an image is based on two main reasons. First, as it can be observed in the figure 4.10 from the studio made by LEE et al. (2017a), Adam optimizer performed the best results achieving minimum loss in fewer time in both studies. Secondly, Adam was used among the others in the work from RAWAT and SHEN (2019b) when specifying that "Adam is preferred over other optimizers because

of the continuous decay of learning rate". For these reasons, and due to the high number of parameters to be computed in order to process a picture, the optimization algorithm selected to be used in the programming of both, the generator and the discriminator in the GAN, was Adam.

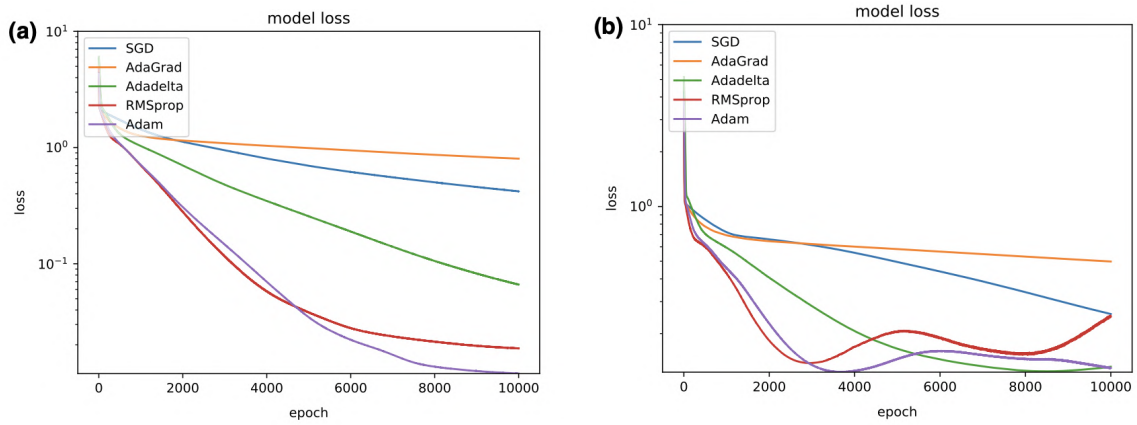


Figure 4.10: Comparisons among various gradient descent optimization algorithms (LEE et al. 2017a)

4.3.5 Methods Data Augmentation and Learning Rates

As it was mentioned in the section 2.2.9, two undesired situations can occur. Either the generator or discriminator can learn faster than the other one, creating noisy pictures or not generating pictures respectively. For these reason, three methods were employed to balance the learning rate and avoid the unfavorable situations. First, random noise was introduced to every picture from the databank to easily fool the discriminator. In order to do this, an arbitrary positive and negative noise of 15 % was implemented. This was programmed by concatenating it to the one channel information figures. Second, the function Dropout was programmed in the discriminator, as it can be observed in the figure 4.8, to avoid bottlenecking. This layer avoided the information to overload, and therefore, facilitated the generator to fool the discriminator.

Third, the learning rate was adapted so that the learning rate of the generator was 10 times faster as the discriminator. In this code, the initial generator's learning rate was set up as $1e^{-5}$ while the initial one used in the discriminator was $1e^{-6}$. These parameters were modified until obtaining the desired output. The discussion of how it was varied is presented in the section 5.3.1. As previously explained, there is a need to facilitate the generator's pattern learning to create the figures. However, this three implementations were reversed to make the generator learn less, i.e., fool less the discriminator, in case it was generating non-desired noisy figures.

4.4 Programming the CNN

The aim of the CNN, in this master thesis, was to obtain a NN that could predict the exact position of a load in the simulated cantilever beam, of dimensions 500×100 meters. It should be mentioned that both the generator and the discriminator are CNNs (RAWAT and SHEN

2019a). However, this thesis aimed to perform a classification of the figures into multiple categories. Therefore, the implementation of a different type of architecture of the CNN was needed.

Having five meters as grid size, and evaluating the position of the load in the simulation, a picture could be classified according to the X and Y axis in 91 classes and 21 classes respectively, as shown in the figure 4.2. However, the process of reshaping the figures of the databank into a quadratic shape made this task more challenging. This classification was studied by analyzing the number of sections in which the cantilever beam could be divided. As it can be observed in the figure 4.11, when compressing the cantilever beam into a quadratic shape, the divisions are positioned closer, which will make more difficult, for a CNN, sorting the figures. The methodology to solve this issue was addressed by dividing the problem into simpler cases. This subsection is emphasizing the different requirements and goals to be achieved in each case.

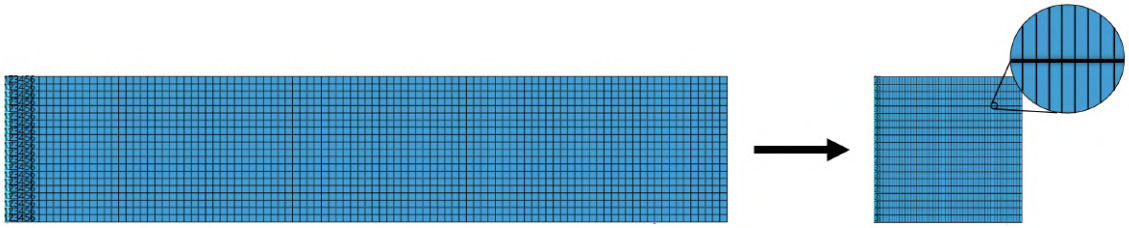


Figure 4.11: Cantilever beam divided in the different classes in the original shape (left) and compressed to enter the CNN (right)

With the purpose of obtaining a CNN able to classify the position of the load in the beam, as precise as possible, the number of positions, were the load is located, was examined. This parameter was increased for every case, to achieve the maximum number of classifications. This section starts with the classification of the load in two different positions. The CNN was programmed to be able to categorize a figure in two classes according to the X axis, i.e. whether the position was on the right or left side of the beam. Afterwards, the evaluation to classify when three or more categories exist was as well performed.

The programming of the CNN followed the same process as the GAN. First, layers of max pooling and batch normalization, to analyze the databank, and dense and flatten layers, to provide the final result of the analysis, were programmed for the first iteration. Afterwards depending on the convergence of the training data and the validation data other layers such as dropout or batch normalization were added. If needed, parameters such as batch size of learning ratio could be as well changed.

4.4.1 CASE 1: CNN Classifies in Two Categories

The programmed CNN able to sort the pictures in two categories, is provided in the figure 4.12. This structure was computed to be able to classify the position of the load in the cantilever beam according to two different locations, either right or left side as the figure 4.14 shows.

As it can be observed in the figure 4.12, the three channel picture of the result of the TO of the cantilever beam was introduced in the CNN and passed through different blocks. First,

three identical blocks composed of a convolutional layer and a max pooling layer oversees the figure and decomposes into a simpler numerical form. Afterwards, a flatten layer and a dense layer were in charge of establishing the needed relationships to provide the final output, that decided the position of the load.

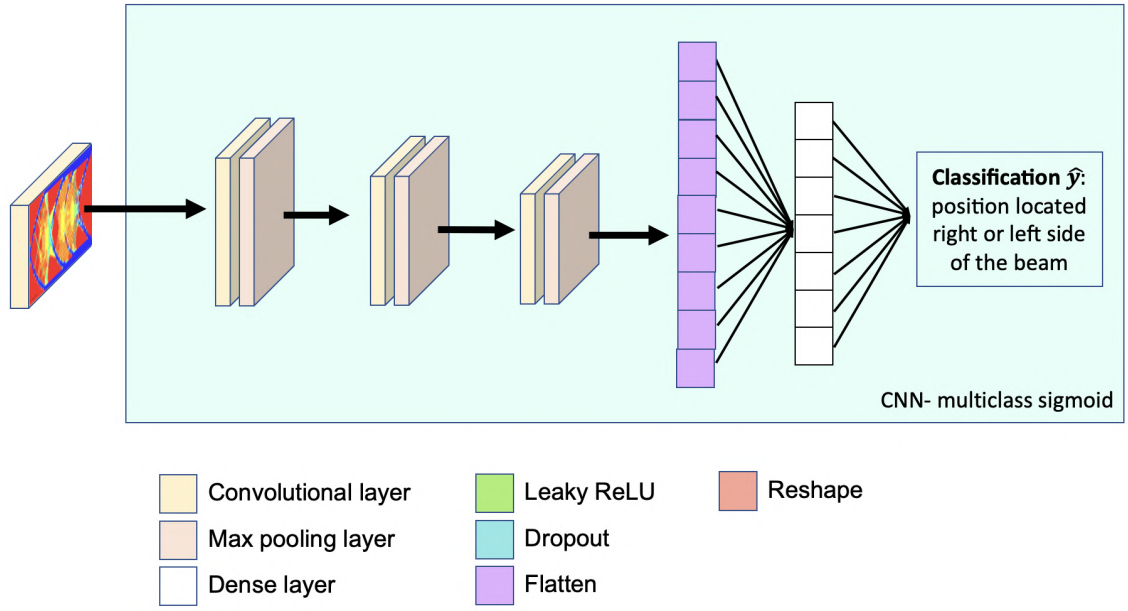


Figure 4.12: Internal structure of the layers of the CNN with 2 classification

The figure 4.13 shows the parameters programmed in the different layers of the CNN. These layers computed the information of the figures until obtaining, at the end, the output of shape *(None,1)*. This value refers to the final solution calculated, that assigned the given picture a classification. In the case of the classification of two different classes, only one output was needed.

As mention in the theory, in the section 2.2.1, the activation functions modifies the value of the NN through an mathematical equation. Sigmoid was used as an activation function to classify the two different categories. The sigmoid mathematical formula to compute the model is provided below:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1} \tag{4.1}$$

For every value of x, the values obtained in $S(x)$ are comprehend between zero and one. This worked as a threshold, such that for high values of x, the output is one, and for negative values, zero. Consequently, implementing it in the penultimate step of the NN, the output results in a interpolated number between zero and one.

Due to the reason explained above, the sigmoid function was used to classify the input data into two different classes. The CNN is programmed to classify whether the position of the load was on the right or left side of the beam as it is represented in the figure 4.14. In this case, a value close to one, meant that the position of the load was located on the left side of the beam, and a value close to zero, on the right side.

Once the figures have been adapted to enter the NN, as explained in the section 4.2, and the NN has being programmed, the databank was divided into three different sections in order

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 254, 254, 16)      448

 max_pooling2d (MaxPooling2D  (None, 127, 127, 16)     0
 )

 conv2d_1 (Conv2D)            (None, 125, 125, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 32)       0
 2D)

 conv2d_2 (Conv2D)            (None, 60, 60, 16)        4624

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 16)       0
 2D)

 flatten (Flatten)            (None, 14400)             0

 dense (Dense)                (None, 256)               3686656

 dense_1 (Dense)              (None, 1)                 257

=================================================================
Total params: 3,696,625
Trainable params: 3,696,625
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.13: CNN used to classify in 2 different subgroups

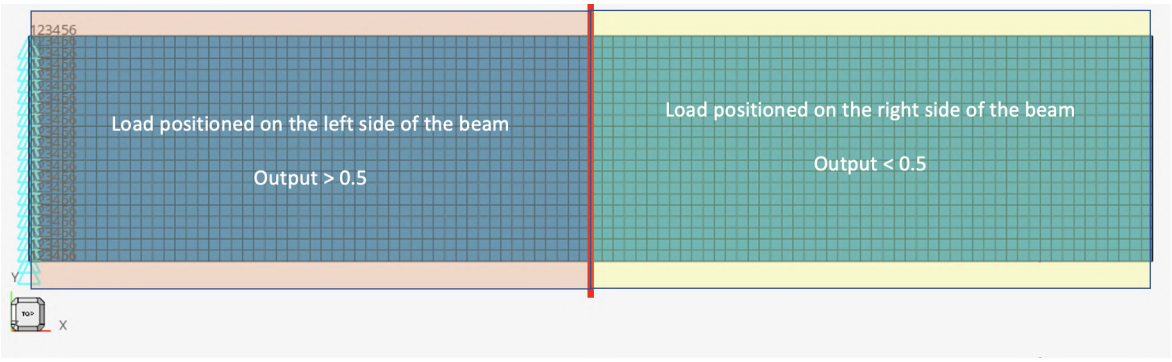


Figure 4.14: Classification of the position of the load for the CNN in case 1

to train the CNN of two categories. The division was composed of 70 % of the databank as training data, 20 % as validation data and 10 % as testing data. Therefore, when training the NN with the 70 % of the databank, the network was evaluating the *trainning loss* and the *trainning accuracy*, and when validating it with the 20 %, the network will evaluate the parameters *validation loss* and *validation accuracy* as explained in the theory in the section 2.2.3.

Lastly, before compiling the model, the necessary settings were established to run the epochs. First, based in the paper from LEE et al. (2017a) again, Adam and RMSprop were alternated as optimizers to achieve convergence. Moreover as this CNN was used to classify a figure in two different categories, then binary crossentropy was implemented.

4.4.2 CASE 2: CNN Classifies in Three or More Categories

Once the classification into two different categories was performed, with the aim of achieving a better classification of the position of the load, the new goal of the CNN was to recognize multiple categories. Consequently, a few modifications needed to be performed when obtaining the data from the directory files. The figure 4.15, shows the structure of a CNN able to classify in different groups.

As it can be observed in the figure 4.16, the output shape in the last dense layer is (None,3). This means that this CNN was able to classify the figures into three different groups. In this case, the use of sigmoid function was not further possible. Given that the sigmoid function is limited to the classification of two classes, another activation function was necessary to differentiate multiple classes.

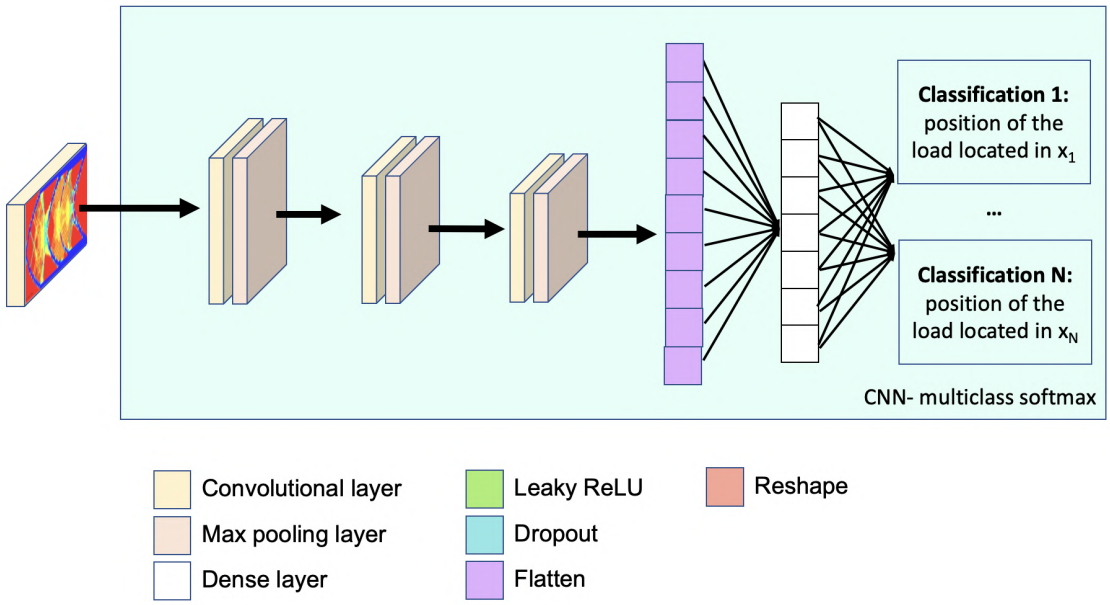


Figure 4.15: Internal structure of the layers of the CNN with 3 classification

With the aim of performing a multi-class classification in the CNN, the new activation function to be implemented was *Softmax*. This parameter provides the NN with a percentage up to one hundred to decide in which ratio, a picture provided to the NN, belongs to a category. The better the NN, the higher the fraction assigned to the figure belonging to the right class. The mathematical formula of the Softmax function is as follows.

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{4.2}$$

The value N is the number of outputs that the CNN provides at the end. In other words, the amount of classes in which the CNN will be able to classify the picture. The value z is the value obtained in the CNN in each of the classes before applying the Softmax activation function.

In addition to the implementation of the Softmax function, another parameter was changed respect to the classification of two categories in subsection 4.4.1. For the purpose of pro-

cessing the pictures and differentiating the different classes, the value *label* obtained when processing the picture with the tfds data, was changed.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 224, 224, 32)      896

batch_normalization_16 (Bat  (None, 224, 224, 32)      128
chNormalization)

conv2d_17 (Conv2D)           (None, 224, 224, 32)      9248

batch_normalization_17 (Bat  (None, 224, 224, 32)      128
chNormalization)

max_pooling2d_8 (MaxPooling  (None, 112, 112, 32)      0
2D)

conv2d_18 (Conv2D)           (None, 112, 112, 64)      18496

batch_normalization_18 (Bat  (None, 112, 112, 64)      256
chNormalization)

conv2d_19 (Conv2D)           (None, 112, 112, 64)      36928

batch_normalization_19 (Bat  (None, 112, 112, 64)      256
chNormalization)

max_pooling2d_9 (MaxPooling  (None, 56, 56, 64)        0
2D)

flatten_4 (Flatten)          (None, 200704)            0

dense_6 (Dense)              (None, 8)                 1605640

dense_7 (Dense)              (None, 3)                 27

=================================================================
Total params: 1,672,003
Trainable params: 1,671,619
Non-trainable params: 384
_________________________________________________________________
```

Figure 4.16: CNN used to classify in 3 different subgroups

In the previous case, the label of each picture was saved with a natural number as it can be observed in the figure 4.6. However, in order for a CNN to perform a multi-class evaluation, the identification of the category had to be saved in a categorical form. The figure 4.17 provides four examples of different cantilever beams, programmed with the CNN from the figure 4.16, labeled with its respective array of classification.

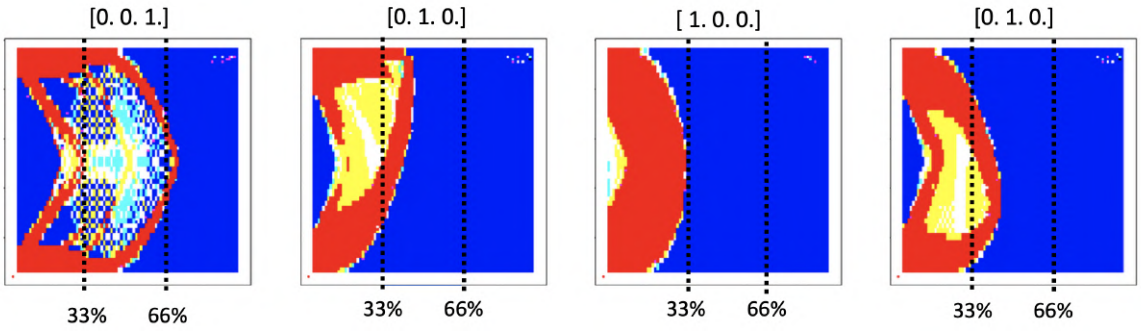


Figure 4.17: Subfigure with a categorical classification above

As it can be observed, the classification comprehended a vector of size N, i.e the size of the classified outputs, where the number one is displayed in the position where the figure will be classified. The figure 4.18 shows the classification of the vectors related with the position of the load of the beam in the cantilever beam. This method allowed the CNN to understand when training, that the picture provided corresponded to the group assigned. This labeling method allowed systematically categorizing the figures into any desired number of classes.

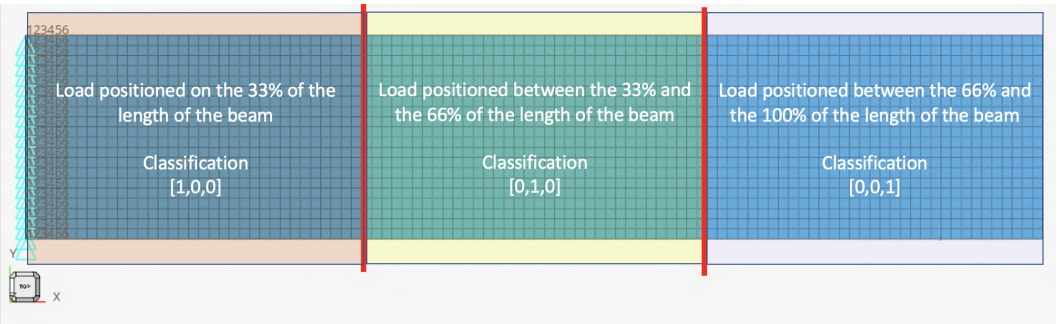


Figure 4.18: Subfigure with a categorical classification above

Lastly, the necessary parameters to compile the model were configured. First, based in the paper from LEE et al. (2017a), Adam as an optimizer was implemented. Moreover, categorical crossentropy was set up as an optimization algorithm to calculate the loss. As explained in this section, the CNN needs to label a figure into multiple classes in a categorical form. Therefore, to compile the model, the binary crossentropy is not further valid to calculate the loss from the multiclass CNN.

Chapter 5

Results and Discussion

The chapter *Results and Discussion* includes all the achievements obtained in this master thesis with its respective analysis. First, the section 5.1 provides the convergence obtained, when performing the TO, to calculate the necessary figures for the databank. Then, the results of the simulations, and the final pictures, with its respective necessary modifications to enter the NN, are discussed in the section 5.2. Afterwards, the section 5.3, explains the analysis performed to validate the GAN and in 5.4 the results are discussed. Finally, the process to maximize the number of categories to classify a figure with the CNN is given in section 5.5 and its corresponding discussion in the section 5.6.

5.1 Results of the Databank

The basis of this master thesis started obtaining the databank of TO simulations of the designed cantilever beam to be able to train the NNs. First, this section provides the convergence implemented to obtain the TO figures. Afterwards, these images aimed to get a balance between computing high quality pictures of the TO simulations, and reducing the quantity of information to enter the NN. Considering this, two different types of dataset were provided for training the GAN and the CNN respectively.

5.1.1 Databank Created of the TO Figures

As mention in the theory, in section 2, the objective of the TO was to minimize the compliance of the simulated structure. Therefore, an analysis of each of the 1911 simulations was performed to achieve this goal. Following, the example provided in the figure 5.1 presents how the results were analyzed. This result corresponds to the convergence of the compliance in the cantilever beam simulation with the load in the the position 500 for the X axis and 100 for the Y axis.

The first result of the compliance of the model, when it was not optimized, was 13.5 Nm, as it can be seen in the figure 5.1. This curve was analyzed, observing that in the iteration number five, the simulation started converging. The result obtained is represented in figure 5.3a. Nevertheless, the output image was varying for each extra iteration. For this reason, the number of iterations was increased until not observing further changes in the final density distribution. As a result, 30 iterations were established for the models to converge. The final

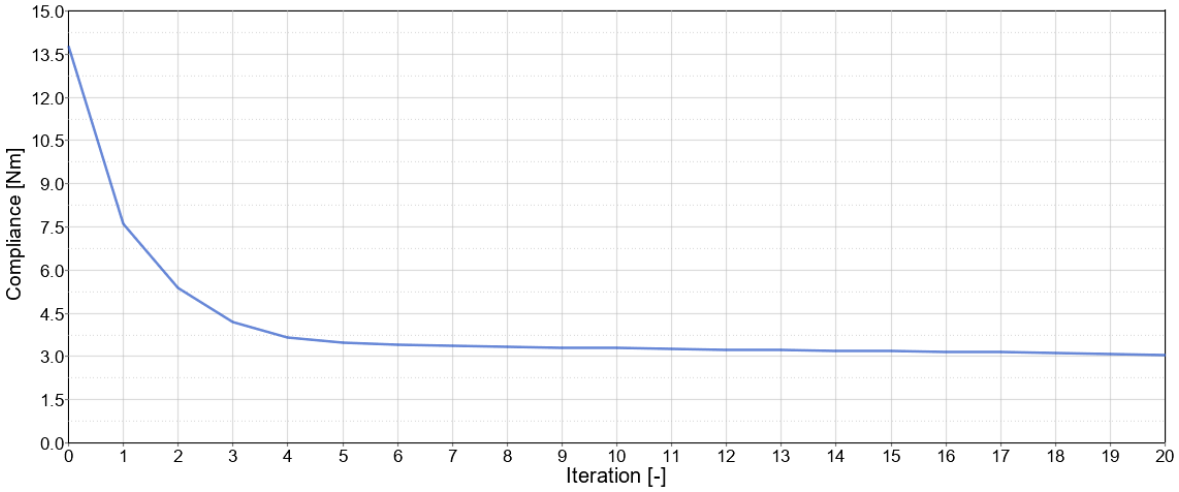


Figure 5.1: Compliance as the objective minimized function for the cantilever beam simulation with its load located 500 meters on the X axis and 100 meters on the Y axis. Represented until iteration 20.

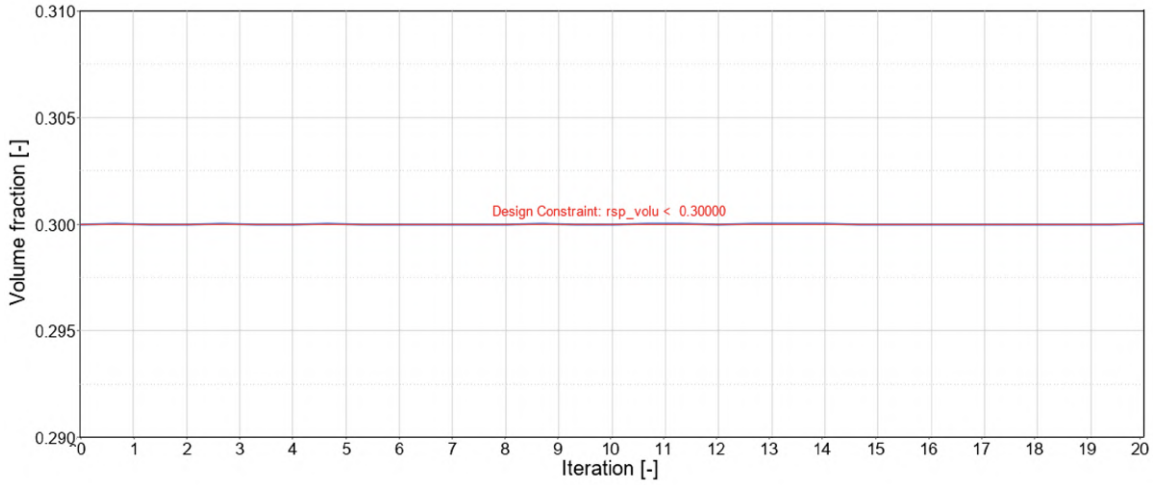
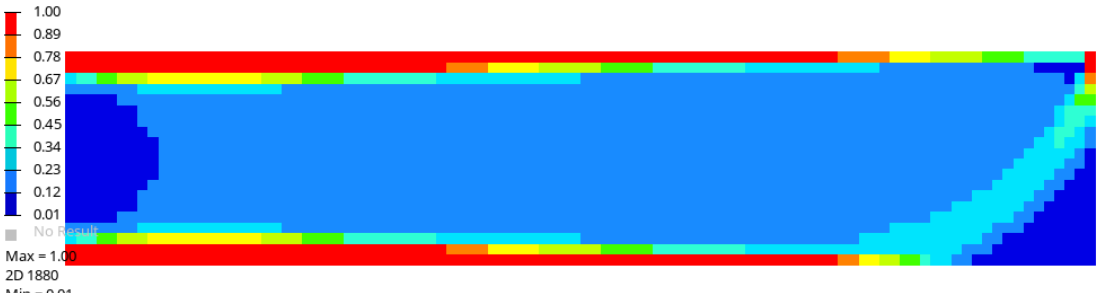


Figure 5.2: Prove of satified constraint of minimum volume fraction of 0.3 for the simulation of the cantilever beam with its load located 500 meters on the X axis and 100 meters on the Y axis

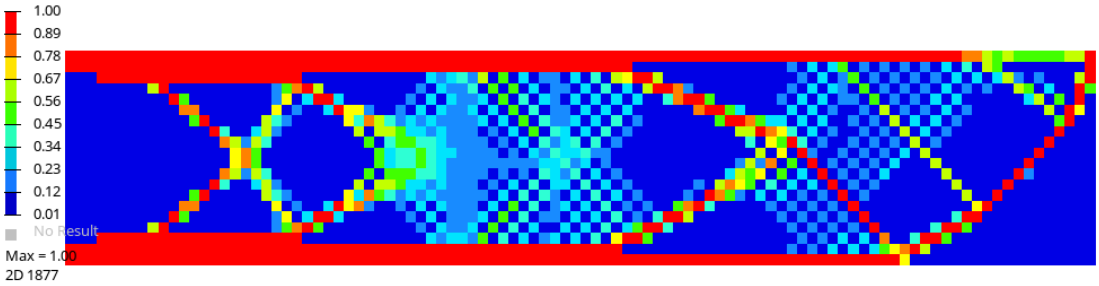
optimized shape represented, and analyzed in this section, can be seen in the figure 5.3b. Furthermore, the result of implementing the constraint, of 0.3 for the minimum volume fraction, along the iterations can be observed in the figure 5.2. The converging process of the compliance, and the implementation of the volume faction as a constraint, were performed for the 1911 samples used in this thesis as explained in this section.

5.1.2 Databank Created for Training the GAN

It needs to be mentioned that the legend, to classify the density distribution of the simulations, was removed from each of the images results in this thesis. The main reason, is that the figures were simplified, and the legend was not part of the computation of the NN. Therefore, the legend that applied to every TO simulation created in Hyperworks is represented in figure 5.4. The scale of colors represents dark blue for zero relative density, or void, and red, for one hundred, or full material.



(a) Optimization of the cantilever beam after the fifth iteration



(b) Optimization of the cantilever beam after the 30^{th} iteration

Figure 5.3: Computation of the picture to be ready to train the NN

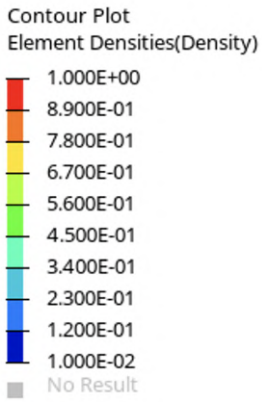
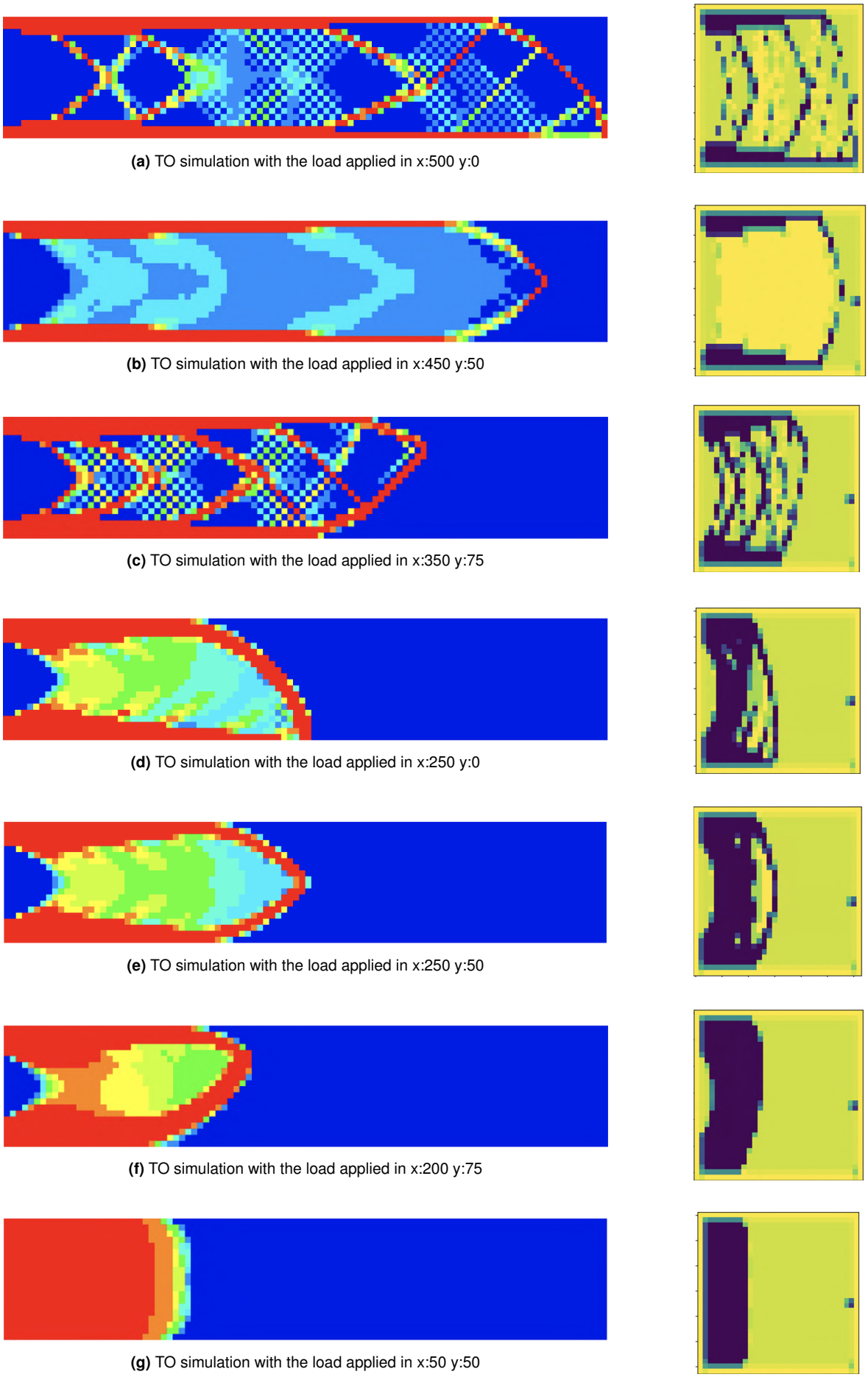


Figure 5.4: The density distribution legend corresponding to the left column of images in the figure 5.5

As mentioned before, the figures to be generated in the GAN, were based on the size of the MNIST dataset (LeCun and Cortes 2010). By implementing that, the final size of the picture was reduced from $100 \times 500 \times 3$ to a size of $32 \times 32 \times 1$ pixel picture. It can be observed, in the figure 5.5, that when compressing the figure, the information of inner structure of the image was reduced compared to the original output. After obtaining the first result of the GAN, the idea was to further improve the quality of the GAN by processing higher quality pictures. But due to an internal error of the server compilation when running the GAN, no further improvements could be performed. This issue will be addressed in the section 5.4. The databank compiled in Hyperworks can be observed in the left column of the figure 5.5 and the corresponding adaption of the figures to enter the GAN on the right column of the same figure.



(a) TO simulation with the load applied in x:500 y:0

(b) TO simulation with the load applied in x:450 y:50

(c) TO simulation with the load applied in x:350 y:75

(d) TO simulation with the load applied in x:250 y:0

(e) TO simulation with the load applied in x:250 y:50

(f) TO simulation with the load applied in x:200 y:75

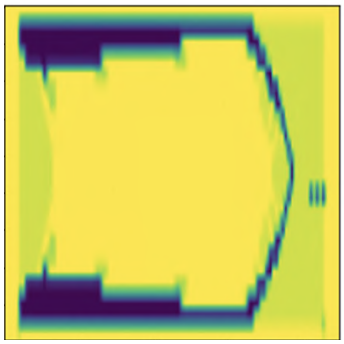
(g) TO simulation with the load applied in x:50 y:50

Figure 5.5: Results of the TO simulations (left). Processed pictures to enter the GAN (right)

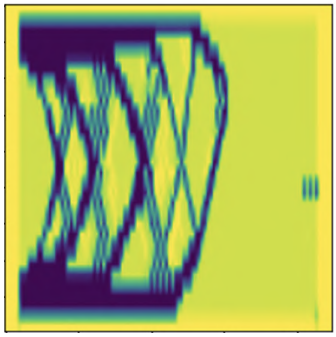
5.1.3 Databank Created for Training the CNN

As mentioned before, in contrast to the GAN, the CNN was trained with higher quality pictures to be able to learn more patterns, and therefore, be able to perform a better classification. For this reason, the pictures on the left column from the figure 5.5, were resized into a quadratic shape with a size of $224 \times 224 \times 1$ pixels to train the CNN. The figure 5.6 provides with an example of the reshaped images used to train the CNN.

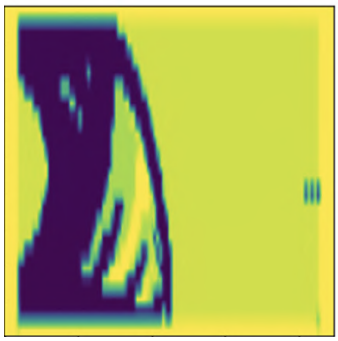
(a) Figure 5.5a adapted for CNN



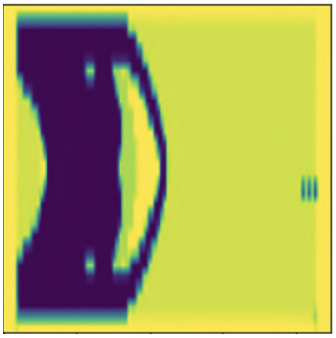
(b) Figure 5.5b adapted for CNN



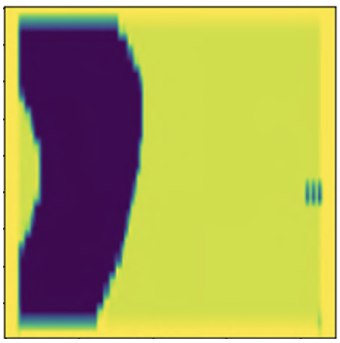
(c) Figure 5.5c adapted for CNN



(d) Figure 5.5d adapted for CNN



(e) Figure 5.5e adapted for CNN



(f) Figure 5.5f adapted for CNN

Figure 5.6: Processed pictures to train the CNN

5.2 Discussion of the Databank Results

First, the TO figures computed, achieved the necessary requirements to converge, an therefore, minimize the compliance and satisfy the constraints. After analyzing the figures, that were used in the GAN and the CNN, it was concluded that two main improvements can be implemented to obtain better quality results. First, the structure of the cantilever beam is characterized by its substantial length in comparison to its width. When analyzing the dataset employed for training both, the GAN and the CNN, it was observed that the reshaping of images resulted in a significant degradation of the information content, i.e quality. This can be perceived when comparing the images in figure 5.6 and the right column of figure 5.5 with the ones of the original database, i.e left column from figure 5.5.

It can be concluded that the needed quadratic shape, for entering the NN, was not suitable for keeping all the details of the structure. Therefore, three different options could be implemented to improve these results. First, an increase of the quality of the images to train the NN

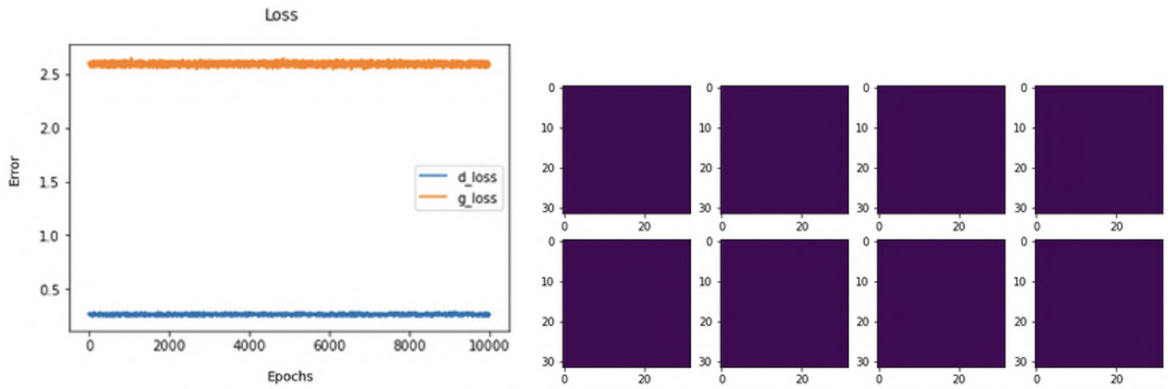
could be performed. As it can be observed, the images employed to train the CNN had higher quality than the ones for the GAN. Therefore, improving the quality of the images, resulted in an increase of the details. Secondly, a different configuration, changing the dimensions of the beam into a more squared shape, would reduce the compression of it, and therefore, reduce the loss of information. Finally, a division of the final structure into multiple squared figures could be implemented. By doing this, two or three figures could be generated to obtain the final structure of a the TO beam. However that would turn into an increase of the computational cost due to the generation of multiple images for a beam. These alternatives are further explained in the *Future works* section in 6.2.

5.3 Results of the GAN's Image Generation

Before validating the GAN, the databank of pictures, obtained from the previous section, was classified into training, testing and validating sets. Then, the GAN was programmed with the needed layers, as explained in the section 4.3. Once everything was ready, the validation of the GAN was conducted. During the compilation process, three different problems were assessed. For this reason, this section provides how the problems were identified, and how they were solved by analyzing the behavior of the generator loss (*g_loss*) and the discriminator loss (*d_loss*).

5.3.1 Converging the GAN

The first time the GAN was computed, the *g_loss* and the *d_loss* were far from each other and there was no convergence, as it can be observed on the graph in the figure 5.7. While the discriminator was achieving good performances with an error around 0.27, the generator was fixed around a loss of 2.57. This situation was discussed in section 2.2.9, and it refers to the scenario where the discriminator learns much faster than the generator and, consequently, does not allow acquiring the needed patterns to generate new images. For this reason, when computed the generated figures they were completely dark, i.e., not generated figures.

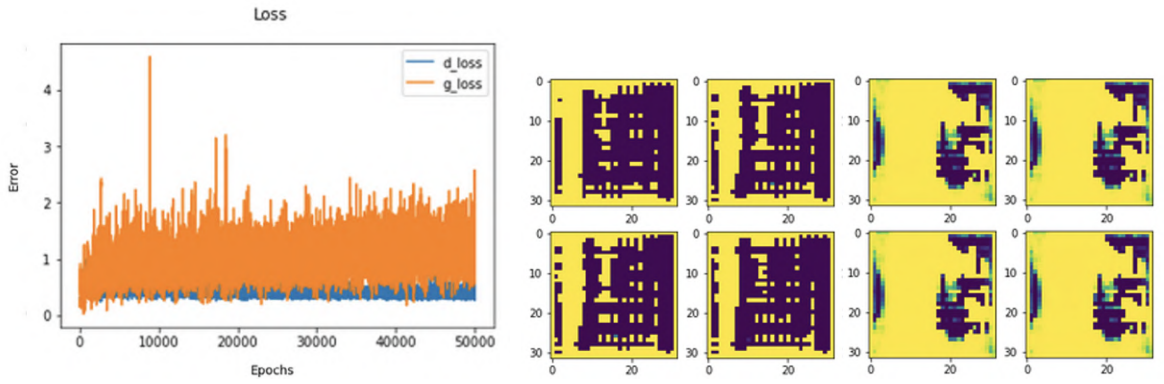


(a) Obtained generator and discriminator loss **(b)** Example of eight generated figures

Figure 5.7: First type of problem validating the GAN

With the aim of solving this problem, two methods where implemented. First, noise was added to the pictures with the command *tf.random.uniform*. This method was implemented

by adding both, a random positive noise and a random negative noise to the pictures of the databank. Secondly, the learning rate of both the generator and the discriminator were modified. By increasing the capacity of the learning rate of the generator and decreasing the discriminator, the loss from the generator was minimized getting closer to the discriminator loss. Several attempts were implemented varying among different new values of learning rates until the loss was improved. At the end, the final learning rate assigned to the generator was $1e^{-4}$ and the discriminator $1e^{-5}$. In other words, the generator learned ten times faster than the discriminator in this GAN. After performing this changes, the graph of the loss and the compiled figures obtained are provided in figure 5.8.



(a) Obtained generator and discriminator loss

(b) Example of eight generated figures

Figure 5.8: Second type of problem overfitting the GAN

When analyzing the pictures reproduced by the generator in figure 5.8, it can be seen that these were not completely random noise. Besides not being the desired output, they were having a similarity with some of the pictures provided in the data bank. More precisely when analyzing the subfigures in figure 5.9 it can be observed that the left part of the first three images, from the data bank, was reproduced in the fourth image when asking the generator to provide an image. Therefore, the learning rates were not further changed.

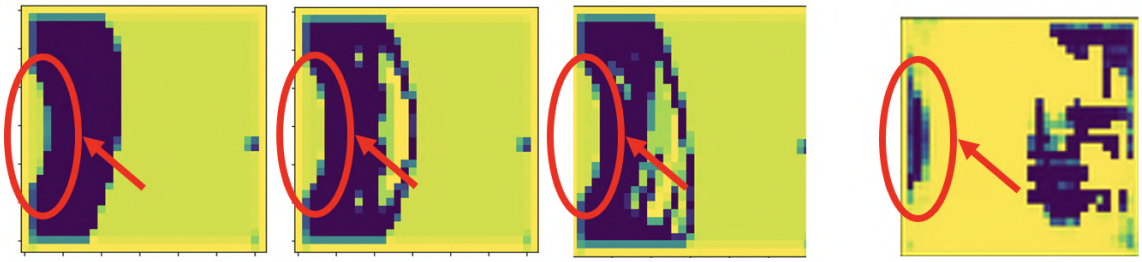


Figure 5.9: Similarities found when generating pictures after implementing noise pictures and varying the learning rates. The first three images corresponds to the provided databank and the last figure corresponds to a generated figure.

Examining the result obtained in the figure 5.8, it can be observed that both losses were closer than in the previous result. However, the loss of the generator was oscillating between the values 0.5 and 2.0, while the discriminator was remaining still around the value 0.3. For this reason, a new parameter should be modified to make the NN converge and provide results. For this reason, based on the paper work from LEE et al. (2017a) Adam optimization algorithm was employed. Once the model compiled for 11000 epochs it stabilized obtaining

an error between 0.4 and 0.6 for both the generator and the discriminator, the generator being slightly lower than the generator as it can be observed in the figure 5.10. After this moment, it was tried to improve the model by alternating other parameters but no further variable made the generator and the discriminator minimize its loss value. Therefore, this is the final convergence accepted for the model in order to generate the figures of the topology optimized cantilever beams.

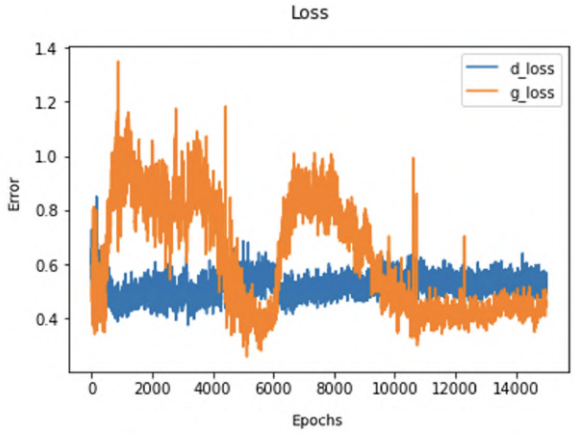


Figure 5.10: Final convergence of loss functions for the generator and discriminator

5.3.2 Learning Path and Training Algorithm

The first similarities observed in the generated images compared to the provided databank appeared after the epoch number 4000. As noticed in the figure 5.10, that was the first moment where the generator loss was under the discriminator. These similarities can be observed in the figure 5.11. The pictures in black and white are the $32 \times 32 \times 32 \times 1$ pixel figure generated, and the pictures in color, the real databank.

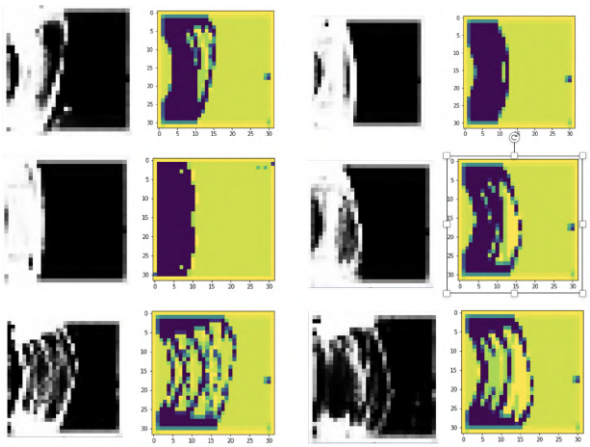


Figure 5.11: First similarities observed after 4000 epochs

5.3.3 Examples of the Final Generated Images

This section provides examples of the final generated figures. The generation of a figure was random such as the figures were trained in the GAN without its respective classification label. The figure 5.12 provides twenty illustrations of generated figures. These are represented in a manner that the yellow part is the optimized cantilever beam and the purple part one is the background. This option was selected to differentiate with different colors from the real databank from the figure 5.5. Each of these images was able to be generated in 35 ms.

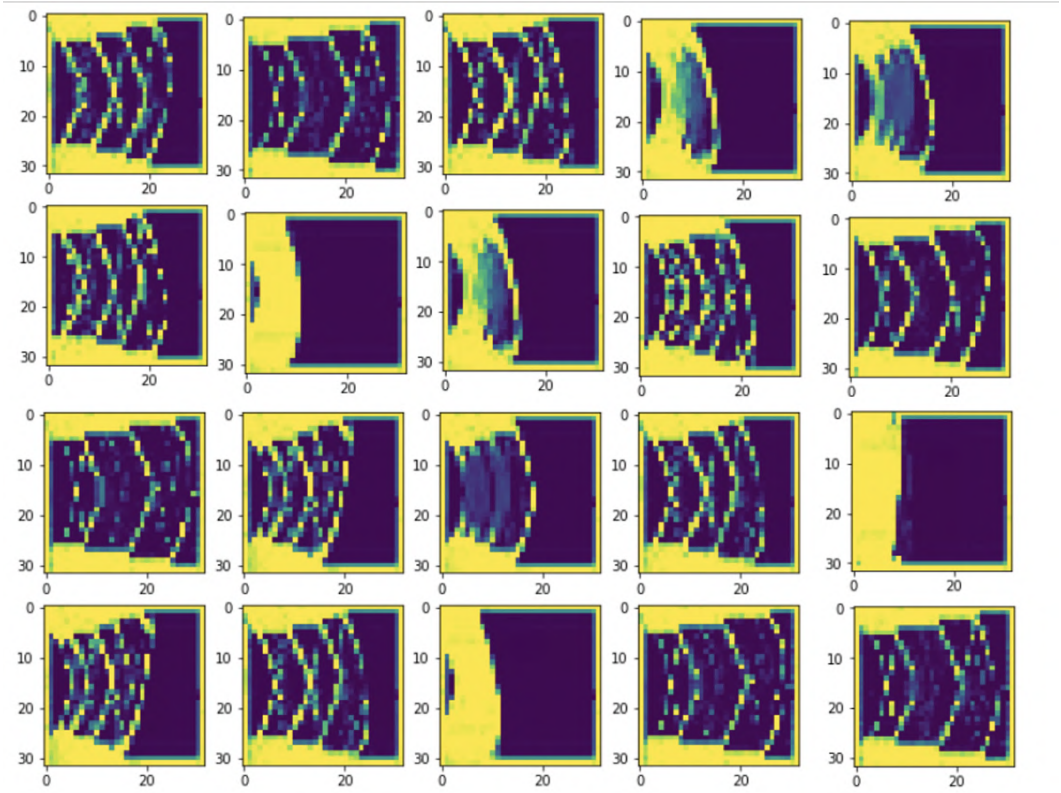


Figure 5.12: Examples of generated figures by the GAN.

As it can be observed, these figures imitate the real ones. However, some of them contain some noisy parts as the one represented in the figure 5.13. The quality has in other words decreased in some cases compared to the real databank.

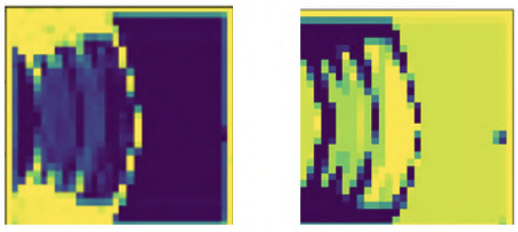


Figure 5.13: Comparison between a figure generated in the GAN (left) and the figure from the databank (right)

5.4 Discussion of the GAN Results

The time needed to generate a figure with this GAN was 35 ms. The computational time used to run a TO simulation in Hyperworks was ten minutes. This means that the figures were able to be generated 17 000 times faster than when computing it in the conventional SIMP methodology. It can therefore be concluded that this methodology is able to greatly reduce the computational cost. The figures computed in Hyperworks contained however more details that the one generated in the GAN. For this reason, a future study should be implemented to compare the two methods when the output obtained is more similar. Different alternatives for the improvement of the databank can be found in the section 5.2.

When analyzing the generation of figures, it can be concluded that the provided databank was correctly shuffled and provided to the GAN. This can be assumed as the generation of images is completely random and all of them are different.The lower quality of the GAN's output figures could be caused by the error in the model convergence in figure 5.10. The more this error can be minimized, the more similar would the generated figures be, compared to the real databank.

The parameters were varied until the first results of convergence of the GAN were obtained. When a sensibility analysis wanted to be performed, the error *DNN library is not found* in charge of building the architecture of the GAN, appeared in each compilation. This occurred due to an internal error in the computer that could not be solved. For this reason, the study of this GAN could not be further analyzed. The parameters that could have been further modified to obtain a better solution were the learning rates, the architecture of the layer and the optimization algorithms.

5.5 Results of the CNN's Classification of the Databank

This section provides the results obtained for the different CNNs programmed. As mentioned before, the number of classifications whether the load was applied in the cantilever beam was maximized. However, the desired number of classifications was not achieved as first established. As it was analyzed in the figure 4.11, the complexity of obtaining a higher classification of the position of the load increased when the beam was resized into a square. Therefore, the results obtained for the two analyzed cases and the process to maximize the number of categories in which a figure can be classified are shown in this section.

5.5.1 Validating the CNN in CASE 1

The CNN programmed to sort the figures in two different categories converged. The CNN was programmed as explained in the section 4.4. The convolutional and the max pooling layers were added to the model together to the final dense layer and the sigmoid function. The model converged after three epochs of 42 steps each. The convergence of the model can be observed in the figure 5.14. The model was not further compiled as the loss stabilized in the epoch number 3 with a loss under $1e^{-3}$ and the accuracy reached the maximum in both training and validation set after the first epoch. Consequently, the next step was the study of the classification in three classes.

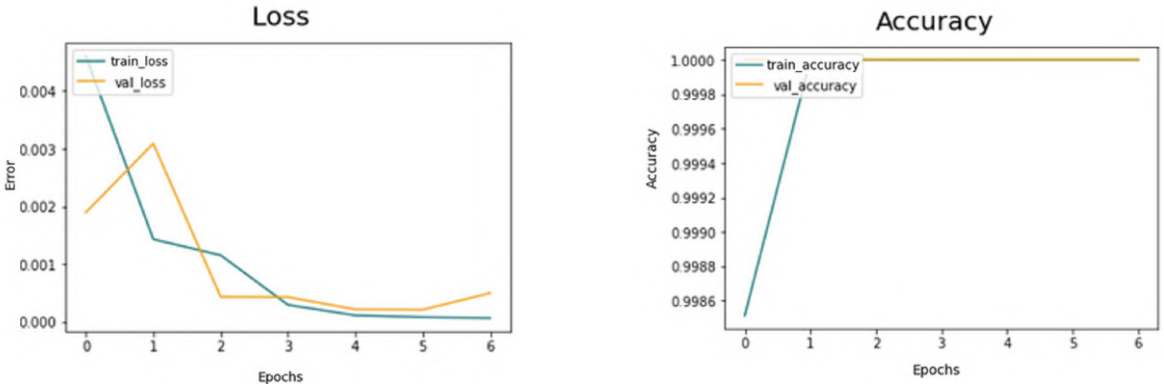


Figure 5.14: Convergence obtained classifying 2 clases with the CNN

5.5.2 Method to Converge the NN Implemented

Converging the CNN while avoiding underfitting and overfitting is challenging. As it will be observed in the figure 5.15, the validation loss was increasing for each epoch in Case 2. For this reason, a method to converge this NN was implemented. This subsection explains the different issues that a CNN could present and how they should be addressed. The problems, and corresponding solutions, in a CNN are described below based on KELLER et al. (2016).

- **High error in the training set.** In this case, the model is too simple and there is a lack of parameters to train the NN. Possible solutions includes increasing the model, training for longer time or expanding the architecture of the model.
- **High error in both the training and validation set.** In this instance, three alternatives could be implemented. Either more data to train the NN, regularization techniques or a new model architecture should be implemented.
- **High error in validation set.** Given this example, three methods can minimize the issue. First, making the data from the validation set similar to the training set. Second, performing a data synthesis of the databank. Or third, modifying the architecture of the model.
- **High error when testing data.** Finally, this could occur because the validation dataset is overfitting. Therefore, a bigger validation set is needed to evaluate the performance of the model.

5.5.3 Validating the CNN in CASE 2

The methodology described in the previous section was implemented to improve the convergence. The first architecture of the CNN was programmed as shown in the figure 4.15.

While both the loss and the accuracy from the training set were converging, the validation was not. As the figure 5.15 displays, the parameter *validation loss* increased while the *validation accuracy* remained stable under the value 0.70. In order to solve this, a solution needed to be implemented to find a better classification of the CNN, and therefore, the third method of the section 5.5.2 was implemented. Given that modifying the databank was not

considered, the architecture of the model was varied with the parameters provided in the first column of the table 5.1. Additionally, the solutions obtained after each compilation were detailed.

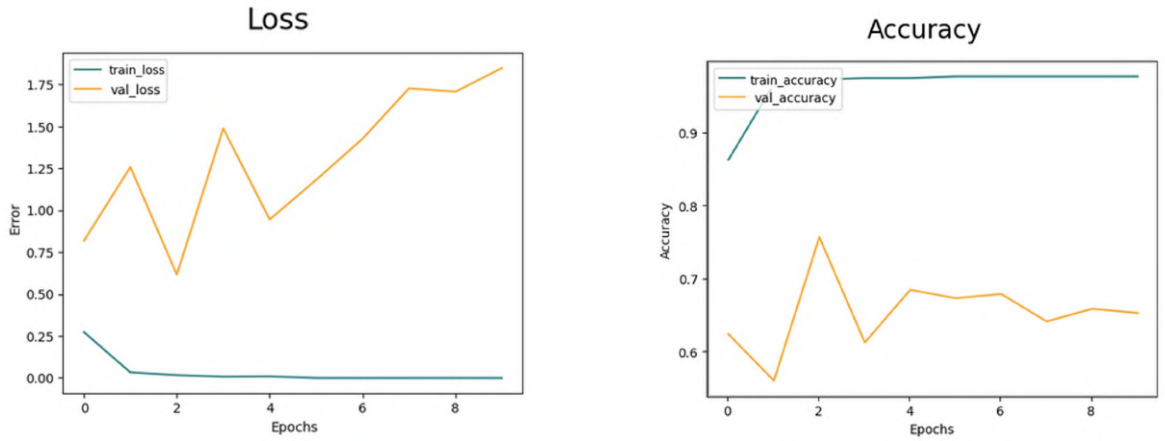


Figure 5.15: Convergence obtained classifying 3 classes with the CNN

Changes implemented	Validation loss	Validation accuracy
Add Batch Normalization	Random oscillated	Random oscillated
Add Dropout	Oscillated to bigger values each iteration	Random oscillated
Change compiler to RMSprop	Random oscillated	Oscillated between 0.6 and 0.8
Add Conv2D with 256 filters	Oscillated between values 1 and 2	Oscillated around 0.7
Change last Conv2D to 32 filters	Oscillated between values 1 and 2	Oscillated around 0.7
Delete the last block (Conv2D 128)	Oscillated between smaller values	Decreased between 0.7 and 0.6
Delete the last two blocks	Oscillated between 0.6 and 1.5	Oscillated between 0.8 and 0.6

Table 5.1: Different set ups implemented in the architecture of the CNN in the figure 4.15 to converge the model

When it was observed that none of the changes made the CNN converge, a new method was considered. The second option described in 5.5.2 was implemented to minimize the growing oscillating error in the validation set. The data augmentation method previously considered in each of the CNNs programmed in table 5.1 was the normalization of the figures with the command *rescale=1./255*. The new additional methods of data augmentation were implemented (MARTÍN ABADI et al. 2015).

- The command *rotation_range=20* was implemented to rotate the images randomly up to 20 degrees.
- The command *width_shift_range=0.2* shifted the width of the images randomly.
- The command *height_shift_range=0.2* modified the height of the images randomly.
- The command *shear_range=0.2* applied shear transformations randomly.
- The command *zoom_range=0.2* altered randomly the zoom of the images.
- The command *horizontal_flip=True* flipped the images horizontally randomly.
- A reduction of the validation set from 20 % of the databank to a 15 % was implemented to obtain more percentage of the images in the training set and therefore, have more parameters to validate the set and reduce the loss.

The last architecture in table 5.1 achieved less validation loss. Therefore, the same parameters were computed with the data augmentation for possible improvements. After computing it, less oscillations were observed. Therefore, the structure was further simplified with the aim of converging the CNN. The changes in table 5.2 were implemented and analyzed.

Changes implemented	Validation loss	Validation accuracy
Original configuration from figure 4.15	Oscillated between 0.6 and 0.9	Oscillated between 0.8 and 0.6
Reduction to 8 filters	Less oscillations than in the previous case	
Change Dense layer to 10	Oscillated between 0.15 and 1.45	Decreased from 0.9 to 0.4
Increase the size batch	Decreased from 0.9 to 0.4	Oscillated between 0.6 to 0.8

Table 5.2: Different change in the architecture after implementing the synthesis of the databank

Consequently, the best structure obtained to train the CNN, contained one block with a convolutional layer with 8 filters, a max pooling layer, and two dense layers with 10 and 3 layers. The exact parameters employed to converge the CNN can be observed in the figure 5.16. Moreover, the results of the convergence of the loss and the accuracy are provided as well in the figure 5.17.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 146, 146, 8)       2408

max_pooling2d_10 (MaxPoolin  (None, 14, 14, 8)         0
g2D)

flatten_9 (Flatten)          (None, 1568)              0

dense_18 (Dense)             (None, 10)                15690

dense_19 (Dense)             (None, 3)                 33

=================================================================
Total params: 18,131
Trainable params: 18,131
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.16: Final CNN used to classify into 3 different groups

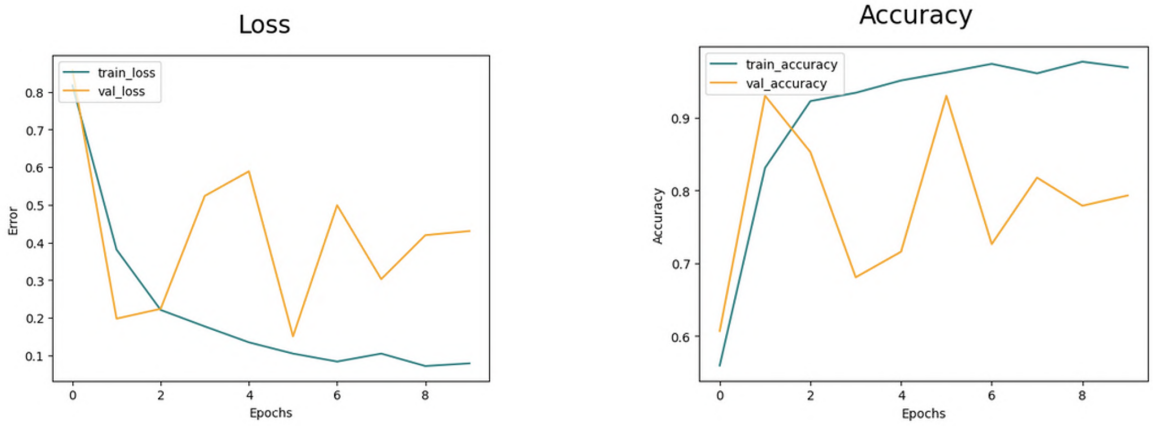


Figure 5.17: Best convergence when modifying the CNN used to classify into 3 different groups

As can be seen in 5.17 the results did not converge as smooth as in the case 1, in the figure 5.14. But among all the CNNs programmed, the graphics plotted in figure 5.17 were having less oscillations. Therefore, more implementations should be added to perform better results.

5.6 Discussion of the CNN Results

The purpose of this section was to achieve the classification the position of the load in the beam in multiple classes. First, a classification of the load in two positions, or CASE 1, was achieved and converged. When classifying the data in three different positions, the CNN did not converge as smooth as the previous case. The problem was located in the validation set. The values that helped the convergence into smaller error values for the loss, and higher values for the accuracy were as follows.

- Simplification of the architecture by deleting architecture blocks
- Implementation of the regularization techniques to modify the databank
- Reduction of the filters used in the convolutional layers
- Minimization of the size of the dense layer
- Increase of the size of the batch

All these modifications, contributed to the reduction and the simplification of the architecture. This way the third implementation in section 5.5.2 could help the CNN achieve better performance when validating the set. However it can be observed that neither the validation accuracy nor the validation loss, were converging in any of the cases. Two out of the three implemented methods did not achieve the desired output. Because of this, the last solution of increasing the databank of figures should be tried to see if better results can be achieved.

Chapter 6

Conclusion and Future Work

This section summarizes all the goals achieved, the steps employed to improve the models and the future work that could be implemented. As discussed, at the beginning of this master thesis, the advances performed in TO and in AI are increasing everyday. Therefore, the research should be continued in this topics to implement better functions. There are numerous opportunities for the implementation of AI in structural optimization and this thesis is only a portion of that exploration.

6.1 Conclusions

This master thesis aimed to get a deeper insight on how to combine NNs with TO. With this research, the techniques were implemented, the results validated and the accuracy of the implemented models maximized.

The principal objective was to automatize the process of generating the TO figures with the GAN and then classify the outcomes with the CNN. Therefore, a code in TCL automatized the TO simulations to obtain simulation results from Hyperworks. This allowed creating the needed databank for training the NNs. In the image processing stage, the figures were resized into squares to enter the GAN and the CNN. After this step, it can be concluded that the dimensions of the model were not the most adequate to resize it into a square. When modifying the size of the beam to be processed by the NNs, the figures were compressed resulting in a decrease of the quality of the figure.

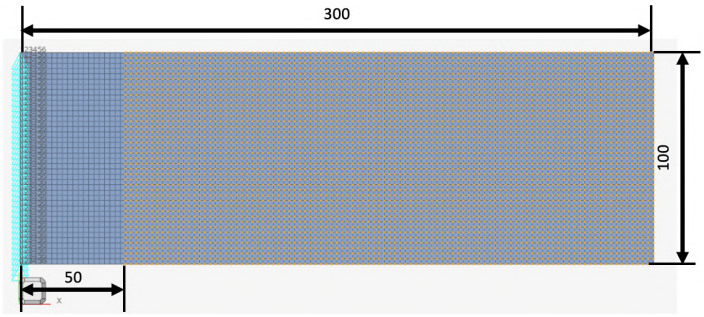
The GAN was trained and validated. The challenge in this section was to stabilize the loss of both the generator and the discriminator to be able to generate valid figures at the end. With the purpose of minimizing the loss, three different problems were addressed to obtain the desired result. The first time the NN was computed, the generator loss and the discriminator loss were far apart. Changing the learning rates of the generator and the discriminator, made the different respective losses come closer. The final rates employed were $1e^{-4}$ for the generator and $1e^{-5}$ for the discriminator. Afterwards, the generated figures started presenting some similarities to the real databank. However, the GAN was not converging because of the oscillating loss of the generator. Changing the optimizer algorithm from RMSprop to Adam solved this issue and converged the GAN. After this, the model was stabilized after 11000 epochs obtaining an error between 0.4 and 0.6 for both the generator and the discriminator. The final achieved result was a loss of the generator slightly lower than the one from the discriminator.

Analyzing the generated items, it can be concluded that some of them contained noise because the details were not completely precise. On the other hand, it was proven that the data was correctly analyzed and provided to the GAN. The created figures were able to be generated in 35 ms, which means a reduction of the computational cost 17 000 times faster compared to computing it in the conventional SIMP methodology in Hyperworks.

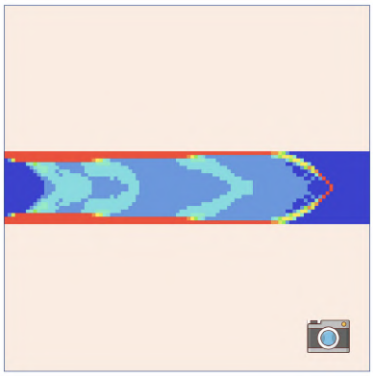
The main goal of the CNN to obtain the precise location of the load in the cantilever beam was not fully achieved. Nevertheless, the process to get to the goal was addressed and analyzed. The first goal was to be able to classify the figures in 21 classes for the Y axis and 91 for the X axis. However, this classification was not feasible due to the resize of the figure and consequently its loss of information. For this reason, the number of divisions was analyzed and maximized. The classification of the beam into two different categories according to the X axis was successfully achieved. The model converged after three epochs of 42 steps each. Then, the analysis into more than two different categories was not fully achieved. In this case, the parameters to simplify the structure of the CNN were varied in order to minimize the loss and maximize the accuracy. It can be concluded that the simplification of the architecture, the implementation of regularization techniques to modify the databank, and the increase of the batch size improved the convergence of the CNN.

6.2 Future Work

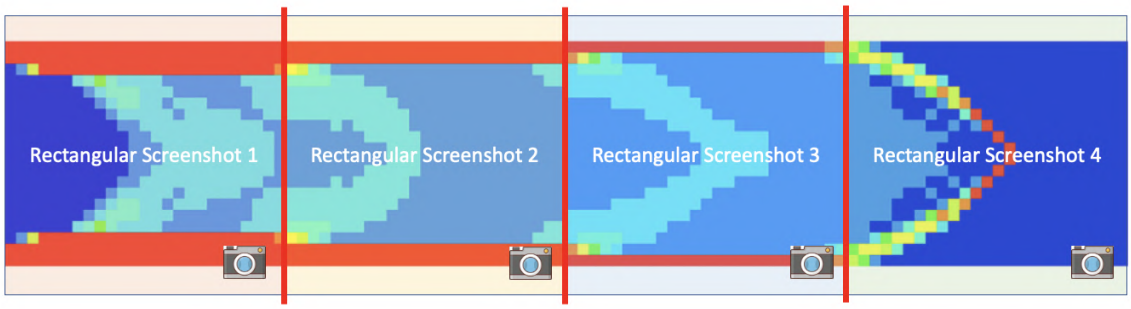
Analyzing all the goals achieved in this master thesis, this section aims to introduce future lines of investigation that can be conducted for further improvements and development.



(a) Different configuration of the beam,i.e. more quadratic



(b) Adding frame to retain the original beam shape



(c) Multiple screenshots of one result of simulation

Figure 6.1: Possible configurations for obtaining better results to be analyzed in future works

First, as said before, how the data bank is introduced in the NNs should be examined. As observed, resizing the shape of the simulated beam carries a lost of information. Therefore, three options can be implemented to improve the data used to train the NN and consequently, obtain better results. First, a new configuration of the cantilever beam, i.e., more squared, can be programmed so when the solution is resized, the lost of information is minimized. This can be observed in the figure 6.1a . Secondly, a quadratic screenshot of the original beam can be obtained, so that the result of the simulated beam would not be compressed as it is shown in the figure 6.1b. The last option to be considered is the segmentation of the actual beam. In other words, the result of the TO beam can be divided into squared screenshots that can be processed in the GAN together. As figure 6.1c is showing, each of the screenshots would be followed by a number, so both the screenshot and the position of the screenshot will be trained together in the GAN. This can be achieved by creating a conditional GAN (CGAN).

Regarding the programming of the GAN, three improvements have been found that would make the NN enhance the desired results. First, it can be observed that the generated figure size of 32×32 pixels entails a big lost of information. Therefore, the first improvement that must be consider is a refinement of the image generation. In other words, figures with higher quality should be generated to be able to use them. Second, a GAN can be trained with more parameters. By implementing this, a new CGAN can be programmed so that the figures can be trained and sorted. The result would be a generator able to provide figures with its corresponding classification. Finally, the last consideration to be taken into account is the creation of a methodology to generate 3D configurations. As analyzed, TO is a complex tool that can save material in a structure. When analyzing the structures in 3D the complexity increases, but the applications are much bigger than in 2D.

Finally, when analyzing the CNN it can be observed that the desired maximum classification of the position of the load could not be achieved. Therefore, trying different methods such as increasing more the databank, or changing the layers of the CNN should be studied.

Chapter 7

Sustainable Development Goals

This master's thesis is linked to the Sustainable Development Goals (UNITED NATIONS and DEVELOPMENT 2015). A list of seventeen targets, defined in 2015, aims to minimize the world's hazards present nowadays. The figure 7.1 shows all of them. The principal objective is to address each of them before 2030 such that for example, poverty can be eradicated, the Earth can be protected, or the well-being of humanity can be improved among others. Between all the objectives achieved in this master's thesis, the sustainable development goals addessed in this project are *Number 9, "Industry innovation and infrastructure"*, and *Number 12, "Responsible, consumption and production"*.

Figure 7.1: Sustainable Development Goals. Source: UNITED NATIONS and DEVELOPMENT (2015)

Nowadays, research is making great attainments in most fields of study. In this project, the TO was analyzed observing how the infrastructures could minimize the mass without risking

the security of it. This decrease of the material is directly correlated with target number 12.5 "*Substantially reduce waste generation*". By reducing the quantity implemented, waste of non-used would be prevented.

Moreover, applying the constraints for the TO , ensures the resilience of the structure. By achieving this, the target number 9.1 "*Develop sustainable, resilient and inclusive Infrastructures*" is accomplished. Implementing save structures ensures human well-being, as well as good accessibility for all.

Figure 7.2: Targets to be achieved by implementing TO

Moreover, the implementation of AI proved the minimization of computational time. Therefore, less energy, consumed by computers, is needed to obtain the same results. This would result in an extension of the lifespan of computer batteries. For this reason, the target 12.4 "*Responsible management of chemicals and waste*" is as well achieved.

Furthermore, the files with every necessary command programmed from the NN occupies much less capacity compared to any software that can perform a simulation. This fact makes the distribution of information easy in the least developed countries. By obtaining this reduction of storage files, the target 9.8 "*Universal access to information and communications technology*" is achieved.

Figure 7.3: Targets to be achieved by implementing AI

Appendix A

Details of the GAN

Layer (type)	Output Shape	Param #
dense_8 (Dense)	(None, 2048)	264192
leaky_re_lu_39 (LeakyReLU)	(None, 2048)	0
reshape_4 (Reshape)	(None, 4, 4, 128)	0
up_sampling2d_14 (UpSampling2D)	(None, 8, 8, 128)	0
conv2d_39 (Conv2D)	(None, 8, 8, 128)	409728
leaky_re_lu_40 (LeakyReLU)	(None, 8, 8, 128)	0
up_sampling2d_15 (UpSampling2D)	(None, 16, 16, 128)	0
conv2d_40 (Conv2D)	(None, 16, 16, 128)	409728
leaky_re_lu_41 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_41 (Conv2D)	(None, 16, 16, 128)	262272
leaky_re_lu_42 (LeakyReLU)	(None, 16, 16, 128)	0
up_sampling2d_16 (UpSampling2D)	(None, 32, 32, 128)	0
conv2d_42 (Conv2D)	(None, 32, 32, 128)	262272
leaky_re_lu_43 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d_43 (Conv2D)	(None, 32, 32, 128)	262272
leaky_re_lu_44 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d_44 (Conv2D)	(None, 32, 32, 128)	262272
leaky_re_lu_45 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d_45 (Conv2D)	(None, 32, 32, 1)	2049

Total params: 2,134,785
Trainable params: 2,134,785
Non-trainable params: 0

Table A.1: Network programmed for the generator

Layer (type)	Output Shape	Param #
conv2d_46 (Conv2D)	(None, 28, 28, 32)	832
leaky_re_lu_46 (LeakyReLU)	(None, 28, 28, 32)	0
dropout_20 (Dropout)	(None, 28, 28, 32)	0
conv2d_47 (Conv2D)	(None, 24, 24, 64)	51264
leaky_re_lu_47 (LeakyReLU)	(None, 24, 24, 64)	0
dropout_21 (Dropout)	(None, 24, 24, 64)	0
conv2d_48 (Conv2D)	(None, 20, 20, 128)	204928
leaky_re_lu_48 (LeakyReLU)	(None, 20, 20, 128)	0
dropout_22 (Dropout)	(None, 20, 20, 128)	0
conv2d_49 (Conv2D)	(None, 16, 16, 256)	819456
leaky_re_lu_49 (LeakyReLU)	(None, 16, 16, 256)	0
dropout_23 (Dropout)	(None, 16, 16, 256)	0
flatten_4 (Flatten)	(None, 65536)	0
dropout_24 (Dropout)	(None, 65536)	0
dense_9 (Dense)	(None, 1)	65537

Total params: 1,142,017
Trainable params: 1,142,017
Non-trainable params: 0

Table A.2: Network programmed for the discriminator

Bibliography

ALTAIR ENGINEERING, INC., (2017). *HyperMesh*. Version 2021.1. URL: https://altairhyperworks.com/product/HyperMesh.

CHANDRASEKHAR, A. and SURESH, K., (Mar. 2021). "TOuNN: Topology Optimization using Neural Networks". In: *Structural and Multidisciplinary Optimization* 63, pp. 1–15. DOI: 10.1007/s00158-020-02748-4.

GOODFELLOW, I. J., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., and BENGIO, Y., (June 2014). "Generative Adversarial Networks". In: URL: https://arxiv.org/abs/1406.2661 (visited on 01/08/2017).

HEATON, J., (June 2018). "Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep Learning". In: *Genetic Programming and Evolvable Machines* 19.1–2, pp. 305–307. ISSN: 1389-2576. DOI: 10.1007/s10710-017-9314-z. URL: https://doi.org/10.1007/s10710-017-9314-z.

KELLER, J., LIU, D., and FOGEL, D., (2016). *Fundamentals of Computational Intelligence: Neural Networks, Fuzzy Systems, and Evolutionary Computation*. IEEE Press Series on Computational Intelligence. Wiley. ISBN: 9781119214342. URL: https://books.google.de/books?id=0Xz9sgEACAAJ.

KINGMA, D. and BA, J., (Dec. 2014). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*.

LECUN, Y. and CORTES, C., (2010). "MNIST handwritten digit database". In: URL: http://yann.lecun.com/exdb/mnist/.

LEE, S., HA, J., ZOKHIROVA, M., MOON, H., and LEE, J., (July 2017a). "Background Information of Deep Learning for Structural Engineering". In: *Archives of Computational Methods in Engineering*. DOI: 10.1007/s11831-017-9237-0.

— (July 2017b). "Background Information of Deep Learning for Structural Engineering". In: *Archives of Computational Methods in Engineering*. DOI: 10.1007/s11831-017-9237-0.

MARTÍN ABADI, ASHISH AGARWAL, PAUL BARHAM, EUGENE BREVDO, ZHIFENG CHEN, CRAIG CITRO, GREG S. CORRADO, ANDY DAVIS, JEFFREY DEAN, MATTHIEU DEVIN, SANJAY GHEMAWAT, IAN GOODFELLOW, ANDREW HARP, GEOFFREY IRVING, MICHAEL ISARD, JIA, Y., RAFAL JOZEFOWICZ, LUKASZ KAISER, MANJUNATH KUDLUR, JOSH LEVENBERG, DANDELION MANÉ, RAJAT MONGA, SHERRY MOORE, DEREK MURRAY, CHRIS OLAH, MIKE SCHUSTER, JONATHON SHLENS, BENOIT STEINER, ILYA SUTSKEVER, KUNAL TALWAR, PAUL TUCKER, VINCENT VANHOUCKE, VIJAY VASUDEVAN, FERNANDA VIÉGAS, ORIOL VINYALS, PETE WARDEN, MARTIN WATTENBERG, MARTIN WICKE, YUAN YU, and XIAOQIANG ZHENG, (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

OPENAI, (2022). *Introducing ChatGPT*. URL: https://openai.com/blog/chatgpt (visited on 04/10/2023).

RASHID, T. and LANGENAU, F., (2020). *GANs mit PyTorch selbst programmieren: Ein verständlicher Einstieg in Generative Adversarial Networks*. O'Reilly, pp. 61–127. ISBN: 9783960103943. URL: https://books.google.de/books?id=ygL9DwAAQBAJ.

RAWAT, S. and SHEN, M. .-. H., (2019a). *A Novel Topology Optimization Approach using Conditional Deep Learning*. arXiv: 1901.04859 [cs.LG].

— (2019b). "Application of Adversarial Networks for 3D Structural Topology Optimization". In: *SAE Technical Paper Series*.

RENOTTE, N., (2022). *GANBasics*. https://github.com/nicknochnack/GANBasics/blob/main/FashionGAN-Tutorial.ipynb.

ROZVANY, G. I. N., (2009). "A critical review of established methods of structural topology optimization". In: *Structural and Multidisciplinary Optimization* 37, pp. 217–237.

ROZVANY, G. I. N., ZHOU, M., and BIRKER, T., (1992). "Generalized shape optimization without homogenization". In: *Structural optimization* 4, pp. 250–252.

SEWAK, M., KARIM, M., and PUJARI, P., (2018). *Practical Convolutional Neural Networks: Implement advanced deep learning models using Python*. Packt Publishing, pp. 1–45. ISBN: 9781788394147. URL: https://books.google.de/books?id=bOlODwAAQBAJ.

SIVA RAMA KRISHNA, L., MAHESH, N., and SATEESH, N., (2017). "Topology optimization using solid isotropic material with penalization technique for additive manufacturing". In: *Materials Today: Proceedings* 4.2, Part A. 5th International Conference of Materials Processing and Characterization (ICMPC 2016), pp. 1414–1422. ISSN: 2214-7853. DOI: https://doi.org/10.1016/j.matpr.2017.01.163. URL: https://www.sciencedirect.com/science/article/pii/S2214785317301633.

SOSNOVIK, I. and OSELEDETS, I., (2017). *Neural networks for topology optimization*. arXiv: 1709.09578 [cs.LG].

SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., and SALAKHUTDINOV, R., (Jan. 2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1, pp. 1929–1958. ISSN: 1532-4435.

TAKAHASHI, Y., SUZUKI, Y., and TODOROKI, A., (2019). *Convolutional Neural Network-based Topology Optimization (CNN-TO) By Estimating Sensitivity of Compliance from Material Distribution*. arXiv: 2001.00635 [cs.LG].

TUSHAR, A. K., ASHIQUZZAMAN, A., and ISLAM, M. R., (2017). "Faster convergence and reduction of overfitting in numerical hand sign recognition using DCNN". In: *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pp. 638–641. DOI: 10.1109/R10-HTC.2017.8289040.

UNITED NATIONS, D. o. E. and DEVELOPMENT, S. A. .-. S., (2015). *Transforming our world: the 2030 Agenda for Sustainable Development*. General Assembly. URL: https://www.globalgoals.org/goals/.

WANG, Y., OYEN, D., GUO, W., MEHTA, A., SCOTT, C. B., PANDA, N., FERNÁNDEZ-GODINO, M. G., SRINIVASAN, G., and YUE, X., (2020). "StressNet: Deep Learning to Predict Stress With Fracture Propagation in Brittle Materials". In: *CoRR* abs/2011.10227. arXiv: 2011.10227. URL: https://arxiv.org/abs/2011.10227.

WARR, K., (2019). *Strengthening Deep Neural Networks: Making AI Less Susceptible to Adversarial Trickery*. O'Reilly Media, pp. 180–212. ISBN: 9781492044925. URL: https://books.google.de/books?id=QKegDwAAQBAJ.

WEIDMAN, S., (2019). *Deep Learning from Scratch: Building with Python from First Principles*. O'Reilly Media, pp. 41–68. ISBN: 9781492041382. URL: https://books.google.de/books?id=MrWuDwAAQBAJ.

XIANG, C., DALEI, W., PAN, Y., CHEN, A., ZHOU, X., and ZHANG, Y., (Mar. 2022). "Accelerated topology optimization design of 3D structures based on deep learning". In: *Structural and Multidisciplinary Optimization* 65. DOI: 10.1007/s00158-022-03194-0.

YING, X., (Feb. 2019). "An Overview of Overfitting and its Solutions". In: *Journal of Physics: Conference Series* 1168.2, p. 022022. DOI: 10.1088/1742-6596/1168/2/022022. URL: https://dx.doi.org/10.1088/1742-6596/1168/2/022022.

ZHANG, Y., CHEN, A., PENG, B., ZHOU, X., and DALEI, W., (Jan. 2019). "A deep Convolutional Neural Network for topology optimization with strong generalization ability". In: p. 14.

ZUO, Z. H. and XIE, Y. M., (2015). "A simple and compact Python code for complex 3D topology optimization". In: *Advances in Engineering Software* 85, pp. 1–11. ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2015.02.006. URL: https://www.sciencedirect.com/science/article/pii/S0965997815000241.

Disclaimer

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Garching, June 1, 2023 (Signature)