UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

DETECCIÓN DE DAÑO EN UNA PLACA COMPUESTA UTILIZANDO VARIATIONAL AUTOENCODERS (VAE)

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MECÁNICO

OSCAR GABRIEL MAURIACA FLORES

PROFESOR GUÍA:
VIVIANA MERUANE NARANJO

MIEMBROS DE LA COMISIÓN:
ENRIQUE LÓPEZ DROGUETT
KARIM PICHARA CARTES

Este trabajo ha sido parcialmente financiado por Proyecto Fondecyt 1170535

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: OSCAR GABRIEL MAURIACA FLORES
FECHA: 2019
PROF. GUÍA: VIVIANA MERUANE NARANJO

DETECCIÓN DE DAÑO EN UNA PLACA COMPUESTA UTILIZANDO VARIATIONAL AUTOENCODERS (VAE)

Para cada nuevo proyecto de ingeniería, siempre existirá la duda sobre que materiales son los más aptos para cumplir los requerimientos de este, buscando la mayor confiabilidad y propiedades mecánicas al menor peso. Muchas veces el peso es algo que debe sacrificarse, teniendo estructuras más pesadas y robustas de lo deseado originalmente, pero en industrias donde el peso es algo fundamental como lo es la aeronáutica, la industria aeroespacial o la automotriz, el peso del producto es de los parámetros más importantes a la hora de evaluar su desempeño.

Los materiales compuestos son aquellos materiales formados por la unión de uno o más materiales diferentes, para conseguir propiedades mecánicas que no era posible con los materiales originales. En el caso de las placas compuestas tipo sándwich del presente estudio, estas placas corresponden a una unión entre dos capas de fibra de carbono, con un núcleo Nomex tipo panal de abeja y una resina epóxica como adhesivo entre las partes.

Las placas compuestas tipo sándwich tienen buenas propiedades mecánicas a un muy bajo peso. Sin embargo, por temas de manufactura poseen muchas veces defectos denominados delaminación. La delaminación se define como la perdida de adherencia entre el núcleo y las capas. Existe una gran dificultad en dar cuenta cuando existe esta falla, debido a que normalmente no es visualmente inspeccionable. Es por esto que se desarrollan herramientas computacionales para poder identificarlas de forma automática.

En el presente documento, se investiga una forma de detectar el daño en una placa compuesta, utilizando una red neuronal denominada Variational Autoencoders (VAE), redes de clasificación binaria de fallas y clasificación de fallas por niveles. La entrada a esta red serán imágenes de índices de daño sobre la superficie de las placas. Las que se obtienen mediante un método de elementos finitos basado en la curvatura de los modos de vibración.

La primera parte de este trabajo corresponde a una revisión bibliográfica sobre los métodos de aprendizaje supervisado de máquinas utilizados para resolver este tipo de problemas. Luego, se realiza un ordenamiento y preprocesamiento de los datos para dividirlos en los conjuntos de entrenamiento, prueba y validación. Con los datos se procede a la fase de entrenamiento y puesta a punto de las diferentes redes neuronales utilizadas, para luego analizar los resultados de las diferentes arquitecturas y modelos probados.

Se concluye que el VAE es un algoritmo no supervisado útil para la clasificación de las placas con daño, teniendo una efectividad de hasta un 86.89 %. Sin embargo también se proponen métodos de clasificación binaria con resultados de hasta un 92.84 % de efectividad, y luego métodos de clasificación multiclase según tamaño de daño con efectividad de un 75.44 %.

A todos los perritos

Agradecimientos

Gracias a mi madre por su amor incondicional en todos estos largos años de universidad, gracias por su valentía y su fortaleza emocional sobre todo bajo las diferentes adversidades que nos propuso la vida. Gracias a mi padre por la educación y por siempre confiar en mí, como me encantaría ver su cara de felicidad al saber que finalmente terminé mi carrera. Gracias a mis hermanos por ayudarme cada vez que se los pedí y por la buena onda constante.

Agradezco de corazón a mis amigos del colegio que también me acompañaron en esta aventura: Manuel, Sebastián, Patricio, Edita. Por todas las juntas hasta las mil de la mañana, todas las cervezas compartidas, todos los juegos y todos los viajes que realizamos juntos, definitivamente me entregaron las energías necesarias para llegar a donde estoy.

Doy gracias a mis amigos de la universidad que hicieron de este espacio un lugar mejor para mi y para el resto. Destaco a mis amigos de sección como a Patrick, Gianfranco, Josefo, Javiera y Cony que sin ellos el primer año hubiese sido imposible. A Ignacio (José) y Tamara por la buena onda, los jueguitos, los nuevos amigos y por ayudarme si estaba en aprietos. Agradezco también a Angelo por los muchos viajes en auto con buena conversación, las tardes de estudios, los trabajos hasta la madrugada y los muchos memes. Por último, doy gracias a Angélica por ser mi amiga y compañera todos estos años, apoyarme en los momentos más difíciles y siempre estar ahí cuando se necesitaba.

Finalmente agradezco a mi profesora Viviana Meruane por confiarme esta oportunidad. Trabajar en esta memoria de titulo me abrió los ojos sobre lo que me gustaría hacer en el futuro. Gracias a mi profesor Enrique López por la ayuda a la hora de resolver dudas.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Introducción

Un material compuesto se define como aquel material que se forman por la unión de uno o mas materiales, con el fin de conseguir propiedades mecánicas que no era posible obtener en los materiales que lo originan. La selección de estos materiales dependerá de la aplicación a la cual el material compuesto será sometido. En general se combinan los materiales originales con el fin de obtener un nuevo material mas rígido y resistente a un bajo peso.

Dentro de la familia de materiales compuestos, nos encontramos con los materiales compuestos estructurales. El resultado final y sus propiedades no solo depende de los materiales que lo componen sino que también depende de la geometría y del diseño utilizado. La estructura tipo sándwich es parte de esta familia de materiales compuestos, se compone de dos placas (superior e inferior), un núcleo y un adhesivo. Estos materiales pueden presentar una falla denominada delaminación, que puede producirse por cargas de impacto o por defectos en la manufactura del material. La delaminación corresponde a la perdida de adherencia entre las placas y el núcleo, lo que conlleva a perdidas de propiedades mecánicas del material y puede incluso ocasionar una falla catastrófica en este.

Los materiales compuestos tipo sándwich se utilizan en industrias como la automotriz, aeroespacial, sistemas eólicos, puentes y otros. Debido a que estas industrias requieren de elevados niveles de confiabilidad, es indispensable detectar tempranamente la presencia de daños y controlar la vida útil de estos materiales.

Para el análisis de daño en estructuras y partes mecánicas, es común utilizar la medición de vibraciones mecánicas. Cuando se presenta daño del tipo delaminación, la integridad estructural se ve afectada. Se producen cambios en las propiedades físicas como la rigidez y amortiguamiento, y como consecuencia se producen cambios en la respuesta dinámica del sistema, por ejemplo en las frecuencias naturales.

Para la identificación de daño por delaminación se utilizan algoritmos de aprendizaje de maquinas (Machine Learning), utilizando como entrada imágenes generadas por un algoritmo de elementos finitos que modelan placas con y sin daño, basándose en la curvatura de los modos de vibración. En particular, se utilizan redes neuronales artificiales conocidas como perceptrón multicapa o MLP (Multilayer Perceptron) y redes neuronales convolucionales o CNN (Convolutional Neural Network).

Combinando las redes MLP y las CNN se pueden formar una gran cantidad de arquitecturas posibles para realizar diferentes tareas. Para la realización de este trabajo de título se utilizan algoritmos de clasificación multiclase, clasificación binaria y clasificación de una clase

o One-Class Classification. La detección de anomalías es un caso especifico de una clasificación de una clase, en el cual se entrena un algoritmo que identifica objetos y patrones de los datos de entrenamiento (considerados datos normales), para luego probar con un conjunto de datos normales y anómalos y confirmar si el algoritmo es capaz de discriminar entre uno y otro. Se utilizan los Variational Autoencoder (VAE) tanto para clasificación multiclase como para la detección de anomalías. El Variational Autoencoder combina la inferencia variacional con las redes neuronales y se utiliza para identificar de manera automática la existencia de daño de delaminación en placas de material compuesto tipo sándwich.

Objetivos

En esta sección se presentan los objetivos de este trbajo de título.

Objetivo General

El objetivo general de este trabajo es desarrollar y evaluar modelos computacionales capaz de detectar daño en placas de material compuesta de forma automatizada, utilizando redes neuronales MLP, CNN y Variational Autoencoders.

Para lograr este objetivo general se pretende cumplir con los siguientes objetivos específicos

Objetivos Específicos

- Generación de una base de datos de imágenes con distintos escenarios de daño
- Diseño de redes neuronales de clasificación multiclase para distintos escenarios de daño
- Diseño de red neuronal tipo VAE para detección de anomalías
- Diseño de red neuronal de clasificación binaria
- Entrenamiento de las redes neuronales con datos generados mediante un modelo numérico
- Analizar la efectividad de los distintos métodos estudiados

Capítulo 1

Antecedentes

1.1. Material Compuesto tipo Sándwich

Las estructuras tipo sándwich son un material compuesto que consta de tres partes especificas: placa superior e inferior, el núcleo y una capa adhesiva que logra mantener la unión de las placas y el núcleo. Las placas pueden ser de diversos materiales, como aluminio o fibra de carbono mientras que el núcleo típicamente es de Nomex tipo panal de abeja. Nomex es una fibra de aramida similar al Nylon. En la figura 1.1 se muestra una estructura tipo sándwich con placas de fibra de carbono y con un núcleo de Nomex tipo panal de abeja y en la figura 1.2 se observan las distintas capas del material y su confección.

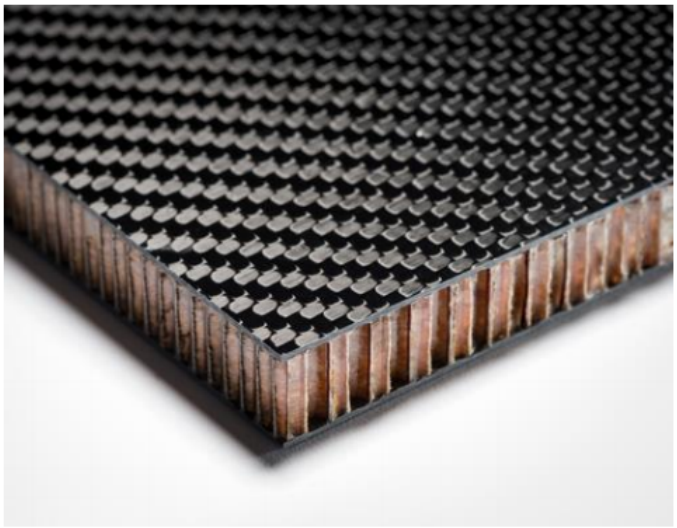

Figura 1.1: Estructura tipo panal de abeja con placas de fibra de vidrio

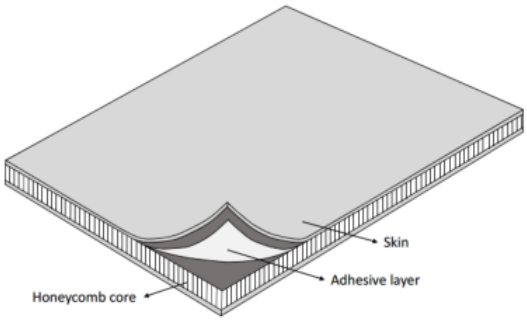

Figura 1.2: Esquema de un panel tipo sandwich con núcleo de panal de abeja [15]

Una de las particularidades de estos materiales es su excelente razón peso-resistencia. La rigidez a flexión EI se define como el par de fuerzas necesario para doblar una estructura sólida por unidad de curvatura producida, donde E es el módulo de Young e I es el segundo momento de inercia. En la figura 1.3 y la tabla 1.1 se muestra la mejora en la rigidez del material compuesto mientras se aumenta la separación entre las dos laminas.

Figura 1.3: Comparación entre rigidez y peso en estructura tipo sándwich

Espesor núcleo	0	t	3t
Rigidez a flexión	1	7	37
Resistencia	1	3.5	9.25
Peso	1	1.03	1.06

Tabla 1.1: Comparación entre rigidez y peso en estructura tipo sándwich [16]

Es por su bajo peso y su alta rigidez que este tipo de materiales es cada vez más utilizado en diferentes áreas de la ingeniería, en particular destacan la industria automotriz, aeronáutica, y sistemas eólicos. Sin embargo, este material no está exento de problemas. Dada su compleja manufactura, es posible que existan defectos como una incompleta cobertura del adhesivo (resina epóxica usualmente) o incluso pueden quedar burbujas de aire en la capa de resina, que conllevan a una adhesión no uniforme entre las placas y el núcleo del material.

La delaminación se define como la perdida de adherencia entre las placas y el núcleo. Este defecto tiene como consecuencia una reducción en la rigidez, resistencia y puede desembocar en la falla del material. Es por este tipo de fallas y la dificultad que existe en dar cuenta de ellas que este tipo de materiales no es más utilizado en la industria, ya que su utilización por el momento está en industrias de alto requerimiento de confiabilidad, por lo que se requieren

métodos de prevenir el daño, identificar y solucionar rápidamente antes que deriven en una falla catastrófica.

1.2. Identificación de Daño

La identificación de daño que se realiza en placas de material compuesto tipo sándwich es una tarea sumamente compleja mediante métodos no destructivos. Es por esto que la utilización de detección mediante vibraciones mecánicas resulta una alternativa atractiva. Definimos la identificación de daño por vibraciones como un conjunto de técnicas para el monitoreo y análisis de las características de un sistema (respuesta a la aplicación de vibraciones) con el propósito de detección de cambios, lo que podría indicar daño o degradación [1].

Los modos de vibración son la forma característica en la que vibrará un sistema mecánico y están asociados a una frecuencia natural respectiva, aunque la vibración de una estructura es una combinación de todos los modos de vibración, pero no todos excitados necesariamente al mismo grado. Los modos de vibración pueden calcularse analíticamente o experimentalmente, siendo el último caso el mas común debido a la dificultad del cálculo analítico para sistemas complejos.

El análisis de la curvatura de los modos de vibración asume que en caso de haber daño estructural, la curvatura de los modos de vibración se ve afectada principalmente en las zonas dañadas. Es por esto que se ha desarrollado algoritmos de identificación de daño a partir de cambios en la curvatura de los modos de vibración [3] y que serán la base de los algoritmos a desarrollar en este trabajo de título.

1.3. Machine Learning

Machine Learning es una rama de la inteligencia artificial que tiene como objetivo crear sistemas que aprendan automáticamente, identificando estructuras y patrones en los datos que se usan para su entrenamiento. El objetivo principal es, a partir de los datos de entrada (entrenamiento), poder clasificar o predecir el comportamiento de datos que no han sido utilizados por el modelo. Un ejemplo clásico de un modelo de Machine Learning es el filtrado de spam en los correos electrónicos. Se utiliza como datos de entrada correos que han sido etiquetados como spam, y en base a esto el modelo debe aprender a identificar si un nuevo correo corresponde o no a spam. Podemos dividir los problemas de Machine Learning en tres tipos:

- **Aprendizaje Supervisado**: A partir de una base de datos de entrenamiento, en la cual cada ejemplo corresponde a un par (x,y) donde x es el input (texto, imágenes, números, etc) e y es la etiqueta asociada a dicho input, es decir, el resultado deseado. El objetivo primordial es que el modelo sea capaz de generalizar para pares de datos a los que no ha sido sometido durante el entrenamiento, o sea, predecir cual será la etiqueta de un nuevo par (x,y) que no corresponde al conjunto de entrenamiento original.

- **Aprendizaje No Supervisado**: Los datos de entrada no poseen etiquetas ni clasificación alguna. La idea principal es que el modelo pueda encontrar (sin ayuda externa como etiquetas) patrones que sirvan para entender el conjunto de datos.
- **Aprendizaje Reforzado**: El modelo debe aprender en base a prueba y error. Debe tomar decisiones para maximizar o lograr un objetivo, y se le premia o se le castiga según las decisiones que vaya tomando.

Para este trabajo de titulo se realizan tanto modelos de clasificación (aprendizaje supervisado) como modelos de aprendizaje no supervisado, específicamente el Variational Autoencoder que se explicará con mayor detalle en la sección 1.4.

1.3.1. Redes neuronales

La unidad básica de una red neuronal es una neurona o perceptrón. El funcionamiento de una única neurona conectada a la entrada se observa en la figura 1.4 .

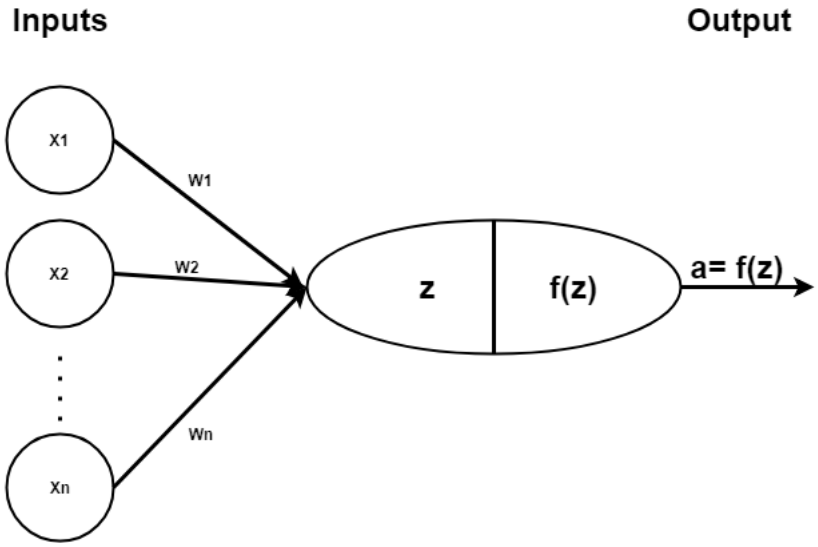

Figura 1.4: Representación gráfica del funcionamiento de un perceptrón conectado a la entrada

La entrada corresponde a un vector $\overrightarrow{x} = [x_1, x_2, ..., x_n]$. Los pesos $\overrightarrow{w} = [w_1, w_2, ..., w_n]$ son los parámetros de la red que se requieren optimizar. Existe una combinación lineal z que se observa en la ecuación 1.1 entre la entrada y los respectivos pesos, y luego se aplica una función de activación no lineal f(z) que será la salida de la neurona.

$$z = \sum_{\mathrm{i}=0}^{N} w_{\mathrm{i}} x_{\mathrm{i}} \tag{1.1}$$

Los pesos no son únicamente una propiedad de la entrada de la red, sino que cada neurona

tiene asociado su propio peso que la conecta con las neuronas de la siguiente capa. Una cantidad de neuronas apiladas proporciona una capa, y a la vez, se puede tener varias capas en una red neuronal. Se muestra el ejemplo de una red neuronal en la figura 1.5, se aprecia en esta figura la presencia de dos capas ocultas, de cuatro neuronas cada una. A la cantidad de capas, neuronas por capa, y función de activación utilizado en las capas intermedias y en la capa de salida se le denomina arquitectura de la red.

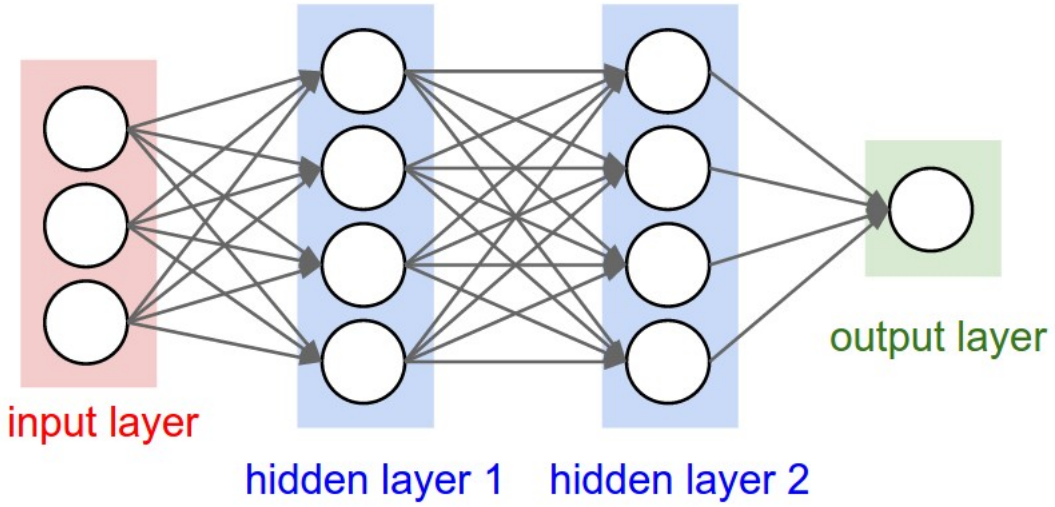

Figura 1.5: Representación gráfica de una red neuronal de dos capas escondidas y una única neurona de salida [14]

1.3.2. Funciones de activación

Como se explicó en la sección anterior, parte esencial de una red es la elección de una función de activación para las neuronas de una capa. El teorema de Aproximación Universal, establece que una sola capa intermedia es suficiente para aproximar, con una precisión arbitraria, cualquier función con un número finito de discontinuidades, siempre y cuando las funciones de activación de las neuronas ocultas sean no lineales. El teorema establece que las redes multicapa no añaden capacidad a menos que la función de activación de las capas sea no lineal, debido a que si las funciones de activación en toda la red son lineales, se puede realizar una función lineal que agrupe todas las capas en un solo perceptrón, lo que eliminaría el carácter profundo de la red.

En capas intermedias, las siguientes funciones de activación son las mas utilizadas en la actualidad:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1.2}$$

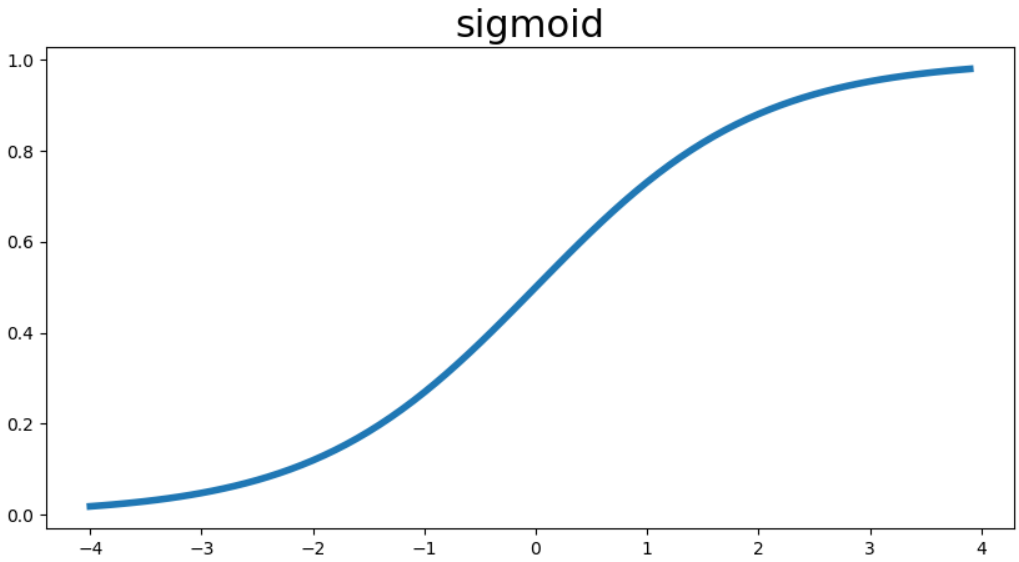

Figura 1.6: Función de activación Sigmoid

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{1.3}$$

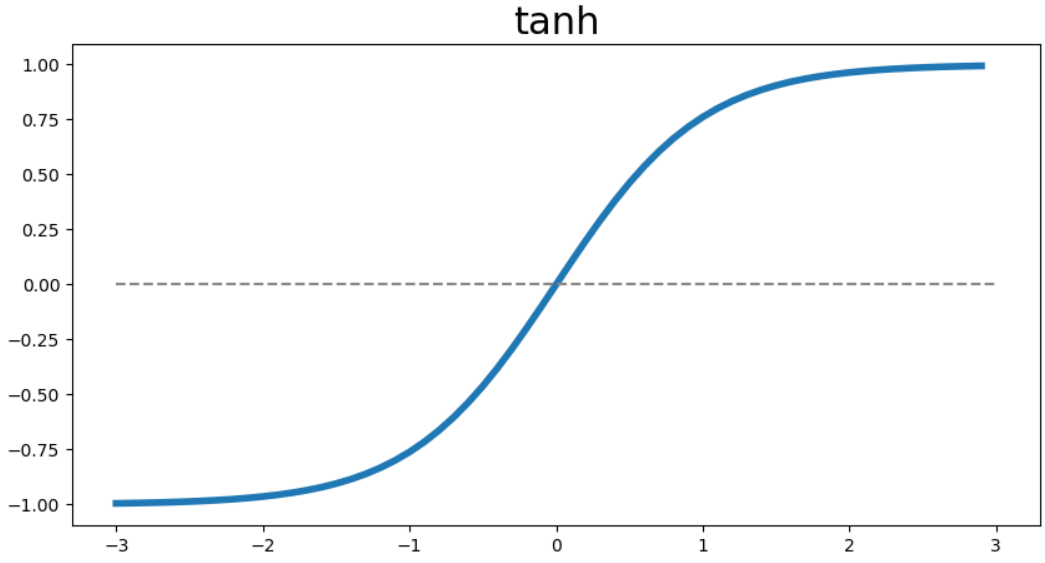

Figura 1.7: Función de activación tangente hiperbólica

$$ReLu = \begin{cases} x \text{ si } x \geq 0 \\ 0 \text{ si } x < 0 \end{cases} \tag{1.4}$$

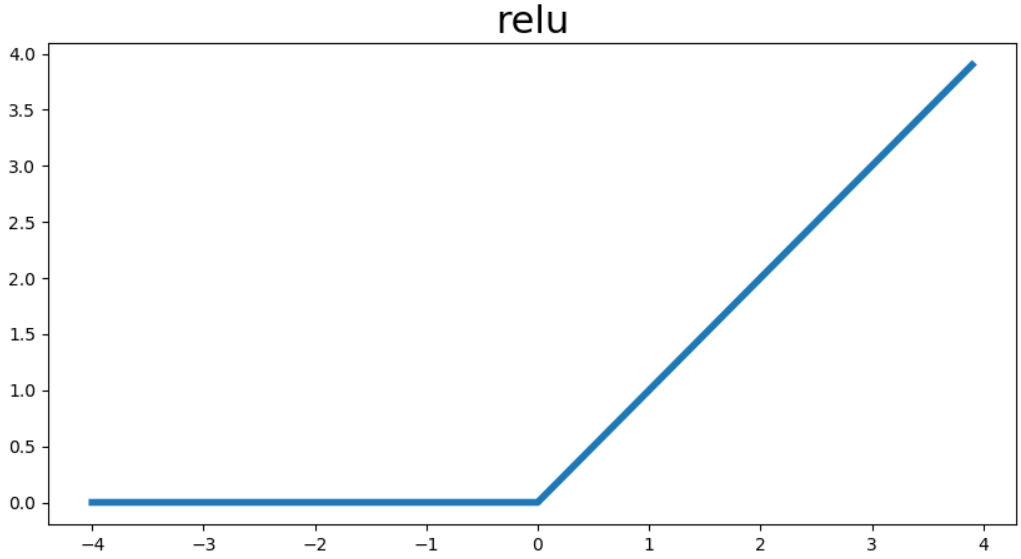

Figura 1.8: Función de activación Relu

Se aprecia la presencia de cotas superiores e inferiores tanto en la función sigmoid como en la tangente hiperbólica. La función sigmoid está acotada en el intervalo (0,1) mientras que la función tanh está acotada en el intervalo (-1,1). La función ReLu posee un mínimo (0) pero no está acotada superiormente. La elección de cual función a utilizar dependerá del problema a resolver, aunque muchas veces es por prueba y error que se termina seleccionando una función de activación por sobre las otras, en la actualidad la mas utilizada para las capas intermedias es la función ReLu.

Si se desea resolver un problema de clasificación multiclase, la función de activación de la última capa de la red (encargada de predecir la salida o output) será la función Softmax. La función Softmax entrega una distribución de probabilidades por sobre las clases de salida, por lo que para cada clase habrá una probabilidad entre (0,1) de que sea la clase correcta, sin embargo, la probabilidad todas las clases sumadas deben ser 1. La función Softmax se presenta en la siguiente ecuación, donde K corresponde a la cantidad de clases y z el vector de entrada.

$$\sigma(\mathbf{z})_j = \frac{\mathrm{e}^{z_j}}{\sum_{k=1}^{K} \mathrm{e}^{z_k}} \tag{1.5}$$

1.3.3. Redes convolucionales

En esta sección se explicará el funcionamiento de una red de convolución, aplicada a imágenes. Una imagen a color no es mas que la unión de tres matrices (canales): rojo, verde y azul (RGB en ingles). Cada cuadro en un canal representa la intensidad de ese píxel, por lo que en la figura 1.9 se observa una imagen de 4x4 píxeles en formato RGB.

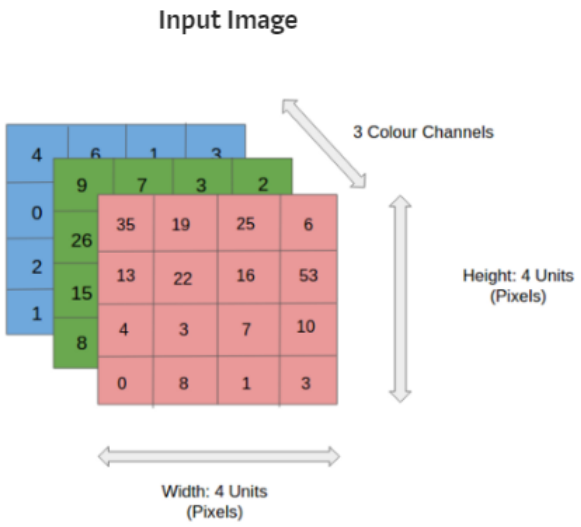

Figura 1.9: Imagen RGB de 4x4 píxeles [14]

Una imagen en escala de grises normalizada, corresponde a un solo canal donde los valores de cada píxel también representan intensidad y varían entre 0 (negro) y 1 (blanco).

La operación básica de una red convolucional es la convolución matemática. Un ejemplo se puede encontrar entre las figuras 1.10 y 1.13. Se tiene una imagen de 7x7 (o input, ya que puede ser una capa convolucional conectada con otra capa convolucional, no necesariamente con la entrada de la red), un filtro de 3x3 y la salida o output de 5x5. Los valores del filtro son los parámetros que debe optimizar la red, pero para el ejemplo se consideran fijos y los valores son los de la figura 1.10.

Se ejecuta una convolución al realizar una multiplicación elemento a elemento entre dos matrices (de igual dimensión) y sumando los resultados, obteniéndose un solo valor final. En la figura 1.10 se aprecia se aprecia una operación de convolución entre el cuadrado señalado en celeste de la imagen original, y el filtro. El resultado se obtiene de la siguiente manera: $(1 \cdot 0) + (2 \cdot 0) + (3 \cdot 1) + (4 \cdot 1) + (5 \cdot 0) + (6 \cdot 0) + (7 \cdot 0) + (8 \cdot 1) + (9 \cdot 1) = 24$.

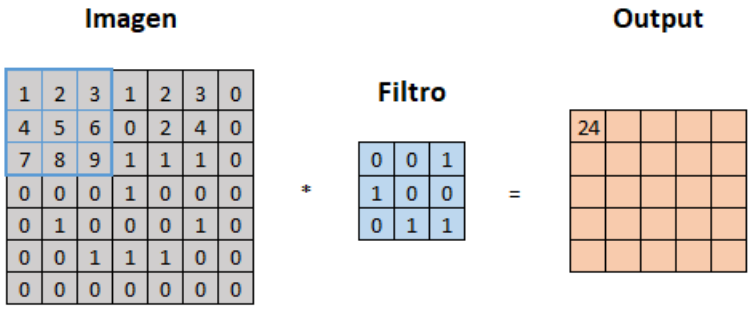

Figura 1.10: Ejemplo de convolución, primer paso

Después de realizada la primera convolución, el filtro se desliza por sobre la imagen de entrada en una posición hacia la derecha en una convolución típica. Se aplica una convolución nuevamente entre el cuadro celeste de la imagen y el filtro. El resultado se obtiene de la siguiente manera: $(2\cdot 0)+(3\cdot 0)+(1\cdot 1)+(5\cdot 1)+(6\cdot 0)+(0\cdot 0)+(8\cdot 0)+(9\cdot 1)+(1\cdot 1)=16$.

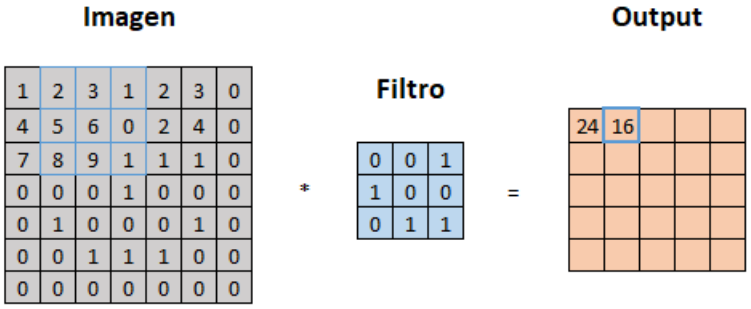

Figura 1.11: Ejemplo de convolución, segundo paso

El procedimiento se repite hasta llegar al final de la entrada. Para continuar, se desliza el filtro en la entrada un espacio hacia abajo y se vuelve a repetir el procedimiento de izquierda a derecha como se aprecia en la figura 1.12.

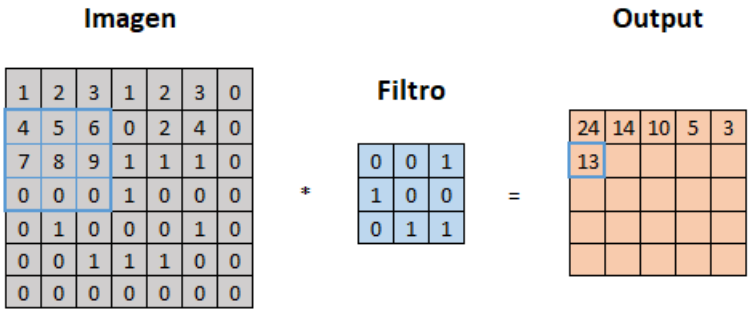

Figura 1.12: Ejemplo de convolución, sexto paso

Finalmente se repite el procedimiento anterior hasta que ya no hay mas opción de seguir

deslizando el filtro por sobre la imagen, el resultado final de la convolución se aprecia en la figura 1.13, seguido de esto se aplica una función de activación no lineal como se explicó en la sección anterior.

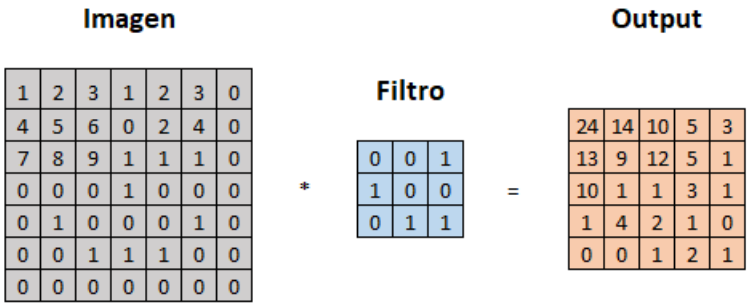

Figura 1.13: Ejemplo de convolución finalizada

Este es el funcionamiento de una convolución simple con un filtro de 3x3. En la practica una capa de convolución tiene más de un filtro (todos del mismo tamaño) y se pueden aplicar algunas técnicas para variar el tamaño de la salida.

En el ejemplo anterior se mostró como se deslizaba el filtro sobre la imagen un espacio a la vez cada vez que realizaba una convolución, a esto se le denomina **Stride**, el Stride es variable y queda a criterio del programador cuanto Stride se le aplica a cada convolución, teniendo en cuenta que al aumentar este valor la dimensión de la salida disminuirá como se observa en la ecuación 1.6.

Un problema de las redes convolucionales es que el filtro desliza sobre los píxeles de los extremos (superior e inferior) considerablemente menos veces que los que se encuentran más hacia el centro de la imagen. El filtro solo se desliza sobre los píxeles de la primera final una vez, para la segunda fila desliza dos veces, mientras que para la mayoría de las filas el filtro desliza tres veces por sobre los píxeles. Esto puede ser un problema cuando existe información relevante de la imagen que se encuentra en las esquinas por ejemplo. Una posible solución es agregar **Padding**. Padding consiste en agregar una cantidad determinada de información irrelevante en los bordes de la entrada, como se aprecia en la figura 1.14. Se observa que la dimensión original de la imagen era de 7x7 y al agregar un Padding simple la imagen aumenta su dimensión a 9x9, se puede aplicar Padding doble, triple o lo que desee el programador según su criterio.

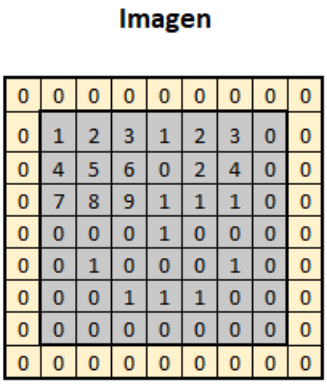

Figura 1.14: Ejemplo de Padding

A continuación se explica como se calculan la dimensión de salida de una capa de convolución. La entrada tiene una dimensión de $L_1 \times A_1 \times C_1$ (largo, ancho y numero de canales). La dimensión de salida se calcula como:

$$L_2 = \frac{L_1 - F + 2P}{S + 1} \tag{1.6}$$

$$A_2 = \frac{A_1 - F + 2P}{S + 1} \tag{1.7}$$

$$C_2 = K \tag{1.8}$$

Donde K es el numero de filtros utilizados, F es el tamaño del filtro, S es el numero de Stride utilizado y P la cantidad de Padding.

En la actualidad, las redes convolucionales son las más utilizadas para tareas que utilizan imágenes como datos de entrada, además tienen la ventaja de que una red convolucional en general tiene una cantidad de parámetros a optimizar considerablemente menor a una red neuronal multicapa, lo que implica que se requiere menos poder computacional. Lo normal es que a medidas que se agregan capas convolutivas, la dimensión de la salida disminuye en términos de largo y ancho, pero aumenta en numero de canales.

En las redes convolucionales existe una operación denominada Flatten, que tiene como función aplanar la entrada (tridimensional) en un vector, como se muestra en la figura 1.15 .

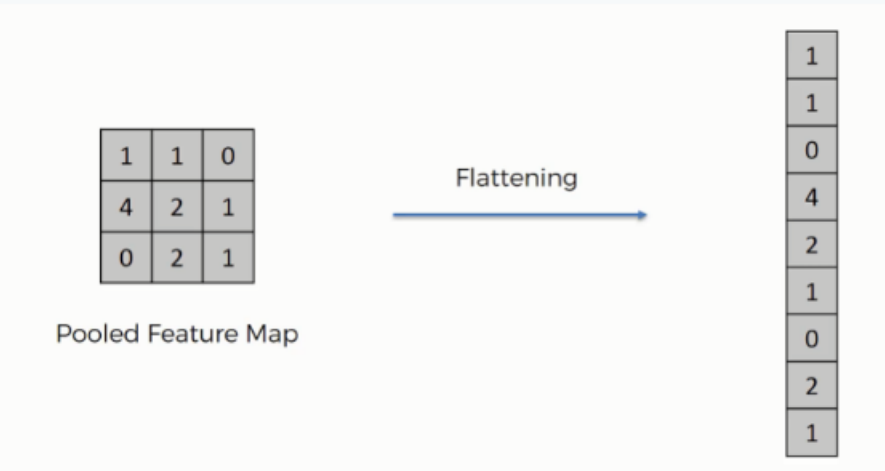

Figura 1.15: Ejemplo de Flatten [17]

El propósito de realizar esta operación es para conectar esta capa con otra capa de neuronas que sirvan para realizar una tarea, como la clasificación. En la figura 1.16 se observa como después de un Flatten, se puede unir esta capa con otra capa completamente conectada, la cual se une a la salida de la red que identifica dos posibles outputs como clasificación.

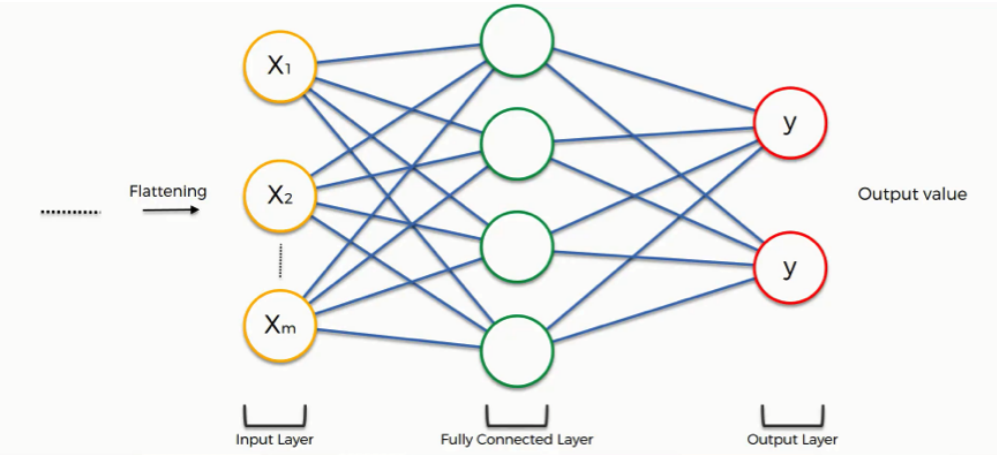

Figura 1.16: Red neuronal posterior a un Flatten [17]

1.3.4. Entrenamiento de una red

Definida la arquitectura de una red y el problema a realizar, se procede a entrenar a la red. Se debe en primer lugar inicializar los pesos de los parámetros a optimizar, para esto generalmente se ocupa la inicialización de Xavier . Debido a que esta inicialización es una distribución Gaussiana, la red en un comienzo no tendrá un buen desempeño para lograr la tarea esperada , por lo que se requiere ir optimizando los valores de los pesos con tal de que la red pueda aproximarse a una función capaz de lograr los objetivos deseados de esta (por ejemplo, clasificar imágenes de gatos o perros).

Según el problema en cuestión, se requiere definir el error entre lo que la red nos está entregando y lo que debiese entregar, durante la fase de entrenamiento. El error será una función de la entrada a la red y los parámetros de la misma. El objetivo primordial se convierte entonces en disminuir este error durante la fase de entrenamiento lo mas posible, esperando lograr que la red también pueda generalizar para casos a los que no ha sido sometida durante el entrenamiento.

El algoritmo llamado *Gradient Descent* en conjunto con la técnica de *Back Propagation* nos permiten en la actualidad entrenar las redes neuronales. Sea $E(\vec{w}, \vec{x})$ el error, en función de los pesos $\vec{w}$ y la entrada de la red $\vec{x}$, se busca encontrar $\vec{w}^*$ conjunto de pesos tal que se minimice el error $E(\vec{w}, \vec{x})$, formalmente se aprecia esto en la ecuación 1.9.

$$\vec{w}^* = \arg\min_{\vec{w}} E(\vec{w}, \vec{x}) \tag{1.9}$$

Si la función $E(\vec{w}, \vec{x})$ es diferenciable con respecto a $\vec{w}$, entonces podemos calcular la derivada parcial del error total con respecto a cada parámetro de la red, la lógica detrás de este cálculo es ver en cuanto aporta al error total cada parámetro.

$$\frac{\partial E}{\partial \vec{w}} = [\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ... \frac{\partial E}{\partial w_D}] \tag{1.10}$$

Luego de calculado el error con respecto a cada parámetro, se procede a actualizar los parámetros según la ecuación 1.11.

$$w_{\mathrm{i}} = w_{\mathrm{i}} - \lambda \frac{\partial E}{\partial w_{\mathrm{i}}} \tag{1.11}$$

Donde λ se denomina *Learning rate* y determinará la actualización de los parámetros. Los calculos de las derivadas con respecto a los parámetros se realiza utilizando la regla de la cadena y *Back Propagation*. La derivada parcial con respecto a la ultima capa se puede calcular analíticamente debido a que la función de error es diferenciable y conocida (y que depende del problema a resolver), sin embargo, para seguir calculando el gradiente hacia las capas mas cercanas a la entrada, se utiliza la regla de la cadena para propagar los errores.

El proceso antes mencionado se repite hasta converger o el error sea aceptable y logre el objetivo deseado.

1.3.5. Métricas de Desempeño

El desempeño de una red de clasificación se mide usualmente en base a un parámetro llamado Accuracy (precisión), que cuantifica el porcentaje de ejemplos que fueron clasificados correctamente. En la tabla 1.2 se observa un ejemplo de matriz de confusión, que sirve para mostrar los ejemplos clasificados correctamente y los mal clasificados.

Clase		Prediccion	
Clasificación real	Positivo	Verdadero Positivo (TP)	Falso Negativo (FN)
	Negativo	Falso Positivo (FP)	Verdadero Negativo (TN)
		Clasificado como positivo	Clasificado como negativo

Tabla 1.2: Matriz de confusión

Podemos utilizar distintas métricas de clasificación en base a los parámetros mencionados en la tabla 1.2:

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{1.12}$$

$$\text{F1 score} = \frac{2TP}{2TP + FP + FN} \tag{1.13}$$

1.4. Variational Autoencoder

Un Autoencoder es un tipo de red neuronal de aprendizaje no supervisado. Su meta es comprimir los datos de entrada lo más posible, en un vector denominado vector latente y luego utilizar ese mismo vector latente para reconstruir los datos de entrada lo mejor posible. Si esta reconstrucción es aceptable, entonces el vector latente está capturando la verdadera naturaleza de los datos, codificando de buena manera los datos de entrada y luego la red neuronal también está decodificando este vector latente correctamente en la imagen de salida (reconstrucción).

Podemos ver un Autoencoder como la unión de dos redes neuronales. Una denominada encoder, que se encarga de representar los datos de entrada en el vector latente y la red decodificadora (decoder) encargada de reconstruir la imagen original a partir del vector latente entrenado por el encoder. En la figura 1.17 se muestra la arquitectura de un Autoencoder simple, donde la entrada x se une directamente con el vector latente z y a la vez, este está unido a la salida x' que es la reconstrucción de la entrada.

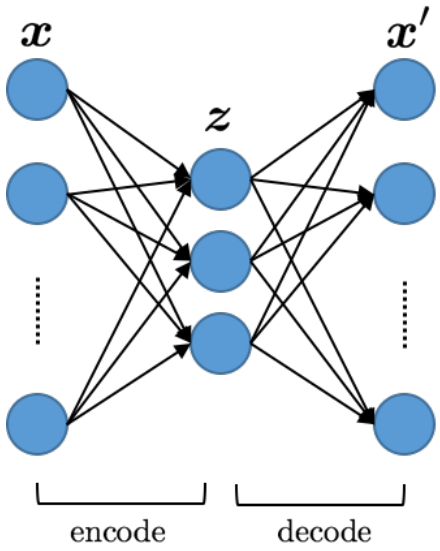

Figura 1.17: Arquitectura de un Autoencoder [13]

Un ejemplo mas concreto se encuentra en la figura 1.18. En este ejemplo se observa como el vector latente encuentra características relevantes de la foto de entrada (sonrisa, color de piel, barba, etc) y las cuantifica en un valor, con esto el decoder intenta reconstruir la imagen en base a estos 6 atributos.

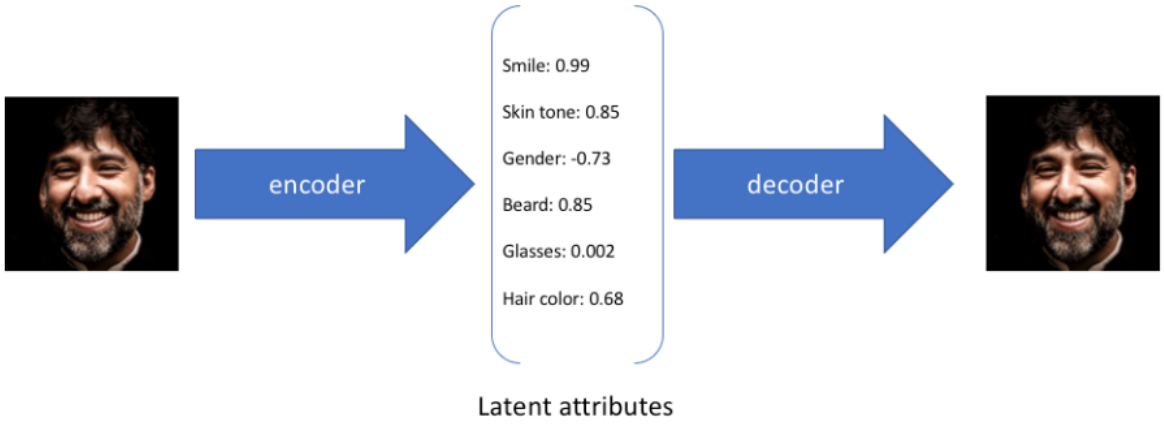

Figura 1.18: Ejemplo de funcionamiento de un Autoencoder [13]

Un Variational Autoencoder (VAE) intenta en vez de tener un vector latente con valores fijos, que cada valor de este vector latente sea en realidad una distribución de probabilidad. El decoder entonces deberá muestrear aleatoriamente un valor de la distribución para cada atributo del vector latente, un ejemplo de esto se observa en la figura 1.19. Se espera que para distintas selección de valores del vector latente, la reconstrucción sea lo mas parecida posible a la imagen de entrada, así, la variabilidad de la selección no genera un error significativo.

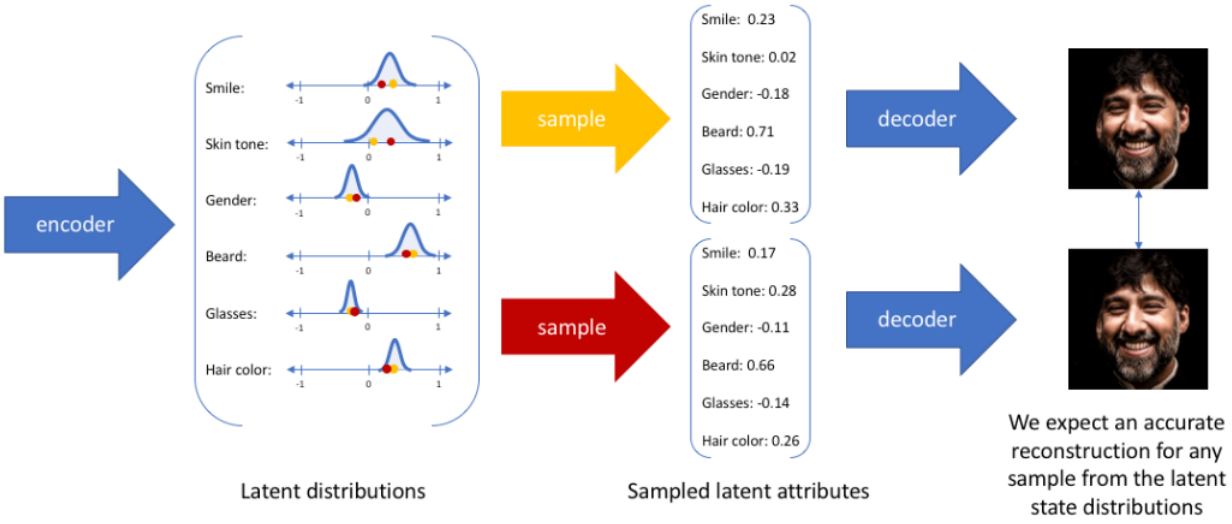

Figura 1.19: Funcionamiento de un Variational Autoencoder [13]

El VAE es un modelo que combina inferencia bayesiana y Machine Learning. Supongamos que existe una variable oculta z que es la que genera nuestras observaciones x (nuestros datos). Nuestras observaciones son nuestros datos, pero también queremos obtener las características de esta variable oculta que genera nuestros datos. Queremos computar $p(z|x)$ que es la probabilidad condicional de esta variable latente dado nuestras observaciones. Utilizando el teorema de Bayes podemos calcular este valor según la ecuación 1.14.

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{1.14}$$

En el denominador se encuentra $p(x)$ que es la evidencia. Podemos calcular esto utilizando la ecuación 1.15.

$$p(x) = \int p(x|z)p(z)\mathrm{d}z \tag{1.15}$$

El problema recae en que esta integral es compleja de calcular y muchas veces imposible ya que tiene que evaluarse en todas las configuraciones de variables latentes. Por esto, calcular $p(z|x)$ no es posible, sin embargo, podemos aproximar $p(z|x)$ utilizando una distribución diferente $q_\theta(z|x)$ que la definimos de forma que si sea calculable la integral anterior. Esta nueva distribución tiene un conjunto de parámetros θ de forma que si podemos encontrar los valores de estos parámetros tal que la distribución de $q_\theta(z|x)$ sea similar a $p(z|x)$, entonces habremos resuelto el problema de encontrar la probabilidad condicional de las variables latente dado las observaciones.

Para optimizar $q_\theta(z|x)$ de forma que sea lo mas parecido posible a $p(z|x)$ minimizamos la Divergencia de Kullback-Leibler entre estas dos distribuciones. La divergencia mide cuanta

información se pierde al utilizar q para aproximar p. En la ecuación 1.16 se muestra el problema de optimización a resolver:

$$q_{\theta^*}(z|x) = \arg\min_{\theta} D_{KL}(q_\theta(z|x) \parallel p(z|x)) \tag{1.16}$$

$$D_{KL}(q_\theta(z|x) \parallel p(z|x)) = \mathbb{E}_{q_\theta(z|x)}[\log q_\theta(z|x)] - \mathbb{E}_{q_\theta(z|x)}[\log p(z|x)] \tag{1.17}$$

Utilizando la ecuación 1.14 y aplicando propiedad del logaritmo, podemos escribir esto de la siguiente forma:

$$D_{KL}(q_\theta(z|x) \parallel p(z|x)) = \mathbb{E}_{q_\theta(z|x)}[\log q_\theta(z|x)] - \mathbb{E}_{q_\theta(z|x)}[\log p(x,z)] + \log p(x) \tag{1.18}$$

Lamentablemente vuelve a aparecer el término $p(x)$ en la ecuación anterior. Por esto, es imposible calcular directamente la minimización propuesta en la ecuación 1.16. Definimos una variable nueva denominada ELBO (Evidence Lower Bound), como se aprecia a continuación:

$$ELBO(\theta) = \mathbb{E}_{q_\theta(z|x)}[\log p(z,x)] - \mathbb{E}_{q_\theta(z|x)}[\log q_\theta(z|x)] \tag{1.19}$$

Con este cambio de variable, podemos re escribir la ecuación 1.18 :

$$\log p(x) = D_{KL}(q_\theta(z|x) \parallel p(z|x)) + ELBO(\theta) \tag{1.20}$$

Debido a que la Divergencia KL siempre es igual o mayor a cero, minimizar la divergencia KL es equivalente a maximizar ELBO, ya que log p(x) es un valor fijo. Podemos reordenar los términos y usar la definición de la divergencia KL y obtener una nueva expresión para el ELBO:

$$ELBO(\theta) = \mathbb{E}_{q_\theta(x|z)}[\log p(x|z)] - D_{KL}(q_\theta(z|x) \parallel p(z)) \tag{1.21}$$

Desde la perspectiva de un modelo de Machine Learning, podemos aproximar la distribución $q_\theta(z|x)$ con una red neuronal de inferencia, denominada encoder. Podemos además, parametrizar $p(x|z)$ con una red generativa $p_\phi(x|z)$ que tiene como entrada las variables latentes y como salida los parámetros de los datos. Por lo tanto, queremos maximizar ELBO (ecuación 1.22) utilizando métodos de Machine Learning.

$$ELBO(\theta) = \mathbb{E}_{q_\theta(z|x)}[\log p_\phi(x|z)] - D_{KL}(q_\theta(z|x) \parallel p(z)) \tag{1.22}$$

El primer termino del lado derecho de la ecuación 1.22 se le denomina como error de reconstrucción y el segundo termino es un regularizador que intenta que la distribución de q sea lo mas cercano posible a la distribución de $p(z)$.

El único problema que queda por resolver tiene que ver con la selección aleatoria de muestreo del vector latente. El algoritmo de Back Propgation no funciona correctamente con unidades estocásticas dentro de la red, debido a que estas son no continuas y por lo mismo, no puede computar el gradiente para estos puntos. Lo que si se puede hacer para utilizar el algoritmo, es un truco de reparametrización que nos permitirá muestrear del vector latente de forma aleatoria y a la vez optimizar los parámetros de la distribución del vector.

$$z = \mu + \sigma \odot \varepsilon \tag{1.23}$$

La selección de atributos del vector latente se hace utilizando la ecuación 1.23 donde $\varepsilon \sim Normal(0, 1)$. Con esto es posible realizar derivadas que involcuren z y $f(z)$ con respecto a los parámetros de su distribución $\mu + \sigma$.

Queda por definir únicamente la naturaleza de las distribuciones paramétricas mencionadas anteriormente. Para los VAE es usual asumir que la distribución de p(z) sigue la siguiente distribución:

$$p(z) \sim Normal(0, I) \tag{1.24}$$

La distribución $p_\phi(x|z)$ representa la probabilidad de que una observación x sea generada por una variable latente z. La forma que tendrá esta distribución dependerá de la naturaleza de los datos utilizados. Si los datos tienen forma binaria o reales acotados entre 0 y 1 se ocupa una distribución de Bernoulli. Si los datos son reales continuos, una distribución Normal es utilizada.

Finalmente, queda por definir la distribución de $q_\theta(z|x)$. Es usual utilizar una distribución normal multivariante, debido a su relación con $p(z)$ esta elección permite un facil calculo y minimizar la divergencia KL de la ecuación 1.22.

$$q_\theta(z|x) \sim Normal(\mu, \sigma^2 I) \tag{1.25}$$

Debido a que los datos de este trabajo de título corresponden a imágenes en escalas de grises contenidos entre (0,1), se utiliza una distribución de Bernoulli para representar $p_\phi(x|z)$. En base a esto, se puede aproximar el error de reconstrucción de la siguiente forma:

$$\mathbb{E}_{q_\theta(z|x)}[\log p_\phi(x|z)] \approx \frac{1}{L} \sum_{\mathrm{i}=1}^{L} (\log \hat{x}_\mathrm{i} + (1 - x_\mathrm{i}) \log(1 - \hat{x}_\mathrm{i})) \tag{1.26}$$

Esta ecuación también tiene el nombre de Sigmoid Binary Crossentropy. Donde x_i es la entrada (imagen) y $\hat{x}_\mathrm{i}$ es la salida de la red, o sea, la reconstrucción de x. El valor L es la cantidad de atributos utilizados del vector latente para la reconstrucción.

Como $p(z)$ y $q_\theta(z|x)$ distribuyen normalmente, el termino de regularización del ELBO podemos computarlo de la siguiente manera, donde J es la dimensión del vector latente:

$$D_{KL}(q_\theta(z|x) \parallel p(z)) = \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right) \tag{1.27}$$

En la figura 1.20 se aprecia la arquitectura de un VAE típico, con una distribución de Bernoulli para el decoder.

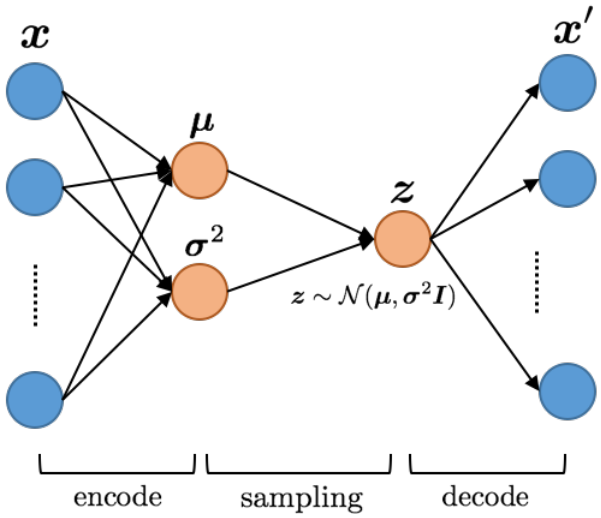

Figura 1.20: Arquitectura de un VAE [13]

Una vez entrenado un VAE de forma satisfactoria, es esperable que este sea capaz de generar una reconstrucción muy parecida a la original. De la misma forma, decimos que el modelo logra captar la verdadera naturaleza de los datos con los que fue entrenado. De esta forma, se puede utilizar un VAE como un algoritmo del tipo One-Class Classification para la detección de anomalías. Este tipo de problema se resuelve cuando se entrena el algoritmo solo con instancias normales, sin presencia de anomalías o outliers. Luego, al entregarle al modelo datos anómalos (que se puede considerar que provienen de una distribución diferente a la de los datos normales) es esperable que el rendimiento no sea bueno y que el algoritmo no logre reconstruir de buena forma la entrada para estos nuevos datos. Por lo mismo, podemos ocupar el error de reconstrucción mencionado anteriormente para identificar la presencia de un dato anómalo [4], para esto se utiliza un valor denominado threshold como métrica de clasificación. Si el error de reconstrucción para un dato es mas alto que el threshold determinado, entonces se clasifica como anómalo, caso contrario se clasifica como dato normal.

Capítulo 2

Metodología

Con el fin de lograr los objetivos planteados al comienzo de este proyecto, se requiere la implementación de diferentes programas. Para esto, como requisito previo, se necesita de una recopilación bibliográfica y un estudio intensivo de funcionamiento de Machine Learning. Se propone la siguiente metodología para el desarrollo de esta memoria:

1. Estudio de Deep Learning
2. Orden y análisis previo de datos
3. Generación de modelos computacionales
4. Entrenamiento del modelo
5. Análisis de resultados

2.1. Estudio Machine Learning

Para la realización de este proyecto, es vital poseer el conocimiento de machine learning y redes neuronales y convolucionales, que permitan poder generar un modelo robusto y que sea capaz de cumplir con los objetivos planteados. Para esto, se participó en el curso online Machine Learning dictado por el profesor Andrew Ng de la universidad de Standford y del curso Bayesian Methods for Machine Learning dictados por Daniil Polykovskiy y Alexander Novikov de National Research University – Higher School of Economics.

Dentro de este ítem se considera también la recopilación bibliográfica necesaria. Es de vital importancia tener una base solida sobre el estado del arte de las técnicas utilizadas para la clasificación en redes neuronales y para la utilización de Variational Autoencoders como modelo de identificación de anomalías.

2.2. Orden y análisis previo de datos

Antes de comenzar a programar el modelo computacional, es importante revisar los datos con los que se cuenta. Estos fueron entregados durante las primeras semanas del proyecto para poder ser analizados. Algunos de estos datos podrán contener algún tipo de error, por lo que se deben seleccionar la mayor cantidad de datos posibles para el entrenamiento y la verificación posterior del modelo computacional.

2.3. Generación del modelo computacional

Cuando se cuenta con los datos y el conocimiento necesario, es momento para la programación de los distintos modelos. Esta etapa es la más importante de todas ya que se deben seleccionar las diferentes arquitecturas a utilizar, lo que es un factor crítico en el correcto funcionamiento del modelo y el cumplimiento de los objetivos. El modelo se programará en Python utilizando Keras (high level API) y Tensorflow (low level API).

2.4. Entrenamiento del modelo

Con el modelo ya listo, se pasa a la fase de entrenamiento. Una red neuronal requiere de una serie de datos para poder ajustar sus parámetros de tal forma de lograr lo deseado. El entrenamiento consiste en dar al modelo todos los inputs de placas con y sin daño, de tal forma que este sea capaz de reconocer cuando existe la presencia delaminación y su tamaño según la clase a la que pertenezca. Debido a la gran cantidad de hiperparametros a optimizar que posee un modelo, la mayor parte de los recursos se utilizan en el entrenamiento y en la generación, siendo un proceso iterativo y lento, debido que a veces por la búsqueda de obtener mejores resultados, se hacen modificaciones en las arquitecturas previamente definidas.

2.5. Análisis de resultados

Esto consiste en evaluar el comportamiento del modelo. Los resultados indicaran si el modelo es capaz o no de detectar las fallas de delaminación en los datos de prueba (no utilizados para el entrenamiento). En base a esto puede ser necesaria una modificación del modelo para lograr una mayor confiabilidad en los resultados.

2.6. Recursos Utilizados

Los recursos utilizados para la realización de esta memoria son:

- Base de datos de imágenes
- Software MATLAB y Excel para la visualización y la división de las imágenes en conjuntos de entrenamiento, validación y prueba.
- Computador compatible con TensorFlow y CUDA. Para esta memoria se utilizo un computador con un procesador i5-2500 3.30 GHZ, 16 GB de RAM y una tarjeta gráfica Nvidia GTX 950 GT.
- Python como lenguaje de programación, sumado TensorFlow y Keras como bibliotecas de aprendizaje automático.

Capítulo 3

Modelos Propuestos

En esta sección se muestran las diferentes arquitecturas y modelos utilizados para la detección de delaminación. Antes de realizar las pruebas experimentales, es importante contar con los datos necesarios para el entrenamiento de los distintos modelos, por lo que en el siguiente ítem se muestran como se componen los conjuntos de entrenamiento, prueba y validación.

3.1. Conjuntos de datos

Los datos utilizados para el entrenamiento de los distintos algoritmos de esta memoria fueron generados computacionalmente utilizando un modelo numérico del panel tipo sándwich. Esto fue modelado con elementos finitos utilizando un modelo simplificado de tres capas en donde la capa adhesiva entre la capa superior y el núcleo tipo panal de abeja fue modelado como resortes lineales con rigidez reducida en la zona de laminada [3]. Con esto, se obtienen imágenes con y sin daño que se dividen en 6 clases. Estas clases se dividen de la siguiente manera:

1. Clase 0: Sin daño
2. Clase 1: Daño de delaminación entre 0-0.05
3. Clase 2: Daño de delaminación entre 0.05-0.1
4. Clase 3: Daño de delaminación entre 0.1-0.15
5. Clase 4: Daño de delaminación entre 0.15-0.2
6. Clase 5: Daño de delaminación entre 0.2-0.25

Ademas de las generadas computacionalmente, se cuentan con imágenes de 5 placas reales [3] de las cuales una no posee daño, una es de la clase 2, dos son de la clase 3 y una de la clase 4. Las imágenes como se observa en la figura 3.1 están en escala de grises y cada píxel tiene un valor real entre 0 y 1. Las imágenes tienen un tamaño de 51 píxeles de ancho y 71 píxeles de largo.

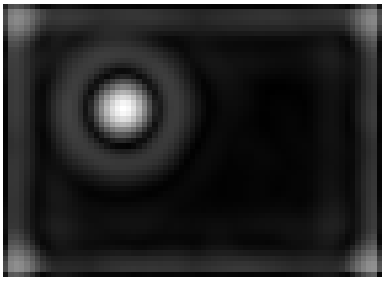

Figura 3.1: Imagen con daño de delaminación

A continuación, entre la figura 3.2 y la figura 3.12 se muestran tres ejemplos de cada clase.

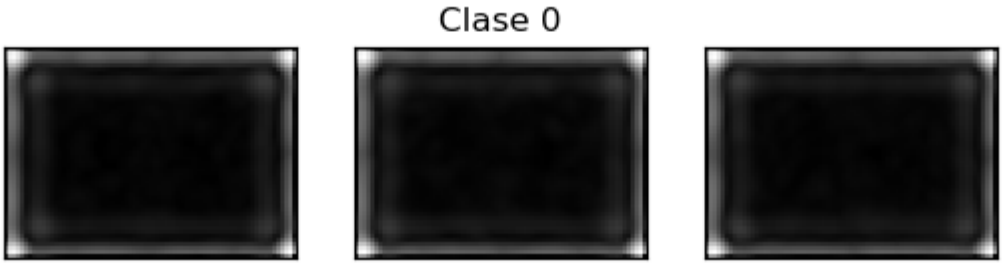

Figura 3.2: Imagen sin daño por delaminación

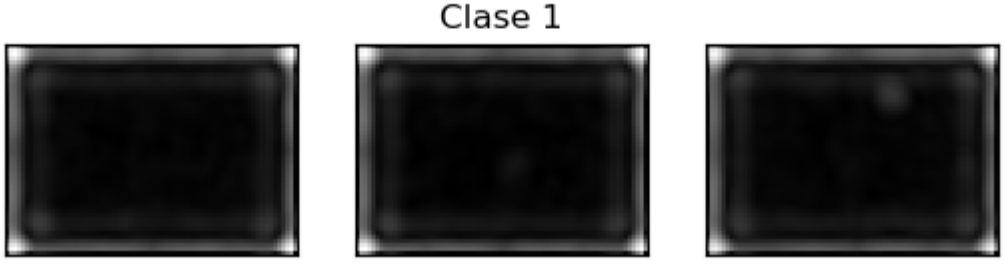

Figura 3.3: Imagen con daño de delaminación, clase 1

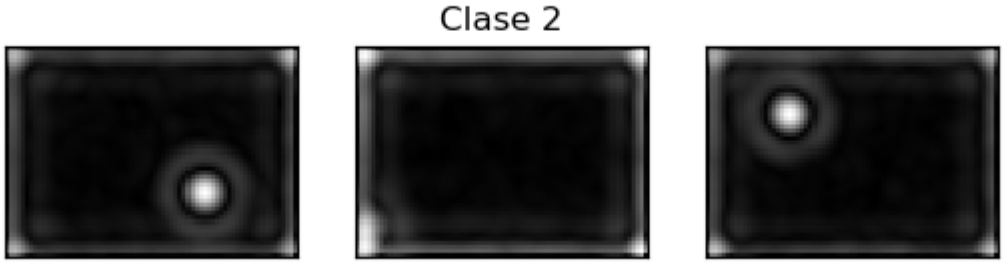

Figura 3.4: Imagen con daño de delaminación, clase 2

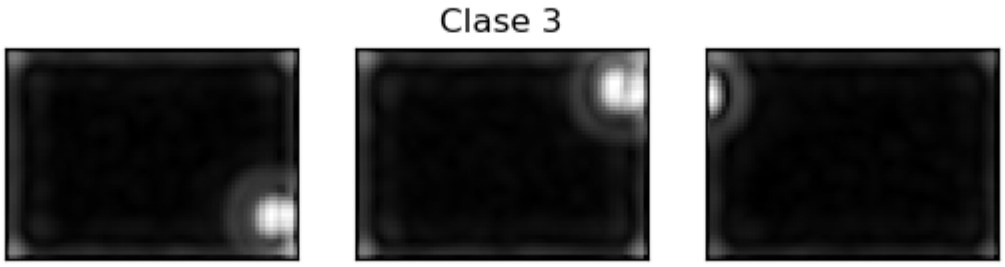

Figura 3.5: Imagen con daño de delaminación, clase 3

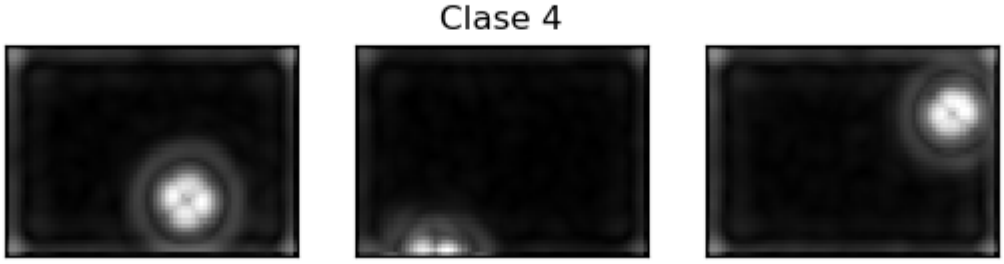

Figura 3.6: Imagen con daño de delaminación, clase 4

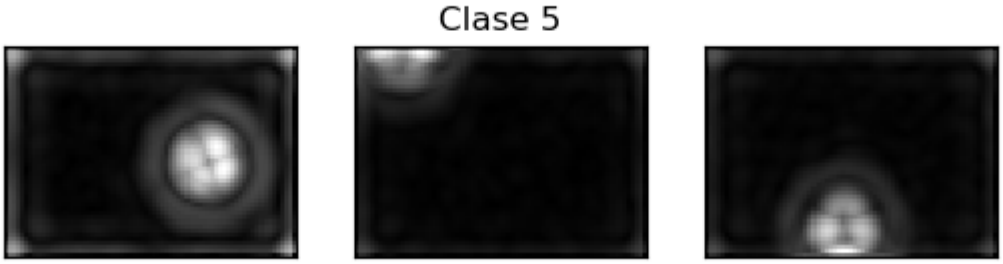

Figura 3.7: Imagen con daño de delaminación, clase 5

Además de los datos generados computacionalmente se cuenta con cinco imágenes de placas reales que fueron creadas con daño por delaminación. Estas se muestran a continuación y etiquetadas a las clases pertenecientes.

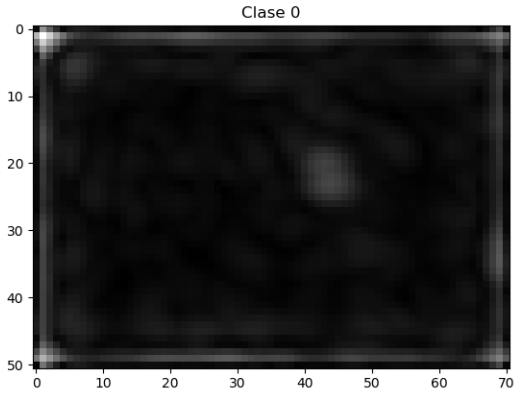

Figura 3.8: Imagen real con daño de delaminación, clase 0

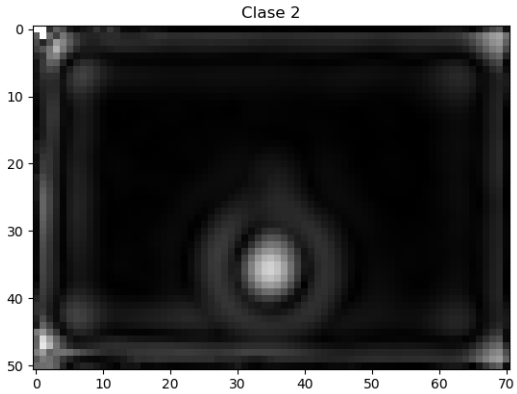

Figura 3.9: Imagen real con daño de delaminación, clase 2

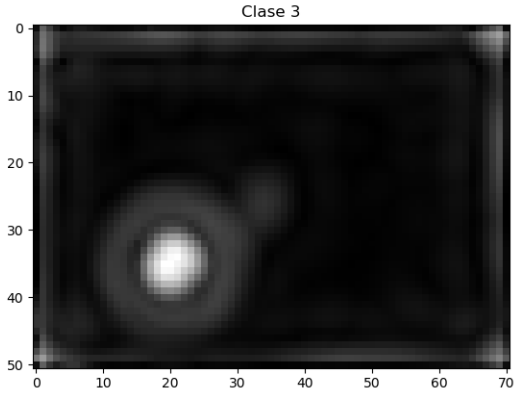

Figura 3.10: Imagen real con daño de delaminación, clase 3

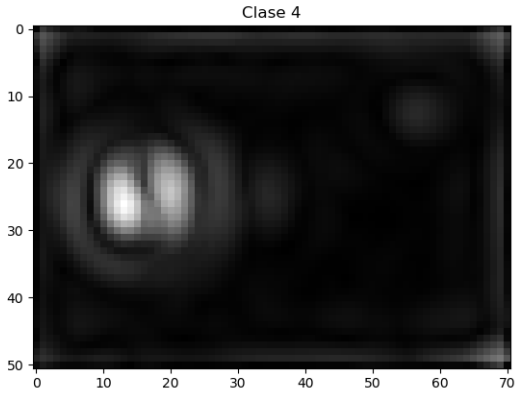

Figura 3.11: Imagen real con daño de delaminación, clase 4

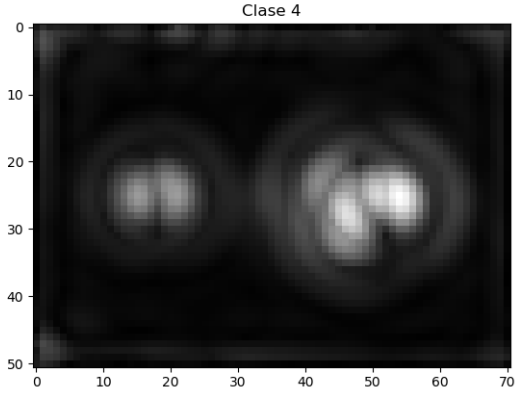

Figura 3.12: Imagen real con daño de delaminación, clase 4

Para cada algoritmo se utiliza un conjunto de datos, denominados conjuntos de entrenamiento, de validación y de prueba que se muestran a continuación.

Clasificación multiclase tipo 1

Para esta tarea se poseen 3500 imágenes diferente con distintos tamaños de daño. El conjunto de entrenamiento posee 2800 imágenes, el de validación posee 175 y el de prueba posee 525. La distribución de imágenes en cada clase para cada conjunto se aprecia en los siguientes histogramas:

Figura 3.13: Distribución datos de entrenamiento

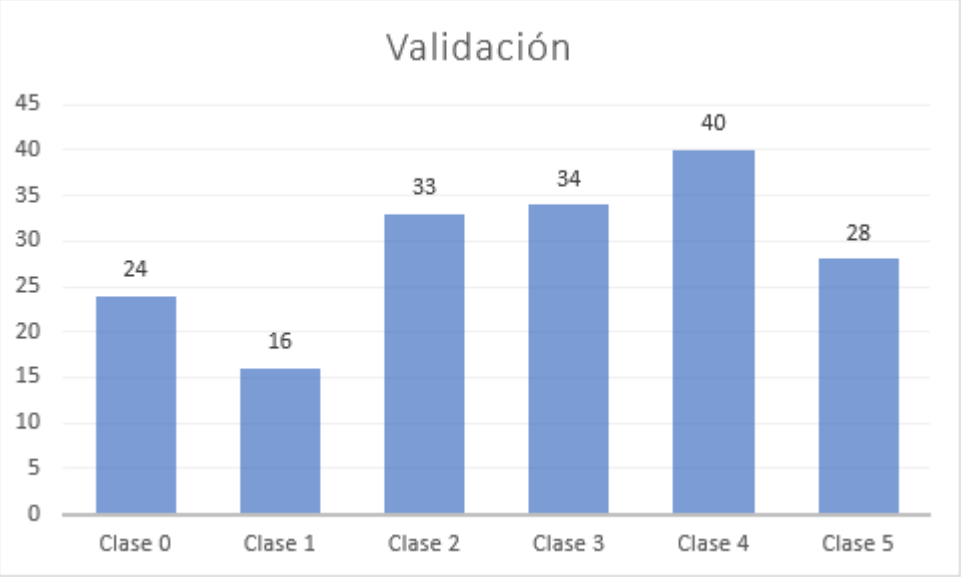

Figura 3.14: Distribución datos de validación

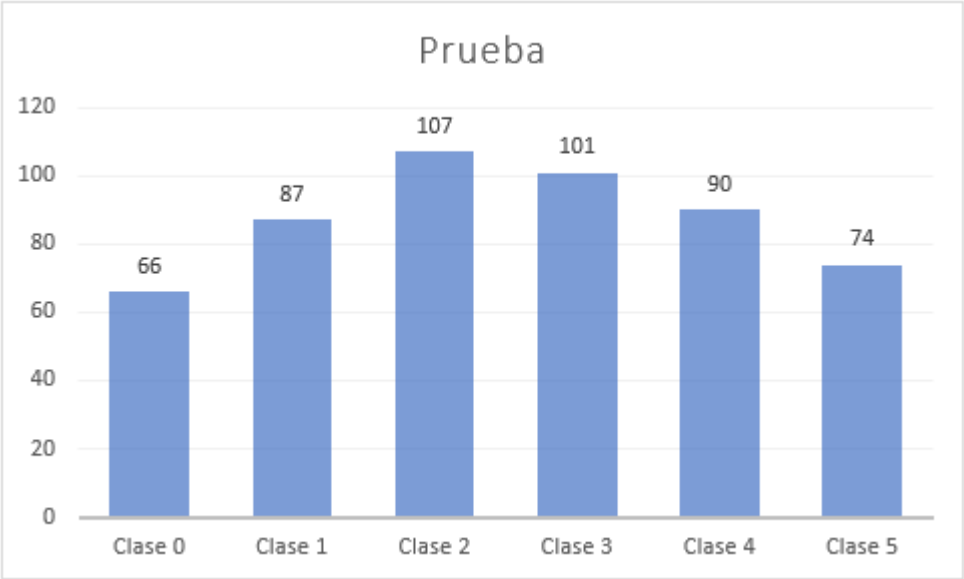

Figura 3.15: Distribución datos de prueba

One-Class Classification

Para el entrenamiento de este modelo, debido a que es utilizado para la detección de anomalías, solo se utilizan imágenes de clase 0. Para esto los conjuntos de entrenamiento/prueba se distribuyen como 1600/300 imágenes cada uno, mientras que el conjunto de validación consiste en en 175 imágenes, donde hay 35 de cada clase desde la 1 a la 5. Esto con el fin de ver la diferencia en el error entre el conjunto de entrenamiento y el de validación a la hora de entrenar.

Clasificación Binaria y Multiclase tipo 2

Uno de los métodos propuestos para la búsqueda de la clasificación es la combinación entre una clasificación binaria seguido por una clasificación multiclase. Para la distribución de datos de la clasificación binaria, se utilizaron los datos de entrenamiento/validación/prueba de la clasificación One-Class como datos de clase 0 y se combinaron los datos de la clase 1 a la 5 en un nuevo conjunto llamado clase 1.

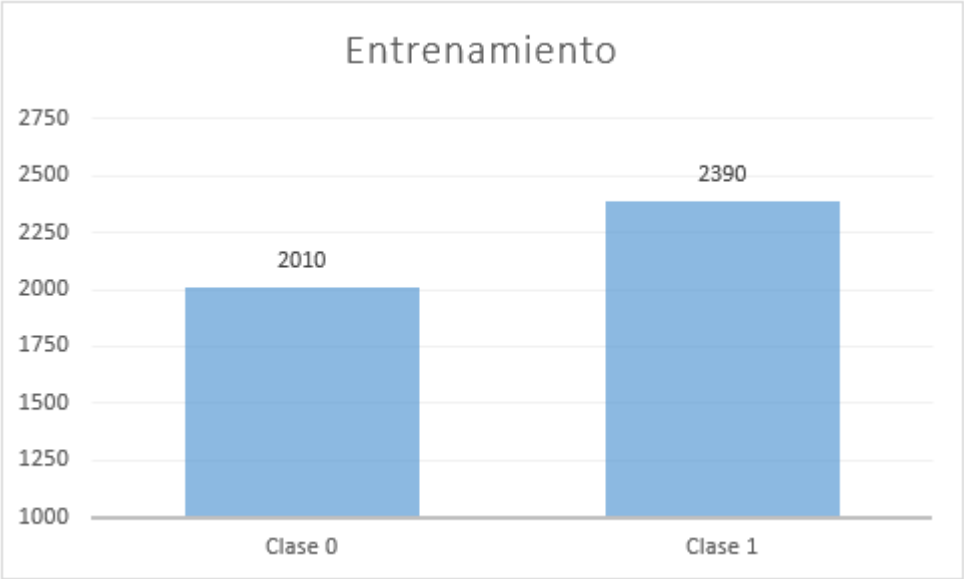

Figura 3.16: Distribución datos de entrenamiento

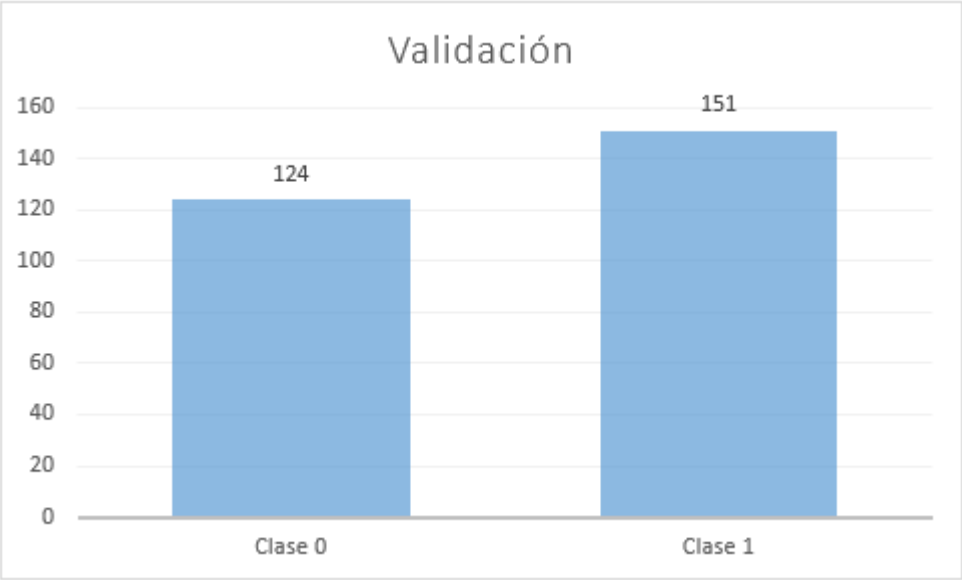

Figura 3.17: Distribución datos de validación

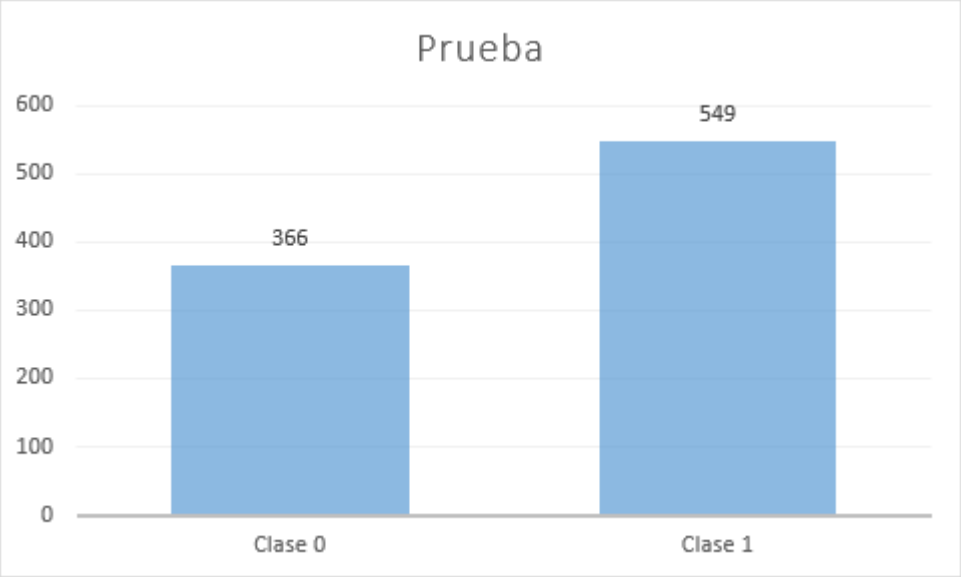

Figura 3.18: Distribución datos de prueba

En cambio, para la siguiente clasificación multiclase, se utilizan los siguientes conjuntos mostrados en las figuras 3.19 a la 3.21.

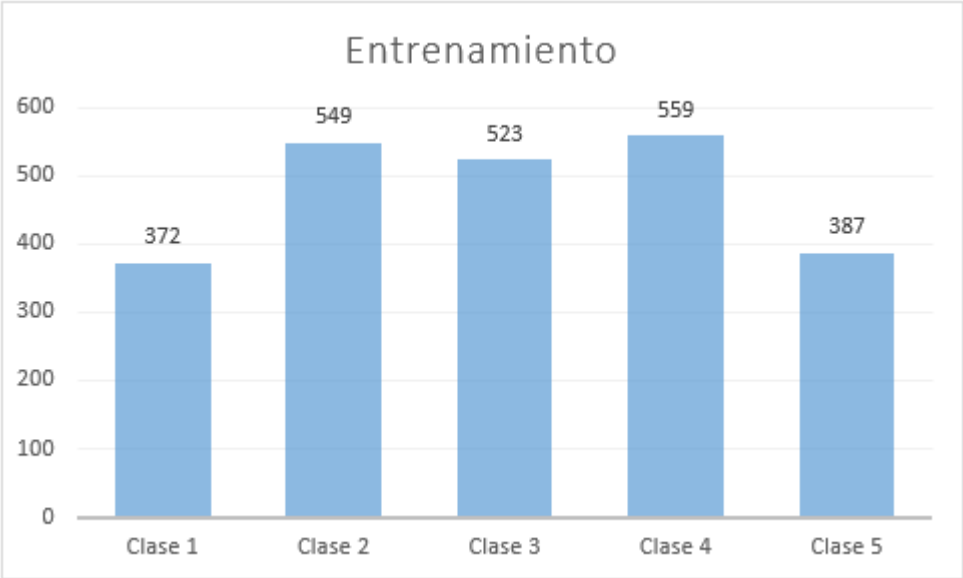

Figura 3.19: Distribución datos de entrenamiento

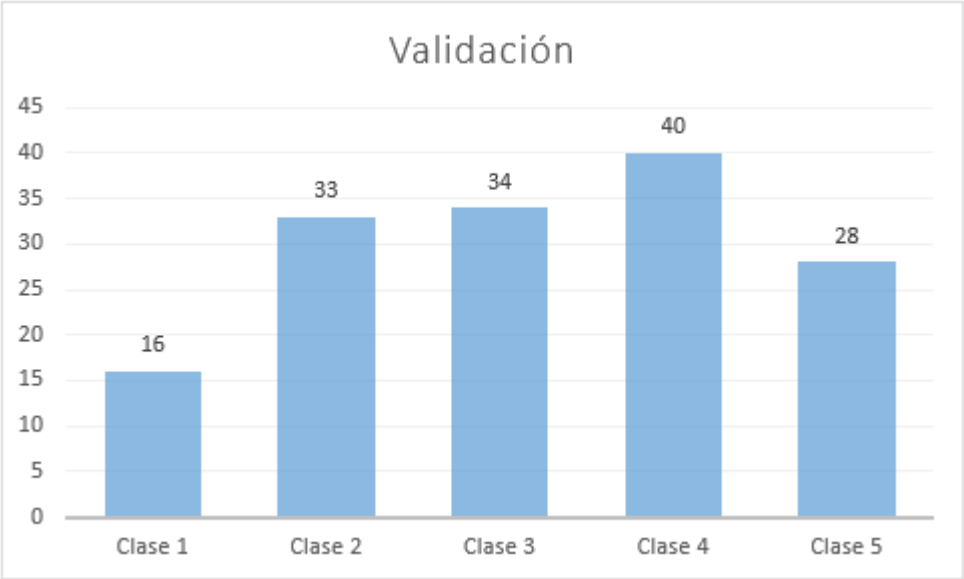

Figura 3.20: Distribución datos de validación

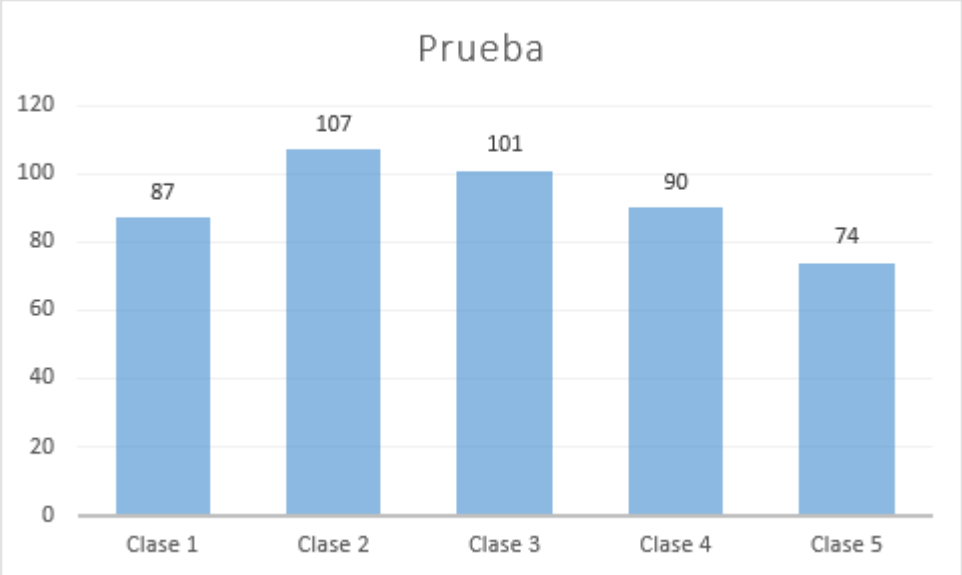

Figura 3.21: Distribución datos de prueba

3.2. One-Class Classification

Se presenta la arquitectura utilizada para el Variational Autoencoder del tipo One-Class Classification. Es necesario entender que las arquitecturas elegidas tanto en este ítem como en los otros modelos se hacen en base a prueba y error, y se seleccionan para este trabajo debido a presentar los mejores resultados dentro de las distintas pruebas. La figura 3.22 muestra la arquitectura del encoder, las flechas denotan las conexiones entre las distintas capas. Se

omitió por simplicidad el truco de reparametrización en la figura. La cantidad de unidades en las ultimas capas se denota como x, donde x puede ser 2, 4, 8 o 16 que fueron las diferentes dimensiones del vector latente probados.

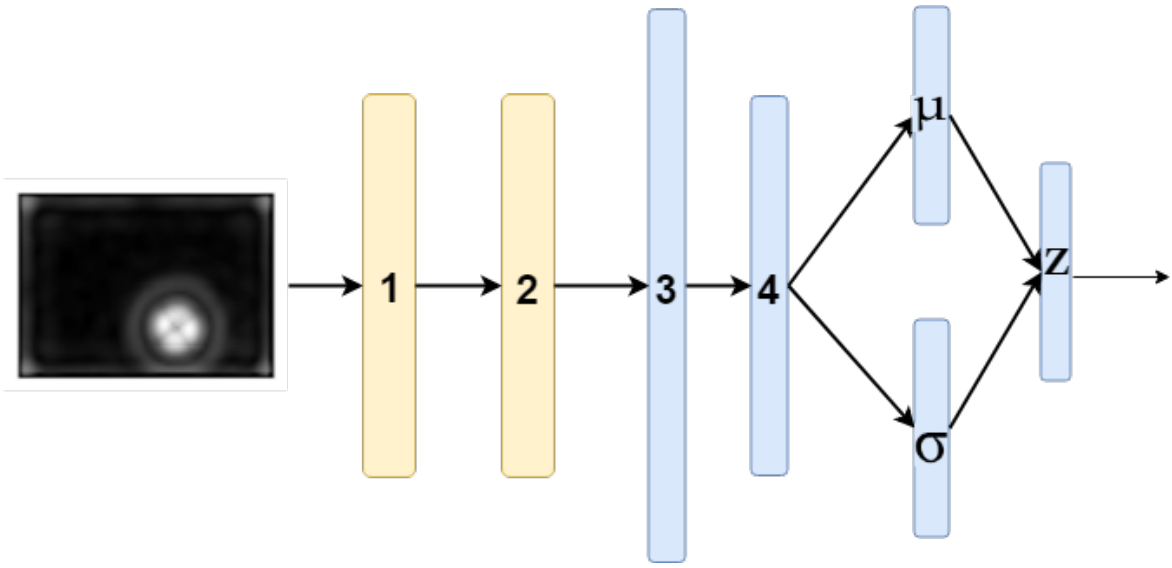

Figura 3.22: Encoder del Variational Autoencoder para One-Class Classification

Capa	**Tipo de capa**	**Activación**
1	Conv2D(filters= 32, kernel_size = (3,3), strides = (2,2))	ReLu
2	Conv2D(filters= 64, kernel_size = (3,3), strides = (2,2))	ReLu
3	Flatten(13056)	-
4	Dense(units=32)	ReLu
μ	Dense(units=x)	ReLu
σ	Dense(units=x)	ReLu
Z	Dense(units=x)	-

Tabla 3.1: Capas del Encoder para One-Class Classification

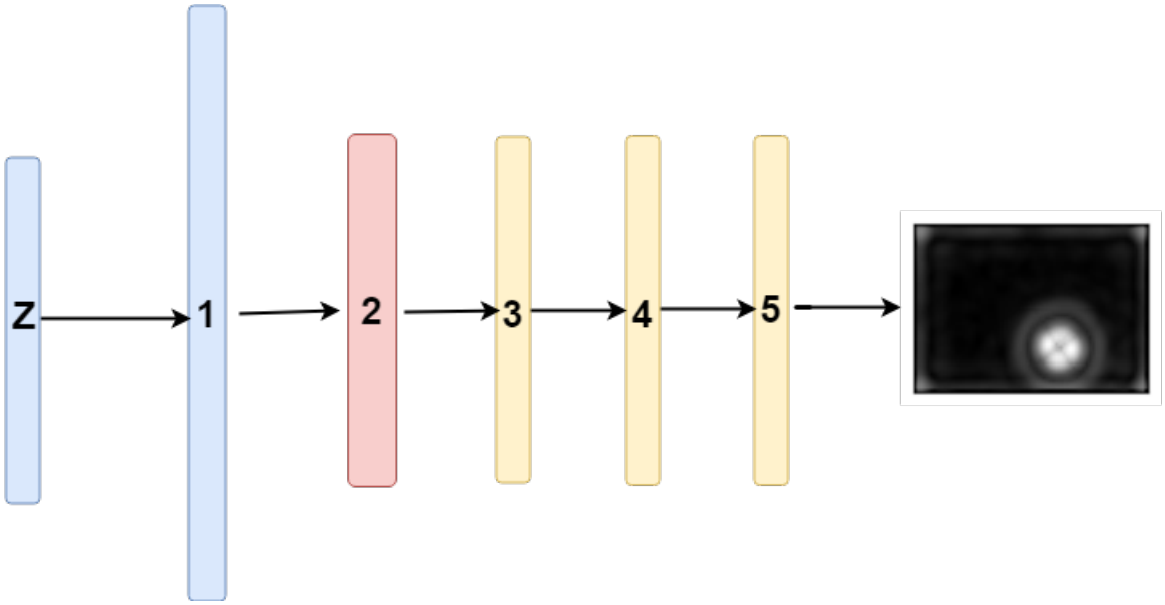

Figura 3.23: Decoder del Variational Autoencoder para One-Class Classification

La figura 3.23 y la tabla 3.2 muestran la arquitectura del decoder.

Capa	Tipo de capa	Activación
1	Dense(units = 13056)	ReLu
2	Reshape(12,17,64)	-
3	Conv2DTranspose(filters= 32, kernel_size = (3,3), strides = (2,2))	ReLu
4	Conv2DTranspose(filters= 64, kernel_size = (3,3), strides = (2,2))	ReLu
5	Conv2DTranspose(filters= 1, kernel_size = (7,7), padding = 'same')	Sigmoid

Tabla 3.2: Capas del Decoder para One-Class Classification

El variational Autoencoder se entrena de forma no supervisada durante 500 epochs, con un batch size de 32 y la optimización se hace mediante el algoritmo Adam con un learning rate de 0.001. Luego de entrenado con datos sin daño, se debe hacer la clasificación binaria entre datos con daño y datos sin daño, para esto, el error de reconstrucción se utiliza como métrica para la clasificación. Se utiliza un threshold por determinar, tal que si el error de reconstrucción de una imagen es mayor a este valor entonces esta imagen se clasifica como imagen con daño, caso contrario, se clasifica como imagen sin daño (clase 0).

3.3. Clasificación Multiclase tipo 1

Para la clasificación multiclase del tipo 1 se utilizan distintos algoritmos con el fin de buscar la mejor clasificación posible. En las tablas 3.3 y 3.4 se aprecian algoritmos de clasificación convolucional y perceptrón multicapa respectivamente. Ambos se entrenan por 200 epochs con un batch size de 64.

Capa	Tipo de capa	Activación
1	Conv2D(filters = 32, kernel = (7, 7))	ReLu
2	Conv2D(filters = 32, kernel = (7, 7), strides = (2,2), Dropout = 0.3)	ReLu
3	Conv2D(filters = 64, kernel = (5, 5))	ReLu
4	Conv2D(filters = 64, kernel = (5, 5), strides = (2,2), Dropout = 0.3)	ReLu
5	Flatten	-
6	Dense(units = 64, Dropout = 0.3)	ReLu
7	Dense(units = 6)	Softmax

Tabla 3.3: Arquitectura para clasificación multiclase tipo 1, CNN

Capa	Tipo de capa	Activación
1	Dense(units = 1024, Dropout = 0.3)	ReLu
2	Dense(units = 512, Dropout = 0.3)	ReLu
3	Dense(units = 256, Dropout = 0.3)	ReLu
4	Dense(units = 6)	Softmax

Tabla 3.4: Arquitectura para clasificación multiclase tipo 1, MLP

También se utilizó un Variational Autoencoder para clasificación multiclase tipo 1. Para la clasificación multiclase utilizando VAE, se entrena en primer lugar el VAE con los mismos datos que se utilizaron para la clasificación CNN y MLP. Luego de entrenado el VAE, unimos el encoder del VAE a una red clasificadora y nos olvidamos del decoder. El encoder mantiene los pesos que adquirió durante el entrenamiento del VAE, por lo que solo se entrenan los parámetros de las nuevas capas que se agregan posterior al encoder. La arquitectura del VAE es la misma utilizada para One-Class Classification, o sea, un VAE convolucional. La red para la clasificación se observa en la figura 3.24 y en la tabla 3.4.

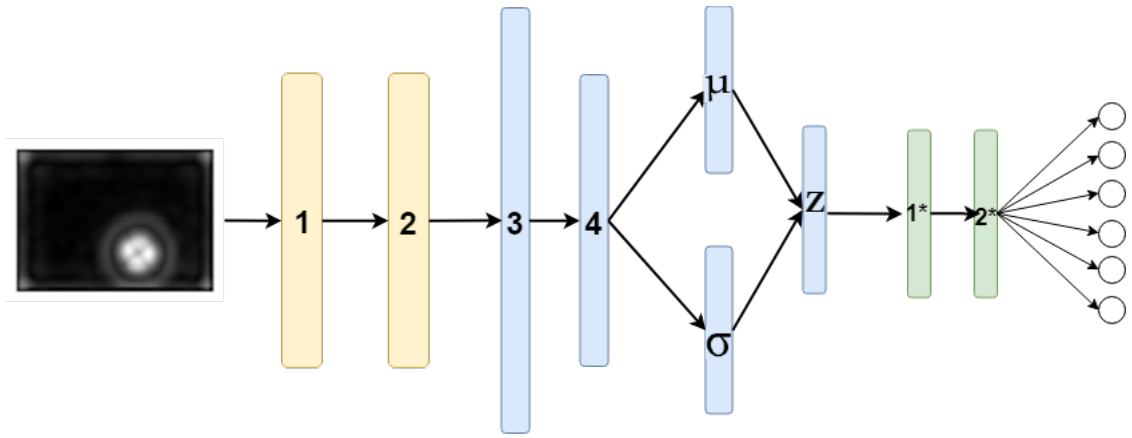

Figura 3.24: Encoder y red de clasificación multiclase del tipo 1

Capa	Tipo de capa	Activación
1*	Dense(units = 256, Dropout = 0.3)	ReLu
2*	Dense(units = 128, Dropout = 0.3)	ReLu
3*	Dense(units = 6)	Softmax

Tabla 3.5: Arquitectura para clasificación multiclase tipo 1 utilizando Variational Autoencoder Convolucional

Tambien se prueba utilizando un Variational Autoencoder del tipo MLP únicamente, sin capas convolucionales. La arquitectura del encoder se observa en la tabla 3.6 y del decoder en la tabla 3.7. Para la clasificación se utiliza el mismo método anterior, utilizando la misma arquitectura para la clasificación, mostrada en la tabla 3.5 .

Capa	Tipo de capa	Activación
1	Dense(units = 512)	ReLu
2	Dense(units = 256)	ReLu
mean	Dense(units = x)	ReLu
var	Dense(units = x)	ReLu
Z	Dense(units = x)	-

Tabla 3.6: Arquitectura del Encoder para clasificación multiclase tipo 1, MLP

Capa	Tipo de capa	Activación
1	Dense(units = 256)	ReLu
2	Dense(units = 512)	ReLu
3	Dense(units = 3621)	Sigmoid

Tabla 3.7: Arquitectura del Decoder para clasificación multiclase tipo 1, MLP

3.4. Clasificación Binaria y Multiclase tipo 2

Otra alternativa para la clasificación multiclase es separarla en dos tareas diferentes. En primer lugar se hace una clasifiación binaria, se prueba un algoritmo convolucional y uno simple MLP (figuras 3.8 y 3.9). Posteriormente, se entrena un algoritmo de clasificación multiclase que solo incluye las clases con daño, es decir, clases de la 1 a la 5. Se puede comparar el resultado de la clasificación multiclase tipo 1 con lo propuesto por este método al multiplicar el Accuracy obtenido por la clasificación binaria y la clasificación multiclase del tipo 2 y se compara con el obtenido por la clasificación de tipo 1.

Capa	Tipo de capa	Activación
1	Conv2D(filters = 32, kernel = (7, 7))	ReLu
2	Conv2D(filters = 32, kernel = (7, 7), strides = (2,2), Dropout = 0.3)	ReLu
3	Conv2D(filters = 64, kernel = (5, 5))	ReLu
4	Conv2D(filters = 64, kernel = (5, 5), strides = (2,2), Dropout = 0.3)	ReLu
5	Flatten	-
6	Dense(units = 64, Dropout = 0.3)	ReLu
7	Dense(units = 1)	Sigmoid

Tabla 3.8: Arquitectura de clasificación binaria convolucional

Capa	Tipo de capa	Activación
1	Dense(units = 512, Dropout = 0.3)	ReLu
2	Dense(units = 512, Dropout = 0.3)	ReLu
3	Dense(units = 1)	Sigmoid

Tabla 3.9: Arquitectura de clasificación binaria MLP

Capa	Tipo de capa	Activación
1	Conv2D(filters = 32, kernel = (7, 7))	ReLu
2	Conv2D(filters = 32, kernel = (7, 7), strides = (2,2), Dropout = 0.3)	ReLu
3	Conv2D(filters = 64, kernel = (5, 5))	ReLu
4	Conv2D(filters = 64, kernel = (5, 5), strides = (2,2), Dropout = 0.3)	ReLu
5	Flatten	-
6	Dense(units = 64, Dropout = 0.3)	ReLu
7	Dense(units = 5)	Softmax

Tabla 3.10: Arquitectura para clasificación multiclase tipo 2, CNN

Capítulo 4

Resultados y Análisis

4.1. One-Class Classification

En la presente sección se encuentran los resultados para el algoritmo de clasificación de daño del tipo One-Class Classification. Se entrena un Variational Autoencoder utilizando únicamente imágenes sin daño, sin embargo, el conjunto de validación está formado por imágenes con daño, específicamente, 35 imágenes de cada clase (sin incluir clase 0). Se utiliza este conjunto de validación con el fin de observar como el error de reconstrucción disminuye por época para los datos de entrenamiento, ya que el algoritmo aprenderá a reconstruir las imágenes originales, sin embargo, no podrá reconstruir correctamente imágenes con daño (alto error de reconstrucción), implicando que estos datos son anómalos y por ende el algoritmo puede reconocer que son placas con daño por delaminación.

En la figura 4.1 se observa el fenómeno mencionado anteriormente, donde se muestra el entrenamiento para el VAE en el caso que el vector latente tiene una dimensión de 16. En la tabla 4.1 se muestran los resultados del entrenamiento para el mismo algoritmo, al variar la dimensión del vector latente.

Entrenamiento VAE						
Dimensión Z	Loss	KL_loss	recon_loss	val_loss	van_KL_loss	val_recon_loss
z=16	992.3671	0.0659	992.3012	1112.9496	0.6730	1112.2765
z=8	992.5239	2.818e-04	992.5236	1136.2517	3.017e-04	1136.2515
z=4	992.4784	0.0883	992.3901	1124.0855	0.9710	1123.1144
z=2	992.5153	2.365e-05	992.5153	1136.4649	5.54e-05	1136.4648

Tabla 4.1: Resultados de entrenamiento VAE CNN One-Class

Una vez entrenado el VAE con los daños sin daño. Se procede a buscar un threshold que nos permita identificar la presencia de daño. Para esto, utilizamos el error de reconstrucción como métrica de separación. La suposición es que el error de reconstrucción promedio será bajo para los datos sin daño y alto para los datos con daño. En la figura 4.2 se observa un histograma del error de reconstrucción para datos de las distintas clases.

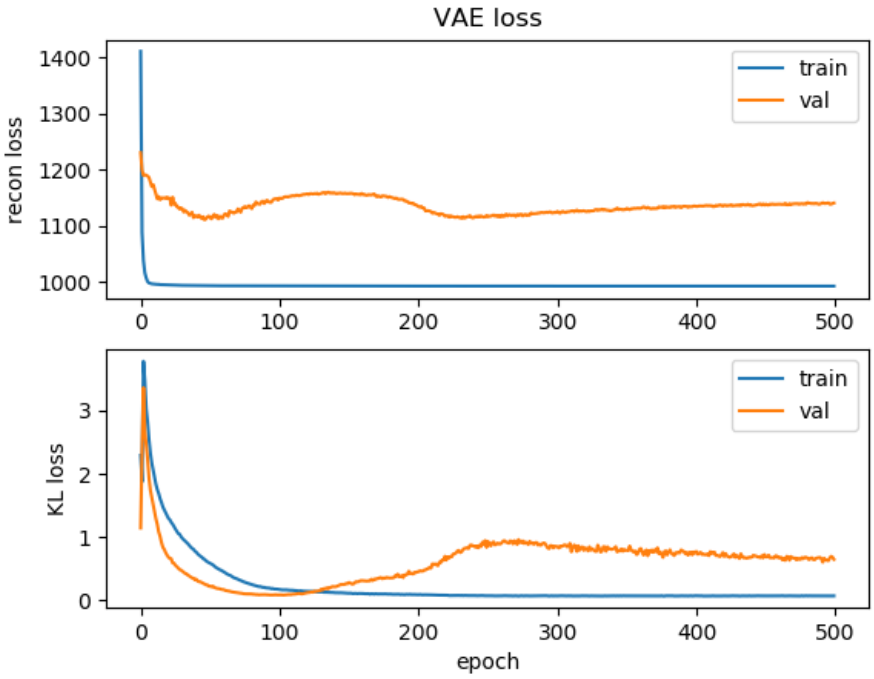

Figura 4.1: Entrenamiento del VAE para vector latente z=16, One-Class Clasification

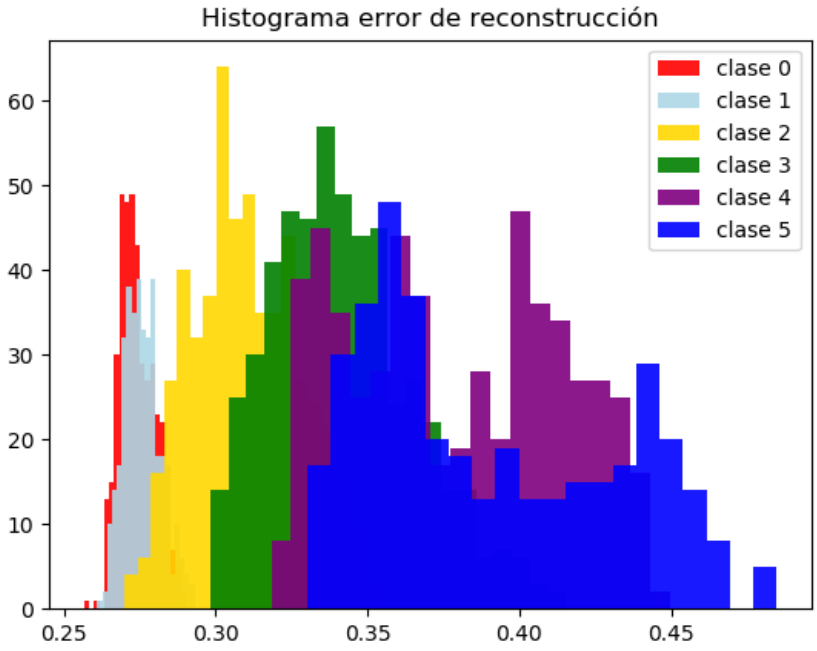

Figura 4.2: Error de reconstrucción para todas las clases

Para observar mejor los datos, separamos el histograma en 5. Las siguientes figuras corresponden al error de reconstrucción entre la clase 0 (sin daño) y las clases 1,2,3,4 y 5.

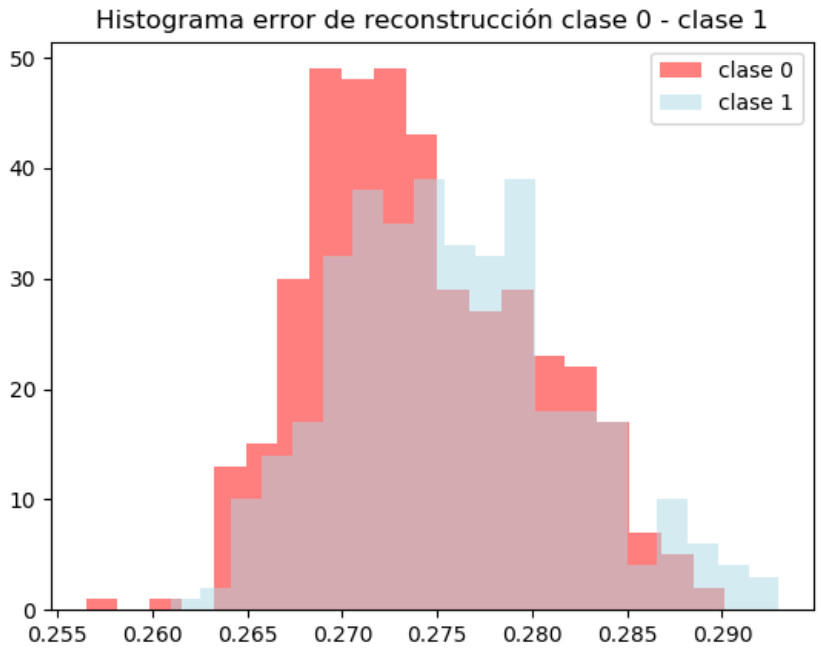

Figura 4.3: Comparación de error de reconstrucción para clase 0 y clase 1

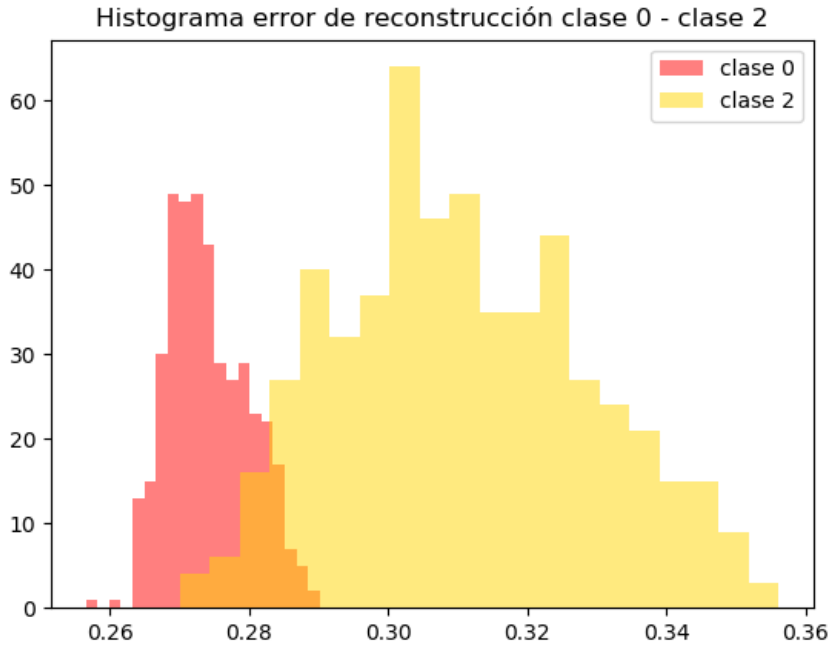

Figura 4.4: Comparación de error de reconstrucción para clase 0 y clase 2

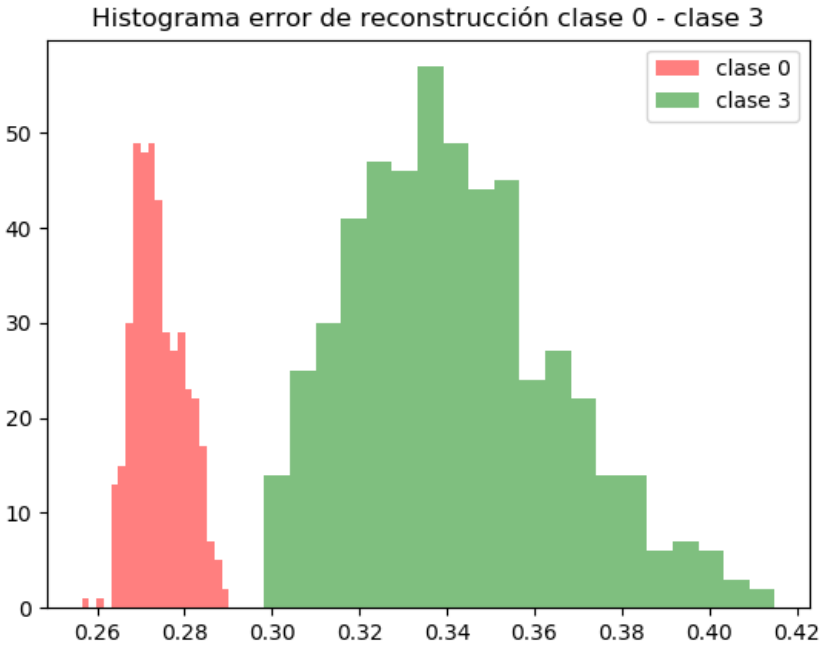

Figura 4.5: Comparación de error de reconstrucción para clase 0 y clase 3

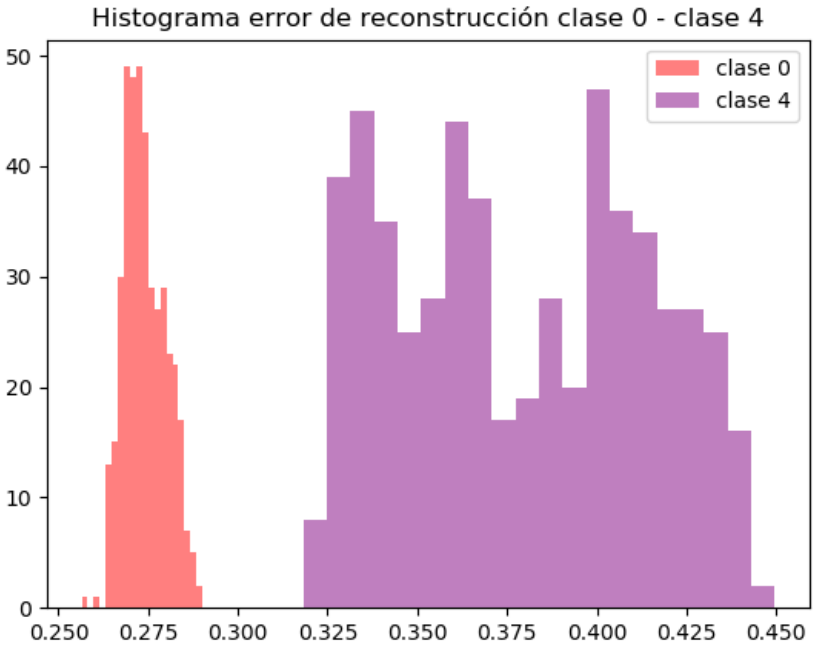

Figura 4.6: Comparación de error de reconstrucción para clase 0 y clase 4

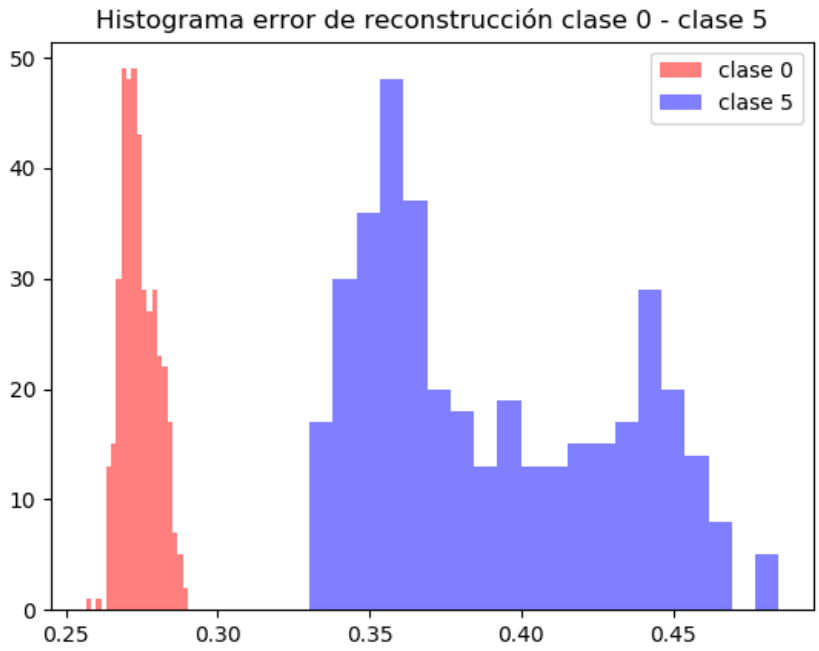

Figura 4.7: Comparación de error de reconstrucción para clase 0 y clase 5

Como se aprecia existe una separación cuando el daño pertenece a la clase 3, 4 y 5. Sin embargo, la clase 0 y la clase 1 tienen un overlap casi completo. Para definir un buen threshold la figura 4.4 será la mas relevante, un threshold muy alto y aumentarán la cantidad de falsos positivos, uno muy bajo y el Accuracy total disminuirá y aumentaran los falsos negativos también.

Un posible threshold si solo se cuenta con los datos sin daño, es fijar el valor máximo de la reconstrucción para la clase 0. Con esto nos aseguramos de clasificar correctamente toda la clase 0, obteniendo ningún falso negativo. En la figura 4.8 se observa la separación entre las clases al utilizar este threshold, de valor **0.2901** y en la figura 4.9 se observa la matriz de confusión en este caso, con un Accuracy de **0.8414**.

Figura 4.8: Error de reconstrucción entre clase 0 y clase 2, cuando el threshold de separación es 0.2901

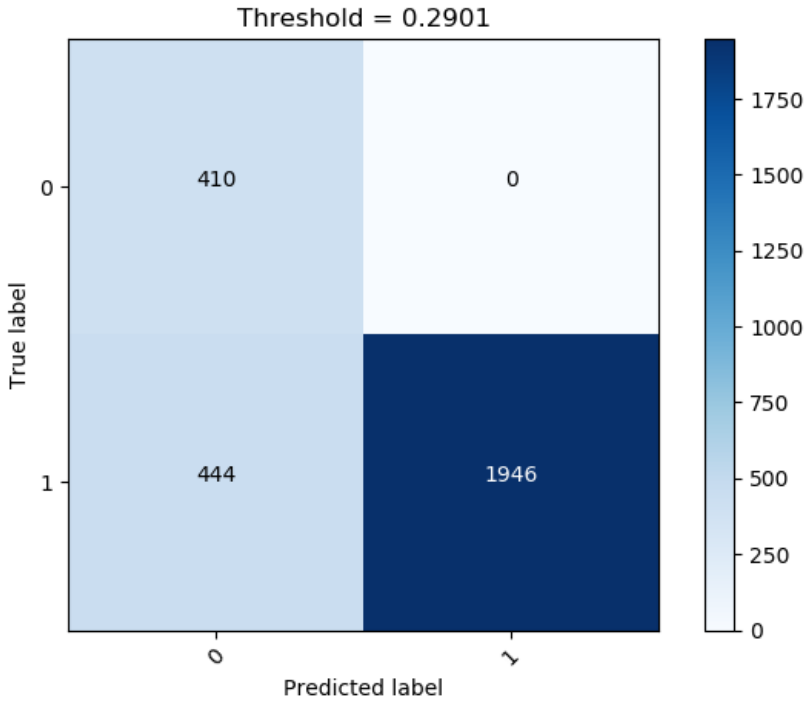

Figura 4.9: Matriz de confusión cuando el threshold es de 0.2901

Otra posibilidad de threshold es realizar una elección manual por parte del programador,

quedará a criterio del mismo, entendiendo que existe un trade-off al variar un threshold, ya que cambiará el Accuracy de la clasificación y por lo mismo, los falsos positivos y falsos negativos. En la tabla 4.2 se observan varias matrices de confusión y su respectivo Accuracy para diferentes thresholds. En negrita se encuentran dos thresholds atractivos, uno con el máximo Accuracy y otro con el menor numero de falsos positivos.

Valor Threshold	Confusion Matrix		Accuracy
0.278	303 263	107 2127	**0.8678**
0.2785	311 272	99 2118	0.8675
0.2787	317 276	93 2114	0.8682
0.2788	319 276	91 2114	**0.8689**
0.2790	321 279	89 2111	0.8685
0.28	336 297	74 2093	0.8675
0.2824	373 340	37 2050	0.8653

Tabla 4.2: Matrices de confusión para diferentes valores de Threshold

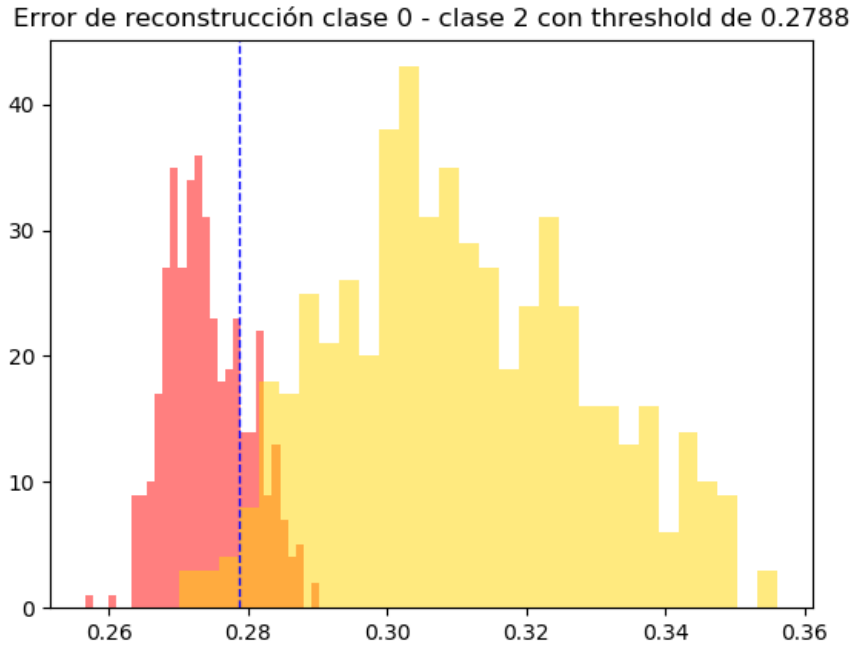

Figura 4.10: Histograma de reconstrucción entre clase 0 y clase 2 con threshold = 0.2788

En las figuras 4.10 y 4.11 se aprecia la separación existente para el threshold seleccionado. Con este threshold se logra un Accuracy de 86.89 % , donde el error proviene casi por completo de la mala clasificación de la clase 1. La clase 1 posee 372 imágenes, de las cuales 269 fueron mal clasificadas (falsos positivos) y la clase 2 tiene 549 imágenes de las cuales 7 fueron mal clasificadas como falsos positivos también. Las clases 3,4 y 5 se clasifican correctamente.

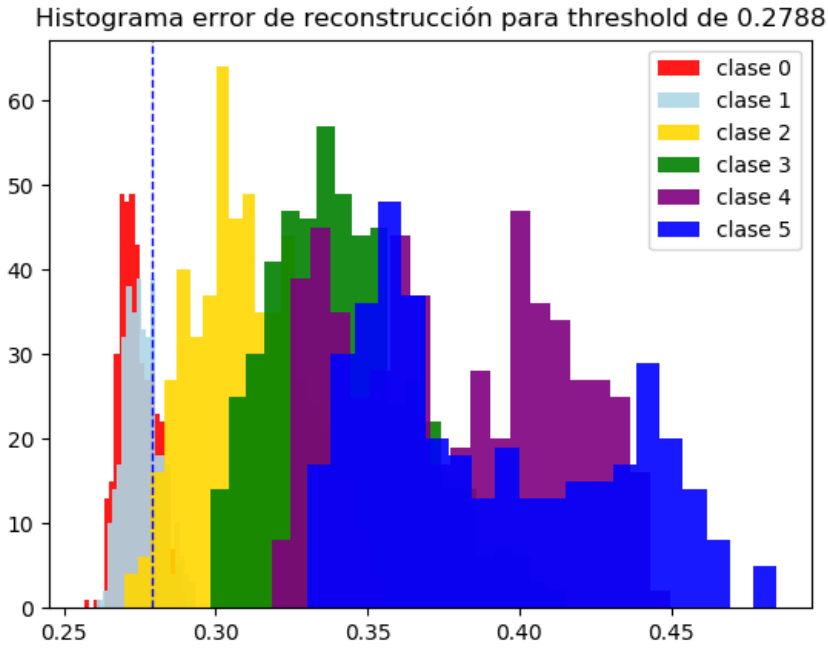

Figura 4.11: Histograma de reconstrucción con threshold = 0.2788

4.2. Clasificación Multiclase tipo 1

En esta sección se presentan los resultados obtenidos para la clasificación convolucional multiclase tipo 1 y MLP multiclase tipo 1.

Red convolucional

En las figuras 4.12 y 4.13 se observa el entrenamiento de la red convolucional de clasificación tipo 1. En la tabla 4.3 se encuentran los resultados finales del entrenamiento y de la fase de prueba.

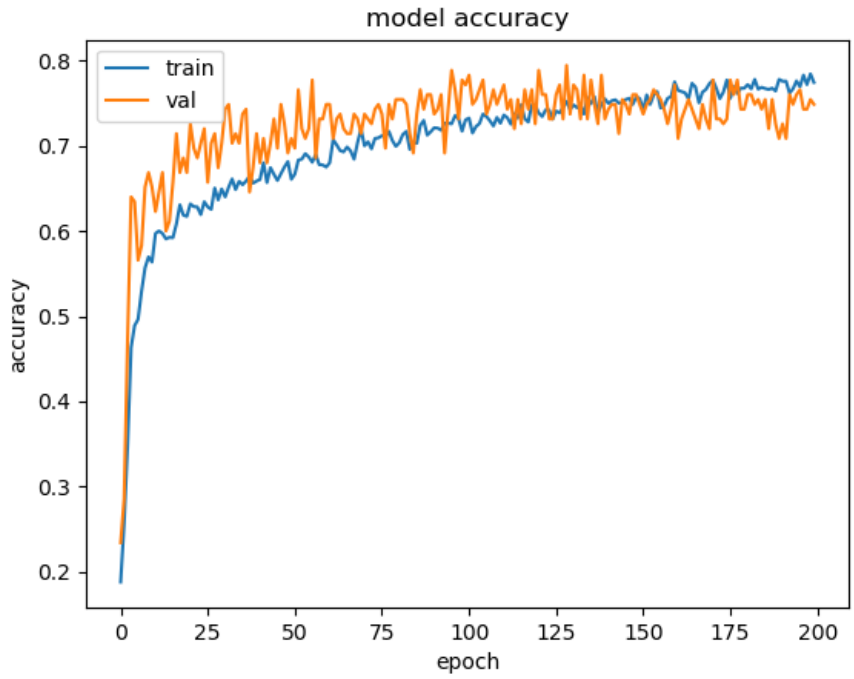

Figura 4.12: Accuracy clasificación multiclase CNN tipo 1

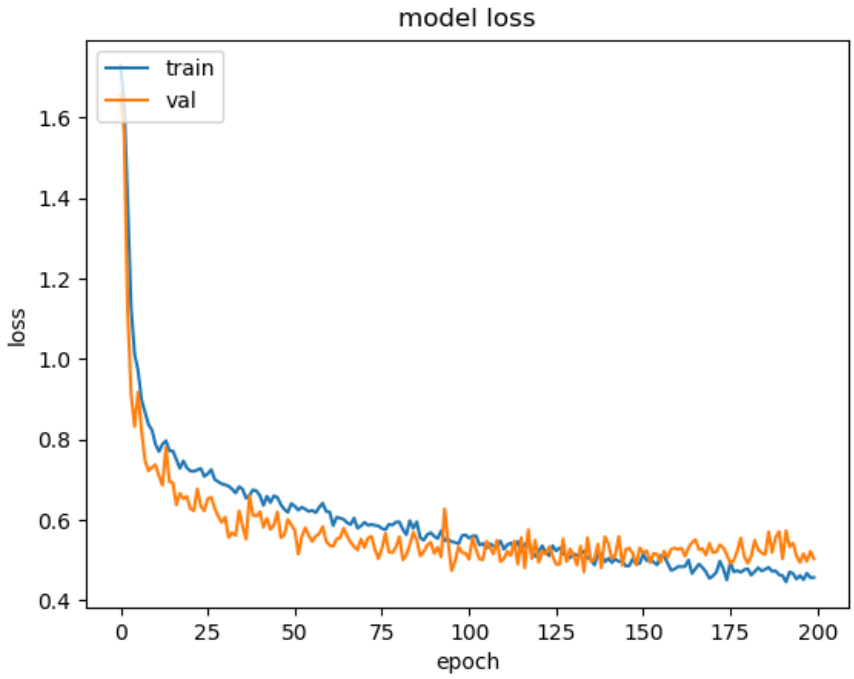

Figura 4.13: Loss clasificación multiclase CNN tipo 1

Entrenamiento				Prueba	
Loss	Accuracy	Val_loss	Val_Accuracy	Loss	Accuracy
0.4575	0.7743	0.5039	0.7486	0.5058	0.7213

Tabla 4.3: Resultados clasificación multiclase CNN tipo 1

En la figura 4.14 se encuentra la matriz de confusión para este problema de clasificación.

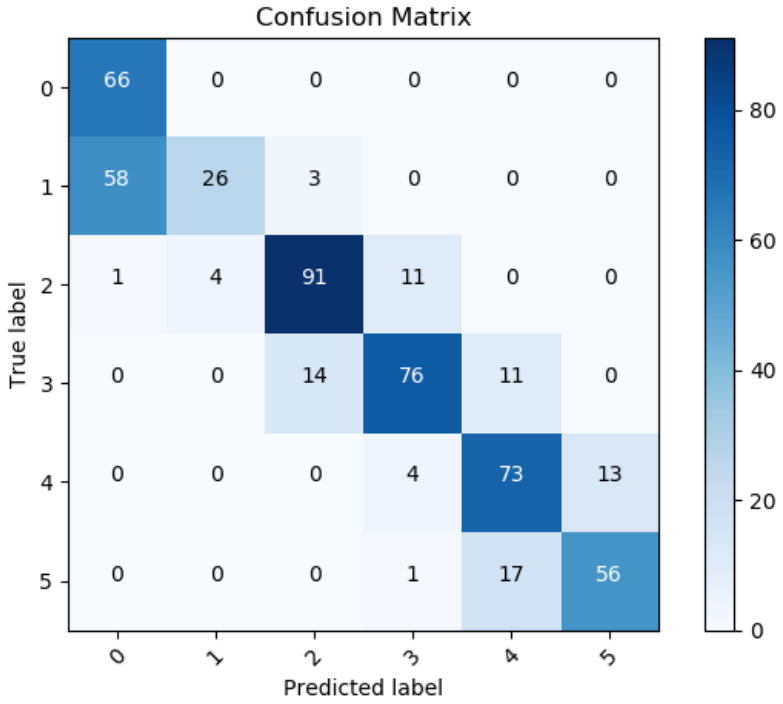

Figura 4.14: Matriz de confusión para clasificación multiclase CNN tipo 1

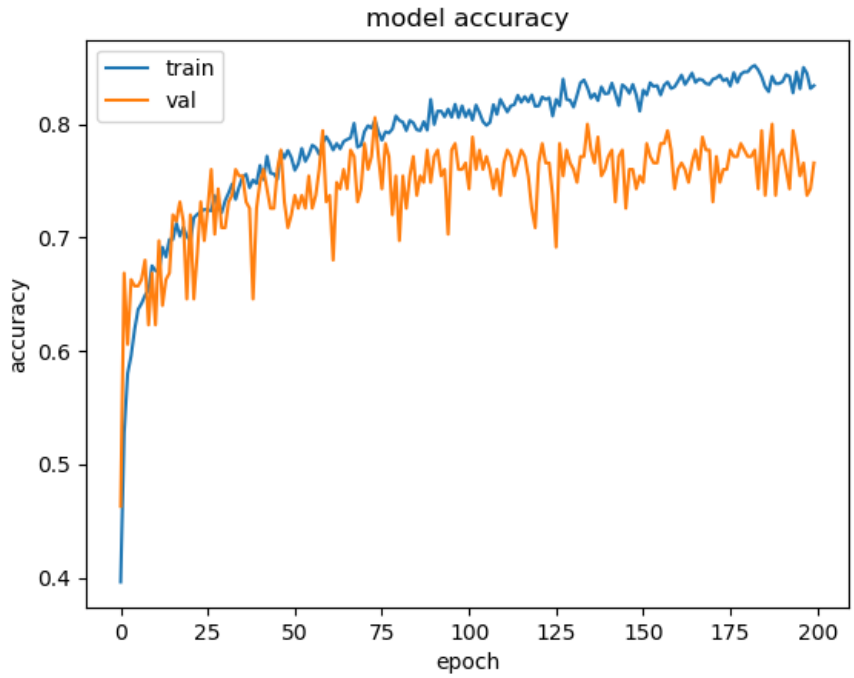

Figura 4.15: Accuracy clasificación multiclase MLP tipo 1

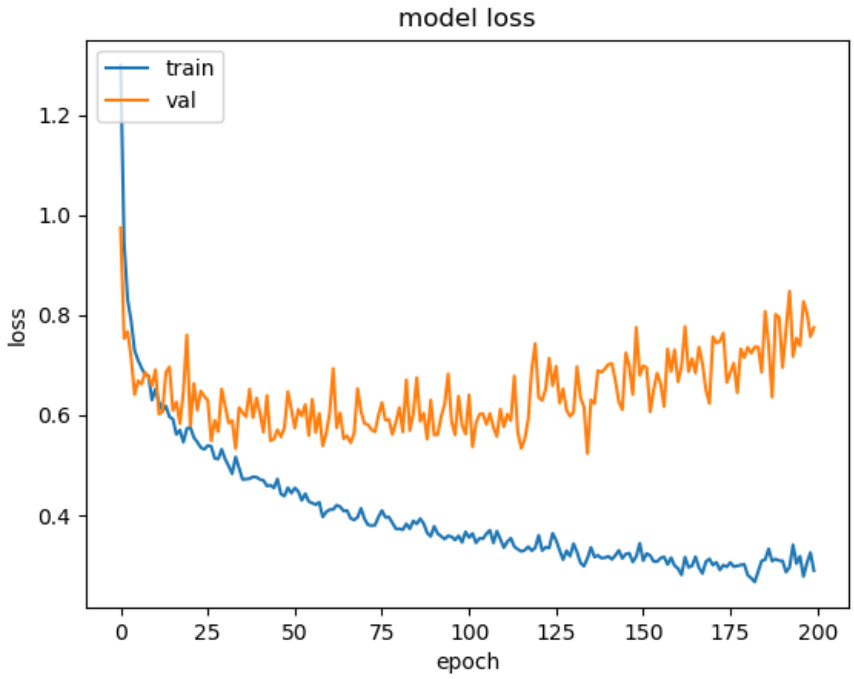

Figura 4.16: loss clasificación multiclase MLP tipo 1

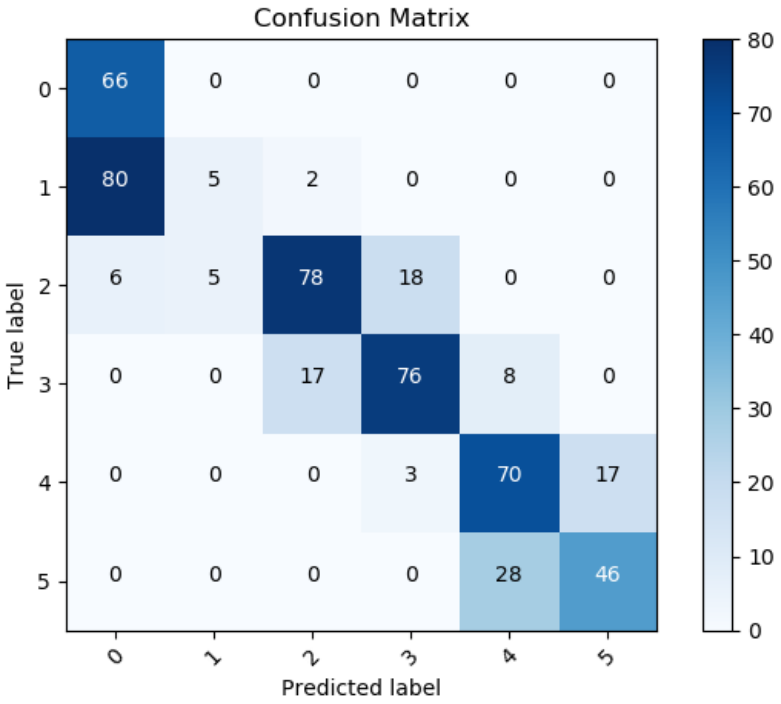

Figura 4.17: Matriz de confusión clasificación multiclase MLP tipo 1

Entrenamiento				**Prueba**	
Loss	Accuracy	Val_loss	Val_Accuracy	Loss	Accuracy
0.2895	0.8339	0.7754	0.7657	0.7995	0.6989

Tabla 4.4: Resultados clasificación multiclase MLP tipo 1

Variational Autoencoder Convolucional

Se presentan los resultados para clasificación multiclase tipo 1 utilizando Variatioanl Autoencoders convolucionales. En primer lugar se entrena el Variational Autoencoder de forma no supervisada, para que pueda recrear las imagenes originales de la mejor forma posible. La reconstrucción se observa en la figura 4.18, esta reconstrucción corresponde al VAE convolucional cuando la dimensión del vector latente es de 16.

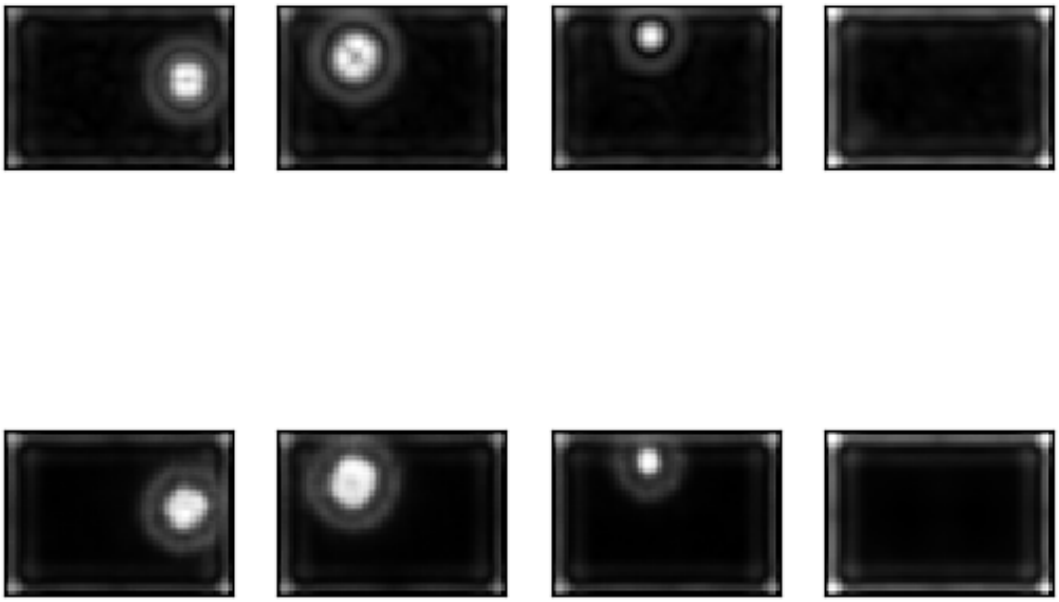

Figura 4.18: Arriba imágenes originales y abajo la reconstrucción del VAE CNN

En la siguiente tabla se encuentran los valores finales del entrenamiento para el VAE variando su dimensión del vector latente z.

Entrenamiento VAE						
Dimensión Z	Loss	KL_loss	recon_loss	val_loss	van_KL_loss	val_recon_loss
z=16	922.5095	2.0052	920.5043	915.1105	1.9905	913.12
z=8	923.8775	2.3599	921.5177	916.3013	2.4264	913.8749
z=4	931.948	3.3075	928.6405	931.2373	3.3367	927.9006
z=2	941.9765	3.8736	938.1029	944.5738	4.01	940.5638

Tabla 4.5: Resultados de entrenamiento VAE CNN

Luego de entrenado el VAE de forma no supervisada, se entrena para la clasificación según se explicó en el capítulo anterior. Los resultados de esta clasificación se encuentran en la siguiente tabla, para distintos valores del vector latente.

Entrenamiento					Prueba	
Dimension Z	Loss	Accuracy	Val_loss	Val_accuracy	Loss	Accuracy
z=16	0.4605	0.7679	0.5306	0.7429	0.5872	0.6857
z=8	0.5152	0.7231	0.5688	0.7163	0.6112	0.6439
z=4	0.6479	0.6793	0.7941	0.68	0.8246	0.5942
z=2	0.9967	0.4861	1.0415	0.4743	1.102	0.4285

Tabla 4.6: Resultados clasificación VAE CNN

Finalmente, se muestra la matriz de confusión para la clasificación utilizando VAE convolucionales. En este caso, solo se muestra la matriz de confusión cuando la dimensión latente tiene un largo de 16, ya que es la que mejor resultados entrega.

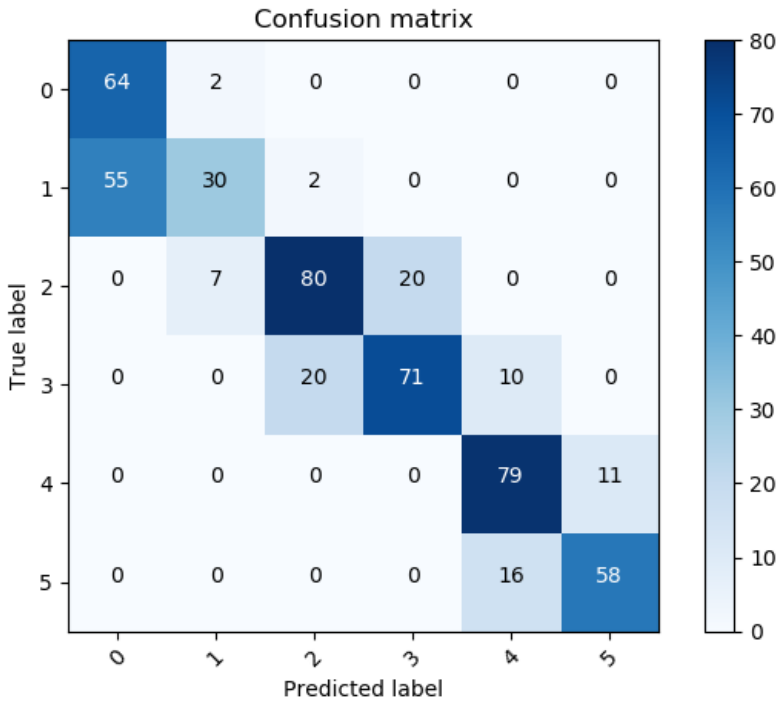

Figura 4.19: Matriz de confusión para clasificación multiclase tipo 1 utilizando VAE CNN

Variational Autoencoder MLP

Se presentan los resultados para clasificación multiclase tipo 1 utilizando Variatioanl Autoencoders MLP. El procedimiento para la clasificación es el mismo de los VAE convolucionales. La reconstrucción se observa en la figura 4.20, esta reconstrucción corresponde al VAE MLP cuando la dimensión del vector latente es de 16.

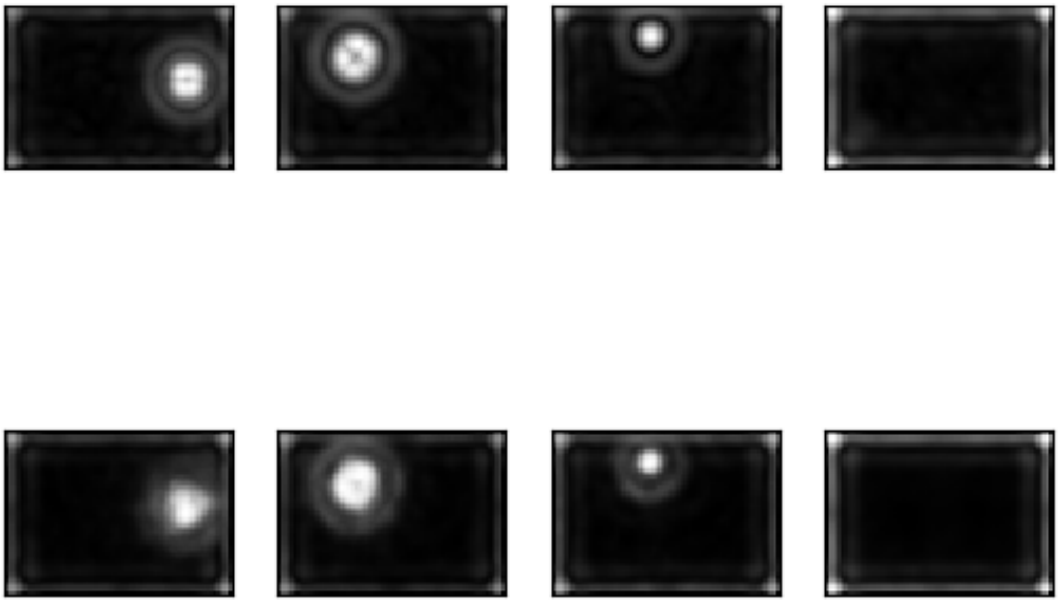

Figura 4.20: Arriba imágenes originales y abajo la reconstrucción del VAE MLP

En la siguiente tabla se encuentran los valores finales del entrenamiento para el VAE variando su dimensión del vector latente z.

Entrenamiento VAE						
Dimensión Z	Loss	KL_loss	recon_loss	val_loss	van_KL_loss	val_recon_loss
z=16	922.4472	2.0187	920.4285	915.9716	2.1174	913.8542
z=8	924.6340	2.3941	922.2399	918.4607	2.5849	915.8759
z=4	928.6650	2.7754	925.8895	922.2785	2.9528	919.3257
z=2	940.4287	3.4746	936.9541	945.2789	3.5979	941.6810

Tabla 4.7: Resultados de entrenamiento VAE MLP

Luego del entrenamiento del VAE de forma no supervisada, se entrena de forma supervisada con los datos etiquetados para la clasificación de tipo 1. Los resultados para distintas dimensiones del vector latente se encuentran en la siguiente tabla, mientras que la matriz de

confusión para z=16 se encuentra en la figura 4.21 , al ser el caso con mejor desempeño de los probados.

	Entrenamiento				Prueba	
Dimension Z	Loss	Accuracy	Val_loss	Val_accuracy	Loss	Accuracy
z=16	0.3763	0.8007	0.5691	0.7486	0.5872	0.6656
z=8	0.4211	0.7793	0.5578	0.7429	0.6112	0.6571
z=4	0.5410	0.7086	0.5808	0.7200	0.8246	0.6036
z=2	1.0665	0.4618	1.1484	0.4171	1.102	0.4247

Tabla 4.8: Resultados clasificación VAE MLP

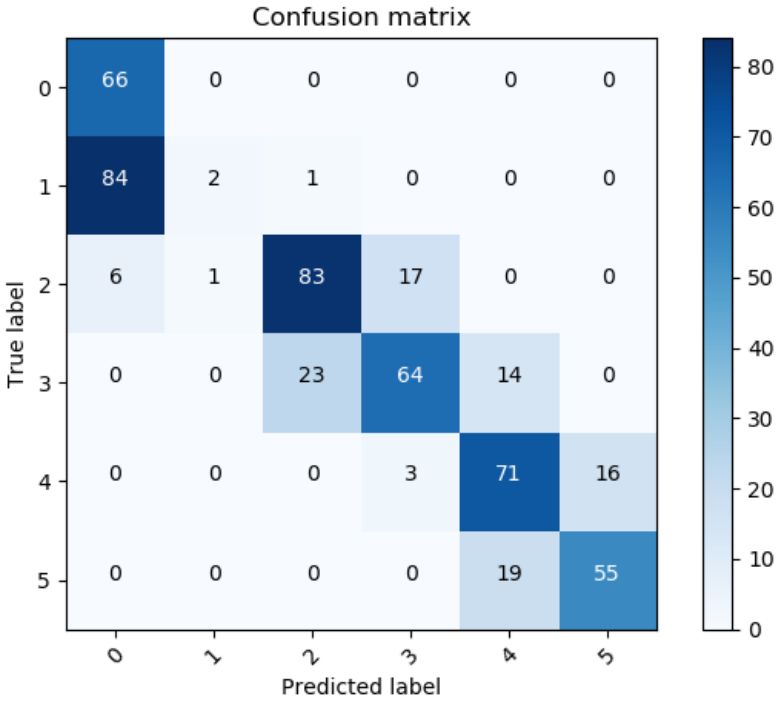

Figura 4.21: Matriz de confusión para clasificación multiclase tipo 1 utilizando VAE MLP

4.3. Clasificación Binaria y Multiclase tipo 2

Como se explicó en el capítulo anterior, este proceso consta de dos partes. En primer lugar se hace una clasificación binaria y luego se realiza una clasificación multiclase para identificar tamaño de daño, solo entre las clases 1 a la clase 5.

4.3.1. Clasificación binaria

Red Convolucional

Entre las figuras 4.22 y 4.24 se presentan los resultados para la red convolucional de clasificación binaria.

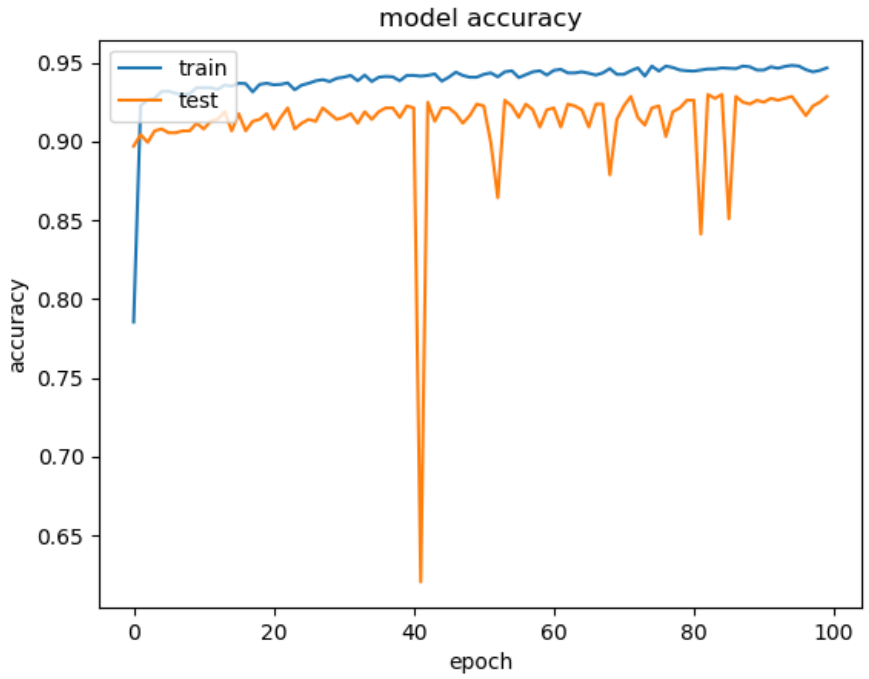

Figura 4.22: Accuracy clasificación binaria convolucional

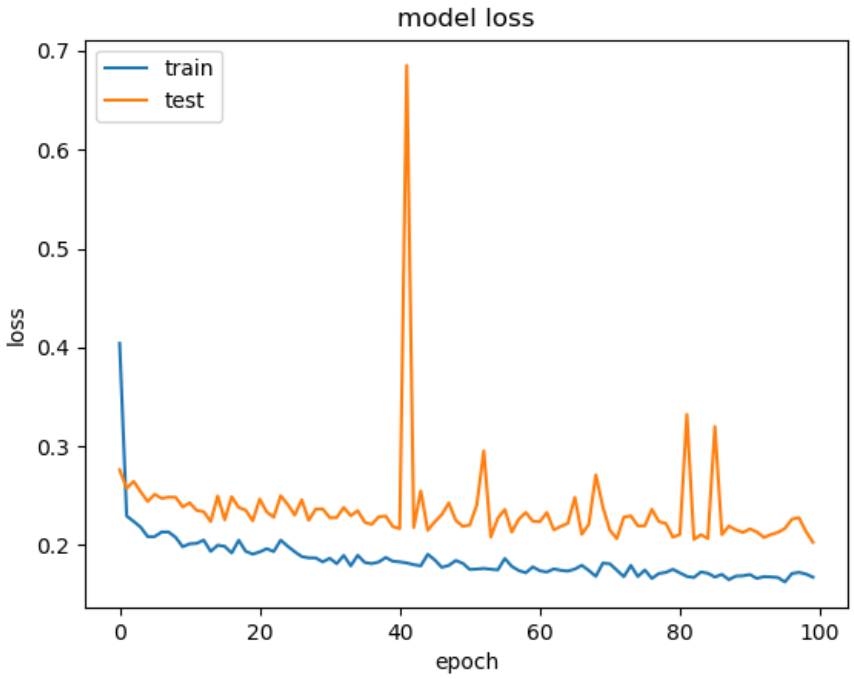

Figura 4.23: Loss clasificación binaria convolucional

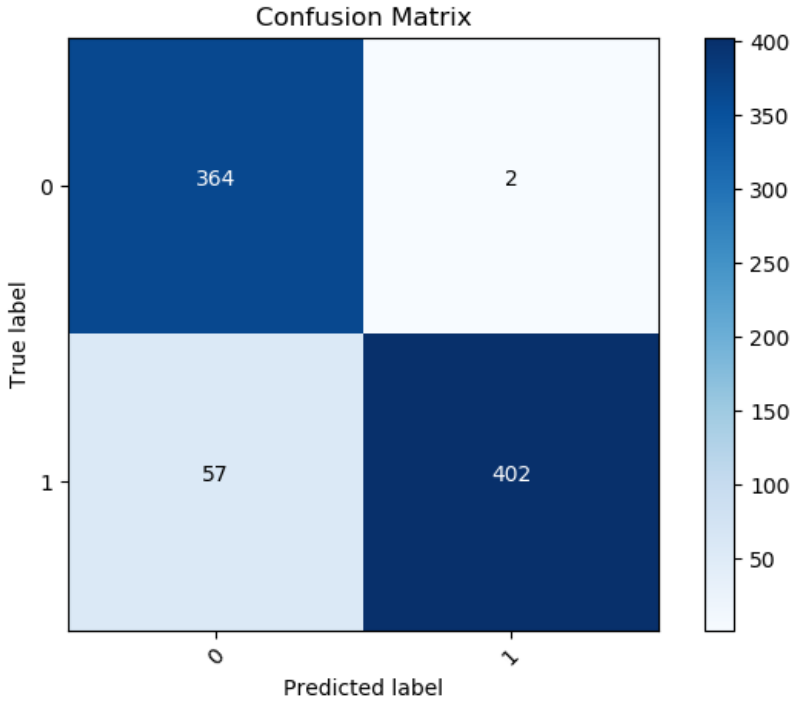

Figura 4.24: Matriz de confusión para clasificación binaria convolucional

En la tabla 4.9 se encuentra el resumen de los resultados de esta red. Se cuenta con el resultado tanto para la fase de entrenamiento como para la fase de prueba

Entrenamiento				Prueba	
Loss	Accuracy	Val_loss	Val_Accuracy	Loss	Accuracy
0.1676	0.9466	0.2029	0.9285	0.2086	0.9284

Tabla 4.9: Resultados clasificación binaria convolucional

Perceptrón Multicapa

Entre las figuras 4.25 y 4.30 se presentan los resultados para la red de clasificación binaria del tipo Perceptrón Multicapa (MLP).

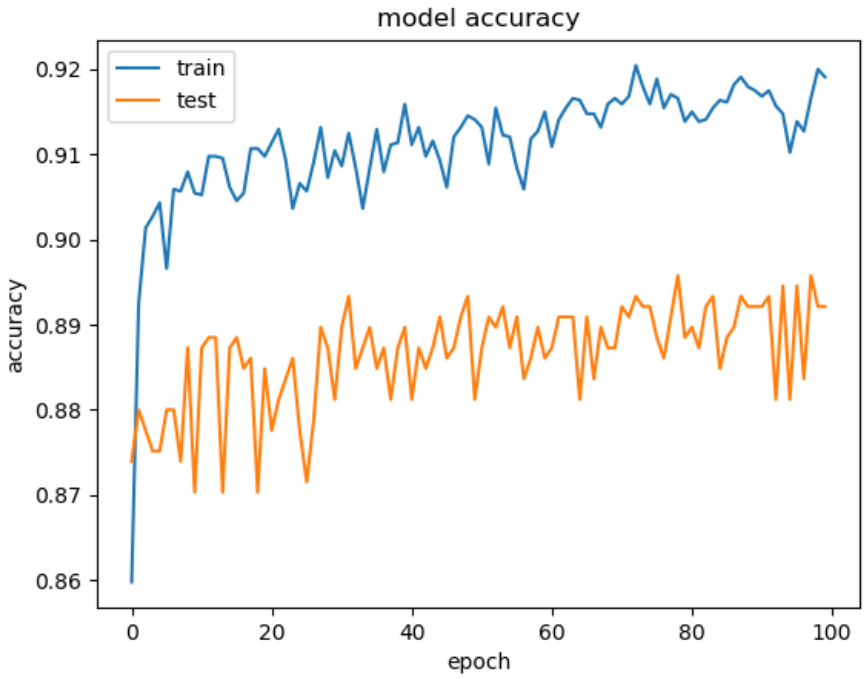

Figura 4.25: Accuracy clasificación binaria MLP

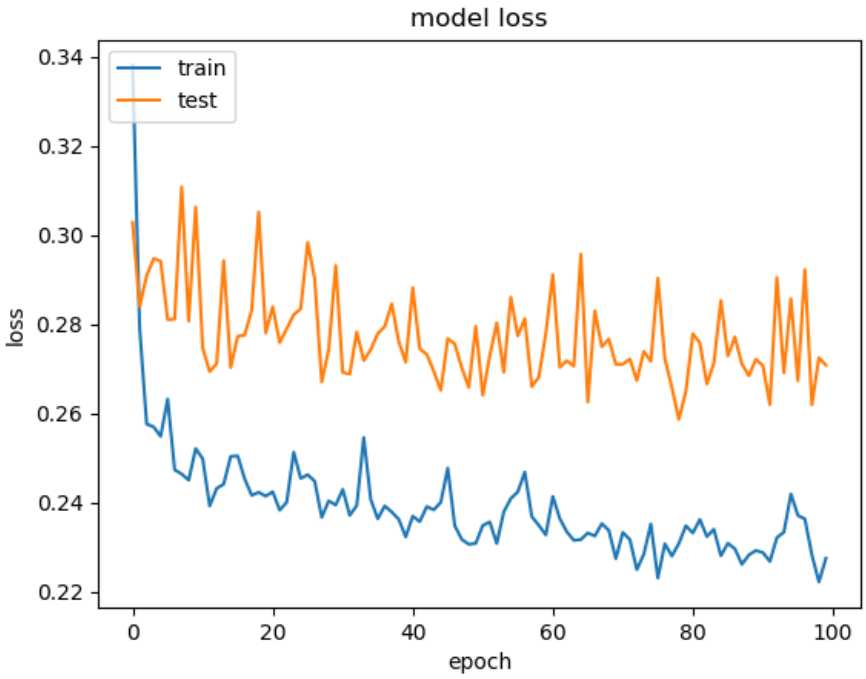

Figura 4.26: Loss clasificación binaria MLP

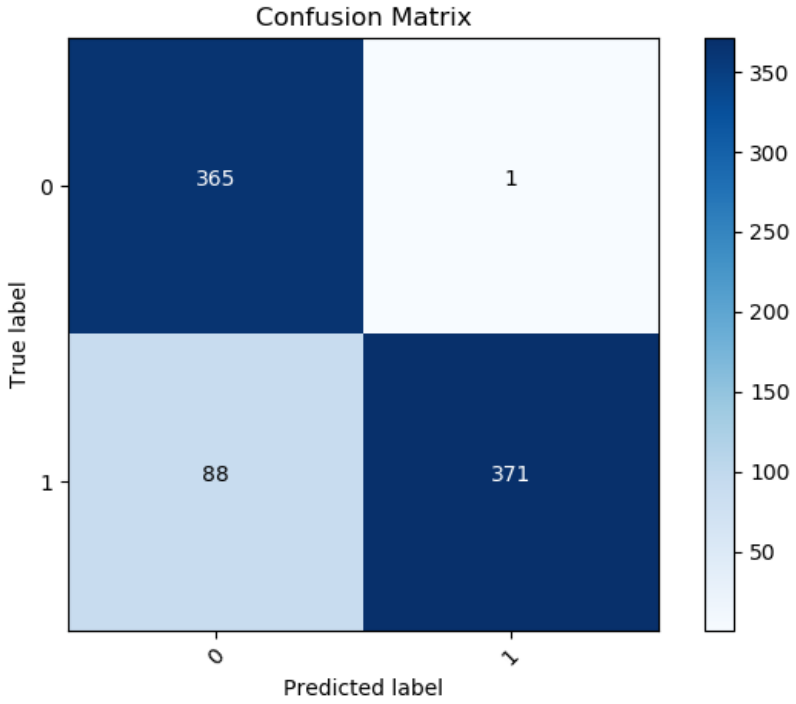

Figura 4.27: Matriz de confusión para clasificación binaria MLP

En la tabla 4.10 se encuentra el resumen de los resultados de esta red. Se cuenta con el resultado tanto para la fase de entrenamiento como para la fase de prueba

Entrenamiento				Prueba	
Loss	Accuracy	Val_loss	Val_Accuracy	Loss	Accuracy
0.2275	0.9191	0.2709	0.8921	0.2708	0.8921

Tabla 4.10: Resultados clasificación binaria MLP

4.3.2. Clasificación multiclase tipo 2

Se realiza un modelo de clasificación multiclase con el fin de determinar el tamaño de daño, es decir, solo se clasifica entre las clases 1 y 5.

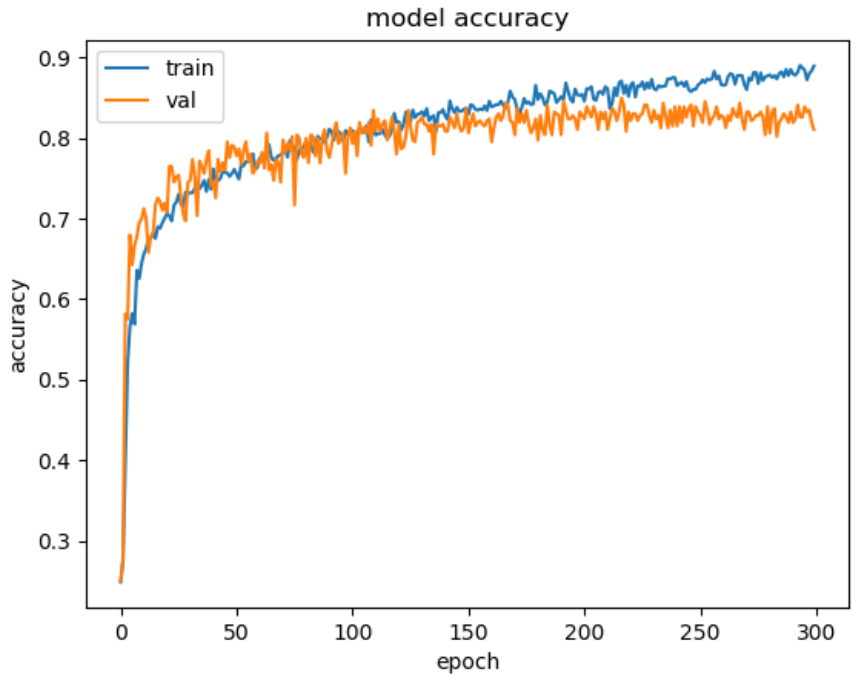

Figura 4.28: Accuracy para clasificación multiclase CNN tipo 2

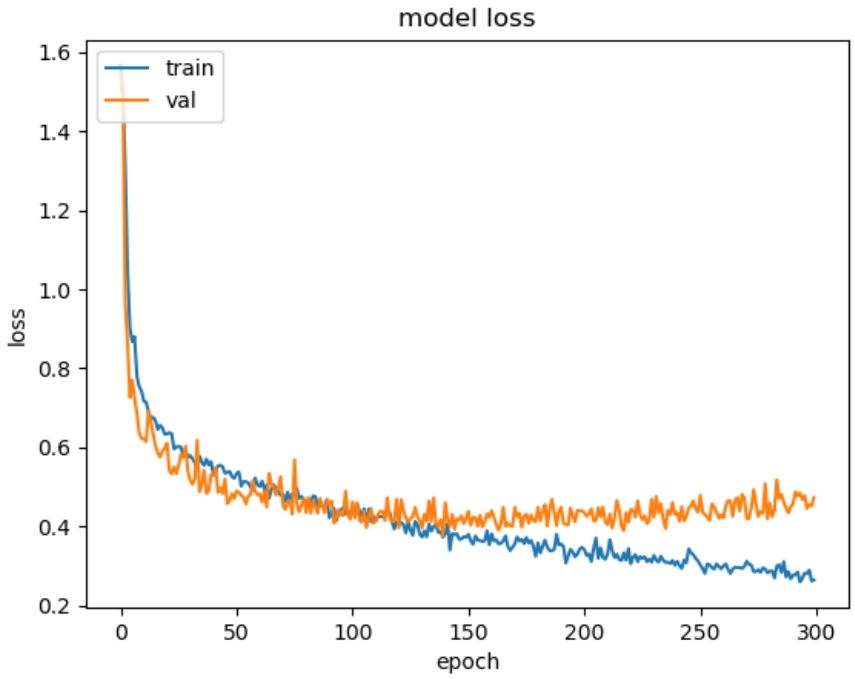

Figura 4.29: Loss para clasificación multiclase CNN tipo 2

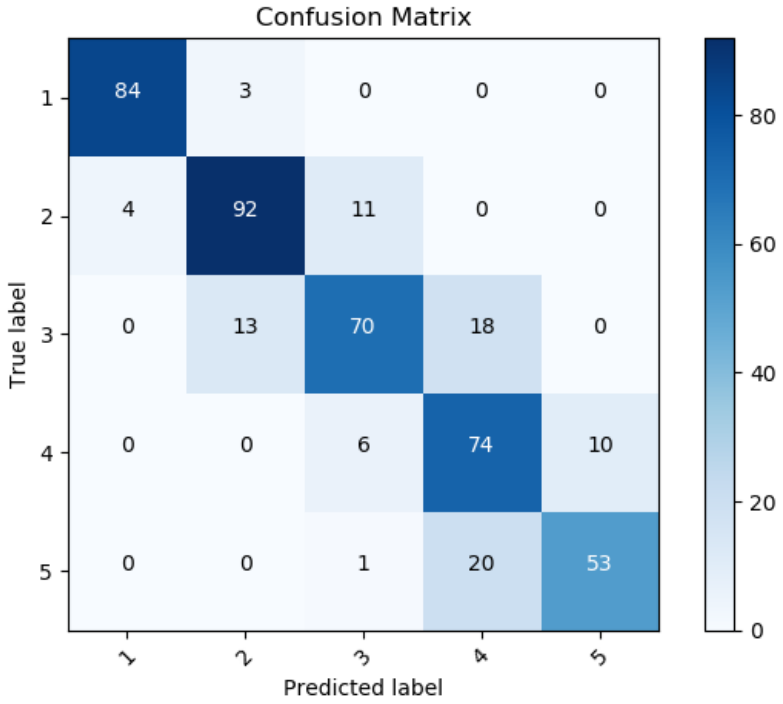

Figura 4.30: Matriz de confusión para clasificación multiclase CNN tipo 2

En la tabla 4.11 se encuentra el resumen de los resultados finales de esta red de clasificación del tamaño de daño, tanto para la fase de entrenamiento como para la fase de prueba.

Entrenamiento				Prueba	
Loss	Accuracy	Val_loss	Val_Accuracy	Loss	Accuracy
0.2783	0.8795	0.4546	0.8013	0.4407	0.8126

Tabla 4.11: Resultados para la clasificación del tamaño de daño

4.4. Modelos Sin Clase 1

Dado que la mayor fuente de error en los diferentes modelos estudiados provienen de la mala clasificación de la clase 1 y considerando que en la literatura ya se ha notado la imposibilidad de identificar la delaminación bajo un tamaño de 0.08 de daño normalizado [3] se propone realizar de nuevo las pruebas con los diferentes modelos pero eliminando los datos de clase 1.

Utilizando uno de los thresholds seleccionados anteriormente, se corre el modelo esta vez con la ausencia de la clase 1. Al hacer esto se obtienen mejores resultados como los mostrados en las figuras 4.34 y 4.35. La efectividad (Accuracy) cambió a un **0.9600 %** siendo esto un aumento cercano al 10 % en efectividad al compararlo con el modelo original con clase 1.

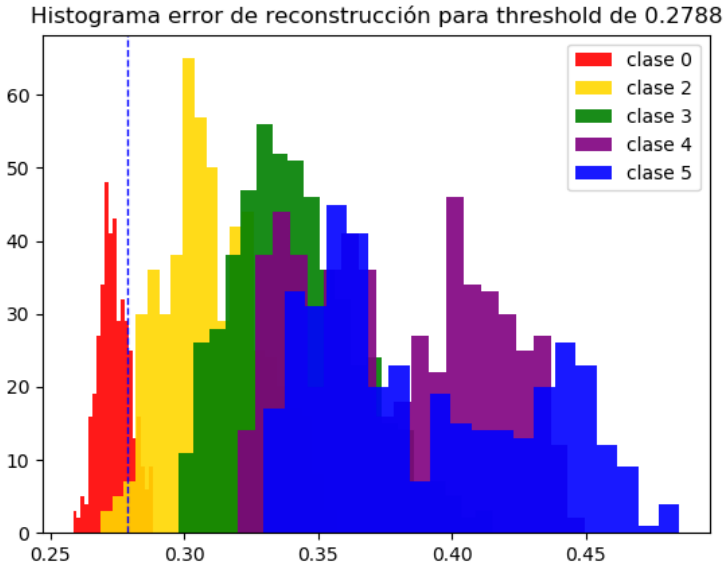

Figura 4.31: Histograma de error de reconstrucción sin clase 1 con threshold de 0.2788, accuracy 96 %

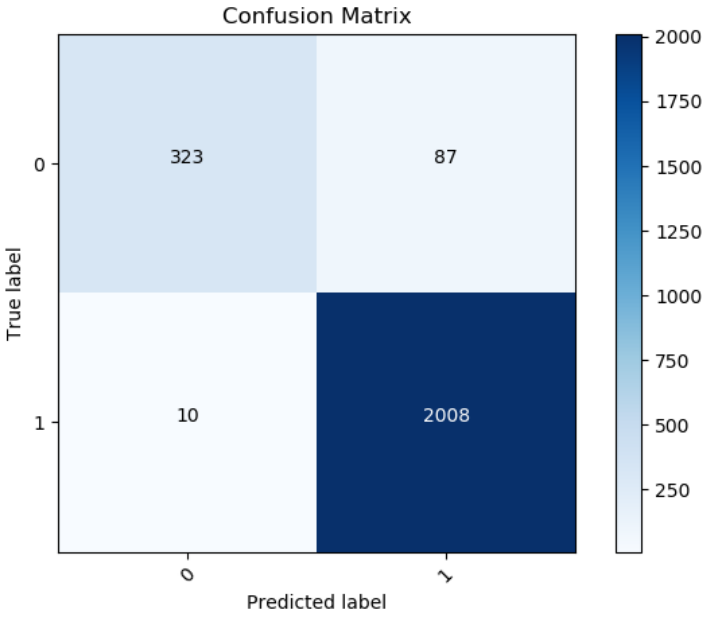

Figura 4.32: Matriz de confusión con threshold de 0.2788

Para la clasificación multiclase se utilizó la misma red convolucional de tipo 1 con los mismos hiperparametros. Los resultados se aprecian a continuación:

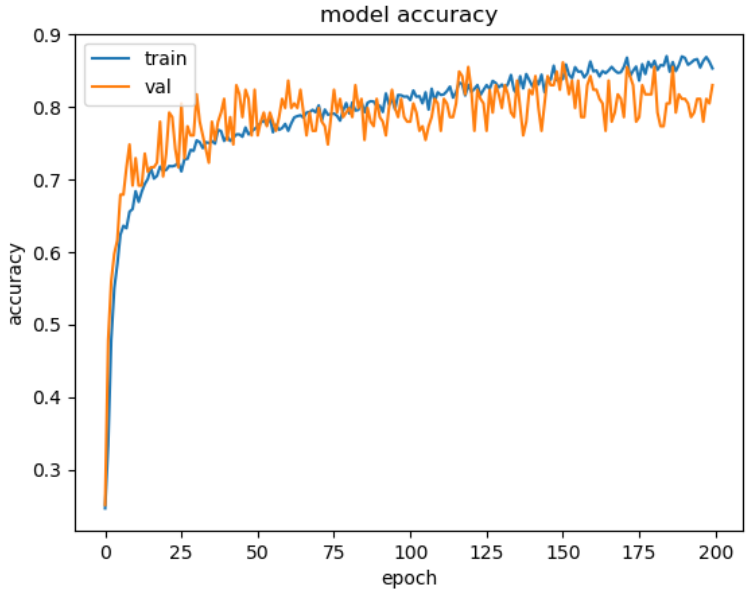

Figura 4.33: Entrenamiento red convolucional multiclase sin clase 1

Entrenamiento				Prueba	
Loss	Accuracy	Val_loss	Val_Accuracy	Loss	Accuracy
0.3282	0.8534	0.3767	0.8302	0.3963	0.8356

Tabla 4.12: Resultados clasificación multiclase CNN tipo 1 sin clase 1

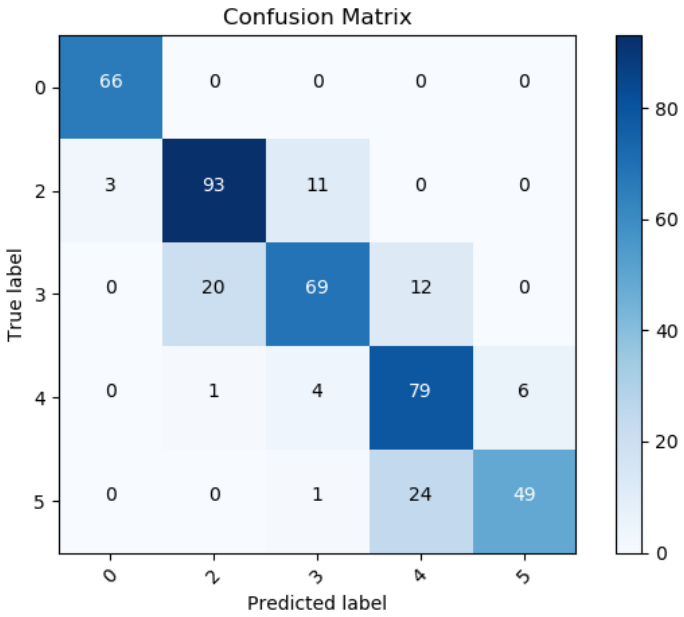

Figura 4.34: Matriz de confusión multiclase sin clase 1

4.5. Clasificación de placas reales

Una vez entrenados los diferentes modelos se procede a comprobar su comportamiento con las placas reales con daño por delaminación impuesto a la hora de su manufactura. Son 5 imágenes de placas de las cuales una no tiene daño, una tiene daño clase 2, dos tienen daño clase 3 y una tiene daño clase 4.

En las siguientes figuras se muestran los resultados obtenidos para el algoritmo del tipo One-Class Classification:

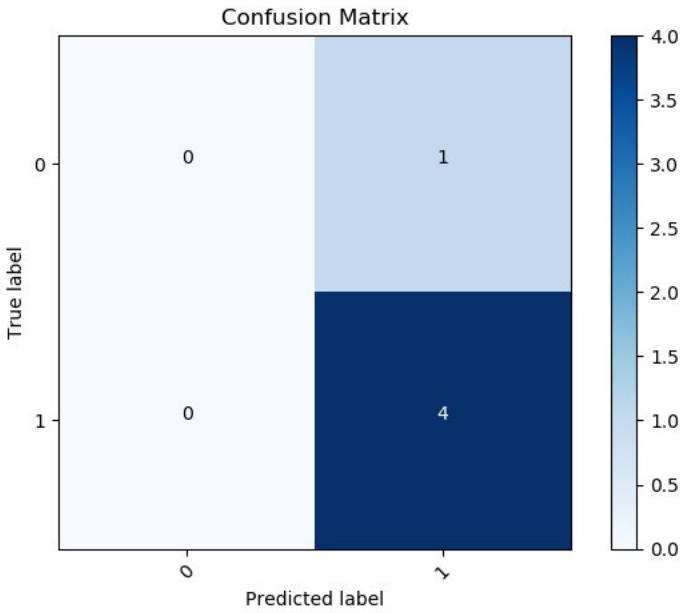

Figura 4.35: Matriz de confusión para placas reales utilizando el método de One-Class

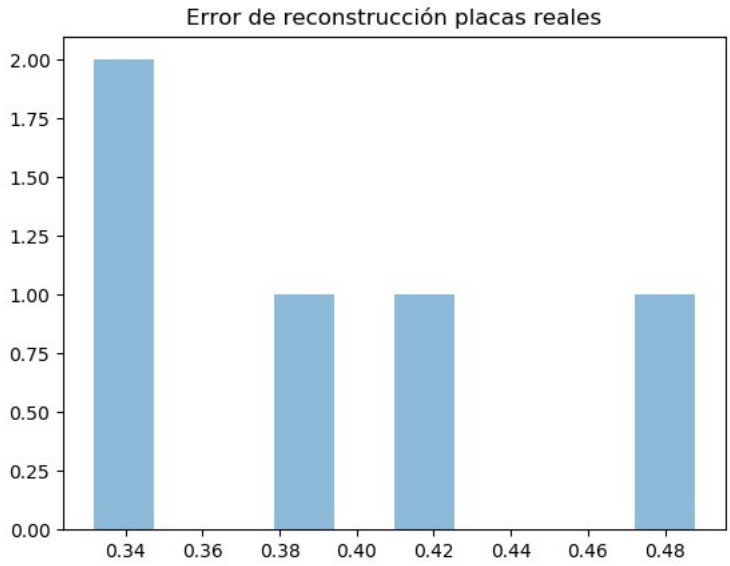

Figura 4.36: Error de reconstrucción para placas reales utilizando el método de One-Class

En la siguiente tabla se muestra el resultado de la clasificación del tipo 1 utilizando la red convolucional mostrada en la sección 4.2.

Por último se muestra la matriz de confusión obtenida por el método de clasificación binaria, tanto CNN como MLP tienen el mismo resultado:

Imágen	Real	Predicción
Caso 0	0	2
Caso 1	2	3
Caso 2	3	3
Caso 3	2	4
Caso 4	4	4

Tabla 4.13: Resultados para la clasificación del tamaño de daño real utilizando CNN del tipo 1

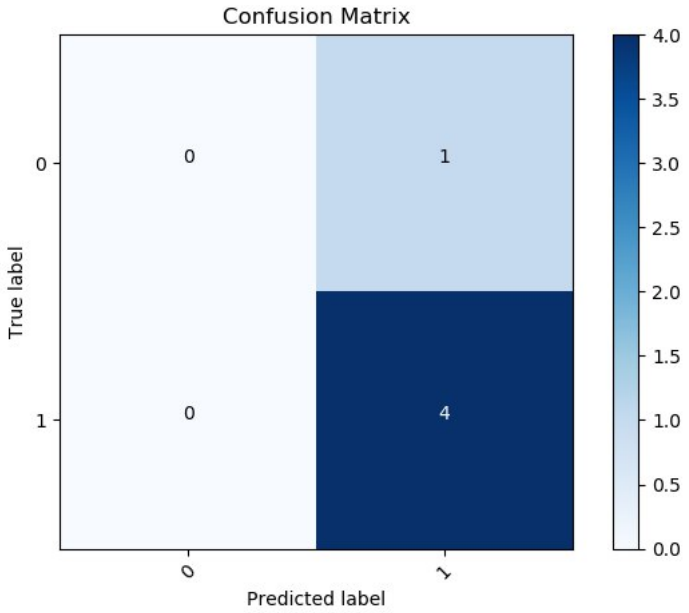

Figura 4.37: Matriz de confusión para placas reales utilizando el método de clasificación binaria

4.6. Análisis de Resultados

En este trabajo de título se estudiaron diferentes modelos para la clasificación de imágenes con y sin daño por delaminación de forma automática, utilizando Machine Learning. El objetivo primario era poder clasificar de forma binaria la existencia o no de daño, utilizando como algoritmo clasificador un Variational Autoencoder entrenado solo para instancias sin daño, y luego usando un threshold como valor límite del error de reconstrucción, clasificar instancias con daño si su error supera el threshold o como instancia sin daño si su error de reconstrucción es menor a este threshold.

En la sección de resultados se proponen tres elecciones diferentes de thresholds y además se muestran otros más junto a su respectiva efectividad y matriz de confusión. La efectividad del primer threshold seleccionado se ve en la matriz de confusión de la figura 4.9. Este threshold corresponde al máximo error de reconstrucción obtenido dentro de la clase 0 (sin daño), con la suposición (y esperanza) de que todas las clases con daño debiesen tener un error de

reconstrucción mayor a este. Con este supuesto, se obtiene una precisión de clasificación de un 84.41 %. Debido a que las instancias normales (sin daño) son en efecto las mas comunes, la elección de este threshold es tentador dado que muchas veces no se cuenta con suficientes datos con daño para hacer la comparación. La obtención de datos y la clasificación de los mismos muchas veces es el problema más grande a la hora de realizar un modelo de Machine Learning.

Dado que en este caso se cuentan con una gran cantidad de datos completamente clasificados, se puede estudiar los histogramas de error de reconstrucción para determinar un mejor threshold que permita una clasificación óptima. En la tabla 4.2 se muestran diferentes matrices de confusión y efectividad de la clasificación para diferentes threshold. La elección más simple dado que la efectividad de la clasificación es la métrica que intentamos maximizar, es el cuarto threshold propuesto que tiene una efectividad de un 86.89 %. Lamentablemente, esta elección no está exenta de problemas, ya que, si bien maximiza la efectividad total de clasificación, posee un numero alto de falsos positivos. Desde un punto de vista ingenieril, el falso positivo es el caso más complejo que podemos encontrar, ya que estamos afirmando que una placa es segura, cuando en realidad tiene una falla. Se debe buscar un balance entre tener la mayor efectividad posible y minimizar los falsos positivos. Este balance óptimo en para este problema no existe, debido a que no podemos tener una cuantificación (económica, social, seguridad, etc.) del trade-off que existe al variar el threshold, por lo que la elección queda a criterio de quien está resolviendo el problema. Un threshold muy bajo implicará un aumento en la confiabilidad y la seguridad de un sistema que utiliza este tipo de materiales, sin embargo, también tendrá un alto costo económico (mantenciones y cambios de placas dañadas). Por otro lado, un threshold alto es barato en términos de mantenciones y reparación, ya que estamos implicando que la mayoría de los materiales se encuentra en buen estado, sin embargo, es peligroso y puede llevar a fallas catastróficas según el tipo de aplicación. Considerando que la diferencia de efectividad entre los dos threshold marcados en negrita en la tabla 4.2 es de un 0.11 %, la elección de 0.278 como threshold parece mas atractiva ya que disminuye en 13 los casos de falsos positivos.

La mayor fuente de error en la clasificación de este tipo pertenece a la clase 1. Debido a que esta clase cuenta con datos de daño normalizado entre 0 y 0.05, para la red casi la totalidad de las imágenes son clasificadas como clase 0. Si se observa la figura 3.2 y la figura 3.3 la diferencia entre las imágenes de clase 0 y clase 1 es prácticamente imperceptible para la mayoría de los casos. La clasificación de esta clase es un problema recurrente en este trabajo, para todos los modelos utilizados, sin embargo, ya había sido probado esto en [3] .

La clasificación multiclase del tipo 1 tiene como motivación no solo investigar la presencia de daño, sino que además cuantificar su tamaño dentro de cinco diferentes rangos. El mejor resultado obtenido dentro de todos los modelos propuestos corresponde a la red de clasificación del tipo convolucional, la matriz de confusión se observa en la figura 4.14. Si observamos los resultados de los diferentes modelos de clasificación de tipo 1 presentados, la mayor diferencia en la efectividad entre ellos corresponde de nuevo a la clasificación del tipo 1, donde los modelos convolucionales logran clasificar de mejor manera esta clase que los modelos de perceptrón multicapa, afirmando la superioridad de los modelos convolucionales para el trabajo de imágenes en Machine Learning, esto se hace más importante al notar la cantidad de parámetros a optimizar para ambos modelos, en donde la red convolucional

del tipo 1 posee 476.326 parámetros y la red MLP posee 1.987.334 parámetros a optimizar. Por esto, las redes convolucionales no solo son mejores en términos de resultados, sino que también son menos costosas computacionalmente hablando.

Un problema recurrente en la clasificación multiclase es la clasificación con las clases vecinas. Es decir, si por ejemplo la clase correcta de una imagen es la clase 3, el modelo clasifica en un 75 % como clase 3, pero el resto de las de las predicciones se dividen en las clases vecinas, es decir, predice que puede ser clase 2 o clase 4. Este problema sumado al problema de la clasificación de la clase 1 mencionado anteriormente son los que están generando un rendimiento subóptimo de los modelos de clasificación. Se cuentan con dos posibles causas que explican este problema en cuestión: la continuidad de las clases y la ubicación del daño en las placas. El problema de la continuidad podemos ilustrarlo en la figura 4.38. Notemos que, según la definición previa de las clases, la clase 1 contiene daño por delaminación normalizado entre 0-0.05-0.1 (incluido 0.05) y la clase 2 entre 0.05-0.1, sin embargo, en la figura se observan ejemplos cercanos al daño por delaminación de 0.05:

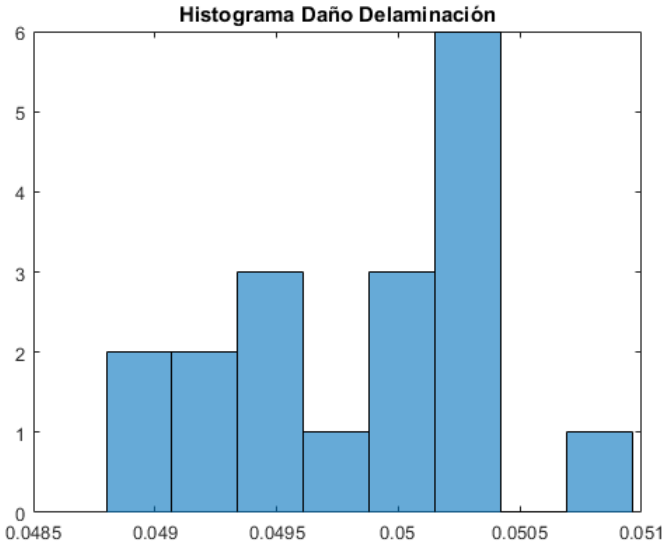

Figura 4.38: Histograma de imágenes según su daño por delaminación

Podemos ver el daño por delaminación como una variable casi continua y sin embargo, estamos haciendo una separación estricta en un valor especifico, en este caso ese valor es 0.05. Se vuelve evidente que los modelos tengan problemas al clasificar imágenes con daño cercano al corte entre la separación de clases, debido a que esta diferencia entre una imagen con daño cercano al corte por debajo y otra cercano al corte pero por sobre, es casi nula. En este ejemplo siete imágenes que están siendo clasificadas como clase 2 que están por sobre el corte por menos de 0.003. Es importante notar que este problema no es único de la separación entre clase 1 y clase 2 sino que es un problema recurrente en la separación de todas las otras clases.

El otro problema mencionado anteriormente es el de la ubicación del daño. En las imágenes utilizadas para el entrenamiento, el daño por delaminación se observa como un circulo blanco,

sin embargo, la posición en la placa es aleatorio. Esto trae como complicación algo que se ejemplifica en la figura 4.39, donde se hace una comparación entre una figura con daño de la clase 2 y otra de la clase 3. Cuando el daño está en los bordes de la imagen, el tamaño del daño no se logra apreciar correctamente. En este caso, la diferencia entre la imagen de la clase 2 y la clase 3 es nuevamente imperceptible a simple vista y para los diferentes modelos, ambas pertenecen a la misma clase.

Figura 4.39: Comparación entre dos imágenes, clase 2 a la izquierda y clase 3 a la derecha

Para el caso binario se obtuvo una efectividad de 92.84 % para el clasificador convolucional. Posterior a esto, se realiza una clasificación multiclase, pero solo entre las clases con daño, es decir, entre la clase 1 y 5. Se observa en la matriz de confusión de la figura 4.30 una clasificación casi perfecta para la clase 1, y un rendimiento similar para el resto de las clases comparándolo con el algoritmo de clasificación multiclase tipo 1. Se obtuvo una efectividad de un 81.26 % para este clasificador multiclase, bastante superior al 72.13 % de la clasificación multiclase tipo 1. Sin embargo, esto no es así, ya que al separar el problema en dos clasificaciones diferentes (binaria primero y multiclase posterior) el problema de la clasificación multiclase tipo 2 se vuelve un problema condicionado por la clasificación binaria en primera instancia. Por lo que el verdadero resultado de esta combinación, para clasificar todas las clases sería la probabilidad condicional de la efectividad de la clasificación multiclase tipo 2, dado la clasificación binaria. La efectividad total de este modelo entonces será $81{,}26\,\% \cdot 92{,}84\,\% = 75{,}28\,\%$, que sigue siendo superior a la clasificación multiclase del tipo 1 por un 3 %.

Si consideramos que según investigaciones previas [3] la identificación de delaminación no es posible cuando el daño normalizado es menor a 0.08, podemos utilizar los modelos entrenados para una identificación de daño únicamente factible, es decir eliminando la clase 1 en su totalidad. Como se observa en la sección 4.4 los resultados mejoran considerablemente al eliminar la clase 1, ya que esta siempre fue una fuente de error constante como se mencionó anteriormente. Tanto para el caso del Variational Autoencoder (One-Class) y para la clasificación multiclase la efectividad de los modelos mejoro alrededor de un 10 %.

Finalmente, se pueden comparar los modelos de clasificación binaria con la clasificación utilizando el Variational Autoencoder y el threshold para la separación entre las clases. Solo observando los resultados, es fácil llegar a la conclusión de que la clasificación binaria supervisada es claramente superior y por lo tanto, debiese ser el camino a seguir para resolver este tipo de problemas. Este análisis es correcto si y solo si se cuentan con la cantidad de datos clasificados como los que se tienen para este trabajo. La mayoría de las veces, cuando se tiene un problema como este, lo más común es contar con pocos datos y la mayoría de estos

son de los casos normales. El cuello de botella en muchos problemas de Machine Learning es la adquisición y la correcta clasificación de los datos de entrenamiento, muchas veces es un proceso costoso en términos de tiempo y monetario. Por lo mismo, el modelo utilizando Variational Autoencoder nos permite realizar esta tarea de identificación de daño de manera de solo utilizar datos normales y con buenos resultados también. Es por esto que no es necesariamente que un modelo sea mejor que otro, sino que dependen del problema que se quiera resolver y de la cantidad y del tipo de datos que se utilicen.

Conclusión

A lo largo del presente trabajo de título se realiza el diseño e implementación de diferentes modelos para la detección automática de daño en placas de material compuesto tipo sándwich con daño del tipo delaminación. Sobre esto, se puede concluir lo siguiente

- El modelo propuesto para la identificación de daño utilizando Variational Autoencoder resuelve el problema de la clasificación de forma satisfactoria, sobr etodo cuando la cantidad de datos con daño es escasa.
- Es importante cuantificar el daño (social y económico) que genera un falso positivo y un falso negativo, para poder elegir un threshold correcto de tal forma de tener la mejor efectividad posible para el modelo.
- Si se cuenta con suficientes datos de entrenamiento y prueba de diferentes tamaños de daño, realizar un modelo de Machine Learning de clasificación binaria de forma supervisada trae mejores resultados para la identificación de daño que el modelo no supervisado utilizando Variational Autoencoder.
- Otra metodología propuesta si se cuenta con datos de diferentes tamaños de daños, es clasificar las imágenes según el tamaño de la delaminación. Los modelos de aprendizaje supervisado nuevamente son superiores para la clasificación multiclase.
- Las redes neuronales convolucionales son superiores a las MLP para la clasificación de imágenes, además de ser computacionalmente menos costosas ya que la cantidad de parámetros a optimizar es considerablemente menor y con mejores resultados.
- La clasificación multiclase de tipo 2 es el mejor modelo propuesto para clasificación multiclase.
- La clasificación de la clase 1 como imagen con daño sigue siendo un problema difícil de resolver y con poca efectividad en los diferentes modelos propuestos en este trabajo.
- Los métodos computacionales de este trabajo de título entrenados con imágenes generadas computacionalmente no son del todo capaces de cuantificar el daño de delaminación para las placas reales.

Bibliografía

[1] Seguel F. *Identificacion de daño en placas tipo sandwich usando un sistema de correlación de imágenes digital y la curvatura de los modos de vibración*

[2] C. Doersch. *Tutorial on variational autoencoders*

[3] Meruane V, Fernandez I, Petrone G, Droguett EL. *Gapped Gaussian smoothing technique for debonding assessment with automatic thresholding. Accepted for publication in Structural Control and Health Monitoring.*

[4] Jinwon An, Sungzoon Cho. *Variational Autoencoder based Anomaly Detection using Reconstruction Probabilty.*

[5] Mark N Helfrick, Christopher Niezrecki, Peter Avitabile, and Timothy Schmidt. *3d digital image correlation methods for full-field vibration measurement.*

[6] Varano C.*Disentangling Variational Autoencoders for Image Clasification*

[7] Ben Letham, Cynthia Rudin. *Probabilistic Modeling and Bayesian analisis*

[8] Zhu Shengqing, Chai Gin Boay. *Damage and failure mode maps of composite sandwich panel subjected to quasi-static indentation and low velocity impact.*

[9] Suwon Suh, Daniel H.Chae, Hyon-Goo Kang, Seungjin Choi. *Echo-State Conditional Variational Autoencoder for Anomaly Detection.*

[10] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, Lawrence Carin. *Variational Autoencoder for Deep Learning of images, Labels and Captions.*

[11] D. P. Kingma and M. Welling. *Auto-encoding variational Baye*

[12] Vasilev A, Golkov V, Meissner M, Lipp I, Sgarlata E, Tomassini V, Jones D, Cremes D. *q-Space Novelty Detection with Variational Autoencoders*

[13] Jordan J. *Variational autoencoders https://www.jeremyjordan.me/variational-autoencoders/*

[14] A. Karpathy, F.-F. Li, and J. Johnson. *CS231n: Convolutional Neural Networks for Visual Recognition. [En línea]. Available : http://cs231n.github.io/convolutional-networks/*

[15] V. d. F. A. O. V.Meruane, *A Maximum Entropy Approach to Assess Debonding in Honeycomb Aluminum Plates*

[16] Campbell, F. C. *Adhesive Bonding and Integrally Cocured Structure: A Way to Reduce Assembly Costs through Parts Integration. Manufacturing Processes for Advanced Composites*

[17] R Partridge. *Convolutional Neural Networks (CNN). https://github.com/Achronus/Machine-Learning-101/wiki/Convolutional-Neural-Networks-(CNN)*