UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Vectores de características de alto nivel para la descripción y clasificación de objetos

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Víctor Sebastian Martínez Pozos

DIRECTOR DE TESIS

Dr. Victor Manuel Lomas Barrié

Ciudad Universitaria, Cd. Mx., 2022

A mi familia y a mis maestros

Gracias por cada una de sus enseñanzas, las cuales me han ayudado a crecer, entender mejor el mundo y ser una mejor persona.

Indice

Indice de tablas y figuras

Glosario

C

Curva ROC: Las curvas ROC (característica operativa del receptor) constituyen una herramienta importante para evaluar el rendimiento de un modelo de machine learning. Por lo general, se utilizan en problemas de clasificación binaria, concretamente, problemas con dos clases de salidas distintas.[1]

Callbacks de entrenamiento: Un callback es un objeto que puede realizar acciones en varias etapas de entrenamiento (por ejemplo, al comienzo o al final de una época, antes o después de un solo lote, etc.).[2]

D

Discriminador: Un modelo discriminador es la red neuronal que evalúa los datos del objetivo sintético y los datos del objetivo real para determinar cuál es el real. El discriminador, en este caso, es un modelo CNN; toma una matriz tridimensional como entrada.

Distribución gaussiana: La distribución normal, a veces llamada gausiana, es una familia de curvas de dos parámetros. La justificación habitual del uso de la distribución normal para la modelización es el teorema del límite central, que establece, a grandes rasgos, que la suma de muestras independientes de cualquier distribución con media y varianza finitas converge a la distribución normal a medida que el tamaño de la muestra se eleva al infinito.

E

Etiquetas: En el aprendizaje automático, el etiquetado de datos es el proceso de identificar datos sin procesar (imágenes, archivos de texto, videos, etc.) y agregar una o más etiquetas significativas e informativas para proporcionar contexto para que un modelo de aprendizaje automático pueda aprender de ellos.[3]

Epoca: Una época se refiere a un ciclo a través del conjunto de datos de entrenamiento completo.

F

Función de perdida: Una función de pérdida mide la discrepancia entre la predicción de un algoritmo de aprendizaje automático y la salida supervisada y representa el costo de equivocarse.[4]

H

Hiperparámetros: Los hiperparámetros son parámetros ajustables que permiten controlar el proceso de entrenamiento de un modelo. Por ejemplo, con redes neuronales, puede decidir el número de capas ocultas y el número de nodos de cada capa. El rendimiento de un modelo depende en gran medida de los hiperparámetros.

---

[1] Mathworks - Curva ROC

[2] Keras - Callbacks API

[3] AWS - SageMaker

[4] MicrosoftML

M

Modelo: Un modelo de machine learning es la salida de información que se genera cuando entrena su algoritmo de machine learning con datos.[5]

Matriz de confusión: Una matriz de confusión, también conocida como matriz de error, es una tabla resumida que se utiliza para evaluar el rendimiento de un modelo de clasificación. El número de predicciones correctas e incorrectas se resumen con los valores de conteo y se desglosan por cada clase.[6]

N

Normalización: La normalización es una técnica que se aplica a menudo como parte de la preparación de datos para el aprendizaje automático. El objetivo de la normalización es cambiar los valores de las columnas numéricas en el conjunto de datos para usar una escala común, sin distorsionar las diferencias en los rangos de valores ni perder información.[7]

O

Optimizador: Los optimizadores son algoritmos o métodos que se utilizan para cambiar los atributos de la red neuronal, como los **pesos** y **la tasa de aprendizaje,** para reducir las pérdidas. Los optimizadores se utilizan para resolver problemas de optimización minimizando la función.[8]

P

Patrones: Una forma particular en la que algo se hace, se organiza o sucede.[9]

R

Red deconvolucional: Las redes desconvolucionales son redes neuronales convolucionales (CNN) que funcionan en un proceso inverso. Las redes desconvolucionales, también conocidas como redes neuronales desconvolucionales, son de naturaleza muy similar a las CNN que se ejecutan a la inversa, pero son una aplicación distinta de la inteligencia artificial (IA).[10]

Red objetivo: Una red objetivo es una copia de la función acción-valor (o función Q) que se mantiene constante para servir como un objetivo estable para el aprendizaje durante un número fijo de pasos de tiempo[27].

S

Sobre ajuste de un modelo: El sobreajuste es un concepto en la ciencia de datos, que ocurre cuando un modelo estadístico se ajusta exactamente a sus datos de entrenamiento. Cuando esto sucede, el algoritmo lamentablemente no puede funcionar con precisión contra datos no vistos, lo que anula su propósito.[5]

---

[5] IBM - analytics

[6] Datasource.ai - data-science-articles

[7] Microsoft - Azure/ML

[8] Kdnuggets - optimization-algorithms-neural-networks

[9] Cambridge - diccionario

[10] techtarget - searchenterpriseai

1. Introducción

Al día de hoy es muy fácil el pensar en una infinidad de aplicaciones que usan la visión por computadora de manera cotidiana para su funcionamiento, como por ejemplo cuando usamos una cámara digital y ésta realiza la detección de los rostros presentes en la imagen o cuando un celular es capaz de reconocer la identidad de una persona evaluando de manera automática sus rasgos faciales, sin embargo, el proceso que se lleva a cabo dentro de estos algoritmos no siempre es claro o explicable y depende en gran medida de diversos factores como lo son el tipo de tarea en cuestión, el o los algoritmos empleados y las limitaciones propias del dispositivo físico.

Dentro de las metas de la visión por computadora la tarea de clasificar a un objeto mediante las imágenes de este siempre ha sido considerada como algo esencial o básica, ya que los desarrollos o mejoras sobre los métodos de representación de objetos a este nivel son mucho más simples de estudiar y por tanto de entender, de modo que es posible analizar y extraer a los principios que hacen posible esta mejora en la representación para que pueda ser extrapolada a tareas de mayor complejidad, lo cual ha permitido a las redes neuronales pasar de clasificar dígitos a detectar cualquier clase de objeto, Dicho lo anterior podemos argumentar que la clasificación de imágenes y otras tareas de visión por computadora tienen su base y su límite en relación directa a la calidad de representación que cada método ofrece, siendo la representación ideal aquella que sea capaz de abstraer correctamente las características esenciales que definen a un objeto invariablemente de su estado, como lo puede ser la iluminación, su pose, etc.

No obstante el rendimiento y la precisión en los modelos de clasificación ha tenido un avance notable en la última década la forma en que estas codifican la información visual contenida en las imágenes carece de una abstracción general ya que las redes ignoran la jerarquía natural y relación existente en los elementos de una imagen y codifican únicamente patrones comunes entre objetos que solo tienen sentido bajo el mismo modelo, siendo actualmente un tema de investigación activa el desarrollo de nuevos modelos que sean capaces no solo de alcanzar un rendimiento excepcional sino de explicar o relacionar la información visual contenida en una imagen como pueden ser por ejemplo las redes de cápsulas o SCAE. Este trabajo busca en el mismo sentido el desarrollo de un método que mejore la codificación existente dentro de las redes neuronales de modo que se exprese de manera única e invariante lo que un objeto en concreto es haciendo uso de distribuciones de probabilidad tal que a partir de dicha representación y la selección de una muestra dada pueda ser posible reconstruir cualquier objeto que pertenezca a la misma categoría.

1.1 Objetivos

Este trabajo tiene por objetivo el desarrollo e introducción de un nuevo método basado en redes neuronales artificiales denominado "*Generalized Variational Auto Encoder*"(G-VAE), que se inspira en el funcionamiento de los "*Variational Auto Encoders*"(VAEs) y las "*Generative Adversarial Networks*"(GANs) y busca ser capaz de aprender a representar a los objetos de forma única en forma de parámetros de funciones de distribución. De modo que a través de ellos y por tanto de sus distribuciones sea posible describir las características esenciales de cada objeto las cuales puedan ser humanamente interpretables así como empleadas para tareas de destilación de conocimiento dentro del aprendizaje profundo. Finalmente se plantea la evaluación de dichas representaciones en tareas de clasificación en las que se compararán en términos de rendimiento y tiempo con un método de detección de características conocido como "función de contorno de objetos"(BOF) que es capaz de transformar un objeto en un único vector de características mediante el cual es posible llevar a cabo tareas de clasificación o regresión sobre este.

1.2 Metas

Con la finalidad de alcanzar los objetivos anteriormente propuestos se plantea el desarrollo de las siguientes metas:

- Creación del set de datos de entrenamiento, el cual contempla la proyección de objetos tridimensionales en el plano y la asignación de una clase.
- Carga de los datos de entrenamiento y preprocesamiento de estos.
- Programación de funciones auxiliares para el entrenamiento del modelo.
- Definición y entrenamiento de un modelo base.
- Mejora y perfeccionamiento del modelo base.
- Análisis de las representaciones generadas por el modelo final.
- Definición y entrenamiento de diferentes modelos compactos de clasificación.
- Análisis y comparación de los modelos reclasificación.

º

1.3 Estructura

Este trabajo se encuentra estructurado en tres secciones, en donde la primera sección describe los principios, modelos y principales trabajos que sirven como inspiración para el desarrollo del método propuesto. En la segunda sección se describe la metodología a seguir y el proceso de desarrollo del modelo comenzando desde la creación del conjunto de datos de entrenamiento y finalizando con los resultados obtenidos. Finalmente en la tercera sección se analizan y discuten los resultados y el efecto que cada modificación en el modelo original tiene en la representación final de los objetos.

2. Marco teórico

En esta sección se estudiarán los conceptos más relevantes para este trabajo dentro de las áreas de la visión por computadora, el aprendizaje máquina y el aprendizaje profundo. Comenzando con la definición de lo que es la visión por computadora, la importancia del procesamiento digital de imágenes, y los elementos que componen a un sistema de clasificación o reconocimiento de imágenes. Posteriormente se estudiará qué es el aprendizaje máquina y las diferentes tareas que la componen como los son el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo. Finalmente se estudiará un tipo de aprendizaje máquina conocido como aprendizaje profundo así como su relevancia en la clasificación y reconocimiento de imágenes, los principios de funcionamiento de éste y la introducción de lo que son los modelos de variable latente, su relevancia y algunos de los modelos de aprendizaje profundo que funcionan bajo este paradigma.

2.1 Visión por computadora

La visión por computadora es un área de la inteligencia artificial que permite a las computadoras y a los sistemas inteligentes obtener información significativa a partir de imágenes, videos digitales y otro tipo de información visual y tomar acciones o recomendaciones basadas en el uso de dicha información[3]. También se puede definir a esta como un área de enfoque semejante al de la inteligencia artificial que surge con la finalidad de replicar en las máquinas las capacidades cognitivas humanas, pero a diferencia de esta última, la visión por computadora se enfoca únicamente en los procesos relacionados a la percepción buscando permitir a las computadoras ver, observar y entender la información visual presente en el mundo real.

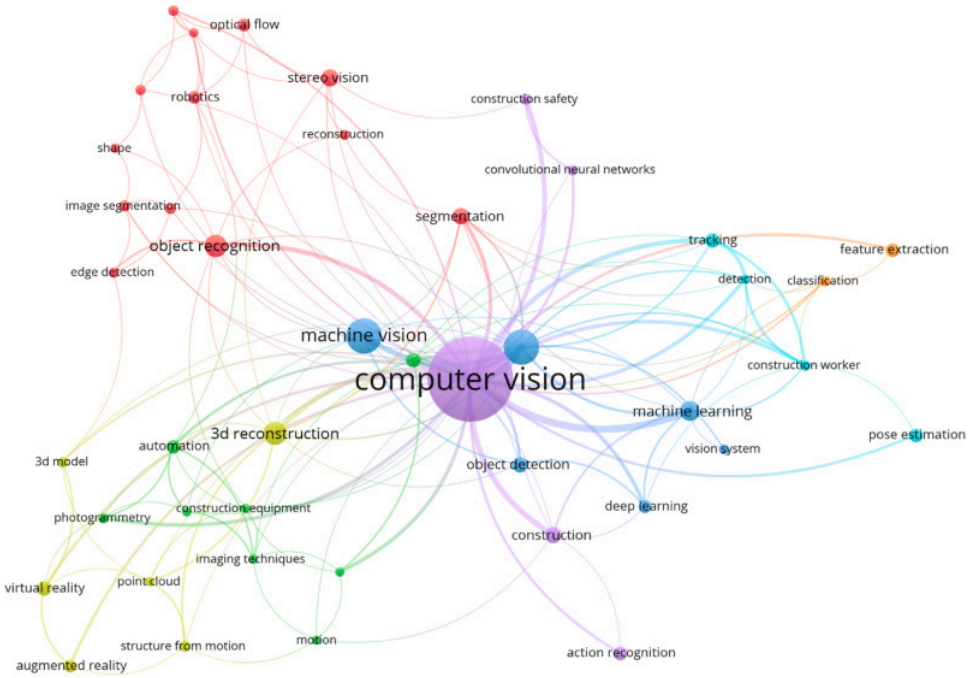


FIGURA 2.1 GRAFO DE CO-OCURRENCIAS DE LOS ELEMENTOS QUE COMPONEN A LA VISIÓN POR COMPUTADORA.
(Figura tomada de [25])

La visión por computadora surge además como la intersección de diversas disciplinas las cuales pueden apreciarse dentro de la figura 2.1 en donde se observa la relación existente entre las diferentes áreas, elementos y técnicas que componen al campo de la visión por computadora.

El estudio y desarrollo de la visión por computadora tiene como base las ideas provenientes de la neurociencia, la óptica y la psicología, las cuales explicaban, basadas en los últimos descubrimientos, el funcionamiento individual y grupal de los elementos que formaban parte del sistema de visión humano, es por ello que entre las primeras áreas de investigación se destacan la obtención de imágenes en un contexto físico, el procesamiento digital de ellas así como de las señales relacionadas con estas y el reconocimiento de patrones.

2.1.1 Procesamiento de imágenes

El procesamiento de imágenes es normalmente la primera etapa [3, p. 32] que se realiza posterior a la obtención de imágenes en un algoritmo de visión por computadora y busca el mejoramiento visual en la calidad de la imagen de entrada de modo que esta cumpla las necesidades para su posterior procesamiento.

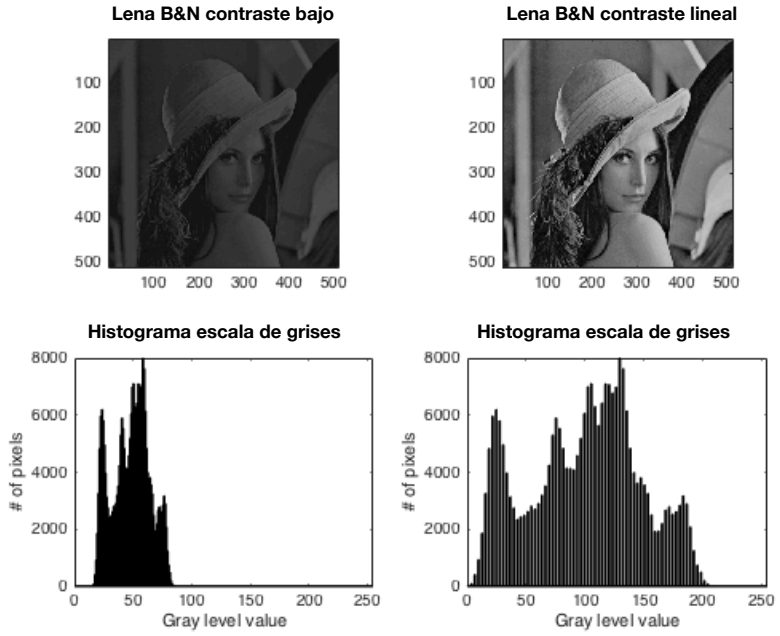


FIGURA 2.2 MEJORA DEL CONTRASTE DE UNA IMAGEN HACIENDO USO DEL PROCESAMIENTO DE IMÁGENES.
(Figura modificada de [5])

Entre las aplicaciones más comunes del procesamiento de imágenes se encuentra la disminución de ruido, la nivelación de brillo, el balanceo de color, la mejora de los bordes y remover problemas de enfoque, entre otros. Los métodos que componen al procesamiento de imágenes se pueden clasificar en función del tipo de transformación que estos realizan sobre los pixeles de la imagen de entrada como

operadores de punto mismos que se describirán a continuación y que actúan de manera independiente sobre cada píxel de la imagen de entrada o como operadores de vecindad que toman en cuenta el valor de los pixeles vecinos en cada transformación de los pixeles de la imagen.

De manera general se puede modelar a un operador de procesamiento de imágenes como una función “h” que reciben una o más imágenes de entrada y producen una imagen de salida dentro del dominio continuo. Y puede ser expresado matemáticamente mediante las siguientes ecuaciones[3, p. 43]:

$$g(x) = h(f(x)) \quad \text{..... (1.1)}$$
$$g(x) = h(f_0(x), \ldots, f_n(x)) \quad \text{..... (1.2)}$$

En donde x son los indices(renglón, columna) de cada píxel en una imagen dada cuyos valores se encuentra en el dominio de R^n, siendo n=2 la dimensión más común que representa a los valores espaciales que componen una imagen, y que al ser evaluados en las funciones de entrada y de salida “f” y “g” respectivamente genera los valores de cada elemento o píxel que forma parte de una imagen. Para el caso de imágenes discretas el dominio x consiste en un conjunto de elementos finitos de la forma (i, j) en donde ambos representan índices enteros que hacen referencia a un píxel en especifico, renglón y columna, de modo que en términos matemáticos se define a x = (i, j) que al ser sustituidas en las ecuaciones anteriores produce la siguiente ecuación:

$$g((i,j)) = h(f((i,j))) \quad \text{..... (1.3)}$$

La ecuación 1.3 describe al igual que las ecuaciones 1.1 y 1.2 el uso de un operador general sobre una imagen pero de naturaleza discreta el cual mapea posiciones espaciales en los valores individuales de cada imagen.

Operadores de punto

Los operadores de punto representan las transformaciones más simples que se le pueden aplicar a una imagen y como ya se mencionó, su funcionamiento radica en la modificación de cada píxel haciendo uso únicamente de la información contenida en estos(más el uso potencial de información o parámetros globales de la imagen).

Los operadores de punto son comúnmente representados mediante la multiplicación y adición de una constante sobre los valores de entrada lo cual se expresa por la ecuación 1.4:

$$g(x) = a * f(x) + b \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1.4)$$

En la cual los parámetros a>0 y b son conocidos como parámetros de ganancia y de desviación y en algunas ocasiones controlan el brillo y saturación de la imagen y f(x) representa el valor del píxel contenido en la posición x de la imagen[5].

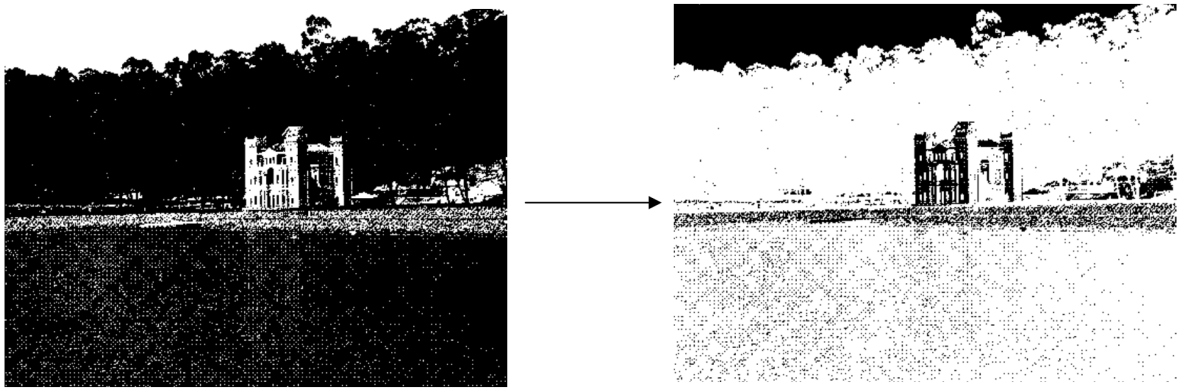


FIGURA 2.3 OBTENCIÓN DEL NEGATIVO DE UNA IMAGEN HACIENDO USO DE UN OPERADOR DE PUNTO.
(Figura tomada de [6])

En la figura 2.3 se muestra la generación de una imagen negativa el cual es un proceso que substrae el valor máximo posible del valor individual de cada píxel y queda expresado en términos matemáticos haciendo uso de la ecuación 1.4 mediante un parámetro alfa = -1 y con un parámetro beta = 255, para formatos de imágenes en el que 255 representa el valor máximo de un píxel.

Operadores de vecindad

Los operadores de vecindad son aquellos operadores que para la transformación de cada píxel que compone a una imagen hacen uso del valor del píxel de interés y de los valores de una región de pixeles limitada y cercana a este. Los operadores de vecindad pueden ser sub clasificados como operadores lineales en el cual radican las operaciones de filtrado y operadores no lineales como lo son las operaciónes morfológicas y de distancia.

Imagen original (Izquierda), Filtro gaussiano aplicado a la imagen (Derecha)

FIGURA 2.4 FILTRADO LINEAL DE UNA IMAGEN PARA OBTENER UN EFECTO DE DESENFOQUE.
(Figura inspirada en [23 y 24])

Entre las principales aplicaciones de los operadores de vecindad se encuentra el ajuste del tono de imagen, el filtrado de una imagen para obtener un desenfoque suave, afinar detalles, acentuar bordes o eliminar el ruido, entre otros.

Operadores de filtrado lineal

Los operadores de filtrado representan el tipo más común de operadores de vecindad y se describen en función de cómo transforman a cada píxel de entrada haciendo uso de este y los valores vecinos los cuales son procesados en forma de una suma ponderada para determinar el valor de dicho píxel en la imagen de salida, lo cual es expresado por medio de la ecuación 1.5.

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \quad \text{.....} \quad (1.5)$$

La ecuación 1.5 expresa el proceso de filtrado de una imagen por medio de una operación conocida como correlación cruzada, el cual opera sobre la región de la

imagen de entrada próxima al índice en el que se efectúa la operación, y realiza la multiplicación de cada elemento de esta región por un elemento perteneciente la función h(x) o coeficiente de filtrado dando como resultado un escalamiento de los valores que componen a la región de entrada mismos que son sumados transformando así el píxel correspondiente al índice (i, j) de la imagen de entrada.

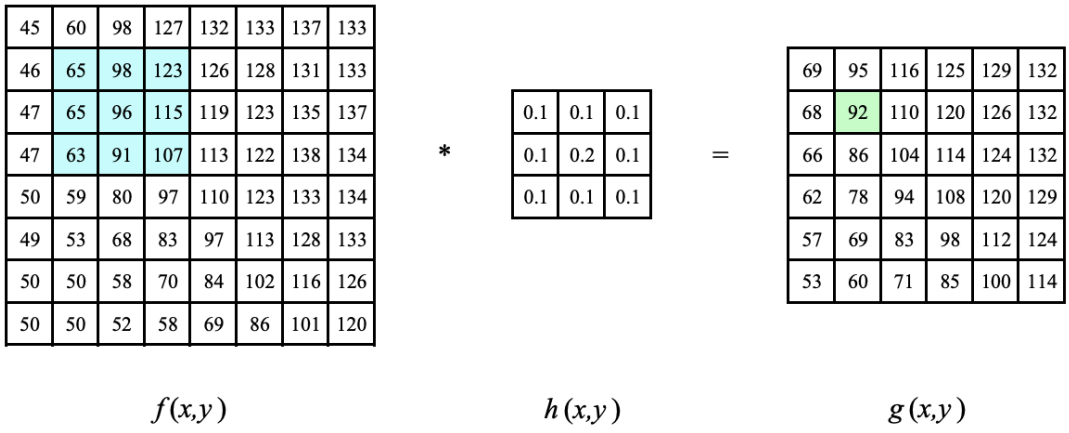


FIGURA 2.5 CONVOLUCIÓN DE UNA MATRIZ DE ENTRADA F POR UN KERNEL H.
(Figura tomada de [3, p. 46])

Una de las operaciones de mayor relevancia en el área de la visión por computadora es conocida como operación de convolución la cual es expresada por una variación de la ecuación 1.5 en la que se invierte a la función f(x, y) dando como resultado la ecuación 1.6.

$$g(i,j) = \sum_{k,l} f(k,l)h(i-k, j-l) \quad \text{.....} \quad (1.6)$$

Tanto la operación de correlación cruzada como la de convolución son funciones ampliamente usadas matemáticamente y están definidas de manera breve mediante los operadores de convolución $\circledast$ y de correlación $\otimes$ entre la función de entrada y una función h(x) conocida en el contexto del filtrado como kernel y en el contexto de señales como función de respuesta impulso.

2.1.2 Reconocimiento de imágenes

El reconocimiento de imágenes es uno de los principales campos que compone a la visión por computadora y se centra como su nombre lo indica en el estudio de las técnicas que permiten a los algoritmos el reconocimiento o clasificación de un elemento o patrón presente en una imagen.

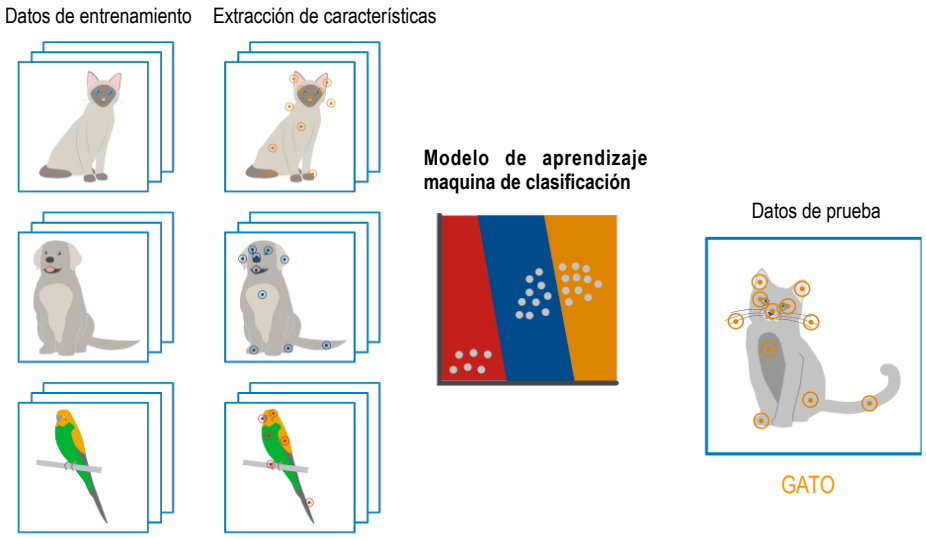


FIGURA 2.6 PROCESO DE CLASIFICACIÓN DE IMÁGENES MEDIANTE CARACTERÍSTICAS Y UN MODELO DE APRENDIZAJE MÁQUINA.
(Figura modificada de [7])

El reconocimiento de imágenes es considerado una tarea de alto nivel debido a que para su funcionamiento se emplea, en algunos casos, el resultado de procesos previos o inferiores aplicados sobre una imagen como lo son por ejemplo la detección de características y la coincidencia de patrones.

Dentro de las aplicaciones o subáreas que componen a el reconocimiento de imágenes destacan los procesos de reconocimiento de instancias, reconocimiento de clases, detección de objetos, segmentación semántica, entre otras, mismos que son enlistados en función de su complejidad siendo los procesos de menor complejidad los relacionados con el reconocimiento general de un objeto, sin embargo independientemente de la complejidad de todos estos, los procesos de reconocimiento hacen uso ya sea de técnicas clásicas junto con técnicas de aprendizaje maquina o de técnicas de aprendizaje profundo para su funcionamiento.

Descriptor BOF

Los algoritmos clásicos de reconocimiento de imágenes funcionan principalmente gracias a su habilidad de detectar características relevantes o únicas en una imagen como lo pueden ser por ejemplo las esquinas de un edificio mismas que posteriormente son descritas en función de los elementos que rodean a estas características por ejemplo la textura del edificio, el trasfondo de este, etc. De modo que mediante una serie de características y su descripción es posible generar una representación global de lo que un objeto o imagen representa.

El algoritmo de "*Boundary object function(BOF)"* plantea la descripción de una imagen como función de su centro geométrico y del contorno de esta, en donde primeramente se discretiza al contorno en N puntos que corresponden a la intersección del contorno con diferentes rectas que parten desde el centro geométrico de la figura y cuya pendiente es función de un parámetro ϕ. De modo que posterior a la discretización del contorno se calcula para cada elemento de este la distancia existente entre este y el centro geométrico de la figura lo cual puede ser graficado como función de ϕ dando como resultado una curva representativa del objeto.

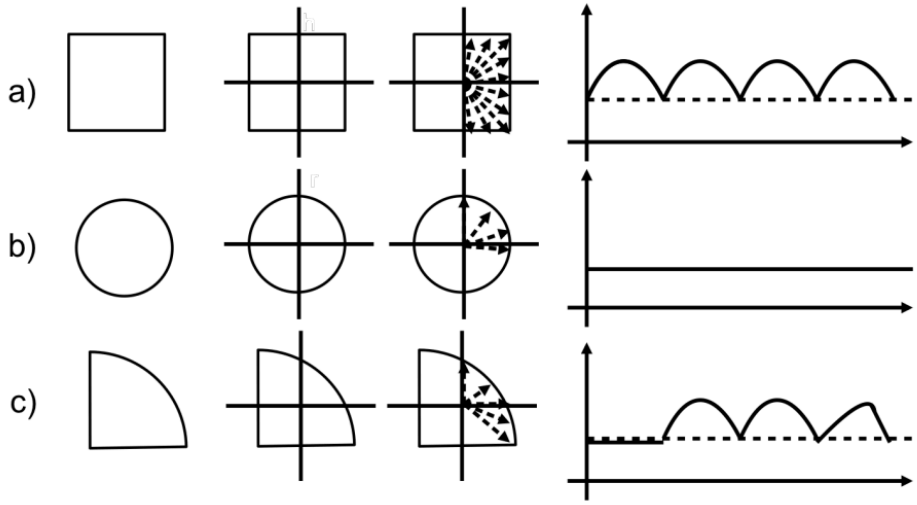


FIGURA 2.7 PROCESO DE DESCRIPCIÓN DE UNA FIGURA UTILIZANDO UNA BOF.
(Figura tomada de [4])

Mediante el uso de una BOF se puede representar a un objeto en forma vectorial de dimensión constante en donde dicha representación cuenta con las ventajas de ser invariante a la translación, invariante a la escala e invariante a la rotación.

2.2 Aprendizaje máquina

El aprendizaje de máquina o *machine learning(ML)* es un área de la inteligencia artificial(IA) que surge de la necesidad de adquirir conocimiento de manera automática por parte de los sistemas de inteligencia artificial mismo que representó un cambio de paradigma dentro del campo de la IA y de la programación debido a que en vez de generar o establecer el conjunto de instrucciones para resolver los problemas este se enfoca en la creación de algoritmos capaces de modificar su comportamiento en función de los datos y de esta forma llegar a la solución[3, p. 2,3].

Formalmente el *machine learning* se define como una subrama de la inteligencia artificial que se dedica al estudio de algoritmos avanzados que según el profesor Tom Mitchell (2006) surgen de la intersección de las ciencias de la computación con la probabilidad y estadística. Y su área de estudio gira alrededor de cómo hacer que las computadoras se programan a sí mismas con base en la definición de una estructura y de experiencias previas en forma de datos de entrenamiento. Entre las definiciones más aceptadas por su relevancia destacan las siguientes:

"*Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.*" Arthur Samuel (1959).

El aprendizaje máquina es el campo de estudio que brinda a las computadoras la habilidad de aprender sin ser explícitamente programadas. Arthur Samuel (1959).

"*A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*" Tom Mitchell (1997)

Se dice que un programa de computadora aprende de la experiencia E respecto a una tarea T y con base en métrica de rendimiento P, si su rendimiento en T medido por P mejora con la experiencia E. Tom Mitchell (1997)

Ambas definiciones recalcan el hecho de que el aprendizaje máquina consiste en un tipo de programa especial que tiene como característica principal la habilidad de "aprender" acerca de una tarea específica mediante el uso de datos de entrenamiento y sin que el ser humano intervenga de manera activa en el proceso. Y se entiende el proceso de aprendizaje como aquel que ocurre siempre y cuando los resultados del algoritmo mejoren con respecto a la cantidad de experiencia acumulada.

A nivel conceptual un programa convencional es una serie de instrucciones que debe ejecutar una computadora para realizar una tarea[1, p. 5], éste debe interactuar con diferentes tipos de datos de entrada y puede generar una o múltiples salidas, sin embargo, para su funcionamiento es necesario un proceso previo de razonamiento

humano, desafortunadamente existen tareas que por su naturaleza resultan muy difíciles de analizar y representar a nivel computacional como por ejemplo el identificar en una foto a los diferentes objetos presentes, o el cómo encontrar un patrón en específico en grandes volúmenes de información, por ello para problemas en donde resulta difícil encontrar una solución directa se han generado algoritmos de aprendizaje máquina que a diferencia de los convencionales no necesitan ser explícitamente programados ya que funcionan con los datos relativos a la tarea en específico que desea realizarse, y mediante el uso de estos algoritmos se aprende a generar las acciones necesarias, o en otras palabras son capaces de generar el programa deseado, que para los ejemplos anteriores sería aprender de manera computacional a identificar objetos o a buscar patrones en los datos.

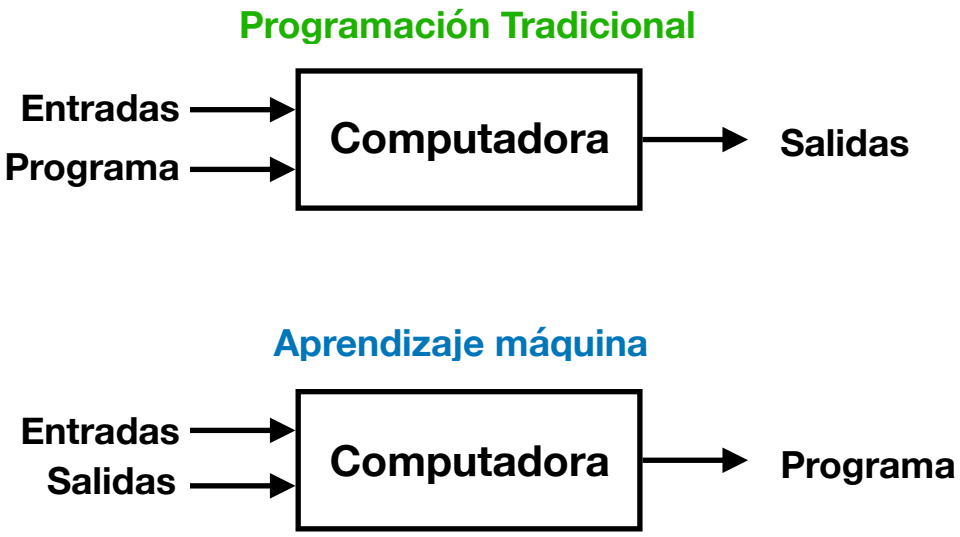


FIGURA 2.8 COMPARACIÓN ENTRE PROGRAMACIÓN TRADICIONAL Y APRENDIZAJE AUTOMÁTICO.
(Figura inspirada en [8])

En la figura 2.8 se muestra una comparación directa a nivel de entradas y salidas, de un programa tradicional y de un programa de aprendizaje automático, donde se observa que mientras que para que una computadora ejecute un programa convencional necesita de un código o programa que indique los pasos a seguir por la computadora, un programa basado en *machine learning* genera estos pasos o programa de manera automática mediante el uso de los datos de entrada y los datos de salidas deseadas para esas entradas.

En la práctica el aprendizaje máquina tiende a ser usado en cuatro áreas principales como lo son: Las situaciones en las que se busca predecir un valor real a partir de un conjunto de valores de entrada, a este proceso se le conoce como

regresión. De manera análoga a la regresión existen casos en donde lo que se desea predecir a partir de los datos de entrada es la pertenencia de estos a una clase o categoría en especifico, a este proceso se le conoce como clasificación. Existen también casos en que solo se busca encontrar relaciones o agrupaciones en los datos, a este proceso se le conoce como *clustering*, finalmente hay casos donde se desea buscar el mapeo de la información de entrada a un espacio de menor dimensión en el que se preserven las características principales de los datos, a este proceso se le conoce como de reducción dimensional.

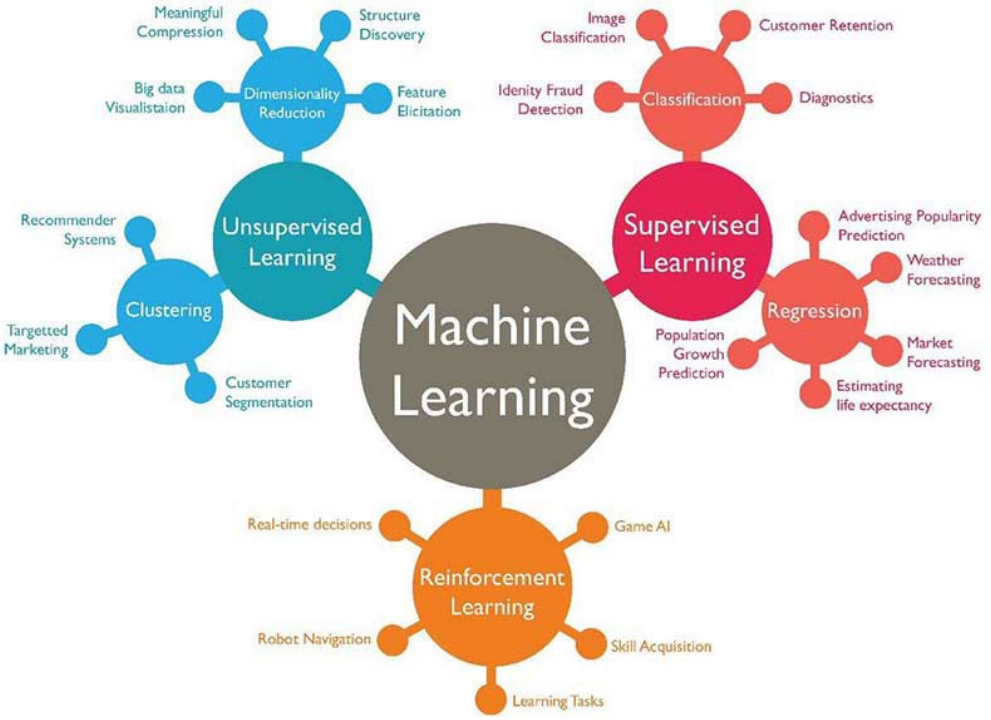


FIGURA 2.9 CLASIFICACIÓN GENERAL DEL APRENDIZAJE MÁQUINA.
(Figura tomada de [9])

En la figura 2.9 se muestran las principales áreas y subáreas que componen el aprendizaje máquina y el tipo de aplicación correspondiente a cada una de ellas.

2.2.1 Aprendizaje supervisado.

El aprendizaje supervisado es aquel que requiere para su funcionamiento un conjunto de datos etiquetados como entrada indicando por cada dato de entrada sus respectivas salida(s) esperadas lo cual puede expresarse mediante la tupla (X, y) en donde “X” representa el conjunto de los datos de entrada del problema y “y” representa el conjunto de salidas deseada para cada entrada en “X” y busca como su nombre lo indica dar sentido a la información de entrada del modelo tomando como guía la salida esperada por parte del modelo para dicha entrada tratando con esto que cada predicción por parte del modelo sea lo más cercana a las etiquetas en el set de entrenamiento, de modo que al finalizar el proceso de entrenamiento se espera el modelo sea capaz de detectar y generalizar las características de entrada o internas que permitan a este producir una salida lo más cercana a la esperada incluso en datos desconocidos para el modelo.

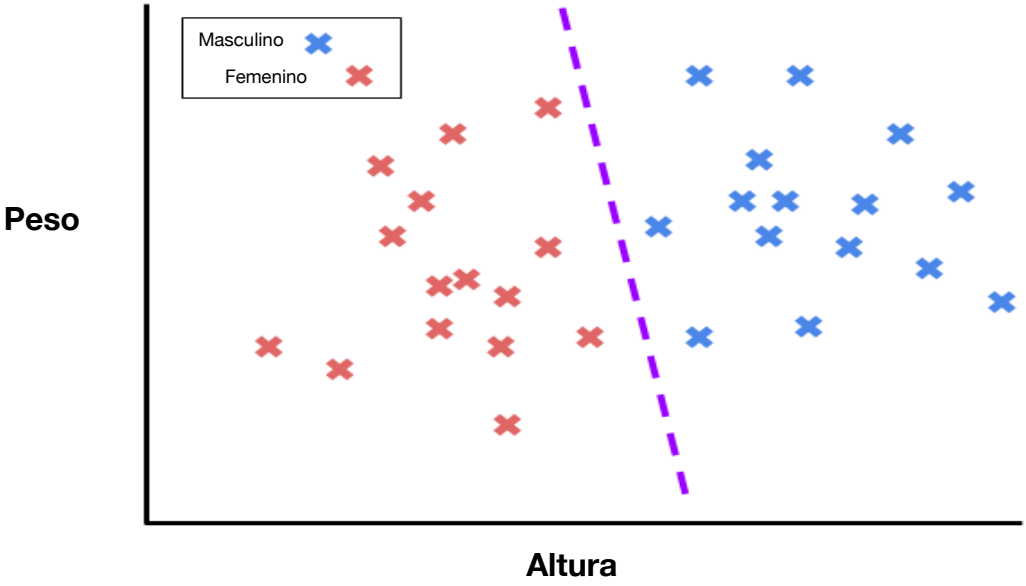


FIGURA 2.10 EJEMPLO DE UN ALGORITMO DE APRENDIZAJE SUPERVISADO.
(Figura modificada de [10])

En la figura 2.10 se muestra un ejemplo del resultado de la aplicación de un modelo de clasificación a un conjunto de individuos representados por su peso y altura. En donde la variable a clasificar es el género y las entradas son el peso y altura, en la figura se aprecian también los diferentes puntos que representan a los individuos y una la línea punteada que representa la frontera de decisión del modelo, es a través de esta recta que el algoritmo es capaz de clasificar a individuos desconocidos usando únicamente su peso y altura.

2.2.2 Aprendizaje no supervisado.

El aprendizaje no supervisado por otra parte engloba a todos aquellos algoritmos que sólo necesiten el conjunto de datos de entrada ”X” sin la necesidad de especificar una salida esperada o etiqueta “Y” para el proceso de aprendizaje. En general existen ocasiones en las que generar una etiqueta para cada entrada en el proceso de aprendizaje resulta impráctico y en ocasiones imposible, por lo que se han desarrollado una serie de algoritmos que buscan encontrar estructuras y grupos en los datos usando únicamente las características que los definen, un ejemplo de esto son los algoritmos de *clustering* que son usados por ejemplo en sistemas de recomendación, primeramente generando grupos de contenido y posteriormente usando los elementos más cercanos en estos para generar posibles preferencias del usuario.

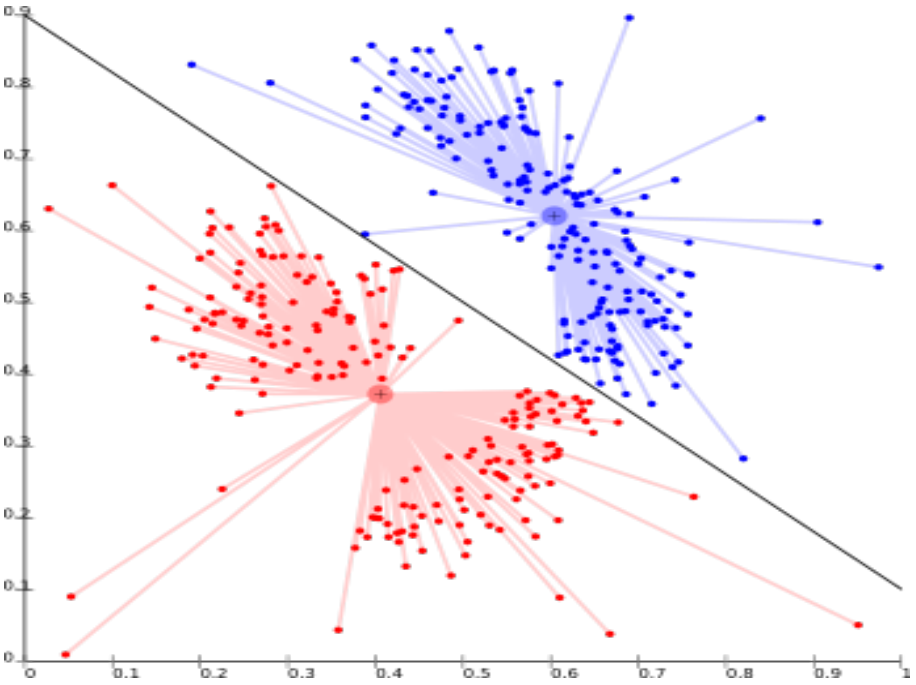


FIGURA 2.11 EJEMPLO DE UN ALGORITMO DE APRENDIZAJE NO SUPERVISADO.
(Figura tomada de [11])

En la figura 2.11 se observa la aplicación de un algoritmo de aprendizaje no supervisado conocido como “*k-means”[26]*, este algoritmo funciona generando N centroides definidos por el usuario en lugares aleatorios y recalculando cada uno de ellos de modo que mejor describan a los datos de entrada, al final del proceso se contará con el conjunto de datos de entrada dividido en el mismo número de clases que centroides especificados.

2.2.3 Aprendizaje por refuerzo.

La última familia del aprendizaje automático abarca a los algoritmos de aprendizaje por refuerzo el cual es un paradigma que plantea un proceso de aprendizaje que se lleva a cabo entre un modelo o agente y un entorno, en donde se desea que el agente aprenda a generar un comportamiento óptimo en función de la situación en la que este se encuentre, lo cual quiere decir que el aprendizaje por refuerzo busca la generación de comportamientos que permitan a un agente cumplir con un objetivo en particular de manera óptima por lo que el entorno debe de contener todas las dinámicas que deseamos el agente aprenda así como una manera de guiar las acciones del agente hacia el objetivo o tareas deseadas, esta retroalimentación por parte del entorno recibe el nombre de señal de recompensa y se representa como un valor numérico que puede ser positivo o negativo, según la situación del agente.

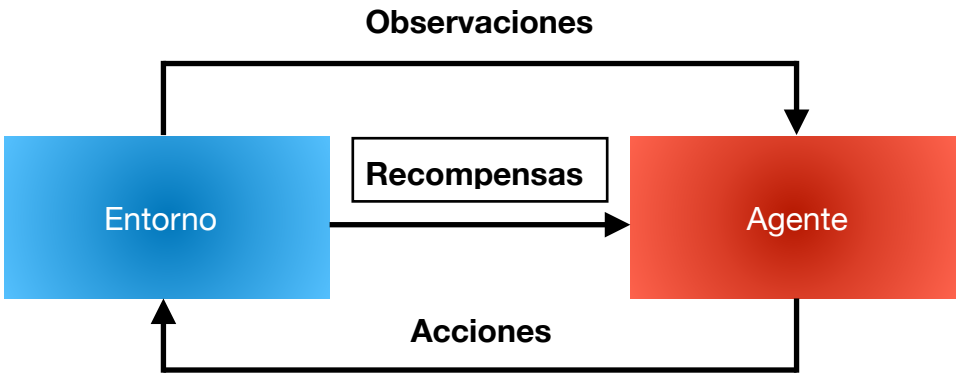


FIGURA 2.12 PARADIGMA DEL APRENDIZAJE POR REFUERZO.
(Figura inspirada en [12])

El diagrama de la figura 2.12 describe el proceso de interacción entre el agente y su entorno, en donde en un principio el agente solo cuenta con una observación inicial, que representaremos de manera abstracta como un estado inicial, posteriormente el agente emitirá en principio una acción aleatoria, ya que desconoce el objetivo y el efecto de sus acciones, como resultado de esa primer acción el agente obtendrá nuevamente una observación que representaremos como un nuevo estado a la vez que obtendrá retroalimentación por parte del entorno en forma de la señal de recompensa, siguiendo de esta manera el modelo o agente aprenderá guiándose en la señal de recompensa y a medida que su aprendizaje progrese sus acciones comenzarán a ser las más óptimas para un estado dado, acorde a lo que haya aprendido.

2.3 Aprendizaje profundo.

El aprendizaje profundo o *Deep learning*(DL) es una subrama del aprendizaje automático y por tanto de la inteligencia artificial que surge del crecimiento y mejora de algoritmos inspirados en el funcionamiento de las neuronas biológicas. El aprendizaje profundo se caracteriza por el uso de redes neuronales artificiales de múltiples capas y se diferencia de otros algoritmos de aprendizaje automático por su capacidad de generar representaciones internas de la información, haciendo innecesaria una etapa previa de extracción de características.

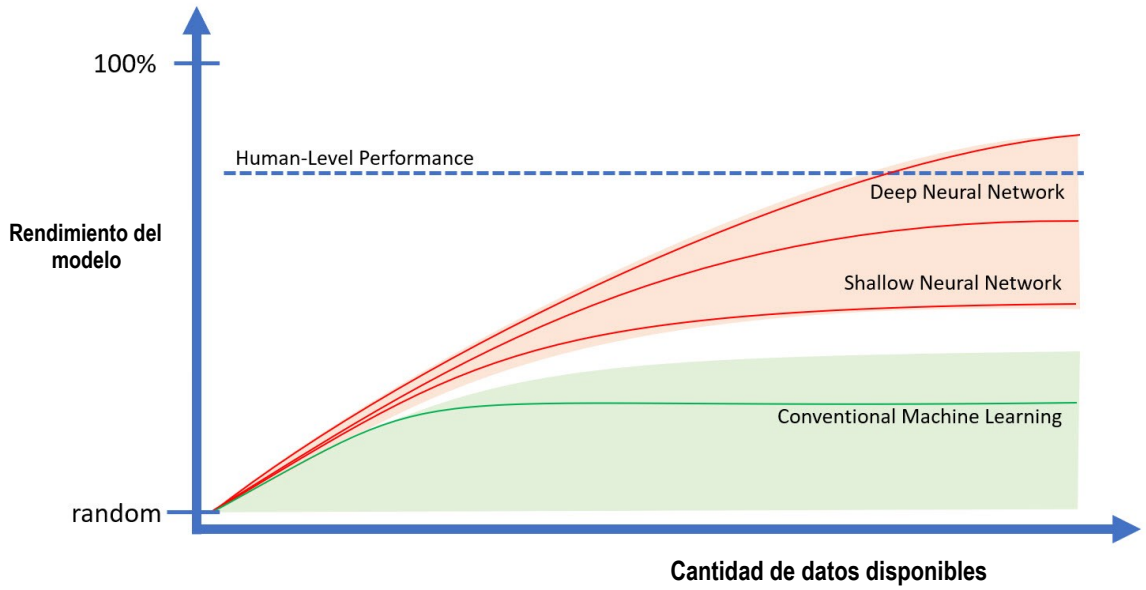


FIGURA 2.13 COMPARACIÓN DEL RENDIMIENTO DEL APRENDIZAJE PROFUNDO EN FUNCIÓN DE LA CANTIDAD DE DATOS DISPONIBLES.
(Figura modificada de [13])

La figura 2.13 Ilustra el rendimiento teórico máximo de los algoritmos que componen al aprendizaje máquina, las redes neuronales simples y el aprendizaje profundo, siendo este último el que mejor aprovecha la cantidad de datos disponibles.

Las redes neuronales artificiales o ANNs son además consideradas un aproximador universal de funciones y buscan en principio imitar algunas de las características básicas de las neuronas biológicas como lo son su capacidad de procesar múltiples entradas en paralelo y la capacidad de reaccionar de manera selectiva a cierto tipo de información de entrada, como lo hacen por ejemplo: El perceptrón; modelo matemático-computacional análogo a la neurona biológica. Y el perceptrón multicapa; modelo que conecta de manera horizontal y secuencial a diferentes perceptrones con la finalidad de mejorar la capacidad de uso respecto al perceptrón simple.

2.3.1 Red Neuronal Convolucional.

Las redes neuronales convolucionales(CNNs) son un tipo de red neuronal que surge como una red especializada en el procesamiento de la información visual y su desarrollo está inspirado en el entendimiento que se tenía a mediados del siglo XX sobre el funcionamiento biológico de nuestras neuronas visuales en el cual se observó que nuestra corteza visual estaba compuesta por dos tipos de células, células simples que reaccionaban ante un patrón determinado en una región determinada y células complejas que reaccionaba a un mismo patrón independientemente de la región en la que se encontrará, de modo que estas poseían un comportamiento invariante en el espacio, en donde se modeló a las células complejas como el resultado de la suma de la salida de diferentes células simples encargadas del mismo patrón. Del mismo modo las CNNs surgen como un modelo que busca determinar primeramente las características más simples en una imagen y posteriormente con ellas generar progresivamente características más complejas que faciliten las diferentes tareas relacionadas al reconocimiento de imágenes.

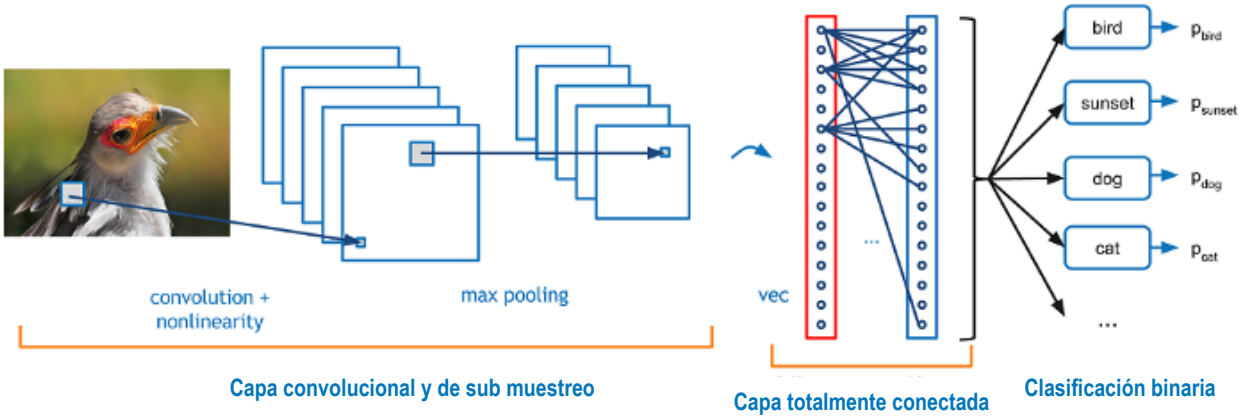


FIGURA 2.14 COMPOSICIÓN INTERNA DE UNA RED CONVOLUCIONAL.
(Figura modificada de [14])

Las CNNs están compuestas por diferentes tipos de capas de neuronas encargadas de un tipo de operación en particular con el fin de detectar de manera eficiente los patrones visuales en una imagen. Las cuales pueden ser:

- Capas convolucionales; que tienen como fin extraer las características más relevantes en una imagen por medio de múltiples convoluciones de matrices.
- Capas de activación; que aplican sobre las características detectadas una función de activación no lineal.
- Capas de submuestreo; que tienen como finalidad la reducción dimensional de las entradas.
- Capas totalmente conectadas; que se sitúan al final de la red y tienen por objetivo clasificar o predecir los valores de salida de la red tomando como entrada las características aprendidas.

Capa convolucional.

La tarea principal de una capa convolucional es la de detectar características o patrones específicos en una entrada dada, la cual puede ser una imagen o el resultado matricial de la detección de patrones realizado en capas anteriores. Dichas detecciones varían en complejidad en función de qué tan alejada o profunda sea la capa en la que se están detectando, siendo las primeras capas las encargadas de detectar los patrones más simples en la imagen como contornos y las capas más profundas las encargadas de detectar los patrones más complejos como la existencia de objetos o características de alto nivel.

FIGURA 2.15 EJEMPLO DE CARACTERÍSTICAS DETECTADAS POR UNA CAPA CONVOLUCIONAL EN FUNCIÓN DE LA PROFUNDIDAD DE ESTA.
(Figura modificada de [15])

El principio de funcionamiento de una capa convolucional se basa en una operación existente del procesamiento de imágenes conocida como filtrado, la cual se lleva a cabo realizando la convolución entre una imagen de entrada y una matriz cuadrada conocida como filtro o kernel, cuyos valores se encuentran diseñados matemáticamente acorde a las características o patrones que se deseen detectar en la imagen de entrada, sin embargo a pesar de que una capa convolucional hace uso de una variante de esta operación, en esta los valores de cada kernel son parámetros entrenables del modelo y se adaptan en el proceso de entrenamiento para detectar características útiles en las imágenes de entrada.

$$G[m,n] = (f * h)[m,n] = \sum_{j} \sum_{k} h(j,k) f(m-j, n-k) \ldots\ldots \quad (2.1)$$

Ecuación 2.1 Cálculo de la convolución de una matriz **f** y una matriz **h** para el valor correspondiente a la posición (m, n) de la matriz de salida.

Matemáticamente se define el uso de una capa convolucional sobre una matriz de entrada $F \in M_{h_x w_x c}(\mathbb{R})$ como el proceso de aplicar la correlación entre la imagen de entrada y múltiples filtros pertenecientes a la capa convolucional, $H^{(i)} \in M_{n_x n_x c}(\mathbb{R})$ en donde i representa el número de filtros a aplicar mismos que contienen a los parámetros entrenables del modelo, posterior a esto se agrupa a los resultados de cada convolución individual en una única matriz de salida G de la forma $G \in M_{h'_x w'_x i}(\mathbb{R})$; finalmente a la matriz de resultados G se le suman parámetros de bias y se le aplica una función no lineal $f(G)$ dando como resultado la salida de la capa.

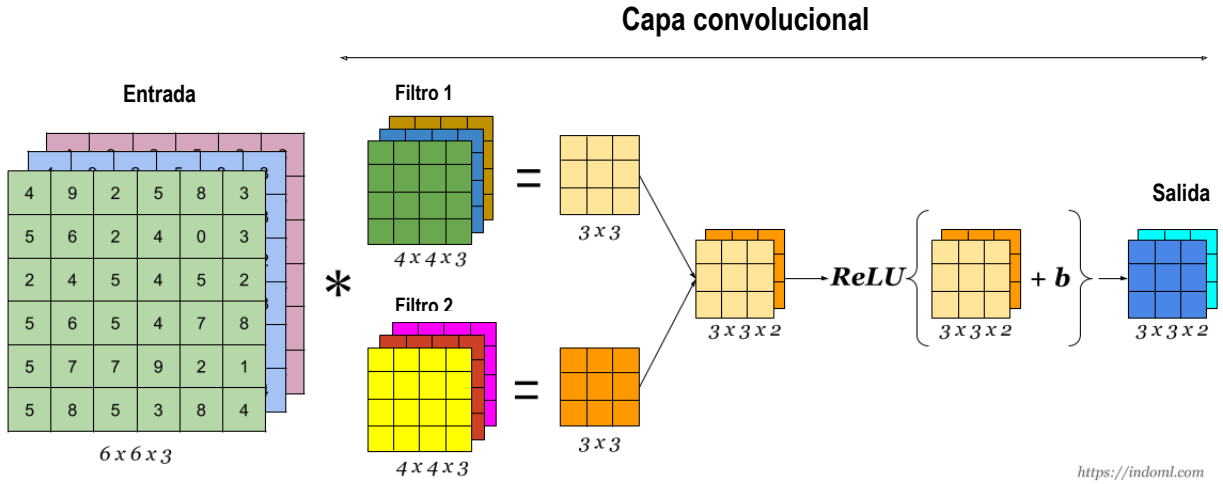


FIGURA 2.16 PROCESAMIENTO DE UNA IMAGEN DE ENTRADA POR UNA CAPA CONVOLUCIONAL.
(Figura modificada de [16])

La figura 2.16 ejemplifica el proceso que se lleva a cabo dentro de una capa convolucional en el cual se aprecia la convolución de una imagen de entrada de tres canales con dos filtros pertenecientes a la capa convolucional dando como resultado dos matrices correspondientes a cada operación conocidas como mapeo de características mismas que son agrupadas y transformadas para generar la salida de la capa.

Capa de sub-muestreo.

Las capas de submuestreo mejor conocidas por su nombre en inglés como "*pooling layers"* tienen la función de disminuir la dimensionalidad de la matriz de entrada haciendo que las operaciones computacionales sean menos costosas además de hacer más robustas algunas de las características detectadas por las capas convolucionales y agregar cierto nivel de invarianza al desplazamiento vertical y horizontal de las entradas.

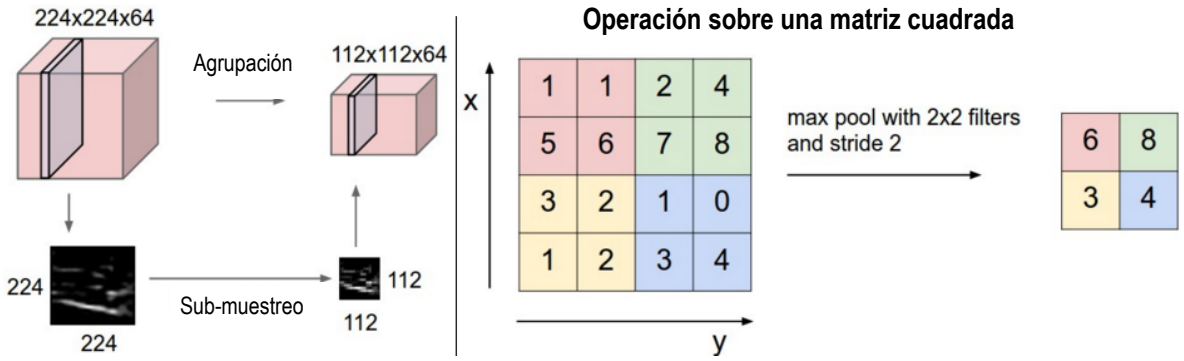


FIGURA 2.17 APLICACIÓN DE UNA CAPA DE SUB-MUESTREO A DIFERENTES ENTRADAS.
(Figura modificada de [17])

Al igual que las capas convolucionales las capas de submuestreo funcionan con base en el uso de kernels y al desplazamiento de este sobre la imagen, sin embargo a diferencia de la convolución matricial el kernel no contiene parámetros entrenables y sólo reduce a un solo valor a los valores de la entrada que se encuentren en la región del kernel, esta reducción ocurre siguiendo alguna de las dos variantes principales de las capas de submuestreo conocidas como "*max pooling*" en donde se selecciona el máximo de los valores de la región o "*average pooling*" en donde se promedia a los valores de la región. Dicho proceso se ilustra en la figura 2.17 que ilustra la aplicación de una capa de max pooling sobre dos diferentes tipos de entrada, en las que se aprecia que el resultado de esta operación contiene el mismo número de canales que la entrada pero su dimensión espacial es reducida a la mitad, esto es debido a los hiperparámetros de la capa que por lo regular suelen ser kernels de 2x2 y desplazamientos de 2 unidades.

Capa totalmente conectada.

Las capas totalmente conectadas, capas densas o capas de propagación hacia delante representan el último componente en una red convolucional y están mayoritariamente situadas al final de esta ya que es gracias a ellas que se generan las predicciones finales de la red. Las capas totalmente conectadas son en esencia percetrones multicapa(MLP) que toman como entrada a las características detectadas en las capas anteriores de la red, mismas que es necesario se encuentren en forma vectorial unidimensional, y haciendo uso de los parámetros de la capa aprenden a generar las salidas finales esperadas por parte de la red las cual es pueden ser por ejemplo la clasificación de una imagen de entrada o la predicción de diferentes valores continuos a partir de esta siendo un ejemplo de ambos la detección de objetos de interés.

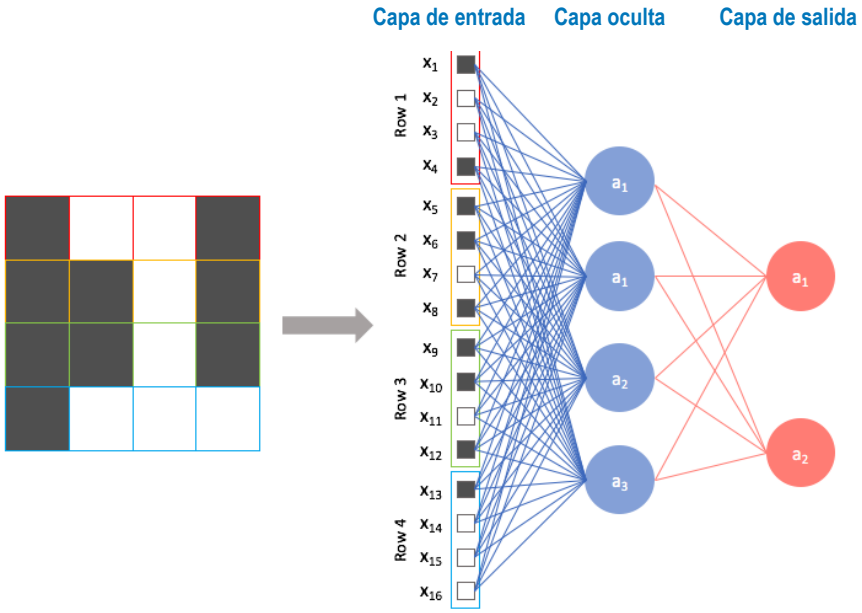


FIGURA 2.18 PROCESAMIENTO DE UNA MATRIZ DE ENTRADA POR PARTE DE UNA CAPA TOTALMENTE CONECTADA.
(Figura modificada de [18])

La figura 2.18 Muestra el ejemplo de aplicación de una capa totalmente conectada sobre una matriz de entrada la cual es primeramente aplanada siguiendo un orden predeterminado, posteriormente la capa realiza una suma ponderada por cada neurona en la capa entre los valores o características de la entrada y los parámetros de esta, finalmente y al igual que en otras capas, al resultado de cada operación se le añade un valor de bias y se le aplica una función no lineal para generar así el resultado de la capa.

2.3.2 Vision Transformer

El *Vision Transformer*(ViT) surge como la adaptación hacia el área de la visión por computadora de uno de los modelos más sobresalientes del aprendizaje profundo conocido como "*Transformer" el cual* fue desarrollado originalmente para tareas de procesamiento del lenguaje natural y que pertenece a la familia de los modelos basados en el concepto de "*attention*". El *ViT* a diferencia de las CNNs usa como operación principal una serie de mecanismos conocidos como de atención, mismos que en esencia funcionan ponderando la relación de cada elemento de entrada con todos los demás elementos de esta, de modo que con ello generan representaciones intermedias conocidas como mapas de atención y representaciones enriquecidas de las entradas.

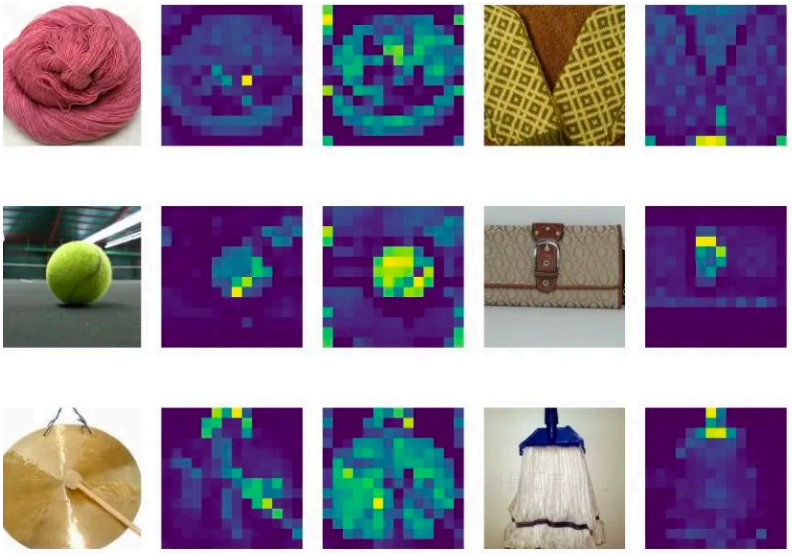


FIGURA 2.19 MAPAS DE ATENCIÓN QUE ILUSTRAN LA IMPORTANCIA QUE LE DA UNA CAPA DEL VIT A UNA REGIÓN EN ESPECIFICO DE LA IMAGEN DE ENTRADA.
(Figura modificada de [19])

Los ViT funcionan convirtiendo primeramente a una imagen de entrada en múltiples regiones cuadradas conocidas como parches, mismos que son reorganizados en forma de vectores unidimensionales y son procesados a continuación por una capa de codificación posicional que sirve para mantener el orden espacial de cada parche, posteriormente se hace uso de una serie de capas de codificación para la generación y detección de características, finalmente para tareas de clasificación es aplicada una capa totalmente conectada sobre la salida de las capas de codificación.

El componente principal del ViT es el bloque codificador el cual es igual que el del modelo original "*Transformer*" y está compuesto de diferentes subcapas y conexiones internas tal como se ilustra en la figura 2.20 En donde en un principio se observa la normalización de las entradas del bloque y su procesamiento por el mecanismo de atención conocido como "*Multi-head attention*" para posteriormente agregar al resultado del mecanismo de atención la entrada de la capa misma que una vez más es normalizada y posteriormente procesada por una capa de propagación hacia delante, finalmente al resultado de la capa totalmente conectada se le suma el resultado obtenido por el mecanismo de atención.

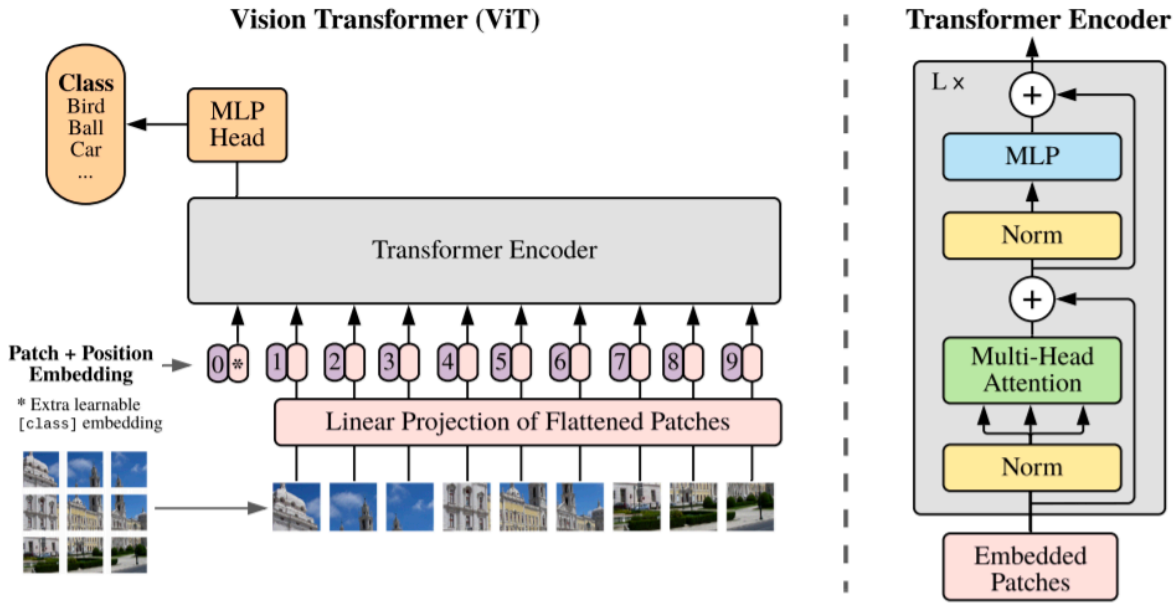


FIGURA 2.20 *(IZQUIERDA)* ARQUITECTURA GENERAL DE UN *VISION TRANSFORMER, (DERECHA) COMPONENTES INTERNOS QUE COMPONEN A EL CODIFICADOR*.
(Figura tomada de [20])

2.3.3 Modelos de variables latente

Dentro de los modelos estadísticos se le denomina como variables latentes al conjunto de variables que se asume existen dentro de los datos pero no pueden ser observadas o medidas de manera directa por lo que su valor es inferido a partir de las variaciones existentes en las variables observables.

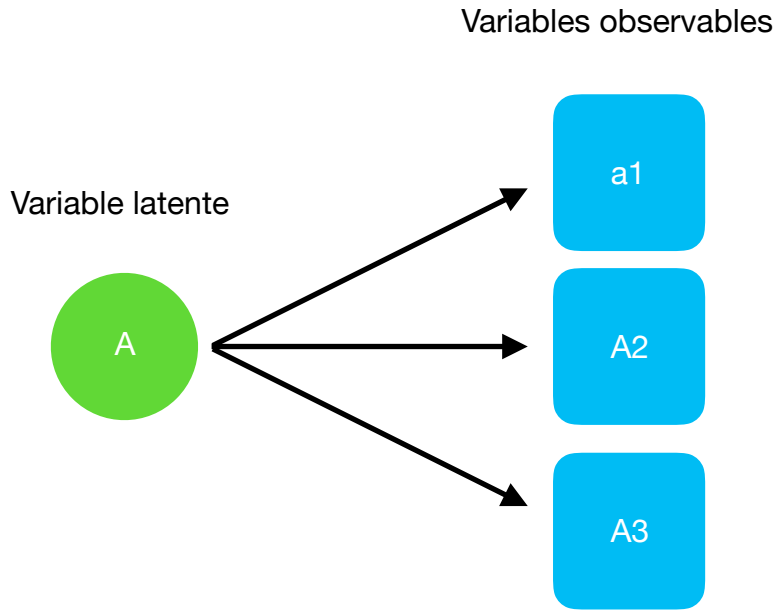


FIGURA 2.21 RELACIÓN ENTRE LAS VARIABLES LATENTES Y LAS VARIABLES OBSERVABLES.

La figura 2.21 muestra la relación existente entre una variable latente A y las variables observables a1, a2, a3 las cuales son modeladas como función de A por lo que para inferir el valor de A se emplea a las variaciones existentes en ellas.

Un ejemplo de variable latente respecto a la información visual podría ser la noción de la pose dentro de un objeto ya que esta se encuentra implícita en el estado del objeto pero no puede ser accesible empleando únicamente los valores de los pixeles que componen la imagen. Y del mismo modo se le denomina modelo de variable latente a aquellos modelos que buscan explicar a las variables observables en función de las variables latentes, lo cual para el ejemplo de la información contenida en una imagen consiste en un modelo que dado un objeto o su descripción se pueda explicar su configuración visual en función de la pose existente en este. Entre los modelos de variable latente destacan dentro de las ANN el uso de autocodificadores variacionales y las redes adversarias generativas.

Autocodificadores.

Los Autocodificadores(AEs) son un tipo de red neuronal artificial que pertenece a la familia de algoritmos de aprendizaje no supervisado y tiene como fin el predecir el mismo valor que se recibe como entrada a la vez que en el proceso se genera un vector intermedio el cual se asume pertenece al espacio latente ya que en él se abstrae la información más relevante de los datos que permite al modelo la reconstrucción de estos, entre los principales usos de un autocodificador destacan su habilidad de comprimir la información cuando la dimensión del vector intermedio es de menor tamaño que la entrada así como su habilidad de pre-entrenar los pesos de las capas de una red cuando se dispone de pocos datos.

Un autocodificador está compuesto de dos elementos principales: un codificador, que es el que procesa la información de entrada a través de una o múltiples capas y un decodificador que utiliza a la representación interna generada por el codificador y que de igual manera procesa esta representación en busca de regenerar a la entrada original mediante el uso de una o más capas.

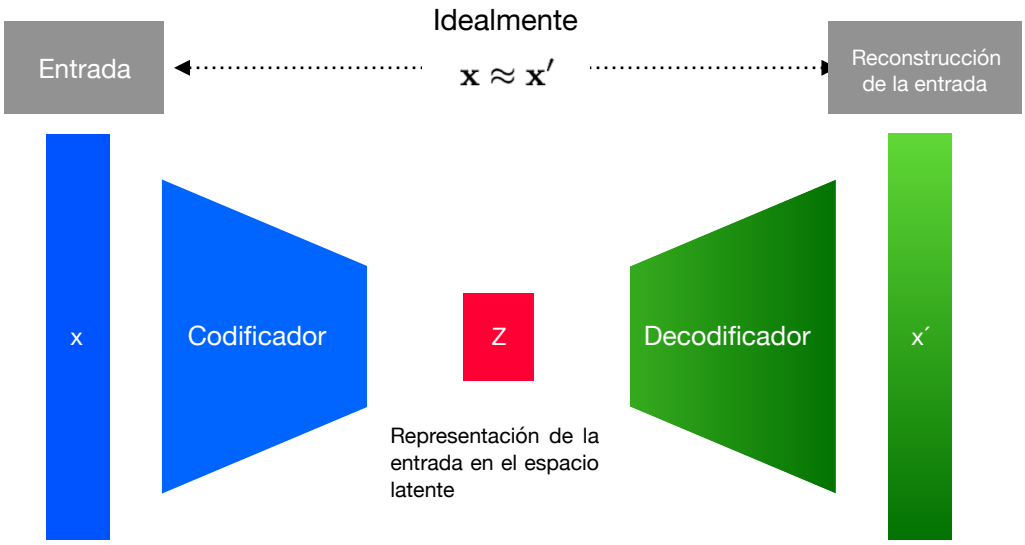


FIGURA 2.22 DETALLE DE LOS BLOQUES Y FUNCIONAMIENTO DE UN AUTOCODIFICADOR.
(Figura inspirada en [21])

Tal como puede apreciarse en la figura 2.22 los módulos y flujo base que componen a un autocodificador son el codificador y el decodificador los cuales no están limitados a un tipo o número de capa en específico y pueden ser adecuados según la tarea que se desee realizar. Además de esto se puede observar en medio del codificador y el decodificador a un vector “Z” que contiene la representación interna de las entradas, la cual posee el filtrado de los atributos más importantes en las entradas.

Autocodificadores Variacionales

Los autocodificadores variacionales(VAEs) representan una mejora sustancial sobre los autocodificadores clásicos ya que en ellos se mejora la capacidad de representación del modelo sobre el espacio latente logrando que cualquier configuración en este tenga una representación coherente en la salida del modelo a lo cual se le conoce como propiedad de integridad, así mismo se logra que las variaciones cercanas en el espacio latente generen variaciones próximas de la misma magnitud en la salida del modelo a lo cual se le conoce como propiedad de continuidad de modo que gracias a ambas propiedades es posible generar resultados coherentes no vistos durante la fase de entrenamiento del modelo.

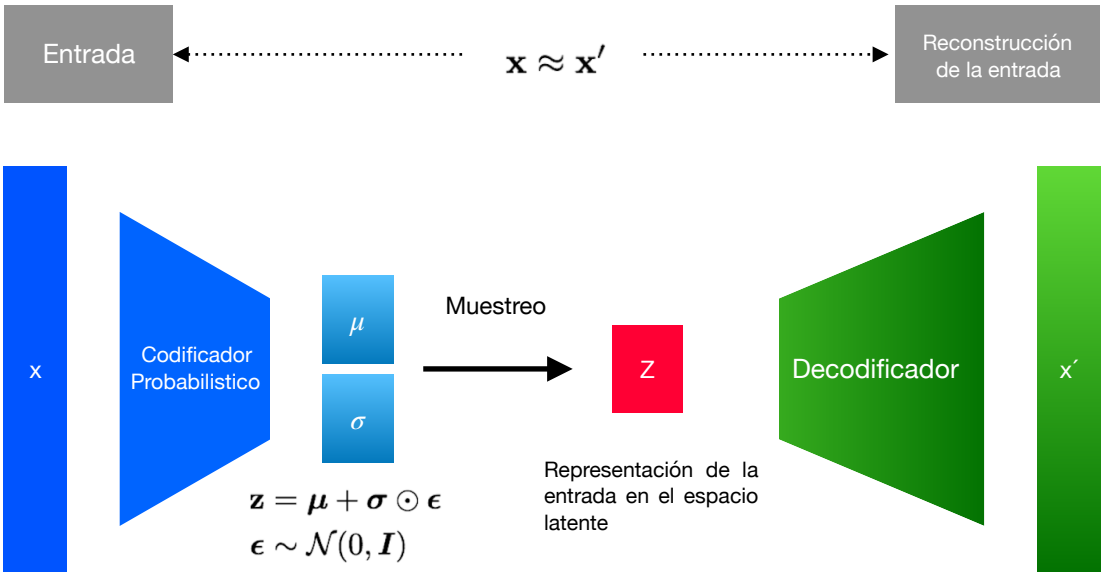


FIGURA 2.23 DETALLE DE LOS BLOQUES Y FUNCIONAMIENTO DE UN AUTOCODIFICADOR VARIACIONAL.
(Figura inspirada en [21])

El funcionamiento de los autocodificadores variacionales es semejante al de los autocodificadores clásicos en los que primeramente se procesa a la información de entrada por medio de una red codificadora sin embargo a diferencia de un autocodificador clásico que genera directamente el vector de espacio latente “z” un autocodificador variacional predice como resultados intermedios dos vectores de parámetros μ y σ mismos que son empleados para modelar mediante una distribución gaussiana la distribución de los datos de entrada en el espacio latente y posteriormente muestrear de esta un elemento que represente a la información de entrada, tal como puede apreciarse en la figura .

Las redes adversarias generativas(GANs) son un tipo de modelo de espacio latente ampliamente usado debido a que cuentan con la capacidad de poder reproducir de manera muy cercana cualquier distribución en los datos de entrada sin importar la naturaleza de estos teniendo como ejemplos la creación de imágenes, música, voz, etc. Para su funcionamiento las redes adversarias generativas a diferencia de los autocodificadores variacionales destacan por su simplicidad ya que carecen de un codificador inicial que busque representar la distribución de los datos en el espacio latente y en su lugar usan como vector de espacio latente ruido aleatorio.

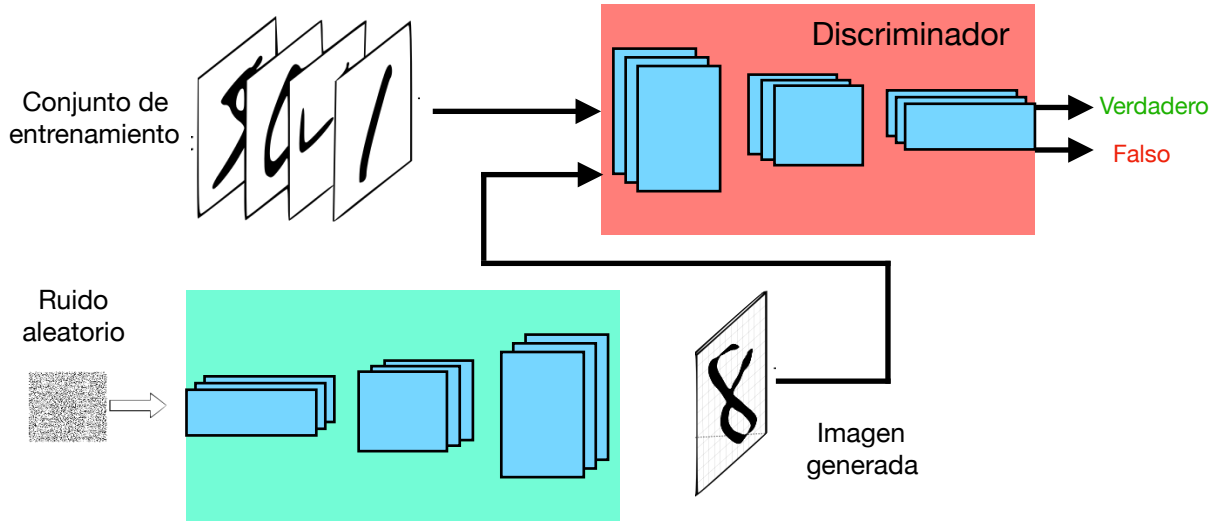


FIGURA 2.24 DIAGRAMA DE BLOQUES DE UNA RED ADVERSARIA GENERATIVA.
(Figura inspirada en [22])

Las redes adversarias generativas están compuestas por dos redes principales conocidas como generador y como discriminador, en donde el generador para el caso de imágenes es normalmente una red de de-convolución y recibe como entrada ruido aleatorio y como su nombre lo indica es el encargado de generar los datos de salida de la arquitectura; y por otro lado, el discriminador también para el caso de imágenes es normalmente una red convolucional que se encarga de supervisar los resultados generados intentando determinar para cada dato de entrada si este pertenece a los datos reales de entrenamiento o si por otra parte es un dato ficticio que ha sido generado de manera artificial. El proceso de entrenamiento de una GAN radica en la interacción y competencia de ambas redes en las cuales se busca para el generador que sus salidas sean gradualmente lo más parecidas a los datos presentes en el set de entrenamiento y para el discriminador que este sea capaz en cada iteración de entrenamiento de diferenciar correctamente entre los datos generados por el generador y los datos reales del set de entrenamiento, es gracias a este proceso de competencia entre ambas redes que es posible la imitación por parte de las GANs de cualquier distribución en los datos de entrada.

3. Método propuesto G-VAE

3.1 Metodología

El desarrollo de modelos de *aprendizaje máquina* es un proceso cíclico e iterativo en el que se necesitan explorar múltiples ideas antes de que el modelo sea implementado de manera productiva, por lo regular este proceso comienza con la preparación de los datos de entrenamiento en donde se debe de explorar y tratar a los datos de modo que el tipo de algoritmo de aprendizaje máquina que queramos usar sea capaz de procesar y usar los datos, una vez que los datos estén listos se debe separar a estos en tres diferentes grupos conocidos como *training*, *test* y *dev set a los cuales se les asignan habitualmente 80%, 15% y 5% de los datos originales.* Esta separación se realiza para que existan tres etapas previas al lanzamiento de un modelo: entrenamiento, prueba y validación.

1. La etapa de entrenamiento como su nombre lo indica sirve para la exploración y desarrollo del modelo base que se entrena usando únicamente el *training set.* En esta etapa se entrena y mejora al modelo iterativamente hasta que el porcentaje de rendimiento sea satisfactorio.

2. Posteriormente se validará al modelo sobre datos que desconoce con el propósito de evaluar la capacidad del modelo de generalizar las características aprendidas durante el proceso de entrenamiento, y con base en los resultados de esta etapa se decidirá si es necesario realizar cambios en la arquitectura del modelo.

3. Finalmente si el modelo pasa satisfactoriamente ambas etapas, en otras palabras, se comporta dentro de los parámetros establecidos tanto en datos conocidos como desconocidos para el modelo, se procede a una última etapa de validación la cual usará los datos contenidos en el dev set.

Por lo que para el desarrollo de este trabajo primeramente se desarrollaran las actividades relacionadas con la generación y carga de los datos de entrenamiento como lo son:

- La construcción del set de datos mediante el desarrollo de scripts de etiquetado y de proyección de imágenes
- La implementación de flujos de datos, que permitan la carga y adecuación de los datos para el entrenamiento del modelo
- Y por último la generación de funciones auxiliares que permitan muestrear de manera aleatoria datos de entrenamiento con el fin de aumentar la generalización del modelo.

Una vez concluidas las actividades anteriormente mencionadas se comenzará con la etapa de entrenamiento del modelo en donde se usará de manera simultánea a los datos de validación para medir continuamente el rendimiento del modelo. Para esta etapa se plantean las siguientes actividades:

- Implementación de un modelo base usando autocodificadores variacionales.
- Definición e implementación de un ciclo de entrenamiento personalizado.
- Evaluación de la capacidad de generalización o rendimiento del modelo base.

Una vez entrenado el modelo base se emplearán como punto de comparación los resultados obtenidos de este, y de modo iterativo se mejorará dicho modelo hasta alcanzar los resultados deseados. Entre las mejoras propuestas para el modelo base se encuentran las siguientes:

- Sustitución del modelo codificador convolucional por un modelo basado en atención(ViT)
- Realización de cambios en el ciclo de entrenamiento que permitan entrenar al VAE y bajo el paradigma de las GANs.
- Implementación de funciones de pérdida auxiliares y cambios en el ciclo de entrenamiento que permitan mejorar la separabilidad de las representaciones internas por clase
- Análisis de las representaciones generadas en términos matemáticos y del efecto de la variación de estas en el espacio latente del modelo

Finalmente una vez terminado el proceso de desarrollo del modelo se plantea el uso y comparación de las representaciones aprendidas por este para una tarea de clasificación de objetos, de acuerdo a las siguientes actividades:

- Entrenamiento de un modelo compacto de clasificación(CNN) en el cual se evaluara el rendimiento de las representaciones del modelo.
- Entrenamiento de un modelo compacto de clasificación(CNN) que sea entrenado sin ningún tipo de guía y actúe como modelo base.
- Entrenamiento de un modelo compacto de clasificación(NN+BOF) como punto de comparación en la clasificación de objetos.
- Comparación directa de los diferentes modelos.

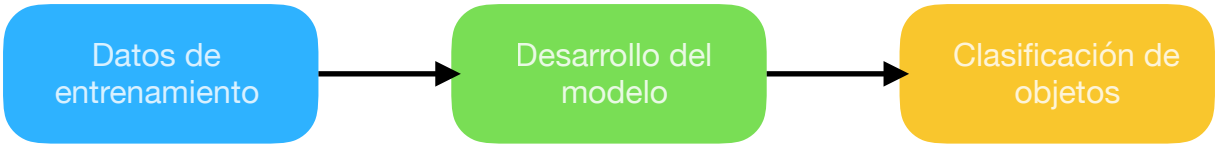


FIGURA 3.0 PRINCIPALES ETAPAS EN EL DESARROLLO DEL MODELO.

3.2 Dataset de entrenamiento.

Para el desarrollo y entrenamiento de este trabajo se hará uso de un dataset sintético propio el cual consiste en diversas proyecciones generadas de un objeto en el plano y está conformado por 50,484 imágenes tridimensionales de veinte categorías diferentes de las cuales la mitad pertenecen a geometrías básicas como lo son la esfera, el cubo, las pirámides triangulares y rectangulares, etc. y la otra mitad a imágenes de objetos simples de manufactura como lo son el engrane, el tornillo, la tuerca, etc.

La creación de este dataset de imágenes surge como una alternativa a la necesidad de capturar físicamente las diversas vistas que un objeto tridimensional puede poseer, mismas que para tareas de reconocimiento facilitan la elaboración de modelos de aprendizaje máquina. Para la creación de este dataset se emplearon técnicas de computación gráfica que reciben veinte modelos tridimensionales correspondientes a las categorías anteriormente descritas y genera múltiples proyecciones en el plano a partir de estos.

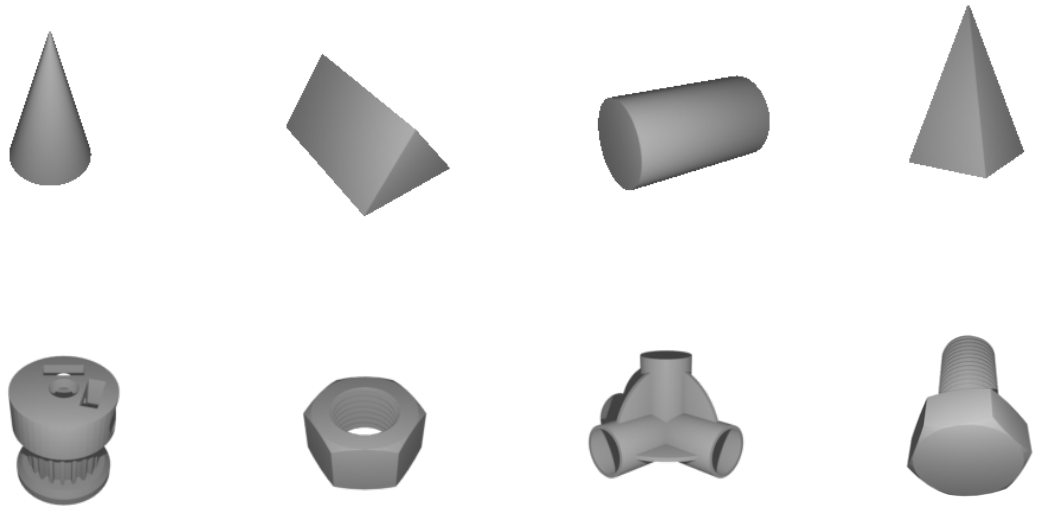


FIGURA 3.1 MUESTRA DE LAS IMÁGENES QUE CONFORMAN EL DATASET.

La figura 3.1 Ilustra diferentes muestras del conjunto de datos las cuales corresponden a diferentes imágenes de objetos tridimensionales en un ángulo especifico.

Proyección de un objeto en el plano

El proceso de proyección de un objeto tridimensional comienza al cargar un modelo tridimensional en una herramienta 3d general conocida como blender, en el cual se deben primeramente normalizar las dimensiones de cada objeto así como su posición con la finalidad de disminuir el sesgo del modelo al momento de entrenar a este, por lo que se busca que todos los objetos se encuentren en el origen y que sus dimensiones sean lo más homogéneas posibles de modo que no exista una desviación o tendencia inicial en los datos que los modelos puedan aprovechar para el reconocimiento de estos.

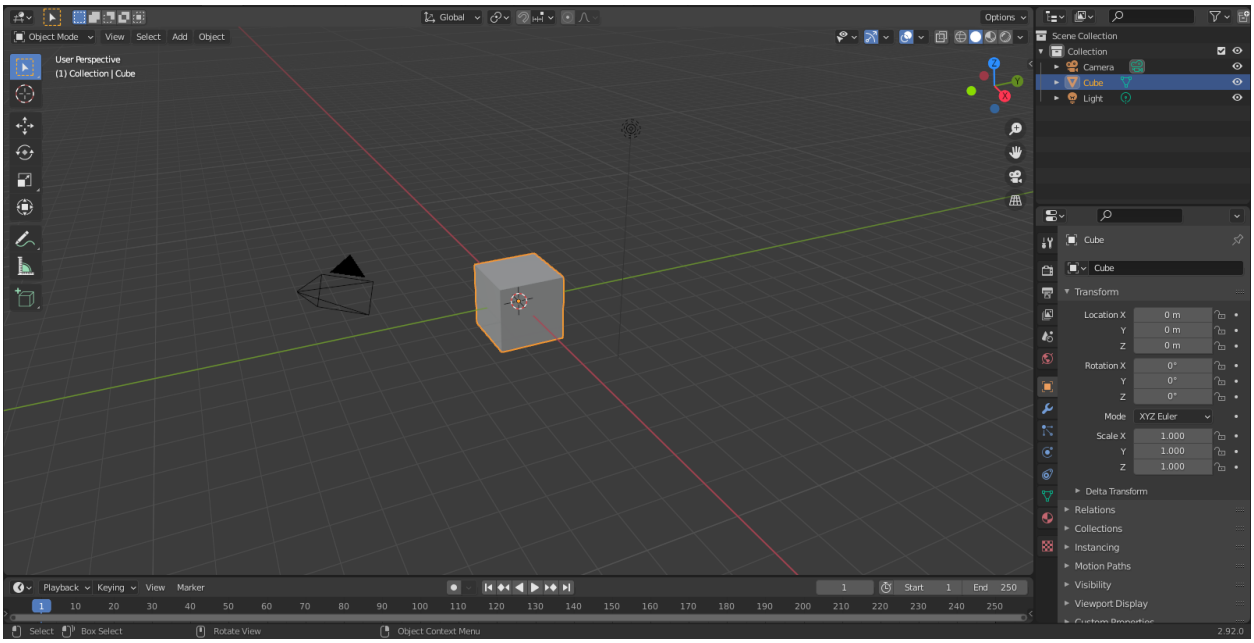


FIGURA 3.2 CARGA DE UN MODELO TRIDIMENSIONAL EN BLENDER.

La figura 3.2 refleja la carga de un modelo en el editor blender, en donde además del modelo en cuestión es necesaria para la generación de las proyecciones la configuración de elementos de iluminación y de captura de imágenes.

Posterior a la normalización de cada objeto se proceden a generar las diferentes imágenes en dos dimensiones del objeto modelando en un principio al objeto y a la cámara del editor en coordenadas esféricas en las cuales como ya se mencionó el objeto se encuentra en el origen y la cámara se coloca a un radio R constante. Además de la cámara se agrega un elemento de iluminación en la misma posición y dirección que la cámara, posteriormente con el uso de un script de python se rota y traslada a la cámara y a la iluminación en los ángulos phi y theta de las coordenadas esféricas en las cuales para el caso de phi varían en el rango de [0º, 180º] y en el caso de theta varían de [0º, 355º] con un desplazamiento de 5º en ambos casos de modo que a

través de estas variaciones se capture a todas las posibles vistas de cada objeto. Finalmente a las imágenes resultantes se les aplica un post procesamiento a nivel imagen para eliminar el canal alfa de las imágenes y reemplazarlo por un fondo color blanco, con la finalidad de aumentar el contraste del objeto hacia el fondo.

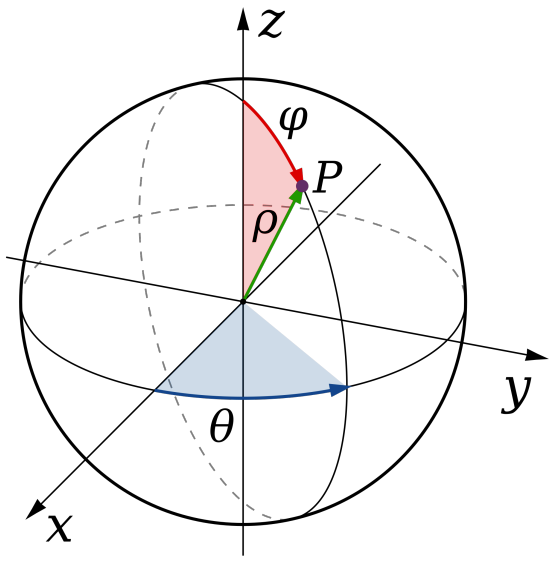


FIGURA 3.3 RELACIÓN ENTRE LAS COORDENADAS ESFÉRICAS Y LAS CARTESIANAS.
(Figura tomada de [22])

División del dataset y procesamiento de los datos

Una vez conformado el set de datos se divide a los elementos existentes en tres conjuntos independientes conocidos como set de entrenamiento set de pruebas y set de validación en donde para este el proceso de desarrollo se emplearan únicamente los datos de las geométrias básicas dividiendo el 80% de estos para entrenamiento y 20% para pruebas y para la validación del modelo se hará uso del resto de los datos de manufactura.

La etapa de procesamiento de los datos comienza con la aplicación de una etapa de preprocesamiento de estos misma que reescala la dimensión de las imágenes de entrada a 56 x 56 pixeles de modo que el modelo pueda ser entrenado de manera eficiente y no se pierdan muchos detalles en las imágenes, además se convierte el formato de la imagen a RGB y normaliza los valores contenidos en cada píxel al rango de [-1, 1] y convierten las imágenes en tensores de entrada con la finalidad de optimizar el entrenamiento del modelo. Posterior al preprocesamiento de los datos se generó una etapa de etiquetado la cual toma como base el nombre de la carpeta en la cual se encuentran los datos para la determinación de su clase.

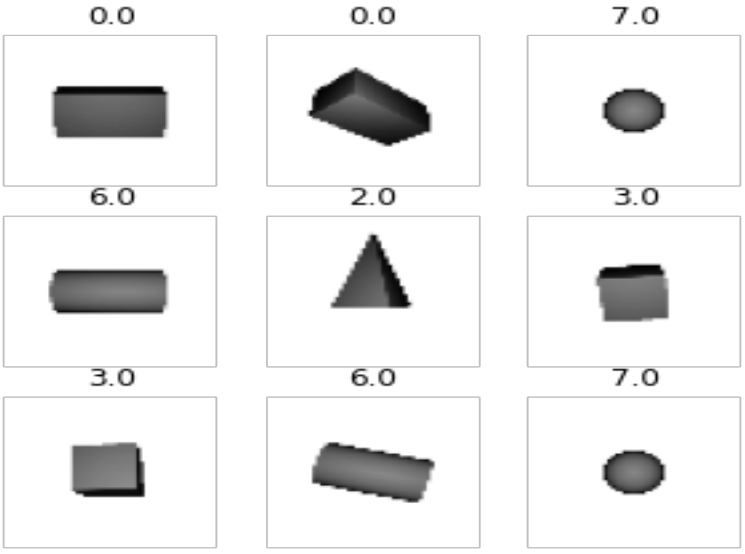


FIGURA 3.4 MUESTRA DE IMÁGENES Y SUS CLASES.

Adicional al preprocesamiento y etiquetado de las imágenes se programo una etapa de muestreo de imágenes la cual tiene como fin generar pares de imágenes tomando como base un conjunto cualquiera de imágenes y dependiendo de la necesidad devolver para cada imagen en dicho conjunto una imagen de salida de la misma categoría o bien una imagen de salida de categoría diferente.

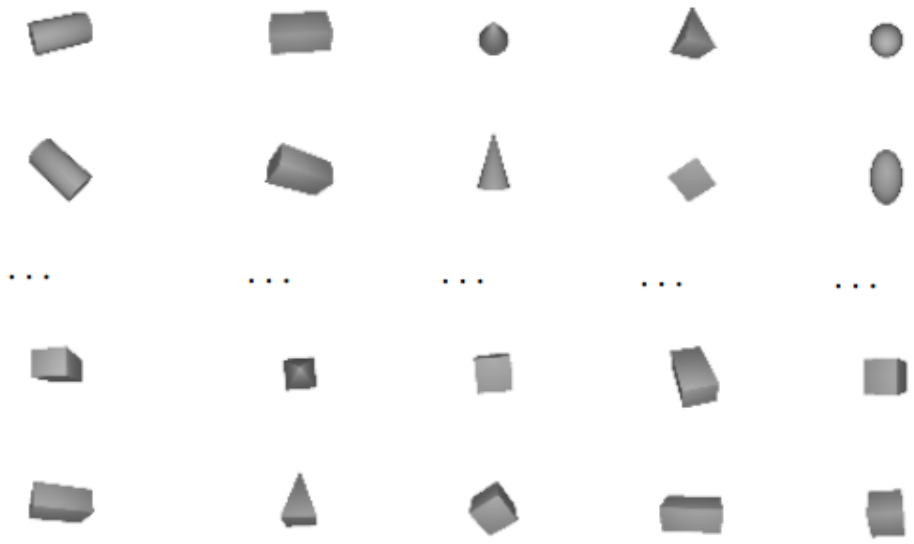


FIGURA 3.5 MUESTREO DE IMÁGENES DE UNA MISMA CLASE PERO DE DIFERENTE PERSPECTIVA.

3.3 Desarrollo del modelo.

Para el desarrollo del modelo se trabajó sobre el framework de aprendizaje profundo conocido como “Tensorflow”, en el cual una vez desarrolladas las funciones de entrada y preprocesamiento de los datos se prosiguió a desarrollar el modelo base al definir y programar la arquitectura de la red neuronal, para la cual se opto por un autocodificador variacional convolucional debido a que este genera en su espacio latente de múltiples distribuciones normales que buscan describir la distribución de las imágenes de entrada, siendo esto un elemento común con el resultado final esperado.

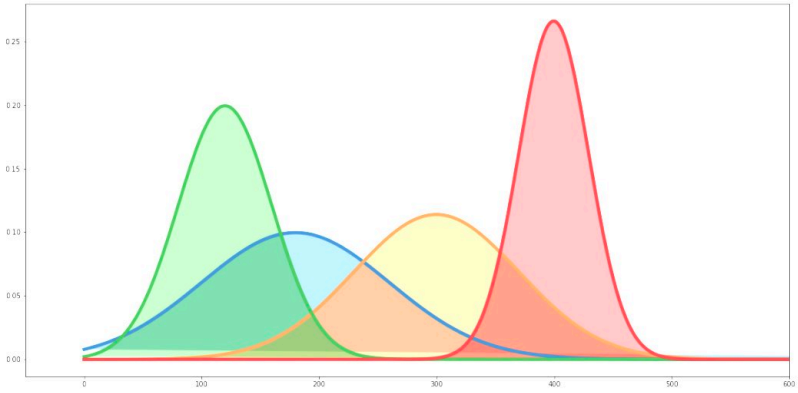


FIGURA 3.6 MÚLTIPLES DISTRIBUCIONES GAUSSIANAS.

El objetivo final de este trabajo es que un modelo de aprendizaje profundo pueda aprender a modelar a cada categoría de imágenes en forma de una serie de distribuciones normales únicas por categoría en el espacio latente del modelo, de modo que al muestrear un elemento desde dichas distribuciones se pueda reconstruir a una imagen cualquiera que pertenezca a la misma categoría.

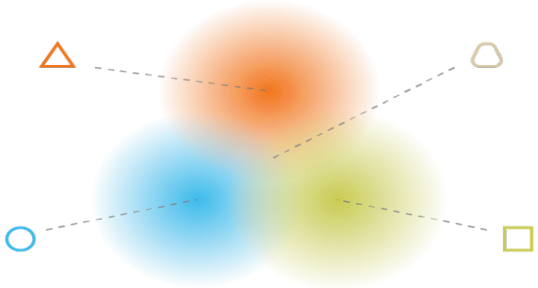


FIGURA 3.7 MUESTREO DESDE LAS DISTRIBUCIONES.

Arquitectura del modelo base.

Como ya se mencionó la arquitectura del modelo base es un autocodificador variacional simplificado el cual tiene como entrada a tensores de imagen de 56x56 pixeles en formato RGB a los cuales se les aplica un codificador convolucional compuesto por 3 capas convolucionales y una capa totalmente conectada para la generación de los parámetros del espacio latente mu, log_var y posterior a esto se emplea un decodificador de-convolucional que hace uso de una capa totalmente conectada y 3 capas de convolución transpuesta para volver a convertir el vector de espacio latente en una imagen. El detalle de los parámetros de la arquitectura puede apreciarse en las figuras 3.8 y 3.9.

```
Model: "Encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 128)       1280

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        73792

 conv2d_2 (Conv2D)           (None, 7, 7, 32)          18464

 flatten (Flatten)           (None, 1568)              0

 dense (Dense)               (None, 64)                100416

=================================================================
Total params: 193,952
Trainable params: 193,952
Non-trainable params: 0
```

FIGURA 3.8 DETALLE DE LOS PARÁMETROS DEL CODIFICADOR DEL MODELO BASE.

```
Model: "Decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 1568)              51744

 reshape (Reshape)           (None, 7, 7, 32)          0

 conv2d_transpose (Conv2DTra  (None, 14, 14, 32)       9248
 nspose)

 conv2d_transpose_1 (Conv2DT  (None, 28, 28, 64)       18496
 ranspose)

 conv2d_transpose_2 (Conv2DT  (None, 56, 56, 128)      73856
 ranspose)

 conv2d_transpose_3 (Conv2DT  (None, 56, 56, 1)        1153
 ranspose)

=================================================================
Total params: 154,497
Trainable params: 154,497
Non-trainable params: 0
_________________________________________________________________
```

FIGURA 3.9 DETALLE DE LOS PARÁMETROS DEL DECODIFICADOR DEL MODELO BASE.

Ciclo de entrenamiento del modelo base.

El modelo base representa la iteración más simple y el primer punto de comparación en el desarrollo de este trabajo y para su entrenamiento se emplea un ciclo de entrenamiento personalizado y el uso de *callbacks auxiliares de* Keras*(ReduceOnPlateau, EarlyStop, etc.) que permiten un entrenamiento óptimo del modelo.*

TABLA 3.1. HIPERPARÁMETROS DE ENTRENAMIENTO.

Hiperparámetros del entrenamiento	
Dimensión del espacio latente	32
Learning rate	0.0002
Batch size	64
Epocas	400
Frecuencia de muestreo de imágenes	0.5

El ciclo de entrenamiento personalizado del modelo base es semejante al ciclo estándar de entrenamiento de un autocodificador variacional, haciendo uso de una función de pérdida ELBO(Evidence lower bound) y un optimizador ADAM(*adaptive moment estimation*) con la diferencia de que para la salida esperada del modelo X´ se usa una imagen aleatoria perteneciente a la misma categoría, lo anterior se hace con la intención de que el modelo genere una codificación única para cualquier imagen de la misma categoría y a partir de ella reconstruya cualquier imagen de dicha categoría.

$$L(\theta, \phi) = E_z \sim q_\phi(z \mid x)[log \frac{p_\theta(x, z)}{q_\phi(z \mid x)}] \leq logp(x) \ldots\ldots \quad (3.1)$$

Ecuación 3.1 función de pérdida ELBO

Finalmente se agregó una función de pérdida TRIPLETE[11] con la intención de monitorear la separabilidad de las distribuciones por clase usando para ello la etiqueta a la que pertenece cada imagen.

$$L(x_i) = \sum_i^N [\,||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + a] \ldots\ldots \quad (3.2)$$

Ecuación 3.2 función de pérdida TRIPLETE

[11] La función de pérdida "*triplet*" compara una entrada base con una entrada positiva y una negativa y busca que la distancia entre la entrada base y la entrada positiva disminuya al mismo tiempo que la distancia entre la entrada base y la entrada negativa incremente.

Evaluación y análisis del modelo base.

Para tener un mayor rango de comparativas disponibles se entrenó al modelo base usando tres diferentes porciones del set de datos, con dos, cinco y diez clases, bajo la idea de que al observar el comportamiento en los diferentes casos de complejidad se puedan generar ideas de mejora sobre este.

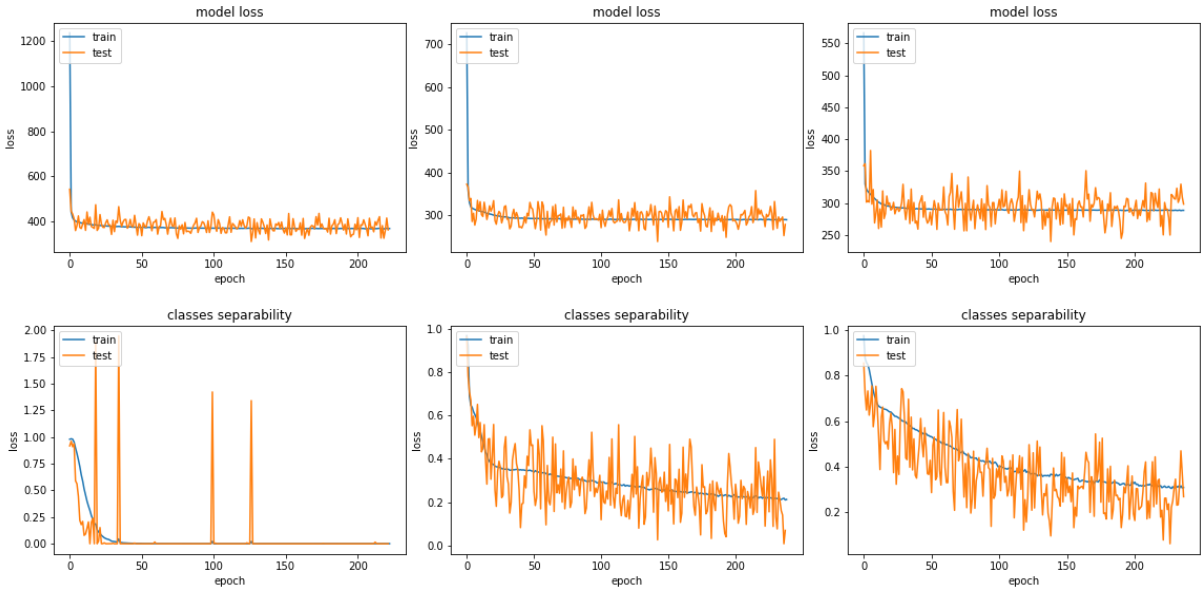


FIGURA 3.10 GRÁFICAS DE LAS FUNCIONES DE PÉRDIDA(SUPERIOR) Y SEPARABILIDAD(INFERIOR) DEL MODELO PARA DIFERENTES ENTRENAMIENTOS DE ESTE SOBRE EL DATASET USANDO DOS CLASES(IZQUIERDA), CINCO CLASES(CENTRO) Y DIEZ CLASES(DERECHA) .

Acorde a lo esperado el caso binario, que representa el extremo más simple, fue en donde mejor se alcanzó la separabilidad en el espacio latente, logrando una separabilidad casi perfecta de 0.005 a las 50 épocas, no obstante la calidad en las representaciones en el espacio latente no fueron satisfactorias ya que el modelo no fue capaz de reconstruir adecuadamente a todas las imágenes que pertenecen a cada categoría, lo cual puede reflejar una falta de datos o una falta de capacidad del modelo. El caso de cinco clases muestra una tendencia semejante al caso anterior en donde la separabilidad del modelo tiende a mejorar con cada época sin embargo a diferencia del caso anterior se mejora más en la capacidad del modelo de generar representaciones en el espacio latente y con ello la capacidad de poder reconstruir mejor a las imágenes que componen el set de pruebas, lo cual demuestra que la cantidad de datos extra ayuda en el proceso de aprendizaje aunque no lo suficiente como para alcanzar los niveles de generalización y separabilidad requeridos. Finalmente el caso en el que se usan a las diez clases resulta semejante al anterior en donde existe una tendencia favorable a la separabilidad y un aprendizaje parcial de las representaciones únicas por clase en el espacio latente.

Mejoras sobre el modelo base

Los primeros experimentos realizados sobre el modelo base demuestran que al modificar el ciclo de entrenamiento de un VAE acorde a lo propuesto se logra una tendencia favorable en cuanto a la separabilidad por clase dentro de las distribuciones del espacio latente, sin embargo existe una falta de capacidad en el modelo base que le permita representar mejor a cada clase dentro del espacio latente, ya que algunas de las reconstrucciones hechas por el modelo carecen de sentido. Para abordar esta carencia en la capacidad del modelo se propone modificar la arquitectura del codificador convolucional a un codificador basado en atención conocido como ViT(*Vision Transformer*), con la esperanza de que este ultimo pueda codificar mejor dentro del espacio latente a cada clase, fuera de esto se decidió conservar el decodificador sin modificaciones para evitar que este pudiera memorizar las salidas esperadas.

```
Model: "Encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 56, 56, 1)]       0

 Encoder (Sequential)        (None, 1, 2, 196, 64)     1018624
=================================================================
| patches_2 (Patches)          (None, None, 16)          0       |
|                                                                |
| patch_encoder_1 (PatchEncoder)  (None, 196, 128)       27264   |
|                                                                |
| encoder_layer_1 (EncoderLayer)  (None, 196, 128)       198272  |
|                                                                |
| encoder_layer_2 (EncoderLayer)  (None, 196, 128)       198272  |
|                                                                |
| encoder_layer_3 (EncoderLayer)  (None, 196, 128)       198272  |
|                                                                |
| encoder_layer_4 (EncoderLayer)  (None, 196, 128)       198272  |
|                                                                |
| encoder_layer_5 (EncoderLayer)  (None, 196, 128)       198272  |
|                                                                |
| reshape (Reshape)            (None, 1, 2, 196, 64)     0       |
|________________________________________________________________|
```

FIGURA 3.11 CODIFICADOR BASADO EN VIT EMPLEADO EN EL MODELO.

Adicional a las modificaciones sobre la arquitectura se decidió aumentar a nivel general la dimensión del espacio latente pasando de usar 32 distribuciones a 64 bajo la misma premisa de buscar una mejor capacidad de representación por parte del modelo, ademas se decidió probar una nueva variación sobre el ciclo de entrenamiento en el que adicionalmente se optimizó la separabilidad en el espacio latente mediante la función de pérdida “triplete”.

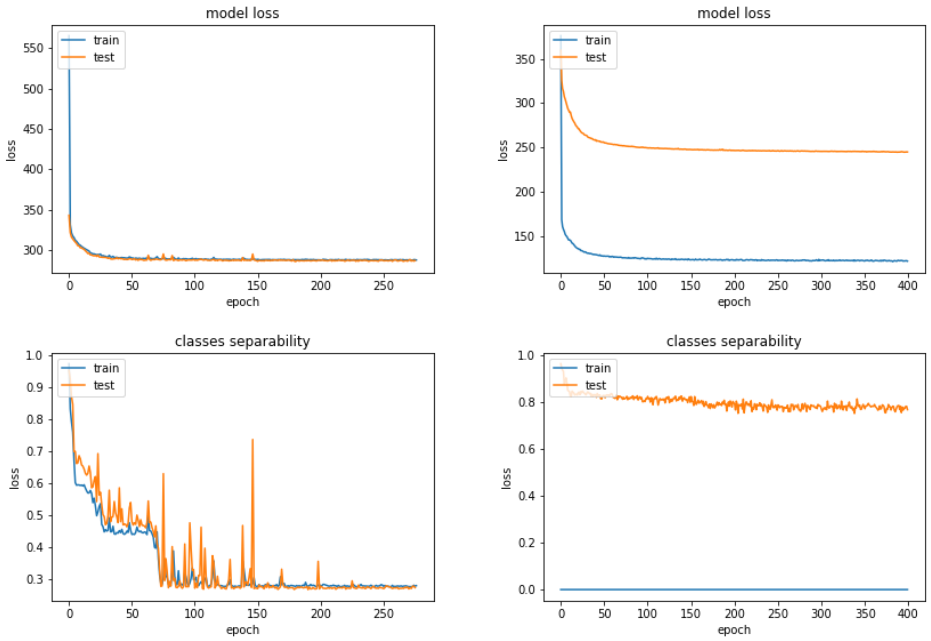


FIGURA 3.12 GRÁFICAS DE LAS FUNCIONES DE PERDIDA(SUPERIOR) Y SEPARABILIDAD(INFERIOR) DEL MODELO PARA LOS ENTRENAMIENTOS DE ESTE SIN OPTIMIZAR LA SEPARABILIDAD(IZQUIERDA) Y OPTIMIZANDO LA SEPARABILIDAD(DERECHA)

Para los nuevos experimentos se mantuvo el uso de los mismos hiperparámetros de entrenamiento y como puede apreciarse en la figura 3.12 existe una mayor estabilidad en los entrenamientos, no obstante, en el caso en el que se entrenó modificando únicamente la arquitectura puede apreciarse que por sí sola esta mejora no es capaz de abstraer adecuadamente las características de los objetos dentro del espacio latente, sin embargo si analizamos el caso en el que adicionalmente se optimiza la separabilidad del modelo, podemos observar que se consigue para el caso del set de entrenamiento que la nueva arquitectura logre alcanzar una separabilidad satisfactoria así como la capacidad de reconstruir adecuadamente a los datos de entrenamiento. Los resultados anteriormente discutidos revelan por una parte la capacidad de la nueva arquitectura para llevar a cabo el objetivo propuesto, además de que se demuestra la necesidad de mejorar el ciclo de entrenamiento ya que el caso en el que se optimiza explícitamente la separabilidad el modelo es incapaz de generalizar lo aprendido hacia el set de prueba lo cual nos habla de un sobre ajuste del modelo sobre el set de entrenamiento.

Dado lo anterior se estudió el efecto que tiene el ciclo de entrenamiento propuesto y se llegó a la conclusión de que al variar la salida esperada del modelo entre la misma imagen de entrada y una imagen diferente pero de la misma clase, es imposible para el modelo saber cómo reconstruir la imagen de salida cuando esta no es igual a la de entrada, hecho que dificulta el entrenamiento del modelo, es por ello que se decidió cambiar la forma en la que se evalúa la capacidad de reconstrucción

del modelo pasando de comparar imagen a imagen, a clasificar la imagen generada haciendo uso de un modelo auxiliar conocido como discriminador.

G-VAE

Como resultado final del proceso iterativo de experimentación y mejora, se llegó al desarrollo de un modelo final denominado “Generalized Variational Autoencoder” o G-VAE, que emplea a un VAE como modelo principal encargado de la representación de los objetos así como de la generación condicional de imágenes, junto con dos modelos auxiliares que permiten el entrenamiento del VAE para el objetivo propuesto los cuales son un discriminador que permite entrenar la capacidad de generación del VAE aumentando la variabilidad en las salidas de este y una red objetivo que suaviza el aprendizaje de la separabilidad por clases dentro del VAE.

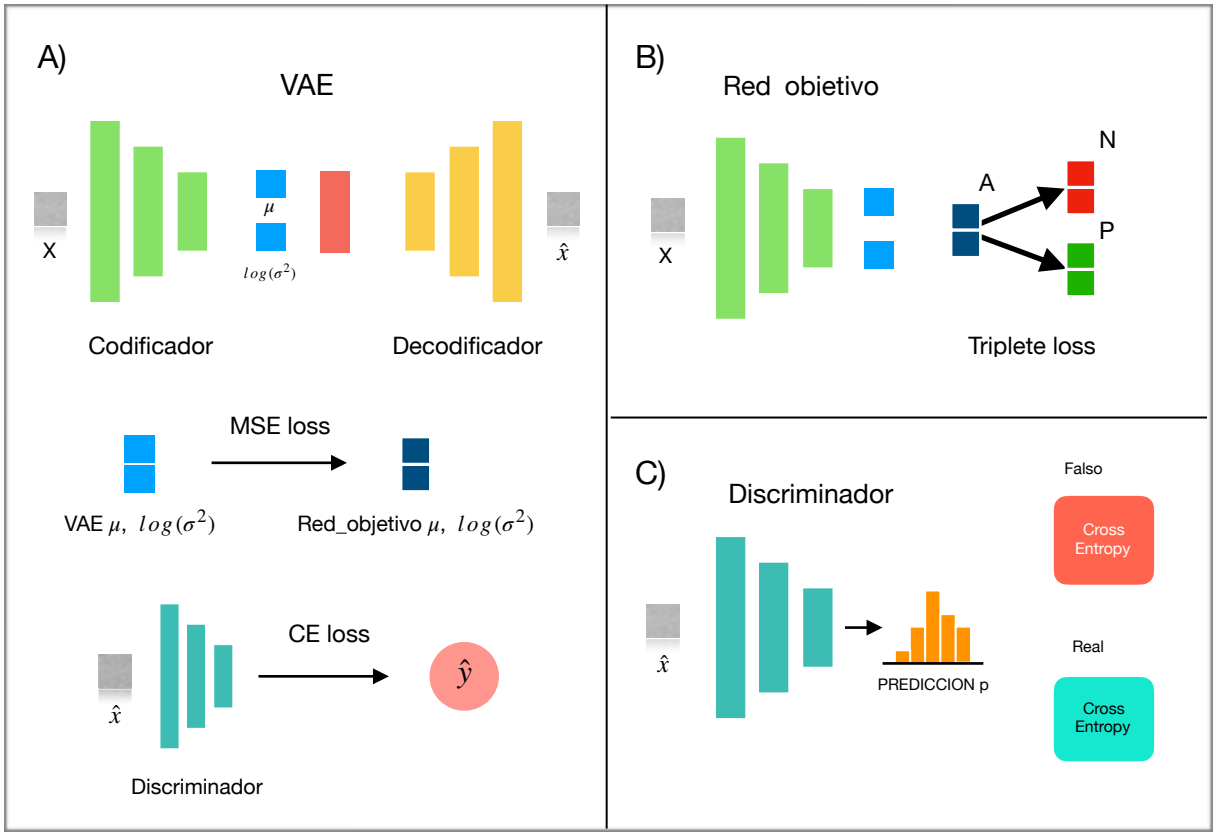


FIGURA 3.13 ELEMENTOS QUE COMPONEN A EL MODELO G-VAE.

En la figura 3.13 pueden apreciarse a los tres diferentes modelos que componen el G-VAE los cuales son; un VAE(A), una red objetivo(B) y un discriminador(C), así como también pueden apreciarse las entradas y salidas de estos modelos, en donde “x”

representa a las imágenes de entrenamiento, "y" representa a las clases de cada imagen, "$\hat{x}$" representa a las imágenes generadas por el decodificador y "$\hat{y}$" representa a las clases inferidas por parte del discriminador para las imágenes generadas.

En cuanto a la configuración individual de cada modelo se refieren para el caso del VAE se conserva el uso de un "*vision transformer"* como codificador, bajo la idea de que los mecanismos de atención pueden resultar más ventajosos al momento de abstraer la información visual, mientras que el decodificador permanece siendo una red deconvolucional a la que se le agregan normalizaciones con el fin de mejorar el proceso de aprendizaje. Por otro lado el discriminador está conformado por una red convolucional a la que igualmente se le agregan normalizaciones para mejorar el entrenamiento. Por último, la red objetivo usa los mismos parámetros que el codificador del VAE y por tanto está conformada por un "*vision transformer ViT*".

TABLA 3.2 DETALLE DE LOS PARÁMETROS DEL VAE.

VAE	Input 56x56x1
Encoder(ViT)	
Patches	Size = 4x4
PatchEncoder	Patches= 196, Dimension= 32
EncoderLayer (x3)	Dropout = 0.1, epsilon = 1e-6
- *Multi head attention*	Dimension = 32, Heads = 8
- FedForward	Dimension = 128, Output Dimension = 32
Reshape	New shape = (-1, 2, 196, 16)
Split	Axis = 1
GlobalAvgPooling1D	Mu = (-1, 16), Logvar = (-1, 16)
Decoder(Deconvolucional)	
Dense	Units = 7*7*16, Activation = relu
BatchNorm	-
Reshape	(7, 7, 16)
Conv2DTranspose	Filters = 16, Kernel = 3, Strides = 2, Padd = same, Activ = relu
BatchNorm	-
Conv2DTranspose	Filters = 32, Kernel = 3, Strides = 2, Padd = same, Activ = relu
BatchNorm	-
Conv2DTranspose	Filters = 64, Kernel = 3, Strides = 2, Padd = same, Activ = relu
BatchNorm	-
Conv2DTranspose	Filters = 1, Kernel = 3, Strides = 1, Padd = same

TABLA 3.3 DETALLE DE LOS PARÁMETROS DE LA RED OBJETIVO.

Red objetivo	Input 56x56x1
Patches	Size = 4x4
PatchEncoder	Patches= 196, Dimension= 32
EncoderLayer (x3)	Dropout = 0.1, epsilon = 1e-6
- *Multi head attention*	Dimension = 32, Heads = 8
- FedForward	Dimension = 128, Output Dimension = 32
Reshape	New shape = (-1, 2, 196, 16)
Split	Axis = 1
GlobalAvgPooling1D	Mu = (-1, 16), Logvar = (-1, 16)

TABLA 3.4 DETALLE DE LOS PARÁMETROS DEL DISCRIMINADOR

Discriminador	Input 56x56x1
Conv2D	Filters = 16, Kernel = 7, Strides = 2, Padd = same
LeakyRelu	-
Dropout	Rate = 0.15
Conv2D	Filters = 32, Kernel = 5, Strides = 2, Padd = same
LeakyRelu	-
BatchNorm	-
Dropout	Rate = 0.15
Conv2D	Filters = 64, Kernel = 3, Strides = 2, Padd = same
LeakyRelu	-
BatchNorm	-
Dropout	Rate = 0.15
Flaten	-
Dense	Units = 10 + 1

Ciclo de entrenamiento del G-VAE

El ciclo de entrenamiento del G-VAE es un proceso mixto en el que por una parte se entrena cada 5 pasos a la red objetivo, para que busque la separabilidad de las distribuciones en el espacio latente, lo cual puede apreciarse en la sección B) de la figura 3.13. Por otra parte se entrenan de manera adversaria al discriminador y al VAE, en donde para el caso del entrenamiento del discriminador se evalúa a las imágenes generadas por el VAE junto con las imágenes originales de entrenamiento con el objetivo de que el discriminador aprenda a clasificar a las imágenes de entrada acorde a su clase y a si estas son producidas de manera artificial por parte del decodificador, tal como puede ilustrarse en la sección C) de la figura 1.13. Finalmente para el entrenamiento del VAE primeramente se procesa a las imágenes de entrada para generar, como resultado, un conjunto de imágenes de salida y los vectores en el espacio latente mu y log_var, posteriormente se usa al discriminador para predecir las distribuciones de las imágenes generadas, mismas que son empleadas para calcular la pérdida de reconstrucción del VAE, además de esto se emplea a la red objetivo haciendo uso de las mismas entradas del VAE para predecir los valores separables de mu y log_var que son empleados para guiar la separabilidad general de sus equivalentes dentro del VAE, lo cual es representado por las funciones de pérdida "*MSE_loss*" y "*CE_loss*" en la figura 3.14 y dentro de la sección A) de la figura 3.13.

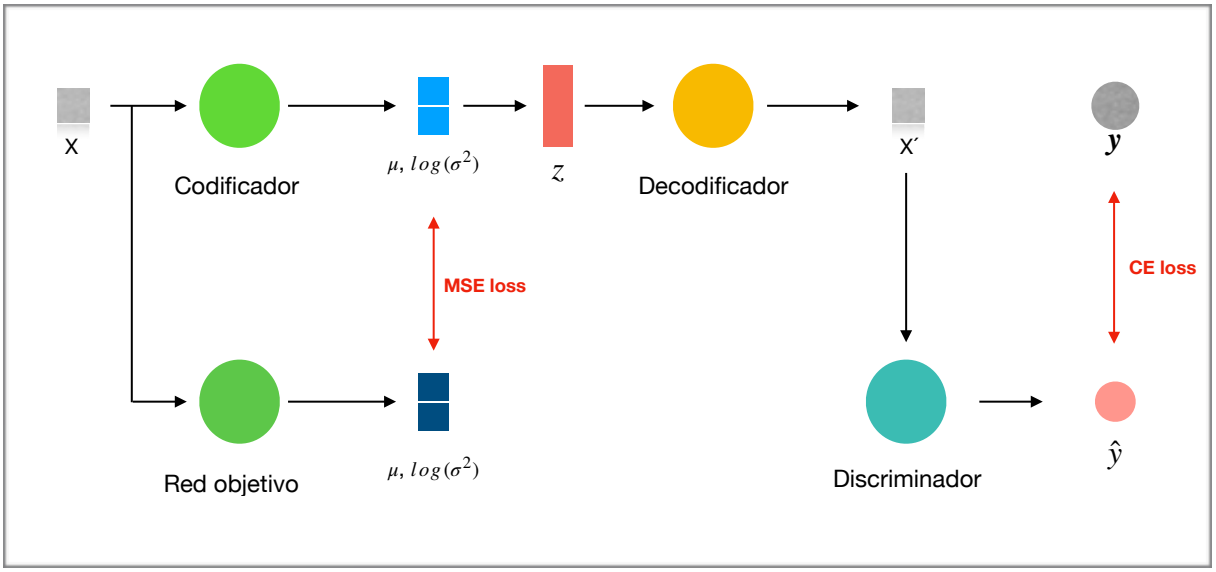


FIGURA 3.14 CICLO DE ENTRENAMIENTO DE EL AUTOCODIFICADOR VARIACIONAL.

3.4 Clasificación de objetos

Como parte de una etapa posterior al entrenamiento se decidió emplear a las representaciones internas aprendidas por el G-VAE para clasificar a las diferentes imágenes de objetos de manufactura, emulando un detector de características en donde se evaluara su desempeño, en cuanto a precisión, número de parámetros y tiempo de ejecución, sin embargo a diferencia del codificador ViT el modelo en cuestión será una red convolucional de pocos parámetros la cual buscará emular un detector de características, entrenándose primeramente para predecir las representaciones aprendidas por el G-VAE y posteriormente para clasificar a los objetos mediante el uso de estas representaciones. Es importante mencionar que como puntos de comparación se evaluará primeramente a la misma red convolucional compacta pero sin ningún tipo de preentrenamiento o guía, así mismo se usará clasificador optimizado basado en características el cual usa como extractor de características una función BOF y como clasificador a una red neuronal.

TABLA 3.5 ARQUITECTURA PROPUESTA PARA EL CLASIFICADOR CNN Y BOF+NN.

CNN		**CNN+Emb**		**BOF+NN**	
Input	56,56,1	Input	56,56,1	Input	56,56,1
Conv2D	Filters = 128, Kernel = 3, Strides = 2, Padd = same	Conv2D	Filters = 64, Kernel = 3, Strides = 2, Padd = same	BOF	Dim = 128
Relu	-	Elu	-	Dense	Units = 64
Conv2D	Filters = 32, Kernel = 3, Strides = 2, Padd = same	Conv2D	Filters = 32, Kernel = 3, Strides = 2, Padd = same	Dense	Units = 32
Relu	-	Elu	-	Dense	Units = 10
Conv2D	Filters = 16, Kernel = 3, Strides = 2, Padd = same	Conv2D	Filters = 16, Kernel = 3, Strides = 2, Padd = same		
Relu	-	Elu	-		
Conv2D	Filters = 8, Kernel = 3, Strides = 2, Padd = same	Conv2D	Filters = 8, Kernel = 3, Strides = 2, Padd = same		
Relu	-	Elu	-		
Dense	Units = 16	Dense	Units = 64		
Dense	Units = 10	Dense	Units = 16		
		Dense	Units = 10		
Parametros	**46194**		**34274**		**33834**

La tabla 3.5 muestra el detalle de las dos aproximaciones propuestas para el proceso de clasificación de imágenes en donde el clasificador CNN es una red compacta estándar mientras que por otro lado el clasificador BOF+NN es un modelo híbrido.

El clasificador BOF+NN fue programado mediante el uso de la librería de procesamiento de imágenes conocida como "*opencv*" en lo referente a la extracción de características y mediante el framework de "*tensorflow*" en lo referente a la clasificación de ellas. En donde el primer paso, tanto para el entrenamiento como para el uso del modelo, es procesar a cada imagen de entrada a través del extractor BOF que primeramente binariza a las imágenes y a partir de ellas extrae y normaliza la distancia que hay entre el centro de las figuras y los diferentes puntos de interés que componen su perímetro, los cuales para esta aproximación suman 180, finalmente una vez extraídos estos 180 valores o representaciones por cada imagen se procesa a estos mediante una red compacta densamente conectada, la cual genera la predicción final del modelo.

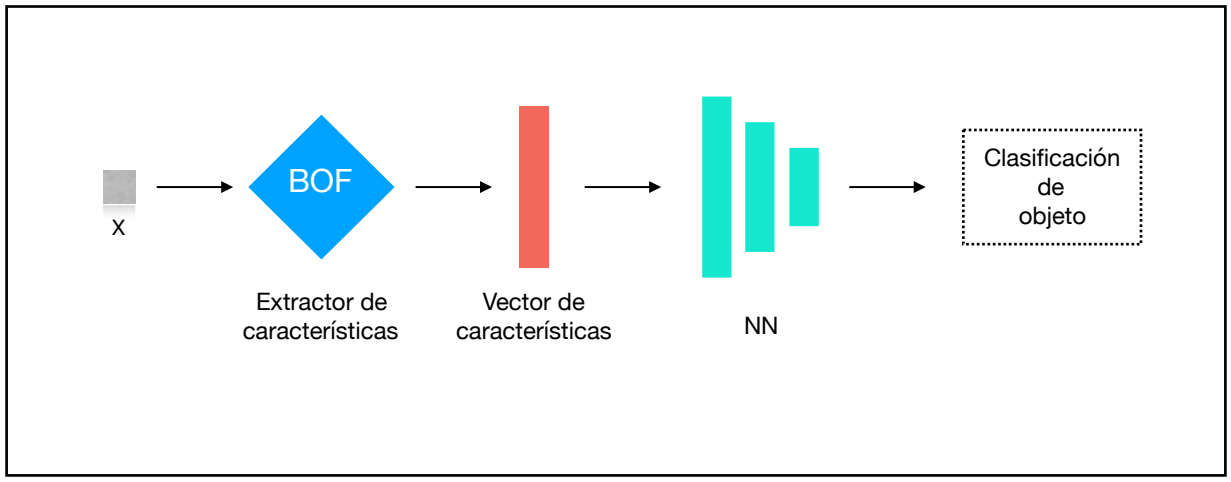


FIGURA 3.15 PROCESO DE INFERENCIA MEDIANTE EL USO DEL MODELO BOF+NN

Finalmente en cuanto al ciclo de entrenamiento de las redes se refiere, se trata de un ciclo de clasificación multiclase estándar que entrena a las redes por 200 épocas y hace uso de una función de pérdida "*Cross-Entropy*", un optimizador "Adam(lr=1e-3)" y "*callbacks*" que evitan el sobreajuste, deteniendo el entrenamiento y minimizando la taza de aprendizaje(lr), en donde para el caso particular del modelo CNN que usa a las representaciones del G-VAE se entrena por separado la parte de la red encargada de reproducir las mismas representaciones y una vez entrenada esta se congelan sus parámetros para entrenar únicamente los parámetros de la capa de clasificación.

4. Resultados

4.1 Resultados del entrenamiento del G-VAE

La aproximación empleada para resolver el problema planteado en este trabajo puede resumirse como una aproximación mixta de colaboración y competencia que es llevada a cabo entre diferentes redes auxiliares y una red principal y gracias al apoyo de ambas redes la red principal es capaz para la tarea en cuestión de aprender a reconstruir adecuadamente elementos que pertenezcan a la misma clase que al de la imagen de entrada del modelo así como de alcanzar la separabilidad general por clase de objeto hasta un valor de 0.0501 para el caso de figuras geométricas y de 0.0138 para el caso de figuras de manufactura además de poder alcanzar una separabilidad de 0.0278 y de 0.0055 respectivamente para el caso en el que se optimiza la separabilidad de las clases usando únicamente a la media de las distribuciones, siendo 0.0 el caso de una separabilidad perfecta.

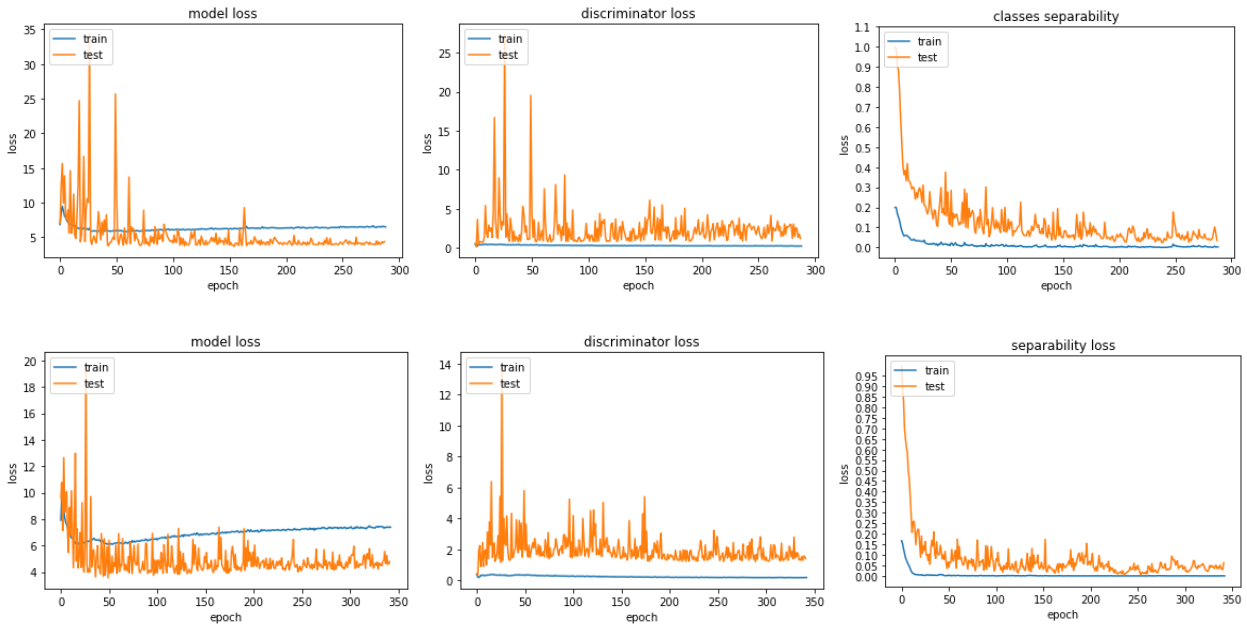


FIGURA 4.1 GRÁFICAS DEL ERROR TOTAL DEL MODELO (IZQUIERDA), DEL ERROR POR PARTE DEL DISCRIMINADOR(CENTRO) Y DEL ERROR DE SEPARABILIDAD DEL MODELO(DERECHA) PARA LOS ENTRENAMIENTOS EN EL QUE SOLO SE BUSCA LA SEPARABILIDAD DE LA MEDIA PARA EL DATASET DE FIGURAS GEOMÉTRICAS(SUPERIOR) Y EL DATASET DE OBJETOS DE MANUFACTURA(INFERIOR).

Las gráficas de la figura 4.1 son el resultado del entrenamiento del G-VAE para el caso en el que se entrena la separabilidad en la media, sin regularizar a la varianza, en donde se comprobó que el modelo propuesto alcanzó satisfactoriamente el nivel de separabilidad requerido para esta etapa en la cual se observó que el aprendizaje para

el caso en el que se entrena y valida a los datos correspondientes a los objetos de manufactura es llevado a cabo con mayor facilidad que el de figuras geométricas por parte del modelo posiblemente debido a su baja relación geométrica en comparación de las geométricas simples, que para ciertas vistas de objetos diferentes se genera una proyección muy semejante para ambos objetos como lo es por ejemplo el caso de una esfera y un elipsoide desde una vista superior en la que ambas figuras parecen ser círculos. Este entrenamiento acotado se realizó primeramente por la necesidad de determinar si la solución propuesta era capaz de resolver ese subcaso de mayor facilidad, además de que el mismo vector de la media puede ser tomado como vector de características para una etapa de clasificación posterior.

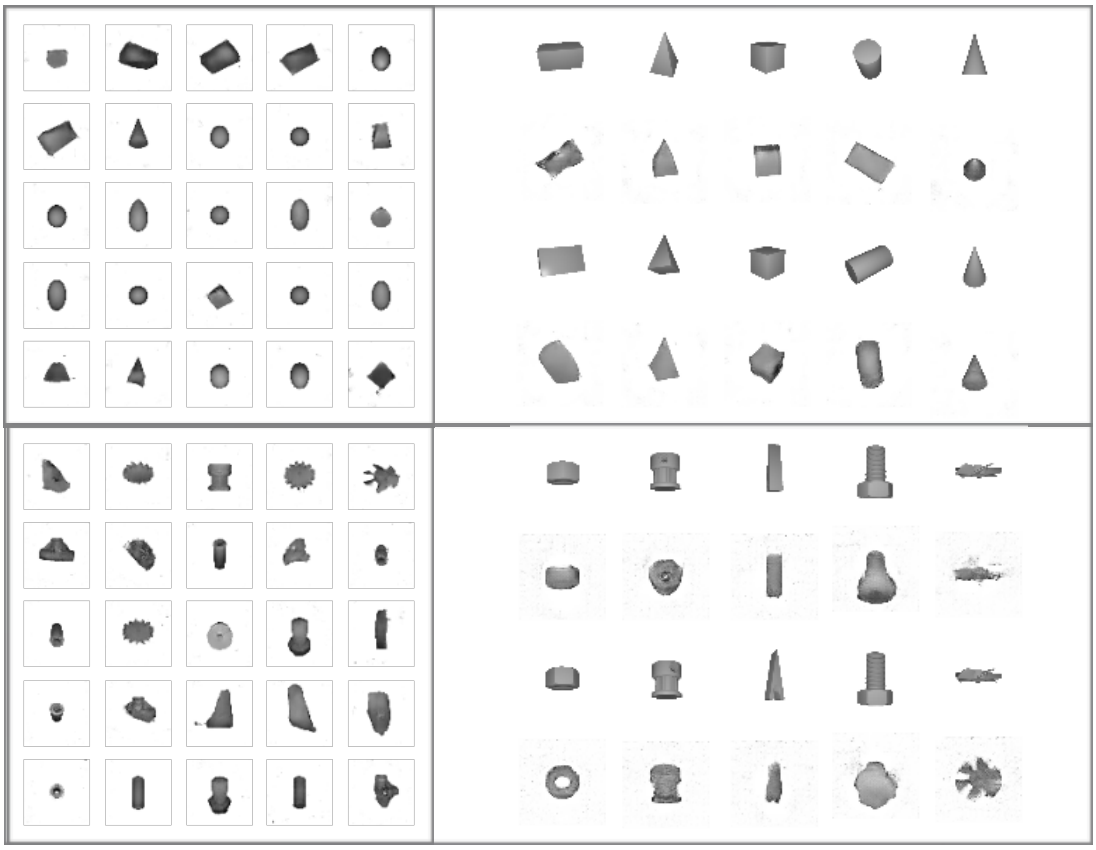


FIGURA 4.2 MUESTRA DE LAS IMÁGENES GENERADAS POR EL G-VAE(IZQUIERDA) ASÍ COMO LA GENERACIÓN CONDICIONAL(DERECHA).

En cuanto a la capacidad generativa del modelo se refiere se observaron durante todo el entrenamiento reconstrucciones mayoritariamente congruentes por parte de este, exceptuando los momentos de ajuste en la separabilidad de la media en donde dichos cambios degradaban momentáneamente la generación de las imágenes, así mismo se observó el resultado de la generación condicional de imágenes en la cual se procesó a diferentes imágenes de muestra de una clase determinada para así generar salidas condicionales en las que su correspondencia con respecto a sus clases de entrada fue satisfactoria para el 96% de las imágenes en el set de validación.

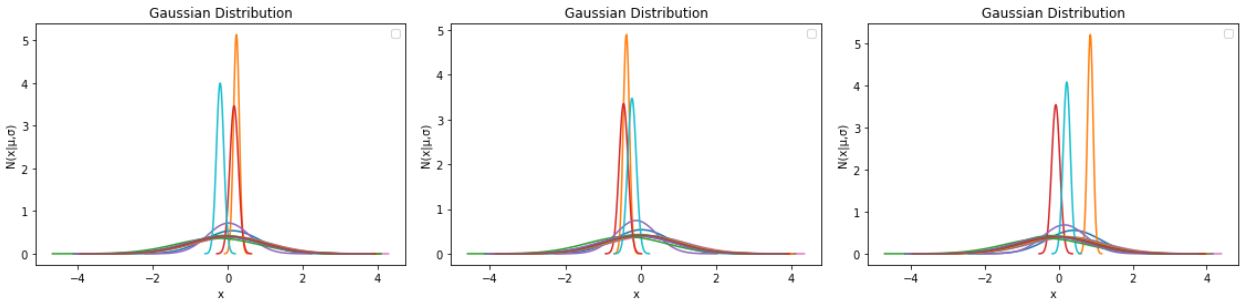


FIGURA 4.3 DISTRIBUCIONES DE DIFERENTES CLASES DE OBJETOS DENTRO DEL ESPACIO LATENTE.

Otro punto clave a evaluar en los entrenamientos fue el aspecto de las distribuciones aprendidas dentro del espacio latente por parte del modelo y para su visualización y comparación primeramente se promediaron todos los valores de la media y varianza dentro del espacio latente dando como resultado la gráfica de distribuciones a la izquierda dentro de la figura 4.3, posteriormente se procedió a evaluar mediante el mismo proceso a las representaciones generadas por la clase 2(engrane) y la clase 8(aspas) de objetos mediante el uso de una máscara en los datos de evaluación que permitiera procesar únicamente a los datos pertenecientes a cada clase, dando como resultado las gráficas de distribuciones del centro(engrane) y de la derecha(aspas) de la figura 4.3, mediante el uso de las tres gráficas se puede concluir que el modelo es capaz de alcanzar una separabilidad exitosa en cuanto a las distribuciones internas se refiere ya que no existe una correlación directa entre los datos analizados en cada caso.

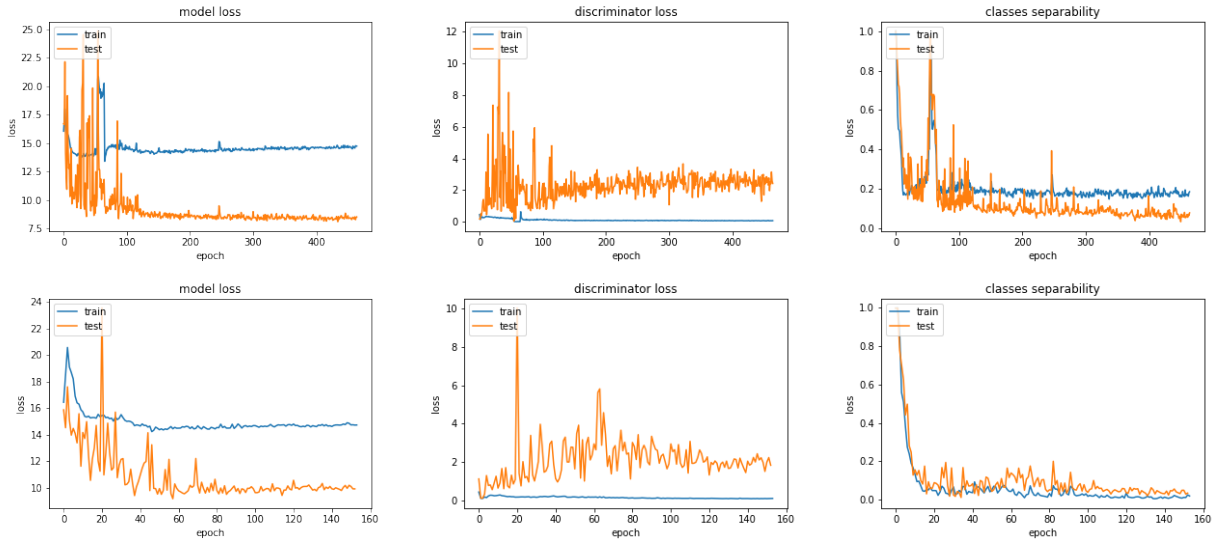


FIGURA 4.4 GRÁFICAS DEL ERROR TOTAL DEL MODELO (IZQUIERDA), DEL ERROR POR PARTE DEL DISCRIMINADOR(CENTRO) Y DEL ERROR DE SEPARABILIDAD DEL MODELO(DERECHA) PARA LOS ENTRENAMIENTOS EN LOS QUE SE BUSCA LA SEPARABILIDAD DE TODOS LOS PARÁMETROS DE LAS DISTRIBUCIONES PARA EL DATASET DE FIGURAS GEOMÉTRICAS(SUPERIOR) Y EL DATASET DE OBJETOS DE MANUFACTURA(INFERIOR).

Las gráficas de la figura 4.4 son el resultado del entrenamiento del G-VAE para el caso en el que se entrenó a todos los parámetros de las distribuciones dentro del espacio latente, dicho entrenamiento resultó ser mucho más complejo que el caso anterior siendo incluso necesario guiar el aprendizaje de las distribuciones mediante el uso de las etiquetas de las clases a las que pertenecen las imágenes de entrenamiento, además de hacer uso del discriminador ya entrenado para inferir dicha etiqueta en el proceso de evaluación. Al igual que en el caso anterior se observó que el entrenamiento del modelo para el caso de las figuras geométricas resultó ser más difícil de realizar lo cual de igual forma puede estar relacionado con la gran similitud de algunas imágenes de diferentes clases.

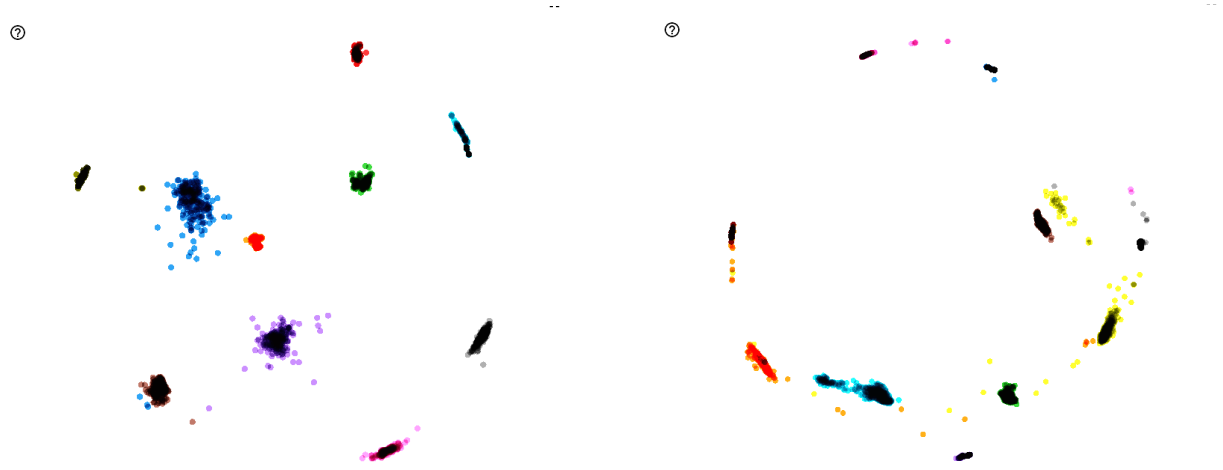


FIGURA 4.5 VISUALIZACIÓN DE LOS PARÁMETROS DEL ESPACIO LATENTE DEL MODELO HACIENDO USO DE UNA REDUCCIÓN PCA PARA EL CASO EN EL QUE SE REGULARIZA LA SEPARABILIDAD DE LOS PARÁMETROS DE MEDIA(IZQUIERDA) Y EN LA CUAL SE REGULARIZA A TODOS LOS PARÁMETROS DE LAS DISTRIBUCIONES(DERECHA) .

Como último punto de evaluación se analizaron en forma de "*embedings*" a todos los parámetros en el espacio latente resultantes del procesamiento de los datos de validación por parte del modelo, en los que se observó que a pesar de contar con niveles de separabilidad semejantes el caso en el que se buscó la separabilidad de la media cuenta con un grado de agrupamiento y proximidad mayor que el caso en el que se buscó la separabilidad de todos los parámetros de las distribuciones.

4.2 Resultados en la clasificación de imágenes

Los resultados obtenidos en el desarrollo del G-VAE demuestran su potencial uso en tareas de clasificación de imágenes debido a la separabilidad de sus representaciones mismas que se emplearon como parte de una tarea aplicativa con la finalidad de clasificar adecuadamente a las imágenes de objetos de manufactura, pertenecientes al set de validación. Además de que de manera paralela se desarrollaron y evaluaron para el mismo conjunto de datos el rendimiento de un modelo base “CNN” y un modelo de extracción de características “BOF+NN”.

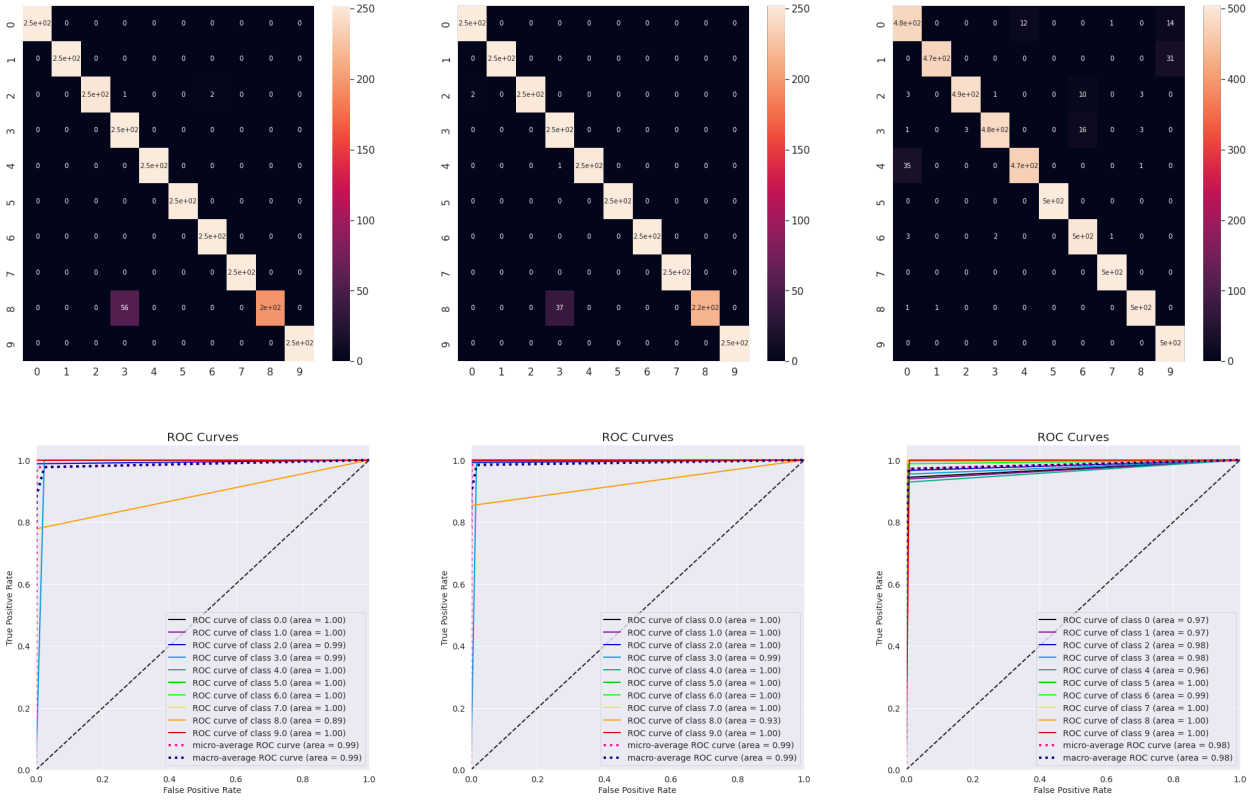


FIGURA 4.6 RESULTADOS DEL PROCESO DE CLASIFICACIÓN DE IMÁGENES DEL MODELO CNN(IZQUIERDA), DEL MODELO QUE USA A LAS REPRESENTACIONES APRENDIDAS(CENTRO) Y DEL MODELO BOF+NN(DERECHA).

Como parte del proceso de evaluación de los resultados obtenidos con el entrenamiento de los modelos se graficó el rendimiento de cada uno de estos mediante el uso de matrices de confusión y de curvas ”ROC”, las cuales facilitan el análisis visual del desempeño general de los modelos y los casos en los que estos tienen mayor dificultad, mostrando para el caso de los modelos basados en convolución un comportamiento semejante entre si en el cual los modelos clasifican con mucha precisión, mayor al 99%, 9 de las 10 clases de objetos siendo la clase 9 el caso de mayor dificultad, mientras que por otro lado el clasificador basado en BOF

demostró un desempeño muy bueno en todas las clases, mayor al 96%, pero sólo se obtuvo un rendimiento mayor al 99% en 4 de las clases, tal como puede apreciarse en la figura 4.6.

TABLA 4.1 COMPARACIÓN DE LOS MODELOS DE CLASIFICACIÓN PROPUESTOS.

Modelo	Precision	Perdida	Parametros	Tiempo de ejecución
CNN	0.9769	0.0721	46,194	3.5 ms / batch(1-32) imagenes
CNN+Embed	0.9843	0.0511	34274	3.5 ms / batch(1-32) imagenes
BOF+NN	0.9720	0.0808	33,834	3.2 ms / batch(1-64) + 1.5 ms / 1 img BOF

Para finalizar el análisis de los resultados de clasificación se comparó en términos de tiempo de ejecución, precisión general, pérdida y cantidad de parámetros a cada uno de los modelos dando como resultado los datos presentes en la tabla 4.1 en la cual se aprecia un rendimiento muy semejante en todos los rubros para todos los modelos aquí propuesto, resaltando la gran versatilidad de los embedings y las características BOF mediante los cuales se pueden alcanzar resultados óptimos en arquitecturas neuronales compactas.

5. Conclusiones

El hecho de basarse en autocodificadores variacionales para buscar representar a los objetos en forma de distribuciones gaussianas únicas dentro del espacio latente resultó ser un buen punto de partida ya que al monitorear la separabilidad natural que existe en el proceso de entrenamiento de estos modelos se descubrió que esta tendía a mejorar conforme se avanzaba el entrenamiento del modelo, sin embargo dicha separabilidad se estancaba al llegar a cierto umbral determinado. Posteriormente se observó que la primer propuesta de entrenamiento permitió aumentar aun más el nivel de separabilidad en el modelo pero introdujo nuevos problemas de reconstrucción condicional. También se demostró que no era posible mediante una regularización directa alcanzar una separabilidad adecuada en el modelo ya que el efecto de esto era un sobreajuste de la separabilidad en las distribuciones del modelo y carecía de capacidad de generalización hacia datos no vistos.

Otro punto importante a mencionar es el hecho de que se logró modificar el proceso de generación de las GANs para que estas pudieran aprender a reconstruir condicionalmente imágenes de múltiples clases de procedencia, lo cual resultó ser consistente incluso en circunstancias de baja separabilidad. También se destaca el hecho de haber alcanzado la separabilidad planteada dentro del espacio latente del modelo para los dos casos propuestos de separabilidad (separabilidad de las distribuciones únicamente enfocadas a la media y separabilidad de las distribuciones usando tanto a la media como a la varianza), en donde surge la duda de la relevancia de un caso sobre otro ya que aunque el caso de la separabilidad de la media se planteó como un subcaso de la separabilidad general, es igualmente válido el considerar y enfocarse en los resultados alcanzados en éstas siempre y cuando se cumpla que los valores aprendidos de las medias de las distribuciones se encuentran lo suficientemente separados entre si y que los valores de varianza de estas tiendan a cero, ya que de ser el caso se asegura que aunque pueden existir distribuciones con valores semejantes o idénticos de varianza, la representación de las diferentes clases de imágenes en el espacio latente pertenecen principalmente a una única distribución.

Finalmente en cuanto a la utilidad y calidad de las representaciones internas para la clasificación de objetos se comprobó que estas permiten con éxito alcanzar niveles de rendimiento similares a las obtenidas por un extractor de características como lo es el extractor BOF empleando un número de parámetros similares.

La tarea de representar a los objetos en forma de distribuciones gaussianas dentro del espacio latente involucró el diseño de mecanismos de entrenamiento ingeniosos con el fin de alcanzar una separabilidad satisfactoria de las representaciones del modelo así como de dotar de un sentido congruente en el dominio espacial a las diferentes distribuciones aprendidas, sin embargo dada la capacidad generativa de las GANs tal vez sea conveniente para futuros desarrollos el

intentar aprender estas distribuciones eliminando la etapa de decodificación/ reconstrucción y sustituyéndola por una etapa de clasificación convencional que se base en el muestreo proveniente de las distribuciones del espacio latente con el fin de simplificar el entrenamiento de estas, además de intentar buscar la separabilidad de las distribuciones de manera indirecta haciendo uso de las muestras provenientes de cada distribución y no de los parámetros que componen a estas.

Bibliografía

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L.,, Stein, C. (2001). *Introduction to Algorithms*. The MIT Press. ISBN: 0262032937

[2] Goodfellow, I., Bengio, Y.,, Courville, A. (2016). *Deep Learning*. MIT Press.

[3] Szeliski, R. (2021). Computer Vision: Algorithms and Applications 2nd Edition. *Springer*.

[4] Lomas-Barrie, V., Pena-Cabrera, M., Lopez-Juarez, I., & Navarro-Gonzalez, J. L. (2021). Fuzzy artmap-based fast object recognition for robots using FPGA. *Electronics (Switzerland)*, *10*(3). https://doi.org/10.3390/electronics10030361

[5] Sanzhar Askaruly (2022). Lab 5 - Digital Signal Processing. Image Enhancement. (https://www.mathworks.com/matlabcentral/fileexchange/54535-lab-5-digital-signal-processing-image-enhancement), MATLAB Central File Exchange. Retrieved May 20, 2022.

[6] Olmos, I. (2022). *Operaciones Orientadas a Punto* [Diapositivas en pdf]. (https://www.cs.buap.mx/~iolmos/pdi/Sesion4_Operaciones_Punto.pdf), BUAP México, Recuperado 20 Mayo, 2022.

[7] MathWorks, What Is Image Recognition?. (https://la.mathworks.com/discovery/image-recognition-matlab.html), *MathWorks[Pagina web]*, Recuperado 20 Mayo, 2022.

[8] mc.ai , *101 : Building Blocks*, (https://mc.ai/101-building-blocks/), *mc.ai [Figura]*, Recuperado Agosto, 2019.

[9] Hunter Heidenreich(2018), The Future with Reinforcement Learning, Part 2: Comparisons and Applications[Figura], (https://miro.medium.com/max/800/0*F23wacf0xFsV4I40.jpg), *chatbotsmagazine.com*, Recuperado 20 Mayo, 2022.

[10] Aidan, Wilson, (2019), *A Brief Introduction to Supervised Learning* [Figura], (https://miro.medium.com/max/960/1*pQOmuWjgevfagaHqGh6HXA.png), *towardsdatascience.com*, Recuperado 20 Mayo, 2022.

[11] Geeksforgeeks, *Different Types of Clustering Algorithm*. (https://knowm.org/wp-content/uploads/KMeans-density-data1.png), *geeksforgeeks.org* [Figura], Recuperado 20 Mayo, 2022

[12] Sibanjan Das, (2017), *Reinforcement Learning for the Enterprise*, (https://dzone.com/storage/temp/6976061-screen-shot-2017-10-20-at-22200-pm.png), *dzone.com* [Figura], Recuperado 20 Mayo, 2022.

[13] Richard Hackathorn, *(2018), How Managers Should Prepare for Deep Learning: New Values*, (https://towardsdatascience.com/how-managers-should-prepare-for-deep-learning-new-values-f29a98b70bd8), *towardsdatascience.com* [Figura], Recuperado 20 Mayo, 2022

[14] Yatheendra Prava, (2021), Project Idea | Cat vs Dog Image Classifier using CNN implemented using Keras. (https://www.geeksforgeeks.org/project-idea-cat-vs-dog-image-classifier-using-cnn-implemented-using-keras/), *geeksforgeeks.org* [Figura], Recuperado 20 Mayo, 2022

[15] Rina Buoy, (2019), Convolutional Neural Network — An Informal Introduction (Part-1). (https://medium.com/analytics-vidhya/convolutional-neural-network-an-informal-intro-part-1-db9fca86a750), *Analytics Vidhya* [Figura], Recuperado 20 Mayo, 2022

[16] Menyukai in, (2018), Student Notes: Convolutional Neural Networks, (https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/), *indomi.com* [Figura], Recuperado 20 Mayo, 2022

[17] Leonardo Araujo, (2019), Pooling Layer, (https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/pooling_layer), artificial-inteligence [Figura], Recuperado 20 Mayo, 2022

[18] Jeremy Jordan, (2017), Convolutional neural networks, (https://www.jeremyjordan.me/convolutional-neural-networks/), medium.com, Recuperado 20 Mayo, 2022.

[19] Chen, X., Hsieh, C. J., & Gong, B. (2021). When vision transformers outperform ResNets without pre-training or strong data augmentations. *arXiv preprint arXiv:2106.01548*.
[20] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). An image is worth 16*16 words: transformers for image recognition at scale. *Advances in Neural Information Processing Systems*, *2017-Decem*.
[21] Lil'Log, (2018), From Autoencoder to Beta-VAE, (https://lilianweng.github.io/posts/2018-08-12-vae/), lilianweng.github.io, *Recuperado 20 Mayo, 2022*.
[21] Skymind.ai, generative-adversarial-network-gan, (https://skymind.ai/fsph/generative-adversarial-network-gan), *Recuperado 20 Mayo, 2022*.
[22]wikimedia, Spherical Coordinates, (https://commons.wikimedia.org/wiki/File:Spherical_Coordinates_(Colatitude,_Longitude).svg), *Recuperado 20 Mayo, 2022*.
[23] Evan Czako (2022). Gaussian filtering for images. (https://github.com/EvanCzako/gaussian-image-filtering), GitHub [Figura], Recuperado Mayo 2022.
*[24]Cookie_Studio (2022). Portrait of cute caucasian. (*https://www.freepik.com/free-photo/portrait-cute-caucasian*), Freepik [Figura], Recuperado Mayo 2022.*
[25] Martinez, Pablo & Al-Hussein, Mohamed & Ahmad, Dr Rafiq. (2019). A scientometric analysis and critical review of computer vision applications for construction. Automation in Construction. 107. 10.1016/j.autcon.2019.102947.
[26] Lloyd, Stuart P. "Least squares quantization in PCM." Information Theory, IEEE Transactions on 28.2 (1982): 129-137
[27] Seungchan Kim, Kavosh Asadi, Michael Littman, and George Konidaris. 2019. Removing the Target Network from Deep Q-Networks with the Mellowmax Operator. In Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '19). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2060–2062.

Apéndices

A.- Función de codificación de una imagen por medio del extractor de características BOF.

```
def get_bof(gray, thres = 115, n_points = 180, disp=False):
    img_vert, img_hor = gray.shape

    gray = cv2.GaussianBlur(gray, (5, 5), 0)
    gray = np.around(gray).astype(np.uint8)
    b_w = cv2.threshold(gray, thres, np.amax(gray), cv2.THRESH_BINARY_INV)[1]

    contours = cv2.findContours(b_w, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    contours = imutils.grab_contours(contours)

    x_limit = img_hor / 8
    y_limit = img_vert / 8

    for c, contour in enumerate(contours):
        M = cv2.moments(contour)

        if (M['m00'] != 0):
            cX = int(M['m10'] / M['m00'])
            cY = int(M['m01'] / M['m00'])
        else:
            cX, cY = 0, 0

        if((cX >= x_limit and cY >= y_limit) and (cX < (x_limit * 7) and (cY < (y_limit * 7)))):
            contours = np.array(contours[c]).reshape(len(contours[c]), 2)
            coords = np.asarray(np.where(contours == cX)).T
            pos = np.zeros((coords.shape[0]))

            for k in reversed(range(0, coords.shape[0])):
                if(coords[k, 1] == 1):
                    coords = np.delete(coords, (k), axis = 0)

            coords = np.delete(coords, (1), axis = 1)
            min_val = 1000
            indice = 0

            for l in range (coords.shape[0]):
                if (contours[coords[l, 0], 1] < min_val):
                    min_val = contours[coords[l, 0], 1];
                    indice = l;

            indice = int(coords[indice])

            vals_2 = contours[0 : indice, :]
            vals_1 = contours[indice : contours.shape[0], :]
            contours = np.vstack((vals_1, vals_2))

            space = int((contours.shape[0] / n_points) * 100)
            space = space / 100

            fig_contour = np.zeros((n_points, 2)).astype(int)

            for g in range (n_points):
                a = int(np.floor(space * g))

                fig_contour[g, 0] = contours[a, 0]
                fig_contour[g, 1] = contours[a, 1]

            distances = np.sqrt(np.sum((fig_contour - [cX, cY]) **2, axis = 1))
            distances /= np.amax(distances)

            break
        else:
            print(f' C Not in limits {cX}, {cY}')

    return distances
```

B.- Función de proyección y codificación de modelos tridimensionales en forma de imágenes bidimensionales y un arreglo de Numpy.

```
files = os.listdir('models/')

for object_stl in files[:10]:
    try:
        print(object_stl)
        stl_mesh = mesh.Mesh.from_file(f'./models/{object_stl}')

        volume, cog, inertia = stl_mesh.get_mass_properties()
        stl_mesh.translate(-cog)

        figure = plt.figure(1, dpi=100.0)
        canvas = FigureCanvasAgg(figure)

        ax = Axes3D(figure)
        ax.add_collection3d(mplot3d.art3d.Poly3DCollection(stl_mesh.vectors))
        scale = stl_mesh.points.flatten()
        ax.auto_scale_xyz(scale, scale, scale)
        ax.set_axis_off()

        mesh_representation = []

        name=object_stl.split('.')[0]
        os.makedirs(f'geometries/{name}', exist_ok=True)

        for elev in  range(-90, 90, 5):
            print(elev)
            angle_representation = []
            for ii in range(0, 360, 5):
                ax.view_init(elev=float(elev), azim=ii)
                plt.savefig(f"geometries/{name}/img_{elev}_{ii}.png")
                img_arr = get_img_from_fig(figure)
                local_bof = get_bof(img_arr)

                angle_representation.append(local_bof)
            mesh_representation.append(angle_representation)
        bof_mesh_array = np.asarray(mesh_representation)

        print(bof_mesh_array.shape)

        np.save(f"./projections/{name}.npy", bof_mesh_array)

        del figure
        del canvas
        del stl_mesh

    except Exception as e:
        print(e)
        pass
```

C.- Función de inferencia empleando un extractor BOF como codificador y una red totalmente conectada como clasificador.

```
x = get_bof(img).astype('float32')
probs = model.predict(x[None, ...])
ob_class = np.argmax(probs, axis=-1)
print(f'class: {ob_class}\nprobs: {probs[0]}')
```

D.- Función de proyección de modelos tridimensionales en imágenes bidimensionales empleando Blender.

```
import bpy
import mathutils
from math import sin, cos, pi, radians

def update_camera(camera, light, p=20.0, phi=0, theta=0):

    x=p*sin(phi)*cos(theta)
    y=p*sin(phi)*sin(theta)
    z=p*cos(phi)

    camera.location = mathutils.Vector((x, y, z))
    light.location = camera.location/2

    looking_direction = camera.location
    rot_quat = looking_direction.to_track_quat('Z', 'Y')
    camera.rotation_euler = rot_quat.to_euler()


camera = bpy.data.objects['Camera']
light = bpy.data.objects['Light']
bpy.context.scene.render.resolution_x = 256
bpy.context.scene.render.resolution_y = 256
bpy.context.scene.render.film_transparent = True

for i in range(0,180+1,5):
    for j in range(0,360,5):
        update_camera(camera, light, phi=radians(i), theta=radians(j))
        bpy.context.scene.render.filepath = f'pro/{i}_{j}'
        bpy.ops.render.render(write_still=True)
        if i==0 or i==180:
            break
    #break

print("Done")
```

Nota: Dado que las proyecciones generadas por *Matplot* carecían de luz ambiental, se perdía la información tridimensional del modelo al momento de llevar a cabo las proyecciones en dos dimensiones lo cual impactaba notablemente en la capacidad de aprendizaje de los modelos que se entrenaron.

E.- Función de muestreo condicional empleada en el entrenamiento del modelo base.

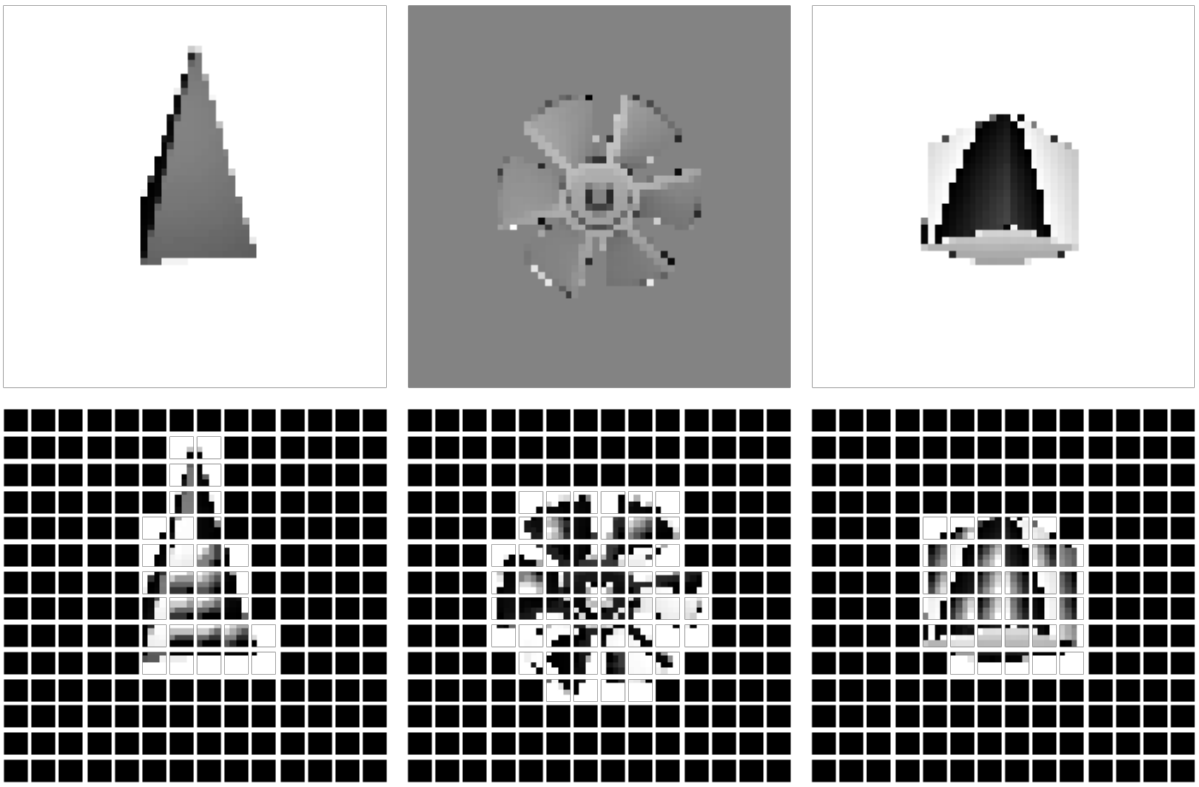
```
#@tf.function
def sample_from_class(x, y, classes=CLASSES, samples=SAMPLES):
    batch_size = x.shape[0]
    values = []

    for index in range(len(classes)):
        mask = y[:, 0]==index
        mask = mask[..., tf.newaxis, tf.newaxis, tf.newaxis]
        sample = samples[index]
        total = sample.shape[0]
        elements = np.random.choice(total, batch_size)
        data = sample[elements, ...]
        #np.random.shuffle(data)

        values.append(tf.matmul(data[:batch_size], tf.cast(mask, precision)))

    samples = tf.concat(values=values, axis=-1)
    return tf.reduce_sum(samples, axis=-1)[..., tf.newaxis]
```

F.- Descomposición de diferentes imágenes de entrada en forma de secuencia de parches que son empleados en el ViT.



G.- Optimizadores y funciones de pérdida empleadas en el G-VAE

Optimizadores y funciones de pérdida estándar

```
lr = 2e-4
optimizer1 = tf.keras.optimizers.Adam(lr)
optimizer2 = tf.keras.optimizers.Adam(lr)
optimizer3 = tf.keras.optimizers.Adam(lr)

if precision == tf.float16:
    optimizer1 = mixed_precision.LossScaleOptimizer(optimizer1)
    optimizer2 = mixed_precision.LossScaleOptimizer(optimizer2)
    optimizer3 = mixed_precision.LossScaleOptimizer(optimizer3)

triplet_loss = tfa.losses.TripletSemiHardLoss()
mse_loss = tf.keras.losses.MeanSquaredError()
cross_entropy = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction='none')
```

Funciones de pérdida personalizadas por bloque

```
def compute_results(model, x, y):
  mean, logvar = model.encode(x)
  z = model.reparameterize(mean, logvar)
  x_logit = model.decode(z)

  return [(mean, logvar), z , x_logit]

def log_normal_pdf(sample, mean, logvar, raxis=1):
  log2pi = tf.cast(tf.math.log(2. * np.pi), precision)
  return tf.reduce_sum(-.5 * ((sample - mean) ** 2. * tf.cast(tf.exp(-logvar), precision) + logvar + log2pi), axis=ra
  #return tf.reduce_sum(-.5 * ((sample - mean) ** 2. * tf.exp(-logvar) + logvar + log2pi), axis=raxis)

def compute_loss(intermediate_results, y, beta=0.80):
  [(mean, logvar), z, gen_output] = intermediate_results

  cross_ent = cross_entropy( y+1, gen_output )
  logpx_z =  -cross_ent
  logpz = log_normal_pdf(z, 0., 0.)
  logqz_x = log_normal_pdf(z, mean, logvar)
  return -tf.reduce_mean(logpx_z + (logpz - logqz_x)*beta)

def discriminator_loss(reference, real_output, fake_output):
  real_loss = cross_entropy(reference[0]+1, real_output)
  fake_loss = cross_entropy(reference[1], fake_output)
  total_loss = real_loss + fake_loss
  return tf.reduce_mean(total_loss)

def contrastive_loss(intermediate_results, y):
  [(mean, logvar), _, _] = intermediate_results
  #y_pred = tf.concat([mean, logvar], axis=1)
  #y_pred = tf.nn.dropout(mean, rate = 0.25)
  y_true = y[:, 0]+1
  loss = triplet_loss(y_true, mean)#tf.math.l2_normalize(y_pred, axis=1))
  return tf.reduce_mean(loss)

def generalization_loss(intermediate_results, x):
  y_true, _ = target_network.encode(x)
  [(y_pred, _), _, _] = intermediate_results
  return mse_loss(y_true, y_pred)
```

H.- Ciclo de entrenamiento personalizado del G-VAE

Entrenamiento del VAE y el Discriminador.

```
@tf.function#(jit_compile=True)
def train_step(model, x, y, optimizer1, optimizer2):
  with tf.GradientTape() as vae_tape, tf.GradientTape() as disc_tape:
    intermediate_results = compute_results(model, x, y)

    real_output = discriminator(x, training=True)
    intermediate_results[-1] = discriminator(intermediate_results[-1], training=True)

    vae_loss = compute_loss(intermediate_results, y) + generalization_loss(intermediate_results, x)
    disc_loss = discriminator_loss([y, tf.zeros_like(y)], real_output, intermediate_results[-1])

    if precision == tf.float16:
        scaled_vae_loss = optimizer1.get_scaled_loss(vae_loss)
        scaled_disc_loss = optimizer2.get_scaled_loss(disc_loss)

  if precision == tf.float16:
    scaled_gradients = vae_tape.gradient(scaled_vae_loss, model.trainable_variables)
    gradients_vae = optimizer1.get_unscaled_gradients(scaled_gradients)

    scaled_gradients = disc_tape.gradient(scaled_disc_loss, discriminator.trainable_variables)
    gradients_disc = optimizer2.get_unscaled_gradients(scaled_gradients)

  else:
    gradients_vae = vae_tape.gradient(vae_loss, model.trainable_variables)
    gradients_disc = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

  optimizer1.apply_gradients(zip(gradients_vae, model.trainable_variables))
  optimizer2.apply_gradients(zip(gradients_disc, discriminator.trainable_variables))

  return vae_loss, disc_loss
```

Entrenamiento de la Red objetivo.

```
@tf.function#(jit_compile=True)
def train_sps(model, x, y, optimizer):
  with tf.GradientTape() as cls_tape:
    mean, logvar = model.encode(x)
    cls_loss = contrastive_loss([(mean, logvar), _, _], y)

    if precision == tf.float16:
        scaled_cls_loss = optimizer1.get_scaled_loss(cls_loss)

  tvars = model.encoder.trainable_variables
  if precision == tf.float16:
    scaled_gradients = cls_tape.gradient(scaled_cls_loss, tvars)
    gradients_cls = optimizer.get_unscaled_gradients(scaled_gradients)
  else:
    gradients_cls = cls_tape.gradient(cls_loss, tvars)

  optimizer.apply_gradients(zip(gradients_cls, tvars))

  return cls_loss
```

I.- Visualización de la continuidad de las representaciones dentro del espacio latente.