

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UN SISTEMA DE VISIÓN ARTIFICIAL PARA DETECTAR AUTOMÓVILES ESTACIONADOS EN LUGARES NO PERMITIDOS

**TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE
INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN**

ARTEAGA POZO MADAI CARLOS

madai.arteaga@epn.edu.ec

DIRECTOR: ING. HENRY PATRICIO PAZ ARIAS, MSc.

henry.paz@epn.edu.ec

Quito, julio de 2018

DECLARACIÓN DE AUTORÍA

Yo, Arteaga Pozo Madai Carlos, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes de este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Arteaga Pozo Madai Carlos

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Arteaga Pozo Madai Carlos, bajo mi supervisión.

Ing. Henry Patricio Paz Arias, MSc.

DIRECTOR DE PROYECTO

DEDICATORIA

Para el mundo, para que sea un mejor lugar.

AGRADECIMIENTO

Primero, agradezco a las grandes mentes de la humanidad por que gracias a su curiosidad se inició un sempiterno proceso de desarrollo de la ciencia y que a través de los años ha estado en continuo progreso, porque gracias a ellas es posible que proyectos como éste sean desarrollados y así seguir contribuyendo al “beneficio y comodidad” de la humanidad.

Agradezco al Ing. Fausto Miranda Director General de la Agencia Metropolitana de Tránsito (AMT) quien me brindó su apoyo con el lugar de implementación del proyecto.

Un agradecimiento especial al Ing. Henry Patricio Paz Arias, mi director en este Proyecto de Titulación, por su apoyo, dedicación y paciencia con la realización de este trabajo.

Quiero hacer llegar mis agradecimientos a mis padres y hermanos por ser el modelo de vida a seguir, quienes construyeron los cimientos de mis valores éticos y morales; ellos son me motivan siempre a ser alguien mejor y los que me impulsan a seguir creciendo hasta lograr lo inefable.

A todos mis amigos y compañeros de clase con los cuales a través de este proceso de formación hemos compartido variedad de momentos y siempre han estado formado parte de mi pasado y mi diario presente.

ÍNDICE DE CONTENIDO

DECLARACIÓN DE AUTORÍA	ii
CERTIFICACIÓN.....	iii
DEDICATORIA	iv
AGRADECIMIENTO	v
1. INTRODUCCIÓN.....	1
1.1. Redes Neuronales Artificiales.....	3
1.1.1. Estructura de una Red Neuronal Artificial	3
1.1.2. Entrenamiento de una Red Neuronal Artificial	5
1.1.3. Redes Feed-forward multicapa	12
1.1.4. Aprendizaje Profundo – <i>Deep Learning</i>	14
1.1.5. Aplicaciones y Usos de la Red Neuronal	15
1.2. Redes Neuronales Convolucionales	16
1.2.1. Arquitectura de una Red Neuronal Convolutiva	18
1.2.1.1. Entrada.....	20
1.2.1.2. Capa de Convolución	22
1.2.1.3. Capa de Agrupado	27
1.2.1.4. Capa de Unidad de Rectificador Lineal.....	30
1.2.1.1. Capa de Deserción.....	31
1.2.1.2. Capa Totalmente Conectada.....	32
1.2.1.3. Salida	32
1.2.2. Entrenamiento y Validación en CNN.....	33
1.2.3. Consideraciones Computacionales sobre las ConvNet.....	35
1.3. Support Vector Machine	36
1.3.1. Clasificadores de Margen Máximo.....	36
1.3.1.1. Hiperplanos	36
1.3.1.2. Clasificación usando Hiperplanos.....	36
1.3.1.3. Clasificador de Margen Máximo	38

1.3.1.4.	Clasificadores de Soporte Vectorial.....	40
1.3.2.	Máquinas de Soporte Vectorial.....	42
1.3.3.	Matriz de Confusión.....	44
1.4.	Herramientas y Entorno de Desarrollo.....	46
1.4.1.	Hardware y Sistema Operativo.....	47
1.4.2.	<i>Frameworks</i> para ConvNet.....	48
1.4.3.	<i>Frameworks</i> para Máquinas de Soporte Vectorial.....	49
1.4.4.	<i>Frameworks</i> para Visión por Computador.....	50
2.	METODOLOGÍA.....	52
2.1.	Análisis del set de datos.....	52
2.1.1.	Limpieza de las imágenes y Estandarización de set de datos.....	54
2.1.2.	Resultado de limpieza del set de datos.....	57
2.2.	Modelo ConvNet – SVM.....	58
2.2.1.	Hiperparámetros e Inicialización de pesos.....	59
2.2.2.	Esquema de la CNN-SVM.....	60
2.3.	Entrenamiento y Validación del modelo CNN-SVM.....	61
2.3.1.	Fase Uno.....	63
2.3.2.	Fase Dos.....	66
2.3.3.	Fase Tres.....	70
2.3.4.	Fase Cuatro.....	74
2.3.5.	Fase Cinco.....	78
2.3.6.	Resumen del Entrenamiento, Validación y Testeo.....	82
2.4.	Caso de Estudio.....	84
2.4.1.	Implementación del Sistema de Visión Artificial.....	85
2.4.2.	Ejecución y Resultados Aplicados al Caso de Estudio.....	90
3.	CONCLUSIONES.....	95
4.	RECOMENDACIONES.....	96
5.	BIBLIOGRAFÍA.....	97
	GLOSARIO.....	104

ANEXOS.....	107
Anexo I.....	108
Anexo II.....	109

ÍNDICE DE ECUACIONES

ECUACIÓN 1. MODELO BÁSICO DE UNA NEURONA.	4
ECUACIÓN 2. ECUACIÓN GENERAL DE UNA RED NEURONAL CON FUNCIÓN DE ACTIVACIÓN SIGMOIDE.	4
ECUACIÓN 3. MODELO DE APRENDIZAJE SUPERVISADO.	6
ECUACIÓN 4. MODELO DE APRENDIZAJE NO SUPERVISADO.	7
ECUACIÓN 5. MINIMIZACIÓN DEL ERROR CUADRÁTICO.	7
ECUACIÓN 6. SENSIBILIDAD DE APRENDIZAJE.	11
ECUACIÓN 7. DERIVADA GENERAL DEL ERROR CON RESPECTO A CUALQUIER PESO.	11
ECUACIÓN 8. FUNCIÓN DE COSTE DEL ERROR CUADRADO.	13
ECUACIÓN 9. MODELO CONVOLUCIONAL.	22
ECUACIÓN 10. PROCEDIMIENTO DE CONVOLUCIÓN COMPUTACIONAL.	24
ECUACIÓN 11. NÚMERO DE NEURONAS A ENTRENAR.	24
ECUACIÓN 12. BACK-PROPAGATION EN CONVNET.	27
ECUACIÓN 13. MODELO DE MAX POOLING.	28
ECUACIÓN 14. MODELO DE AVERAGE POOLING.	28
ECUACIÓN 15. MATRIZ DE INDICADORES EN LA CAPA POOLING.	29
ECUACIÓN 16. CÁLCULO DE BACK-PROPAGATION EN LA CAPA DE POOLING.	29
ECUACIÓN 17. TRUNCAMIENTO DE ENTRADA EN EL MODELO RELU.	30
ECUACIÓN 18. DIFERENCIAL DE LA CAPA RELU.	30
ECUACIÓN 19. MODELO DE UNA CAPA DROPOUT.	32
ECUACIÓN 20. FUNCIÓN DE PÉRDIDA.	34
ECUACIÓN 21. DEFINICIÓN DE UN HIPERPLANO.	36
ECUACIÓN 22. SOLUCIÓN BÁSICA DE PROBLEMAS DE CLASIFICACIÓN LINEAL.	37
ECUACIÓN 23. CLASIFICADOR DE MARGEN MÁXIMO.	39
ECUACIÓN 24. DISTANCIA DE UN PUNTO AL HIPERPLANO.	40
ECUACIÓN 25. HIPERPLANO CON VECTORES DE SOPORTE.	40
ECUACIÓN 26. HIPERPLANO CON VECTORES DE SOPORTE REESCRITO COMPUTACIONALMENTE.	42
ECUACIÓN 27. FUNCIÓN DE LAGRANGE APLICADA AL PROBLEMA DE OPTIMIZACIÓN.	43
ECUACIÓN 28. SOLUCIONES DE LA FUNCIÓN DE LAGRANGE.	43
ECUACIÓN 29. SOLUCIONES PARA EL PROBLEMA DE OPTIMIZACIÓN DEL HIPERPLANO CON VECTORES DE SOPORTE.	43
ECUACIÓN 30. EXACTITUD (AC).	45
ECUACIÓN 31. TAZA DE VERDADEROS POSITIVOS (TPR).	45
ECUACIÓN 32. TAZA DE FALSOS POSITIVOS (FPR).	45
ECUACIÓN 33. TAZA DE VERDADEROS NEGATIVOS (TNR).	45
ECUACIÓN 34. TAZA DE FALSOS NEGATIVOS (FNR).	46
ECUACIÓN 35. TAZA DE ERROR (ERR).	46
ECUACIÓN 36. COEFICIENTE DE CORRELACIÓN DE MATTHEWS.	46

ÍNDICE DE FIGURAS

FIGURA 1. ESTRUCTURA DE UNA NEURONA ARTIFICIAL.	3
FIGURA 2. ARQUITECTURAS DE UNA RED NEURONAL.....	5
FIGURA 3. ALGORITMO DE BACK-PROPAGATION.	10
FIGURA 4. ARQUITECTURA DE UNA RED NEURONAL ARTIFICIAL FEED-FORWARD.	12
FIGURA 5. CÁLCULO DEL ERROR EN UNA ANN.	13
FIGURA 6. EJEMPLO DE DEEP LEARNING.	14
FIGURA 7. EJEMPLO DEL PROCESO DE UNA RED NEURONAL CONVOLUCIONAL.	17
FIGURA 8. MODELO DE UNA ARQUITECTURA TÍPICA DE UNA CONVNET.	18
FIGURA 9. COMPARACIÓN DE LA ARQUITECTURA DE REDES NEURONALES ARTIFICIALES Y CONVOLUCIONALES.	19
FIGURA 10. ARQUITECTURA BÁSICA DE UNA CONVNET.	20
FIGURA 11. EJEMPLO DE CONVOLUCIÓN EN MATRICES.....	23
FIGURA 12. PROCESO DE LA CAPA DE CONVOLUCIÓN.	26
FIGURA 13. EJEMPLO DE POOLING.	27
FIGURA 14. FUNCIÓN RELU.	30
FIGURA 15. EJEMPLO DE DROPOUT.	31
FIGURA 16. INICIALIZACIÓN DE PESOS EN UNA RED NEURONAL.	33
FIGURA 17. EJEMPLO DE HIPERPLANOS QUE SEPARAN LAS MUESTRAS.	37
FIGURA 18. EJEMPLO DE CLASIFICACIÓN USANDO HIPERPLANO.	38
FIGURA 19. HIPERPLANO DE MARGEN MÁXIMO.....	39
FIGURA 20. SET DE DATOS CON MUESTRAS NO SEPARABLES	39
FIGURA 21. SOLUCIÓN GRÁFICA DE HIPERPLANO CON VECTORES DE SOPORTE.....	41
FIGURA 22. USO DE VARIOS PARÁMETROS C EN EL CLASIFICADOR CON VECTORES DE SOPORTE.....	42
FIGURA 23. MUESTRA DEL SET DE IMÁGENES DE AUTOMÓVILES DE LA UNIVERSIDAD DE STANFORD.	52
FIGURA 24. MUESTRA DEL SET DE IMÁGENES DE AUTOMÓVILES DE LA UNIVERSIDAD POLITÉCNICA DE MADRID.	53
FIGURA 25. MUESTRA DEL SET DE IMÁGENES DE AUTOMÓVILES DE IMAGENET.	53
FIGURA 26. REPRESENTACIÓN GRÁFICA DEL SET DE IMÁGENES.....	54
FIGURA 27. MUESTRA ALEATORIA LUEGO DE LA LIMPIEZA DEL SET DE IMÁGENES.	55
FIGURA 28. RESULTADO DE LIMPIEZA Y ESTANDARIZACIÓN.....	56
FIGURA 29. RESULTADO DE LA APLICACIÓN DE DATA AUGMENTATION.....	57
FIGURA 30. RESULTADO DE LA LIMPIEZA DEL SET DE DATOS.	57
FIGURA 31. MODELO PROPUESTO DEL CNN-SVM.	60
FIGURA 32. FASE UNO: EXACTITUD Y PÉRDIDA.	63
FIGURA 33. FASE UNO: REPRESENTACIÓN DE LA EXTRACCIÓN DE CARACTERÍSTICAS.	64
FIGURA 34. FASE UNO: COMPARACIÓN CLASIFICADORES SVM.	65
FIGURA 35. FASE UNO: MATRIZ DE CONFUSIÓN REALIZADA CON SVM KERNEL LINEAL.....	65
FIGURA 36. FASE DOS: EXACTITUD Y PÉRDIDA.	67

FIGURA 37. FASE DOS: REPRESENTACIÓN DE LA EXTRACCIÓN DE CARACTERÍSTICAS.....	68
FIGURA 38. FASE DOS: COMPARACIÓN CLASIFICADORES SVM.....	69
FIGURA 39. FASE DOS: MATRIZ DE CONFUSIÓN REALIZADA CON SVM KERNEL LINEAL.	69
FIGURA 40. FASE TRES: EXACTITUD Y PÉRDIDA.....	71
FIGURA 41. FASE TRES: REPRESENTACIÓN DE LA EXTRACCIÓN DE CARACTERÍSTICAS.	72
FIGURA 42. FASE TRES: COMPARACIÓN DE CLASIFICADORES SVM.....	73
FIGURA 43. FASE TRES: MATRIZ DE CONFUSIÓN REALIZADA CON SVM KERNEL LINEAL.....	73
FIGURA 44. FASE CUATRO: EXACTITUD Y PÉRDIDA.	75
FIGURA 45. FASE CUATRO: REPRESENTACIÓN DE LA EXTRACCIÓN DE CARACTERÍSTICAS.	76
FIGURA 46. FASE CUATRO: COMPARACIÓN DE CLASIFICADORES SVM.	77
FIGURA 47. FASE CUATRO: MATRIZ DE CONFUSIÓN REALIZADA CON SVM KERNEL LINEAL.	77
FIGURA 48. FASE CINCO: EXACTITUD Y PÉRDIDA.....	79
FIGURA 49. FASE CINCO: REPRESENTACIÓN DE LA EXTRACCIÓN DE CARACTERÍSTICAS.....	80
FIGURA 50. FASE CINCO: COMPARACIÓN DE CLASIFICADORES SVM.....	80
FIGURA 51. FASE CINCO: MATRIZ DE CONFUSIÓN REALIZADA CON SVM KERNEL LINEAL.....	81
FIGURA 52. TEST: REPRESENTACIÓN DE LA EXTRACCIÓN DE CARACTERÍSTICAS.	83
FIGURA 53. TEST: MATRIZ DE CONFUSIÓN APLICANDO EL MODELO ENTRENADO.	83
FIGURA 54. DIAGRAMA DE FLUJO DEL PROGRAMA.	87
FIGURA 55. FOTOGRAMA QUE MUESTRA LA PERSPECTIVA DEL VIDEO.	88
FIGURA 56. BARRIDO DE RECORTES EN FOTOGRAMA.	89
FIGURA 57. MUESTRA DEL PROCESO DE RASTREO DE SISTEMA DE VISIÓN ARTIFICIAL.	92
FIGURA 58. PROCESO DE CAPTURA DE INFRACCIÓN DE UN VEHÍCULO.	93
FIGURA 59. EJEMPLO DE PROCESO DE CAPTURA DE INFRACCIÓN.....	93
FIGURA 60. ZONAS DEL FOTOGRAMA	109

ÍNDICE DE TABLAS

TABLA 1. EJEMPLO DE MATRIZ DE CONFUSIÓN.....	45
TABLA 2. HARDWARE USADO EN EL PROYECTO.	47
TABLA 3. ARQUITECTURA INICIAL DE LA CONVNET.	58
TABLA 4. HIPERPARÁMETROS DE CNN.	59
TABLA 5. HIPERPARÁMETROS DE SVM.	59
TABLA 6. FASE UNO: VALORES PRESENTES EN LA ÚLTIMA ÉPOCA ANALIZADA EN CNN.....	64
TABLA 7. FASE UNO: ENTRENAMIENTO Y VALIDACIÓN SVM.	64
TABLA 8. FASE UNO: ANÁLISIS DE LA MATRIZ DE CONFUSIÓN.	66
TABLA 9. FASE DOS: MODELO CONVNET.....	66
TABLA 10. FASE DOS: VALORES PRESENTES EN LA ÚLTIMA ÉPOCA ANALIZADA EN CNN.	67
TABLA 11. FASE DOS: ENTRENAMIENTO Y VALIDACIÓN SVM.....	68
TABLA 12. FASE DOS: ANÁLISIS DE LA MATRIZ DE CONFUSIÓN.....	70
TABLA 13. FASE TRES: MODELO CONVNET.....	70
TABLA 14. FASE TRES: VALORES PRESENTES EN LA ÚLTIMA ÉPOCA ANALIZADA EN CNN.....	71
TABLA 15. FASE TRES: ENTRENAMIENTO Y VALIDACIÓN SVM.	72
TABLA 16. FASE TRES: ANÁLISIS DE LA MATRIZ DE CONFUSIÓN.	74
TABLA 17. FASE CUATRO: MODELO CONVNET.	74
TABLA 18. FASE CUATRO: VALORES PRESENTES EN LA ÚLTIMA ÉPOCA ANALIZADA EN CNN.....	75
TABLA 19. FASE CUATRO: ENTRENAMIENTO Y VALIDACIÓN SVM.	76
TABLA 20. FASE CUATRO: ANÁLISIS DE LA MATRIZ DE CONFUSIÓN.	78
TABLA 21. FASE CINCO: MODELO CONVNET.....	78
TABLA 22. FASE CINCO: VALORES PRESENTES EN LA ÚLTIMA ÉPOCA ANALIZADA EN CNN.....	79
TABLA 23. FASE CINCO: ENTRENAMIENTO Y VALIDACIÓN SVM.	80
TABLA 24. FASE CINCO: ANÁLISIS DE LA MATRIZ DE CONFUSIÓN.	81
TABLA 25. RESUMEN DE LAS FASES DE ENTRENAMIENTO Y VALIDACIÓN.	82
TABLA 26. FASE UNO: ANÁLISIS DE LA MATRIZ DE CONFUSIÓN.	84

RESUMEN

El presente proyecto muestra la implementación de un modelo que será la unión de dos técnicas de clasificación: redes neuronales convolucionales y máquinas de soporte vectorial aplicados en un sistema de visión artificial. Éste funciona en tiempo real para detectar autos estacionados en lugares no permitidos. Dicha implementación se enfoca en realizar un control de infracciones de forma autónoma.

Las redes neuronales convolucionales previo a un entrenamiento y validación se encargan de extraer un mapa de características de las imágenes de entrada. Luego, este mapa ingresa al clasificador, máquinas de soporte vectorial, creado por un proceso previo de entrenamiento y validación el cual genera una salida que determina si existe o no un automóvil. Las imágenes de entrada son los fotogramas de un video, que ingresan al modelo; el que genera una salida que confirma que en estos fotogramas se encuentra un automóvil. Si este ingresa a una zona de infracción y se detiene en ella se realiza una captura de la infracción, caso contrario el rastreador sigue al auto hasta que este salga del cuadro de video.

La parte final del trabajo contiene los resultados de la implementación del sistema de visión artificial donde se presentan las conclusiones y recomendaciones sobre el uso de la unión de estas dos técnicas de clasificación.

Palabras clave: Redes neuronales convolucionales, Máquinas de soporte vectorial, Visión artificial, Set de datos, Entrenamiento, Validación.

ABSTRACT

This project shows an implementation of a model that will be the union between two classification techniques: convolutional neural networks and vector support machines applied in an artificial vision system. It works in real time in order to detect cars parked in places not allowed. This implementation focuses on controlling infringements autonomously.

The convolutional neural networks, previously training and validated, are responsible for generating a map of characteristics of the input images. Then, this map goes into the classifier, the vector support machine which is created by a previous process of training and validation. This activity generates an output that determines whether or not a car is located in the area. The input images are the frames of a video. These images go into the model; which generates an output to confirm that there is a car in the frames. If the car goes into an infringement area and takes too much time in it, a capture of the infraction is made; otherwise the tracker follows the car until it leaves from the video frame.

The final part of this project contains the results of the implementation of the artificial vision system. There, the conclusions and recommendations will be espoused in order to determine the use of the union between the two classification techniques.

Keywords: Convolutional neural networks, Support vector machine, Artificial Vision, Dataset, Training, Validation.

1. INTRODUCCIÓN

El presente proyecto se enmarca en uno de los enfoques de la Inteligencia Artificial. El denominado “sistemas que piensan como humanos”. En este sentido, se trata de automatizar acciones que son vinculadas con los procesos del pensamiento humano, “actividades como la toma de decisiones, resolución de problemas, aprendizaje, entre otras” [1]. De forma particular se refiere a la integración de las técnicas de clasificación: redes neuronales artificiales, en inglés *Artificial Neural Networks* (ANNs) y máquinas de soporte vectorial, en inglés *Support Vector Machines* (SVMs). Estas técnicas de clasificación y detección serán, en tiempo real, aplicadas en la detección de autos estacionados en lugares no permitidos. Las ANNs son modelos computacionales que agrupan varias neuronas artificiales, cada neurona artificial pretende simular el comportamiento de los axones observados en las neuronas de cerebros biológicos [2]. El SVM es un clasificador de clases, su objetivo es buscar un hiperplano que separe las muestras de cada clase con la máxima distancia posible entre ellas [3].

Para desarrollar el proyecto es necesario recordar que no existe una metodología específica en lo que respecta al aprendizaje de máquina. Sin embargo, se han determinado fases o etapas de entrenamiento y validación [4]. Con el entrenamiento tanto la ANN como el SVM reconocen patrones con el propósito de generar un resultado esperado; con la validación se comprueba que las salidas esperadas son correctas. Se debe considerar que la clasificación deberá ser similar a la concebida por un ser racional [5].

El desarrollo del sistema de visión artificial inicia con un análisis previo de un set de datos de imágenes. Para este caso, todas las imágenes del set deben mantener un mismo tamaño y formato de archivo de imagen, caso contrario se deberá modificar el set de imágenes hasta tener el set estandarizado. A continuación, inicia el proceso de aprendizaje supervisado, en el que se realiza un entrenamiento y validación tanto en las redes neuronales como en el SVM. Las redes neuronales son usadas para extraer las características de las imágenes y el SVM es el clasificador de las características obtenidas.

Los resultados obtenidos del aprendizaje supervisado se validan mediante la matriz de confusión. En la matriz cada fila representa las instancias en la clase predicha mientras que las columnas representan las instancias en la clase actual [6]. Para la validación de la medida de aceptabilidad se usará el coeficiente de correlación de Mathews [7]. En caso de que los resultados no sean los deseados, se deberá modificar los hiperparámetros de la red neuronal y/o del SVM; de forma que se vuelve a las etapas de entrenamiento y validación hasta lograr obtener el mejor modelo posible. De este proceso se obtiene el

modelo de clasificación que deberá ser implementado como la parte medular del sistema de visión artificial para la detección de vehículos estacionados en lugares no permitidos.

El sistema de visión artificial analizará cada fotograma del video de entrada, realizando un barrido de cada fotograma; de este proceso se toman recortes en la zona esperada de vehículos en movimiento. Estos recortes serán ingresados al clasificador, a continuación se generará una zona donde se encuentra el auto para que éste sea rastreado. En caso de ingresar al sector de “prohibido estacionar” y permanecer cinco segundos inmóvil, el sistema de visión realizará una captura del auto y guardará en disco la fotografía de la infracción.

Se debe destacar que el modelo no es absoluto pues se basa en probabilidades y tiene un margen de error el cual tiene que ser minimizado lo máximo posible. Del proyecto se espera obtener altos resultados de exactitud y precisión en cuanto a la clasificación de imágenes se refiere.

Esta iniciativa nace como un aporte a la sociedad en general; se pretende mitigar de forma autónoma las infracciones de vehículos estacionados en lugares no permitidos; así se contribuirá en la generación de una cultura de respeto y cumplimiento a las normas de tránsito.

Para que el proyecto se desarrolle con normalidad se tiene previsto explorar de forma independiente las ANNs, en especial las Redes Neuronales Convolucionales y los SVM para comprender bajo que fundamentos matemáticos y probabilísticos se encuentran desarrolladas éstas técnicas de detección y clasificación. Este proceso se realizará con ayuda de librerías de visión por computador para rastrear el vehículo; librerías de aprendizaje de máquina y librerías de Redes Neuronales Convolucionales. Además, es necesario contar con Software y Hardware dedicado al procesamiento gráfico.

1.1. Redes Neuronales Artificiales

En esta sección se trata temas relacionados con las redes neuronales artificiales. Su estructura y funcionamiento, además se presenta brevemente su ámbito de aplicación. Evolutivamente las ANNs se basan en las neuronas biológicas y al igual que éstas, las neuronas artificiales se encuentran conectadas en varias capas. Históricamente la evolución de esta teoría nace con las investigaciones de McCulloch y Pitts' sobre las neuronas sensibles que están localizadas y orientadas selectivamente en el sistema visual de un gato, realizada en 1940. Posteriormente, en 1960, Rosenblatt's desarrolla el teorema de convergencia del perceptrón [8]. Luego, desde 1982, gracias al esfuerzo de varios investigadores se produce el mayor desarrollo de las ANN cuando se introduce el algoritmo *back-propagation* para perceptrones multicapa [9], [10], [11], [12], [13].

1.1.1. Estructura de una Red Neuronal Artificial

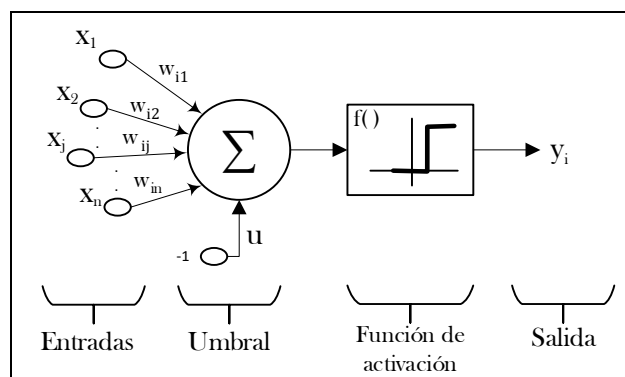


Figura 1. Estructura de una neurona artificial.

La red neuronal artificial es un modelo computacional, que se basa en la estructura natural de una red neuronal biológica y como tal, la red está compuesta por neuronas artificiales que procesan información a través de varias capas que la forman. Generalmente la ANN está compuesta de tres capas: la capa de entrada, la capa oculta y la capa de salida. En la inicial, todas las entradas son multiplicadas por pesos. Estos pesos son calculados mediante una función matemática que determina la activación de la neurona. Las entradas y los pesos en conjunto representan el flujo de información. Luego de este proceso, otra función realiza el cálculo de la salida de la neurona artificial que a menudo depende de un umbral [14]. En la Figura 1 se muestra el proceso previamente descrito.

Modelo computacional de una neurona artificial

Matemáticamente una neurona calcula la suma de los pesos de las n entradas con las señales $x_j, j = 1, \dots, n$, El resultado es un modelo de clasificación donde sus salidas 1 o 0 son calculadas con la siguiente ecuación:

$$y(x, w) = f\left(\sum_{j=1}^M w_j x_j - u\right),$$

Ecuación 1. Modelo básico de una neurona.

Donde el vector x es la entrada, w son los pesos, el vector y es la salida, $f(\cdot)$ es la función de activación y u es el umbral, a menudo se considera que el peso $w_0 = -u$ y la entrada $x_0 = 1$.

En la Ecuación 1 los pesos positivos significan paso de información y los negativos cese de información. En analogía con las neuronas biológicas, los valores positivos corresponden a un proceso de alta sinapsis; mientras que los negativos a un proceso de inhibición [15].

De forma general, una neurona usa funciones de activación y umbrales. Las funciones de activación más conocidas son: Identify, Step, Sigmoid, Tanh, ReLU, Softmax, entre otras [16]. A pesar de que en la práctica se utiliza funciones de activación tipo sigmoide debido a su función de crecimiento suave por sus propiedades asintóticas. Una red neuronal con funciones de activación tipo sigmoide se representa como en la Ecuación 2 [17]:

$$y_k(x, w) = \sigma\left(\sum_{j=1}^M w_{kj}^{(2)} f\left(\sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right),$$

Ecuación 2. Ecuación general de una red neuronal con función de activación sigmoide.

Donde $\sigma^{(L)} = \frac{1}{1 + \exp(-\beta_i^{(L)})}$. Para los otros tipos de funciones de activación se tienen análisis similares. Una función de activación se aplica sobre las salidas de estas neuronas [18].

Arquitecturas de una Red Neuronal Artificial

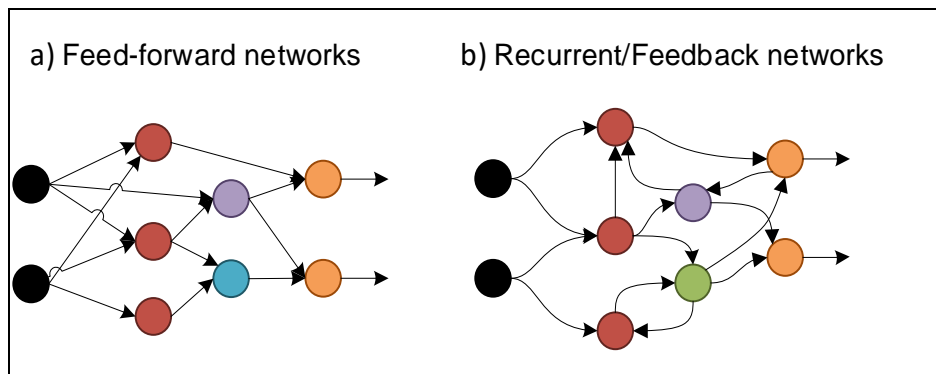


Figura 2. Arquitecturas de una red neuronal.

Las ANNs pueden ser construidas directamente mediante grafos. Los nodos son las neuronas y las aristas con pesos son las conexiones entre las neuronas de entrada y salida. Al basarse en los patrones que forman los grafos se han definido dos categorías [19]:

- a) **Redes del tipo *feed-forward***: No presentan ciclos en sus grafos. Por ejemplo: *Single-layer perceptron*, *Multilayer perceptron*, *Radial Basis Functions Nets* entre otras. Un ejemplo gráfico se muestra en el literal a) de la Figura 2.
- b) **Redes del tipo *recurrent* o *feedback***: son las que tienen lasos debido a sus conexiones del tipo *feedback*. Por ejemplo: *Competitive networks*, *Kohonen's SOM*, *Hopfield network*, *ART models*, entre otras. Un ejemplo gráfico se muestra en el literal b) de la Figura 2..

1.1.2. Entrenamiento de una Red Neuronal Artificial

Para que una ANN muestre salidas correctas necesita un entrenamiento previo. En el proceso de entrenamiento la red neuronal calibra los pesos de las conexiones hasta generar un modelo óptimo para que una entrada determinada genere una salida esperada. A esto se conoce como un proceso de aprendizaje. La habilidad que poseen las ANN de aprender desde ejemplos en vez de un conjunto de reglas especificadas por el ser humano hace que sean muy usadas para determinar las relaciones y patrones entre entradas y salidas [20].

Aprendizaje en Redes Neuronales Artificiales

Un proceso de aprendizaje en el contexto de las ANN actualiza la arquitectura y los pesos de la conexión para que una red sea eficiente al momento de resolver una tarea específica. Los pesos de una conexión se aprenden de patrones de entrenamiento y su rendimiento es implementado por la actualización dinámica de los pesos de las conexiones. El proceso de aprendizaje de una red neuronal consta de: información previa y de algoritmos de aprendizaje. Éstos se usan para la actualización de los pesos de la red. En este proceso se conocen tres paradigmas de aprendizaje [21]:

- a) **El aprendizaje supervisado** también es conocido como aprendizaje con profesor y consiste en que a la red se le da a conocer cuáles son las salidas por cada patrón de entrada.
- b) **El aprendizaje no supervisado** es aprender sin profesor, en este caso no se requiere de una respuesta correcta asociada con cada patrón de entrada.
- c) **Aprendizaje híbrido** se combinan las dos técnicas anteriores. Una cantidad de los pesos son determinados por el aprendizaje supervisado y la otras con aprendizaje no supervisado.

Aprendizaje supervisado

El aprendizaje supervisado tiene las siguientes variables conocidas: un conjunto de variables de entrada y un conjunto de salidas asociadas a las variables de entrada. Estas variables determinan la relación existente entre estos conjuntos. Para construir un modelo que clasifique se debe modelar la relación entre la entrada y la salida. Para ello se tiene un número finito de muestras $D = \{(x_i, y_i). i = 0 \dots n - 1\}$, usadas para construir la predicción a partir de una entrada aleatoria. El modelo de aprendizaje es:

$$\hat{Y} = F(x; D),$$

Ecuación 3. Modelo de Aprendizaje Supervisado.

Donde \hat{Y} es la predicción que se realiza de Y . Para el modelo de aprendizaje generado luego de que este pasó por un proceso de aprendizaje supervisado, cualquier entrada será clasificada basándose en ese modelo [21].

Aprendizaje no supervisado

El aprendizaje no supervisado estima de forma precisa una función de probabilidad de un conjunto de entradas denominada como $f_x(x)$. Esto representa la estructura estadística de la entrada y está asociada al proceso computacional a modelar. De esta forma se tiene la entrada en términos estadísticos. Este aprendizaje se basa en construir una función de x a partir de un conjunto de entrenamiento $D = \{x_i, i = 0 \dots n - 1\}$, se debe usar D para generar una predicción de la entrada. El modelo de aprendizaje no supervisado es:

$$\hat{X} \simeq F(x; D)$$

Ecuación 4. Modelo de Aprendizaje no supervisado.

Donde \hat{X} es la predicción que se realiza de X después de que la entrada haya sido procesada por el modelo de predicción [21].

Ajuste en los modelos

Para obtener un modelo correcto, tanto en el aprendizaje supervisado como en el aprendizaje no supervisado, usualmente se realiza la minimización del error cuadrático medio que se resuelve con la Ecuación 5.

$$\begin{aligned} I_{emp}^U[F(x; D)] &= \frac{1}{2n} \sum_{i=0}^{n-1} \|x_i - F(x_i)\|^2 \\ &= \frac{1}{2n} \sum_{i=0}^{n-1} \sum_{j=0}^{p-1} (x_{ij} - F_j(x_j))^2 \end{aligned}$$

Ecuación 5. Minimización del error cuadrático.

Dependiendo de la forma de aproximar F se encuentran varias soluciones al mismo problema estadístico. F se puede aproximar de forma local usando varios métodos entre los principales [22], [23]:

- Análisis de clúster.
 - Matrices de proximidad.
 - *K-means*.
 - Algoritmos de *Clustering*.
- Auto organización de mapas.

- Componente Principal, Curvas y Superficies.
 - *Clustering* Espectral.
 - Componente principal de kernel.
 - Componente principal de *Sparse*.
- Factorización de Matrices no negativas.
 - Análisis de arquetipos.
- Análisis de componente independiente.
 - Análisis de Factor y Variables latentes.
 - Proyección exploratoria de *Persitut*.
- Escala multidimensional.
- Algoritmo de Google *PageRank*.

Reglas de Aprendizaje

La teoría de aprendizaje tiene tres problemas fundamentales a considerar, estos problemas son [24]:

- a) **La capacidad de almacenamiento.** Es la cantidad de patrones (entrada y salida), funciones y límites de decisión de la red que puede ser almacenada.
- b) **La complejidad de las muestras.** Esta determina el número de patrones de entrenamiento de la red que garantizan una generalización de un modelo válido. Si se tiene pocas muestras se puede causar un sobre ajuste al entrenamiento, en inglés *overfitting*. Esto ocurre cuando la red trabaja bien con el set de entrenamiento, pero muestra resultados muy pobres para patrones independientes relacionados con el set de entrada.
- c) **La complejidad computacional.** Esta se relaciona directamente con el tiempo requerido por el algoritmo de aprendizaje para estimar una solución desde los patrones de entrenamiento.

Para resolver estos problemas hay cuatro reglas básicas de aprendizaje: La regla del error de corrección, el aprendizaje de Boltzman, las reglas Hebbianas y las reglas de aprendizaje competitivo.

Reglas de Corrección de Errores

Basándose en el paradigma del aprendizaje supervisado, una red neuronal tiene un conjunto de salidas deseadas para sendas entradas. Durante el proceso de aprendizaje, una salida y generada por la red puede o no ser igual a la salida deseada d . El principio básico de corrección de errores de las reglas de aprendizaje es usar la señal de error $(d - y)$ para modificar los pesos de conexión y reducir gradualmente el error.

El aprendizaje del perceptrón se basa en el principio de la corrección de errores. Un perceptrón es una neurona artificial con pesos ajustables $w_j, j = 1, \dots, n$, y un umbral u . El algoritmo del perceptrón relacionado con el aprendizaje es el siguiente:

1. Inicializar los pesos y el umbral con pequeños números de forma aleatoria.
2. Procesar el vector de entrada $(x_1, \dots, x_n)^t$ y evaluar la salida de la neurona.
3. Actualizar los pesos de acuerdo con $w_j(t + 1) = w_j(t) + \eta(d - y)x_j$, donde d es la salida deseada, t es el número de iteración y $\eta(0 < \eta < 1.0)$ es la ganancia.

Dado como entrada un vector $\mathbf{x} = (x_1, \dots, x_n)^t$, la entrada de la neurona es:

$$y = \sum_{j=1}^n w_j x_j - u,$$

Donde la salida y del perceptrón es $+1$ cuando $y > 0$ y 0 cuando $y \leq 0$.

Para un clasificador de dos clases el perceptrón mostrará para cada entrada una salida $y = 1$ si pertenece a una clase o $y = 0$ si pertenece a otra clase. Cabe recalcar que el aprendizaje ocurre cuando el perceptrón genera una salida errónea. Según el teorema de la convergencia del perceptrón, éste deberá converger luego de un número finito de iteraciones. A pesar de que en ocasiones las muestras de entrada no son linealmente separables, se han propuesto variaciones de este algoritmo [25].

Un algoritmo particular basado en el principio de corrección del error es el algoritmo de *back-propagation*. Éste se usa para calcular las derivadas del error con respecto a los pesos. Su objetivo es realizar ajustes en los pesos de las conexiones para minimizar el error de clasificación de la red neuronal [26].

Algoritmo de Back-propagation

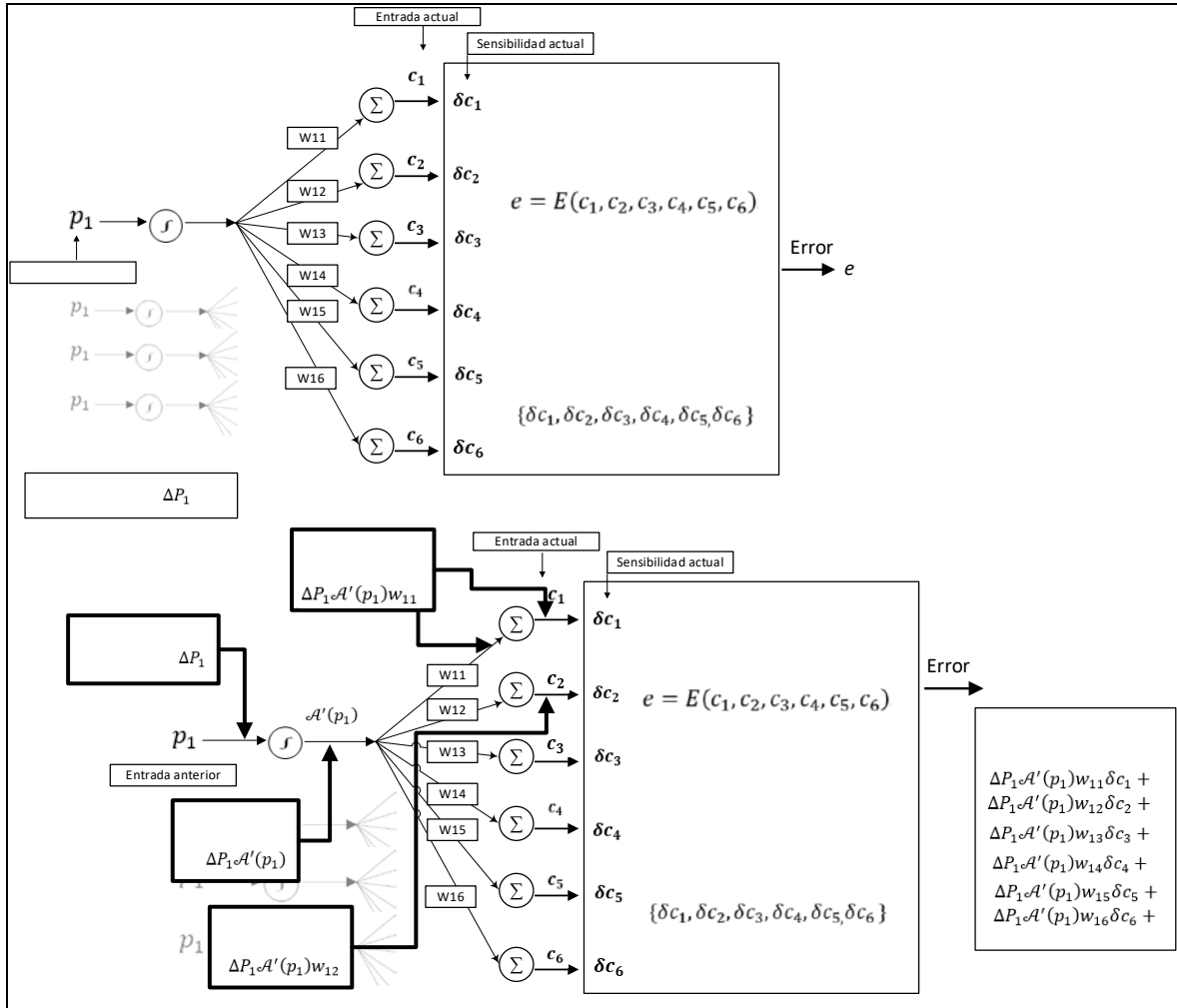


Figura 3. Algoritmo de back-propagation.

El algoritmo de *back-propagation* busca minimizar el error de la función con respecto a los pesos usando el método de descenso de gradiente. Dado que la combinación adecuada de los pesos minimiza el error de la función. Se considera al algoritmo de *back-propagation* como una solución al problema de aprendizaje. Éste calcula el gradiente de la función de error para cada iteración [27].

La explicación del algoritmo que ejecuta el back-propagation se muestra a continuación: Si luego de un proceso se tiene como salida $\{o_1, o_2, o_3, o_4\}$ y como valores objetivo $\{t_1, t_2, t_3, t_4\}$. El error de éstos puede ser calculado como $e = (o_1 - t_1)^2 + (o_2 - t_2)^2 + (o_3 - t_3)^2 + (o_4 - t_4)^2$, dado que el error se calcula en base a los pesos de la entrada de cada capa en una red neuronal; para cada entrada local $\{c_1, c_2, c_3, c_4, c_5\}$ el error es un conjunto de perceptrones $e = E(c_1, c_2, c_3, c_4, c_5)$. La gradiente del vector en cada entrada local se calcula como la derivada con respecto al peso de la entrada, dando como resultado

un vector $\{\delta c_1, \delta c_2, \delta c_3, \delta c_4, \delta c_5\}$ conocido como sensibilidad. En cada entrada del algoritmo los valores del set de entrada no son iguales. Luego de procesarlos, la salida tiene una perturbación que se debe añadir a la sensibilidad. Este proceso se repite para las capas precedentes $\{p_1, p_2, p_3, p_4\}$. Es necesario notar que las funciones de activación son importantes para actualizar los pesos de las conexiones.

En la Figura 3 se presenta como $\delta p_1 \left(\rightarrow \frac{\partial e}{\partial p_1} \right)$ puede ser calculada con la entrada p_1 que genera una cantidad infinitesimal Δp_1 , dando como resultado un cambio en toda la red. Mediante la repetición se permite trabajar hacia atrás y calcular la sensibilidad del error en base a los cambios de la entrada en cada capa. Para generalizar, la expresión de la sensibilidad de la capa anterior (l) en términos de la sensibilidad de la capa actual ($l + 1$) se muestra en la Ecuación 6:

$$\delta p_i^l = S'(p_i^l) \sum_j w_{ij}^{(l+1)} \delta c_j^{(l+1)},$$

Ecuación 6. Sensibilidad de aprendizaje.

Donde $S'(p_i)$ es la derivada de la función evaluada en p_i , dado que el error cambia con respecto a pequeñas alteraciones en los pesos de las conexiones de entrada, se deduce que el error con respecto al cualquier peso es:

$$\frac{\partial z}{\partial w_{ij}} = S(p_i) \delta c_j,$$

Ecuación 7. Derivada general del error con respecto a cualquier peso.

La Ecuación 7 interactivamente puede actualizar los pesos y minimizar el error usando el método de *steepest descent* [26], [28].

En resumen, el algoritmo computacional de *back-propagation* es [29], [30]:

1. Inicializar los pesos para pequeños valores aleatorios.
2. Seleccionar aleatoriamente una muestra del set de entrada y su respectiva salida.
3. Calcular la entrada y salida de cada capa y la salida de la capa final.
4. Calcular la sensibilidad.
5. Calcular la gradiente de las componentes y actualizar los pesos.
6. Regresar al paso 2 y repetir para la siguiente entrada hasta que el error de salida sea menor al umbral o se haya alcanzado el número máximo de iteraciones.

Validación

Una vez aprendidos los parámetros del modelo en un entrenamiento, se realiza una validación del modelo con otro set de datos. La mayoría de los métodos de validación sobre el rendimiento de un modelo de una red neuronal son métodos basados en el error estadístico. Los criterios que evalúan el rendimiento del modelo son basados en pruebas de validación gráfica y/o numérica [31].

Los modelos más comunes se basan en: método de Holdout, técnicas de sub-muestreo y separación *three-way* [32]. Un método de validación muy usado en el entrenamiento de modelos es *cross-validation*. Este se basa en un submuestreo y consiste en tomar un conjunto de evaluación o validación. Este conjunto se divide en conjunto más pequeños y son procesados en el modelo dando como resultado varios datos de clasificación, los cuales se pueden asociar mediante la media aritmética y para tener una precisión del modelo [33].

1.1.3. Redes Feed-forward multicapa

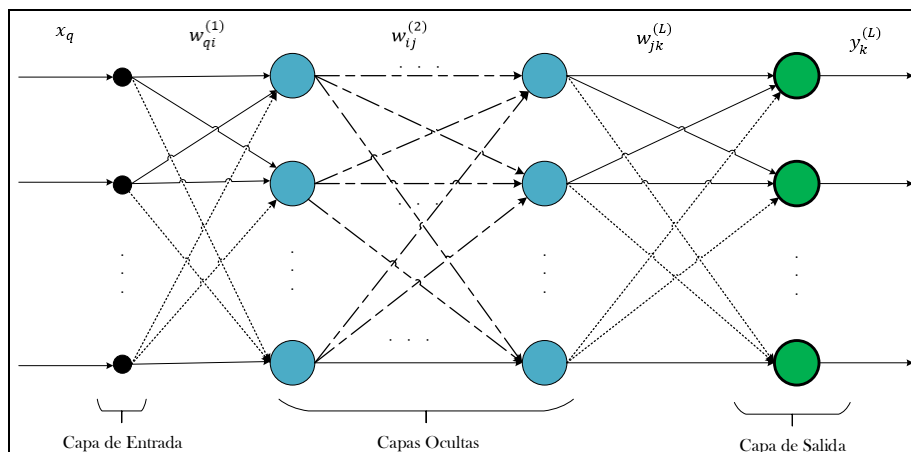


Figura 4. Arquitectura de una red neuronal artificial feed-forward.

La Figura 4 muestra un perceptrón típico de 3 capas. En general las redes tienen L capas: la capa de entrada, $L - 1$ capas ocultas y la capa de salida de unidades total o localmente conectadas. Las redes neuronales que se presentan a continuación tienen el esquema de *feed-forward*. Esto significa que no posee conexiones recursivas entre sus capas.

Perceptrón Multicapa

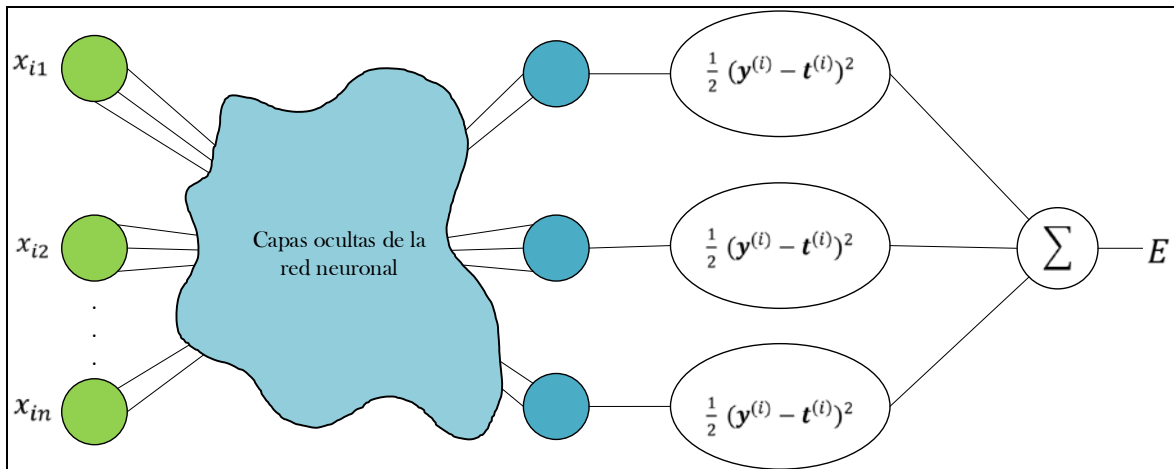


Figura 5. Cálculo del error en una ANN.

Una de las clases más comunes de redes multicapa *feed-forward* son los perceptrones multicapa. En su arquitectura principalmente usan una función umbral o una función sigmoide. Su principal característica es que pueden generar modelos de decisión muy complicados y representar cualquier función tipo booleana [34].

Un perceptrón que clasifica sus entradas en m clases tiene el siguiente procedimiento: Dado un vector de pesos $w_{ij}^{(l)}$ para conexiones entre la i -ésima unidad en la capa $(l - 1)$ hasta la j -ésima unidad en la capa l . Éste vector tiene un set de muestras de entrenamiento $p = \{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), \dots, (\mathbf{x}^{(p)}, \mathbf{t}^{(p)})\}$, donde la entrada $\mathbf{x}^{(i)} \in \mathbb{R}^n$ es el vector en el espacio n -dimensional del conjunto de muestras. Su salida $\mathbf{t}^{(i)} \in [0,1]^m$ es el vector esperado para cada muestra en un espacio m -dimensional. Como ejemplo de este proceso se presenta la Figura 5.

La función de coste del error cuadrático asociado al proceso anterior se define como:

$$E = \frac{1}{2} \sum_{n=1}^p \|\mathbf{y}^{(i)} - \mathbf{t}^{(i)}\|^2$$

Ecuación 8. Función de coste del error cuadrado.

El algoritmo de *back-propagation* es un método que garantiza minimizar la función de coste del error cuadrado como se mostró en la sección (1.1.2). Además de los resultados de Ecuación 8 se analiza de forma general los resultados del entrenamiento con la interpretación obtenida [23], [27].

1.1.4. Aprendizaje Profundo – *Deep Learning*

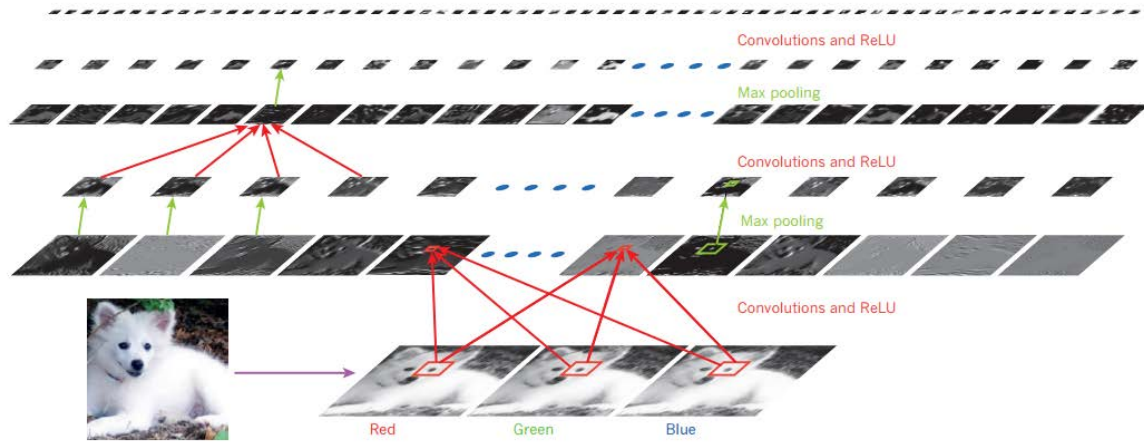


Figura 6. Ejemplo de *Deep Learning*.

El aprendizaje profundo o *Deep Learning* se refiere a un aspecto muy particular del aprendizaje de máquina o en inglés *Machine Learning*. Además, se relaciona con los modelos computacionales que se componen de múltiples capas de procesamiento para aprender representaciones de datos con múltiples niveles de abstracción. Estos modelos toman una entrada compleja y por cada capa el procesamiento genera nueva información cada vez de un nivel más abstracto; de manera que las transformaciones realizadas en las capas permiten aprender funciones muy complejas [35].

El *Deep Learning* se enfoca principalmente en tareas de clasificación. Esto es debido a que las capas superiores amplifican aspectos de la entrada que son importantes para la clasificación y suprimen las variaciones irrelevantes de la misma. Su arquitectura es una pila multicapa de módulos simples. Cada uno de los módulos está sujeto a aprender y realizan transformaciones de su entrada para aumentar la selectividad y la invarianza. Por ejemplo, un *Deep Learning* de profundidad 10 implementa funciones tan complejas que le permite diferenciar pequeños detalles, por ejemplo diferenciar entre los tigres y los ligres¹. Estas funciones son insensibles a grandes variaciones como el fondo, la iluminación y los objetos circundantes [30].

Una de las ventajas más importantes del *Deep Learning* es generar aprendizaje de máquina usando procedimientos de aprendizaje de propósito general. Por ejemplo: si se desea extraer las características de una entrada para generar un clasificador una opción es usar métodos de *kernel* para la extracción de características [36]. Las cuales solo

¹ Un ligre es un felino híbrido, producto del cruce de un león y una tigresa

funcionarán para entradas muy similares. Por otra parte, la presencia de varianza en la misma entrada generará un aprendizaje diferente. Sin embargo, frente a este problema las técnicas de *Deep Learning* como las Redes Neuronales Convolucionales son más robustas para extraer características de entradas muy complejas. Un ejemplo del proceso de *Deep Learning* se observa en la Figura 6 [33].

1.1.5. Aplicaciones y Usos de la Red Neuronal

Actualmente existe una gran variedad de modelos de ANN [37] y algoritmos de aprendizaje [38]. Todos pueden ser usados para resolver los problemas en las siguientes categorías [24]:

1. **Reconocimiento de patrones:** El reconocimiento de patrones es la asignación de una etiqueta a un valor de entrada. Algunas aplicaciones típicas de técnicas de reconocimiento de patrones son: el reconocimiento automático, la detección autónoma de números manuscritos en códigos postales [39], el reconocimiento automático de rostros [Lawrence], detección de texto manuscrito [40], entre otros [41] y [42].
2. **Clustering / Categorización:** Es conocido como un clasificador de patrones no supervisado; la categorización es el proceso en el cual las ideas y objetos son reconocidos, diferenciados y entendidos. Esto implica que los objetos sean agrupados en categorías, usualmente para un propósito específico [43]. Las aplicaciones de *clustering* incluyen minería de datos [44] y compresión de data [45].
3. **Funciones de aproximación:** Comprenden tareas de aproximación. Matemáticamente, dado un set de muestras de entrenamiento con ruido etiquetadas $\{(x_1, y_1), \dots, (x_n, y_n)\}$ que han sido generadas desde una función $\mu(x)$, se encuentra la estimación $\hat{\mu}$ de una función desconocida μ . En esta categoría hay varios problemas de modelamiento que requieren funciones de aproximación [46].
4. **Predicción / Forecasting:** La predicción es el proceso de relatar un echo futuro en el presente y tiene un impacto significativo en los procesos de decisión de negocios [47], ciencia e ingeniería [28]. Ejemplos de esto son: el estado futuro de la bolsa de valores y el estado futuro del clima [48].

5. **Optimización:** En el mundo existe una gran variedad de problemas en matemática, ingeniería, medicina, economía y otros campos que pueden ser propuestos como problemas de optimización. El objetivo de un algoritmo de optimización es encontrar una solución que satisfaga la mayor cantidad de restricciones. Entre los problemas más comunes a los que se busca optimización son los *NP-complete* como, por ejemplo: *The Traveling Salesman Problem* [49] y [50].
6. **Content-addressable memory:** En el modelo computacional de Neumann, un espacio en memoria es independiente de los demás y solo puede ser accedido por una única dirección de memoria. Por otra parte, en un modelo con memoria asociativa o *content-addressable memory* se puede acceder por el contenido de la memoria. Cabe recalcar que el contenido en memoria puede estar desordenado. Una memoria asociativa es lo más óptimo para construir bases de datos de información multimedia [51] y [52].
7. **Control:** Los modelos de control consideran un sistema dinámico $\{x(t), y(t)\}$ donde $x(t)$ es la entrada del control y $y(t)$ es el resultado de la salida del sistema en un tiempo t . El objetivo es generar un control de entrada $x(t)$, de forma que el sistema siga una trayectoria esperada, la cual está determinada por el modelo de referencia. Un ejemplo de esto es: "*Engine idle speed control system*" [51].

1.2. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales o por su nombre en inglés *Convolutional Neural Network* que se abrevia como ConvNet o CNN [53], [54]. Son arquitecturas multicapa, compuestas de múltiples etapas. Este tipo de arquitectura se ha usado muchas veces en modelos neuronales para aprendizaje visual. Mediante las capas de la red neuronal se puede extraer características visuales elementales como: bordes orientados, puntos finales, esquinas, entre otras [55].

En una red neuronal convolucional, con campos receptivos en diferentes lugares de la imagen, genera diferentes mapas de características. La implementación secuencial de esta idea es escanear la imagen de entrada con una sola neurona que tenga un campo receptivo local, y almacenar los estados de esta neurona en las ubicaciones correspondientes en el mapa de características. Esta operación es equivalente a una convolución seguida de una función de aplastamiento; en literatura inglesa comúnmente se conoce como *squashing*

function. Este proceso se puede realizar en paralelo implementando el mapa de características como un plano de neuronas que comparten un solo vector de peso. Las neuronas en un mapa de características realizan la misma operación en diferentes partes de la imagen. Varios mapas de características generalmente componen una capa convolucional. Si se tiene diferentes vectores de peso en las conexiones se puede extraer múltiples características de cada ubicación [56].

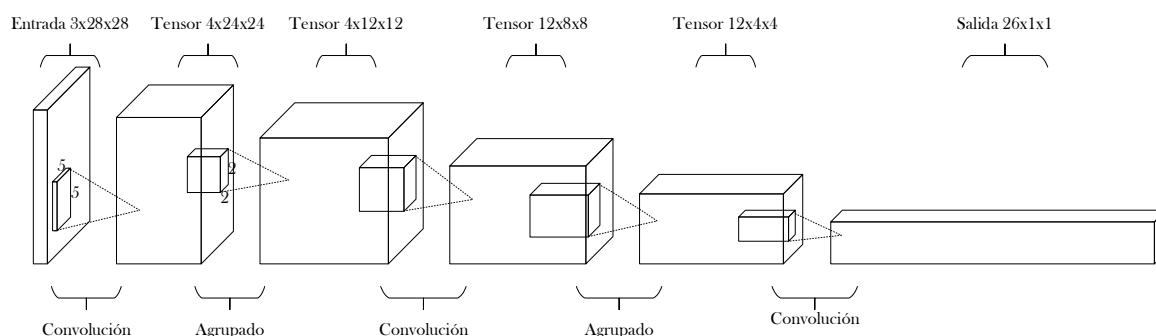


Figura 7. Ejemplo del proceso de una Red Neuronal Convolucional.

Por ejemplo: la primera capa oculta en la Figura 7 tiene cuatro mapas de características con campos receptivos de 5x5. El cambio de la entrada de una capa convolucional cambia la salida. Una vez que se detectó una característica, su ubicación exacta pierde importancia, siempre que se conserve su posición aproximada con respecto a otras características. Consecuentemente, a cada capa convolucional le sigue una capa adicional que realiza un promediado local y una agrupación. De esta forma se reduce la resolución del mapa de características y por lo tanto se reduce la sensibilidad de la salida a cambios y distorsiones. La segunda capa oculta realiza promedios y agrupaciones de 2x2, seguidos por un peso entrenable, un sesgo (*bias*) entrenable y una operación de rectificación lineal.

El peso y el sesgo controlan el efecto de la no linealidad. Las capas sucesivas de convoluciones y agrupación suelen alternarse, dando como resultado un modelo bpiramidal en cada capa. El número de mapas de características aumenta a medida que disminuye la resolución espacial.

La combinación de convolución y agrupación, inspirada en las nociones de Hubell y Wiesel de células visuales "simples" y "complejas", fue implementado en el Modelo *Neocognitron* por Fukushima. Éste es una red neuronal jerárquica multicapa capaz del reconocimiento de robustos patrones visuales a través de un proceso de aprendizaje [57].

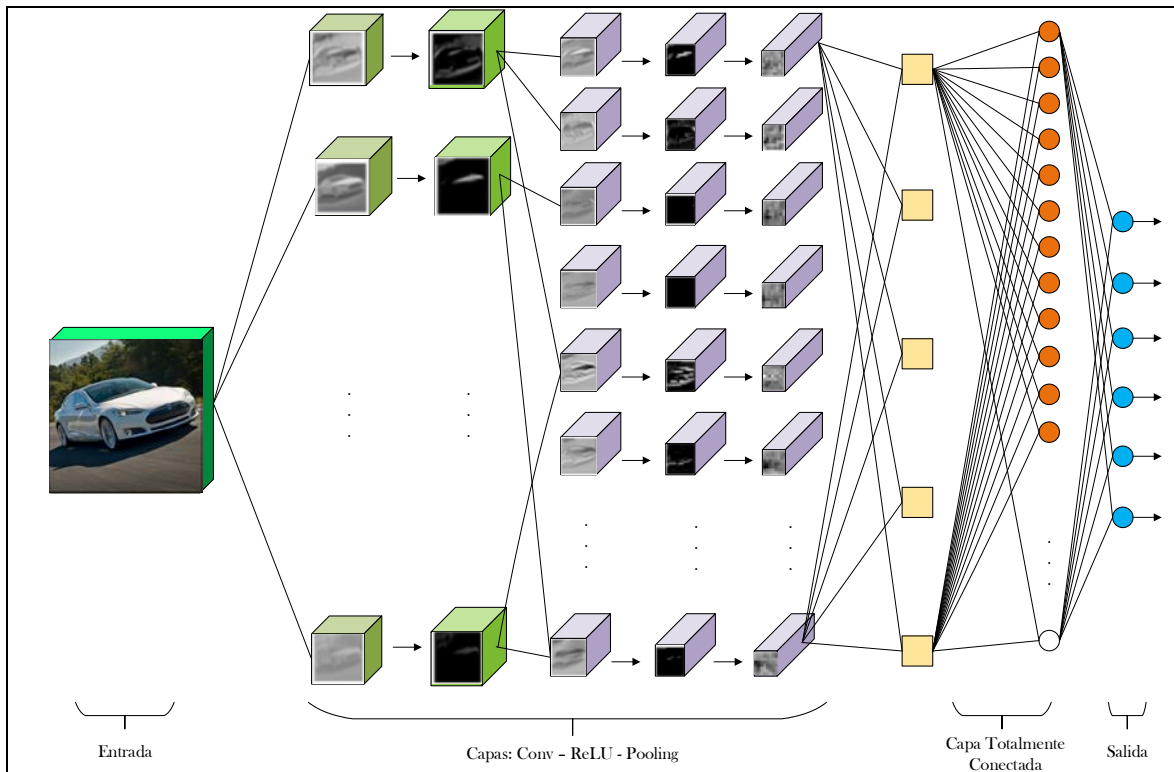


Figura 8. Modelo de una arquitectura típica de una ConvNet.

Como se presentó en la sección 1.1.2, los pesos de las conexiones de la red se aprenden con algoritmos de *back-propagation*. Las redes neuronales convolucionales sintetizan su propio extractor de características y su técnica de compartir peso reduce el número de parámetros. Ésto reduce los procesos computacionales y mejora su capacidad de generalización [58]. La cantidad de pesos que puede tener una ConvNet depende de las conexiones que sean necesarias para procesar todos los mapas de características. El ejemplo de la ConvNet de la Figura 8 contiene alrededor de 100,000 conexiones, pero solo alrededor de 2.600 parámetros debido al reparto de peso. Este tipo de redes, al ser comparadas con otro tipo de técnicas de redes neuronales artificiales, presentan mejores resultados. Entre las aplicaciones más conocidas tenemos las tareas de reconocimiento de caracteres escritos a mano [59], reconocimiento facial [60], entre otras [61].

1.2.1. Arquitectura de una Red Neuronal Convolucional

Algunas arquitecturas recientes de las ConvNet tienen de entre 8 a 200 capas, miles de millones de pesos y billones de conexiones entre cada neurona [62]. Esto provoca que el entrenamiento tarde meses en ser completado. Sin embargo, en los últimos años se redujo debido a hardware [63], software [64], [65] y algoritmos de paralelización [66], [67].

La arquitectura de una ConvNet como se muestra en la Figura 9 es similar a la de una red neuronal artificial. Una red neuronal recibe una entrada, ésta es transformada a través de una serie de capas ocultas. Cada capa oculta se compone de un conjunto de neuronas, donde cada neurona está totalmente conectada a todas las neuronas en la capa anterior, donde se calcula una única función que a su vez es independiente de las demás operaciones. La última capa totalmente conectada (*fully-connected*) se denomina capa de salida y en la configuración de clasificación que representa las puntuaciones de clase.

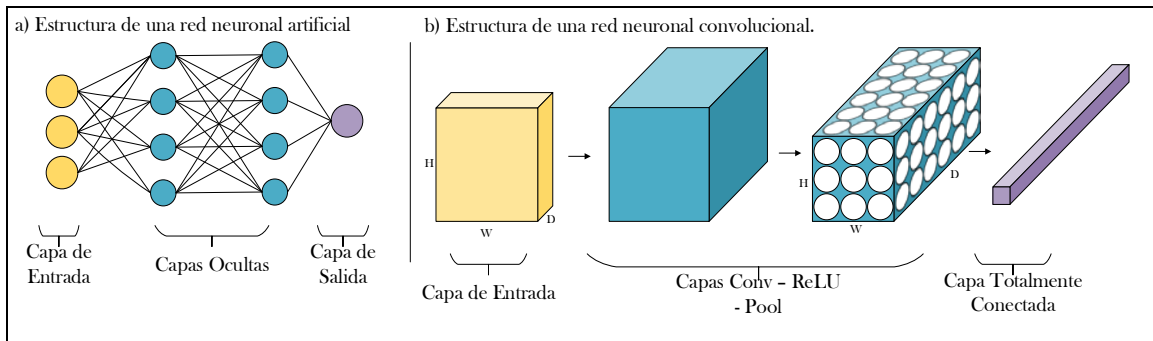


Figura 9. Comparación de la arquitectura de Redes Neuronales Artificiales y Convolucionales.

Las redes neuronales no tienen buena escalabilidad para imágenes complejas. En CIFRAR-10, las imágenes tienen un tamaño de $32 \times 32 \times 3$ (una matriz de 32×32 con 3 canales, usualmente son RGB). En este caso una sola neurona *fully connected* en la primera capa oculta de una red neural tendría $32 \times 32 \times 3 = 3\,072$ pesos a ser entrenados. De forma similar, si se tiene imágenes de entrada de mayor tamaño. Por ejemplo: una imagen en alta definición de un celular móvil promedio tiene un tamaño de $1280 \times 720 \times 3$. En este caso la primera neurona *fully-connected* tendrá $1280 \times 720 \times 3 = 2\,764\,800$ pesos a ser entrenados [68]. Para resolver este problema las Redes Neuronales Convolucionales aprovechan que las entradas son imágenes porque tienen operaciones que simplifican el entrenamiento de la red neuronal. A diferencia de una Red Neuronal, las capas de la ConvNet son analizadas en 3 dimensiones: ancho, alto y profundidad. Este tipo de vectores tridimensionales son tensores de tercer orden. Por ejemplo: para las imágenes de entrada de CIFRAR-10, las neuronas que toman estas entradas están conectadas a una pequeña región de la entrada de la ConvNet.

En una CNN, la capa *fully-connected* es un tensor de $1 \times 1 \times 10$, este es un vector de características de 10 valores. El proceso se puede observar en el literal b) de la Figura 9.

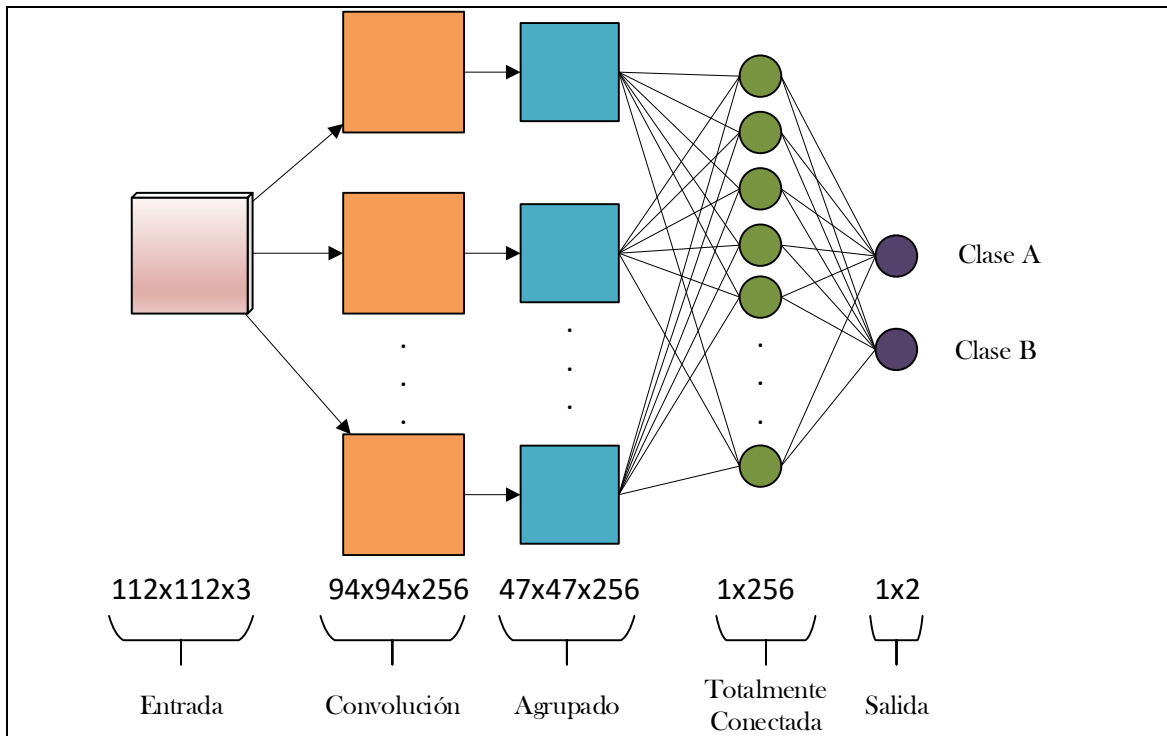


Figura 10. Arquitectura Básica de una ConvNet.

Como se observa en la Figura 10, la ConvNet es una secuencia de capas, cada capa transforma el volumen de la entrada mediante una función diferenciable. Las principales capas de la ConvNet son:

- Entrada.
- Capa convolucional (*Convolutional Layer*).
- Capa agrupación (*Pooling Layer*).
- Capa totalmente conectada (*Fully-connected Layer*).
- Salida.

A continuación, se presenta una descripción de cada capa de las ConvNets

1.2.1.1. Entrada

La entrada para la ConvNet es un set de imágenes. Este set de datos tiene H filas, W columnas y D Canales del color. Los canales dependen del tipo de codificación de la imagen [69]. Considerando que: "...mientras más atributos existan es más probable llegar a un modelo más exacto" [70] y las experiencias realizadas en CIFRAR [71], Kaggle [72], Yahoo Research [73], SNAP [74], AWS [75], Texas University [76]. En general se aconseja tener una fuente masiva de datos de entrenamiento, y en caso de no tenerlas se recomienda realizar un proceso de *data augmentation*. Este proceso consiste en realizar

transformaciones, recortes, cambios de brillo, desenfoques, cambios de saturación, y otro tipo de efectos sobre las imágenes de entrada [71], [77].

Un set de datos, en resumen, es el conjunto de información con el que va a trabajar la red. Si se parte de un único set de datos, éste debe ser dividido en [78]:

- **Set de entrenamiento:** Es un set de muestras usadas para el aprendizaje del modelo, éste es el entrenamiento de los parámetros del modelo.
- **Set de validación:** Es un set de muestras usado para afinar los parámetros del clasificador
- **Set de pruebas:** Es un set de muestras usado solo para asegurar el rendimiento de un clasificador que ya se encuentra modelado.

Partiendo de los tipos de set de datos nombrados anteriormente, se realiza una separación del set de entrada. Por ejemplo, un 70% corresponderá al conjunto de entrenamiento y el 30% restante corresponderá al conjunto de pruebas. El motivo de la separación es debido a que no se debe evaluar la calidad del modelo sobre los mismos datos que sirvieron para construirlo. Lo que se busca mediante esta separación es estimar el error de generalización del entrenamiento y evitar que exista un *overfitting* o un *underfitting* [79], [80].

Limpieza de set de imágenes

La creación de un set de datos que proveniente de varias fuentes causa problemas para la generación de un buen modelo. Esto es debido a que existen muestras que no corresponden a las clases definidas para un determinado proceso de entrenamiento. Las muestras que no corresponden a las clases definidas se les conoce como ruido [81]. Por ejemplo: considérese a un set con cuatro clases de imágenes que corresponden a bosques en las estaciones del año (invierno, verano, otoño y primavera). Si al set de datos ingresan imágenes de automóviles o casas en la clase de verano, estas imágenes no corresponden a dicha clase y por lo tanto son una presencia muy clara de ruido. Por inferencia, al realizar el entrenamiento con un set de datos con ruido se provocaría la generación de un modelo erróneo.

Para identificar la presencia de ruido en las clases de un set de datos existe varios métodos. Uno de los métodos más sencillos se basa en la representación de forma gráfica del set de entrada y a partir de observaciones se puede determinar que parte del set de entrada no tiene los parámetros referentes a las clases. Una vez detectadas las regiones que no corresponden a los patrones de las imágenes se puede realizar una eliminación. Luego se comprueba si dicha eliminación es correcta [82], [83].

Formato de Entrada del Set de Datos

Los formatos de entrada que se usan en las CNNs tienen el formato $B \times H \times W \times D$ o $B \times D \times H \times W$ [84], [85], [71]. Donde B es la cantidad de imágenes que tienen $H \times W \times D$ o $D \times H \times W$. Uno de los formatos más usado en los sets de imágenes es el presentado por MNIST [86]. El proceso consiste en crear dos archivos: el primero contiene información relacionada con el tensor de imágenes del tipo $B \times H \times W \times D$ y un segundo archivo que tiene las etiquetas que corresponden a cada tensor de imagen guardado. Por ejemplo, para un set de datos de autos. Si el primer elemento es un auto, éste será guardado como el elemento $(1 \times H \times W \times D)$ del set y su etiqueta será $(1 \times T)$. Para el segundo elemento será $(2 \times H \times W \times D)$ con su etiqueta es $(2 \times T)$. Generalizando sería para la i -ésima imagen guardada en el tensor $(B_i \times H \times W \times D)$ tiene su etiqueta correspondiente en el tensor $(i \times T)$.

1.2.1.2. Capa de Convolución

La capa de convolución se la denomina comúnmente por su nombre en inglés *Convolutional Layer* y se le abrevia como *Conv Layer*. Esta capa es el núcleo de la Red Neuronal Convolutiva debido a que en ella se realiza un trabajo de alto rendimiento a nivel computacional.

La convolución es una operación matemática que se realiza sobre 2 funciones para producir una tercera, la cual es una versión modificada de una de las funciones originales. Se expresa como: dadas las entradas $g(x)$ y $h(x)$ se debe producir una salida $y(x)$, donde $y(x) = g(x) * h(x)$. Como se necesita que las funciones de entrada determinen un valor para la nueva función $y(x)$ cada cálculo de $g(x)$ y $h(x)$ produce desplazamientos en la respuesta $y(x)$. Estos desplazamientos se calculan con separaciones infinitesimales. El resultado de este proceso es la suma de todas las respuestas individuales. Para el cálculo de la convolución se usa la siguiente fórmula:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

Ecuación 9. Modelo Convolutiva.

La fórmula de la convolución se escribe como $f * g$ y describe el peso promedio de la función $f(\tau)$, en el momento t donde el peso desplazado por t es dado por $g(-\tau)$ [87].

Convolución en Imágenes

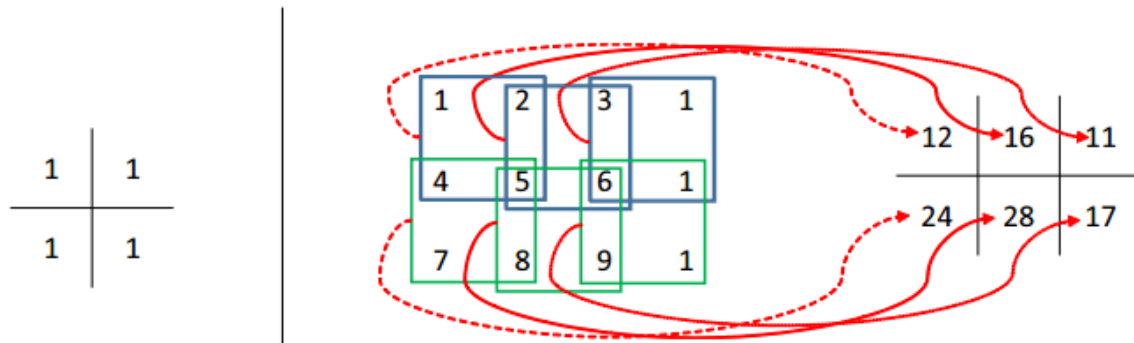


Figura 11. Ejemplo de convolución en matrices.

La convolución en las CNNs toma el mismo principio matemático de las convoluciones. Su aplicación en imágenes se describe a continuación: dada una imagen de entrada con un solo canal $3 \times 4 \times 1$, se le aplicará un filtro (kernel convolucional) de 2×2 , como se puede observar en la Figura 11. Al superponer el kernel de convolución en la parte superior de la imagen de entrada se usa la Ecuación 9. Se inicia calculando el producto entre los números en la misma ubicación. Luego, se suma los productos para obtener un solo número de respuesta.

El ejemplo para la Figura 11 es: al sobreponer el kernel con la función de entrada, el resultado es $(1 \times 1) + (1 \times 4) + (1 \times 2) + (1 \times 5) = 12$, al moverse al siguiente pixel el cálculo de la convolución es $(1 \times 2) + (1 \times 3) + (1 \times 5) + (1 \times 6) = 16$ de forma secuencial se continúa realizando convoluciones sobre el tensor de entrada para obtener como salida una matriz de tamaño $2 \times 3 \times 1$.

Hiperparámetros de la Convolución

Tres hiperparámetros controlan el tamaño del volumen de salida:

1. **Profundidad (depth)** Representa el volumen de salida y es el número de kernels que se usarán. En una capa de convolución se usan múltiples filtros de convolución, si se asume que se usan D kernels para tensores de entrada de $H \times W$, la cantidad total de kernels usados se denota como $\mathbf{k} \in \mathbb{R}^{(H \times W \times D^l \times D)}$. Un elemento específico se obtiene con $0 \leq i < H, 0 \leq j < W, 0 \leq d < D^{(l)}$.
2. **Salto (stride)** Representa el tamaño o la zancada con la cual se deslizará el filtro. Cuando el salto es 1, se mueven los filtros, un pixel a la vez como se puede observar en la Figura 11. Para una entrada, si se considera que el salto es de 1 y no existe

relleno, se tiene luego de procesar una entrada x^l , la salida $y \in \mathbb{R}^{H^{l+1} \times W^{l+1} \times D^{l+1}}$ con $H^{l+1} = H^l - H + 1, W^{l+1} = W^l - W + 1, D^{l+1} = D$.

3. **Zero-relleno (zero-padding)** Permite llenar con ceros alrededor de la frontera de la imagen de entrada, esto sirve para controlar el tamaño de las salidas. Por ejemplo: si se desea que la entrada y salida tengan el mismo tamaño después de una convolución; una entrada $H^l \times W^l \times D^l$ con tamaño del kernel $W \times W \times D^l \times D$, la convolución resultante tendrá la entrada $(H^l - H + 1) \times (W^l - W + 1) \times D$. Para cada canal de entrada si se rellena con $\lfloor \frac{H-1}{2} \rfloor$ filas sobre la primera fila y $\lfloor \frac{H}{2} \rfloor$ debajo de la última fila, y con $\lfloor \frac{W-1}{2} \rfloor$ columnas a la izquierda de la primera columna y $\lfloor \frac{W}{2} \rfloor$ a la derecha de la última columna. De forma que la salida será $H^l \times W^l \times D^l$.

Con todos estos hiperparámetros la Ecuación 9 aplicada a imágenes se la escribe de la siguiente forma:

$$y_{i^{(l+1)}, j^{(l+1)}, d} = \sum_{i=0}^H \sum_{j=0}^W \sum_{d^l=0}^{D^l} f_{i,j,d^l,d} \times x_{i^{(l+1)+1, j^{(l+1)+j}, d^l}^{(l)}, \forall \{0 \leq d \leq D = D^{(l+1)}\};$$

Ecuación 10. Procedimiento de convolución computacional.

para cualquier lugar espacial $(i^{(l+1)}, j^{(l+1)})$ que satisfaga la condición $0 \leq i^{(l+1)} < H^{(l)} - H + 1 = H^{(l+1)}, 0 \leq j^{(l+1)} < W^{(l)} - W + 1 = W^{(l+1)}$. En la Ecuación 10 el término $x_{i^{(l+1)+1, j^{(l+1)+j}, d^l}^{(l)}$ representa a x^l indexado por $i^{(l+1)} + 1, j^{(l+1)} + j, d^l$.

Para el calcular el tamaño espacial de la salida de una función de volumen (W), el campo respectivo de las neuronas de convolución (F), los saltos aplicados (S) y el relleno agregado (P) sobre el borde. El cálculo del número de neuronas a entrenar viene dado por la fórmula [88].

$$\Lambda = \frac{W - F + 2P}{S} + 1$$

Ecuación 11. Número de neuronas a entrenar.

Intercambio de parámetros

La técnica de intercambio del parámetro se utiliza en capas convolucionales para controlar el número de parámetros con lo que se va a operar. Por ejemplo, un tensor $45 \times 45 \times 86$ tiene un total de 174150 neuronas en la primera capa de convolución, cada una tiene $11 \times 11 \times 3 = 363$ pesos y con sesgo 1. En conjunto suman $174150 \times 364 = 63\,390\,600$ parámetros en la primera capa de la ConvNet. Para reducir esta cantidad de parámetros, se usa un salto de 2 dimensiones de profundidad. Limitando a las neuronas a usar los mismos pesos y sesgo dando como resultado un total de $86 \times 11 \times 11 \times 3 = 31218$ pesos. Esta cantidad se descompone en 31132 parámetros más 86 sesgos [89].

Algoritmo de la capa Convolutiva

Todas las capas convolucionales de una CNN en general siguen el algoritmo que se encuentra descrito a continuación:

1. Toma una entrada $\mathbf{x}^{(l)}$ ó $(H^{(l)} \times W^{(l)} \times D^{(l)})$
2. Se aplican los hiperparámetros necesarios:
 - 2.1. kernel convolutiva K .
 - 2.2. Extensión espacial F .
 - 2.3. El salto S .
 - 2.4. Zero-Relleno P .
3. Produce una salida de volumen $\mathbf{y}^{(l)} = \mathbf{x}^{(l+1)}$ ó $(H^{(l+1)} \times W^{(l+1)} \times D^{(l+1)})$ donde:
 - 3.1. $W^{(l+1)} = \frac{W^{(l)} - F + 2P}{S} + 1$
 - 3.2. $H^{(l+1)} = \frac{H^{(l)} - F + 2P}{S} + 1$
 - 3.3. $D^{(l+1)} = K$
4. Con los parámetros compartidos, esto introduce pesos $F * F * D^{(l)}$ para un total de $(F * F * D^{(l)}) * K$ pesos and K sesgos.
5. En el volumen de salida, las d -ésimas salto de tamaño $(H^{(l+1)} \times W^{(l+1)})$ es el resultado de entrenar una convolución valida de el d -ésimo filtro sobre la entrada de volumen con el sesgo S , y compensado por el d -ésimo sesgo.

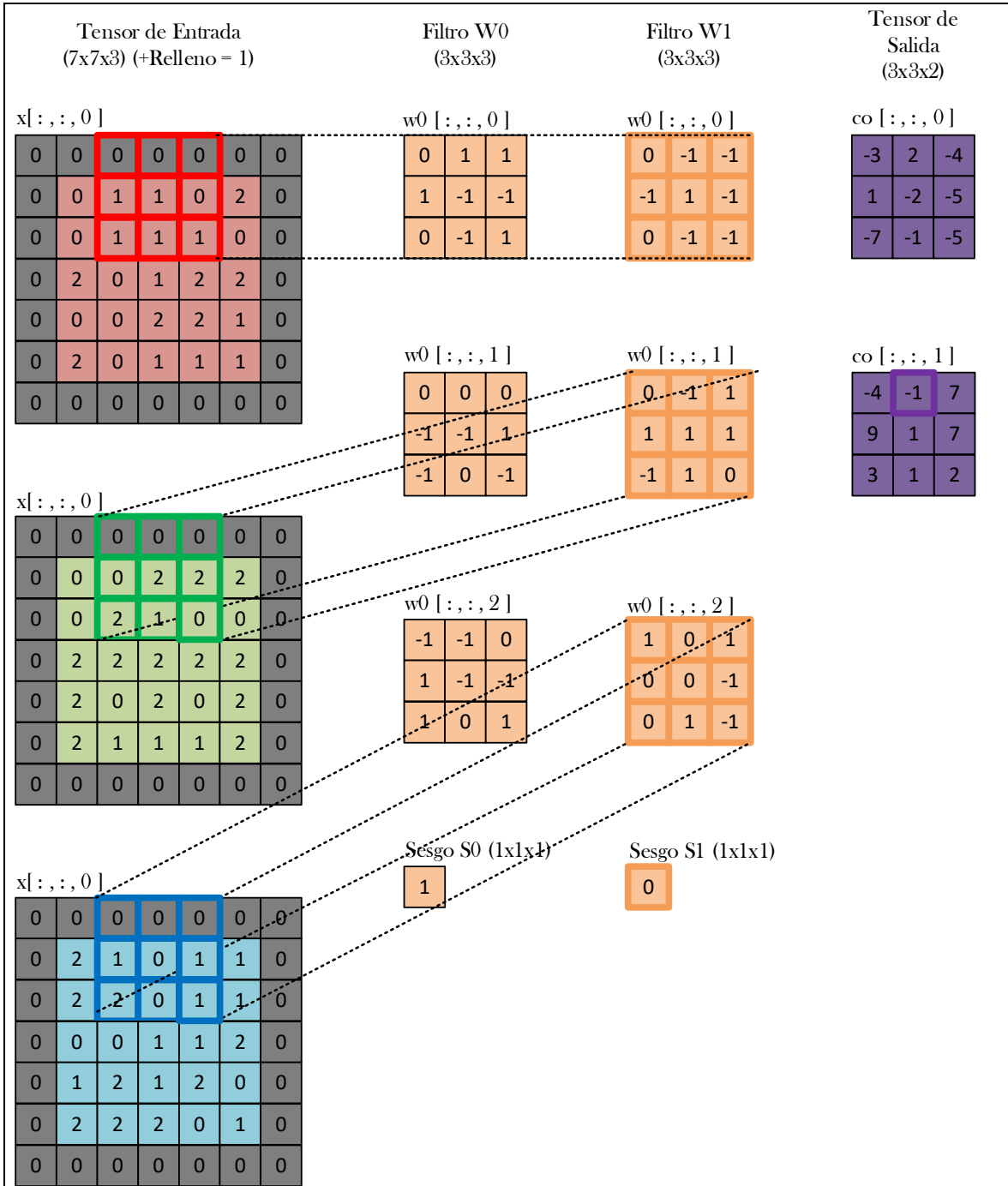


Figura 12. Proceso de la capa de convolución.

A continuación, en la Figura 12, se muestra un ejemplo de convolución con los siguientes hiperparámetros $F = 3$ $S = 1$ ó 2 $P = 1$. Partiendo con el tensor $5 \times 5 \times 3$ que se observa de forma separada a la izquierda de la Figura 12. Para una convolución, el volumen de entrada es $W_1 = 5$, $H_1 = 5$, $D_1 = 3$, y los parámetros de la capa Conv son $K = 2$, $F = 3$, $S = 2$, $P = 1$. Es decir, dos filtros de tamaño de 3×3 que se aplican con un salto de 2. Esto genera un volumen de salida con tamaño de $\frac{(5 - 3 + 2)}{2} + 1 = 3$.

Además, el relleno de $P = 1$, hace que el borde exterior de la entrada sea relleno con ceros. A continuación, se itera sobre las activaciones de salida, cada elemento se calcula multiplicando los elementos de la entrada resaltada con el filtro (naranja), sumando hasta recorrer todo el tensor de entrada. La salida de este proceso es un tensor de $3 \times 3 \times 2$

Back-propagation para actualizar los parámetros

En la sección 1.1.2 se presenta el algoritmo de *back-propagation*. Para ello se calcula la derivada $\frac{\partial z}{\partial \text{vec}(x^{(l)})}$ que se usará en la *back-propagation* para las capas anteriores ($l - 1$) y la derivada $\frac{\partial z}{\partial \text{vec}(w^l)}$ en la capa actual. Éstas derivadas sirven para determinar que parámetros serán actualizados. Por lo tanto, para realizar la *back-propagation* en la capa de convolución se debe resolver la siguiente ecuación:

$$\frac{\partial z}{\partial w^{(l)}} = \phi(x^{(l)})^T \frac{\partial z}{\partial x^{(l+1)}}$$

Ecuación 12. Back-propagation en ConvNet

Donde $\phi(x^{(l)})$ es la expansión vectorizada de $x^{(l)}$ [88].

1.2.1.3. Capa de Agrupado

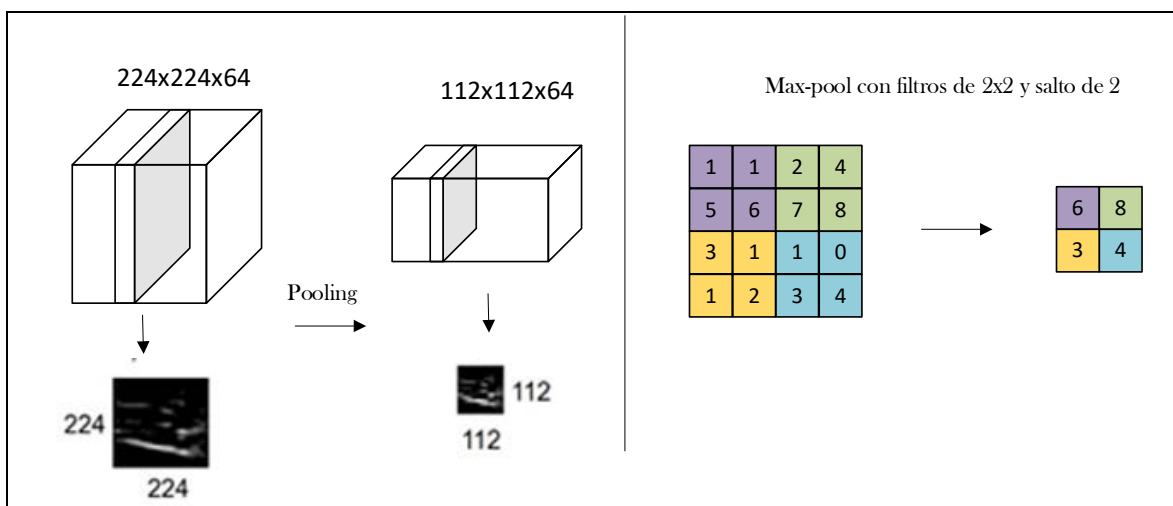


Figura 13. Ejemplo de Pooling.

La capa de agrupado se la denomina comúnmente por su nombre en inglés *Pooling Layer*; se le abrevia como capa de *pooling*. Una capa de *pooling* no necesita parámetros de entrenamiento. La dimensión espacial de la capa ($W \times H$) luego de ser procesada por la capa de agrupado, generará otro tensor de salida con un tamaño espacial menor con el fin

de reducir la cantidad de parámetros y la computación realizada en la CNN, si se asume que:

$$H^{(l+1)} = \frac{H^{(l)}}{H}, W^{(l+1)} = \frac{W^{(l)}}{W}, D^{(l+1)} = D^{(l)},$$

una capa de *pooling* opera sobre la entrada $x^{(l)}$ canal por canal de forma independiente. Ente cada canal, la matriz con los elementos $(H^{(l)} \times W^{(l)})$ están siendo divididos en subregiones no superpuestas $(H^{(l+1)} \times W^{(l+1)})$; cada región de tamaño $(H \times W)$. El operador de pooling mapea cada subregión en un número [88].

Los tipos de operadores de agrupado más usados son el *max pooling* y el *average pooling*. En el *max pooling* el operador de agrupado mapea una subregión con su máximo valor, mientras que en el *average pooling* se mapea una subregión con su valor promedio matemáticamente se representa en las Ecuación 13 y Ecuación 14 [88]:

$$y_{i^{(l+1)}, j^{(l+1)}, d} = \max_{0 \leq i < H, 0 \leq j < W} x_{i^{(l+1)} \times H + i, j^{(l+1)} \times W + j, d}^{(l)}$$

Ecuación 13. Modelo de max pooling.

$$y_{i^{(l+1)}, j^{(l+1)}, d} = \frac{1}{HW} \sum_{0 \leq i < H, 0 \leq j < W} x_{i^{(l+1)} \times H + i, j^{(l+1)} \times W + j, d}^{(l)}$$

Ecuación 14. Modelo de average pooling.

Donde $0 \leq i^{(l+1)} < H^{(l+1)}, 0 \leq j^{(l+1)} < W^{(l+1)}$ y $0 \leq d < D^{(l+1)} = D^{(l)}$.

Un ejemplo se puede observar en la Figura 13, el volumen de entrada de tamaño [224x224x64] se mezcla con el tamaño del filtro 2, paso 2 en volumen de salida de tamaño [112x112x64]. Derecha: La operación max pooling ejecutada sobre la entrada con un paso de 2 produce como resultado 4 números [90].

Algoritmo de la capa de agrupado

1. Toma una entrada $x^{(l)}$ ó $(H^{(l)} \times W^{(l)} \times D^{(l)})$
2. Se aplican los hiperparámetros necesarios:
 - 2.1. Extensión espacial F .
 - 2.2. El salto $S, \forall 0 \leq S \leq 2$.
3. Produce una salida de volumen $y^{(l)} = x^{(l+1)}$ ó $(H^{(l+1)} \times W^{(l+1)} \times D^{(l+1)})$ donde:
 - 3.1. $W^{(l+1)} = (W^{(l)} - F) / S + 1$
 - 3.2. $H^{(l+1)} = (H^{(l)} - F) / S + 1$
 - 3.3. $D^{(l+1)} = K$

Back-propagation en la capa de agrupado

La *back-propagation* usa los conceptos de la sección 1.1.2. Si se parte de un elemento $(i^{(l)}, j^{(l)}, d^{(l)})$ de entrada $\mathbf{x}^{(l)}$ y otro elemento $(i^{(l+1)}, j^{(l+1)}, d^{(l+1)})$ perteneciente a la salida \mathbf{y} . Al aplicar la capa de *pooling* el resultado de la entrada $x_{i^{(l)}, j^{(l)}, d^{(l)}}^{(l)}$, será $y_{i^{(l+1)}, j^{(l+1)}, d^{(l+1)}}$ sí y solo si se cumplen las siguientes condiciones:

- Se encuentran en el mismo canal.
- La $(i^{(l)}, j^{(l)})$ -ésima entrada espacial pertenece a la $(i^{(l+1)}, j^{(l+1)})$ -ésima subregión.
- La $(i^{(l)}, j^{(l)})$ -ésima entrada espacial es la más grande en la subregión.

Si se define un indicador de matriz $S(\mathbf{x}^{(l)}) \in \mathbb{R}^{(H^{(l+1)}W^{(l+1)}D^{(l+1)}) \times (H^{(l)}W^{(l)}D^{(l)})}$, un tensor de índices $(i^{(l+1)}, j^{(l+1)}, d^{(l+1)})$ especificará una fila mientras que $(i^{(l)}, j^{(l)}, d^{(l)})$ especificará una columna. De forma que los dos tensores unidos especifican un elemento de $S(\mathbf{x}^{(l)})$ de forma que:

$$\text{vec}(\mathbf{y}) = S(\mathbf{x}^{(l)})\text{vec}(\mathbf{x}^{(l)})$$

Ecuación 15. Matriz de indicadores en la capa pooling.

Un diferencial de la Ecuación 15 es:

$$\frac{\partial \text{vec}(\mathbf{y})}{\partial (\text{vec}(\mathbf{x}^{(l)}))^T} = S(\mathbf{x}^{(l)}); \quad \frac{\partial z}{\partial (\text{vec}(\mathbf{x}^{(l)}))^T} = \frac{\partial z}{\partial (\text{vec}(\mathbf{x}^{(l)}))^T} S(\mathbf{x}^{(l)})$$

Y por lo tanto *back-propagation* puede ser calculada como:

$$\frac{\partial z}{\partial \text{vec}(\mathbf{x}^{(l)})} = S(\mathbf{x}^{(l)})^T \frac{\partial z}{\partial \text{vec}(\mathbf{y})}$$

Ecuación 16. Caculo de back-propagation en la capa de pooling.

Del análisis de la Ecuación 16 se tiene que solo existe una entrada diferente de cero por cada fila por lo que solo habrá $H^{l+1} W^{l+1} D^{l+1}$ en cada entrada $S(\mathbf{x}^{(l)})$ [88]. Para explicar este proceso se toma como ejemplo una matriz similar al de la Figura 13, si se aplica una *max pooling* de 2x2 y con un salto de 1 en todas direcciones, el elemento 7 es el número mayor que comparte las dos regiones de *pooling* $\begin{bmatrix} 1 & 1 \\ 5 & 7 \end{bmatrix}$ y $\begin{bmatrix} 1 & 2 \\ 7 & 6 \end{bmatrix}$. Entonces la columna $S(\mathbf{x}^{(l)})$ corresponde a 7 indexado por el tensor de entrada $(2, 2, d^{(l)})$ y además existen dos entradas diferentes de cero que corresponden a $(i^{(l+1)}, j^{(l+1)}, d^{(l+1)}) = (1, 1, d^{(l)})$ y $(1, 2, d^{(l)})$. Si se aplica la Ecuación 16 el resultado es el siguiente:

$$\left[\frac{\partial z}{\partial \text{vec}(\mathbf{x}^{(l)})} \right]_{(2,2,d^{(l)})} = \left[\frac{\partial z}{\partial \text{vec}(\mathbf{y})} \right]_{(1,1,d^{(l)})} + \left[\frac{\partial z}{\partial \text{vec}(\mathbf{y})} \right]_{(1,2,d^{(l)})}$$

1.2.1.4. Capa de Unidad de Rectificador Lineal

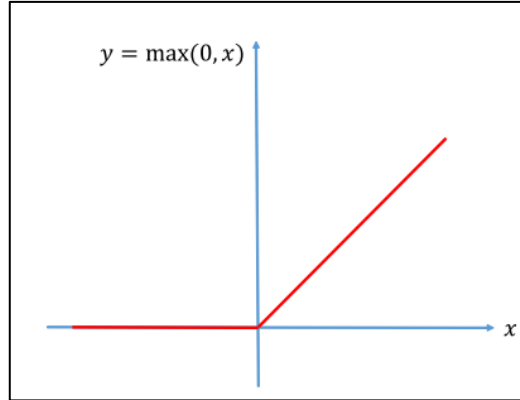


Figura 14. Función ReLU.

La capa de Unidad de Rectificador Lineal, por su nombre en inglés *Rectified Linear Unit* se abrevia como capa ReLU. La función de activación se aplica a todos los elementos del tensor de entrada. Esta operación no cambia la información espacial o la profundidad de la entrada. Sin embargo, convierte en 0 todos los elementos negativos del tensor [88].

Dada la entrada con el mismo tamaño $\mathbf{x}^{(l)} \in \mathbb{R}^{H \times W \times D}$ y la salida \mathbf{y} que a su vez se convierte en la entrada de la siguiente capa $\mathbf{y} = \mathbf{x}^{(l+1)}$; al pasar por una capa ReLU se realiza un truncamiento individual por cada elemento de la entrada de forma que:

$$y_{i,j,d} = \max\{0, x_{i,j,d}\}$$

Ecuación 17. Truncamiento de entrada en el modelo ReLU.

para $0 \leq i < H^{(l)} = H^{(l+1)}, 0 \leq j < W^{(l)} = W^{(l+1)}, 0 \leq d < D^{(l)} = D^{(l+1)}$. Considerando que no existen parámetros de peso dentro de la capa ReLU, ésta no necesita de un aprendizaje. Basándose en la Ecuación 17, un diferencial del argumento de la entrada mostrará un valor de 1 si es un valor máximo o caso contrario un valor de 0, de la siguiente forma:

$$\left[\frac{\delta z}{\delta \mathbf{x}^{(l)}} \right]_{i,j,d} = \begin{cases} \left[\frac{\delta z}{\delta \mathbf{y}} \right]_{i,j,d}, & \text{si } \mathbf{x}_{i,j,d}^{(l)} > 0 \\ 0, & \text{si } \mathbf{x}_{i,j,d}^{(l)} \leq 0 \end{cases}$$

Ecuación 18. Diferencial de la Capa ReLU.

A pesar de que la función $\max(0, x)$ matemáticamente no es diferenciable para $x = 0$ el uso de la capa ReLU en las CNNs de forma práctica no genera problemas. De forma gráfica la función ReLU se la puede observar en la Figura 14 [88].

El propósito de la capa ReLU es incrementar la no linealidad de la ConvNet; la información semántica en una imagen es un mapa no lineal de píxeles de entrada. Si $x_{i,j,d}^{(l)}$ es una de las características extraídas de la entrada $H^{(l)}xW^{(l)}xD^{(l)}$ desde la capa 1 hasta la capa $l - 1$, el resultado puede ser positivo, cero o negativo. Un ejemplo simple es la foto de un auto sobre un fondo de un solo color, los mapas de características que tengan un patrón de una parte del auto tendrán como resultado $x_{i,j,d}^{(l)} > 0$ y los mapas de características que no tengan un patrón que represente a un auto tendrán como resultado $x_{i,j,d}^{(l)} \leq 0$, La capa ReLU genera para los valores negativos un resultado de 0, esto significa que $y_{i,j,d}^{(l)}$ solo puede ser activada cuando una imagen procese los patrones en determinada región de la entrada. Por ejemplo, dada una entrada que contiene un gato, con mapas de características que son activados para el patrón de la cabeza, otro para las patas y otros para las demás partes del cuerpo se podrá determinar que en esa sección probablemente exista un gato [88].

1.2.1.1. Capa de Deserción

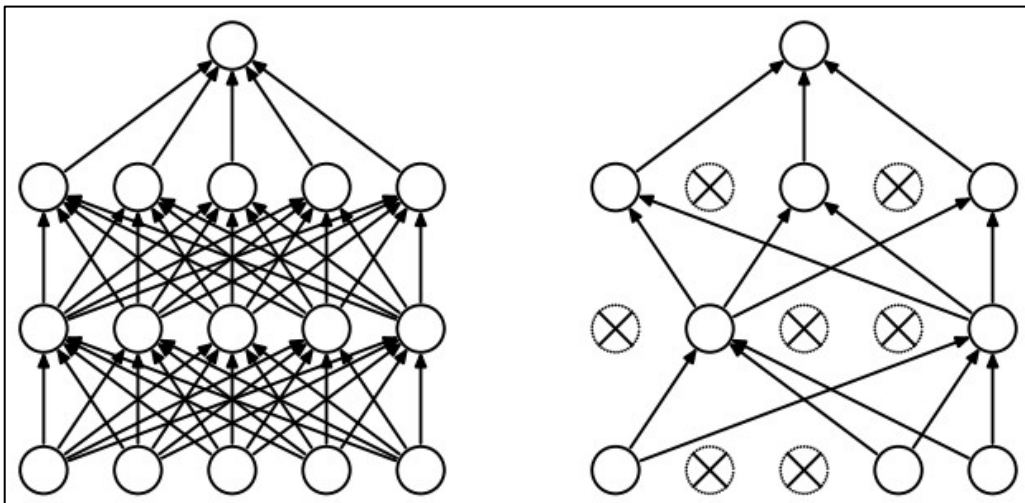


Figura 15. Ejemplo de Dropout.

La capa de deserción se la denomina comúnmente por su nombre en inglés *Dropout Layer*. En la ConvNet funciona como una técnica para reducir el *overfitting*. Fue propuesta como una forma de regularización para capas de redes neuronales totalmente conectadas. Cada

elemento de una capa de salida se mantiene con probabilidad p . Cuando se aplica la función de deserción a las salidas de una capa totalmente conectada se modelan como:

$$\mathbf{y} = \mathbf{m} \star a(\mathbf{w}^{(l)}\mathbf{x}^{(l)})$$

Donde \star es un operador de m con la máscara binaria del vector de tamaño d por cada elemento, y la salida \mathbf{y} de cada elemento es tomada de la forma $m_i \sim \text{Bernoulli}(p)$. Dado que las funciones de activación tienen como propiedad que $a(0) = 0$; y si se aplica a una matriz binaria M de capas conectadas. La salida de la capa totalmente conectada está dada por:

$$\mathbf{y} = a\left((M \times \mathbf{w}^{(l)})\mathbf{x}^{(l)}\right)$$

Ecuación 19. Modelo de una capa dropout.

Un ejemplo en el que se aplica deserción se observa en la Figura 15. Una red neuronal tiende a estar totalmente conectada y esto puede causar *overfitting* en el entrenamiento de la red; por lo que al aplicar una capa de *dropout* en el entrenamiento se evita que una entrada pase por una misma neurona de forma que le fuerza a aprender por otras neuronas; por esto se recomienda usar *dropout* en las Capas totalmente conectadas [91], [92].

1.2.1.2. Capa Totalmente Conectada

La capa Totalmente conectada se la denomina comúnmente por su nombre en inglés *Fully-Connected Layer* y se abrevia como FC. Las neuronas de la FC tienen conexiones completas a todas las actividades en la capa anterior. Esto es, cualquier elemento de la salida $\mathbf{x}^{(l+1)}$ requiere todos los elementos de la entrada $\mathbf{x}^{(l)}$. Por ejemplo: si una entrada $\mathbf{x}^{(l)}$ de tamaño $H^{(l)} \times W^{(l)} \times D^{(l)}$ al aplicar una convolución con tamaño $H^{(l)} \times W^{(l)} \times D^{(l)}$ y un kernel D la salida será $\mathbf{y} \in \mathbb{R}^D$. Obviamente para calcular cualquier elemento en \mathbf{y} será necesario usar todos los elementos de la entrada $\mathbf{x}^{(l)}$ [88]

1.2.1.3. Salida

Una imagen que atravesó el proceso de una ConvNet con varias capas en las que se extrajo características genera una salida. Ésta será la clasificación de la imagen en una clase previamente entrenada [93].

1.2.2. Entrenamiento y Validación en CNN

El entrenamiento en las Redes Neuronales Convolucionales presenta los mejores resultados cuando tienen una correcta inicialización de pesos, hiperparámetros y número de épocas. Para la Validación se verifica la función de pérdida como se observó en la sección 1.1.1.

Inicialización de pesos

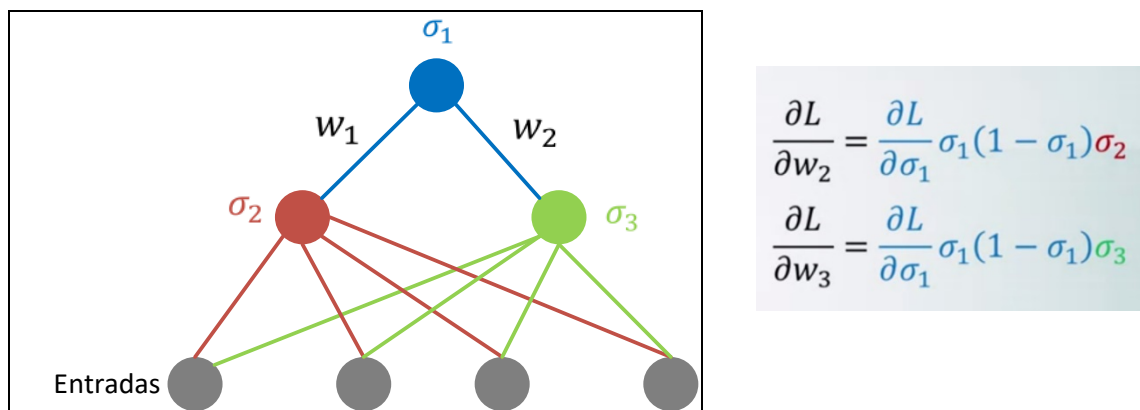


Figura 16. Inicialización de pesos en una red neuronal.

Todas las neuronas tienen pesos a ser afinados que en un principio deben ser inicializados con un valor. Para una correcta inicialización de los pesos se debe tener en cuenta que si los pesos son inicializados con cero cuando se realice la *back-propagation* los pesos serán actualizados, los mismos para todas las neuronas. Lo que se sugiere es realizar una inicialización aleatoria de números entre 0,01 y 1.

El ejemplo de la Figura 16, se tiene cuatro entradas y tres neuronas. σ_1, σ_2 y σ_3 son funciones de activación. Al realizar *back-propagation* lo que sucederá es que los valores de σ_2 y σ_3 tendrán los mismos valores actualizados cuando se tengan pesos iniciales de ceros. Esto significa que se tienen neuronas similares y el aprendizaje no será correcto [94].

Hiperparámetros

Los hiperparámetros de una Red Neuronal Convolutiva son:

1. **Taza de aprendizaje.** Es un parámetro que controla que tan rápido se realiza el aprendizaje en una red. Para elegir una cantidad adecuada se considera lo siguiente [95]:
 - a. La velocidad de decrecimiento por época. Si no hay decrecimiento en muchas épocas la tasa de aprendizaje es muy bajo.
 - b. Si la función de pérdida es NAN significa que la tasa de aprendizaje es demasiado alta.
 - c. Experimentalmente las tasas de aprendizaje optimas se encuentran entre $[0,1 - 0,00001]$.
2. **Momentum.** Este parámetro es parte de los algoritmos de descenso de gradiente (SGD). El momentum ayuda a acelerar el SGD en dirección al mínimo global, el valor más común usado es 0,9 [96]. Existe la presencia de otro tipo de momentum que son ampliamente usados, entre los que mejor convergen se encuentra Adagrad [97].
3. **Tamaño del batch.** Es el tamaño de procesos paralelos que ingresan a la red neural por cada iteración. Experimentalmente debe ser entre 16 y 128 [98].
4. **Pérdida de peso.** También conocido como penalidad de peso, su función es mantener los pesos pequeños o que tiendan a 0 con el objetivo de generar buenos modelos. Experimentalmente un buen coeficiente de pérdida de peso es 0,0005 [99].

Función de Pérdida

La función de pérdida en inglés *Loss Function* cuantifica la calidad predicción de clase de un vector de características y se utiliza para definir una función objetivo. Para problemas de clasificación la función de pérdida es típicamente un método de entropía cruzada (*cross-entropy*) con regularizadores l_1 o l_2 [100].

$$E_{CE}(W) = \underbrace{\sum_{x \in X} \sum_{k=1}^K [t_k^x \log(o_k^x) + (1 - t_k^x) \log(1 - o_k^x)]}_{\text{cross-entropy pérdida de data}} + \underbrace{\lambda_1 \times \sum_{w \in W} |w| + \lambda_2 \times \sum_{w \in W} w^2}_{\text{modelo de pérdida}}$$

Ecuación 20. Función de pérdida.

Donde W son los pesos, X es el *dataset* de entrenamiento, K es el número de clases y t_k^x indica si el ejemplo de entrenamiento x es de la clase k . o_k^t es la salida del algoritmo de clasificación que depende de los pesos. $\lambda_1, \lambda_2 \in [0, \infty)$ son los pesos de la regularización y típicamente son menores que 0,1. Una de las razones de uso de la función de pérdida es

porque contiene más información sobre la calidad del modelo. En la función de pérdida se debe tener las siguientes observaciones [95]:

1. Si la pérdida disminuye en varias épocas, se debe a que la tasa de aprendizaje es muy baja. Además, el proceso de optimización no será capaz de encontrar un mínimo global.
2. Si la pérdida genera NAN significa que la tasa de aprendizaje es muy alta.
3. Si la curva de validación de las épocas de pérdida tiene una meseta al inicio la inicialización del peso puede ser mala.

1.2.3. Consideraciones Computacionales sobre las ConvNet

En la construcción de arquitecturas ConvNet el cuello de botella de memoria es uno de los principales limitantes. Muchos GPU tienen una memoria limitada, actualmente el récord de capacidad de memoria en GPU la posee NVIDIA con la tarjeta TITAN Xp [101].

En una ConvNet las fuentes principales de memoria hacen un seguimiento de [89]:

- **Los tamaños de volúmenes intermedios:** son el número total de las activaciones y gradientes en cada capa de la ConvNet. Por lo general, la mayoría de las activaciones se encuentran en las primeras capas de una ConvNet. Estos volúmenes intermedios se guardan porque son necesarios para la *back-propagation*.
- **Los tamaños de los parámetros:** Son los números que contienen los parámetros de red, sus gradientes durante la *back-propagation*, y comúnmente también un caché de saltos siempre y cuando la optimización use Adagrad o RMSProp. Por lo tanto, para que la memoria almacene el vector de parámetros solo debe multiplicarse por un factor de al menos 3 o más.
- **La miscelánea de la ConvNet en memoria:** Las implementaciones de las ConvNet requieren tener una miscelánea en la memoria, entre las cosas que se almacenan son: lotes de datos de imagen, versiones aumentadas, entre otras.

Luego de realizar una estimación aproximada de la cantidad total de información a procesar, se debe verificar que: los parámetros a ser procesados caben en VRAM de la GPU. Si no es así, una heurística común para que pueda caber es disminuir el tamaño del batch y así tener menos entradas que procesar en paralelo [89].

1.3. Support Vector Machine

Los SVMs son un conjunto de modelos de aprendizaje supervisado. Tienen como entrada un set de datos, éste se agrupa en clases basándose en ciertas reglas comunes para cada clase. El set de datos de entrada es usado para aprender un hiperplano que separe los grupos en las diferentes categorías previamente especificadas [102].

1.3.1. Clasificadores de Margen Máximo

Los clasificadores de margen máximo utilizan los elementos del borde de la clase para poder separar los elementos mediante un hiperplano.

1.3.1.1. Hiperplanos

Un hiperplano se define de la siguiente forma: En un espacio p -dimensional, un hiperplano es un subespacio afín al plano de la dimensión del hiperplano ($p - 1$). De forma que, en un espacio de dos dimensiones, un hiperplano será un subespacio de una dimensión (una línea). En tres dimensiones, un hiperplano es un subespacio de dos dimensiones (un plano). Para espacios dimensionales superiores a tres a pesar de que es difícil visualizarlos, la noción de un hiperplano se mantiene.

Matemáticamente el hiperplano p -dimensional se define por

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p = 0$$

$$\{x: f(x) = x^T \beta + \beta_0 = 0\}$$

Ecuación 21. Definición de un hiperplano.

Donde β es una unidad del vector $\|\beta\| = 1$ y x^T permite generar un hiperplano de tamaño $p - 1$ [17].

1.3.1.2. Clasificación usando Hiperplanos

La teoría de clasificación de hiperplanos está tomada de [17] y [22]. En resumen, el modelo lineal de regresión el que involucra las combinaciones de las variables de entrada:

$$y(x, w) = w_0 + w_1 x_1 + \dots + w_i x_i$$

Donde $x = (x_1, \dots, x_i)^T$ es conocida como regresión lineal y los parámetros w_0, \dots, w_i forman parte de la función lineal de las variables de entrada x_i . La extensión del modelo considerando las combinaciones no lineales de las variables de entrada tiene la forma:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

Donde $\phi_j(\mathbf{x})$ es conocida como funciones base; el valor máximo de j es $M - 1$, el número total de parámetros en este modelo será M y el parámetro w_0 permite un desplazamiento fijo de los datos se lo conoce como sesgo (*bias*) que a menudo se lo suma a la función base, entonces:

$$\begin{aligned} y(\mathbf{x}, \mathbf{w}) &= \sum_{j=0}^M w_j \phi_j(\mathbf{x}) \\ &= \mathbf{w}^T \phi(\mathbf{x}) \end{aligned}$$

Donde $\mathbf{w} = (w_0, \dots, w_{M-1})^T$ y $\phi = (\phi_0, \dots, \phi_{M-1})^T$. Entonces un modelo de solución básica de problemas de clasificación lineal esta dado por la Ecuación 22:

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

Ecuación 22. Solución básica de problemas de clasificación lineal

Donde $\phi(\mathbf{x})$ es el espacio donde se encuentran las muestras y b es el sesgo. Partiendo de un set de entrenamiento N con vectores de entrada $\mathbf{x}_1, \dots, \mathbf{x}_i$, los cuales tienen valores objetivo y_1, \dots, y_i donde $y_i \in \{-1, 1\}$ y los puntos \mathbf{x} son clasificados de acuerdo con el signo de $y(\mathbf{x})$. El objetivo es desarrollar un clasificador basado en datos de entrenamiento que clasifique correctamente las muestras de cada clase.

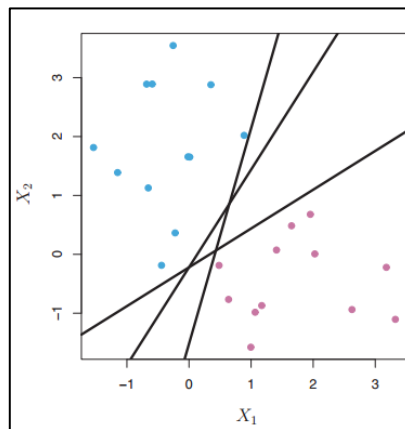


Figura 17. Ejemplo de hiperplanos que separan las muestras.

Partiendo de la suposición de que se puede construir un hiperplano que separe las muestras de entrenamiento perfectamente de acuerdo con la clase objetivo, entonces posibles resultados de estos hiperplanos que clasifiquen las muestras serán como se

muestran en la Figura 17. Si se etiqueta las observaciones de forma que las muestras azules $y_i = 1$ y las muestras púrpuras $y_i = -1$.

Un hiperplano que separe las muestras tiene la propiedad de

$$x^T \beta + \beta_0 > 0, \text{ si } y(x_i) > 0 \text{ para } t_i = 1$$

$$x^T \beta + \beta_0 < 0 \text{ si } y(x_i) < 0 \text{ para } t_i = -1$$

Y el hiperplano separador tiene la propiedad de

$$y_i(x^T \beta + \beta_0) > 0, \forall i \geq 1$$

De esta forma se puede clasificar las muestras x en clases, basándose en el signo de $y(x_i)$. Si $y(x_i)$ es positivo se asigna a la clase 1 y si es negativo se asigna a la clase -1 tal como se muestra en la Figura 18.

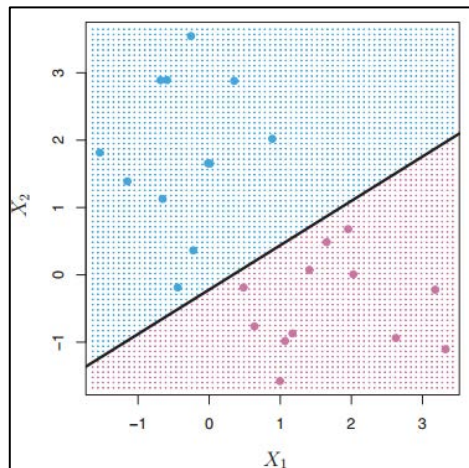


Figura 18. Ejemplo de clasificación usando hiperplano.

1.3.1.3. Clasificador de Margen Máximo

Existe un infinito número de hiperplanos que pueden separar a los datos en sus diferentes clases. De esos planos lo más eficiente es encontrar un hiperplano que separe a las muestras y se encuentre lo más alejado de ellas. El hiperplano de separación óptimo debe estar lo más alejado posible de las muestras más cercanas de set de entrenamiento.

Matemáticamente se clasificará para el set de muestras x basado en el signo de $y(x_i)$. La figura 11 muestra un ejemplo. La creación de un hiperplano de margen máximo. Los puntos equidistantes del margen máximo del hiperplano. Estos puntos se los conocen como vectores de soporte que se encuentran en un espacio p -dimensional y dan soporte al margen máximo de hiperplano.

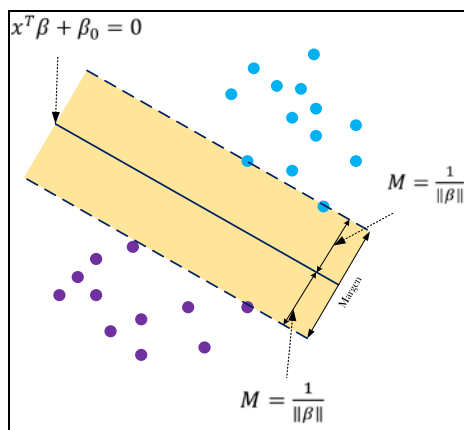


Figura 19. Hiperplano de margen máximo.

La construcción de un hiperplano con margen máximo basado en un set de datos de entrenamiento n con muestras $x_1, x_2, \dots, x_n \in \mathbb{R}^p$ cada una de ellas asociada a las etiquetas $y_1, \dots, y_n \in \{-1, 1\}$. Como el objetivo es optimizar el margen máximo del

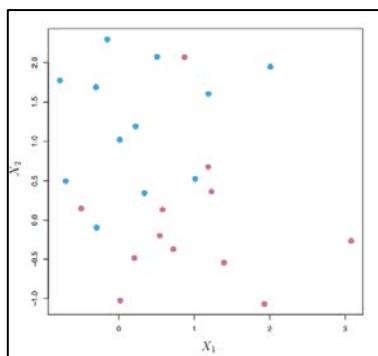


Figura 20. Set de datos con muestras no separables

hiperplano; primero se debe garantizar que cada muestra se encuentra correctamente clasificada, esto es:

$$\max_{\beta, \beta_0, \|\beta\|=1} M$$

$$y_i(x^T \beta + \beta_0) \geq M, \forall i = 1, \dots, n$$

Ecuación 23. Clasificador de margen máximo.

Donde M es el ancho del margen. La Ecuación 23 garantiza que cada muestra se encuentra clasificada de forma correcta. Sin embargo, existe el caso en el que las muestras no son separables en su totalidad y no se cumple la Ecuación 23. Un ejemplo de esto se puede observar en la Figura 20, donde las muestras azules y púrpuras no pueden ser separadas por un hiperplano y por lo tanto no se puede usar un clasificador de margen máximo [17], [22].

1.3.1.4. Clasificadores de Soporte Vectorial

Cuando las muestras de un set de datos no son perfectamente separables o cuando el margen máximo del hiperplano es muy reducido se debe considerar el método de clasificación por vectores de soporte o clasificadores de margen suave. Estos permiten que algunas muestras se clasifiquen en el lado incorrecto del margen, pero dan mayor robustez para la clasificación de nuevos datos relacionados con el set de entrenamiento.

Según [103], el modelo de distribución con vectores x para cada clase se usa el estimado de densidad de Parzen con kernels Gaussianos teniendo como común el parámetro σ^2 . El cual define un clasificador con un margen de error. por lo que se debe determinar el mejor hiperplano que minimice la probabilidad del error de forma que el $\lim_{\sigma \rightarrow 0} \sigma^2$ tiene el máximo margen.

La distancia perpendicular de un punto x desde el hiperplano está dado por $y(x) = 0$ donde $y(x)$ toma la forma $\frac{y(x)}{\|\beta\|}$. Si solo se toman los puntos correctamente clasificados $t_i y(x_i) > 0$ Entonces la distancia de un punto x_i está dado por

$$\frac{t_i y(x_i)}{\|\beta\|} = \frac{t_i (x^T \beta + \beta_0)}{\|\beta\|}$$

Ecuación 24. Distancia de un punto al hiperplano.

El margen está dado por la distancia perpendicular a los puntos más cercanos al set de entrenamiento x_i . Por lo que se desea optimizar los parámetros β y b maximizando su distancia. La solución del margen máximo se resuelve por

$$\arg \max_{w, b} \left\{ \frac{1}{\|\beta\|} \min_n [t_n (x^T \beta + \beta_0)] \right\}$$

Donde el factor $\frac{1}{\|\beta\|}$ se encuentra fuera de la optimización para n debido que β no depende de n . La solución de optimización del margen se resuelve partiendo de la Ecuación 14

$$\begin{aligned} & \min_{\beta, \beta_0} \|\beta\| \\ & y_i (x^T \beta + \beta_0) \geq 1, \forall i = 1, \dots, n \\ & y_i (x^T \beta + \beta_0) \geq M(1 - \epsilon_i), \epsilon_i \geq 0, \sum_{i=1}^n \epsilon_i \leq C \end{aligned}$$

Ecuación 25. Hiperplano con vectores de soporte.

Donde C es el parámetro de ajuste, M es ancho del margen que se busca hacer lo más grande posible, $\epsilon_1, \dots, \epsilon_n$ son variables de holgura que permiten que las muestras que se encuentran en el lado equivocado del plano no se consideren para el cálculo del margen máximo. Para las variables de holgura se debe considerar lo siguiente:

- Si $\epsilon_i = 0$ la i -ésima muestra se encuentra en el lugar correcto.
- Si $\epsilon_i > 0$ la i -ésima muestra se encuentra en el lado equivocado del margen.
- Si $\epsilon_i > 1$ la i -ésima muestra se encuentra en el lado equivocado del hiperplano.

El parámetro de ajuste C determina el número y la severidad de las violaciones al margen e hiperplano que se toleran. Para el este parámetro se considera lo siguiente:

- Si $C = 0$ no existen violaciones al margen y además se verifica que $\epsilon_1 = \dots = \epsilon_n = 0$,
- Si $C > 0$ solo deben estar C muestras en el lado incorrecto del hiperplano y además se verifica que $\epsilon_i > 1$ y se requiere que $\sum_{i=1}^n \epsilon_i \leq C$
 - Si C aumenta, el margen aumenta y es más tolerante a las violaciones de las muestras.
 - Si C disminuye, el margen disminuye y es menos tolerante a las violaciones de las muestras.

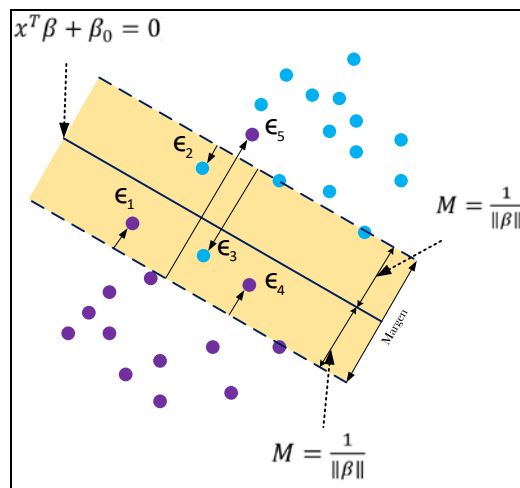


Figura 21. Solución gráfica de hiperplano con vectores de soporte.

Una solución de forma gráfica de los vectores de soporte se mira en la Figura 21. Dado que solo los vectores de soporte afectan al clasificador, el parámetro C controla el sesgo y la varianza del calificador. Cuando C es grande, el margen es ancho y muchas muestras violan el margen por lo que hay muchos vectores de soporte, un alto sesgo y baja varianza. Para este caso muchas muestras están involucradas en la determinación del hiperplano. Por el contrario, si C es pequeño hay menos vectores de soporte, se tiene un sesgo bajo y alta varianza.

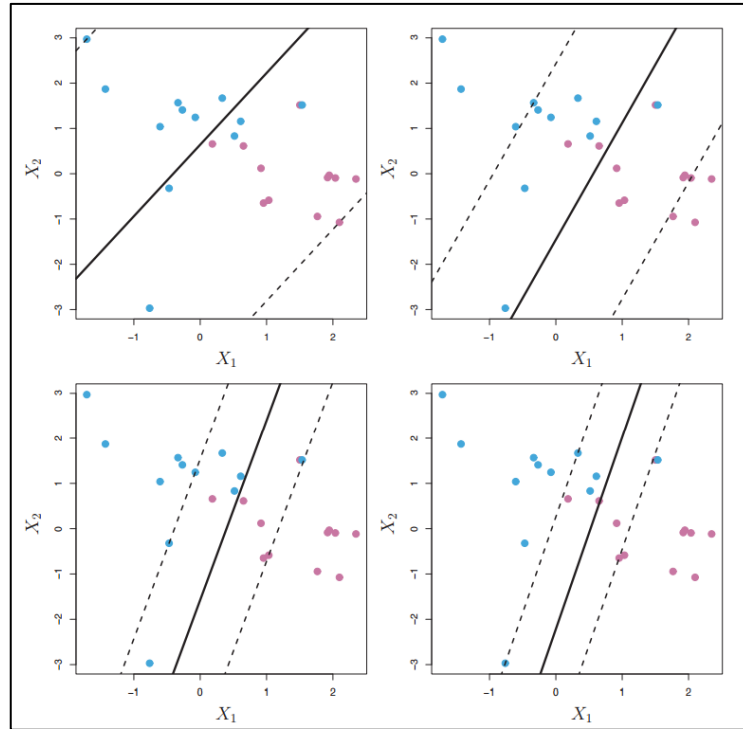


Figura 22. Uso de varios parámetros C en el clasificador con vectores de soporte.

En la Figura 22 se puede observar como cada una de las variaciones del parámetro C afecta a la creación del hiperplano; en la parte superior izquierda se encuentra un C muy grande y al contrario en la parte inferior derecha se encuentra un C muy pequeño, ésta formado por ocho vectores de soporte.

El hecho de que la creación del hiperplano solo se realiza en base a un subconjunto de muestras de entrenamiento significa que el clasificador es muy robusto para el comportamiento de las muestras que se encuentran alejadas del hiperplano [17], [22].

1.3.2. Máquinas de Soporte Vectorial

La Ecuación 16 es un problema cuadrático con restricciones lineales de desigualdad y por tanto es un problema de optimización convexa. La solución de programación cuadrática se resuelve usando multiplicadores de Lagrange y computacionalmente se la reescribe de la siguiente forma [23]:

$$\min_{\beta, \beta_0} \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^N \epsilon_i$$

$$\text{Sujeta a } \epsilon \geq 0, y_i(x_i^T \beta + \beta_0) \geq 1 - \epsilon_i, \forall i$$

Ecuación 26. Hiperplano con vectores de soporte reescrito computacionalmente.

La función principal de Lagrange es:

$$L_p = \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^N \epsilon_i - \sum_{i=1}^N \alpha [y_i(x_i^T \beta + \beta_0) - (1 - \epsilon_i)] - \sum_{i=1}^N \mu_i \epsilon_i$$

Ecuación 27. Función de Lagrange aplicada al problema de optimización.

Y minimiza con respecto a β, β_0 y ϵ_i . Configurando las respectivas derivadas a cero se obtiene:

$$\beta = \sum_{i=1}^N \alpha_i y_i x_i,$$

$$0 = \sum_{i=1}^N \alpha_i y_i,$$

$$\alpha_i = C - \mu_i, \forall i,$$

Ecuación 28. Soluciones de la función de Lagrange.

Tomando como restricción que $\alpha_i, \epsilon_i, \mu_i \geq 0 \forall i$. Se tiene el Lagrangiano de Wolfe, la función dual objetivo es

$$L_D = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{i'=1}^N \alpha_i \alpha_{i'} y_i y_{i'} x_i^T x_{i'},$$

La cual genera un límite inferior en la función objetivo de la Ecuación 28 es para cualquier punto factible. Al maximizar L_D con $0 \leq \alpha_i \leq C$ y $\sum_{i=1}^N \alpha_i y_i = 0$ se aplica a las soluciones de la función de Lagrange con las condiciones Karush-Kuhn-Trucker para obtener

$$\alpha_i [y_i(x_i^T \beta + \beta_0) - (1 - \epsilon_i)] = 0$$

$$\mu_i \epsilon_i = 0$$

$$y_i(x_i^T \beta + \beta_0) - (1 - \epsilon_i) \geq 0$$

Ecuación 29. Soluciones para el problema de optimización del hiperplano con vectores de soporte.

Para $i = 1, \dots, N$. En conjunto las ecuaciones resuelven el problema primario y dual. A partir de la Ecuación 29 se puede observar que la solución para β tiene la forma $\hat{\beta} = \sum_{i=1}^N \hat{\alpha}_i y_i x_i$ para coeficientes de $\hat{\alpha}_i \neq 0$ las muestras i se encuentran exactamente dentro de la solución de la Ecuación 21. Entonces estas muestras son los vectores de soporte. Entre los puntos de soporte, algunos se encuentran en el margen ($\epsilon_i = 0$) y por lo tanto pueden

ser caracterizados como $0 < \hat{\alpha}_i < C$; el resto de los puntos ($\epsilon_i > 0$) tienen $\hat{\alpha}_i = C$ y su margen se encuentra en los puntos ($0 < \hat{\alpha}_i, \epsilon_i = 0$).

La solución a la función $f(x)$ se la escribe como:

$$f(x) = h(x)^T \beta + \beta_0$$

$$= \sum_{i=1}^N \alpha_i y_i \langle h(x), h(x_i) \rangle + \beta_0$$

En este caso se involucra $h(x)$ como producto punto y para la solución es necesario conocer la función kernel

$$K(x, x') \langle h(x), h(x') \rangle$$

La cual calcula los productos punto en el espacio. K puede ser una función simétrica definida positivamente; Entre las más básicas estas pueden ser [104]:

dth – Grado polynomial: $K(x, x') = (\langle x, x' \rangle + c)^d, \forall d > 1$

Gaussian Radial Basis Function: $K(x, x') = \exp(-\gamma \|x - x'\|^2); \forall \gamma > 0, \gamma = 1/2\sigma^2$

1.3.3. Matriz de Confusión

El método más usado para obtener la precisión generada por un modelo de clasificación es realizado mediante la matriz de confusión. El método permite la visualización del rendimiento de un algoritmo de aprendizaje supervisado debido a que detalla la precisión correcta de la clasificación [105]. Por esta cualidad, las matrices de confusión se usan para:

- Estimar la precisión general de la técnica de clasificación [106].
- Identificar un error específico que afecta a las categorías individuales [107].
- Medir el grado en que la técnica de interpretación subestima o sobreestima una categoría particular y por lo tanto puede usarse para ajustar los valores obtenidos mediante el método de clasificación [108].

Tabla 1. Ejemplo de Matriz de Confusión.

		Condición verdadera	
		Condición negativa	Condición Positiva
Técnica de clasificación	Predicción condición negativa	Verdadero Negativo a	Falso Negativo b
	Predicción condición positiva	Falso Positivo c	Verdadero Positivo d

Una matriz de confusión se construye tal como se muestra en la Tabla 1. Las filas indican las categorías que determinan una técnica de clasificación; las columnas indican las categorías verdaderas y el valor de las celdas indica el número de observaciones. Del ejemplo de la matriz de confusión obtiene la siguiente información [109]:

1. **Exactitud (AC).** Es la proporción del número total de predicciones que fueron correctas. Se calcula con la Ecuación 30:

$$AC = \frac{a + d}{a + b + c + d}$$

Ecuación 30. Exactitud (AC).

2. **Taza de verdaderos positivos (TPR).** Es la proporción de casos positivos que fueron identificados. Se calcula con la Ecuación 31:

$$TPR = \frac{d}{b + d}$$

Ecuación 31. Taza de verdaderos positivos (TPR).

3. **Taza de falsos positivos (FPR).** Es la proporción de casos negativos que fueron incorrectamente clasificados como positivos. Se calcula con la Ecuación 32:

$$FPR = \frac{c}{a + c}$$

Ecuación 32. Taza de falsos positivos (FPR).

4. **Taza de verdaderos negativos (TNR).** Es la proporción de casos negativos que fueron clasificados correctamente. Se calcula con la Ecuación 33:

$$TNR = \frac{a}{a + c}$$

Ecuación 33. Taza de verdaderos negativos (TNR).

5. **Taza de falsos negativos (FNR).** Es la proporción de casos positivos que fueron incorrectamente clasificados como negativos. Se calcula con la Ecuación 34:

$$FN = \frac{b}{b + d}$$

Ecuación 34. Taza de falsos negativos (FNR).

6. **Taza de Error (ERR).** Es la proporción de todas las predicciones incorrectas divididas para el total elementos. Se calcula con la Ecuación 35:

$$P = \frac{b + c}{a + b + c + d}$$

Ecuación 35. Taza de Error (ERR).

Coeficiente de Correlación de Matthews

El Coeficiente de Correlación de Matthews (MCC) es usado en aprendizaje de máquina como una medida de la calidad de un clasificador binario. Toma como entradas las predicciones positivas, las falsas positivas y las negativas. Comúnmente se usa para clases con tamaños muy diferentes. El MCC genera un valor entre -1 y +1, este valor es un coeficiente de correlación entre las clasificaciones binarias actuales y predichas. Un coeficiente de +1 representa una predicción perfecta, -1 indica un error entre la predicción y el valor actual y 0 se considera que genera predicción aleatoria. Desde la Ecuación 31 hasta la Ecuación 34 este coeficiente se calcula con la siguiente fórmula [110]:

$$MCC = \frac{TPR \times TNR - FPR \times FNR}{\sqrt{(TPR + FPR)(TPR + FNR)(TNR + FPR)(TNR + FNR)}}$$

Ecuación 36. Coeficiente de correlación de Matthews.

1.4. Herramientas y Entorno de Desarrollo

Como se muestra en las secciones 1.2 y 1.3, los cómputos que deben ser llevados a cabo generan una carga de trabajo extenuante para el software y hardware, por lo que es necesario contar con herramientas adecuadas. A continuación, se realiza un análisis las principales librerías usadas y la estructura general del entorno de desarrollo.

De forma principal se usa el lenguaje de programación Python 3 que según los estudios de Thoughtworks desde el año 2014 y a través de sus revistas en sus volúmenes [111], [112], [113] y [114], este último publicado a finales del año 2017 consideran que se debe adoptar el uso de Python 3 para usarlo en proyecto. La razón es porque "...es una herramienta con gran alcance y fácil de usar" [115]. Además, un punto importante a considerar es el uso de la librería NumPy dado que es uno de los paquetes más importantes para la computación

científica con Python. La librería puede trabajar con tensores, funciones de difusión, tiene herramientas para integrar código C/C++ y Fortran, entre otras ventajas [116].

1.4.1. Hardware y Sistema Operativo

Hardware

Según expertos [117], [118] más que tener un buen procesador, se necesita una tarjeta gráfica dedicada, entre las más conocidas tenemos: NVIDIA [119], AMD [120], Zotac [121], MSI [122], entre otras [123], [124]. Cada una de ellas con diferentes precios y arquitecturas; NVIDIA por su parte para este año presentó la arquitectura de GPU PASCAL incluida en sus tarjetas NVIDIA TITAN Xp [125]; las características principales se muestran en la columna GPU Titan de Tabla 1. Para proyecto se hará uso de CPU y GPU con las siguientes características:

Tabla 2. Hardware usado en el proyecto.

	CPU [126]	GPU [127]	GPU Titan
Marca y Modelo	<i>i7 - 4720HQ</i>	<i>Nvidia GeForce 960M</i>	<i>Titan GeForce</i>
Cantidad de núcleos	4	640	3840
RAM/VRAM (soportada/integrada)	<i><= 32 GB</i>	<i>2 GB</i>	<i>12 GB</i>
Frecuencia del procesador	<i>2.60 GHz - 3.60 GHz</i>	<i>1096 MHz</i>	<i>1582MHz</i>

Sistema Operativo

Según expertos [128], [129] y [130] un sistema operativo adecuado para *deep learning* debe estar de acuerdo con la familiaridad del usuario y con el alto grado de configuración que se pueda realizar. Por lo tanto, para el proyecto se trabajará con Ubuntu Desktop en su versión 16.04 LTS por las siguientes características [131]:

- Es un sistema estable.
- Consume pocos recursos en memoria RAM y VRAM GPU.
- Se puede realizar una alta configuración del Sistema operativo.
- Está altamente documentado.
- Es Software Libre.

- Es un sistema ampliamente usado por usuarios que trabajan con temas de Inteligencia Artificial.

1.4.2. **Frameworks para ConvNet**

Entre los frameworks más conocidos KDnugget [132], una página web orientada a *Business Analytics, Big Data, Data Mining, Data Science* y *Machine Learning*. Que actualmente es editada por el Ph.D. Gregory Piatetsky-Shapiro. [133] y Matthew May MSc. [134], muestra al menos 50 diferentes frameworks dedicados a *deep learning*. Sin embargo, entre los más difundidos son los siguientes [98]:

1. **Tensorflow** [135]:

- Es una librería de software *open source* para computación numérica que usa grafos de flujo de datos.
- Desarrollado por investigadores e ingenieros que trabajan en el *Google Brain Team* dentro de la Organización de Inteligencia Artificial, el objetivo de su desarrollo fue realizar investigaciones de aprendizaje automático y redes neuronales profundas.
- La mayoría de las librerías de *deep learning* se encuentran escritos en Python sobre C/C++, esto hace que su ejecución será rápida.
- Tiene ejemplos y código bien documentados.

2. **MxNet** [136]:

- Este framework de deep learning está basado en las especificaciones de la librería GLUON lo que genera una API limpia concisa y simple para trabajar en varios lenguajes como R, Python y Julia.
- Tiene capas, optimizadores e inicializadores predefinidos lo que hace que iniciar un proceso de entrenamiento sea sencillo.
- El desarrollo de los modelos neuronales es dinámico.

3. **Caffe** [137]:

- Esta construido con expresiones enfocadas a la velocidad y modularidad.
- Este proyecto fue propuesto como proyecto de Doctorado de Yangqing Jia en la Universidad de Berkeley y desarrollado por investigadores de Inteligencia Artificial de Berkeley (BAIR).
- Tiene una amplia comunidad de desarrolladores, actualmente son 260 contribuidores [138] y realizan cambios a diario en el repositorio de github.
- Su velocidad de ejecución que permite cerca de 60 millones de imágenes por día.

4. CNTK [139]:

- *Microsoft Cognitive Toolkit o CNTK* es un conjunto de herramientas que describe a las redes neuronales como una serie de pasos como grafos dirigidos.
- CNTK permite realizar y combinar fácilmente tipos de modelos populares como ConvNet y RNN.
- Implementa el aprendizaje de descenso de gradiente estocástico (SGD) con diferenciación automática y paralelización en múltiples GPU.

Para motivos del proyecto se hace uso de Tensorflow principalmente porque se encuentra en una versión muy estable y por su amplia documentación detallada sobre ConvNets. A diferencia de Caffe que se mantiene en continuo desarrollo lo que genera poca estabilidad; MxNet en comparación con Tensorflow tiene pocos ejemplos y se encuentra menos documentada o CNTK que está desarrollada para ejecutarse en entorno Windows. Ninguno de éstos *frameworks* contribuyen potencialmente al desarrollo del proyecto como lo hace Tensorflow.

1.4.3. **Frameworks para Máquinas de Soporte Vectorial**

Existe una gran cantidad de frameworks que integran SVM, entre los más conocidos tenemos:

1. **Scikit-Learn** [140] [141]: es un framework construido sobre Numpy, SciPy y matplotlib enfocado a la solución de problemas matemáticos. El kit es open source por lo que lo vuelve un framework reutilizable. Tiene actualmente 1022 contribuidores en su página de github. Sus herramientas principales tareas son:
 - Clasificación: Identifica a que categoría pertenece un objeto.
 - Regresión: Predice un atributo asociado con un objeto.
 - *Clustering*: Agrupación automática de objetos similares en sets.
 - Reducción de dimensionalidad PCA: Reducción del número de variables.
 - Selección de modelos: Comparación, validación y selección de los parámetros de los modelos.
 - Preprocesamiento: Extracción de características y normalización de datos.
2. **Apache Singa** [142]: Es una plataforma general de aprendizaje profundo orientado al trabajo con grandes conjuntos de datos. Se encuentra desarrollado en C con un modelo intuitivo basado en la abstracción de capas. Admite una gran variedad de modelos de aprendizaje profundo, redes neuronales. Actualmente la desarrollan 30 contribuidores en su página de github.

3. **Amazon Machine Learning** [143]: es un servicio que facilita a los desarrolladores el uso de la tecnología de aprendizaje automático. Proporciona herramientas de visualización y asistentes que guían a un usuario en el proceso de la creación de modelos de aprendizaje automático.
4. **Azure ML Studio** [144]: está orientada a los usuarios de Microsoft Azure para crear y entrenar modelos que posteriormente pueden ser convertidos a API. Tiene una amplia gama de algoritmos disponibles. Entre sus principales tareas se encuentran la minería de datos, el mantenimiento de modelos predictivos y Data Science.
5. **LIBSVM** [145]: La librería para SVM o abreviado LIBSVM está desarrollada para clasificación de soporte vectorial, regresión, distribución de probabilidades y soporta clasificación multi-clase. Es compatible para varios lenguajes de programación entre los principales Python, R, MATLAB, Perl, Weka, LISP, entre otros.

Para propósitos del proyecto se hace uso del `sckit-learn` debido a que es compatible con Python y permite generar y leer modelos basados en Numpy que son compatibles con otros *frameworks* lo que le da más flexibilidad al momento de unir varios *frameworks*. Al contrario que Apache Singa o LIBSVM que tienen integrados su propio formato para la generación de modelos. Otra particularidad de su elección es porque no se trabaja con las plataformas en internet como lo hacen Amazon Machine Learning o Azure ML Studio a pesar de que eso tiene ventajas en cuanto a la velocidad de procesamiento, para este proyecto se requiere integrar varios *frameworks* y usar plataformas dedicadas conllevaría problemas y tiempo en su desarrollo.

1.4.4. Frameworks para Visión por Computador

Los frameworks más conocidos para visión por computador son los siguientes:

1. **OpenCV** [146]: Es un framework gratuito tanto para uso académico como para comercial. Tiene interfaces para los lenguajes C++, C, Python y Java y es compatible con Windows, Linux, Mac OS, iOS y Android. OpenCV fue desarrollado para la eficiencia computacional con un fuerte enfoque a las aplicaciones en tiempo real. En procesamiento su ventaja es que puede procesar en múltiples núcleos.
2. **SimpleCV** [147]: Es un framework *open source* desarrollado para construir aplicaciones de visión por computadora. SimpleCV tiene acceso a varias bibliotecas como OpenCV con la diferencia que una gran cantidad de parámetros ya se encuentran preconfigurados esto hace que ciertas tareas de visión por computador sea sencillas realizar.

3. **Accord.Net.** [148]: Es un framework de aprendizaje automático que combina bibliotecas de audio y procesamiento de imagen completamente escrito en C#. Es un framework completo para construir aplicaciones de visión artificial, audición por computadora y procesamiento de señales.

Para motivos del proyecto se usa OpenCV primero porque todo el código estará desarrollado en Python y requiere análisis en tiempo real de video. Por lo que un framework liviano como SimpleCV o el Accord.Net Framework desarrollado para C# no funcionarían para este proyecto.

Rastreador de Objetos

Puesto que se realizará un rastreo múltiple de objetos analizados bajo OpenCV se hace uso de la librería DLIB que es compatible con OpenCV dado que las funciones de rastreo incorporadas en OpenCV son basadas en DLIB [149]. Sus principales características son:

- Detección de objetos múltiple clase.
- Compatibilidad con CNN.
- Alta documentación.
- Interfaz de programación de aplicaciones para Python.
- Varios algoritmos de *Machine Learning*, entre otras [149].

2. METODOLOGÍA

En este capítulo se relata las diferentes etapas que se realizarán para obtener el modelo CNN-SVM. Estas etapas corresponden al entrenamiento y validación del modelo planteado. Las entradas del modelo son un conjunto de imágenes de varios tipos de vehículos, cada uno de ellos con su respectiva etiqueta ya sea vehículo o no-vehículo, las cuales deben ser estandarizadas y tener la menor cantidad de ruido posible como se mostró en la sección 1.2.1.1.

Una imagen ingresa al modelo compuesto de dos etapas: La primera es la extracción de características con el uso de Redes Neuronales Convolucionales y la segunda es la clasificación con el uso de Máquinas de Soporte Vectorial. La imagen procesada por la CNN genera un vector de características el cual será la entrada del SVM que lo clasificará de acuerdo con las etiquetas anteriormente mencionadas.

2.1. Análisis del set de datos

El primer paso a realizar antes del entrenamiento consiste en obtener el set de imágenes. En este proyecto se considera dos clases etiquetadas como vehículos y no vehículos. Como se mostró en la sección 1.2.1.1, estas dos clases deben tener gran variedad de imágenes. Una de las mejores fuentes para conseguir los datos se encuentra en Internet. A continuación, se describen las fuentes que contienen set de datos relacionados con imágenes de autos.

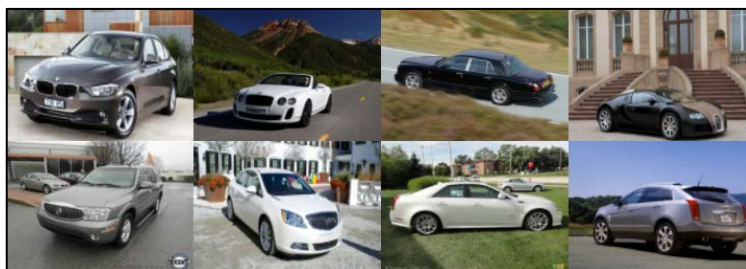


Figura 23. Muestra del de Imágenes de automóviles de la Universidad de Stanford.

1. **Universidad de Stanford:** Este set de datos contiene 16185 imágenes con 196 clases de autos. Este tipo de imágenes se caracteriza por tener un auto en el centro de cada fotografía. Cada una de ellas muestra un auto con diferentes puntos de vista; estos determinan al observador (cámara) que se posiciona de forma aleatoria en un campo de 360 grados alrededor de un auto. Las fotos están estandarizadas en un formato de codificación. Finalmente, cada fotografía tiene diferente tamaño y

un espacio con fondo diferente (montañas, construcciones, otros autos, etc.) como se puede observar en la Figura 23 [150].



Figura 24. Muestra del set de imágenes de automóviles de la Universidad Politécnica de Madrid.

- Universidad Politécnica de Madrid:** El set de datos de vehículos contiene 3425 imágenes con diferentes puntos de vista, pero solo enfocados en la parte posterior del automóvil cada una con diferente postura la postura: rango medio/cerca, frente a la cámara, rango medio/cerca, a la izquierda, rango cercano/medio a la derecha y rango lejano. Además, 3900 imágenes que no contienen automóviles. Las imágenes fueron tomadas y recortadas a 64x64 de videos de vigilancia de las autopistas de Madrid, Bruselas y Turín. Una muestra de este set de datos se puede observar en la Figura 24 [151]

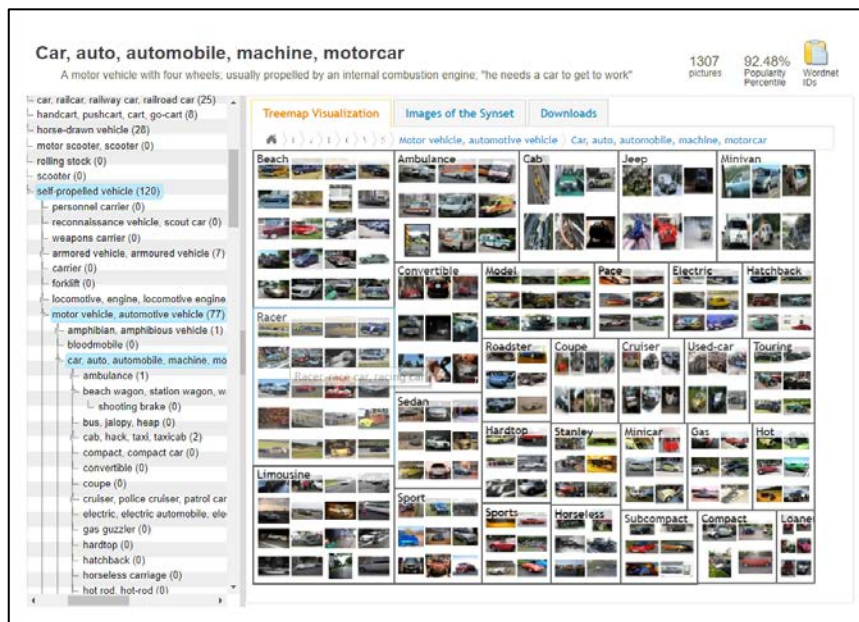


Figura 25. Muestra del set de imágenes de automóviles de IMAGENET.

- IMAGENET:** Esta es una base de datos de imágenes organizada según la jerarquía de WordNet [152], cada nodo de ésta tiene al menos quinientas imágenes por nodo.

El set de datos proporciona para descargar las imágenes las URL donde se encuentran alojadas las mismas. Desde este set de datos se descargó el link con 1307 imágenes de autos. Cada uno de ellos perteneciente a una de las 29 clases como se puede observar en la Figura 25. De este set de datos, al tener solo las referencias, existe la posibilidad de que éstas presenten enlaces a las páginas web que ya no se encuentren disponibles [153].

4. **Google Images:** En la sección de imágenes se puede tener acceso mediante el buscador a varias imágenes de autos e imágenes que no tienen autos. De forma que se puede agregar imágenes ya sea al conjunto de datos de autos o al conjunto de datos de no autos. Para este caso se realizó una agregación al set de datos de no autos. Para ello se consideró asfalto y calles sin autos como información asociada a no autos [154]. Para optimizar el trabajo de descargar cada imagen se usa aplicaciones de descarga en batch. La aplicación usada es Fatkun, ésta es una aplicación que permite descargar todas las imágenes de una página web [155].

2.1.1. Limpieza de las imágenes y Estandarización de set de datos

Una vez que se tiene el set de imágenes inicial; es necesario realizar una revisión del contenido del set debido a que éste puede presentar irregularidades. Como la extracción de las imágenes se realizó de varias fuentes, primero se debe realizar una limpieza de las imágenes y luego una estandarización del set para que el procesamiento se lleve a cabo con menos errores y genere buenos resultados. Adicionalmente una buena práctica es realizar la normalización del set de datos que consiste en dividir para 250 cada dato de entrada. [156].

Limpieza

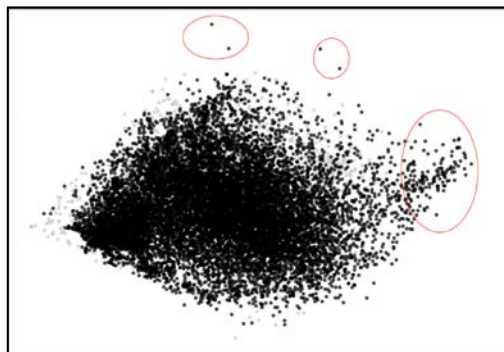


Figura 26. Representación gráfica del set de imágenes.

Para el proyecto se parte con una entrada de 18 714 imágenes de las cuales 11 983 son imágenes de autos y 6 731 son imágenes que no lo son. Luego del proceso de limpieza se espera que la reducción de ruido en el set de imágenes sea adecuada para ser usado en el entrenamiento y así generar un buen modelo. Para la limpieza se realiza una media aritmética de los canales. Una representación gráfica, generada mediante la reducción de dimensionalidad a dos, se observa en la Figura 26. Esta representación gráfica del set, hace posible que se pueda determinar si en el set existe o no ruido. Mediante un análisis se la data, se puede encontrar imágenes que se encuentran fuera de los grupos concentrados. Para eliminar el ruido, la solución es eliminar las posibles imágenes que mediante una lógica humana no se encuentran en el espectro de imágenes de entrenamiento.

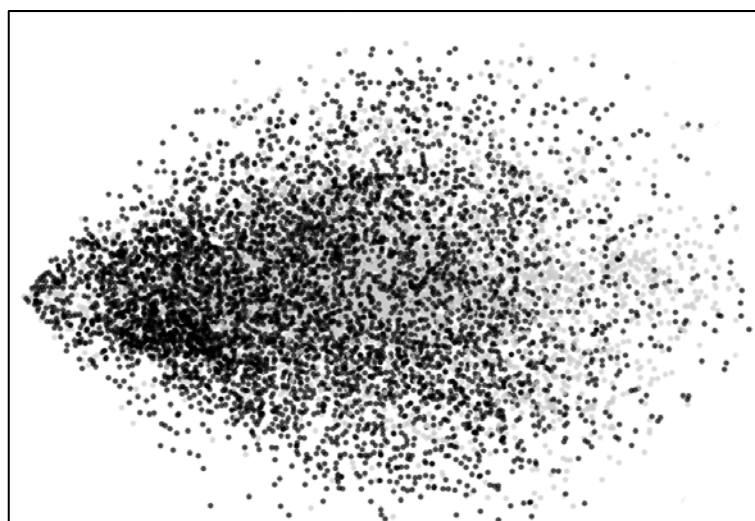


Figura 27. Muestra aleatoria luego de la limpieza del set de imágenes.

En la Figura 27, realizada reduciendo la dimensionalidad de cada imagen a dos, se observa una impresión aleatoria de cómo es el set de datos después de la limpieza. Los datos corresponden a las dos clases: muestras con etiquetas de vehículos y muestras de etiquetas no-vehículos.

Estandarización

Luego del proceso de limpieza se continua con la estandarización, el formato con que se debe tener el set de datos será similar al formato del set de datos de MNIST. Para ello se plantean los siguientes pasos:

1. Leer las imágenes desde la carpeta contenedora. Estas imágenes deben estar separadas por carpetas con la etiqueta general de vehículo o no vehículo.

2. Para cada imagen se cambia el tamaño a $127 \times 127 \times 3$, éste debe corresponder con la entrada planteada con el modelo de la ConvNet. A continuación, se verifica que corresponde al tensor $H \times W \times D$, donde D debe ser igual a 3.
3. Se almacena cada imagen en un tensor de tipo $B \times W \times H \times D$ donde B es el número de imágenes que se almacenan.
4. Se guarda las etiquetas en un tensor T con 1 si la imagen asociada presenta un vehículo caso contrario será guardado como 0, donde T es el valor de la etiqueta.
5. Normalizar cada imagen, se tiene los valores de los pixeles de 0 a 1, se realiza la división de cada pixel para 255.

Como resultado del proceso anterior se genera dos archivos: uno con imágenes en formato $B \times H \times W \times D$ y otro de etiquetas con formato $B \times T$.

Data Augmentation

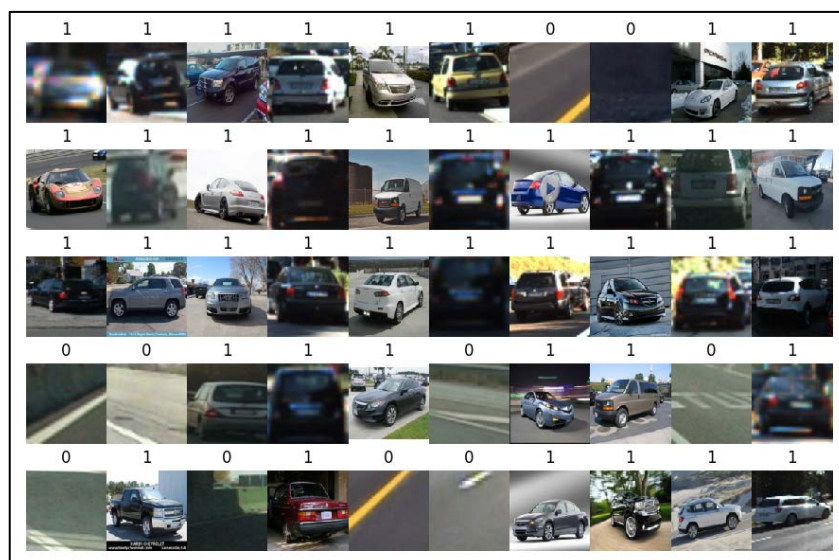


Figura 28. Resultado de Limpieza y Estandarización.

El resultado de la limpieza y estandarización se muestra las imágenes con el mismo tamaño de tensor de entrada. Para este caso son tensores $112 \times 112 \times 3$ y cada uno de ellos asociado una etiqueta correspondiente como se puede observar en la Figura 28. Sin embargo, además del proceso descrito se aplica el método de *data augmentation* sobre la base de datos de imágenes ya recolectada con el objetivo de simular condiciones que las imágenes obtenidas no tienen y así obtener mayor variedad de imágenes de entrada.

Las modificaciones realizadas a las imágenes son:

- Volteo de la imagen en las direcciones horizontal o vertical.
- Recortes en un 10% del contorno de la imagen en forma horizontal y vertical.

- Desenfoque gaussiano de hasta un 30%.
- Ruido gaussiano por cada canal de hasta un 20%.
- Eliminación aleatoria de forma aleatoria hasta un 10% de los píxeles de la imagen.
- Agregación o sustracción del brillo a las imágenes de entre un 10% por cada canal.
- Agregación o sustracción de contraste de hasta un 20%.

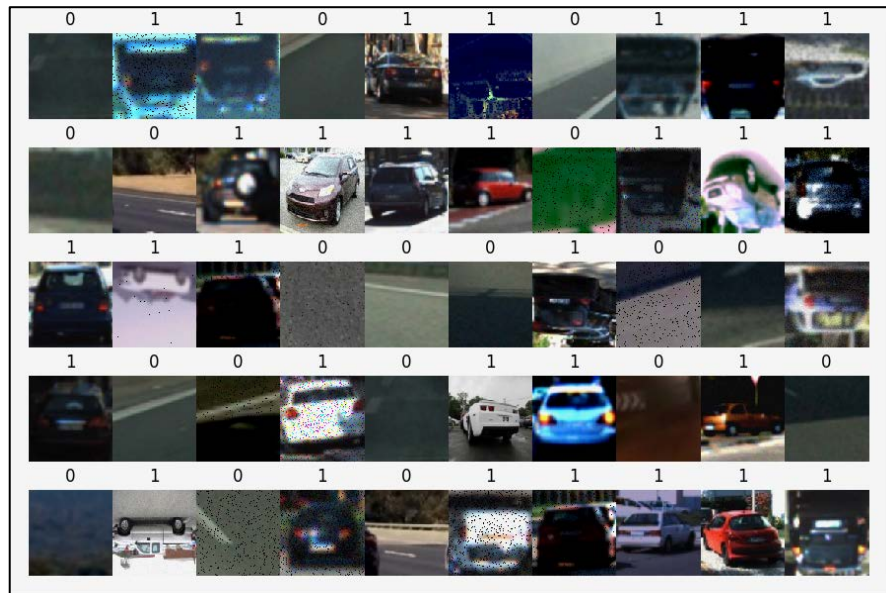


Figura 29. Resultado de la aplicación de data augmentation.

Cada una de estas modificaciones a la imagen real se la realiza de forma aleatoria y cada nueva imagen resultante se la guarda con su respectiva etiqueta como se puede observar en la Figura 29.

2.1.2. Resultado de limpieza del set de datos.

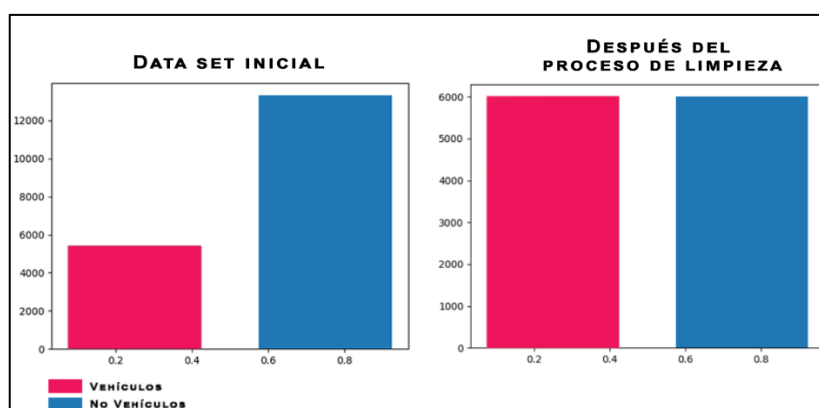


Figura 30. Resultado de la limpieza del set de datos.

El proceso que lleva el set de datos después de ser recopilado inicialmente como se mira en la parte izquierda de la Figura 30 debe ser sometido al proceso de limpieza y

estandarización. En la limpieza se eliminan las imágenes que por error se filtraron como parte de este set particular. Luego en la estandarización y data aumentación, primero se lleva todas las imágenes a un solo formato y tamaño; y a partir de eso se trata de igualar los sets de vehículos y no vehículos. Al final del proceso se tiene 6 000 imágenes por cada clase como se mira en la parte derecha de la Figura 30.

2.2. Modelo ConvNet – SVM

Bajo la motivación de [157], [158] y [159] se realizará una arquitectura combinando un modelo de una red neuronal convolucional con un clasificador SVM. A continuación se plantea una arquitectura ConvNet que toma como base las arquitecturas de AlexNet [158] e *Inception* [160]. Ésta tiene como objetivo en primera instancia extraer las características de cada una de las imágenes del set de entrada y posterior ingresar dichos vectores de características al clasificador SVM [161].

Tabla 3. Arquitectura inicial de la ConvNet.

	K		S		Borde	g	Tensor	Memoria		Pesos	
	h	w	h	w							
INPUT							[112x112x3]	112*112*3=	150		
CONV	25	25	3	3	same	1	[38x38x32]	38*38*64=	92416	(3*3*64)*64=	36864
RELU							[38x38x32]				
LRN							[38x38x32]				
MAXPOOL	3	3	2	2	valid		[18x18x32]	18*18*32=	10368		
CONV	3	3	1	1	same	2	[18x18x64]	18*18*64=	20736	(3*3*64)*64=	36864
RELU							[18x18x64]				
MAXPOOL	3	3	2	2	valid		[8x8x64]	8*8*64=	4096		
CONV	3	3	1	1	same	1	[8x8x128]	8*8*128=	8192	(3*3*128)*128=	147456
CONV	3	3	1	1	same	2	[8x8x128]	8*8*128=	8192	(3*3*128)*128=	147456
CONV	3	3	1	1	same	2	[8x8x128]	8*8*128=	8192	(3*3*128)*128=	147456
MAXPOOL	3	3	2	2	valid		[3x3x128]	3*3*128=	0	(3*3*128)*128=	0
FC	256	256	1	1			[1x1x1152]	1*1*1152=	1152	(3*3*1152)*1152=	11943936
FC	1152	1	1	1			[1x1x256]	1*1*256=	256	(3*3*256)*256=	589824
FC	256	1	1	1			[1x1x2]	1*1*2=	2	(3*3*2)*2=	36
								112*112*3=	150		
Tamaño total de memoria necesaria para procesar una imagen de tamaño 112x112x3 en el modelo anterior									153752	Bites	
									19,219	KB	
Total de pesos a entrenar en el modelo anterior									50319396		

El resumen de la arquitectura que procesará tensores de 112x112x3 es el que se muestra en la Tabla 3. En ella, en la primera columna de la tabla se encuentra el nombre de las capas que la componen. En las siguientes columnas, se presenta los hiperparámetros usados: Kernel, Salto y Borde. Para cada capa de la ConvNet se presentan varios tipos de configuraciones que dan como resultado la columna de Tensores. Además, se tiene otra de Memoria y Pesos que permite conocer cuántos recursos computacionales son

necesarios para cada capa y cuantos pesos deberán ser entrenados por cada capa respectivamente.

2.2.1. Hiperparámetros e Inicialización de pesos

Los hiperparámetros y los pesos iniciales que se asignan a las neuronas juegan un papel importante en el entrenamiento y los resultados de la validación de la ConvNet y del SVM. Para la CNN se definen hiperparámetros de la Tabla 4:

Tabla 4. Hiperparámetros de CNN.

Hiperparámetro	Valor
Taza de aprendizaje	0,005
Tamaño de batch de entrada	112
Pérdida de peso	0,001
Momentum	Adagrad
Total de épocas	20
Hiperparámetro	Valor

Los pesos iniciales para la CNN se definen como aleatorios y sus valores se encuentran entre 0 y 0,5 [30]. Para SVM se definen los hiperparámetros de la Tabla 5.

Tabla 5. Hiperparámetros de SVM.

Kernel	Hiperparámetro	Valor
Linear	C (coeficiente de Penalización)	0,1
	γ (Gamma)	0,1
Radial Basis Function (RBF)	C (coeficiente de Penalización)	1
	γ (Gamma)	$\frac{1}{(\text{total de entradas})}$

En la Figura 31 se puede observar una sección de la red neuronal convolucional del modelo planteado hasta la primera capa totalmente conectada. El modelo toma como entrada tensores de 112x112x3 y presenta como salida la clasificación de auto o no auto.

2.2.2. Esquema de la CNN-SVM

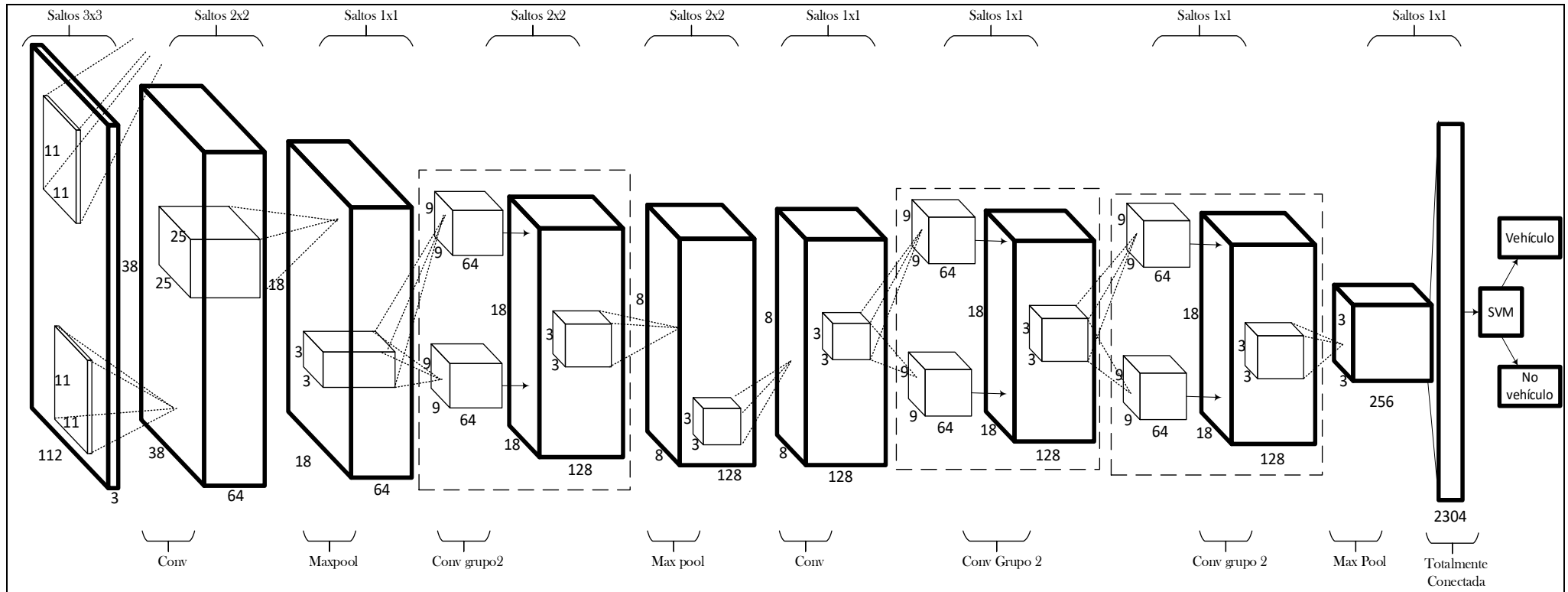


Figura 31. Modelo propuesto del CNN-SVM.

Una lectura detallada de la Figura 31 es la siguiente:

1. Se toma como entrada una imagen con dimensiones 112x112x3.
2. A la entrada se le aplica una convolución con kernel de tamaño de 25x25, saltos de 3x3. Como resultado se tiene un tensor de tamaño 38x38x32.
3. A continuación, en la capa de agrupado se aplica la operación max pooling con un kernel de tamaño 3x3 y saltos de 2x2. Esto genera como salida un tensor de tamaño 18x18x32.
4. La siguiente capa es una convolución modificada, los filtros que se aplican son: kernel de 3x3 y salto de 1x1. La modificación es que por cada sector se realizarán dos convoluciones que luego serán agrupadas (basado en la arquitectura ConvNet Inception) dando como resultado un tensor de tamaño 12x12x64.
5. Se aplica la operación max pooling con un kernel de tamaño 3x3 y saltos de 2x2. Como resultado se genera un tensor de tamaño 8x8x128.
6. Se ejecuta una convolución normal con kernel de 3x3 y saltos de 1x1, como resultado se tiene el tensor 8x8x128.
7. Se ejecuta una convolución modificada con kernel de 3x3 y saltos de 1x1, como resultado se tiene el tensor 8x8x128.
8. Se ejecuta una convolución modificada con kernel de 3x3 y saltos de 1x1, como resultado se tiene el tensor 8x8x128.
9. Se aplica la operación max pooling con un kernel de tamaño 3x3 y saltos de 2x2. Como resultado se genera un tensor de tamaño 3x3x256.
10. La siguiente es una capa totalmente conectada que genera el vector de características de tamaño 1x1x2304.
11. A continuación, realiza el procesamiento con el SVM el cual produce como resultado la clasificación de vehículo o no vehículo.

2.3. Entrenamiento y Validación del modelo CNN-SVM

La generación de un modelo para la clasificación correcta de objetos debe atravesar varias fases o etapas de entrenamiento y validación. Una vez desarrollado un modelo inicial tanto para la CNN como para el SVM, éste se debe afinar hasta tener buenos valores de clasificación. De forma estadística, entre más se aproximen a 100% de clasificación el modelo será considerado adecuado para ser usado como clasificador debido a que el modelo obtenido puede estimar con mayor exactitud la clasificación de una entrada. En el caso de que en la primera etapa los resultados no sean favorables se deberá corregir el modelo o realizar ajustes en los hiperparámetros tanto en la CNN como en SVM y a

continuación se debe volver a realizar el entrenamiento y la validación del modelo. El proceso debe ser repetido hasta conseguir los mejores resultados posibles. Cabe recalcar que para la realización de este proceso se considera que el set de datos corresponde al 100%, de éste se toma de forma aleatoria un 70% para el entrenamiento y un 30% para la validación.

El modelo CNN – SVM tiene dos subetapas independientes: primero, se inicia con el entrenamiento de forma independiente de la ConvNet. Los resultados experimentales muestran que el mapa de características debe ser tomado desde la primera capa totalmente conectada [159]. La extracción de características de la primera capa totalmente conectada de una imagen en la ConvNet da como resultado el tensor BxT donde B es la cantidad de imágenes extraídas y T es el tamaño que presenta la capa totalmente conectada. A continuación, el tensor es ingresado en el SVM como datos de entrenamiento y validación. De este proceso se tiene el modelo que clasifica las entradas. Una de las pruebas comunes para validar el modelo, se realiza con la Matriz de Confusión sobre el modelo CNN – SVM en la cual verifica cuan efectivo es el modelo. Todas las fases tienen la siguiente información común para la subetapa CNN:

- a) Tensor de entrada de cada imagen = (112, 112, 3).
- b) Número de clases = 2.
- c) Número de muestras de entrenamiento = 8400.
- d) Número de muestras de validación = 3600.
- e) Número de muestras total de vehículos para entrenamiento = 4203.
- f) Número de muestras totales de no vehículos para entrenamiento = 4197.
- g) Tensor de imágenes de entrada para CNN: (8400, 112, 112, 3).
- h) Tensor de etiquetas de entrada para CNN: (8400).
- i) Número de capas de entrenamiento y validación = 20 (última capa).
- j) Se aplica *Dropout* para las capas FC solo en el entrenamiento.
- k) La dimensión del tensor inicial considera la región de características inicial a analizar. Ésta depende del tamaño de la imagen de entrada. Sin embargo, en cada capa de convolución y agrupado, el tamaño del tensor se duplicará [30].

Para todas las fases se tiene la siguiente información común para la subetapa SVM:

- a) Tensor del Vector de características: (19335, 256).
- b) Conjunto de entrenamiento: (12954, 2).
- c) Conjunto de validación: (6381, 2).

Todas las imágenes que muestran la representación de la extracción de características fueron realizadas aplicando PCA (*Principal Component Analysis*) un procedimiento que usa una transformación ortogonal (ortogonalización de vectores) para convertir un conjunto de entrada con variables correlacionadas en un conjunto de valores de variables linealmente no correlacionadas conocidas como componentes principales.

Las gráficas de Exactitud y Pérdida representan lo siguiente:

- La Exactitud representa la medida que el clasificador predice correctamente una entrada sobre a cuál clase pertenece.
- La pérdida representa la media aritmética de la función de pérdida de todas las entradas y representa la medida en que el clasificador predice incorrectamente. Para su cálculo se usa *softmax cross entropy* y lo que se presenta en los cuadros de resumen “Última época analizada” de cada fase es la pérdida total de cada época siguiendo la Ecuación 20.

2.3.1. Fase Uno

La primera fase inicia con la información del modelo previamente desarrollado en la sección 2.2. Esta sub-fase a realizar es el entrenamiento de la ConvNet. Se toma como entrada el set de imágenes que ya pasó por la fase de limpieza y estandarización. Al tensor se lo ingresa en la red neuronal convolucional y al procesarlo se presenta la siguiente información:

Sub-Fase Uno: Entrenamiento y Validación Modelo CNN

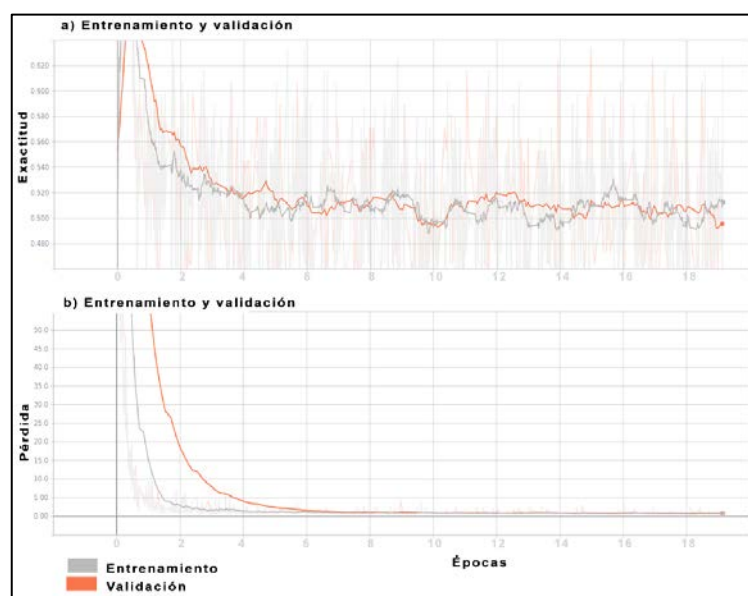


Figura 32. Fase Uno: Exactitud y Pérdida.

En la Figura 32 se muestra el proceso de entrenamiento y validación de la ConvNet. El resultado para la última capa analizada, tanto para el entrenamiento como para la validación se muestra en la Tabla 6.

Tabla 6. Fase Uno: Valores presentes en la última época analizada en CNN.

	Entrenamiento	Validación
Exactitud	0,5113	0,4857
Pérdida	0,8100	0,8702

Para la exactitud, la diferencia entre la validación y el entrenamiento para la última época es -0,0256. Lo que indica que existe un error en aprender el modelo con los datos de entrenamiento. Para la función de pérdida se verifica que tanto para el entrenamiento como para la validación su valor es de 0,81 y 0,87 respectivamente. Esto indica que la red tiene poca pérdida en su aprendizaje y que los hiperparámetros seleccionados generan un buen modelo.

Sub-Fase Uno: Entrenamiento y Validación Modelo SVM

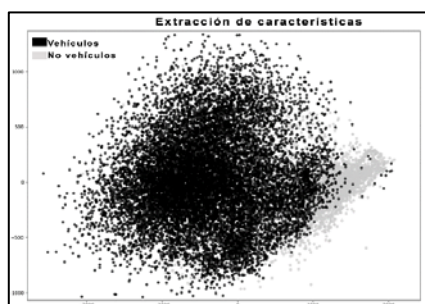


Figura 33. Fase Uno: Representación de la extracción de características.

En la siguiente sub-fase se parte del vector de características extraído de cada imagen por el proceso de la CNN. Gráficamente la extracción de características se puede observar en la Figura 33. Partiendo de esta información se realiza el proceso de entrenamiento y validación.

Tabla 7. Fase Uno: entrenamiento y validación SVM.

	Kernel lineal	Kernel RBF
Hiperparámetros	C=0,1	C=0,1
	Gamma = 0,1	Gamma = 1/19335
Resultados (Exactitud)	Entrenamiento = 0,89895	Entrenamiento = 0,97881272
	Validación = 0,90377	Validación = 0,7230841

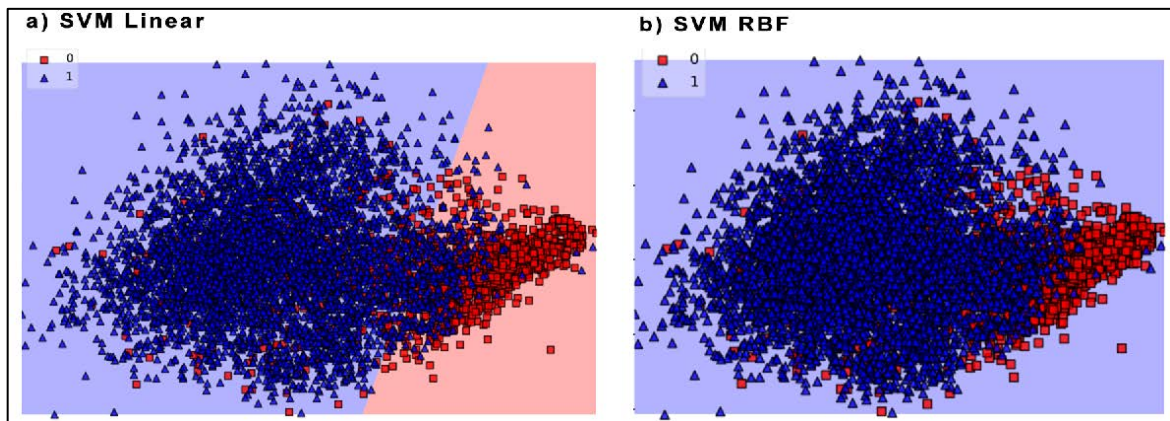


Figura 34. Fase Uno: Comparación clasificadores SVM.

Los resultados sobre el entrenamiento y validación del SVM se muestran en la Tabla 7. Para la exactitud del modelo con kernel lineal, la diferencia entre el entrenamiento y la validación presenta un valor de 0,005. Lo que indica que el conjunto de datos es adecuado para generar un buen modelo. Para el caso de kernel RBF el análisis es similar pero no genera un buen modelo. En la Figura 34 se observa el resultado de la generación del vector de soporte que clasificará las muestras para cada uno de los casos analizados.

Validación del clasificador CNN-SVM

		CONDICIONES VERDADERAS	
		No vehículo	Vehículo
Clasificación SVM	0	1385	393
	1	326	4277
		0	1

Figura 35. Fase Uno: Matriz de confusión realizada con SVM kernel lineal.

Al modelo generado por el SVM con kernel lineal, se aplica el método de matriz de confusión (1.3.3) para validar la generalización del clasificador CNN-SVM. En la Figura 35 se muestra la matriz de confusión realizada sobre el conjunto de validación. De la matriz se extrae la siguiente información:

- 1385 son verdaderos negativos que se predijeron correctamente como no autos.

- 4277 son verdaderos positivos que se predijeron correctamente como autos.
- 393 son falsos negativos que se predijeron como autos, pero no lo eran.
- 326 son falsos positivos que se predijeron como no autos, pero si lo eran.

Tabla 8. Fase Uno: Análisis de la Matriz de Confusión.

Propiedad Matriz de Confusión	Coficiente
AC	0,887
ERR	0,113
TPR	0,916
FPR	0,191
TNR	0,809
FNR	0,084
MCC	0,725

En la Tabla 8 se muestra el análisis de la matriz de confusión. Para el modelo de la fase uno su exactitud es de 88,7%, su precisión es de 91,6% y el coeficiente Matthews (MCC) es de 0,725. De esta fase se concluye que: el modelo es una buena línea base de partida.

2.3.2. Fase Dos

Tabla 9. Fase Dos: Modelo ConvNet.

	K		S		Borde	g	Tensor	Memoria		Pesos	
	h	w	h	w							
INPUT							[112x112x3]	112*112*3=	150		
CONV	11	11	3	3	same	1	[38x38x64]	38*38*64=	92416	(3*3*64)*64=	36864
RELU							[38x38x64]				
LRN							[38x38x64]				
MAXPOOL	3	3	2	2	valid		[18x18x64]	18*18*64=	20736		
CONV	3	3	1	1	same	2	[18x18x128]	18*18*128=	41472	(3*3*128)*128=	147456
RELU							[18x18x128]				
MAXPOOL	3	3	2	2	valid		[8x8x128]	8*8*128=	8192		
CONV	3	3	1	1	same	1	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
CONV	3	3	1	1	same	2	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
CONV	3	3	1	1	same	2	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
MAXPOOL	3	3	2	2	valid		[3x3x256]	3*3*256=	2304	(3*3*256)*256=	589824
FC	256	256	1	1			[1x1x2304]	1*1*2304=	2304	(3*3*2304)*2304=	47775744
FC	2304	1	1	1			[1x1x256]	1*1*256=	256	(3*3*256)*256=	589824
FC	256	1	1	1			[1x1x2]	1*1*2=	2	(3*3*2)*2=	36
Tamaño total de memoria necesaria para procesar una imagen de tamaño 112x112x3 en el modelo anterior									216984	Bites	
									27,123	KB	
Total de pesos a entrenar en el modelo anterior									50909220		

En la fase dos se realiza una modificación en la arquitectura de la ConvNet. Los cambios son: el tamaño del kernel en la primera convolución y el tamaño de la dimensión del tensor de salida. Estas modificaciones se observan en la Tabla 9.

Sub-Fase Dos: Entrenamiento y Validación Modelo CNN

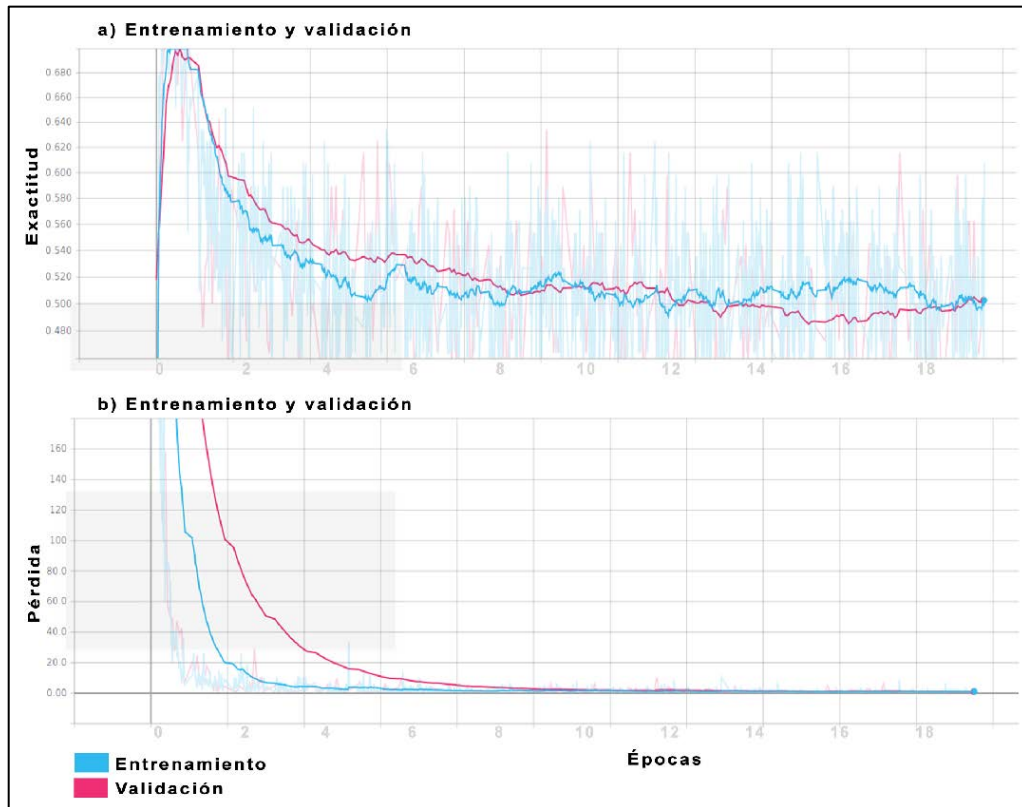


Figura 36. Fase Dos: Exactitud y pérdida.

En la Figura 36 se muestra el proceso de entrenamiento y validación de la ConvNet. El resultado para la última capa analizada, tanto para el entrenamiento como para la validación se muestra en la Tabla 10.

Tabla 10. Fase Dos: Valores presentes en la última época analizada en CNN.

	Entrenamiento	Validación
Exactitud	0,5015	0,5143
Pérdida	1,1732	0,8038

Para la exactitud, la diferencia entre el entrenamiento y la validación para la última época es 0,0128. Lo que indica que el conjunto de datos es adecuado para generar un buen modelo. Para la función de pérdida se verifica que tanto para el entrenamiento como para

la validación su valor es de 1.17 y 0,80 respectivamente. Esto indica que la red tiene en el entrenamiento una clasificación incorrecta para ciertas entradas.

Sub-Fase Dos: Entrenamiento y Validación Modelo SVM

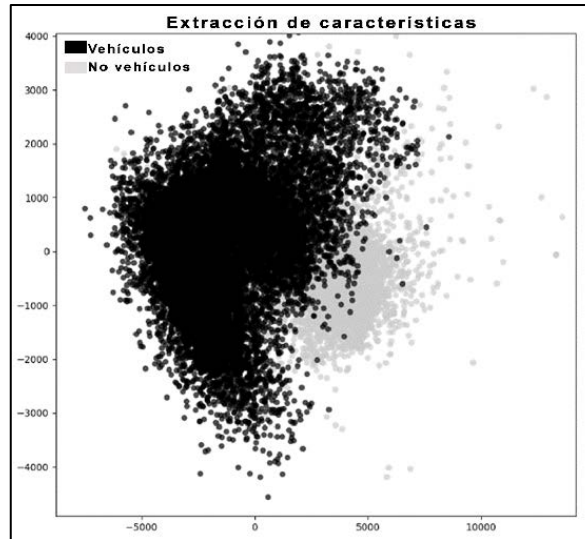


Figura 37. Fase Dos: Representación de la extracción de características.

En la siguiente sub-fase se parte del vector de características extraído de cada imagen por el proceso de la CNN. Gráficamente la extracción de características se puede observar en la Figura 37. Partiendo de esta información se realiza el proceso de entrenamiento y validación.

Tabla 11. Fase Dos: Entrenamiento y validación SVM.

	kernel lineal	kernel RBF
Hiperparámetros	C=0,1	C=0,1
	Gamma = 0,1	Gamma = 1/19335
Resultados (Exactitud)	Entrenamiento = 0,88829	Entrenamiento = 0,9669272
	Validación = 0,893276	Validación = 0,72190841

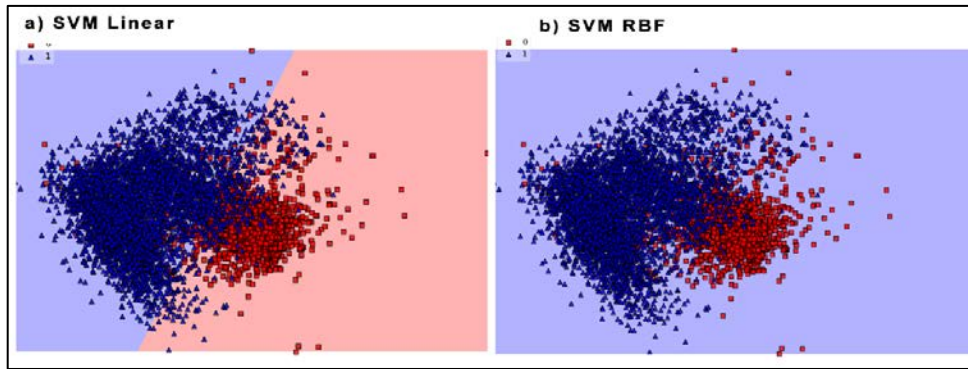


Figura 38. Fase Dos: Comparación clasificadores SVM.

Los resultados sobre el entrenamiento y validación del SVM se muestran en la Tabla 11. Para la exactitud del modelo con kernel lineal, la diferencia entre el entrenamiento y la validación presenta un valor de 0,005. Lo que indica que el conjunto de datos es adecuado para generar un buen modelo. Para el caso de kernel RBF el análisis es similar pero no genera un buen modelo. En la Figura 38 se observa el resultado de la generación del vector de soporte que clasificará las muestras para cada uno de los casos analizados.

Validación del clasificador CNN-SVM

		MATRIZ DE CONFUSIÓN	
		Condiciones Verdaderas	
		No vehículo	Vehículo
Clasificación SVM	0 No vehículo	1518	260
	1 Vehículo	421	4182
		0	1

Figura 39. Fase Dos: Matriz de confusión realizada con SVM kernel lineal.

Al modelo generado por el SVM con kernel lineal, se aplica el método de matriz de confusión (1.3.3) para validar la generalización del clasificador CNN-SVM. En la Figura 39 se muestra la matriz de confusión realizada sobre el conjunto de validación. De la matriz se extrae la siguiente información:

- 1518 son verdaderos negativos que se predijeron correctamente como no autos.
- 4182 son verdaderos positivos que se predijeron correctamente como autos.

- 260 son falsos negativos que se predijeron como autos, pero no lo eran.
- 421 son falsos positivos que se predijeron como no autos, pero si lo eran.

Tabla 12. Fase Dos: Análisis de la Matriz de Confusión.

Propiedad Matriz de Confusión	Coficiente
AC	0,893
ERR	0,107
TPR	0,941
FPR	0,217
TNR	0,783
FNR	0,059

MCC	0,724
------------	--------------

En la Tabla 12 se muestra el análisis de la matriz de confusión. Para el modelo de la fase uno su exactitud es de 89,3%, su precisión es de 94,1% y el coeficiente Matthews (MCC) es de 0,724. De esta fase se concluye que a pesar de que es mejor que en la Fase Uno, aun puede mejorar el modelo de clasificación.

2.3.3. Fase Tres

Tabla 13. Fase Tres: Modelo ConvNet.

	K		S		Borde	g	Tensor	Memoria		Pesos	
	h	w	h	w							
INPUT							[112x112x3]	112*112*3=	150		
CONV	33	33	3	3	same	1	[38x38x64]	38*38*64=	92416	(3*3*64)*64=	36864
RELU							[38x38x64]				
LRN							[38x38x64]				
MAXPOOL	3	3	2	2	valid		[18x18x64]	18*18*64=	20736		
CONV	3	3	1	1	same	2	[18x18x128]	18*18*128=	41472	(3*3*128)*128=	147456
RELU							[18x18x128]				
MAXPOOL	3	3	2	2	valid		[8x8x128]	8*8*128=	8192		
CONV	3	3	1	1	same	1	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
CONV	3	3	1	1	same	2	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
CONV	3	3	1	1	same	2	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
MAXPOOL	3	3	2	2	valid		[3x3x256]	3*3*256=	2304	(3*3*256)*256=	589824
FC	256	256	1	1			[1x1x2304]	1*1*2304=	2304	(3*3*2304)*2304=	47775744
FC	2304	1	1	1			[1x1x256]	1*1*256=	256	(3*3*256)*256=	589824
FC	256	1	1	1			[1x1x2]	1*1*2=	2	(3*3*2)*2=	36
Tamaño total de memoria necesaria para procesar una imagen de tamaño 112x112x3 en el modelo anterior									216984	Bites	
									27,123	KB	
Total de pesos a entrenar en el modelo anterior									50909220		

En la fase dos se realiza una modificación en la arquitectura de la ConvNet. El cambio con respecto a Fase Dos es: el tamaño del kernel en la primera convolución. Esta modificación se observa en la Tabla 13.

Sub-Fase Tres: Entrenamiento y Validación Modelo CNN

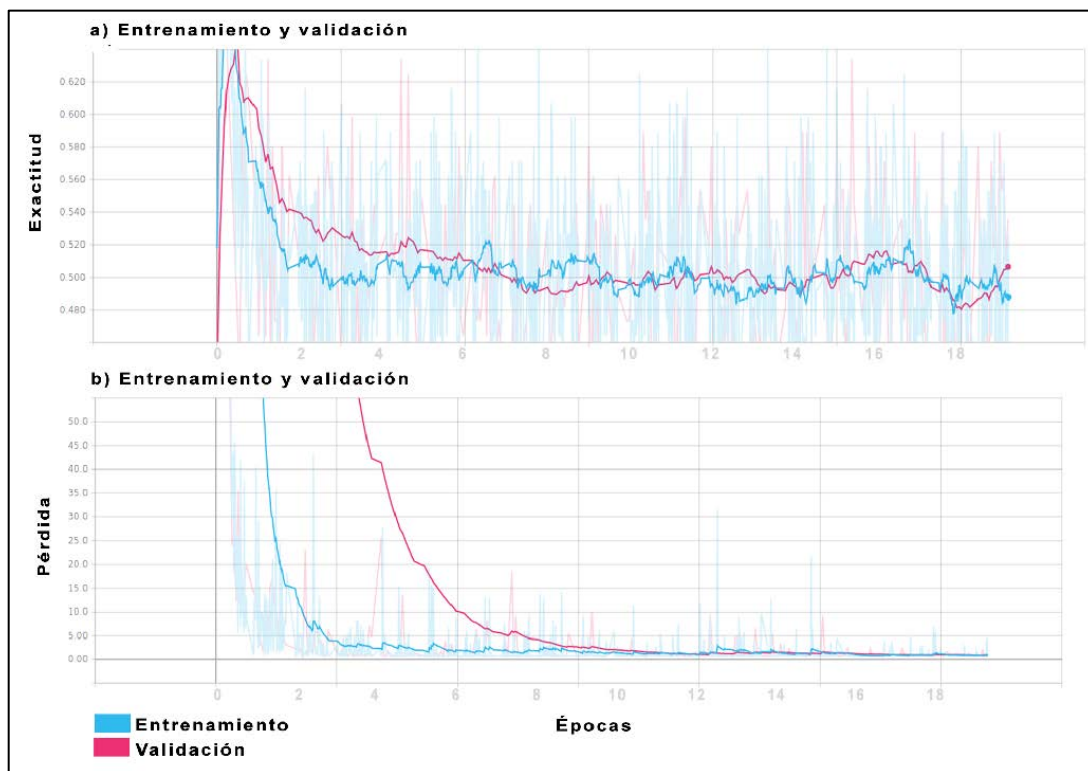


Figura 40. Fase Tres: Exactitud y pérdida.

En la Figura 40 muestra el proceso de entrenamiento y validación de la ConvNet. El resultado para la última capa analizada, tanto para el entrenamiento como para la validación se muestra en la Tabla 14.

Tabla 14. Fase Tres: Valores presentes en la última época analizada en CNN.

	Entrenamiento	Validación
Exactitud	0,4910	0, 5256
Pérdida	0,8858	0,7880

Para la exactitud, la diferencia entre el entrenamiento y la validación para la última época es 0,0035. Lo que indica que el conjunto de datos es adecuado para generar un buen modelo. Para la función de pérdida se verifica que tanto para el entrenamiento como para la validación su valor es de 0,89 y 0,80 respectivamente. Esto indica que la red tiene una baja pérdida en su aprendizaje y que los hiperparámetros generan un buen modelo.

Sub-Fase Tres: Entrenamiento y Validación Modelo SVM

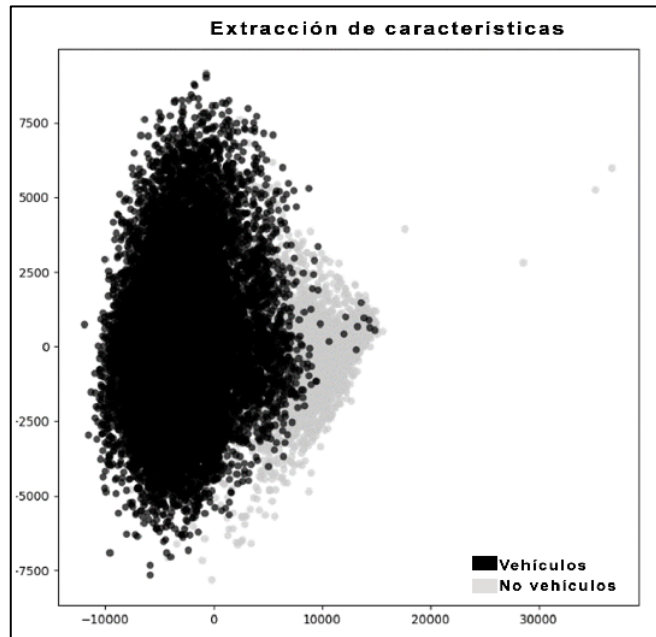


Figura 41. Fase Tres: Representación de la extracción de características.

En la siguiente sub-fase se parte del vector de características extraído de cada imagen por el proceso de la CNN. Gráficamente la extracción de características se puede observar en la Figura 41. Partiendo de esta información se realiza el proceso de entrenamiento y validación.

Tabla 15. Fase Tres: Entrenamiento y validación SVM.

	kernel lineal	kernel RBF
Hiperparámetros	C=0,1	C=0,1
	Gamma = 0,1	Gamma = 1/19335
Resultados (Exactitud)	Entrenamiento = 0,8987	Entrenamiento = 0,96743
	Validación = 0,8975	Validación = 0,72292

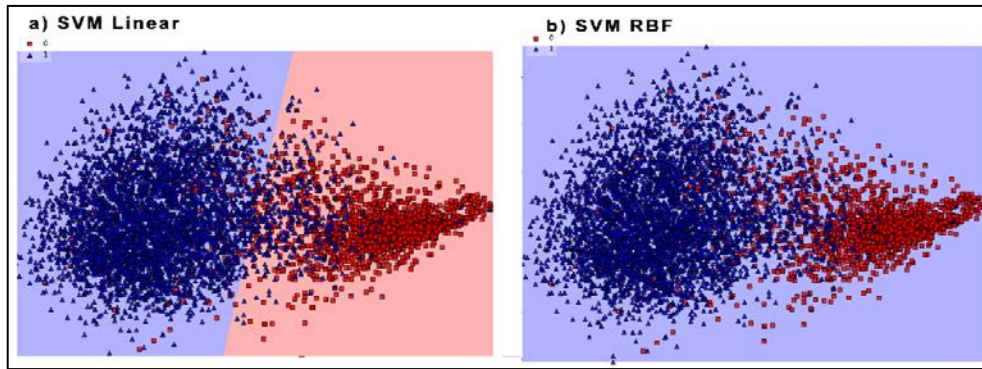


Figura 42. Fase Tres: Comparación de clasificadores SVM.

Los resultados sobre el entrenamiento y validación del SVM se muestran en la Tabla 15. Para la exactitud del modelo con kernel lineal, la diferencia entre el entrenamiento y la validación presenta un valor de 0,001. Lo que indica que el conjunto de datos es adecuado para generar un buen modelo. Para el caso de kernel RBF el análisis es similar pero no genera un buen modelo. En la Figura 42 se observa el resultado de la generación del vector de soporte que clasificará las muestras para cada uno de los casos analizados.

		MATRIZ DE CONFUSIÓN	
		Condiciones Verdaderas	
		No vehículo	Vehículo
Clasificación SVM	0 No vehículo	1529	249
	1 Vehículo	387	4216
		0	1

Figura 43. Fase Tres: Matriz de confusión realizada con SVM kernel lineal.

Al modelo generado por el SVM con kernel lineal, se aplica el método de matriz de confusión (1.3.3) para validar la generalización del clasificador CNN-SVM. En la Figura 43 se muestra la matriz de confusión realizada sobre el conjunto de validación. De la matriz se extrae la siguiente información:

- 1529 son verdaderos negativos que se predijeron correctamente como no autos.
- 4216 son verdaderos positivos que se predijeron correctamente como autos.
- 249 son falsos negativos que se predijeron como autos, pero no lo eran.
- 487 son falsos positivos que se predijeron como no autos, pero si lo eran.

Tabla 16. Fase Tres: Análisis de la Matriz de Confusión.

Propiedad Matriz de Confusión	Coefficiente
AC	0,900
ERR	0,100
TPR	0,944
FPR	0,202
TNR	0,798
FNR	0,056

MCC	0,742
------------	--------------

En la Tabla 16 se muestra el análisis de la matriz de confusión. Para el modelo de la fase uno su exactitud es del 90%, su precisión es de 94,4% y el coeficiente Matthews es de 0,742. De esta fase se concluye que: a pesar de que es mejor que en la Fase Dos, aun puede mejorar el modelo de clasificación.

2.3.4. Fase Cuatro

Tabla 17. Fase Cuatro: Modelo ConvNet.

	K		S		Borde	g	Tensor	Memoria		Pesos	
	h	w	h	w							
INPUT							[112x112x3]	112*112*3=	150		
CONV	35	35	3	3	same	1	[38x38x64]	38*38*64=	92416	(3*3*64)*64=	36864
RELU							[38x38x64]				
LRN							[38x38x64]				
MAXPOOL	3	3	2	2	valid		[18x18x64]	18*18*64=	20736		
CONV	3	3	1	1	same	2	[18x18x128]	18*18*128=	41472	(3*3*128)*128=	147456
RELU							[18x18x128]				
MAXPOOL	3	3	2	2	valid		[8x8x128]	8*8*128=	8192		
CONV	3	3	1	1	same	1	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
CONV	3	3	1	1	same	2	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
CONV	3	3	1	1	same	2	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
MAXPOOL	3	3	2	2	valid		[3x3x256]	3*3*256=	2304	(3*3*256)*256=	589824
FC	256	256	1	1			[1x1x2304]	1*1*2304=	2304	(3*3*2304)*2304=	47775744
FC	2304	1	1	1			[1x1x256]	1*1*256=	256	(3*3*256)*256=	589824
FC	256	1	1	1			[1x1x2]	1*1*2=	2	(3*3*2)*2=	36
Tamaño total de memoria necesaria para procesar una imagen de tamaño 112x112x3 en el modelo anterior									216984	Bites	
									27,123	KB	
Total de pesos a entrenar en el modelo anterior									50909220		

En la fase dos se realiza una modificación en la arquitectura de la ConvNet. El cambio con respecto a Fase Tres es: el tamaño del kernel en la primera convolución. Esta modificación se observa en la Tabla 17.

Sub-Fase Cuatro: Entrenamiento y Validación Modelo CNN

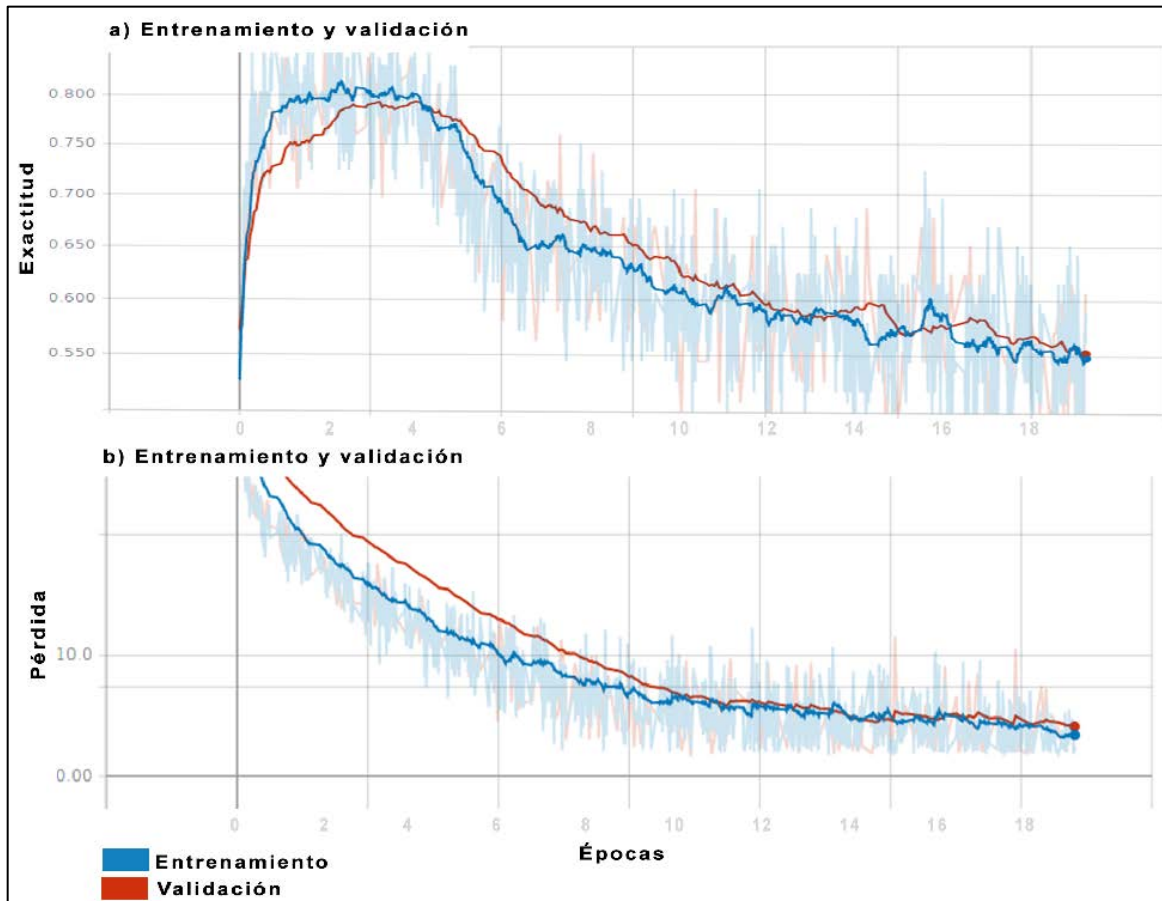


Figura 44. Fase Cuatro: Exactitud y pérdida.

En la Figura 44 muestra el proceso de entrenamiento y validación de la ConvNet. El resultado para la última capa analizada, tanto para el entrenamiento como para la validación se muestra en la Tabla 18.

Tabla 18. Fase Cuatro: Valores presentes en la última época analizada en CNN.

	Entrenamiento	Validación
Exactitud	0,5477	0,5440
Pérdida	1,2958	1,8800

Para la exactitud, la diferencia entre el entrenamiento y la validación para la última época es -0,004. Lo que indica que existe un error en aprender el modelo con los datos de

entrenamiento. Para la función de pérdida se verifica que tanto para el entrenamiento como para la validación su valor es de 1,30 y 1,88 respectivamente. Esto indica que tanto el entrenamiento como la validación tienen errores en la clasificación correcta de algunas entradas.

Sub-Fase Cuatro: Entrenamiento y Validación Modelo SVM

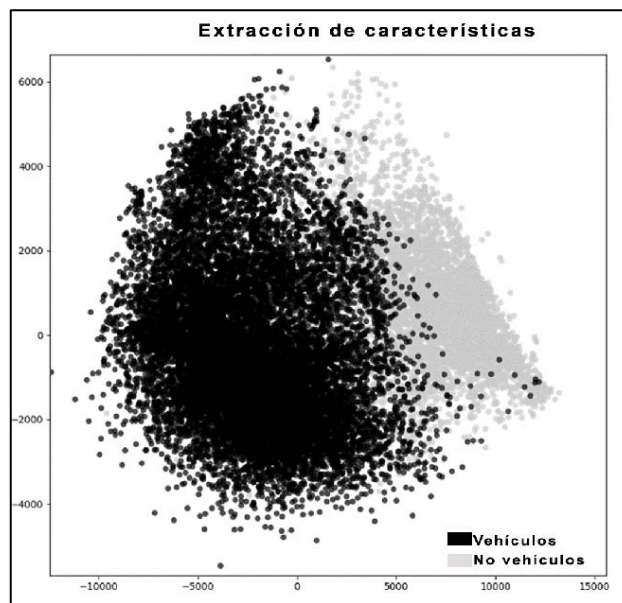


Figura 45. Fase Cuatro: Representación de la extracción de características.

En la siguiente sub-fase se parte del vector de características extraído de cada imagen por el proceso de la CNN. Gráficamente la extracción de características se puede observar en la Figura 45. Partiendo de esta información se realiza el proceso de entrenamiento y validación.

Tabla 19. Fase Cuatro: Entrenamiento y validación SVM.

	kernel lineal	kernel RBF
Hiperparámetros	C=0,1	C=0,1
	Gamma = 0,1	Gamma = 1/19335
Resultados (Exactitud)	Entrenamiento = 0,8987	Entrenamiento = 0,9666
	Validación = 0,8975	Validación = 0,72308

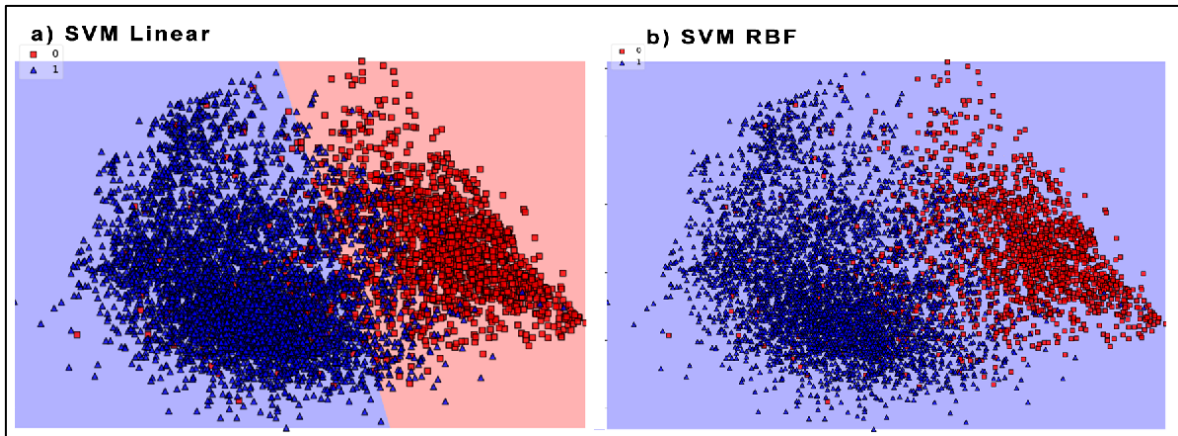


Figura 46. Fase Cuatro: Comparación de clasificadores SVM.

Los resultados sobre el entrenamiento y validación del SVM se muestran en la Tabla 19. Para la exactitud del modelo con kernel lineal, la diferencia entre el entrenamiento y la validación presenta un valor de -0,0012. Lo que indica que existe un error en aprender el modelo con los datos de entrenamiento. Para el caso de kernel RBF el análisis es similar pero no genera un buen modelo. En la Figura 46 se observa el resultado de la generación del vector de soporte que clasificará las muestras para cada uno de los casos analizados.

		CONDICIONES VERDADERAS	
		No vehículo	Vehículo
Clasificación SVM	0	1516	262
	1	392	4211
		0	1

Figura 47. Fase Cuatro: Matriz de confusión realizada con SVM kernel lineal.

Al modelo generado por el SVM con kernel lineal, se aplica el método de matriz de confusión (1.3.3) para validar la generalización del clasificador CNN-SVM. En la Figura 47 se muestra la matriz de confusión realizada sobre el conjunto de validación. De la matriz se extrae la siguiente información:

- 1516 son verdaderos negativos que se predijeron correctamente como no autos.
- 4211 son verdaderos positivos que se predijeron correctamente como autos.
- 262 son falsos negativos que se predijeron como autos, pero no lo eran.
- 392 son falsos positivos que se predijeron como no autos, pero si lo eran.

Tabla 20. Fase Cuatro: Análisis de la Matriz de Confusión.

Propiedad Matriz de Confusión	Coficiente
AC	0,898
P	0,102
TPR	0,941
FPR	0,204
TNR	0,796
FNR	0,059
MCC	0,737

En la Tabla 20 se muestra el análisis de la matriz de confusión. Para el modelo de la fase uno su exactitud es del 89,8%, su precisión es de 94,1% y el coeficiente Matthews es de 0,737. De esta fase se concluye que: el modelo es menos exacto para la clasificación comparado con las fases anteriores.

2.3.5. Fase Cinco

Tabla 21. Fase Cinco: Modelo ConvNet.

	K		S		Borde	g	Tensor	Memoria		Pesos	
	h	w	h	w							
INPUT							[112x112x3]	112*112*3=	150		
CONV	25	25	3	3	same	1	[38x38x64]	38*38*64=	92416	(3*3*64)*64=	36864
RELU							[38x38x64]				
LRN							[38x38x64]				
MAXPOOL	3	3	2	2	valid		[18x18x64]	18*18*64=	20736		
CONV	3	3	1	1	same	2	[18x18x128]	18*18*128=	41472	(3*3*128)*128=	147456
RELU							[18x18x128]				
MAXPOOL	3	3	2	2	valid		[8x8x128]	8*8*128=	8192		
CONV	3	3	1	1	same	1	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
CONV	3	3	1	1	same	2	[8x8x256]	8*8*256=	16384	(3*3*256)*256=	589824
MAXPOOL	3	3	2	2	valid		[3x3x256]	3*3*256=	2304	(3*3*256)*256=	589824
FC	256	256	1				[1x1x2304]	1*1*2304=	2304	(3*3*2304)*2304=	47775744
FC	2304	1	1	1			[1x1x256]	1*1*256=	256	(3*3*256)*256=	589824
FC	256	1	1	1			[1x1x2]	1*1*2=	2	(3*3*2)*2=	36
Tamaño total de memoria necesaria para procesar una imagen de tamaño 112x112x3 en el modelo anterior									200600	Bites	
									25,075	KB	
Total de pesos a entrenar en el modelo anterior									50319396		

En la fase dos se realiza una modificación en la arquitectura de la ConvNet. Los cambios con respecto a Fase Cuatro son: el tamaño del kernel en la primera convolución y la eliminación de la última capa convolucional. Esta modificación se observa en la Tabla 21:

Sub-Fase Cinco: Entrenamiento y Validación Modelo CNN

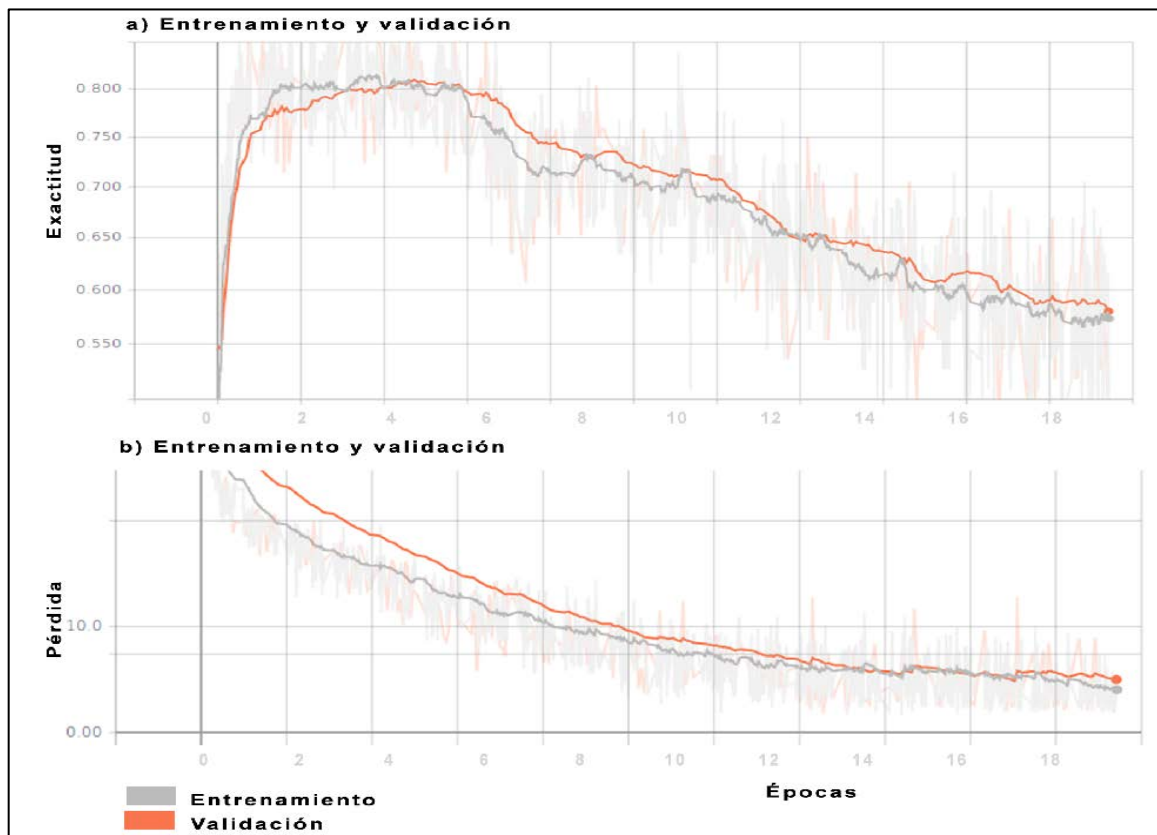


Figura 48. Fase Cinco: Exactitud y pérdida.

En la Figura 48 muestra el proceso de entrenamiento y validación de la ConvNet. El resultado para la última capa analizada, tanto para el entrenamiento como para la validación se muestra en la Tabla 22.

Tabla 22. Fase Cinco: Valores presentes en la última época analizada en CNN.

	Entrenamiento	Validación
Exactitud	0,5740	0,5774
Pérdida	1,6335	2,3880

Para la exactitud, la diferencia entre el entrenamiento y la validación para la última época es 0,003. Lo que indica que el conjunto de datos es adecuado para generar un buen modelo. Para la función de pérdida se verifica que tanto para el entrenamiento como para la validación su valor es de 1,633 y 2,388 respectivamente. Esto indica que la red tiene pérdida en su aprendizaje debido a que tiene errores en la clasificación de elementos. Sin embargo, son valores superiores a los obtenidos que en fases anteriores lo que significa la ConvNet necesita más épocas de entrenamiento para bajar la función de pérdida.

Sub-Fase Cinco: Entrenamiento y Validación Modelo SVM

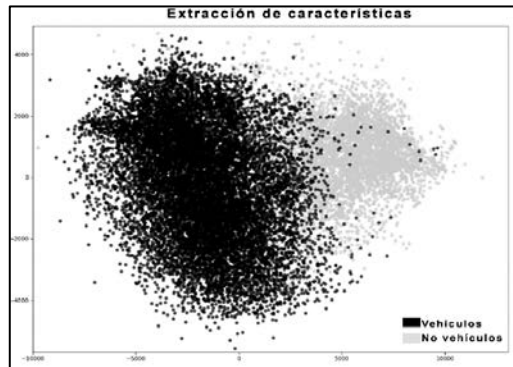


Figura 49. Fase Cinco: Representación de la extracción de características.

En la siguiente sub-fase se parte del vector de características extraído de cada imagen por el proceso de la CNN. Gráficamente la extracción de características se puede observar en la Figura 49. Partiendo de esta información se realiza el proceso de entrenamiento y validación.

Tabla 23. Fase Cinco: Entrenamiento y validación SVM.

	kernel lineal	kernel RBF
Hiperparámetros	C=0,1	C=0,1
	Gamma = 0,1	Gamma = 1/19335
Resultados (Exactitud)	Entrenamiento = 0,91709	Entrenamiento = 0,967423
	Validación = 0,919761	Validación = 0,722927

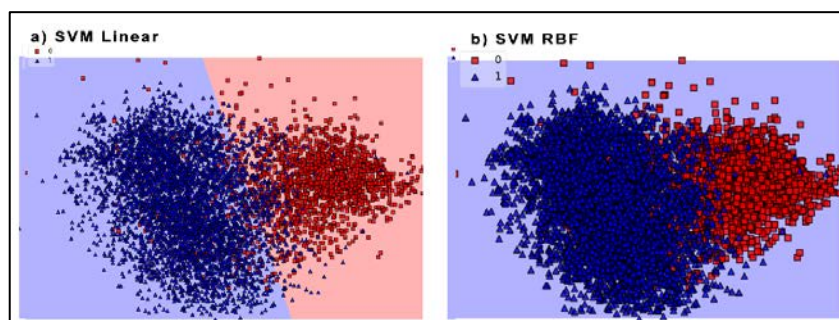


Figura 50. Fase Cinco: Comparación de clasificadores SVM

Los resultados sobre el entrenamiento y validación del SVM se muestran en la Tabla 23. Para la exactitud del modelo con kernel lineal, la diferencia entre el entrenamiento y la validación presenta un valor de 0,002. Lo que indica que el conjunto de datos es adecuado para generar un buen modelo. Para el caso de kernel RBF el análisis es similar pero no

genera un buen modelo. En la Figura 50 se observa el resultado de la generación del vector de soporte que clasificará las muestras para cada uno de los casos analizados.

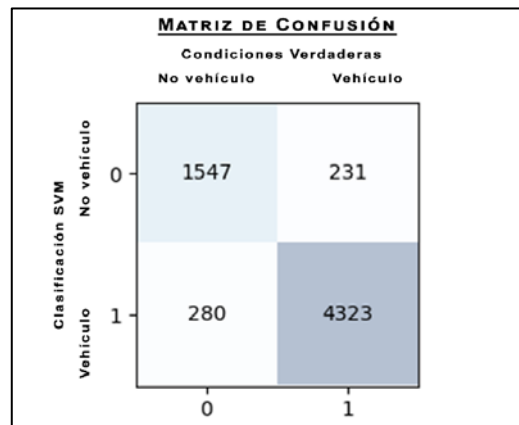


Figura 51. Fase Cinco: Matriz de confusión realizada con SVM kernel lineal.

Al modelo generado por el SVM con kernel lineal, se aplica el método de matriz de confusión (1.3.3) para validar la generalización del clasificador CNN-SVM. En la Figura 51 se muestra la matriz de confusión realizada sobre el conjunto de validación. De la matriz se extrae la siguiente información:

- 1547 son verdaderos negativos que se predijeron correctamente como no autos.
- 4323 son verdaderos positivos que se predijeron correctamente como autos.
- 231 son falsos negativos que se predijeron como autos, pero no lo eran.
- 280 son falsos positivos que se predijeron como no autos, pero si lo eran.

Tabla 24. Fase Cinco: Análisis de la Matriz de Confusión.

Propiedad Matriz de Confusión	Coficiente
AC	0,920
ERR	0,080
TPR	0,949
FPR	0,153
TNR	0,847
FNR	0,051
MCC	0,796

En la Tabla 24 se muestra el análisis de la matriz de confusión. Para el modelo de la fase uno su exactitud es del 92%, su precisión es de 94,9% y el coeficiente Matthews es de 0,796. De esta fase se concluye que: el modelo de clasificación es muy bueno.

2.3.6. Resumen del Entrenamiento, Validación y Testeo

En la Tabla 25 se presenta un resumen de las fases de entrenamiento y validación realizadas:

Tabla 25. Resumen de las fases de Entrenamiento y Validación.

Fase	CNN - Exactitud		SVM – Exactitud		Exactitud CNN-SVM	Precisión CNN-SVM	Coeficiente de correlación de Matthews	Sobreajuste (-) Bajo ajuste (+)	Cantidad de pesos a entrenar	Memoria por imagen
	Entrenamiento	Validación	Entrenamiento	Validación						
Uno	0,5113	0,4857	0,8989	0,9037	0,887	0,916	0,725	0,0021	50,32 M	26.83 KB
Dos	0,5015	0,5143	0,8883	0,8932	0,893	0,941	0,724	0,0049	50,90 M	27.13 KB
Tres	0,4910	0,5256	0,8987	0,8975	0,900	0,944	0,742	-0,0012	50,91 M	27.12 KB
Cuatro	0,5477	0,5440	0,8987	0,8975	0,898	0,941	0,737	-0,0012	50,91 M	27.13 KB
Cinco	0,5740	0,5774	0,9171	0,9197	0,920	0,949	0,796	0,0026	50,32 M	25.08 KB

Para la selección del mejor modelo se toma en consideración lo siguiente:

- El modelo debe consumir la menor cantidad de memoria en procesamiento gráfico ya sea en el CPU o GPU debido a que debe ser integrado con el sistema de visión artificial.
- El modelo debe tener un coeficiente de Matthews muy cercano a uno.
- El modelo debe tener presentar la mejor generalización posible.
- El modelo debe tener la menor cantidad de parámetros a entrenar.

Bajo las restricciones expuestas, se opta por el modelo de la Fase Cinco. La razón es que cumple con la mayor cantidad de requisitos siendo así el mejor modelo que tiene la menor cantidad de: pesos a entrenar y cantidad de memoria.

Testeo del modelo

Una vez seleccionado el modelo se le realiza un testeo del modelo con otro conjunto de datos que no se usó en el entrenamiento y validación. El nuevo conjunto de datos contiene 815, de las cuales 618 imágenes catalogadas como autos y 197 imágenes catalogadas como no autos.

Luego de aplicar el modelo se presenta gráficamente la extracción de características que se puede observar en la Figura 52. Partiendo de esta información se aplica el modelo SVM previamente entrenado.

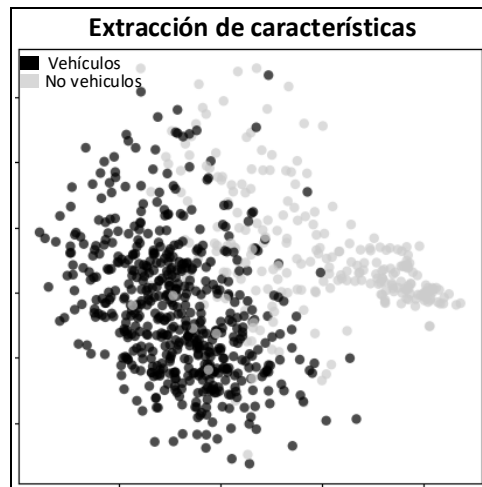


Figura 52. Test: Representación de la extracción de características.

El resultado del entrenamiento del conjunto de test se observa en la siguiente matriz de confusión en la Figura 53.

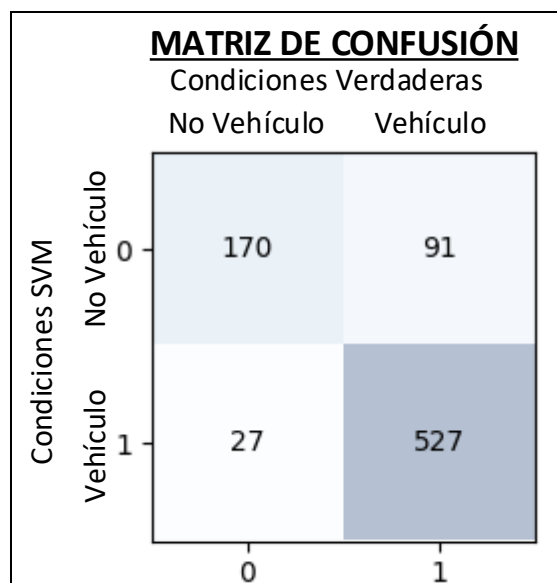


Figura 53. Test: Matriz de confusión aplicando el modelo entrenado.

En la matriz de confusión se obtiene la siguiente información:

- 170 son verdaderos negativos que se predijeron correctamente como no autos.
- 527 son verdaderos positivos que se predijeron correctamente como autos.
- 91 son falsos negativos que se predijeron como autos, pero no lo eran.
- 27 son falsos positivos que se predijeron como no autos, pero si lo eran.

Tabla 26. Fase Uno: Análisis de la Matriz de Confusión.

Propiedad Matriz de Confusión	Coficiente
AC	0,855
ERR	0,145
TPR	0,853
FPR	0,137
TNR	0,863
FNR	0,147
MCC	0,716

En la Tabla 26 se muestra el análisis de la matriz de confusión. Aplicado al conjunto de test, su exactitud es de 85,5%, su precisión es de 85,3% y el coeficiente Matthews (MCC) es de 0,716. De esto se concluye que el modelo entrenado es aceptable y en comparación con los resultados obtenidos en la validación los valores del MCC se reduce en 0,08%. Esto significa que el modelo a pesar de que se encuentra bien generalizado, para las nuevas muestras su capacidad de clasificación se reduce en 0,08%. Debido a que la diferencia no es muy grande se considera que el modelo se encuentra correctamente generalizado.

2.4. Caso de Estudio

La aplicación del clasificador al caso de estudio consiste en realizar un sistema de visión artificial que pueda detectar y clasificar vehículos que se encuentran estacionados en lugares no permitidos. El lugar exacto donde se aplica es en la Calle Ulpiano Páez, frente al SRI que es un lugar con mayor demanda de lugares de estacionamientos e infractores según la AMT (Anexo).

El proceso de análisis que se realiza sobre el lugar es el siguiente:

- a) Situar la cámara de video sobre la calle en un lugar donde pueda capturar todo el flujo vehicular.

- b) Del video obtenido se realizará un estudio de la zona de flujo y la zona donde comúnmente los autos cometen infracciones.
- c) Se crea el sector de escaneo de autos, el sector de infracción y los sectores límites. La definición de sectores y cálculos se encuentran en el Anexo II.

Para el sector de escaneo de autos se realiza un barrido con recortes de cada fotograma. Todos los recortes se procesan en paralelo en el modelo CNN-SVM y generan como resultado las coordenadas del recorte de una ventana donde existe una gran posibilidad de la presencia de un auto. Este recorte se enlaza a un método de rastreo, el motivo es usar la movilidad del auto para determinar cuando está ingresando estacionado por más de 5 segundos en la zona definida como no permitida. Además, tomar una captura de la infracción que será usada como parte del historial de incidentes que se generan de este tipo.

Se realiza un video de la calle propuesta por la AMT donde ocurren infracciones de forma regular y se toma una muestra del flujo vehicular de esta calle. Para determinar la zona de interés, se realiza un conjunto de pruebas mediante la observación. Estas pruebas cubren temas relacionados con:

- a) Determinar las diferentes zonas que existen en el video.
- b) Determinar el tamaño de salto entre recortes.
- c) Determinar el porcentaje de crecimiento de la ventana de barrido.
- d) Determinar el tiempo de escaneo entre fotogramas.
- e) Calibrar el rastreador.
- f) Calibrar las capturas de infracciones en el sistema de visión artificial.

Todas estas pruebas sirven para calibrar el sistema de visión artificial. Luego de su ejecución, los históricos que son generados de la detección y rastreo de cada vehículo se determina que el método es aplicable funciona, de forma que: clasifica, detecta y rastrea automóviles estacionados en lugares no permitidos.

2.4.1. Implementación del Sistema de Visión Artificial

La implementación del sistema de visión artificial toma en cuenta las herramientas definidas en la sección 1.4 por lo tanto principalmente se realiza bajo el Lenguaje Python en su versión 3.6. En el siguiente esquema se generaliza el proceso y las herramientas usadas:

1. Extracción de recortes de un fotograma.
 - a. **Herramientas:** skimagem scipy.misc, NumPy.

2. Análisis del set de recortes con el modelo CNN-SVM y generación del recuadro de localización de automóvil.
 - a. **Herramientas:** tensorflow, NumPy, sklearn.
3. Rastreo del auto, detección de infracciones y finalización de rastreo.
 - a. **Herramientas:** tensorflow, sklearn, NumPy, scipy.misc, cv2, dlib.

Debido a que este proceso se debe realizar en tiempo real los pasos del esquema anterior se deberán ejecutar indefinidamente. A continuación, se presenta un diagrama de flujo que detalla todas las actividades del sistema de visión artificial.

Diagrama de Flujo del Sistema de Visión Artificial

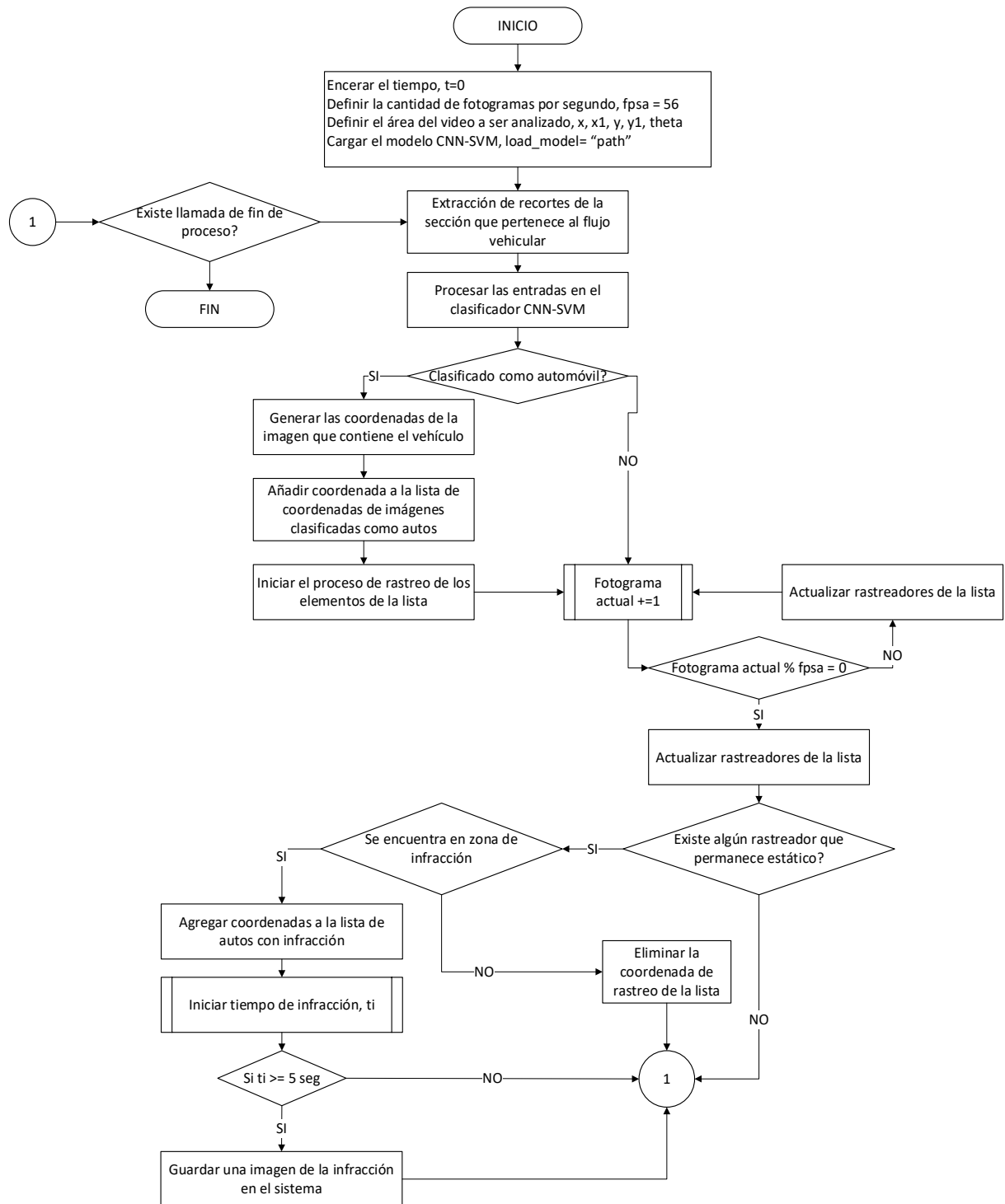


Figura 54. Diagrama de flujo del programa.

En la Figura 54 se presenta el diagrama de flujo del sistema de visión artificial. En éste se muestran todas las acciones que se realiza para resolver el problema de la detección de vehículos estacionados en lugares no permitidos. A continuación, se detalla las acciones importantes dentro del flujo.

Extracción de recortes de un fotograma

Partiendo del hecho de que un video es una secuencia de imágenes (fotogramas), cada una de ellas tiene información que será analizada. Para la extracción de recortes en cada fotograma se realizó lo siguiente:



Figura 55. Fotograma que muestra la perspectiva del video.

1. Se determina el área que va a ser escaneada. Se debe considerar que el video presenta una perspectiva en la cual se mira que los autos aparecen un tiempo t_i pequeños y luego en un tiempo t_n el auto se acerca a la cámara y crece. Por lo que la zona de estudio debe presentar perspectiva tal como se muestra en la Figura 55.
2. Se determina que la extensión máxima de análisis sea de un 30% desde la zona que aparecen los autos.
3. Se determina que el tiempo de captura de cada fotograma sea cada segundo. Un video HD que tiene un procesamiento de 56fps. La razón principal para realizar solo un análisis por segundo es debido a la limitación de hardware con el que se está trabajando (1.4.1).



Figura 56. Barrido de recortes en fotograma.

4. Para el barrido de los recortes que se realizan por fotograma se determina que las separaciones de escaneo sean de 25% de cada imagen anterior tanto en horizontal como en vertical como se muestra en la Figura 56.
5. Todas las capturas se almacenan y se estandarizan en un tensor de tamaño $B \times 112 \times 112 \times 3$ donde B es el número de recortes realizados.

Análisis del set de recortes con el modelo CNN-SVM y generación del recuadro de localización de automóvil

Luego de que se realizan los recortes de un fotograma, el tensor que se obtiene se convierte en la entrada al modelo CNN-SVM, el esquema que se sigue es el siguiente:

1. El tensor ingresa al modelo para su análisis. La salida del modelo es la clasificación de cada entrada como vehículo o no-vehículo. En caso de que sea vehículo guarda las coordenadas del recuadro. Caso contrario no se realiza ninguna acción.
2. Con las coordenadas favorables se aplica el algoritmo NMS (Learning non-maximum suppression), es un algoritmo usado después del procesamiento y es responsable de mezclar todas las detecciones pertenecientes a un objeto [162].
3. Como salida de este proceso se tiene un vector con las coordenadas donde se detectó el vehículo.

Rastreo del auto, detección de infracciones y finalización de rastreo.

Para el rastreo del auto se toma como entrada la o las coordenadas en las cuales se detectó un vehículo. Para las acciones de rastreo, detección de infracciones y finalización de rastreo se tiene el siguiente esquema:

1. Las coordenadas generadas por el NMS tienen información inicial del auto. En caso de que éste se mueva los algoritmos de rastreo se encargarán de seguirle el rastro.
2. Si el auto se mantiene por la zona de flujo. Éste al llegar a la parte inferior del marco del video se eliminará para liberar memoria.
3. Si el auto permanece en la zona de generación de recortes. Se realizará una verificación sobre el estado de los dos recortes y si no difiere mucho una de las dos coordenadas serán eliminadas.
4. En caso de que el auto ingrese a zona de infracción se iniciará un conteo de 5 segundos. En caso de que las coordenadas que representan al auto no se muevan por ese tiempo, se realiza una captura de que verifica la infracción. Esta es almacenada con fecha y hora del sistema.

2.4.2. Ejecución y Resultados Aplicados al Caso de Estudio

Para la ejecución del proyecto se presenta la funcionalidad en Scripts separados. Esta funcionalidad se describe a continuación:

1. Script 1: Realiza preprocesamiento de imágenes (2.1).
2. Script 2: Realiza el entrenamiento y validación de la CNN (2.3).
3. Script 3: Realiza el entrenamiento y validación de SVM (2.3).
4. Script 4: Implementa el sistema de visión artificial (2.4).







Figura 57. Muestra del proceso de rastreo de sistema de visión artificial.

En la Figura 57 se muestra un proceso de detección y captura de las coordenadas de un vehículo. Como se puede observar iniciando en la primera captura, se inicia un proceso de escaneo progresivo de la región activa, luego en las capturas 15 y 16 se mira que se realiza el proceso de detección. Finalmente, para la captura 16 ya se tiene el objeto rastreado de forma correcta.



Figura 58. Proceso de captura de infracción de un vehículo.

En la Figura 58 se observa el proceso de captura del sistema de visión artificial. En la primer, segunda y tercera captura se observa que un automóvil de color amarillo está ingresando a la zona de no estacionarse. En la cuarta captura se inicia el proceso de tiempo de espera para la captura de una fotografía que evidencie su infracción.



Figura 59. Ejemplo de proceso de Captura de infracción.

La detección de infracciones de autos estacionados en lugares no permitidos se puede observar en la Figura 59 la cual muestra en a) que el auto ingresó a la zona de infracción. En b) se observa el resultado de la captura de la infracción.

Evaluación del Sistema de visión artificial.

El sistema de visión artificial está formado por una parte que clasifica una determinada región de la entrada mediante el clasificador CNN-SVM y a partir de ello proceso de rastreo se encarga de perseguir las coordenadas en las cuales aparece un auto hasta la zona de infracción. El clasificador CNN-SVM fue evaluado con el Coeficiente de Correlación de Mathews dando como resultado un 79,6% con un margen de error de 0.08%. En lo relacionado con el rastreador se usó "ACCURATE SCALE ESTIMATION FOR ROBUST VISUAL TRACKING" que permite realizar el rastreo de los objetos a pesar de que estos cambien de tamaño [163]. La validación se la realizó mediante la verificación de cuantos autos son rastreados correctamente. Mediante la obtención de los registros del tracking se determinó que del 100% de autos que el modelo CNN-SVM clasificó como "autos" luego del proceso de rastreo, el 10% de ellos perdió la zona de rastreo.

Como resultado en conjunto el sistema tiene un rendimiento promedio del 84.8%, lo que significa que el sistema funciona correctamente en el 84,8%.

Históricos

Un histórico es un conjunto de reportes que se realizan y se guardan en una dirección física [164]. En la ejecución del programa se almacena la siguiente información en tiempo real:

1. Registro de posición de imagen que fue clasificada como auto.
2. Registro de posición de imagen que no fue clasificada como auto
3. Registro del cambio de posición y rastreo del automóvil
4. Captura la imagen del automóvil infractor
5. Registra el total de autos que cometieron la infracción de estacionarse en lugares no permitidos.

En el sistema los registros se presentan se guardan como una lista de posiciones en la imagen y tienen la siguiente estructura:

Objetivo encontrado: sáb 10 mar 2018 11:12:14 -05, -> en posiciones [(757, 222, 0.91017429584905363, 81, 81)]

3. CONCLUSIONES

Con la ejecución del presente proyecto se concluye lo siguiente:

- La obtención de un conjunto de imágenes con gran variedad de objetos de la misma clase que será analizada es fundamental para el entrenamiento y validación del modelo que se produce con la red neuronal. Esta gran variedad permite que el modelo pueda ser altamente generalizado. En este sentido, un set de datos que represente una buena muestra y cubra el mayor conjunto de posibilidades que se puedan presentar, es beneficioso para cualquier tipo de modelo que se desee entrenar. Sin embargo, existe la posibilidad de que no se pueda obtener un set de datos que cubra todas las posibilidades por lo que es necesario recurrir a técnicas de *data augmentation*.
- El diseño de un modelo CNN-SVM que usa redes neuronales convolucionales para la extracción de características y máquinas de soporte vectorial para la clasificación y detección de vehículos, conlleva a lineamientos estructurales específicos que se basan principalmente en el tamaño del filtro que se toma para la extracción de características como se puede mirar en 2.2. Además, cabe recalcar que los modelos CNN dependen estrictamente de las restricciones físicas como la cantidad de memoria RAM para procesamiento en CPU y en el caso de procesamiento por GPU la cantidad de VRAM disponible. De forma similar el algoritmo SVM al ser no paralelizadle su tiempo de entrenamiento dependerá de los datos de entrenamiento según la sección 1.3.
- En la sección 2.3, en el entrenamiento, validación y pruebas del modelo, se obtuvo que el sistema de clasificación tiene un Coeficiente de Correlación de Matthews de 71,6%, lo que indica que el modelo es adecuado para ser usado como clasificador.
- La implementación del modelo CNN-SVM en un sistema de visión artificial se aplicó para la detección y rastreo de automóviles que cometen infracciones de tránsito del tipo autos estacionados en lugares no permitidos. Muestra que el sistema funciona correctamente en un 84,8%. Esto permite que se pueda realizar el control de las infracciones de forma autónoma.
- La generación de históricos de la sección 2.4.2 creó evidencia de las infracciones realizadas que es guardada en forma de fotografías con hora exacta en la que sucedió la infracción.

4. RECOMENDACIONES

Las recomendaciones sugeridas por el autor del presente proyecto son las siguientes:

- Una gran cantidad de imágenes sobre una misma clase puede ser una tarea que demande de mucho tiempo y en ocasiones no es posible, para ello es necesario realizar data aumentación para agregar variabilidad del set de datos de entrada y así lograr un modelo generalizado para la clasificación de las entradas.
- La técnica mostrada CNN-SVM puede ser aplicada en un video con más fotogramas por segundo para el análisis de vehículos que se trasladan a mayor velocidad. Para este caso se deberá considerar un hardware con mejores prestaciones que las usadas en el presente proyecto.
- Para la obtención del vector de características se puede usar cualquier modelo de ConvNets por ejemplo AlexNet o Inception, recordando que se debe extraer la información desde la primera capa totalmente conectada, los resultados que se obtendrán serán similares a los presentados en la sección 2.3. Cabe recalcar que el limitante de la capacidad de hardware puede ser calculado como se muestra en las secciones 1.2.2 y 2.2; con ello determinar un modelo acorde a la capacidad de hardware que se posee.
- El presente sistema no fue evaluado en condiciones de lluvia, neblina y noche. Sin embargo, se intentó simular estas condiciones con *data augmentation*. Sin embargo, si las condiciones de visión son muy desfavorables el sistema puede no funcionar correctamente.

5. BIBLIOGRAFÍA

- [1] R. Bellman, *Artificial Intelligence: Can Computers Think?*, Thomson Course Technology, 1978.
- [2] E. Guresen y G. Kayakutlu, «Definition of artificial neural networks with comparison to other networks,» *Procedia Computer Science*, vol. 3, pp. 426-433, 2011.
- [3] V. Vapnik, I. Guyon y T. Hastie, «Support vector machines,» *Mach. Learn.*, vol. 20, pp. 273-297, 1995.
- [4] A. Duardo-Sánchez, «Herramientas informáticas y de inteligencia artificial para el meta-análisis en la frontera entre la bioinformática y las ciencias jurídicas,» *Universidad Coruña*, 2016.
- [5] E. Rich y K. Knight, «Artificial intelligence,» *McGraw-Hill, New*, 1991.
- [6] J. T. Townsend, «Theoretical analysis of an alphabetic confusion matrix,» *Perception & Psychophysics*, vol. 9, pp. 40-50, 1971.
- [7] G. Jurman, S. Riccadonna y C. Furlanello, «A comparison of MCC and CEN error measures in multi-class prediction,» *PloS one*, vol. 7, p. e41882, 2012.
- [8] M. Minsky y S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*, Expanded Edition, The MIT Press, 1987.
- [9] J. J. Hopfield, «Neural networks and physical systems with emergent collective computational abilities,» *Proceedings of the national academy of sciences*, vol. 79, pp. 2554-2558, 1982.
- [10] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Harvard University, 1975.
- [11] D. E. Rumelhart, J. L. McClelland y P. D. P. R. Group, *Parallel Distributed Processing, Vol. 1: Foundations*, A Bradford Book, 1987.
- [12] J. A. Anderson, *Neurocomputing: Foundations of Research (v. 1)*, Bradford Books, 1988.
- [13] D. N. N. Study, *Darpa Neural Network Study: October 1987-February 1988*, Afcea Intl Pr, 1988.
- [14] C. Gershenson, «Artificial Neural Networks for Beginners,» *CoRR*, vol. cs.NE/0308031, 2003.
- [15] A. K. Jain, J. Mao y K. M. Mohiuddin, «Artificial neural networks: A tutorial,» *Computer*, vol. 29, pp. 31-44, 1996.
- [16] «comprehensive list of activation functions in neural networks 2018,» 2018. [En línea]. Available: <https://stats.stackexchange.com/questions/115258/comprehensive-list-of-activation-functions-in-neural-networks-with-pros-cons>.
- [17] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer, 2011.
- [18] P. Ramachandran, B. Zoph y Q. V. Le, «Searching for Activation Functions,» *ArXiv e-prints*, 10 2017.
- [19] R. L. Welch, S. M. Ruffing y G. K. Venayagamoorthy, «Comparison of feedforward and feedback neural network architectures for short term wind speed prediction,» de *2009 International Joint Conference on Neural Networks*, 2009.
- [20] M. T. Hagan, H. B. Demuth, M. H. Beale y others, *Neural network design*, vol. 20, Pws Pub. Boston, 1996.
- [21] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan Coll Div, 1994.
- [22] G. James, D. Witten, T. Hastie y R. Tibshirani, *An Introduction to Statistical Learning*, Springer New York, 2013.
- [23] T. Hastie, R. Tibshirani y J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Second Edition (Springer Series in Statistics), Springer, 2009.
- [24] N. Gupta, «Artificial neural network,» *Network and Complex Systems*, vol. 3, pp. 24-28, 2013.
- [25] J. Cohen, «A power primer.,» *Psychological bulletin*, vol. 112, p. 155, 1992.
- [26] S. Sathyanarayana, «A Gentle Introduction to Backpropagation,» *ACM Digital Library*, 7 2014.
- [27] R. Rojas, «Neural Networks: A Systematic Introduction,» Springer, 1996.

- [28] J. Khan, J. S. Wei, M. Ringnér, L. H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. R. Antonescu, C. Peterson y P. S. Meltzer, «Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks,» *Nature Medicine*, vol. 7, pp. 673-679, 6 2001.
- [29] D. F. Specht, «A general regression neural network,» *{IEEE} Transactions on Neural Networks*, vol. 2, pp. 568-576, 1991.
- [30] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*, MIT Press, 2016.
- [31] R. Modarres, «Multi-criteria validation of artificial neural network rainfall-runoff modeling,» *Hydrology and Earth System Sciences*, vol. 13, pp. 411-421, 2009.
- [32] R. G. Osuna, «Validation,» *Wright State University*, 2018.
- [33] R. C. Lee, Z. Wang, M. Heo, R. Ross, I. Janssen y S. B. Heymsfield, «Total-body skeletal muscle mass: development and cross-validation of anthropometric prediction models,» *The American Journal of Clinical Nutrition*, vol. 72, pp. 796-803, 2000.
- [34] S. Douglas, C. Hundhausen y D. McKeown, *Exploring human visualization of computer algorithms*, Department of Computer and Information Science, University of Oregon, 1994.
- [35] O. Yadan, K. Adams, Y. Taigman y M. Ranzato, «Multi-GPU Training of ConvNets,» *CoRR*, vol. abs/1312.5853, 2013.
- [36] O. G. Selfridge, *Mechanisation of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory on 24th, 25th, 26th and 27th November 1958*, H.M. Stationery Office, 1959, p. 513-526.
- [37] P. Tino, L. Benuskova y A. Sperduti, «Artificial Neural Network Models,» de *Springer Handbook of Computational Intelligence*, Springer Berlin Heidelberg, 2015, pp. 455-471.
- [38] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*, Cambridge University Press, 2003.
- [39] P. Shah, S. Karamchandani, T. Nadkar, N. Gulechha, K. Koli y K. Lad, «OCR-based chassis-number recognition using artificial neural networks,» de *2009 {IEEE} International Conference on Vehicular Electronics and Safety ({ICVES})*, 2009.
- [40] J. Gllavata, R. Ewerth y B. Freisleben, «Text detection in images based on unsupervised classification of high-frequency wavelet coefficients,» de *Proceedings of the 17th International Conference on Pattern Recognition, 2004. {ICPR} 2004.*, 2004.
- [41] Kohonen, Barna y Chrisley, «Statistical pattern recognition with neural networks: benchmarking studies,» de *{IEEE} International Conference on Neural Networks*, 1988.
- [42] C. M. Bishop, *Neural Networks for Pattern Recognition (Advanced Texts in Econometrics (Paperback))*, Clarendon Press, 1996.
- [43] G. F. Hepner, «Artificial neural network classification using a minimal training set. Comparison to conventional supervised classification,» *Photogrammetric Engineering and Remote Sensing*, vol. 56, pp. 469-473, 4 1990.
- [44] S.-M. Chou, T.-S. Lee, Y. E. Shao y I.-F. Chen, «Mining the breast cancer pattern using artificial neural networks and multivariate adaptive regression splines,» *Expert Systems with Applications*, vol. 27, pp. 133-142, 7 2004.
- [45] A. Iwata, Y. Nagasaka y N. Suzumura, «Data compression of the ECG using neural network for digital Holter monitor,» *{IEEE} Engineering in Medicine and Biology Magazine*, vol. 9, pp. 53-57, 9 1990.
- [46] Z. Zainuddin y O. Pauline, «Function Approximation Using Artificial Neural Networks,» de *Proceedings of the 12th WSEAS International Conference on Applied Mathematics*, Stevens Point, Wisconsin, USA, 2007.
- [47] V. Sharma, «Artificial neural network applicability in business forecasting,» *International journal of emerging research in management & technology*, pp. 62-65, 2012.
- [48] K. Abhishek, M. P. Singh, S. Ghosh y A. Anand, «Weather Forecasting Model using Artificial Neural Network,» *Procedia Technology*, vol. 4, pp. 311-318, 2012.
- [49] Y. Fu y P. W. Anderson, «Application of statistical mechanics to NP-complete problems in combinatorial optimisation,» *Journal of Physics A: Mathematical and General*, vol. 19, p. 1605, 1986.

- [50] C. Papadimitriou y M. Yannakakis, «Optimization, Approximation, and Complexity Classes,» de *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, New York, NY, USA, 1988.
- [51] G. V. Puskorius, L. A. Feldkamp y L. I. Davis, *Trained neural network engine idle speed control system*, Google Patents, 2000.
- [52] B. Widrow, J. C. Aragon y B. M. Percival, *Cognitive memory and auto-associative neural network based search engine for computer and network located images and photographs*, Google Patents, 2008.
- [53] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard y L. D. Jackel, «Handwritten digit recognition with a back-propagation network,» de *Advances in neural information processing systems*, 1990.
- [54] Y. LeCun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE*, vol. 86, pp. 2278-2324, 1998.
- [55] W. Endres, W. Bambach y G. Flösser, «Voice spectrograms as a function of age, voice disguise, and voice imitation,» *The Journal of the Acoustical Society of America*, vol. 49, pp. 1842-1848, 1971.
- [56] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella y J. Schmidhuber, «Flexible, High Performance Convolutional Neural Networks for Image Classification,» *ASSOCIATION FOR THE ADVANCEMENT OF ARTIFICIAL INTELLIGENCE*, 2011.
- [57] K. Fukushima, «Neocognitron,» *Scholarpedia*, vol. 2, p. 1717, 2007.
- [58] Y. LeCun y others, «Generalization and network design strategies,» *Connectionism in perspective*, pp. 143-155, 1989.
- [59] L. M. Pechuán, D. A. Rosso-Pelayo y J. Brieua, «Reconocimiento de dígitos escritos a mano mediante métodos de tratamiento de imagen y modelos de clasificación.,» *Research in Computing Science*, vol. 93, pp. 83-94, 2015.
- [60] S. S. Farfade, M. J. Saberian y L.-J. Li, «Multi-view face detection using deep convolutional neural networks,» de *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, 2015.
- [61] J. Kim, «kfw0612/awesome-deep-vision,» 2018. [En línea]. Available: <https://github.com/kfw0612/awesome-deep-vision>.
- [62] J. Johnson, «cnn-benchmarks,» 10 01 2018. [En línea]. Available: <https://github.com/jcjohnson/cnn-benchmarks>. [Último acceso: 10 01 2018].
- [63] T. Dettmers, «Hardware Archives - Tim Dettmers,» 2018. [En línea]. Available: <http://timdettmers.com/category/hardware/>.
- [64] P. Krill, «For developers, the focus is deep learning, multiplatform, and coding skills,» 2018. [En línea]. Available: <https://www.infoworld.com/article/3253308/application-development/for-developers-the-focus-is-deep-learning-multiplatform-and-coding-skills.html>.
- [65] Silicon-Software, «ITE 2017 in Japan with Focus on Deep Learning / CNN,» 2018. [En línea]. Available: <https://silicon.software/blog-ite-2017/>.
- [66] B. E. Boser, E. Sackinger, J. Bromley, Y. L. Cun y L. D. Jackel, «An analog neural network processor with programmable topology,» *(IEEE) Journal of Solid-State Circuits*, vol. 26, pp. 2017-2025, 1991.
- [67] C. U. Press, *Scaling up Machine Learning: Parallel and Distributed Approaches*, C. U. Press, Ed., Cambridge University Press, 2011.
- [68] Alex Krizhevsky y G. Hinton, «CIFAR-10 and CIFAR-100,» 2018. [En línea]. Available: <https://www.cs.toronto.edu/kriz/cifar.html>.
- [69] T. G. a. A. G. a. J. v. d. W. a. J.-M. Geusebroek, «Color in Computer Vision: Fundamentals and Applications,» Wiley, 2012.
- [70] S. J. a. N. P. a. C. J. F. a. M. J. M. a. E. D. D. Russell, «Artificial intelligence: a modern approach,» Prentice hall Upper Saddle River, 2003.
- [71] A. Krizhevsky y G. Hinton, «Learning multiple layers of features from tiny images,» *CIFAR*, 2009.
- [72] Kaggle, «Kaggle Datasets,» 2018. [En línea]. Available: <https://www.kaggle.com/datasets>.
- [73] Y. Labs, «Data sets YahooLabs,» 2018. [En línea]. Available: <https://webscope.sandbox.yahoo.com>.

- [74] Stanford, «Large Network dataset collection,» 2018. [En línea]. Available: <https://snap.stanford.edu/data>.
- [75] A. WebServices, «AWS Datasets,» 2018. [En línea]. Available: https://aws.amazon.com/es/datasets/?_encoding=UTF8&jiveRedirect=1.
- [76] U. Texas, «Computer vision image datasets,» 2018. [En línea]. Available: <http://www.cs.utexas.edu/grauman/courses/spring2008/datasets.htm>.
- [77] T. Gale, S. Eliuk y C. Upright, «High-Performance Data Loading and Augmentation for Deep Neural Network Training,» 2017.
- [78] B. D. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, 2008.
- [79] C. G. Cambroner y I. G. Moreno, «Algoritmos de aprendizaje: knn & kmeans,» *Univeridad Carlos III Madrid*, 2006.
- [80] I. V. Tetko, D. J. Livingstone y A. I. Luik, «Neural network studies. 1. Comparison of overfitting and overtraining,» *Journal of chemical information and computer sciences*, vol. 35, pp. 826-833, 1995.
- [81] R. C. Gonzalez y R. E. Woods, Digital Image Processing (2nd Edition), Prentice Hall, 2002.
- [82] K. G. Karibasappa y K. Karibasappa, «AI Based Automated Identification and Estimation of Noise in Digital Images,» de *Advances in Intelligent Informatics*, Cham, 2015.
- [83] Mathworks, «how to detect noisy data,» 2018. [En línea]. Available: <https://la.mathworks.com/matlabcentral/answers/346083-how-to-detect-noisy-data-outliers?requestedDomain=true>.
- [84] Y. LeCun, C. Cortes y C. J. C. Burges, «THE MNIST DATABASE of handwritten digits,» 2018. [En línea]. Available: <http://yann.lecun.com/exdb/mnist/>.
- [85] Github, «NMIST in diferent formats,» 2018. [En línea]. Available: <https://github.com/mrgloom/MNIST-dataset-in-different-formats>.
- [86] R. C. A. Nonis, «Intelligenza Artificiale A.A.,» *Univerisdad Stanford*, 2013.
- [87] S. W. Smith, The Scientist & Engineer's Guide to Digital Signal Processing, California Technical Pub, 1997.
- [88] J. Wu, «Introduction to convolutional neural networks,» *National Key Lab for Novel Software Technology. Nanjing University. China*, 2017.
- [89] Standford, «convolutional neural networks for visual recognition,» 2018. [En línea]. Available: <http://cs231n.github.io/convolutional-networks>.
- [90] P. S. a. D. E. a. X. Z. a. M. M. a. R. F. a. Y. LeCun, «OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks,» *DBLP:journals/corr/SermanetEZMFL* 13, 2013.
- [91] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever y R. R. Salakhutdinov, «Improving neural networks by preventing co-adaptation of feature detectors,» *ArXiv e-prints*, 7 2012.
- [92] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun y R. Fergus, «Regularization of neural networks using dropconnect,» de *International Conference on Machine Learning*, 2013.
- [93] A. Sharif Razavian, H. Azizpour, J. Sullivan y S. Carlsson, «CNN Features off-the-shelf: an Astounding Baseline for Recognition,» *ArXiv e-prints*, 3 2014.
- [94] A. Perunicic, *Understanding Neural Network Weight Initialization*, 2017.
- [95] M. Thoma, «Analysis and Optimization of Convolutional Neural Network Architectures,» *ArXiv e-prints*, 7 2017.
- [96] N. Qian, «On the momentum term in gradient descent learning algorithms,» *Neural Networks*, vol. 12, pp. 145-151, 1 1999.
- [97] J. a. H. E. a. S. Y. Duchi, «Adaptive subgradient methods for online learning and stochastic optimization,» *Journal of Machine Learning Research*, 2011.
- [98] DL4J, «Comparing Top Deep Learning Frameworks: Deeplearning4j, PyTorch, TensorFlow, Caffe, Keras, MxNet, Gluon & CNTK,» 1 2018. [En línea]. Available: <https://deeplearning4j.org/compare-dl4j-tensorflow-pytorch>.
- [99] T. v. Laarhoven, «L2 Regularization versus Batch and Weight Normalization,» *CoRR*, 2017.

- [100] S. J. Nowlan y G. E. Hinton, «Simplifying Neural Networks by Soft Weight-Sharing,» *Neural Computation*, vol. 4, pp. 473-493, 7 1992.
- [101] Nvidia, «Titan Graphic Card,» 1 2018. [En línea]. Available: <https://www.nvidia.com/en-us/titan/titan-xp/>.
- [102] T. Evgeniou y M. Pontil, «Support Vector Machines: Theory and Applications,» de *Machine Learning and Its Applications*, Springer Berlin Heidelberg, 2001, pp. 249-257.
- [103] S. Tong y D. Koller, «Support vector machine active learning with applications to text classification,» *Journal of machine learning research*, vol. 2, pp. 45-66, 2001.
- [104] J. Zhang, «A Complete List of Kernels Used in Support Vector Machines,» *Biochemistry {&} Pharmacology: Open Access*, vol. 04, 2015.
- [105] S. V. Stehman, «Selecting and interpreting measures of thematic classification accuracy,» *Remote Sensing of Environment*, vol. 62, pp. 77-89, 10 1997.
- [106] G. H. Rosenfield y K. Fitzpatrick-Lins, «A coefficient of agreement as a measure of thematic classification accuracy.,» *Photogrammetric engineering and remote sensing*, vol. 52, pp. 223-227, 1986.
- [107] S. Aronoff, «Classification accuracy: a user approach,» *Science Direct*, 1982.
- [108] A. HAY, «The derivation of global estimates from a confusion matrix,» *International Journal of Remote Sensing*, vol. 9, pp. 1395-1398, 8 1988.
- [109] T. Fawcett, «An introduction to ROC analysis,» *Pattern Recognition Letters*, vol. 27, pp. 861-874, 6 2006.
- [110] B. W. Matthews, «Comparison of the predicted and observed secondary structure of T4 phage lysozyme,» *Biochimica et Biophysica Acta (BBA) - Protein Structure*, vol. 405, pp. 442-451, 10 1975.
- [111] Thoughtworks, «Technolohy Radar,» 7 2014. [En línea]. Available: <https://assets.thoughtworks.com/assets/technology-radar-july-2014-es.pdf>.
- [112] Thoughtworks, «Technolohy Radar,» 11 2015. [En línea]. Available: <https://assets.thoughtworks.com/assets/technology-radar-nov-2015-es.pdf>.
- [113] Thoughtworks, «Technolohy Radar,» 11 2016. [En línea]. Available: <https://assets.thoughtworks.com/assets/technology-radar-nov-2016-es.pdf>.
- [114] Thoughtworks, «Technolohy Radar,» 7 2017. [En línea]. Available: <https://assets.thoughtworks.com/assets/technology-radar-vol-17-es.pdf>.
- [115] Thoughtworks, «Frameworks,» 7 2016. [En línea]. Available: <https://www.thoughtworks.com/radar/languages-and-frameworks/python-3>.
- [116] NumPy, «NumPy,» 7 2016. [En línea]. Available: <http://www.numpy.org/>.
- [117] Landbot, «Which hardware components (CPU, RAM, GC, etc.) are needed for a machine learning/deep learning home PC/computer to run fast?,» 1 2018. [En línea]. Available: <https://www.quora.com/Which-hardware-components-CPU-RAM-GC-etc-are-needed-for-a-machine-learning-deep-learning-home-PC-computer-to-run-fast>.
- [118] T. Dettmers, *Deep Learning Hardware Limbo*, 2017.
- [119] Nvidia, «Nvidia Products,» 1 2018. [En línea]. Available: <http://www.nvidia.com/page/products.html>.
- [120] AMD, «AMD Products,» 1 2018. [En línea]. Available: <https://shop.amd.com/en-us/business/pro-graphics>.
- [121] Zotac, «Zotac Products,» 1 2018. [En línea]. Available: <https://www.zotac.com/ec/page/geforce-gtx-10-series>.
- [122] MSI, «MSI Products,» 1 2018. [En línea]. Available: <https://us.msi.com/Graphics-cards/>.
- [123] Gigabyte, «Gigabyte Products,» 1 2018. [En línea]. Available: <https://www.gigabyte.com/us/Graphics-Card>.
- [124] Gainward, «Gainward Products,» 1 2018. [En línea]. Available: <http://www.gainward.com/main/index.php?lang=en>.
- [125] Nvidia, *Nvidia titan xp graphics card with pascal architecture*, 2018.
- [126] Intel, «Intel Core i7 4720HQ,» 1 2018. [En línea]. Available: https://ark.intel.com/es/products/78934/Intel-Core-i7-4720HQ-Processor-6M-Cache-up-to-3_60-GHz.

- [127] Geforce, «Geforce GTX 960m,» 1 2018. [En línea]. Available: <https://www.geforce.com/hardware/notebook-gpus/geforce-gtx-960m>.
- [128] O. L. Biewald, *Build a super fast deep learning machine for under 1,000*, 2017.
- [129] Reddit, «Does the choice of operating system make a big difference for machine learning?,» 1 2016. [En línea]. Available: https://www.reddit.com/r/MachineLearning/comments/3938ii/does_the_choice_of_operating_system_make_a_big/.
- [130] I. *Faster machine learning is coming to the Linux kernel*, 2017.
- [131] Ubuntu, «Guía del escritorio de Ubuntu 16.04,» 2018. [En línea]. Available: <https://help.ubuntu.com/lts/ubuntu-help/index.html>.
- [132] G. B. P. KDnuggets, «50 Deep Learning Software Tools and Platforms, Updated,» 12 2016. [En línea]. Available: <https://www.kdnuggets.com/2015/12/deep-learning-tools.html>.
- [133] G. Piatetsky, «Bibliografía de Greory,» 1 2018. [En línea]. Available: <https://www.kdnuggets.com/gps.html>.
- [134] M. Mayo, *Coeficiente de Matthew*, 2018.
- [135] Tensorflow, «Tensorflow Official Web Page,» 1 2018. [En línea]. Available: <https://www.tensorflow.org/>.
- [136] Apache, «Mxnet Official Web Page,» 1 2018. [En línea]. Available: <https://mxnet.apache.org/>.
- [137] Berkeleyvision, «Caffe Official Web Page,» 1 2018. [En línea]. Available: <http://caffe.berkeleyvision.org/>.
- [138] Caffe, «Caffe Official Web Page,» 1 2018. [En línea]. Available: <https://www.csie.ntu.edu.tw/~cjlin/libsvm/>.
- [139] Microsoft, «Cognitive toolkit,» 1 2018. [En línea]. Available: <https://www.microsoft.com/en-us/cognitive-toolkit/>.
- [140] Thoughtworks, «Scikit-learn Official Web Page,» 1 2018. [En línea]. Available: <https://www.thoughtworks.com/radar/tools/scikit-learn>.
- [141] Scikit-Learn, «SVM Official Web Page,» 1 2018. [En línea]. Available: <http://scikit-learn.org/stable/>.
- [142] Apache, «Singa Official Web Page,» 1 2018. [En línea]. Available: <https://svn.apache.org/repos/infra/websites/production/singa/content/docs/overview.html>.
- [143] Amazon, «Machine Learning,» 1 2018. [En línea]. Available: <https://aws.amazon.com/es/machine-learning/>.
- [144] A. Studio, «zureml Official Web Page,» 1 2018. [En línea]. Available: <https://studio.azureml.net/>.
- [145] C.-C. Chang y C.-J. Lin, «LIBSVM -- A Library for Support Vector Machines,» 1 2018. [En línea]. Available: <http://caffe.berkeleyvision.org/>.
- [146] OpenCV, «OpenCV Official Web Page,» 1 2018. [En línea]. Available: <https://opencv.org/>.
- [147] SimpleCV, *SimpleCV Official Web Page*, 2018.
- [148] Accord, «Accord framework,» 1 2018. [En línea]. Available: <http://accord-framework.net/>.
- [149] Dlib, «dlib Official Web Page,» 1 2018. [En línea]. Available: <http://blog.dlib.net/2017/09/fast-multiclass-object-detection-in.html>.
- [150] S. University, «Car dataset,» 1 2018. [En línea]. Available: http://ai.stanford.edu/~jkrause/cars/car_dataset.html.
- [151] U. P. Madrid, «Vehicle database,» 1 2018. [En línea]. Available: https://www.gti.ssr.upm.es/data/Vehicle_database.html.
- [152] Princeton, «WordNet,» 1 2018. [En línea]. Available: <http://wordnet.princeton.edu/>.
- [153] Imagenet, «Vehicle database,» 1 2018. [En línea]. Available: <http://image-net.org>.
- [154] Google, «Google Images,» 1 2018. [En línea]. Available: <https://images.google.com/>.
- [155] K. Batch, «Fatkun Batch,» 1 2018. [En línea]. Available: <https://chrome.google.com/webstore/search/Fatkun%20Batch>.
- [156] R. Yasrab, N. Gu y X. Zhang, «An Encoder-Decoder Based Convolution Neural Network (CNN) for Future Advanced Driver Assistance System (ADAS),» *Applied Sciences*, vol. 7, p. 312, 2017.
- [157] A. F. Agarap, «An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification,» *ArXiv e-prints*, 12 2017.

- [158] A. Krizhevsky, I. Sutskever y G. E. Hinton, «Imagenet classification with deep convolutional neural networks,» de *Advances in neural information processing systems*, 2012.
- [159] L. Hertel, E. Barth, T. Käster y T. Martinetz, *Deep Convolutional Neural Networks as Generic Feature Extractors*.
- [160] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke y A. Rabinovich, «Going Deeper with Convolutions,» *CoRR*, vol. abs/1409.4842, 2014.
- [161] X.-X. Niu y C. Y. Suen, «A novel hybrid CNN–SVM classifier for recognizing handwritten digits,» *Pattern Recognition*, vol. 45, pp. 1318-1325, 4 2012.
- [162] M. H. Brown, *Algorithm Animation (ACM Distinguished Dissertation)*, The MIT Press, 1988.
- [163] M. Danllejan, G. Häger, S. F. Khan y M. Felsberg, «Accurate Scale Estimation for Robust Visual Tracking,» *En Proceedings of the IEEE International Conference on Computer Vision*. , pp. 4310-4318, 2015.
- [164] R. A. d. I. Lengua, «Históricos,» 2018. [En línea]. Available: [<http://dle.rae.es/?id=KX9HI9t>].

GLOSARIO

- **API:** En la programación, es el acrónimo de *Application Programming Interface* (API), es un conjunto de definiciones de subrutinas, protocolos y herramientas para crear aplicaciones de software.
- **Asintóticas:** En geometría, la asíntota de una curva es una línea cuya distancia entre la curva y la línea se aproxima a cero cuando una de sus coordenadas tiende a infinito.
- **Batch:** Grupo de muestras de entrenamiento
- **Big Data:** Son conjuntos de datos que son tan voluminosos y complejos que el software de aplicación de procesamiento de datos tradicional es inadecuado para manejarlos.
- **Business Analytics:** Se refiere a las habilidades, tecnologías, prácticas para la exploración iterativa continua y la investigación del desempeño empresarial pasado para obtener una visión e impulsar la planificación empresarial.
- **Campo Receptivo:** Se refiere a la región particular del espacio sensorial de una neurona.
- **CIFRAR:** Es el acrónimo de *Canadian Institute for Advance Research*. Es un instituto de investigaciones dedicado al desarrollo de la tecnología para mejorar la calidad de vida.
- **Clustering:** En español agrupamiento, es una técnica de minería de datos que consiste en la división de un set de datos en grupos de objetos similares.
- **Condiciones Karush-Kuhn-Trucker:** Se dice que son los requerimientos necesarios y suficientes para que la solución de un problema de programación matemática sea óptima.
- **CPU:** Es el acrónimo de Central Processing Unit, es un circuito electrónico dentro que lleva a cabo las instrucciones de un programa informático realizando las operaciones aritméticas, lógicas, de control y de entrada/salida.
- **Data Minig:** Es el proceso de descubrir patrones en grandes conjuntos de datos que involucran métodos en la intersección de aprendizaje automático, estadísticas y sistemas de bases de datos.
- **Data Science:** Es un campo interdisciplinario de métodos científicos, procesos, algoritmos y sistemas para extraer conocimiento o información de datos en diversas formas, ya sea estructuradas o no estructuradas, similar a la minería de datos.
- **Distribución de Bernoulli:** Es la distribución de probabilidad de una variable aleatoria que toma el valor 1 con probabilidad p y el valor 0 para $q = 1 - p$.

- **Época** : Una época describe la cantidad de veces que el algoritmo ve el conjunto de datos completo. Entonces, cada vez que el algoritmo ha visto todas las muestras en el conjunto de datos, se ha completado una época.
- **Fotograma**: Imagen cinematográfica considerada aisladamente, representa a cada una de las fotografías de un video.
- **Frameworks**: En la programación de computadoras, un marco de software es una abstracción en la cual el software que proporciona una funcionalidad genérica puede ser cambiado selectivamente por un código adicional escrito por el usuario, proporcionando así un software específico de la aplicación.
- **Función Logistic**: Es una curva sigmoidea definida por la ecuación $f(x) = \frac{L}{1+e^{k(x-x_0)}}$, donde L es el valor máximo de la curva, x el valor del punto medio del sigmoide y k es la pendiente de la curva.
- **Funciones de Activación**: En redes computacionales, la función de activación de un nodo define la salida de ese nodo dada una entrada o conjunto de entradas. Sus salidas son 1 o 0.
- **Gbps**: Es la velocidad de transferencia de datos que se transmiten en una unidad de tiempo. El caso particular de Gbps es 125 megabytes por segundo
- **Github**: Es un servicio de alojamiento web basado en el control de versiones que utiliza git.
- **GPU**: Es el acrónimo de Graphics Processing Unit, es un circuito electrónico especializado diseñado para manipular y modificar rápidamente la memoria acelerando la creación de imágenes en un búfer de cuadros destinado a la salida a un dispositivo de visualización.
- **Hiperparámetro**: En la estadística bayesiana es un parámetro previo en un problema concreto.
- **Iteración**: Describe el número de veces que un set de datos pasó a través del algoritmo. En el caso de las redes neuronales, eso significa el pase hacia adelante y el pase hacia atrás. Entonces, cada vez que pasa un set de datos a través de la red neuronal, completa una iteración.
- **Kernel**: En aprendizaje automático, el método de kernel son una clase de algoritmos para el análisis de patrones.
- **Logits**: Es la función inversa de la función sigmoidea. Cuando una variable de la función representa una probabilidad p , la función logit genera las probabilidades logarítmicas $p/(1 - p)$.

- **Machine Learning:** Es un campo de la informática que da a los sistemas informáticos la capacidad de aprender con datos. Esto es mejorar progresivamente el rendimiento en una tarea específica.
- **Overfitting:** En español sobreajuste, se dice que un modelo estadístico está sobreajustado, cuando se entrena con una gran cantidad de datos y se genera un sobreajuste en el modelo de clasificación.
- **Paradigmas de Aprendizaje:** En aprendizaje de máquina, son los modelos que pueden ser usados para entrenar una red neuronal.
- **RGB:** Representa a los canales de color rojo, verde y azul o por su nombre en inglés Red, Blue and Green.
- **Sesgo (bias):** En aprendizaje automático y estadística, el sesgo es el problema de minimizar simultáneamente dos fuentes de error que impiden que los algoritmos de aprendizaje supervisado generalicen más allá de su conjunto de entrenamiento.
- **Sigmoide:** Es una función matemática que tiene una curva característica en forma de "S" o curva sigmoidea. Se define con la siguiente fórmula. $S(x) = \frac{1}{1+e^{-x}}$.
- **Tensor:** Un tensor es un objeto matemático que generaliza escalares y vectores n-dimensionales. Por ejemplo, un tensor de rango cero es un escalar, un tensor de rango uno es un vector, un tensor de rango dos es una matriz o un vector de dos dimensiones y de la misma forma para las dimensiones siguientes.
- **Underfitting:** En español subajuste, se dice que un modelo estadístico o un algoritmo de aprendizaje automático tiene un ajuste insuficiente cuando no puede capturar la tendencia subyacente de los datos.
- **Varianza:** La varianza es la cantidad que la estimación de la función objetivo cambiará si se utilizaron diferentes datos de entrenamiento.
- **VRAM:** Es el acrónimo de *Video Random Access Memory*, es un tipo de memoria RAM que utiliza el controlador gráfico para poder manejar toda la información visual que le manda la CPU del sistema.

ANEXOS

Anexo I

Oficio N° 0129 – AMT – CTO -2017
DM Quito, 05 de Julio del 2017

Señor
Madai Carlos Arteaga
Presente.-

Asunto: En respuesta a Oficio S/N del 20 de Junio del 2017.
Zonas Azules mayor demanda, conflictos y evasores

De mi consideración:

En respuesta Oficio S/N del 20 de Junio del 2017, por el cual solicitan las zonas azules de mayor demanda, conflictos y mayor número de evasores, me permito indicar lo siguiente.

La EPMOP cuenta con 4 zonas de estacionamiento permitido, en el estacionamiento de ZONA AZUL, que está distribuido los sectores del CENTRO-NORTE de la ciudad, estas zonas son:

1. Mariscal
2. La Pradera
3. Carolina
4. González Suarez

También cuentan con edificios de estacionamiento ubicados en el centro de la ciudad, estos son:

1. Cadisan
2. El Tejar
3. Montufar 1
4. Montufar 2
5. Yaku

Amazonas 133-229 e Inglaterra PBX: 395 2300 denuncias.amt@quito.gob.ec

AGENCIA METROPOLITANA DE TRÁNSITO

Además de lo mencionado también cuenta con plazas de estacionamiento en el parque de la Carolina, el antiguo aeropuerto (Parque Bicentenario), Terminal Quitumbe y Terminal Carcelén.

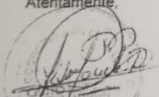
De estos sitios de estacionamiento de zona azul, el área de mayor demanda y de evasores se encuentra en la zona azul de la Mariscal



Según lo analizado en las áreas de mayor demanda y de evasores que encuentran ubicadas en los siguientes sitios, se recomienda en las mismas 19la implantación de su proyecto:

- Zona Azul Mariscal:
Calle Upiano Páez (frente al SRI)
- Zona Azul Carolina:
Calle Alemania (Frente a oficinas de la EPMAPS)

Particular que pongo en su conocimiento para los fines pertinentes.

Atentamente,


Arq. Wilson Paredes
COORDINADOR TÉCNICO DE OPERACIONES (E)
AGENCIA METROPOLITANA DE TRÁNSITO

ACCIÓN	RESPONSABLE	SIGLA UNIDAD	FECHA	SUMILLA
Elaboración:	Dayana Cevallos	CTO	05-jul-17	
Aprobación:	Arq. Wilson Paredes	CTO	05-jul-17	

Ejemplar 1: Madai Carlos Arteaga
Ejemplar 2: Archivo Coordinación Técnica de Operaciones

Amazonas 133-229 e Inglaterra PBX: 395 2300 denuncias.amt@quito.gob.ec

Anexo II



Figura 60. Zonas del fotograma

Las diferentes zonas del video se observan en la Figura 60 y se realizan bajo las siguientes consideraciones:

Zona café: Es una zona que muestra desde que altura del video se toma se inicia el proceso de análisis. Esta zona se calcula mediante el porcentaje de la altura del video.

Zona amarilla: Es una zona que muestra la región del fotograma que no debe ser analizada. Esta zona se calcula mediante la siguiente fórmula: $x = b - i * \tan(\alpha)$, donde $b = \frac{a}{\tan(\alpha)}$, α es el ángulo de inclinación de la zona amarilla y i es la altura de la iteración analizada.

Zona roja: Es una zona que muestra la región del fotograma que analiza los autos que comenten infracciones. Esta zona se calcula mediante la siguiente fórmula: $x = b - i * \tan(90 - \beta)$, donde $b = \frac{\text{restante}}{\tan(\beta)}$, $\beta - 90$ es el ángulo de inclinación de la zona roja y i es la altura de la iteración analizada.

Zona verde: Es una zona que muestra la región del fotograma en la que se conoce que siempre existe flujo vehicular.