ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

TEMA: “RECONOCIMIENTO DE IMÁGENES EN FRAMES DE VIDEO UTILIZANDO REDES NEURONALES”

AUTOR: AMORES HEREDIA, ANDRES ESTEBAN

DIRECTOR: ING. SILVA TAPIA, RODRIGO

SANGOLQUÍ
2017

CERTIFICACIÓN

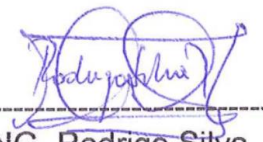
DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

CERTIFICACIÓN

Certifico que el trabajo de titulación, **"RECONOCIMIENTO DE IMÁGENES EN FRAMES DE VIDEO UTILIZANDO REDES NEURONALES"** realizada por el señor **ANDRÉS ESTEBAN AMORES HEREDIA**, ha sido revisado en su totalidad y analizado por el software anti plagio, el mismo cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de las Fuerzas Armadas – ESPE, por lo tanto me permito acreditarlo y autorizar al señor **ANDRÉS ESTEBAN AMORES HEREDIA** para que lo sustente públicamente.

Sangolquí, 17 de agosto del 2017



ING. Rodrigo Silva.
DIRECTOR

AUTORÍA DE RESPONSABILIDAD

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

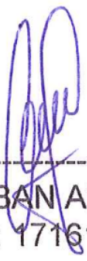
CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORÍA DE RESPONSABILIDAD

Yo, **ANDRÉS ESTEBAN AMORES HEREDIA**, con cédula de identidad N° 1716175888 declaro que este trabajo de titulación **"RECONOCIMIENTO DE IMÁGENES EN FRAMES DE VIDEO UTILIZANDO REDES NEURONALES"** ha sido desarrollada considerando los métodos de investigación existentes, así como también se ha respetado los derechos intelectuales de terceros considerándose en las citas bibliográficas.

Consecuentemente declaro que es trabajo de mi autoría, en virtud de ello me declaro responsable del contenido, veracidad y alcance de la investigación mencionada.

Sangolquí, 17 de agosto del 2017



ANDRÉS ESTEBAN AMORES HEREDIA
C.C.: 1716175888

AUTORIZACIÓN

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORIZACIÓN

Yo, **ANDRÉS ESTEBAN AMORES HEREDIA,** autorizo a la Universidad de las Fuerzas Armadas – ESPE publicar en la biblioteca virtual de la institución el presente trabajo de titulación **"RECONOCIMIENTO DE IMÁGENES EN FRAMES DE VIDEO UTILIZANDO REDES NEURONALES"** cuyo contenido, ideas y criterios son de mi autoría y responsabilidad.

Sangolquí, 17 de agosto del 2017

ANDRÉS ESTEBAN AMORES HEREDIA
C.C.: 1716175888

CLÁUSULA DE LICENCIA DE USO DE PUBLICACIÓN DE TESIS

Yo Andrés Esteban Amores Heredia, autor de la tesis intitulada RECONOCIMIENTO DE IMÁGENES EN FRAMES DE VIDEO UTILIZANDO REDES NEURONALES, mediante el presente documento dejo constancia de que la obra es de mi exclusiva autoría y producción, que la he elaborado para cumplir con uno de los requisitos previos para la obtención del título de INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES en la Universidad de las Fuerzas Armadas – ESPE.

- Licencio gratuitamente a la Universidad de las Fuerzas Armadas – ESPE, los derechos de reproducción, comunicación pública, distribución y divulgación, durante 48 meses a partir de mi graduación, pudiendo por lo tanto la Universidad, utilizar y usar esta obra para cualquier medio conocido o por conocer, siempre y cuando no se lo haga para obtener beneficio económico. Esta autorización incluye la reproducción total o parcial en los formatos virtual, electrónico, digital, óptico, como usos en red local y en internet.

- Declaro que, en caso de presentarse cualquier reclamación de parte de terceros respecto de los derechos de autor de la obra antes referida, yo asumiré toda la responsabilidad frente a terceros y a la Universidad.

- En esta fecha entrego a la Secretaria General, el ejemplar respectivo y sus anexos en formato impreso y digital o electrónico.

Quito, 17 de agosto del 2017

ANDRÉS AMORES
C.C.: 1716175888

DEDICATORIA

A toda mi familia, por haberme apoyado incondicionalmente durante toda mi carrera, gracias por no dejarme caer en este duro camino, sin ustedes no sería nada. A mi madre por enseñarme que todo es posible en esta vida que con esfuerzo y dedicación las cosas llegan de manera fácil. A mi padre por enseñarme a siempre ser humilde y que en la vida siempre triunfan las personas perseverantes y honradas. A mis hermanos por darme todo el apoyo que necesité durante todo este largo trayecto. A Dios por darme una familia maravillosa y por ayudarme en toda mi vida.

Andrés Esteban Amores Heredia

AGRADECIMIENTO

A mis padres Luis y María Elena por ser mi motivación para cada día ser no solo un profesional sino una persona de bien y poder cumplir cualquier meta que me la proponga. Agradezco cada palabra de aliento que me dieron para salir adelante a pesar de los problemas que se me presentaron a lo largo de mi carrera. Gracias por ser mi ejemplo de bien. A mis hermanos Luis Alberto y Dayana por ayudarme en cada paso que daba, por palabras de motivación que me ayudaron a salir adelante. A mis amigos de la universidad que siempre estuvieron conmigo compartiendo gratos momentos durante varios años no solo en las aulas. A mi director Ing. Rodrigo Silva por su incondicional ayuda durante mi etapa final de estudiante, por su paciencia y tiempo.

Andrés Esteban Amores Heredia

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

En la actualidad existen varias aplicaciones para realizar el reconocimiento y clasificación de imágenes en los sistemas de videovigilancia y para esto se requiere procesar el video de manera rápida debido a que son videos en tiempo real. Uno de los métodos que se utiliza para realizar estas aplicaciones es el uso de redes neuronales. En el presente proyecto de investigación se realizó un clasificador de imágenes utilizando las redes neuronales convolucionales (CNN), que se utilizaron para entrenar al clasificador cuyas imágenes a ser reconocidas son una secuencia de cuadros de video obtenido a través de una cámara de videovigilancia. El aporte del proyecto es la clasificación de imágenes (personas, vehículos) desde los *frames* de video que generalmente tienen baja calidad en definición, iluminación, distancia de objetos, etc., y la implementación del clasificador con la red neuronal que requiere una buena capacidad computacional para realizar millones de operaciones simultáneas. En ese contexto para realizar el clasificador de imágenes se utilizó el programa MATLAB que cuenta con una Red Neuronal Convolucional (CNN) denominada Alexnet. Esta red tiene un conjunto de más de 1 millón de imágenes para facilitar el reconocimiento de las mismas. A esta red se la entrenó nuevamente para que únicamente clasifique ciertos objetos en el video (personas, autos). Finalmente se obtuvo el clasificador con redes neuronales que permite reconocer y clasificar personas y autos en videos.

PALABRAS CLAVE:

- **CNN**
- **ALEXNET**
- **CLASIFICADOR**
- **REDES NEURONALES**

ABSTRACT

Nowadays there are applications that recognize and classify objects in the video surveillance systems and for these it is require processing the video as fast as they can due to the real-time videos. One of the methods to achieve these applications is using neural networks. In the present research project, an image classifier was performed using convolutional neuronal networks (CNN), which were used to train a classifier whose images to be recognized are a sequence of video frames obtained through a video surveillance camera. The contribution of the project in first place is, classification of images (people, vehicles, trees, houses) from video frames that generally have low quality in definition, lighting, distance of objects, etc., and, on the other hand, the implementation of the classifier with the neural network that requires a good computational capacity to carry out millions of simultaneous operations. In this context, the MATLAB program was used that has a convolutional neural network (CNN) called Alexnet that has a set of more than 1 million images to facilitate the recognition of them. This network was trained again so that it only classifies certain objects in the video (people, cars). Finally, the classifier was obtained that allows to recognize and classify people and cars in videos.

KEYWORDS:

- **CNN**
- **ALEXNET**
- **CLASSIFIER**
- **NEURAL NETWORKS**

CAPITULO 1

INTRODUCCIÓN

1.1 Antecedentes

Los sistemas de videovigilancia han ido creciendo y mejorando de manera exponencial en los últimos años, debido al incremento de accidentes de tránsito, ataques terroristas, robos, asaltos, etc. Dichos sistemas son utilizados para intentar disminuir y solventar todos estos tipos de problemas que se tiene en la actualidad. A lo largo de los años se han creado varias aplicaciones para ayudar a mejorar los sistemas de seguridad, un claro ejemplo es la detección de objetos que ha ayudado a prevenir varios actos terroristas. De la misma manera la detección de rostros y personas han hecho que actos delictivos disminuyan de una manera excepcional.

Los actuales sistemas electrónicos de seguridad utilizan el recurso del video para desarrollar operaciones de vigilancia remota. Las personas encargadas de monitorear de forma permanente dichos sistemas pueden incurrir en problemas tales como el cansancio físico durante sus largas jornadas cotidianas, por lo que es necesario buscar la manera de crear ayudas tecnológicas que faciliten su trabajo. Una de ellas es el reconocimiento y clasificación de imágenes a través del sistema de videovigilancia para ayudar a los operadores cuando en ciertas ocasiones no puedan verificar de manera inmediata algún objeto (persona, auto, moto, etc) sospechoso.

Existen varios métodos para realizar este tipo de reconocimiento de imágenes, pero en los últimos años ha surgido uno que se ha destacado de los demás que son la utilización de las redes neuronales.

En (García P. P., 2013), se realiza un estudio referente a la extracción de patrones característicos de imágenes con la ayuda de las Redes Neuronales artificiales. Con la información obtenida por la red neuronal y con algunos datos de las imágenes a ser clasificadas, se las almacenó en una base de datos y con la ayuda de un servicio web, por medio de un teléfono celular con sistema operativo Android al momento de tomar una captura de alguna imagen con ayuda de la cámara del mismo el programa de clasificación es capaz de reconocer las imágenes.

En (González, 2009), realiza un prototipo de un sistema que detecta la presencia de figuras humanas a partir de imágenes previamente tomadas de un escenario controlado. Este prototipo consta de dos partes: La primera consta de un módulo que realiza la detección de la silueta del objeto que va a hacer analizado. Para lograr esta primera etapa se diferencia el fondo del objeto comparándolo pixel a pixel, para después mediante varias operaciones eliminar el ruido que pueda existir. A continuación, se aplica un algoritmo de detección de bordes con el fin de obtener el contorno del objeto. El mismo que es utilizado para generar las entradas a la red Neuronal. Para la segunda parte la red Neuronal es la encargada de detectar ciertos patrones mediante un previo entrenamiento con varias imágenes que se cargaron a la misma. El autor concluyó que el trabajo con este tipo de redes ha mostrado ser una muy buena opción al momento de crear identificadores de características, pero que a su vez es una opción difícil de diseñar debido a los varios procesos matemáticos que implica.

1.2 Justificación e Importancia

La investigación y el desarrollo de técnicas para el reconocimiento de imágenes abren algunas posibilidades para la detección de eventos en sistemas de videovigilancia. Así, por ejemplo, dentro de un escenario específico, un evento puede constituir la presencia o ausencia de una persona o un objeto en un instante de tiempo determinado. Para aquello será necesario procesar muy rápidamente el video utilizando algoritmos clasificadores de imágenes y herramientas computacionales muy eficientes.

Existen diversas técnicas para implementar clasificadores de imágenes tales como Hiper-cubos, modelos de Mezcla de Gaussianas, evaluación del vecino más cercano (filtros KNN), máquinas de vectores de soporte (SVM), redes neuronales (Perceptrón Multicapa) y algunas de ellas se han aplicado principalmente en los sistemas de búsqueda de imágenes en internet. Todos los clasificadores requieren un gran conjunto de imágenes de distintas clases predefinidas y organizadas en una base de datos, que sirven como referencia para el entrenamiento del clasificador. (Luis A. Palacios-Sánchez, 2006)

Para este trabajo se utilizará un clasificador implementado con Redes Neuronales.

Las redes neuronales usan un proceso de clasificación un tanto diferente a los clasificadores tradicionales debido a que ya no extraen características y no utilizan los denominados descriptores. Las imágenes que entran a la red para ser entrenadas son analizadas pixel a pixel mediante el uso de filtros en las diferentes capas de convolución y las llamadas capaz de *pooling*. A medida que avanzan por las diferentes capas la red va aprendiendo hasta que llegan a las últimas capas y es allí donde ya con todo lo aprendido sobre las imágenes de entrada la red logra realizar la clasificación de las mismas (Alex Krizhevsky, 2012).

El proyecto nace con el fin de obtener un clasificador de imágenes utilizando un modelo de red neuronal convolucional preexistente y entrenarlo nuevamente para que la clasificación sea a manera de la necesidad de los operadores de los centros de vigilancia remota.

1.3 Alcance del Proyecto

Todos los clasificadores antes mencionados requieren un gran conjunto de imágenes de distintas clases predefinidas y organizadas en una base de datos, que sirven como referencia para el entrenamiento del clasificador. En este proyecto utilizaremos redes neuronales convolucionales para entrenar un clasificador cuyas imágenes a ser reconocidas serán una secuencia de cuadros de video obtenido a través de una cámara de videovigilancia.

El funcionamiento de este tipo de redes neuronales las hace mucho más efectivas que utilizar descriptores de imágenes debido a que las redes analizan las imágenes pixel a pixel, además que estas redes cuentan con cierta tolerancia a pequeñas perturbaciones en los datos de entrada. Al reducir la resolución, las mismas características corresponderán a un mayor campo de activación en la imagen de entrada.

El aporte del proyecto tiene que ver en primer lugar, en lograr la clasificación de imágenes (personas, vehículos) desde los frames de video que generalmente tienen baja calidad en definición, iluminación, distancia de objetos, etc., y, por otro lado, la implementación del clasificador con la red neuronal requiere una buena capacidad computacional para realizar millones de operaciones simultáneas. En ese contexto también se medirá el desempeño de las herramientas que permitan realizar estas operaciones en el menor tiempo posible. (García García, P. P., 2013).

Finalmente se expondrán las conclusiones mediante un análisis de los resultados obtenidos de la utilización del programa MATLAB conjuntamente con el uso del GPU (*Graphics Processing Unit*) que es un dispositivo que hace la función de procesadores de ayuda, y se los utiliza para el tratamiento de gráficos. Se plantearán las posibilidades de ampliar la investigación a los diferentes campos, en base a la investigación realizada presentando un análisis de ventajas frente a las limitaciones tanto en *software* como en *hardware* presentadas.

1.4 Objetivos

1.4.1 General

Realizar el reconocimiento de imágenes en frames de video capturados a través de un sistema de video vigilancia.

1.4.2 Específicos

- Investigar el contexto teórico de la clasificación de imágenes.
- Implementar un clasificador de imágenes utilizando redes neuronales.
- Evaluar el desempeño del clasificador con imágenes de frames de video
- Analizar los datos obtenidos en el entrenamiento de la Red Neuronal.
- Desarrollar pruebas de rendimiento de la GPU

.

1.5 Resumen de contenidos

En el presente trabajo de investigación los contenidos tanto en teoría como en la práctica se han distribuido de la siguiente manera: En el capítulo 2 se habla de el fundamento teórico de los clasificadores de imágenes, sus clases, etc. Contiene además fundamento teórico de las redes neuronales y su clasificación, se analizan las variedades de redes neuronales que existen en especial se hace un análisis respecto a las redes neuronales convolucionales (CNN). Finalmente contiene una revisión de la base matemática de la clasificación de imágenes usando redes neuronales.

En el capítulo 3 se describe el hardware y software requerido para este proyecto. En el capítulo 4 se detalla el procedimiento completo para la implementación del clasificador con las diferentes funciones que se necesitaron crear los diferentes scripts en MATLAB. En el capítulo 5 se analiza los resultados obtenidos en las pruebas del clasificador. Finalmente, en el capítulo 6 se describe algunas conclusiones y recomendaciones sobre los resultados de este proyecto.

CAPITULO 2

FUNDAMENTO TEÓRICO

2.1 Clasificadores de imágenes

Realizar tareas de identificación es un proceso trivial para la inteligencia natural, el cerebro humano, incluso en edades muy tempranas es capaz de hacerlo. En los últimos años gracias al progreso en el campo del procesamiento de imágenes y a la mejora de la capacidad de procesamiento de las computadoras la inteligencia artificial es capaz de realizar este tipo de tarea.

Algunas de las primeras aplicaciones del procesamiento de imagen y video eran mejorar la calidad de las imágenes, pero mientras el poder de procesamiento crecía también lo hicieron el número de aplicaciones. En la actualidad el procesamiento de imágenes tiene muchas aplicaciones diversas como astronomía, medicina, deportes o el control de robots para las industrias productivas (Moeslund, 2012).

Una de las tareas más importantes en el procesamiento y análisis de imágenes es clasificar cada píxel como perteneciente a una cierta categoría o tema (Mihaich, 2014)

Los clasificadores de imágenes consisten en tomar una imagen o un objeto y asignarlo a una clase disponible ya sea: carro, casas, personas, animales,

etc. Dichos objetos se pueden definir por varios tipos de características, como tamaño, textura, etc.

Para realizar la clasificación de objetos se necesita definir fronteras entre las diferentes clases. Generalmente estas se las calculan mediante un proceso de entrenamiento en el que se usan las características de una serie de prototipos de ejemplo de las clases (Satué, 2013).

Clasificar un objeto desconocido consiste en asignarlo a la clase en la cual las características usadas durante el entrenamiento tienen más correspondencia con las características del objeto (Satué, 2013).

Se suele usar la clasificación frente a otras técnicas cuando los objetos tienen similitudes, pero sujetas a variaciones desconocidas. Si estas variaciones son muy pequeñas, existen otros métodos más sencillos para reconocer el objeto como por ejemplo el emparejamiento por plantilla. Los clasificadores se usan para (Satué, 2013):

- Segmentar imágenes
- Reconocer objetos
- Se utilizan para el control de la calidad
- Detecta algunos cambios o defectos.
- Se usan para el reconocimiento óptico de caracteres

El proceso de clasificación, independientemente del tipo de clasificador seleccionado, consta de una serie de pasos:

1. Se reúnen muestras de objetos de clases conocidas. Se elige un juego de características (vector de características) apropiado y se calculan las características de los objetos de muestra (prototipos).
2. El conjunto de vectores de características se usa para entrenar el clasificador. Se calculan las fronteras entre las clases.

3. Se extraen las mismas características de los objetos desconocidos a clasificar.
4. El clasificador usa las fronteras calculadas durante el entrenamiento para decidir a que clases pertenecen los vectores de características de los objetos que queremos reconocer.

Los pasos 1 y 2 se realizan fuera de línea, es decir, fuera de la aplicación de reconocimiento, mientras que los pasos 3 y 4 se realizan en línea (Satué, 2013).

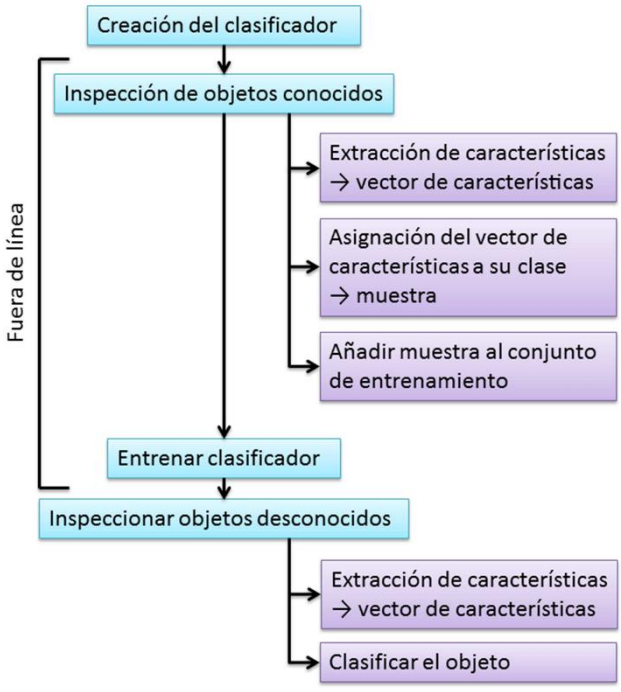


Figura 1. Pasos del proceso de entrenamiento y uso de un clasificador

Fuente: (Satué, 2013)

Existen diversos tipos de clasificadores entre los cuales se destacan los siguientes:

- Modelos de Mezcla de Gaussianas, segmenta imágenes y tareas de clasificación generales (*Gaussian Mixture Models* (GMM)).
- Hiper-Cubos y clasificador de mínima distancia euclídea, segmenta colores.
- Máquinas de vectores soporte (*Support Vector Machines* (SVM)).
- Redes Neuronales.

Para poder decidir a qué clase pertenece un objeto es necesario conocer las diferencias existentes entre las clases, y las similitudes existentes entre los miembros de una misma clase. Este conocimiento se puede obtener analizando las características típicas de los objetos, como: área, circularidad, rectangularidad, anisometría, etc (Satué, 2013).

Las características elegidas se agrupan en un vector de características que a su vez define el denominado espacio de características, donde cada característica se representa en un eje coordenado. El espacio de características puede tener cualquier dimensión (Satué, 2013).

La idea es observar cómo se disponen los vectores de características en el espacio de características para ver si los conglomerados (*clusters*) que se forman se pueden separar. Precisamente, la tarea de un clasificador es separar los *clusters* (durante el proceso de entrenamiento), y asignar los vectores de características de entrada a las clases conocidas. Los clasificadores que usan hiperplanos para separar las clases se denominan clasificadores lineales. Otros clasificadores construyen superficies arbitrarias que permiten separar *clusters* muy cercanos e incluso que se solapen. Estos últimos se denominan clasificadores no lineales (Satué, 2013).

Para nuestro caso no vamos a adentrarnos en todos los tipos de clasificadores ya que el objetivo del proyecto tiene como fin utilizar el método de implementación de las redes neuronales para el clasificador.

2.1.1 Métodos para la Clasificación de imágenes

Para los clasificadores de imágenes existen dos métodos para definir sus clases, que son los siguientes:

- Clasificación No Supervisada
- Clasificación Supervisada

Ambos métodos suelen usarse en conjunto, debido a que son complementarios.

2.1.1.1 Clasificación No Supervisada

La clasificación no supervisada no necesita más información a parte del objeto que se va a clasificar en su algoritmo. Además, necesita algunos parámetros básicos para limitar el número de clases a clasificar. Dichos mecanismos se basan en la búsqueda de las clases con suficiente separabilidad espectral para llegar a conseguir diferenciar unos objetos de otros y de esta manera llegar a clasificarlos.

En la clasificación No supervisada existen algunos métodos o algoritmos que permiten realizar la categorización de los objetos.

2.1.1.2 Clasificación Supervisada

Para la clasificación supervisada se va a disponer de áreas de entrenamiento. Estas áreas que contienen información prioritaria y anterior de la clase a las que los objetos vayan a pertenecer. Servirán para generar una huella espectral muy característica de las clases. En palabras más simples en la clasificación supervisada se cuenta con modelos previamente clasificados

como pueden ser un grupo de objetos agrupados que cuentan con características comunes.

La clasificación supervisada consta de dos fases. En la primera se tiene un grupo de entrenamiento que servirá para el diseño del clasificador y además consta de un conjunto de validación que sirve para la clasificación en si de los objetos. Ambos sirven para la construcción de un modelo para la clasificación.

La segunda fase es ya todo el proceso general de clasificación de los objetos, que no se conoce a que clase pertenecen.

Para ambos casos, la clasificación Supervisada y No Supervisada existen algunos algoritmos que son:

- Algoritmos no Estadísticos
- Algoritmos Estadísticos Clásicos
- Algoritmos basados en inteligencia artificial
- Algoritmos que usan información contextual

2.1.2 Estimación de la precisión de la clasificación de imágenes

Los clasificadores de imágenes no son al 100% precisos, pero se puede determinar su grado de precisión mediante una evaluación. El grado de precisión puede ser definido como el grado en que concuerdan las clases asignadas por el clasificador y los datos de entrenamiento. Para ello se puede recurrir a una validación cruzada como se la llama para realizar una comparación más exhaustiva.

Para realizar este tipo de evaluación existe un método muy efectivo y usado por la gran mayoría de personas que es la matriz de confusión o también llamada matriz de error.

2.1.2.1 Matriz de Confusión

La matriz de confusión es un método que sirve para evaluar la precisión de un clasificador. Esta contiene información acerca de las predicciones realizadas por un Método o Sistema de Clasificación, comparando para el conjunto de individuos en la tabla de aprendizaje, la predicción dada con la clase a la que estos objetos realmente pertenecen. En la figura 2, se muestra una matriz de confusión para un clasificador de dos clases.

		Predicción	
		Negativo	Positivo
Valor Real	Negativo	a	b
	Positivo	c	d

Figura 2. *Matriz de Confusión para Clasificador de dos Clases*

En las filas se tiene el valor real que quiere decir lo que se va a observar en el clasificador, estos valores son el resultado positivo y resultado negativo. En las columnas se va a tener la predicción del modelo, teniendo así el valor positivo de predicción y el valor negativo de predicción.

Para el ejemplo de la figura 2, el valor a representa a los verdaderos negativos que son la cantidad de valores negativos que fueron clasificados correctamente como negativos por el modelo. El valor b representa los falsos positivos que son la cantidad de valores negativos que fueron clasificados incorrectamente como positivos. La variable c representa los falsos negativos que son la cantidad de valores positivos que fueron clasificados incorrectamente como negativos. Y finalmente el valor d representa los verdaderos positivos que son los valores positivos que fueron clasificados correctamente como positivos para el modelo.

La matriz de confusión tendrá algunas medidas de evaluación para verificar el grado de acierto que va a tener el clasificador. Entre ellas están:

Precisión (P). - La precisión de un modelo de predicción se la puede conocer también como la proporción del número total de predicciones que son correctas respecto al total.

Su fórmula se representa como se muestra en la ecuación (1):

$$P = \frac{\mathrm{a+d}}{\mathrm{a+b+c+d}} \qquad (1)$$

Para la precisión en la matriz de confusión se debe tener mucho cuidado. Por ello se debe ser analizado en relación a la dimensión de las clases.

Precisión Positiva (PP). - La precisión positiva es netamente la proporción de casos positivos que se identificaron correctamente. Su fórmula se representa como se muestra en la ecuación (2):

$$PP = \frac{\mathrm{d}}{\mathrm{c+d}} \qquad (2)$$

Precisión Negativa (PN). - La precisión negativa es netamente la proporción de casos negativos que se identificaron correctamente. Su fórmula se representa como se muestra en la ecuación (3):

$$PN = \frac{\mathrm{a}}{\mathrm{a+b}} \qquad (3)$$

Falsos Positivos (FP). – Los falsos positivos son netamente la proporción de casos negativos que se clasificaron incorrectamente como positivos. Su fórmula se representa como se muestra en la ecuación (4):

$$FP = \frac{\mathrm{b}}{\mathrm{a+b}} \qquad (4)$$

Falsos Negativos (FN). – Los falsos negativos son netamente la proporción de casos positivos que se clasificaron incorrectamente como negativos. Su fórmula se representa como se muestra en la ecuación (5):

$$FN = \frac{\mathrm{c}}{\mathrm{c+d}} \qquad (5)$$

Asertividad Positiva (AP). – Asertividad Positiva indica netamente la proporción de buena predicción para los positivos. Su fórmula se representa como se muestra en la ecuación (6):

$$AP = \frac{\mathrm{d}}{\mathrm{b+d}} \qquad (6)$$

Asertividad Negativa (AN). – Asertividad negativa indica netamente la proporción de buena predicción para los negativos. Su fórmula se representa como se muestra en la ecuación (7):

$$AN = \frac{\mathrm{a}}{\mathrm{a+c}} \qquad (7)$$

2.2 Redes neuronales

Las Redes Neuronales Artificiales (RNA) o sistemas conexionistas son sistemas de procesamiento de la información cuya estructura y funcionamiento están inspirados en las redes neuronales biológicas. Consisten en un conjunto de elementos simples de procesamiento llamados nodos o neuronas conectadas entre sí por conexiones que tienen un valor numérico modificable llamado peso (Moreno, 2002).

La actividad que una unidad de procesamiento o neurona artificial realiza en un sistema de este tipo es simple. Normalmente, consiste en sumar los valores de las entradas (*inputs*) que recibe de otras unidades conectadas a ella, comparar esta cantidad con el valor umbral y, si lo iguala o supera, enviar activación o salida (*output*) a las unidades a las que esté conectada. Tanto las entradas que la unidad recibe como las salidas que envía dependen a su vez del peso o fuerza de las conexiones por las cuales se realizan dichas operaciones (Moreno, 2002).

Para entrenar a un sistema conexionista o a una red neuronal, en la realización de una determinada clasificación es necesario realizar dos operaciones. Primero, hay que seleccionar una muestra representativa con respecto a dicha clasificación, de pares de entradas y sus correspondientes salidas. Segundo, es necesario un algoritmo o regla para ajustar los valores modificables de las conexiones entre las unidades en un proceso iterativo de presentación de entradas, observación de salidas y modificación de las conexiones (Moreno, 2002). En la figura 3 se presenta una red neuronal. Donde X (X_1, X_2, … X_n), son las entradas que entran a la red, $W^{(1)}$ es la capa oculta de la red neuronal, $W^{(2)}$ es la capa de salida de la red, y las variables Y (Y_1, Y_2, …Y_m) son las salidas de la red neuronal.

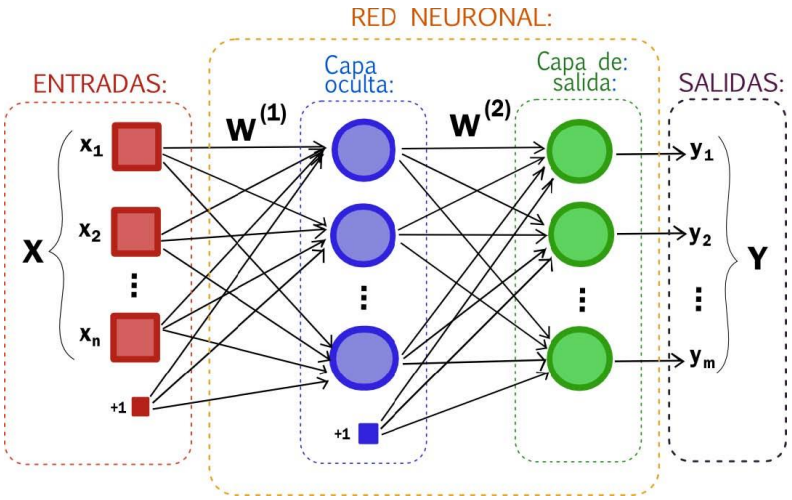


Figura 3. Estructura General de una red Neuronal

Fuente: (García V. G., 2010)

2.2.1 Propiedades de las redes neuronales

Las redes neuronales representan modelos simples del sistema nervioso central; son un conjunto de elementos altamente interconectados que tienen la habilidad de responder simultáneamente a distintas entradas y aprender en entornos cambiantes. Las redes neuronales artificiales han demostrado ser efectivas como procesos computacionales en varias tareas, como, por ejemplo, en el reconocimiento de patrones. Estas exponen un gran número de características deseables, algunas de las cuales no se encuentran en los sistemas convencionales. A continuación, se enumeran estas propiedades (Mihaich, 2014):

- Son muy robustas y tolerantes a fallas.
- Tienen la posibilidad de manejar información difusa, con ruido, incompleta o inconsistente.
- Posee un alto grado en cuanto a paralelismo.
- Tiene capacidad de generalizar.
- Se caracteriza por su aprendizaje adaptativo.

2.2.2 Elementos y características

Como se conoce las redes neuronales artificiales son un intento de imitar la manera en que el cerebro humano se comporta, pero de una manera mucho más simple en base a sus elementos más importantes. Los cuales se describen a continuación:

2.2.2.1 Neurona Artificial

Las redes neuronales artificiales están formadas por una serie de procesadores elementales, denominados neuronas artificiales. Estas constituyen dispositivos simples de cálculo que, a partir de un vector de

entradas procedentes del mundo exterior o de un vector de estímulos recibidos de otras neuronas, proporcionan una respuesta única (salida). Resulta útil la caracterización de tres tipos de neuronas artificiales (Mihaich, 2014):

- Neuronas de entrada: obtienen señales del entorno
- Neuronas de salida: envían señales fuera del sistema cuando ya haya terminado el tratamiento de toda la información.
- Neuronas ocultas: obtienen estímulos y envían salidas dentro del sistema sin ayuda del exterior. Dentro de las mismas se realiza el procesamiento de toda la información.

En la figura 4 se muestran los tipos de neuronas artificiales.

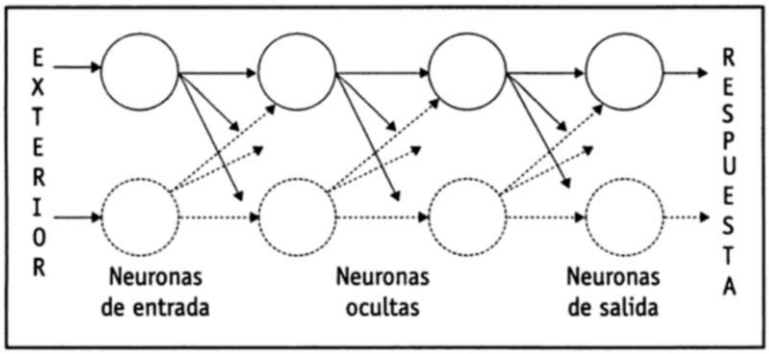


Figura 4. Tipos de Neuronas Artificiales

Fuente: (Raquel Flórez López, 2008)

La información memorizada en el cerebro depende de los valores sinápticos representativos de las conexiones existentes entre las neuronas. De la misma manera, en las redes neuronales artificiales el conocimiento se encuentra representado en los pesos de las conexiones entre las neuronas artificiales, por lo que el proceso de aprendizaje o entrenamiento implica cierto número de cambio de estas conexiones (Mihaich, 2014).

2.2.2.2 Arquitectura de las Redes Neuronales Artificiales

La arquitectura de las redes neuronales artificiales se basa en cómo están organizadas y dispuestas las neuronas y todas las conexiones que existen entre ellas. Depende de cuatro parámetros principales (Mihaich, 2014):

- Número de capas del sistema
- Número de neuronas por capa
- Grado de conectividad entre las neuronas
- Tipo de conexiones neuronales

En función de sus características más notables las redes neuronales se clasifican de la siguiente manera:

- Clasificación según la topología o estructura de red
- Clasificación según su algoritmo de aprendizaje

A continuación, veremos una breve característica de los diferentes tipos de red neuronal según su topología. Las redes neuronales de acuerdo a su topología se clasifican en 2 tipos:

- Redes Neuronales Monocapa: como su nombre lo indica son redes que tienen únicamente una capa y debido a esto las neuronas tienen que crear varias conexiones laterales para conectarse con otras neuronas. Su uso más común es en el ámbito de los circuitos eléctricos debido a su topología.

- Redes Neuronales Multicapa: A diferencia de las redes monocapa las redes Multicapa tienen más de 2 neuronas.

En la figura 5 se muestran la clasificación de las redes de acuerdo a su topología.

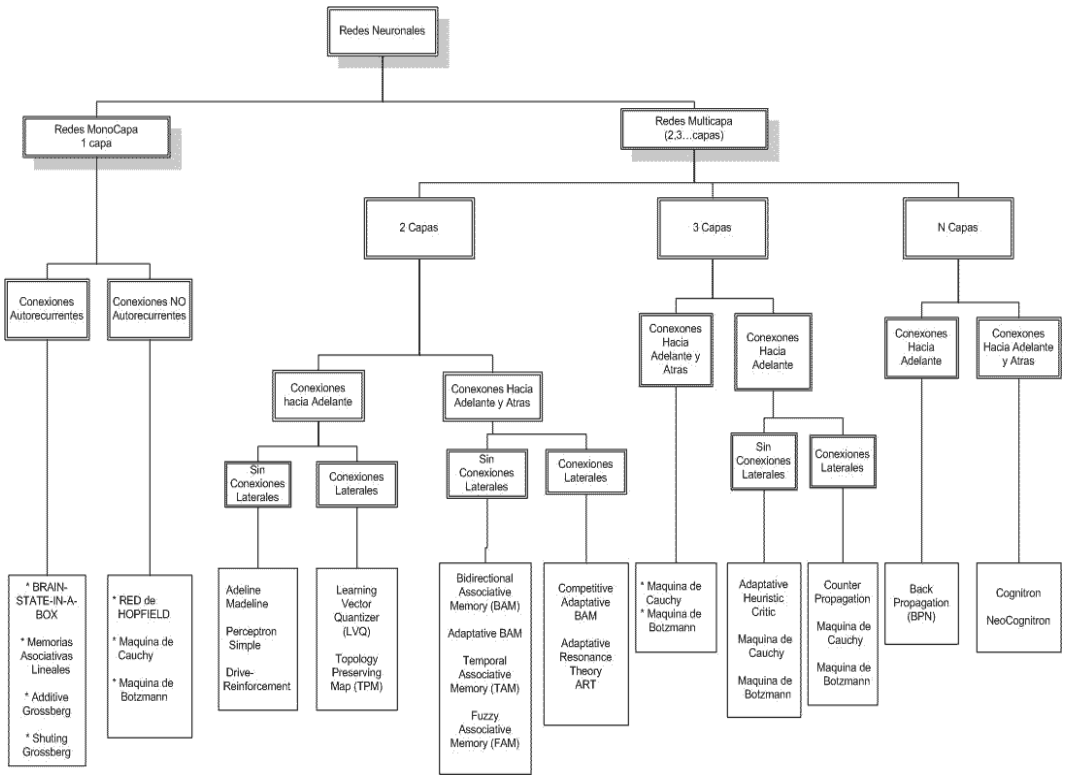


Figura 5. Clasificación de las redes según su topología

Fuente: (Ballesteros, 2008)

2.2.3 Tipos de redes neuronales

Existen varios tipos de redes neuronales artificiales, en esta sección se hablarán de los más importantes:

2.2.3.1 Perceptrón Simple

El perceptrón es la red más antigua que existe. Esta red fue desarrollada en el año de 1943. Lo que hace este tipo de red es sumar señales de entrada, luego las multiplica por valores de pesos que son seleccionados al azar, el mismo valor se lo compara con un patrón para verificar si la neurona esta

activa, si el valor comparado es mayor, significa que la salida nos da un valor de 1, sino da un valor de 0 (Tepán, 2013).

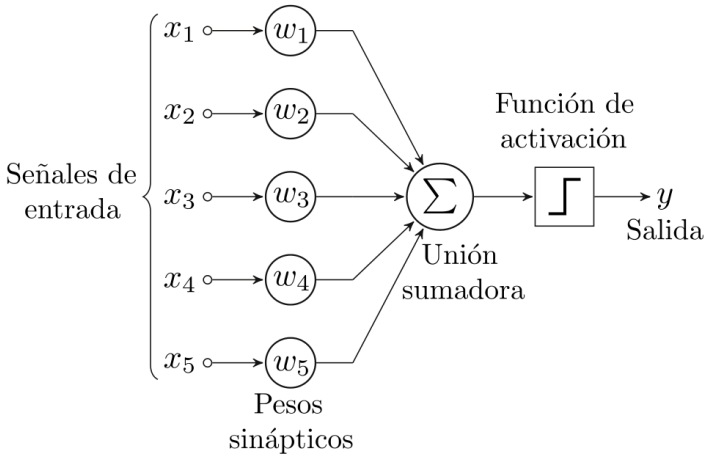


Figura 6. Estructura del Perceptrón Simple

Fuente: (Antonio J. Serrano, 2010) (Tepán, 2013)

En la figura 6, las señales de entrada están representadas por X_1, … X_5. Los pesos reales se los representa con W_1, … W_5. La unión sumadora es el producto escalar entre W y X. La función de activación define la salida de un nodo dada una entrada o un conjunto de entradas. Lo que quiere decir que el valor será un 1 o un 0. Este valor de salida será representado por la letra Y.

Este tipo de red es muy usada para clasificación de objetos ya que obtiene resultados buenos siempre y cuando los patrones sean linealmente separables.

2.2.3.2 Red de Hopfield

La red de Hopfield pese a ser una red de una sola capa es considerada una de las más importantes ya que ha influenciado para desarrollar nuevas redes y nuevos algoritmos. Esta red también puede mostrarse como una red bicapa

es decir que tiene 2 capas, distribuidas en capa de sensores y la otra capa de procesamiento (Ballesteros, 2008).

En la figura 7 se muestra la Red de Hopfield. Donde las entradas son representadas por X_1, … X_N. Las salidas de la red son Y_1, … Y_N. El número de neuronas se los representa con números 1, 2, 3, … N.

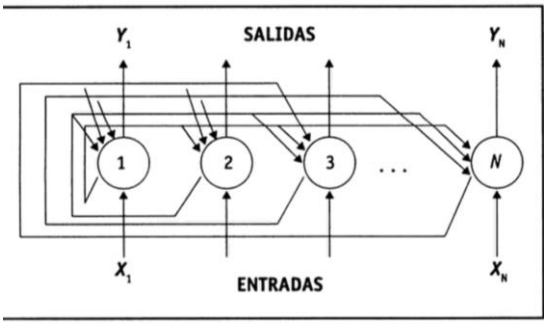


Figura 7. Red de Hopfield

Fuente: (Raquel Flórez López, 2008) (Tepán, 2013)

2.2.3.3 Red Adaline

La red Adaline al igual que la Red de Hopfield es una red de una sola capa, pero a diferencia la red Adaline utiliza otra función de transferencia de tipo lineal. Esta red es uno de los elementos más importantes para el procesamiento digital de señales (Tepán, 2013). En la figura 8 se puede ver una comparación entre la red Adaline y el Perceptrón Simple.

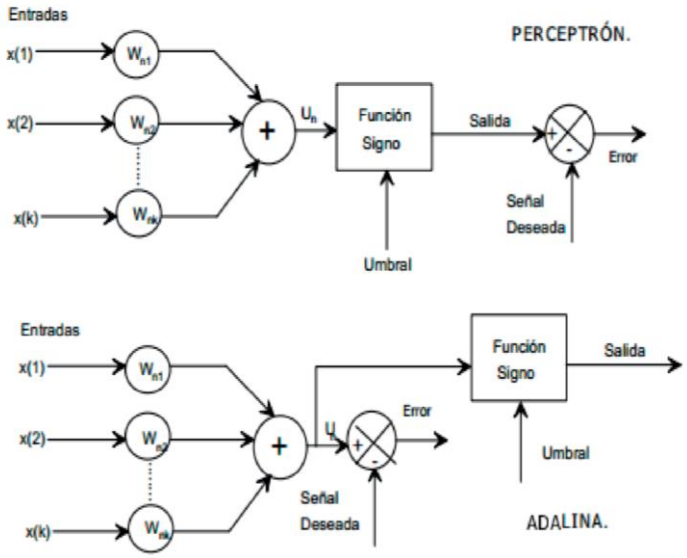


Figura 8. Comparación Perceptrón y Adalina, la red Adalina calcula la señal del error antes de aplicar la función signo.

Fuente: (Tepán, 2013)

Como se observó en la figura 6 el perceptrón tiene las entradas, salidas, pesos, función de activación. De la misma manera la red Adalina pero a diferencia del perceptrón simple, calcula la señal del error antes de aplicar la función signo o función de activación. El umbral está representado por U, que es el grado de inhibición de la neurona.

2.2.3.4 Perceptrón Multicapa

La red Perceptrón Multicapa es una red antigua que está formado por una capa de entrada, una capa oculta y una capa de salida. El perceptrón Multicapa utiliza para el entrenamiento la retro propagación del error. En la figura 9 se muestra la estructura del perceptrón multicapa.

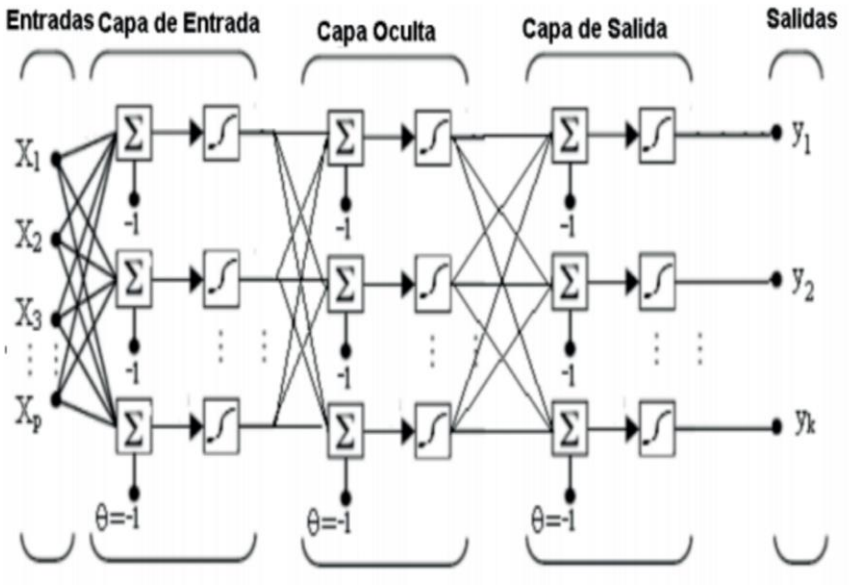


Figura 9. Estructura Perceptrón Multicapa.

Fuente: (Tepán, 2013)

El conjunto de neuronas propaga la señal a la salida, las conexiones que están conectando a las neuronas se las optimizan con ayuda del algoritmo de aprendizaje (Tepán, 2013).

Donde X representa las entradas de la red y Y representa las salidas de la red neuronal.

La red Perceptrón Multicapa usa un algoritmo conocido como algoritmo de retro propagación (*Backpropagation*). Esto significa que la señal se retroalimenta desde la capa de salida hasta la capa de entrada, y de esta manera se optimizan valores de pesos (Tepán, 2013).

El algoritmo tiene dos fases:

- Propagación hacia adelante: En esta fase las señales son propagadas desde la capa de entrada hacia la capa de salida, y de esta manera se genera la salida y además el error que haya tenido la red.

- Propagación hacia atrás: En esta fase de acuerdo al error o errores que haya tenido la capa de salida, se encarga de corregir el valor de los pesos entre las conexiones de las neuronas mediante la retro propagación del error desde la capa de salida hacia la capa de entrada a través de las capas ocultas (Tepán, 2013).

2.3 Aprendizaje Profundo (*DEEP LEARNING*)

Como se conoce durante los últimos años hemos oído hablar de un término que nos llama mucho la atención que es la inteligencia artificial. La misma que ha ido avanzando de una manera muy rápida desde los años 1950s que fue la fecha en que esta nació. La idea surgió de un pensamiento, ¿que tal si los computadores tuviesen ideas propias y pensarán?, y con el pasar de los años esta idea no fue tan descabellada como se pensaba.

Con el nacimiento de la Inteligencia Artificial se crearon varios métodos para realizar lo que varios investigadores soñaron algún día, hacer que los computadores piensen o aprendan por sí solos. Uno de ellos y el más conocido es el Aprendizaje automático (*Machine Learning*) que consta de varias técnicas en el ámbito informático que permiten ayudar a los computadores a tener la capacidad de aprender varias cosas sin tener que ser programadas. Existen varios tipos de algoritmos de aprendizaje automático, entre los más populares se encuentran: Algoritmos genéticos, aprendizaje por refuerzo, árboles de decisión, redes neuronales, y unos de los más usados en la actualidad es el Aprendizaje Profundo (*Deep Learning*).

El Aprendizaje Profundo es un algoritmo que está dentro del Aprendizaje Automático. El mismo que usa varias estructuras de redes neuronales para de esta manera obtener un aprendizaje de varias capas de datos.

2.3.1 Funcionamiento

Para el funcionamiento del Aprendizaje Profundo primero se realiza un mapeo de entrada a un objetivo con ayuda de la red neuronal artificial, la misma que tiene varias capas que están ubicadas de manera jerárquica. El funcionamiento de esta red es simple, la primera capa de la red aprende algo básico y envía esta información a la capa superior. Esta siguiente capa toma esta información básica y lo combina con algo un poco más complejo. Después de este procedimiento envía esta información combinada a la capa superior de esta. Y así sucesivamente hasta que la información combinada llegue hasta la capa final la cual ya tendrá toda la información recopilada por las capas anteriores y de esta manera la red aprenderá debido a cada dato que se obtuvo en cada capa (López, 2017).

La información que cada capa recibe, es almacenada en los pesos de la capa, que son básicamente números. En términos más técnicos se puede decir que la transformación de datos que sucede en cada capa el parametrizada por sus pesos. Se deben encontrar los pesos de todas las capas para que la red haga un mapeo entre los ejemplos de entrada con las salidas, de esta manera la red pueda aprender. Debido a que la red puede tener varios parámetros, pueden existir errores debido a que si un valor falla va a afectar al resto de las capas. Para ello se debe controlar la salida de la red, verificando si es la que se quería obtener desde el principio. La encargada de realizar este proceso es la función de pérdida. Esta toma las predicciones que hace el modelo y los valores objetivos, y de esta manera calcula si estamos cerca del valor que deseábamos. El truco fundamental del *Deep Learning* es utilizar el valor que nos devuelve esta función de pérdida para retroalimentar la red y ajustar los *pesos* en la dirección que vayan reduciendo la pérdida del modelo para cada ejemplo. Este ajuste, es el trabajo del optimizador, el cuál implementa la propagación hacia atrás (López, 2017).

En la figura 10 se muestra el funcionamiento del *Deep Learning.*

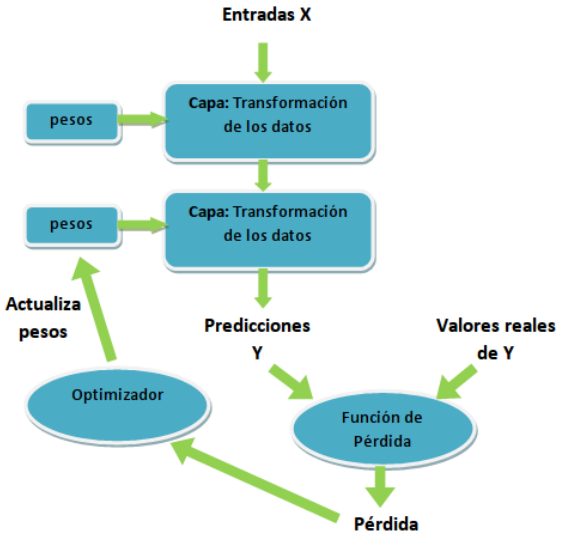


Figura 10. Funcionamiento del Deep Learning.

Fuente: (López, 2017)

Como se mencionó anteriormente para implementar el aprendizaje Profundo se necesita de una red neuronal, y anteriormente ya habíamos visto varios tipos de redes que existen en la actualidad. Para el presente trabajo se va a utilizar una red denominada Red Neuronal Convolucional.

2.3.2 Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (CNN) no varían de las redes neuronales simples como puede ser el perceptrón multicapa. Estas redes tienen una función de pérdida sobre la última capa. La única diferencia entre las redes neuronales convolucionales con el perceptrón multicapa es que las entradas de estas van a ser imágenes, por lo cual al momento de realizar el entrenamiento nos permite ahorrar tiempo, se hace más eficiente y reduce la cantidad de parámetros que utiliza la red.

Este tipo de red tienen una estructura muy similar a la de una red neuronal simple, pero en esta se tiene tres tipos de capas diferentes que son:

- Una capa convolucional
- Una capa de *pooling* o de reducción, con ayuda de esta capa se reduce el número de parámetros requeridos por la red, ya que únicamente obtiene las características más comunes de las imágenes.
- Y finalmente se tiene una capa clasificadora que está conectada en su totalidad, la misma que va a realizar la clasificación de los objetos arrojándonos así el resultado.

Las redes Neuronales Convolucionales son usadas para reconocimiento de imágenes debido a la simplicidad que maneja. En la figura 11 se puede apreciar una diferencia al momento de realizar un clasificador de imágenes utilizando *Machine Learning* y *Deep Learning* con una red neuronal convolucional.

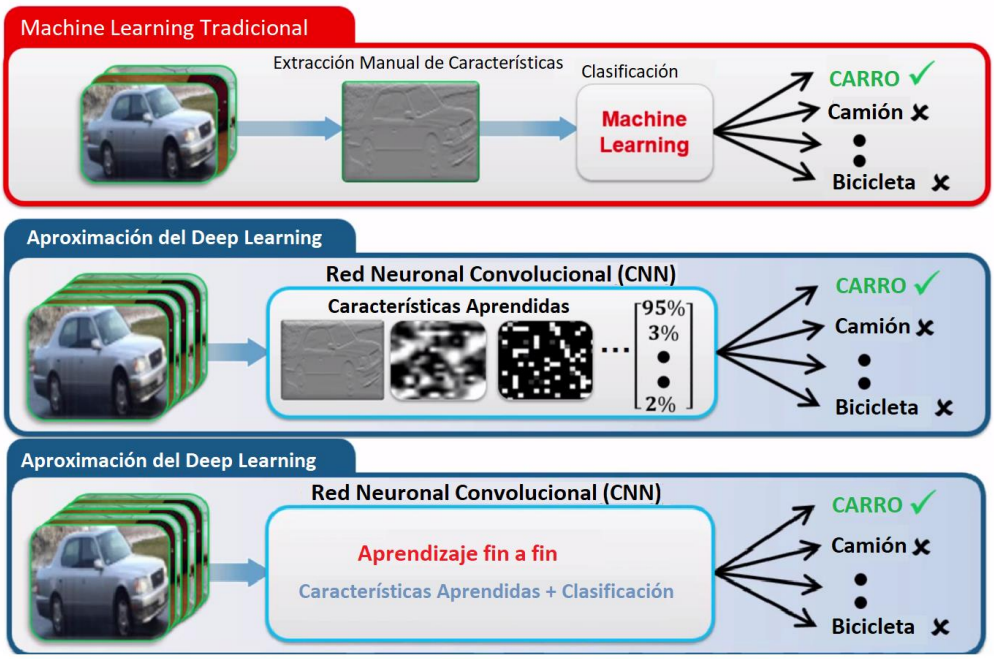


Figura 11. Clasificación de imágenes utilizando *Machine Learning* vs. Clasificación de imágenes con *Deep Learning* y una red neuronal convolucional.

Fuente: (García L. , 2017)

Claramente se puede observar en la figura 11 que al utilizar una red convolucional se optimizan recursos al momento de realizar la clasificación de imágenes.

El programa MATLAB permite entrenar una red neuronal convolucional tanto en una CPU (*Central Processing Unit*), que es el hardware que se encuentra dentro de un computador, como en una GPU, o a su vez con varios GPUs en paralelo, de esta manera se facilita y mejora el procesamiento.

Debido a que las redes Neuronales Convolucionales poseen varias capas para realizar la clasificación, las GPUs van a ser un papel fundamental para poder realizar este procesamiento en menor tiempo.

A continuación, se muestra en la figura 12 una arquitectura típica de la Red Neuronal Convolucional.

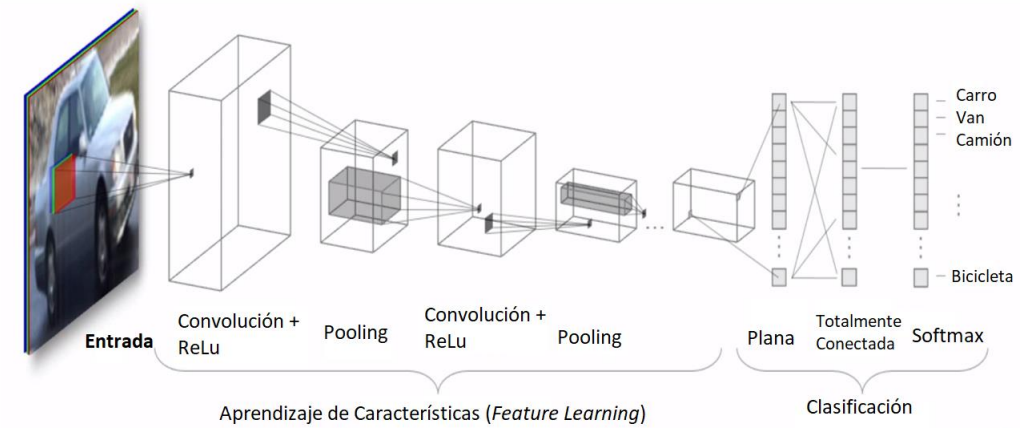


Figura 12. Arquitectura de una Red Neuronal Convolucional.

Fuente: (García L. , 2017)

El primer conjunto de capas Aprendizaje de Características (*Feature Learning*) está detectando características, mientras que las capas finales realizarán la clasificación de las imágenes.

Como se observa en la figura 12 en la entrada de la red se tiene el primer conjunto de capas cuyos parámetros van a venir determinados por un conjunto de imágenes que tendrán un determinado ancho y alto, y tres canales: rojo, verde y azul (Red, Green, Blue (RGB)). A continuación, se tendrán varias capas de tipo convolución, capa de Unidad Lineal Rectificada (*Rectified Linear Unit* (ReLu)) que actúa como una función de activación y capa *Pooling* que reduce el tamaño de las capas. Tendrán la funcionalidad del aprendizaje de características de las imágenes. Cuando se llega al último volumen de capas se la aplanará por así decirlo o se la estirará en un vector, para de esta manera poder trabajar con redes totalmente conectadas y finalmente la capa *softmax* que permitirá determinar a que etiqueta corresponde la imagen de entrada.

2.3.2.1 Primera Capa de una Red Neuronal Convolucional (Capa Convolucional)

La primera capa de una Red Neuronal Convolucional (CNN) va a ser una capa convolucional. La mejor manera de explicar esta primera capa es imaginarse a una linterna que brilla sobre la parte derecha de arriba de una imagen a esta linterna se la conocerá como un filtro y al brillo de esta linterna se la conocerá como el campo de recepción. El filtro es un arreglo de números o en este caso llamados pesos. El filtro va a ir recorriendo la imagen o realizando una convolución, e ir multiplicando los valores en el filtro con los pixeles originales de la imagen. Esto se va a ir sumando hasta que el filtro haya recorrido toda la imagen. Cada una de estas ubicaciones en la que el filtro recorre va a producir un número. Después de haber recorrido toda la imagen se tendrá un arreglo de números a los cuales se los llamará un mapa de activación. En la figura 13 se aprecia un ejemplo de un filtro de 5X5 que se encuentra realizando una convolución alrededor de una imagen de entrada y produce un mapa de activación (Deshpande, 2016).

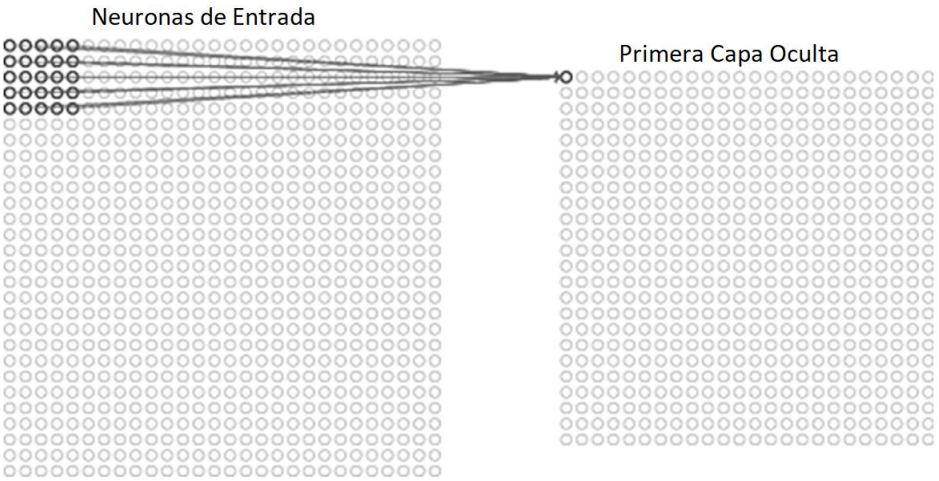


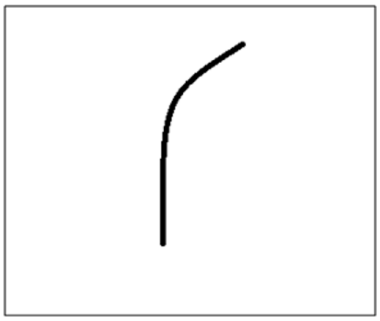
Figura 13. Filtro 5X5 realizando una convolución alrededor de una imagen de entrada y produciendo un mapa de activación.

Fuente: (Deshpande, 2016)

En esta primera capa se va a poder utilizar un sin número de filtros. A estos se los puede considerar como un identificador de características de la imagen. Como puede ser simples líneas, bordes, curvas, colores simples, etc. Con ayuda de un ejemplo se va a apreciar de mejor manera estos filtros. En la figura 14 (a) se aprecia una representación de pixel de un filtro, los números de valor 30 representan la característica o porción de la imagen en este caso una curva. En figura 14 (b) se aprecia el filtro de una porción de la imagen.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

(a)



(b)

Figura 14. Representación de pixel de un filtro (a), Visualización de un filtro (b).

Fuente: (Deshpande, 2016)

Los números 30 estarán representando la porción de la imagen. Como se mencionó anteriormente estos valores del filtro se multiplicarán con los valores originales de pixeles de la imagen. En la figura 15 se aprecia dicha multiplicación con un ejemplo.

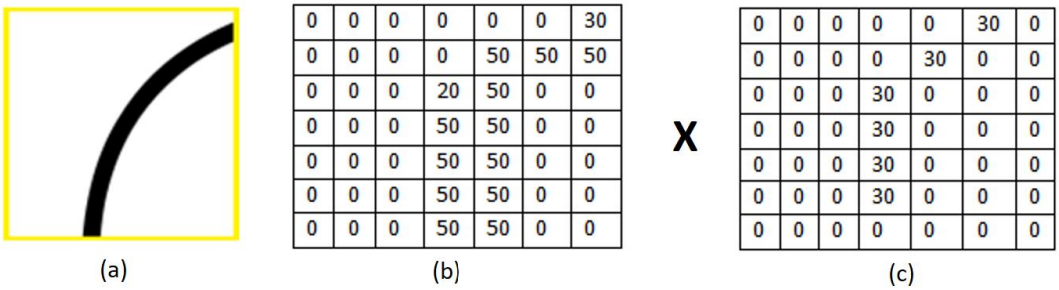


Figura 15. Visualización del campo de recepción (a), Representación de pixel de un campo de recepción (b), Representación de pixel de un filtro (c).

Fuente: (Deshpande, 2016)

Básicamente en la imagen de entrada, si es que existe una forma que represente este filtro en este caso el de una curva, la suma de todas las multiplicaciones va a tener valores altos. En el caso del ejemplo anterior se tiene lo siguiente:

$$Suma\ y\ Multiplicación = (50x30) + (50x30) + (50x30) + (20x30) + (50x30) = 6600$$

En el ejemplo se obtuvo 6600, que quiere decir que el filtro que paso por la imagen es la característica que se estaba buscando. En el caso contrario si se obtiene un valor cerca de 0 significa que la característica que se estaba buscando no concuerda con la del filtro.

2.3.2.2 Capa de Unidad Lineal Rectificada (*Rectified Linear Unit* (ReLu))

La capa de Unidad Lineal Rectificada (*Rectified Linear Unit* (ReLU)) es una capa extremadamente útil debido a que no añade complejidad a la red lo que

significa que esta capa no calcula parámetros o pesos, básicamente actúa como una función de activación.

2.3.2.3 Capa de *Pooling*

Las capas *pooling* son específicamente para reducir el tamaño de las capas y de este modo reducir el número de parámetros de la red. Consiste en realizar una operación de reducción del plano espacial (*downsampling*). Esta capa opera independientemente en cada una de las secciones del volumen de entrada.

2.3.2.4 Capa de Conexión (*Fully Connected Layer*)

La capa de conexión es la capa que se encuentra al final de la Red Neuronal. Lo que hace esta capa es tomar un volumen de entrada (La salida que nos da después de las capas ReLu o capas pool prescedentes) y a la salida va a dar un vector dimensional de valor del número de clases que el programa va a escoger.

2.3.2.5 Funcionamiento

Como se mencionó en la sección anterior lo primero que se va a necesitar para que una Red Neuronal Convolucional realice su entrenamiento es tener un conjunto de imágenes con un determinado ancho y alto, y tres canales (RGB) en la entrada de la red. Así entonces las imágenes pasarán una a una a la primera capa, la misma que obtendrá formas simples de la imagen y a medida que avance de capa a capa esta forma simple se transformará en un patrón característico que sirva para clasificar a la imagen o conjunto de imágenes.

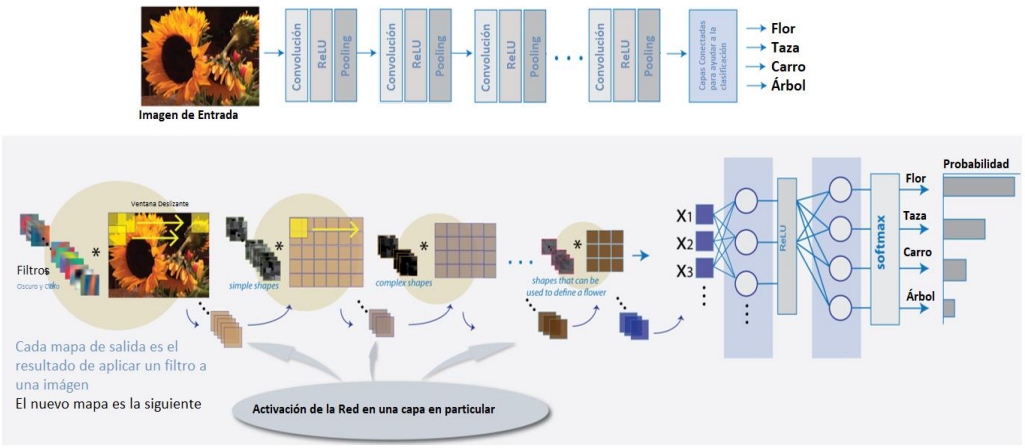


Figura 16. Proceso de las Redes Neuronales Convolucionales mediante capas.

Fuente: (García L. , 2017)

A continuación, se tomó un ejemplo para explicar el procedimiento que lleva a cabo las redes neuronales convolucionales. La primera capa a la que entrarán las imágenes es la capa de convolución. En ella se llevan a cabo una serie de convoluciones para el volumen de datos de entrada como se observa en la figura 17 (a) con una serie de filtros para producir el volumen que se observa en la figura 17 (b). En el ejemplo que se muestra en la figura 17 (a) se tomó valores aleatorios siendo 7 la altura de la imagen de entrada, 3 la profundidad de la imagen y 7 el ancho. De la misma manera en la figura 17 (b) el valor de salida se tiene 3 la altura, 2 la profundidad del volumen de salida y 3 el ancho del volumen de salida. La profundidad del volumen de salida dependerá del número de filtros que se apliquen a la imagen, en este ejemplo se utilizarán 2. Se tendrá que definir el tamaño de los filtros como parte de lo que se conocen como hiper parámetros de la capa de convolución estando la profundidad del volumen de salida determinado por el número de filtros a considerar, en el ejemplo se utilizan dos filtros.

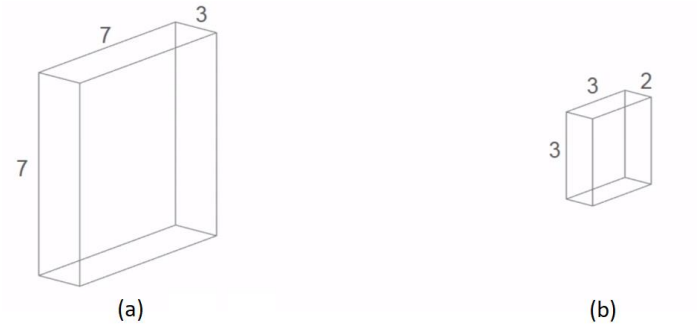


Figura 17. Entrada de los datos a la primera capa. Datos de entrada (a). Datos de Salida después de aplicar el filtro (b)

Fuente: (García L. , 2017)

Para comenzar el procedimiento se inicia con el primer filtro como se observa en la figura 18 siendo W1 el valor del primer filtro.

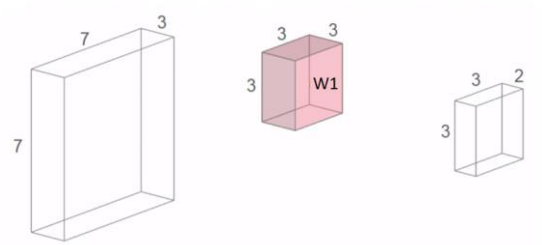


Figura 18. En la capa de convolución se inicia con el primer filtro (W1) capa.

Fuente: (García L. , 2017)

Para cada una de las posiciones que se van a ir cubriendo por el filtro se calcula el producto escalar y se suma el resultado como se observa en la figura 19.

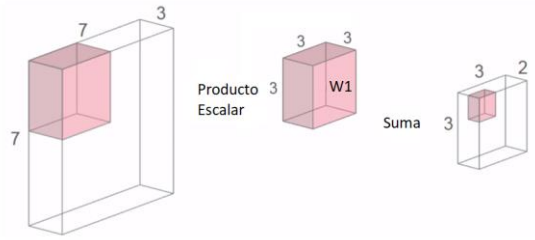


Figura 19. Primera posición del filtro (W1).

Fuente: (García L. , 2017)

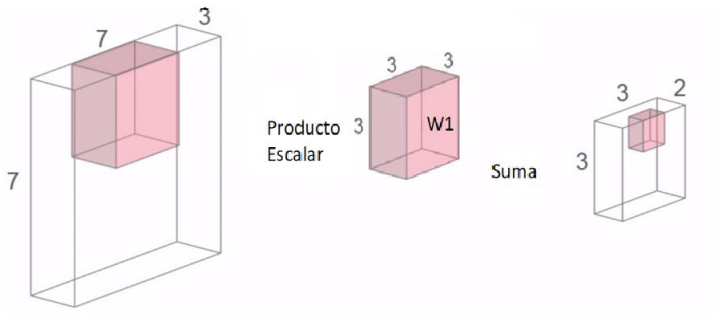


Figura 20. Segunda posición del filtro (W1).

Fuente: (García L. , 2017)

El filtro va a ir recorriendo de izquierda a derecha y luego de arriba hacia abajo hasta haber recorrido toda la imagen. Como se aprecia en la figura 20.

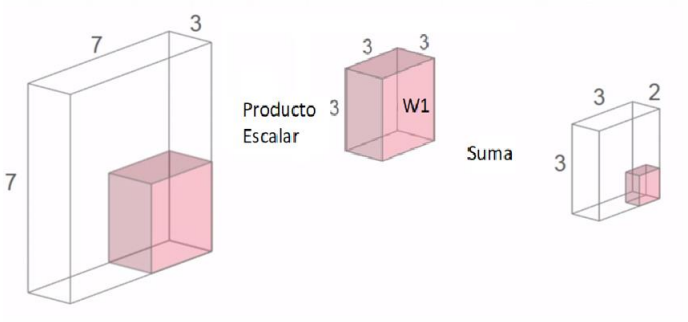


Figura 21. Posición final del filtro (W1), una vez que ha recorrido la imagen para obtener una simple característica.

Fuente: (García L. , 2017)

Con cada aplicación del filtro sobre el volumen de entrada, se obtendrá un único valor escalar como se observa en la figura 21.

A continuación, como se observa en la figura 22, se realiza el mismo procedimiento con el segundo filtro, siendo W2 el valor del segundo filtro:

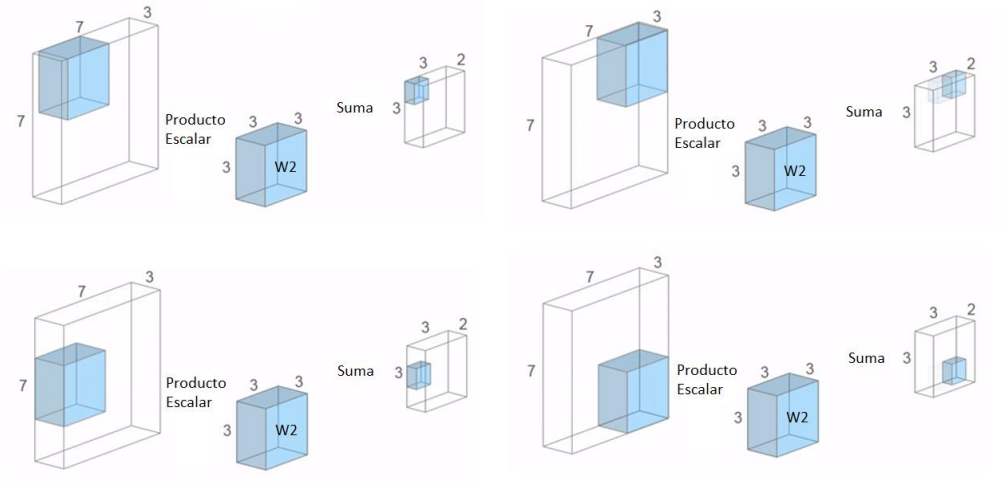


Figura 22. Recorrido del filtro (W2) por la imagen de entrada.

Fuente: (García L. , 2017)

El proceso se va a repetir para todos los filtros, deslizándolos por el volumen de entrada, calculando el valor escalar con la región que corresponde y sumando los valores para obtener el valor de salida, y de esta manera se calcula varios mapas de activación para cada uno de los filtros.

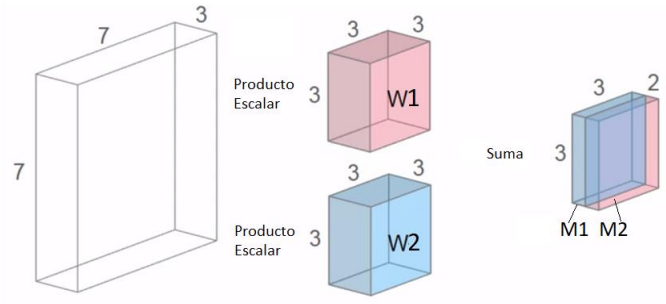


Figura 23. Obtención de los mapas de activación.

Fuente: (García L. , 2017)

Como se indicó al principio de este ejemplo se consideró dos filtros y se obtiene dos mapas de activación M1 y M2 como se observa en la figura 23. Los valores de los filtros estarán siendo aprendidos por la red de modo que se activarán cuando la red vea determinadas características específicas.

Para la realización del proceso en la capa de convolución se habló de los hiper parámetros, a continuación, se mostrarán los más importantes:

- Número de filtros, estará determinado por la letra K. Determinará la profundidad del volumen de salida como se observa en la figura 24, siendo A1 la altura del filtro, B1 el ancho del filtro y D1 la profundidad del filtro. A2 será la altura para el volumen de salida de la capa de convolución, B2 el ancho y D2 la profundidad.

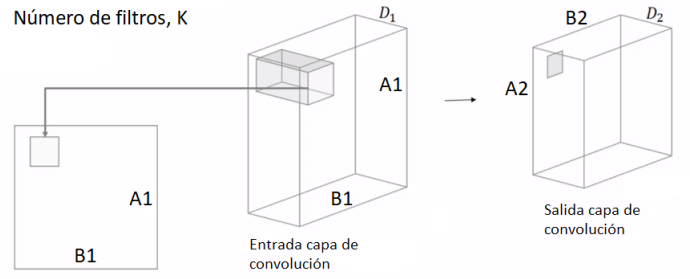


Figura 24. Vista frontal a la entrada en la capa de convolución, Número de filtros (K).

Fuente: (García L. , 2017)

- Tamaño del Filtro, F, se observa en la figura 25.

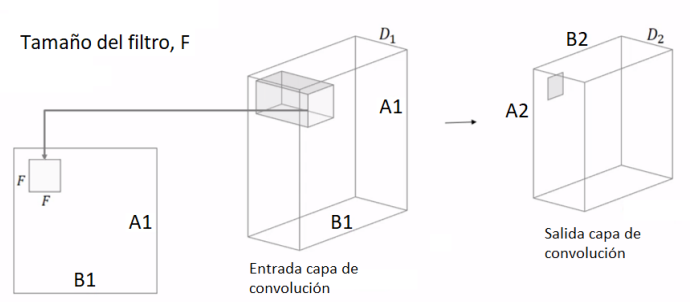


Figura 25. Vista frontal a la entrada en la capa de convolución, Tamaño del Filtro (F).

Fuente: (García L. , 2017)

- *Stride*, S, el *Stride* especifica el intervalo desde el cual aplicar los filtros al volumen de entrada, se observa el *stride* en la figura 26.

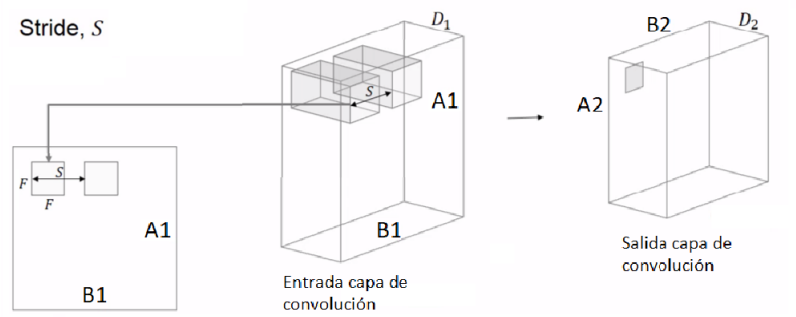


Figura 26. Vista frontal a la entrada en la capa de convolución, *Stride* (S).

Fuente: (García L. , 2017)

- Relleno con ceros (*Zero Padding*), ZP, permite extender inflictamente los lados de la entrada con ceros. Este relleno es relevante cuando se quiere preservar el tamaño del volumen de entrada o para evitar una reducción muy temprana de las dimensiones de la red que nos llevaría a malos resultados. Se muestra en la figura 27.

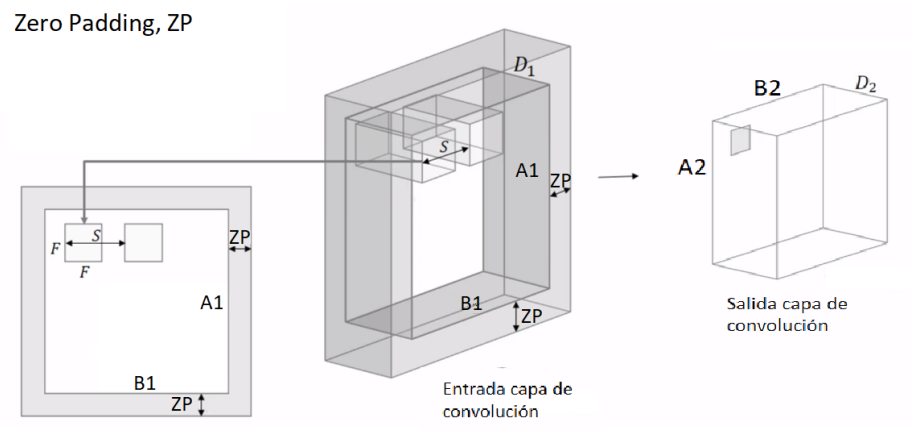


Figura 27. Vista frontal a la entrada en la capa de convolución, Relleno con ceros (ZP).

Fuente: (García L. , 2017)

Una vez ya obtenidos todos los hiper parámetros se los puede calcular con las siguientes fórmulas que se muestran en la ecuación (8), (9) y (10).

$$B2 = \frac{B1-F+2ZP}{S} + 1 \quad (8)$$

$$A2 = \frac{A1-F+2ZP}{S} + 1 \quad (9)$$

$$D2 = K \quad (10)$$

Generalmente una capa de convolución trabaja conjuntamente con las capas ReLU.

Una de las opciones más comunes para las capas Relu es utilizar una función rampa entre 0 y r siendo r un valor que se desee, de modo que los valores inferiores a 0 se fijan a 0.

$$f(r) = \max(0, r)$$

La siguiente capa a la que entrará es la capa de *pooling*. Las operaciones más comunes de la capa *pooling* son el máximo y la media aritmética o promedio. La función principal de la capa *pooling* es reducir progresivamente el tamaño de las capas.

Finalmente, después de realizar todo este procedimiento los datos de las primeras capas entran a la capa de clasificación donde se realizará la elección de la imagen para ver a que clase pertenece.

Para la fase final del entrenamiento de la red se deben tener en cuenta algunos aspectos que no se han mencionado, como: ¿Cómo hacen los filtros en la primera capa para buscar por filos o curvas?, ¿Cómo hace la capa de conexión para saber a que mapas de activación buscar?, ¿Cómo los filtros en cada capa saben que valores tomar?, etc. La manera en la que el computador

puede ajustar los valores de los filtros o pesos es a través de un procedimiento de entrenamiento llamado Propagación en reversa (*backpropagation*) (Deshpande, 2016).

Antes de ver lo que es la *backpropagtion*, se debe analizar algo muy importante que es lo que la red neuronal necesita para de esta manera poder trabajar. En el momento que una red quiere ver las diferentes clases de imágenes, su cerebro por así decirlo se encuentra fresco, no tienen información de ningún tipo. Esto en palabras más técnicas quiere decir que al momento de iniciar la Red Neuronal Convolucional, los valores de los pesos o filtros son puestos al azar. Las CNNs pasan por un proceso de enseñanza en el cual se les indica una gran cantidad de imágenes y cada una con una etiqueta. Para que de esta manera pueda darse cuenta que imagen pertenece a cada categoría.

La propagación en reversa se la divide en 4 secciones diferentes que son:

- Pase adelantado (*forward pass*)
- Función de pérdida (*Loss Function*)
- Pase de reversa (*Backward pass*)
- Actualización de pesos (*Weight Update*)

En la primera sección en *forward pass*, se toma una imagen de entrenamiento y se la pasa por toda la red. Se obtienen todos los pesos, pero con esto todavía no puede llegar a la conclusión de la clasificación de dicha imagen. Por eso va a la siguiente sección la de Loss Function. El dato que se está usando o que se pasó por la red es ya entrenado, es decir que tiene una imagen y una etiqueta. La función de pérdida se puede definirla de algunas maneras, pero la más común es el error cuadrático medio (Mean Squared error (MSE)) (Deshpande, 2016).

La idea es de alguna manera reducir el error total para que sea cercano a 0 por tanto la predicción del clasificador sea acertada. Este proceso se trata de ajustar los pesos para que el error disminuya. Después de este proceso se va a la siguiente sección que es el de *backward pass*, en este se determina que pesos son los que contribuyen a que el error sea bajo y encontrar maneras de ajustarlos para que el error disminuya cada vez más, una de ellas es calcular la derivada de los pesos de una capa en particular. Después de realizar este cálculo la red deberá pasar por la última sección la de actualización de pesos. En esta sección se toman todos los pesos de los filtros y se los actualiza para que puedan cambiar en la dirección opuesta de la gradiente, como se observa en la ecuación (11) (Deshpande, 2016):

$$w = wi - \eta \frac{dL}{dW} \qquad (11)$$

Donde w es el peso, wi es el peso inicial y η es la tasa de aprendizaje (*learning rate*). Este último es un parámetro que es escogido por el usuario. Mientras más grande sea este valor mayor serán los pasos que van a tomar en la actualización de los pesos. Y por lo tanto tomará menor tiempo en entrenar la red. Hay que tener cuidado ya que si este número es muy alto la red no va a tener muy buena precisión.

Todos estos procedimientos son una iteración de entrenamiento. Este parámetro de iteraciones de la misma manera las va a poder elegir el usuario. Mientras más iteraciones el entrenamiento y el clasificador va a tener más efectividad.

2.3.2.6 Transferencia de Aprendizaje (*Transfer Learning*)

La transferencia de Aprendizaje (*Transfer Learning*) es un proceso de tomar modelos de redes que estén previamente entrenadas (los pesos y parámetros ya están entrenados) y a los mismo modificarlos según las diferentes

aplicaciones que quiera realizar el usuario. La idea es que este modelo ya entrenado actúe como un extractor de características. Se removerá la última capa de la red y se la reemplaza con un clasificador propio. Los pesos de las primeras capas ya entrenadas de esta red se los congela y se entrena nuevamente a la red (Deshpande, 2016).

Existen varias redes Neuronales Convolucionales previamente entrenadas, que nos pueden ayudar para realizar una clasificación de manera más fácil únicamente volviendo a reentrenar a la red con los parámetros que se deseen. Existen varias Redes Convolucionales ya entrenadas como son: VGG16, VGG19 y una de las redes más comunes que existen es la Red Neuronal Convolucional AlexNet en la que nos vamos a enfocar debido a que se la utilizó en esta investigación.

2.3.2.7 AlexNet (2012)[1]

La Red Convolucional AlexNet es la red pionera de las redes convolucionales. Fue creada en el año de 2012 por Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton en una competencia que se realiza cada año sobre visión computacional llamada ILSVRC (*ImageNet Large-Scale Visual Recognition Challenge*). Esta Red fue entrenada sobre una base de datos de imágenes llamada ImageNet, la misma que tiene un conjunto de más de 14 millones de imágenes. El modelo de AlexNet está entrenado para clasificar imágenes en 1000 categorías. Por ejemplo, lápices, teléfono, teclados, animales, etc. Los creadores de esta red hicieron un artículo donde discutieron la arquitectura de la red. La red consta con 650,000 neuronas. Utilizaron 5 capas de convolución, algunas capas con *pooling*, y 3 capas de conexión. En total se utilizó 25 capas. Para su entrenamiento se utilizó dos GPUs modelo GTX 580 y el entrenamiento de la misma duro aproximadamente 6 días. Las

[1] https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

imágenes que tiene AlexNet son de un tamaño de 227 x 227 x 3. Para poder utilizar esta red en MATLAB se necesita tener instalado el *Toolbox* de Redes Neuronales para Alexnet (Alex Krizhevsky, 2012).

Esta red consta de 25 capas como se muestra en la figura 28:

1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench', 'goldfish', and 998 other classes

Figura 28. Capas de la red AlexNet.

Fuente: (García L. , 2017)

Las tres últimas capas son las capas de clasificación, para este trabajo se las modificará para obtener la clasificación únicamente de ciertos objetos.

2.3.2.8 ZF NET (2013)[2]

Al siguiente año de la creación de AlexNet, otros modelos de redes neuronales aparecieron para mejorar al pionero de las redes neuronales

[2] https://arxiv.org/abs/1311.2901

convolucionales. En el año de 2013 Matthew Zeiler y Rob Fergus ganaron la competencia creando el modelo de red neuronal convolucional llamada ZF NET. Este modelo superó a AlexNet debido a que el error que aportó el mismo se redujo de 15,4% de su antecesor al 11,2%, dando así una red mucho más precisa. Esta red fue un reajuste de AlexNet ya que se toma su estructura e improvisaron su desempeño.

2.3.2.9 VGG NET (2014)[3]

A pesar de no ser los ganadores del concurso en el año de 2014, Karen Simonyan y Andrew Zisserman de la universidad de Oxford crearon una red muy eficaz llamada VGG NET. Su tasa de error fue del 7,3%. La red tenía 19 capas y que estrictamente usaban filtros de 3 x 3 con un *stride* y un *pad* de valor 1, además utilizaron capas de *pooling* de 2 x 2 con un stride de valor 2. Trataron de hacerla lo más simple posible. A diferencia de AlexNet que utilizaba filtros de 11 x 11. Se la entrenó en 4 GPUs modelo Nvidia *Titan Black* durante alrededor de dos a tres semanas. Su importancia se debe a que reforzó la noción de las redes neuronales convolucionales deben tener capas más profundas para de esta manera mantener más simple al modelo.

2.3.2.10 GoogleNET (2015)[4]

El en año de 2014 Google lanzó su propia red neuronal convolucional la misma con la que ganó el concurso de ILSVRC en el mismo año. Esta red tuvo un error muy bajo en comparación a las otras redes de años anteriores. El error fue de 6,7%. En este nuevo modelo creado por Google se tuvo una notable consideración en memoria y uso de energía. La diferencia entre este modelo y anteriores es que GoogleNET no tiene una estructura secuencial.

[3] http://www.robots.ox.ac.uk/~vgg/research/very_deep/

[4] http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf

Se tiene partes de la red que trabajan de manera paralela, a esto se le dio un nombre, el módulo de incepción. Lo que este módulo permite es que las operaciones que antes se realizaban de manera secuencial trabajen en forma paralela, es decir al mismo tiempo. Pero debido a esto esta red tuvo más de 100 capas en total. Para el entrenamiento de esta red se utilizaron unos poco GPUs de alta tecnología y duró alrededor de una semana.

2.3.2.11 Redes de transformadores espaciales (*Spatial Transformer Networks* (2015))[5]

Las redes de transformadores espaciales son los modelos más actuales que existen por el momento. Sus creadores al igual que el anterior modelo son ingenieros de Google. En este tipo de redes se creó un módulo llamado módulo transformador espacial. Lo que hace este filtro es transformar la imagen de entrada a la red de tal manera que las capas posteriores tengan un tiempo más eficaz para realizar la clasificación. Los autores de esta red se enfocaron en realizar cambios en la imagen antes de que esta entre a la capa de convolución con ayuda del módulo transformador espacial.

[5]https://arxiv.org/pdf/1506.02025.pdf

CAPITULO 3

MATERIALES Y MÉTODOS

3.1 Introducción

En el presente capítulo se muestra el diseño del clasificador de imágenes con redes neuronales con ayuda de MATLAB y las herramientas de procesamiento GPU. Se presentarán los requisitos de *hardware* y *software* que se usaron para la realización del mismo. Además, se muestran las características técnicas de todos los instrumentos que se utilizaron en el proyecto.

3.2 Software MATLAB

El software MATLAB es una herramienta que sirve para realizar un sin número de operaciones matemáticas y con el pasar de los años ha ido creciendo y mejorando en sus funcionalidades. Una de estas mejoras es la utilización de los denominados *ToolBox* que son funciones que se tiene para realizar procesos como: análisis estadísticos de datos, cálculos de variables, entrenamiento de redes neuronales, etc. Otro de los grandes avances del software MATLAB es el poder trabajar en paralelo con la unidad de procesamiento gráfico (GPU) del computador ya que esto facilita y mejora el rendimiento de los programas que se realicen en el mismo.

3.3 Arquitectura de Dispositivos de computo unificadas (CUDA)

La Arquitectura para dispositivos de computo unificadas (*Compute Unified Device Architecture* (CUDA)) es una arquitectura para cálculo en paralelo que

fue creada por la compañía NVIDIA[6]. La misma que fue realizada para que se pueda aprovechar la máxima capacidad de la Unidad de procesamiento gráfico (GPU) y de esta manera el sistema tengo un óptimo rendimiento.

3.3.1 Unidad de Procesamiento Gráfico (GPU)

El GPU es un dispositivo que hace la función de procesadores de ayuda, y se los utiliza para el tratamiento de gráficos. El GPU tiene muchos núcleos para poder realizar un sin número de tareas al mismo tiempo, de esta manera mejora el rendimiento del computador.

3.4 Requisitos del Sistema

Para la realización de este trabajo se utilizó un computador de última generación, cuyas características se detallan en la tabla 1. En el mismo se realizó todo el trabajo.

3.4.1 Características del CPU

En esta sección se muestran las características del CPU en la tabla 1, que se utilizó en el trabajo de investigación.

Tabla 1
Características del CPU utilizado

Marca	Dell
Modelo	Inspiron 7559 Signature Edition
Procesador	Intel Core i7
Memoria RAM	8Gb
Tipo de Sistema	64 bits procesador x64

[6] http://www.nvidia.com/content/global/global.php

3.4.2 Características de la tarjeta de Video

En esta sección se muestran las características de la tarjeta de video que se utilizó en este proyecto. Se observan en la tabla 2. La tarjeta de video ya venía incluida en el CPU.

Tabla 2
Características de la tarjeta de video

Tipo de Chip	Intel HD Graphics Family
Tipo de DAC	Internal
Nombre Adaptador	Intel HD Graphics 530
Visualización	3840 x 2160
Memoria	8128 Mb

3.4.3 Características de la GPU

Se detallan las características de la GPU utilizada en el proyecto en la tabla 3.

Tabla 3
Características de la GPU

Modelo	GeForce GTX 960M
Núcleos CUDA	640
Gráficos del Reloj	1097 MHz
Tasa de datos de Memoria	5010 MHz
Capacidad Computacional	5.0
Interfaz de Memoria	128-bit
Ancho de banda de Memoria	80.16 GB/s
Memoria Gráfica Disponible	8128 MB
Memoria compartida del Sistema	4096 MB GDDR5
Bus	PCI EXPRESS X8 Gen3

3.4.3.1 Procesamiento en paralelo del GPU

Para el uso de las redes convolucionales en MATLAB, se necesita tener instalado un *Toolbox* específico: *Parallel Computing Toolbox*. Generalmente este *Toolbox* viene instalado por defecto en MATLAB al momento de la instalación del software. Caso contrario lo primero que se necesita es ir a la pestaña de *Add-Ons* > subpestaña *Get Add-Ons*. Como se observa en la figura 29.

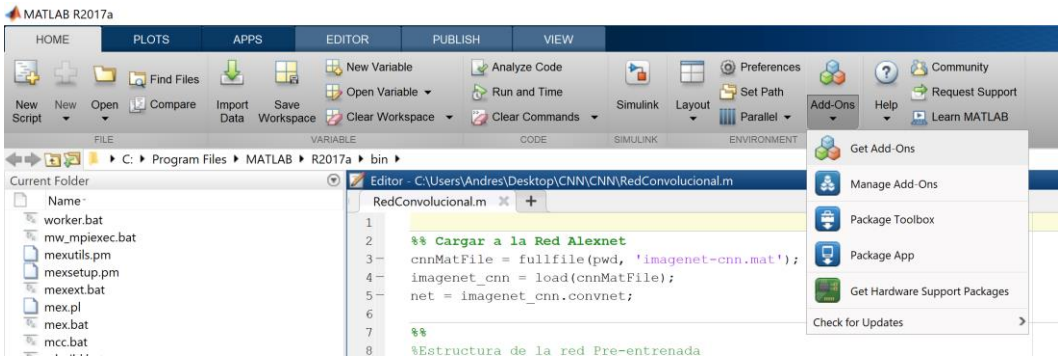


Figura 29. Pestaña *Add-Ons*

Y buscar el nombre de *Parallel Computing Toolbox* e instalar. Se muestra en la figura 30.

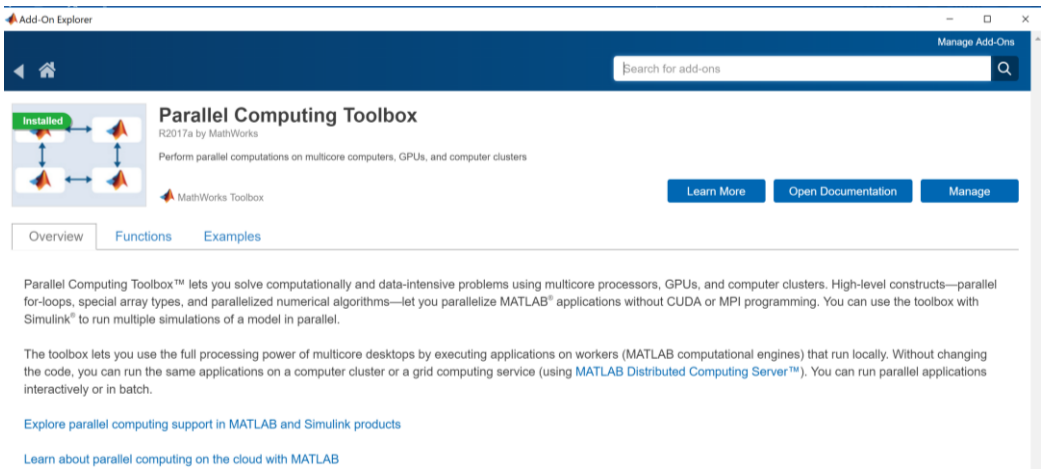


Figura 30. *Parallel Computing Toolbox* en MATLAB.

Una vez instalado el *Toolbox*, para que el GPU realice el procesamiento en paralelo el procedimiento. Se lo puede realizar de dos maneras:

La primera es ir a la parte de abajo izquierda del MATLAB y hacer clic en la opción de *Start Parallel Pool*:

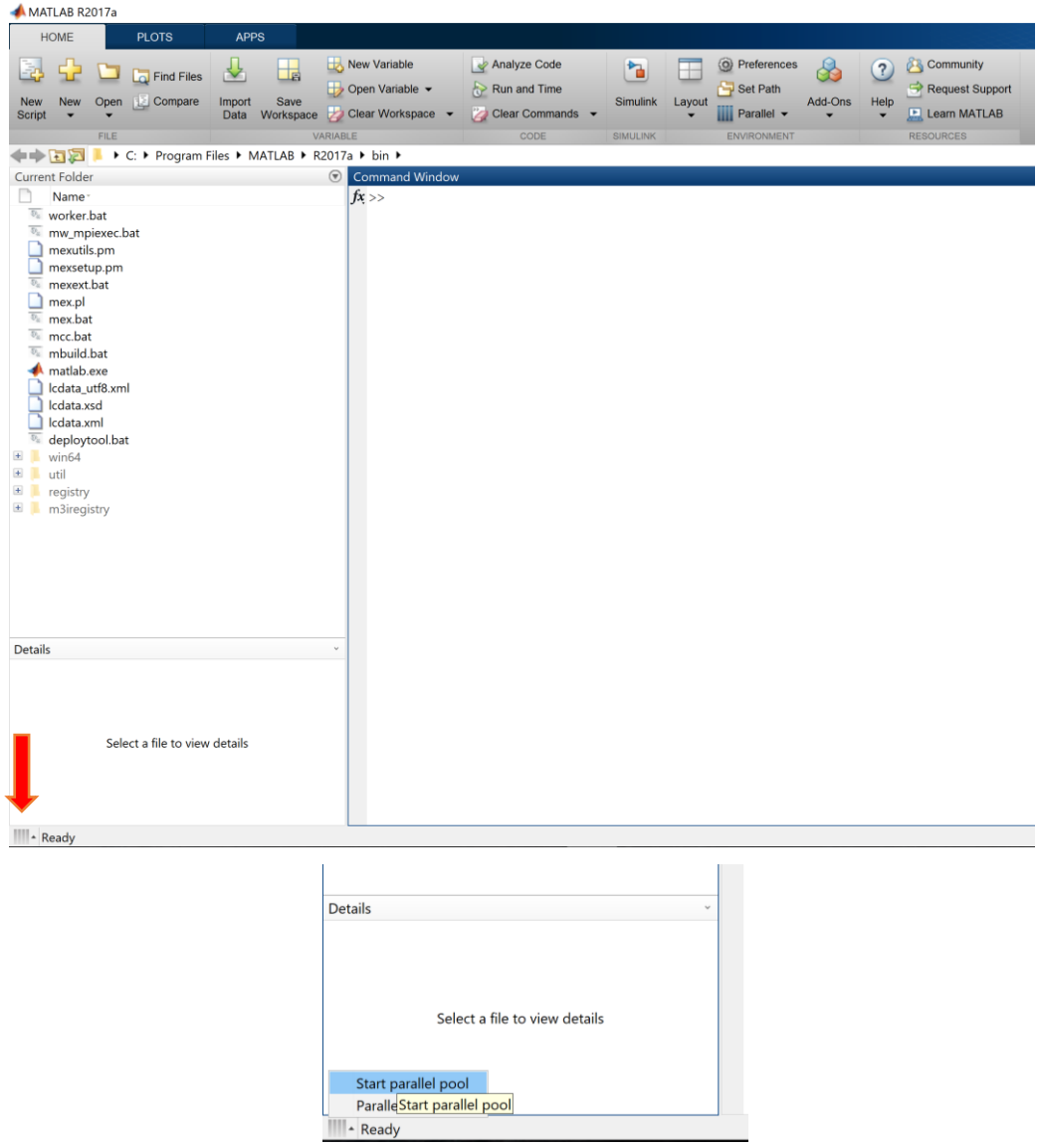


Figura 31. Iniciar el *Parallel Pool*

Y automáticamente el *parallel pool* se iniciará usando el perfil que esta por defecto. De esta manera se podrá trabajar con el procesamiento en paralelo de GPU.

La segunda manera es con comandos, para que se inicie el *parallel pool* se debe escribir el siguiente comando:

parpool (‘local’)

3.4.4 Características de MATLAB

Debido a que se utilizaron Redes Neuronales se necesitó de la Neural *Network Toolbox™ Model for AlexNet Network support package*, la versión que se utilizó para MATLAB es MATLAB R2017a que es la última versión que existe en el mercado. En versiones anteriores esta herramienta para la Red AlexNet no corre de manera correcta.

3.4.4.1 Instalación del *Toolbox Model* for AlexNet

Para la instalación se siguieron los siguientes pasos:

Primero ingresamos al MATLAB. Una vez en el programa vamos a la pestaña *HOME*, ingresamos a la subpestaña *Add-Ons* y damos clic en la opción *Get Add-Ons*

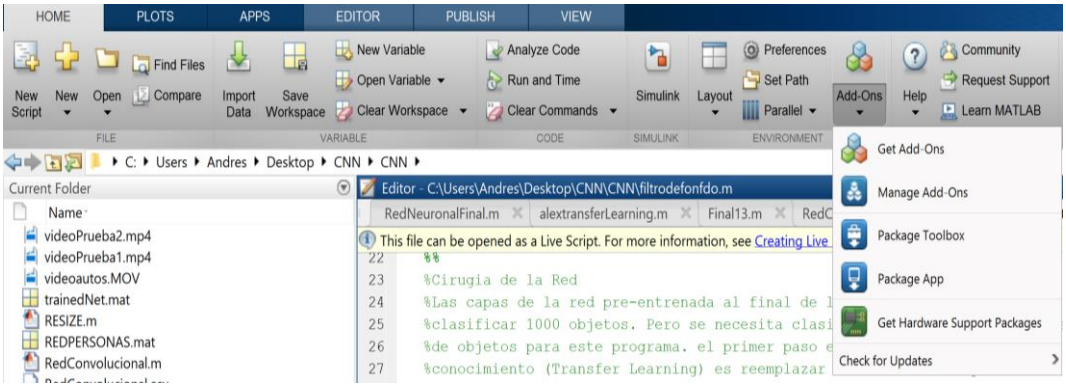


Figura 32. Ingreso a la pestaña *Get Add-Ons*

Una vez en esta ventana, vamos a buscar el nombre de AlexNet *ToolBox* y damos clic en instalar.

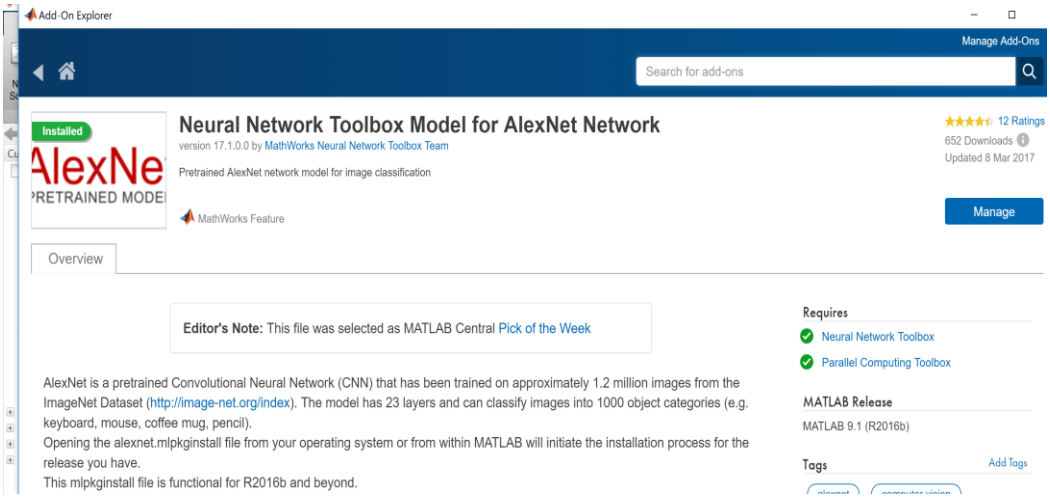


Figura 33. Ventana de *Add-Ons*, instalación del *ToolBox* for AlexNet

CAPITULO 4

DESARROLLO E IMPLEMENTACIÓN DEL CLASIFICADOR DE IMÁGENES USANDO REDES NEURONALES

En esta sección se describe el desarrollo del clasificador utilizando redes neuronales. Para el mismo se va a utilizar la red neuronal AlexNet. Se va a volver a entrenar a dicha red modificando las tres últimas capas para que pueda clasificar únicamente autos y personas. Para que pueda clasificar las imágenes en frames de video se va a implementar un algoritmo el cual permite quitar el fondo del video y detectar a los objetos en este caso autos y personas que se encuentren en movimiento.

4.1 Requerimientos del Diseño

Debido a que las Redes Neuronales tienen alta capacidad de predicción, se las utilizará para ver cómo funciona cuando se las entrena para realizar un clasificador, pero con varios frames de video.

4.2 Diagrama de Bloques del Clasificador

Se puede observar en la figura 34 el diagrama de bloques que tiene el programa.

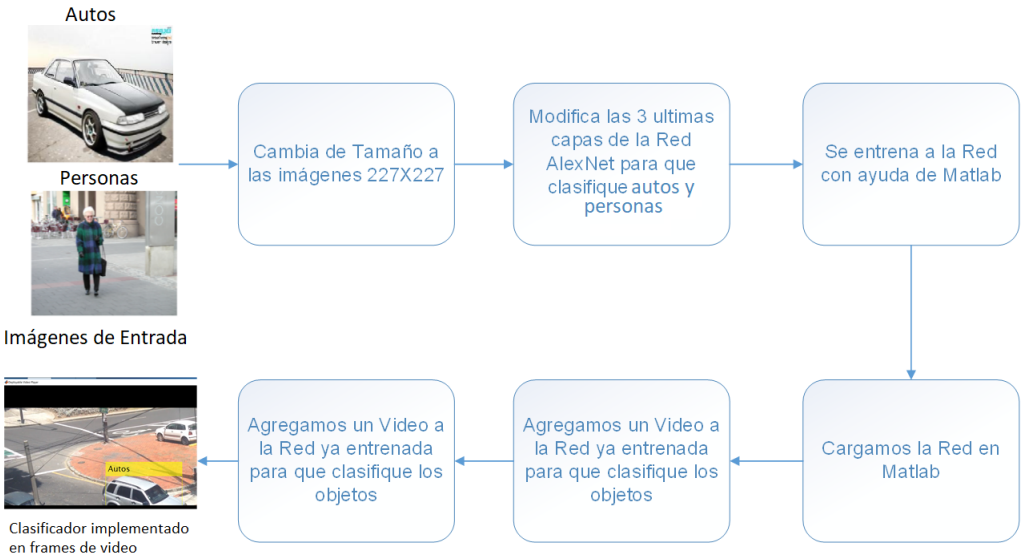


Figura 34. Diagrama de Bloques del Clasificador

4.3 Criterios de Diseño

Para este trabajo se definirán solamente 2 categorías: personas y autos, por lo que se utilizó un banco de imágenes de 300 imágenes.

Antes de iniciar el entrenamiento debe verificarse que el GPU tenga Capacidad computacional mayor a 3 debido a que al momento de entrenar la red neuronal se requiere mayor acceso de memoria y mayor procesamiento ya los modelos inferiores a 3 no poseen tales características.

4.3.1 Comprobación de la capacidad del GPU

Para la comprobación de la capacidad del GPU, se debe ingresar a MATLAB. En la ventana de comandos se escribe *gpuDevice* y desplegará la siguiente información que se observa en la figura 35:

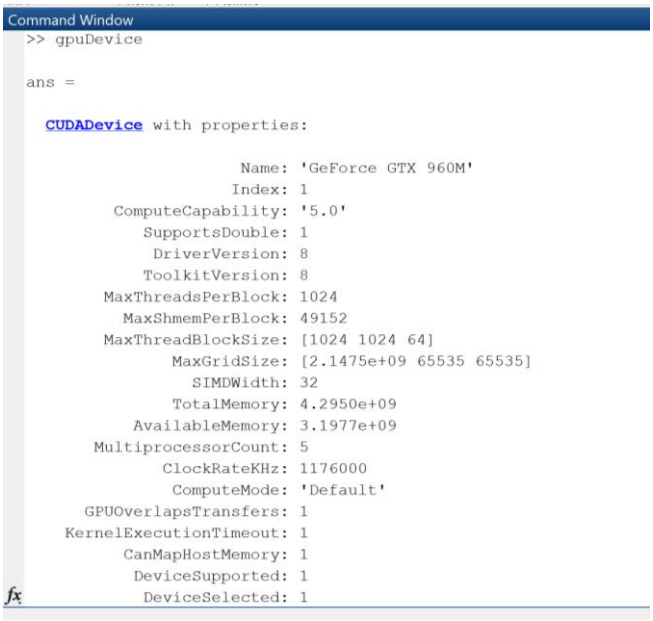


Figura 35. Características de la GPU

Como se observa en la figura 35 en la parte de *ComputeCapability* el GPU que se utiliza en este proyecto es de 5.0.

4.3.2 Comprobación de la instalación del AlexNet *ToolBox*

En la sección 3 ya se mostró como instalar este *ToolBox*. Para su comprobación de que esté funcionando correctamente ir a la pestaña Home, ingresar a la subpestaña *Add-Ons*, e ingresar a la ventana *Get Add-Ons*. En esta ventana aparecerán todos los *Add-Ons* que se tengan instalados y funcionando correctamente como se observa en la figura 36.

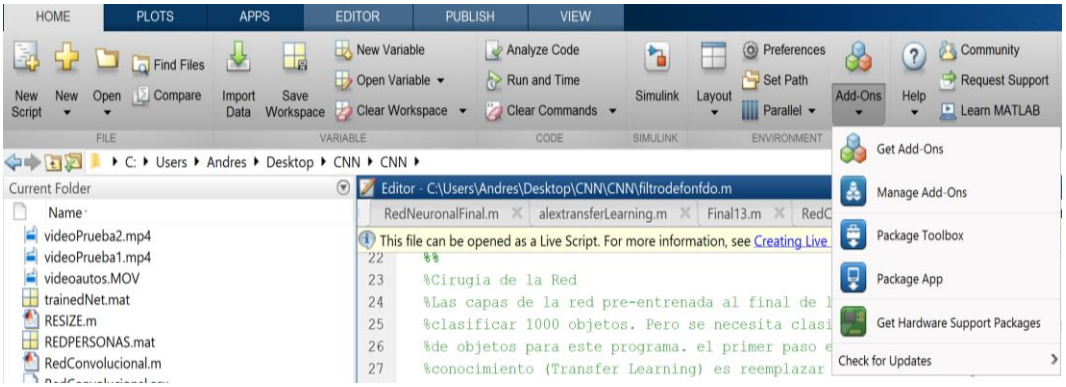


Figura 36. Ingreso a la pestaña Add-Ons

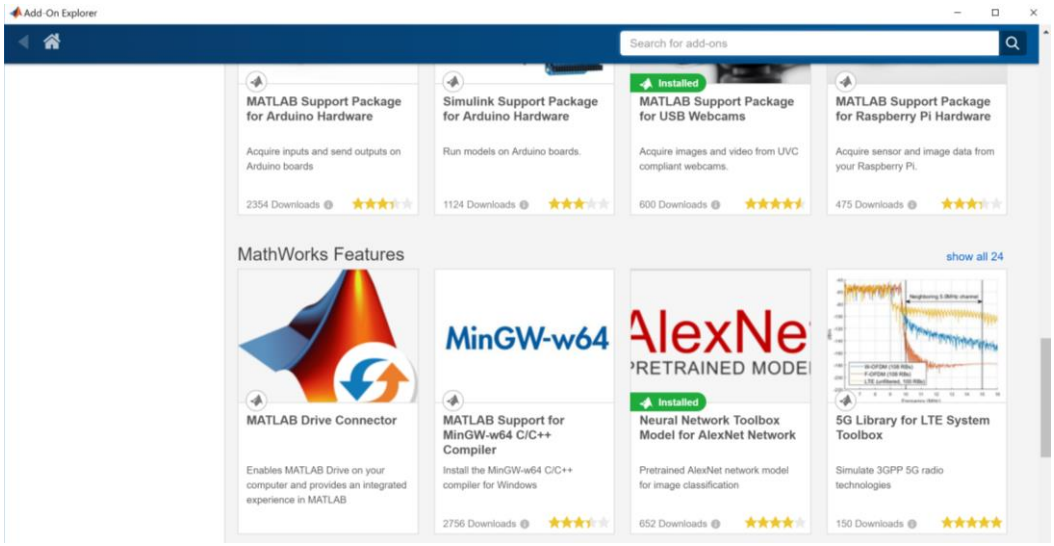


Figura 37. Comprobación de instalación correcta de Alexnet ToolBox

Donde aparece la imagen de Alexnet debe aparecer un ícono verde que diga *Installed*, de esta manera se comprueba que está instalado y funcionando de manera correcta como se observa en la figura 37.

4.4 Creación del Script de la Red en MATLAB

Para comenzar a desarrollar la red neuronal se ingresa a MATLAB. Una vez ahí se crea una nueva función para descargar la red previamente entrenada. Que se describe en la sección 4.4.1

4.4.1 Descarga de la Red Neuronal AlexNet

Para realizar esta descarga desde la página de MATLAB debe utilizarse el siguiente código:

```
function downloadAndPrepareCNN()
% Download AlexNet CNN in MatConvNet format and
converts to SerialNetwork

addpath(fullfile(pwd, 'WebcamClassification'));
mkdir(fullfile(pwd,'networks'));
cnnMatFile = fullfile(pwd, 'networks', 'imagenet-
cnn.mat');
synsetsFile = fullfile(pwd, 'networks',
'synsetMap.mat');

if ~exist(cnnMatFile,'file')

    cnnSourceMatFile  = fullfile(pwd, 'networks',
   'imagenet-caffe-alex.mat');
    cnnURL =
   'http://www.vlfeat.org/matconvnet/models/beta16/imag
   enet-caffe-alex.mat';

   if ~exist(cnnSourceMatFile,'file') % download only
   once
   disp('Downloading 233MB AlexNet pre-trained CNN
   model. This may take several minutes...');
   websave(cnnSourceMatFile, cnnURL);
   end
    convnet = helperImportMatConvNet(cnnSourceMatFile);

    save(cnnMatFile, 'convnet');
    delete(cnnSourceMatFile)
end

if ~exist(synsetsFile,'file')
    synsets2words(convnet);
end
```

Dicho código se obtuvo de la página de MATLAB[7]

[7] https://www.mathworks.com/matlabcentral/fileexchange/58030-example-files-for--deep-learning-for-computer-vision-with-matlab--webinar?focused=7522503&tab=function

Lo que hace este script es simplemente descargar la Red Neuronal Convolucional AlexNet desde la página web de MATLAB. Ejecutar el código y la descarga comenzará de inmediato.

Una vez descargada la red ir a la carpeta donde se realizó la descarga. Y se obtiene el archivo imagenet-cnn.mat como se observa en la figura 38.

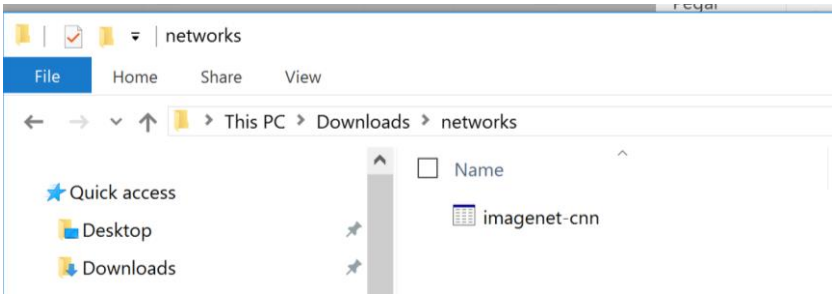


Figura 38. Obtención de la Red Neuronal convolucional

Se comprobó que la red se descargó satisfactoriamente, el nombre del archivo es imagenet-cnn.

Este archivo se lo debe copiar en una carpeta nueva, en la misma que se va a guardar el programa para re entrenar a la red en MATLAB.

4.4.2 Creación de nuevo archivo en MATLAB para entrenamiento de la Red

Ingresar nuevamente a MATLAB. En la parte de Editor se creará un nuevo script. Lo primero que se debe hacer es extraer la red previamente descargada, cargarla a MATLAB y guardarla en una variable. Se lo realiza de la siguiente manera:

Crear un script de nombre RedConvolucional.m con el siguiente código:

```
cnnMatFile=fullfile(pwd,'imagenet-cnn.mat');
imagenet_cnn=load(cnnMatFile);
net=imagenet_cnn.convnet;
```

Con el comando `fullfile` se crea el archivo de la carpeta especificada y con el comando `pwd` se va a extraer el archivo imagenet-cnn.mat. Verificar que el archivo se encuentre en la carpeta donde se está guardando el programa. Para ello vamos a la parte izquierda del editor. Como se observa en la figura 39.

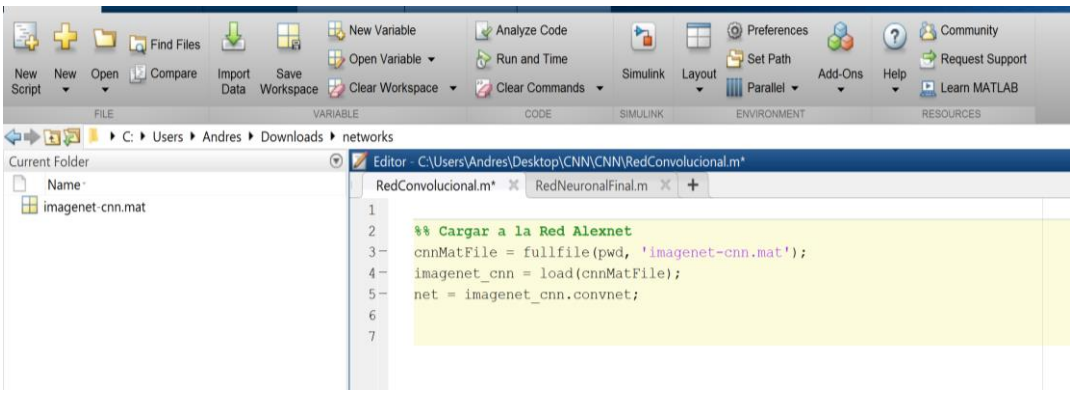
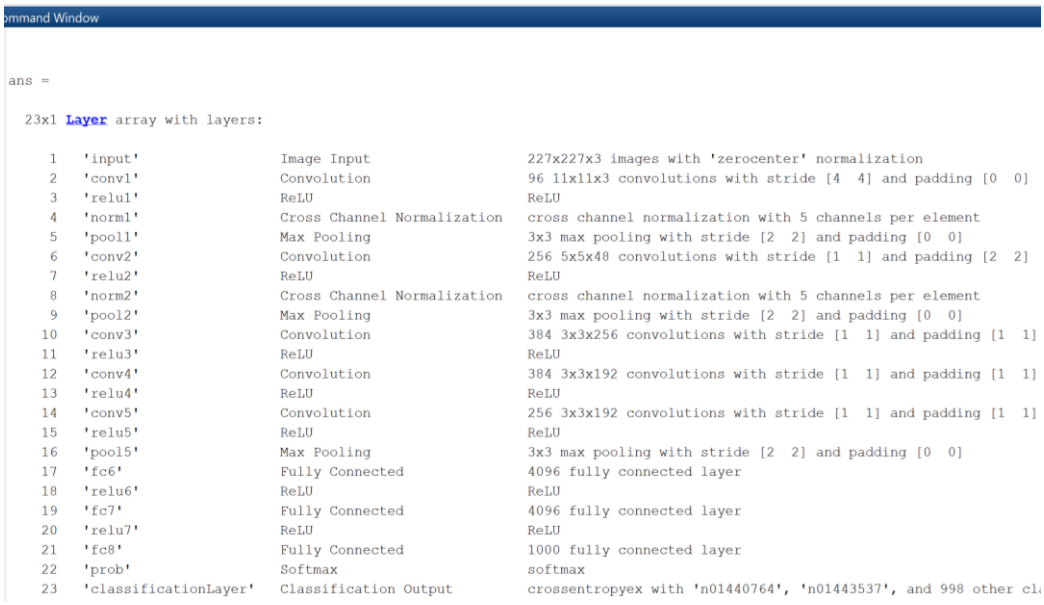


Figura 39. Verificar que el archivo este en la misma carpeta que el programa

Una vez que se extrae el archivo de la carpeta con ayuda del comando `load` se carga a la red en una variable llamada `imagenet_cnn`, y a su vez almacenar en una variable `net`, que va a tener de extensión `convnet`.

4.4.2.1 Verificar las capas de la red

Para verificar las capas de la red únicamente se utiliza el comando `Nombredelared.Layers` que en este caso el nombre de la red es `net`, y se obtiene los parámetros que se muestran en la figura 40.



```
ommand Window

ans =

  23x1 Layer array with layers:

     1   'input'                 Image Input                 227x227x3 images with 'zerocenter' normalization
     2   'conv1'                 Convolution                 96 11x11x3 convolutions with stride [4  4] and padding [0  0]
     3   'relu1'                 ReLU                        ReLU
     4   'norm1'                 Cross Channel Normalization cross channel normalization with 5 channels per element
     5   'pool1'                 Max Pooling                 3x3 max pooling with stride [2  2] and padding [0  0]
     6   'conv2'                 Convolution                 256 5x5x48 convolutions with stride [1  1] and padding [2  2]
     7   'relu2'                 ReLU                        ReLU
     8   'norm2'                 Cross Channel Normalization cross channel normalization with 5 channels per element
     9   'pool2'                 Max Pooling                 3x3 max pooling with stride [2  2] and padding [0  0]
    10   'conv3'                 Convolution                 384 3x3x256 convolutions with stride [1  1] and padding [1  1]
    11   'relu3'                 ReLU                        ReLU
    12   'conv4'                 Convolution                 384 3x3x192 convolutions with stride [1  1] and padding [1  1]
    13   'relu4'                 ReLU                        ReLU
    14   'conv5'                 Convolution                 256 3x3x192 convolutions with stride [1  1] and padding [1  1]
    15   'relu5'                 ReLU                        ReLU
    16   'pool5'                 Max Pooling                 3x3 max pooling with stride [2  2] and padding [0  0]
    17   'fc6'                   Fully Connected             4096 fully connected layer
    18   'relu6'                 ReLU                        ReLU
    19   'fc7'                   Fully Connected             4096 fully connected layer
    20   'relu7'                 ReLU                        ReLU
    21   'fc8'                   Fully Connected             1000 fully connected layer
    22   'prob'                  Softmax                     softmax
    23   'classificationLayer'   Classification Output       crossentropyex with 'n01440764', 'n01443537', and 998 other cl
```

Figura 40. Capas de la Red Neuronal Descargada

Como se observa en la figura 40, AlexNet es una red pre-entrenada de 23 capas. En la capa 21 que es la capa completamente conectada se ve en la descripción que dice: *1000 fully connected layer* lo que significa que el clasificador realiza la clasificación de 1000 categorías de imágenes. Para el caso de este trabajo únicamente se requiere realizar la clasificación de 2 categorías, por lo que se tiene que modificar las tres últimas capas que se explicarán en la siguiente sección.

4.4.2.2 Modificar las capas

La red descargada tiene la capacidad de clasificar 1000 categorías, pero al momento de hacer la clasificación en frames de video la red no tiene la capacidad. Por lo que se va a modificar a la Red para que clasifique únicamente dos categorías.

Las capas definen la arquitectura de la red y contienen los pesos previamente aprendidos. Por tal motivo no se modificarán las capas iniciales

sino únicamente las tres últimas capas que son las de clasificación. Para esto se utiliza el siguiente comando:

```
layers=net.Layers(1:end-3);
```

A las capas iniciales se las almacena en la variable `layers` a excepción de las 3 últimas.

Se modifican las 3 últimas capas nuevamente para que únicamente realice la clasificación de 2 categorías de objetos.

```
layers(end+1) = fullyConnectedLayer(2, 'Name','fc8_2')
layers(end+1) = softmaxLayer('Name','prob_2');
layers(end+1) =
classificationLayer('Name','classificationLayer_2')
```

En esta sección de código se crea las 3 últimas capas: `Fully connected Layer, Softmax Layer y Classification Layer` pero modificando los parámetros para que clasifique solo dos categorías de objetos; autos y personas.

En la entrada de la red se tiene un conjunto de imágenes aproximadamente de 300 por cada categoría y con estas se entrena a la red. Pero debido a que no son muchas imágenes se debe aumentar el número de las mismas. Ya que al momento de entrenar la red el tiempo de entrenamiento durará dependiendo de la cantidad de imágenes, con esta cantidad lo que se realizó es un aumento de las mismas con la ayuda del comando `DataAugmentation` y `randcrop` y antes de eso se cambia el tamaño de las imágenes a 227x227 que es el tamaño de las imágenes de AlexNet.

```
layers(1) = imageInputLayer([227 227
 3],'DataAugmentation', 'randcrop');
```

Como se mencionó anteriormente los pesos de las capas inferiores no van a cambiar caso contrario se deberían modificar todas las capas. Con esta ventaja se puede cambiar a gusto del usuario los pesos de las capas superiores para su mayor aprendizaje. De la siguiente manera:

```
layers(end-2).WeightLearnRateFactor = 100;
layers(end-2).WeightL2Factor = 1;
layers(end-2).BiasLearnRateFactor = 20;
layers(end-2).BiasL2Factor = 0;
```

4.4.2.3 Crear una base de datos para leer las imágenes

Una vez que se obtienen los datos que van a ser entrenados en este caso las imágenes, se van a almacenar en una base de datos, con los siguientes comandos:

```
location = 'CNN';
imds =
imageDatastore(location,'IncludeSubfolders',1,'
LabelSource','foldernames');
tbl = countEachLabel(imds);
```

Con esto las imágenes que se encuentran en las subcarpetas Autos y Personas van a ser almacenados en una base de datos para imágenes.

4.4.2.4 Mostrar las imágenes almacenadas

Para comprobar si las imágenes se guardaron correctamente se utiliza un muestreo al azar de 10 imágenes de cada carpeta con el siguiente código:

```
imagesInMontage = cell(10,2);
carFiles = imds.Files(imds.Labels == 'Cars');
suvFiles = imds.Files(imds.Labels == 'SUV');
imagesInMontage(:,1) =
carFiles(1:length(carFiles)/10:length(carFiles)
);
```

```
imagesInMontage(:,2) =
suvFiles(1:length(suvFiles)/10:length(suvFiles)
);
figure;
subplot(1,2,1);
helperDisplayImageMontage({imagesInMontage{:,1}
});title('Cars');
subplot(1,2,2);
helperDisplayImageMontage({imagesInMontage{:,2}
});title('SUV');
```

En la figura 41 se muestran las 10 imágenes de cada categoría:

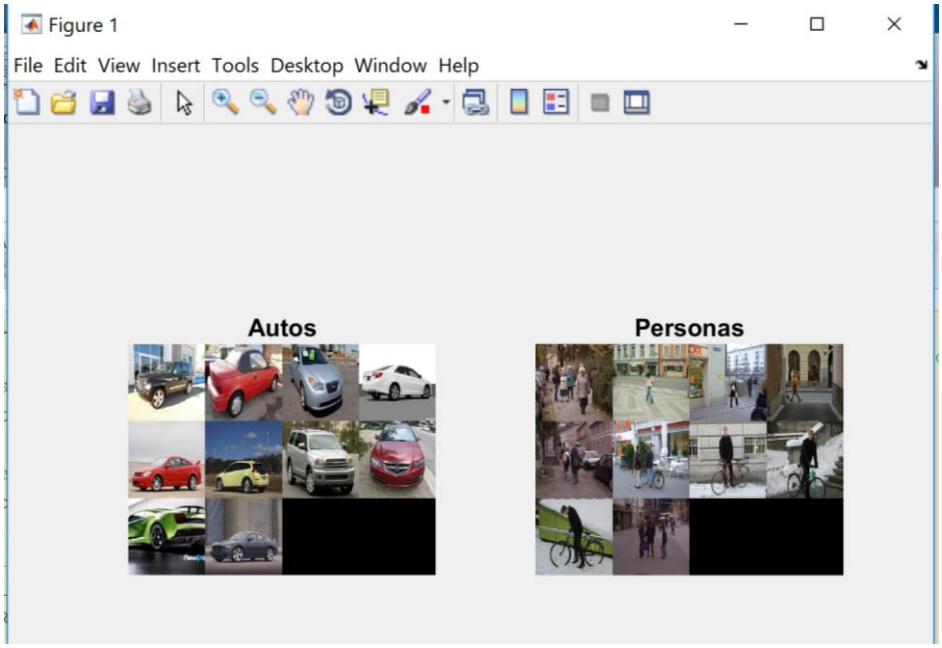


Figura 41. Muestra de las imágenes

4.4.2.5 Igualar el número de imágenes de cada clase

Si no se tiene la misma cantidad de imágenes en cada carpeta se realiza una función para que tengan el mismo número de imágenes en este caso tomará la carpeta con el menor número y a la que contenga más se la iguala con esta. Dicho procedimiento es únicamente necesario si no se tiene la misma cantidad de imágenes en cada carpeta.

```
%% Iguala el número de imagenes de cada clase en el set
de entrenamiento
minSetCount = min(tbl{:,2}); %determina la menor cantidad
de imagenes en una categoria
%El metodo splitEachLabel se usa para cortar el set.
imds = splitEachLabel(imds,minSetCount);
%Cada set de imagenes tiene el mismo numero
countEachLabel(imds)
[trainingDS, testDS] = splitEachLabel(imds,
0.7,'randomize');
%Convierte las leyendas a categorias
trainingDS.Labels = categorical(trainingDS.Labels);
trainingDS.ReadFcn = @readFunctionTrain;
```

El comando `min` se utilizó para determinar el mínimo número de imágenes que existe en cada carpeta. A continuación, con el comando `splitEachLabel` se corta el set de imágenes. Para contar las imágenes se utiliza el comando `countEachLabel`. Después se tomarán imágenes al azar de la carpeta que contenga mayor número de las mismas y se igualará a la de menor imágenes.

Con la ayuda de el comando `categorical` se transforma las leyendas de las carpetas en categorías.

4.4.2.6 Configuración de parámetros para el entrenamiento de la Red

Una vez realizados todos los procesos anteriores la red está lista para ser entrenada, para ello se debe setear algunos parámetros como el *learning rate*, épocas, número máximo de épocas, *BatchSize,* número de iteraciones. Todos estos parámetros se los configura de acuerdo al rendimiento de nuestra GPU. Por ejemplo, si se aumentó el número de épocas y de iteraciones el entrenamiento va a durar mucho más debido a que la red se va a entrenar de manera más lenta, pero va a ser mucho más efectiva.

Para esto se utilizó el siguiente script de MATLAB:

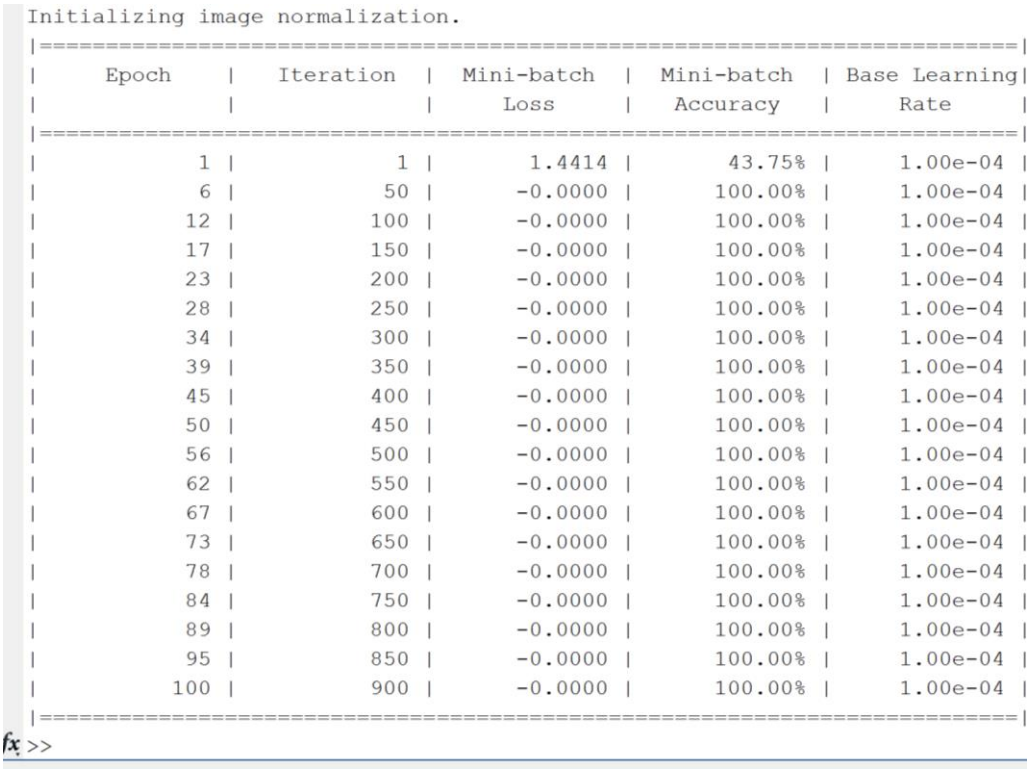
```
miniBatchSize = 32; %Se puede reducir este numero si la
GPU se queda sin memoria
numImages = nume1(trainingDS.Files);

%Se corre el entrenamiento para 5000 iteraciones.
Convierte 20000
%iteraciones en el numero de epocas que se usaran.
numIterationsPerEpoch = numImages/miniBatchSize;
%maxEpochs = round(20000/numIterationsPerEpoch);
maxEpochs = 100;
lr=0.001;
opts = trainingOptions('sgdm', ...
    'InitialLearnRate', lr,...
    'LearnRateSchedule', 'none',...
    'L2Regularization', 0.0005, ...
    'MaxEpochs', maxEpochs, ...
    'MiniBatchSize', miniBatchSize);

net= trainNetwork(trainingDS, layers, opts);
%save('trainedNet.mat','net')
```

Una vez configurados todos estos parámetros utilizamos el comando trainNetwork para comenzar el entrenamiento. Depende de los parámetros que se hayan colocado y el número de imágenes que se tengan en la entrada variará el tiempo de entrenamiento. Los valores de entrenamiento que se utilizó para el proyecto fueron: `miniBatchSize = 32, maxEpochs = 100; lr=0.001;` que son los valores por defecto para el entrenamiento. Si la GPU se queda sin memoria se puede modificar el parámetro `miniBatchSize` y poner valores menores a 32 que es el valor por defecto de este parámetro.

En el caso de este trabajo el entrenamiento tuvo un tiempo aproximado de 1 hora. Teniendo así los siguientes resultados que se muestran en la figura 42:



```
Initializing image normalization.
|=========================================================================|
|     Epoch    |   Iteration  |  Mini-batch  |  Mini-batch  | Base Learning|
|              |              |     Loss     |   Accuracy   |     Rate     |
|=========================================================================|
|            1 |            1 |       1.4414 |       43.75% |     1.00e-04 |
|            6 |           50 |      -0.0000 |      100.00% |     1.00e-04 |
|           12 |          100 |      -0.0000 |      100.00% |     1.00e-04 |
|           17 |          150 |      -0.0000 |      100.00% |     1.00e-04 |
|           23 |          200 |      -0.0000 |      100.00% |     1.00e-04 |
|           28 |          250 |      -0.0000 |      100.00% |     1.00e-04 |
|           34 |          300 |      -0.0000 |      100.00% |     1.00e-04 |
|           39 |          350 |      -0.0000 |      100.00% |     1.00e-04 |
|           45 |          400 |      -0.0000 |      100.00% |     1.00e-04 |
|           50 |          450 |      -0.0000 |      100.00% |     1.00e-04 |
|           56 |          500 |      -0.0000 |      100.00% |     1.00e-04 |
|           62 |          550 |      -0.0000 |      100.00% |     1.00e-04 |
|           67 |          600 |      -0.0000 |      100.00% |     1.00e-04 |
|           73 |          650 |      -0.0000 |      100.00% |     1.00e-04 |
|           78 |          700 |      -0.0000 |      100.00% |     1.00e-04 |
|           84 |          750 |      -0.0000 |      100.00% |     1.00e-04 |
|           89 |          800 |      -0.0000 |      100.00% |     1.00e-04 |
|           95 |          850 |      -0.0000 |      100.00% |     1.00e-04 |
|          100 |          900 |      -0.0000 |      100.00% |     1.00e-04 |
|=========================================================================|
fx >>
```

Figura 42. Obtención de Resultados del Entrenamiento de la red.

Como se observa en la figura 42 en la columna de `Epoch` comienza desde la primera época y va avanzando de 6 en 6 hasta llegar al 100. En la columna de `Iteration` comienza desde 1 y va avanzando de 50 en 50 hasta llegar al 900.

4.4.3 Cargar la Red entrenada y prueba de efectividad

Una vez cargada la red entrenada se debe cargar nuevamente con el comando load y el nombre con el que se guardó. Para probar la efectividad se lo hace con ayuda de la siguiente función:

```
labels = classify(net, testDS, 'MiniBatchSize', 32);
confMat = confusionmat(testDS.Labels, labels);
```

```
confMat = bsxfun(@rdivide,confMat,sum(confMat,2));
mean(diag(confMat))
```

Se ejecutó el código y se obtuvo lo siguiente:

```
ans =
     1
fx >>
```

Figura 43. Efectividad de la Red

Como se observa en la figura 43 se obtuvo resultado de 1 lo que significa que la efectividad de clasificación de la Red es del 100%. Si se hubiese obtenido valores menores a 1 la red hubiese tenido un menor porcentaje de efectividad.

4.4.4 Cargar a la red los *frames* de video

Para realizar la carga del video usamos la función `visión.DeployableVideoPlayer` y la almacenamos en una variable. Debido a que se va a usar un video donde el fondo este estable se necesita poner una función para que aprenda el fondo del video y luego sustraerlas de los *frames* (cuadros) de modo que se pueda encontrar la región correspondiente a los vehículos o personas en este caso. Se hizo una función para que el video aprenda el fondo de acuerda a una región de interés (*Region of Interest* (ROI)) y se creó objetos de sistemas para los objetos largos binarios (*Binary Large Object* (BLOB)), los *Blobs* son imágenes en este caso que se utilizan para almacenar datos de gran tamaño y cambian de forma dinámicamente.

```
%% Carga del video
videoPlayer = vision.DeployableVideoPlayer;

% En esta seccion la red aprende el entorno(background)
foregroundDetector =
vision.ForegroundDetector('NumGaussians', 5, ...
    'NumTrainingFrames',
  150,'MinimumBackgroundRatio',0.7);
roi = [1 1 1080 1900];
    videoReader =
  vision.VideoFileReader('atropello.mov');
for i = 1:150
    frame = step(videoReader); % lee el siguiente frame
    frame = imcrop(frame,roi);
    frameSmall = imresize(frame,0.25);
    foreground = step(foregroundDetector, frameSmall);
end
     blobAnalysis =
   vision.BlobAnalysis('BoundingBoxOutputPort', true,
   ...
    'AreaOutputPort', false, 'CentroidOutputPort',
  false, ...
    'MinimumBlobArea', 300);
```

La función `ForegroundDetector` lo que hace es detectar el primer plano. Con la ayuda de la función `vision.ForegrounDetector` la variable en la que se almacenó en este caso `foregroundDetector` nos va a devolver un sistema de detección de objetos. Para esto se necesita setear tres valores:

- `NumGaussians`: Que son el número de distribuciones que hacen el modelo de detección de fondo. En este caso el número por defecto es
- `NumTrainingFrames`: Son el número de *frames* de video iniciales usadas para entrenar el modelo de fondo. El valor por defecto son 150 tramas.
- `MinimumBackgroundRatio`: Es el valor umbral que se utiliza para determinar los modos gaussianos que constituyen el proceso de fondo. El valor por defecto es 0.7.

4.4.5 Detección de autos y personas

Para realizar la función que detectará carros y personas, se crea un bucle `while` para que de esta manera la red detecté y clasifique las imágenes de manera automática mientras avancen los *frames* de video.

```
%% Detecta carros y personas usando la substracción del
fondo y los clasifica
se = strel('square', 2);
while ~isDone(videoReader)

    frame = step(videoReader);
    frame = imcrop(frame,roi);
    frameSmall = imresize(frame,0.25);
    foreground = step(foregroundDetector, frameSmall);

    filteredForeground = imopen(foreground, se);

   bbox = step(blobAnalysis, filteredForeground);

    numCars = size(bbox,1);
    bbox = 4.*bbox;
    result = frame;
    if numCars > 0
        for j=1:numCars


   im = imcrop(frame,[bbox(j,1)-10 bbox(j,2)-10
   bbox(j,3)+30 bbox(j,4)+30]);

     im = single(im2uint8(im));
     im = imresize(im, [227 227]) ;
     label = char(classify(net,im));
     result =
     insertObjectAnnotation(result,'Rectangle',bbox(j,:
     ),label,'FontSize',32);
        end

    end
    step(videoPlayer,result);


end
```

En una variable en este caso se llama `se`, se filtra la imagen con un *strel* cuadrado. Un *strel* es un elemento de estructura morfológica, en el código se utilizó un *strel* cuadrado que servirá para filtrar el fondo de los *frames* de video.

Se creó el ciclo `while`, lo que hace es analizar *frame* por frame, para ver si existen objetos en movimiento. A estos objetos la red los va a clasificar. Para poder distinguirlos se crea la función `insertObjectAnnotation` que servirá para crear un recuadro alrededor de los mismos con la etiqueta que corresponda a cada clase.

Finalmente, para que los resultados se desplieguen en el video se utilizará la función `step`. La función `release` servirá para detener al video una vez que se termine el mismo.

Se obtuvo el siguiente conjunto de *frames* de video que como se puede observar en la figura 44 la red está clasificando de manera correcta.

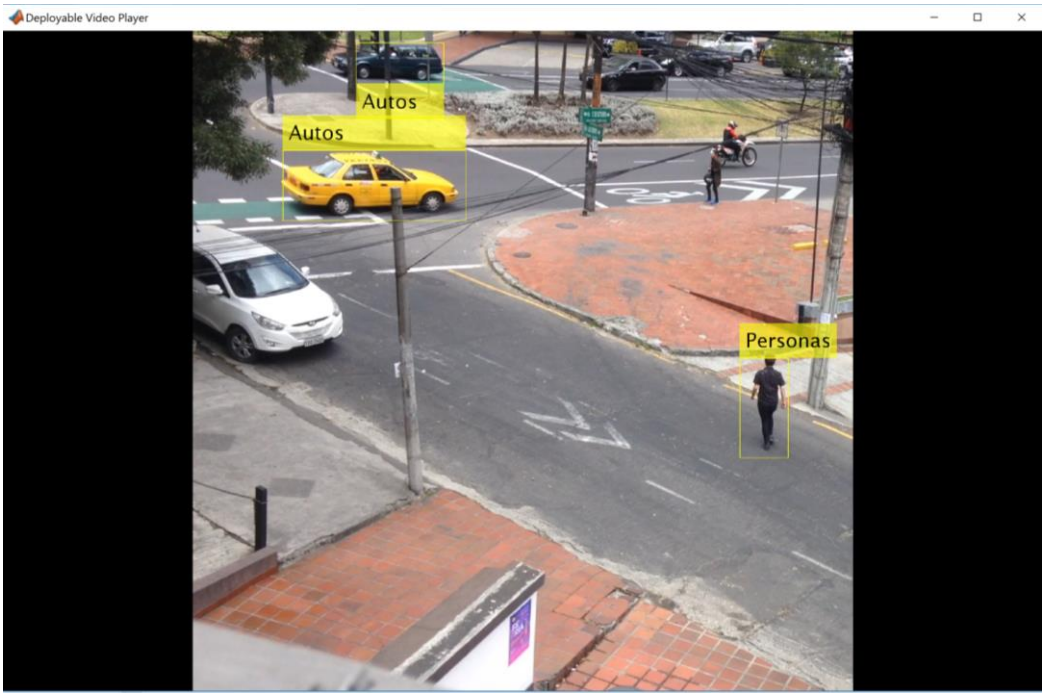


Figura 44. Clasificación de las imágenes en un video

CAPITULO 5

PRUEBAS Y ANÁLISIS DE RESULTADOS

En esta sección se muestran los resultados de las pruebas de entrenamiento de la red variando los parámetros y viendo su tiempo de respuesta y su confiabilidad. Para verificar el rendimiento que se obtuvo con la GPU, se realizó además el entrenamiento de la red con ayuda del CPU, de esta manera se logró verificar y ver comparaciones entre ambas. Para la prueba 1 se va a entrenar a la red con 300 imágenes de entrada para cada categoría (autos y personas) primero sin el uso de la GPU y después con la GPU para comparar los tiempos de entrenamiento. Para la prueba con el clasificador se va a tomar 2 videos con diferente calidad: 1090 X 1080 pixeles, 320 X 240 pixeles. Los mismos que van a tener un fondo fijo es decir que no tenga movimiento en el mismo y dos videos de calidad de 1090 X 1080 pixeles 320 X 240 pixeles pero que su fondo tenga movimiento para de esta manera poder comparar la efectividad del clasificador con respecto a la calidad de video y a cuando el video tenga un fondo con movimiento y ver como varia. Para la prueba 2 se varía los parámetros de entrenamiento, así como también las imágenes de entrada se redujeron de 300 a 200 imágenes por categoría. De la misma manera que en la prueba 1 para probar el clasificador se utilizan los mismos videos. Finalmente, para la prueba 3 se varían los parámetros de entrenamiento de la red y se aumenta las imágenes de entrada de 200 a 400 imágenes por categoría. Al igual que en la prueba 1 y prueba 2, se va a ocupar los mismos videos para realizar la prueba del clasificador.

5.1 Pruebas de entrenamiento de la red

Para poder comparar y verificar el rendimiento de la GPU. Primero se realizó el entrenamiento con procesamiento en paralelo con la GPU, y luego se realizó el mismo entrenamiento, pero ahora únicamente con ayuda del CPU. Los siguientes datos fueron tomados para realizar el entrenamiento de la Red.

5.1.1 Prueba 1

Tabla 4
Parámetros de entrenamiento de la Red – Prueba 1

Parámetro	Valor
miniBatchSize	32
maxEpochs	100
Learning Rate	0.0001
Regularización	0.0005

El primer parámetro es el `miniBatchSize`, que es el número de ejemplos de entrenamiento en un *forward o backward pass*. Mientras mayor sea este número más memoria se ocupará.

Una época (`Epoch`) es una medida del número de veces que los vectores de entrenamiento son usados para actualizar los pesos.

El `learning rate` es un parámetro que controla el tamaño del peso. Es la tasa de aprendizaje con la que aprenderá la red al momento de ser entrenada.

La `regularización` es una técnica que se utiliza por el algoritmo como su nombre lo indica para regularizar el entrenamiento de la red, es decir que se mantenga constante a cada momento.

Estos datos son los que se utilizaron en la explicación del programa logrando así tener los siguientes resultados utilizando el procesamiento en paralelo con la GPU, se muestran en la tabla 5:

Tabla 5
Resultados Prueba 1 con el procesamiento en paralelo

Épocas	Iter.	Tiempo (s)	Mini-batch Loss	Mini-Batch Accuracy (%)	Base Learning Rate
1	1	30	14.414	43,75	0,0001
6	50	124	0	100	0,0001
12	100	320	0	100	0,0001
17	150	400	0	100	0,0001
23	200	720	0	100	0,0001
28	250	910	0	100	0,0001
34	300	1044	0	100	0,0001
39	350	1223	0	100	0,0001
45	400	1402	0	100	0,0001
50	450	1581	0	100	0,0001
56	500	1760	0	100	0,0001
62	550	1940	0	100	0,0001
67	600	2119	0	100	0,0001
78	700	2477	0	100	0,0001
84	750	2656	0	100	0,0001
89	800	2835	0	100	0,0001
95	850	3014	0	100	0,0001
100	900	3193	0	100	0,0001

A continuación, como se observa en la tabla 6, se realizó el entrenamiento de la misma red con los mismos parámetros, pero esta vez no se utilizó el procesamiento en paralelo.

Tabla 6
Resultados Prueba 1 con el procesamiento de la CPU

Épocas	Iter.	Tiempo (s)	Mini-batch Loss	Mini-Batch Accuracy (%)	Base Learning Rate
1	1	57	18,23	60,21	0,0001
6	50	223	0,34	81	0,0001
12	100	578	-0,21	100	0,0001
17	150	678	0,2	100	0,0001
23	200	1021	0,4	54,32	0,0001
28	250	1230	0,01	100	0,0001
34	300	1467	0,5	45,21	0,0001
39	350	1706	0	78,1	0,0001
45	400	1945	0	92,1	0,0001
50	450	2184	0,12	100	0,0001
56	500	2422	0	100	0,0001
62	550	2661	0,31	54,14	0,0001
67	600	2900	0	100	0,0001
73	650	3139	0	100	0,0001
78	700	3378	0	89,68	0,0001
84	750	3617	0,021	100	0,0001
89	800	3855	0	64,54	0,0001
95	850	4094	0,3	100	0,0001
100	900	4333	0	87,54	0,0001

5.1.1.1 Prueba del Clasificador en los *frames* de videos

Con las redes entrenadas tanto con el procesamiento en paralelo como con el CPU, al momento de introducir la red para que analice los *frames* de video se obtuvo resultados muy similares, como se observa en las figuras 45, 46 y 47.

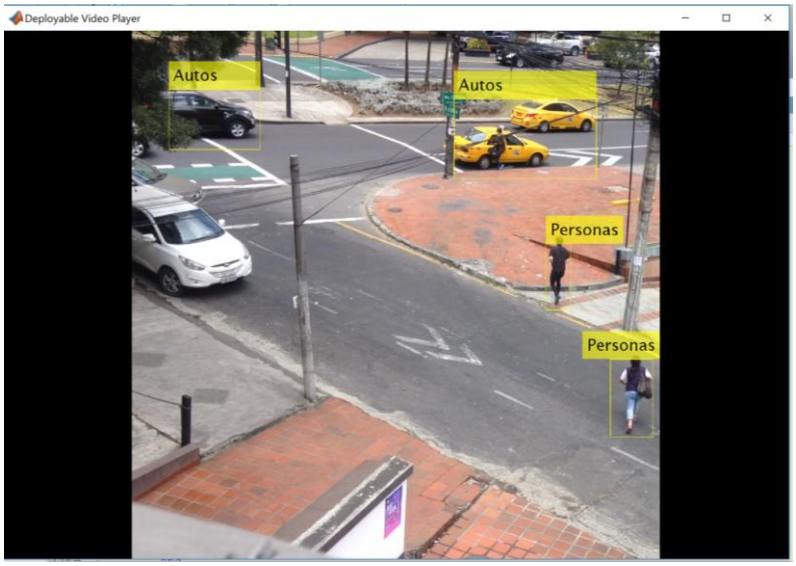


Figura 45. Carga de la Red entrenada en los frames de video para Prueba 1. Video de 1090 x 1080 pixeles.

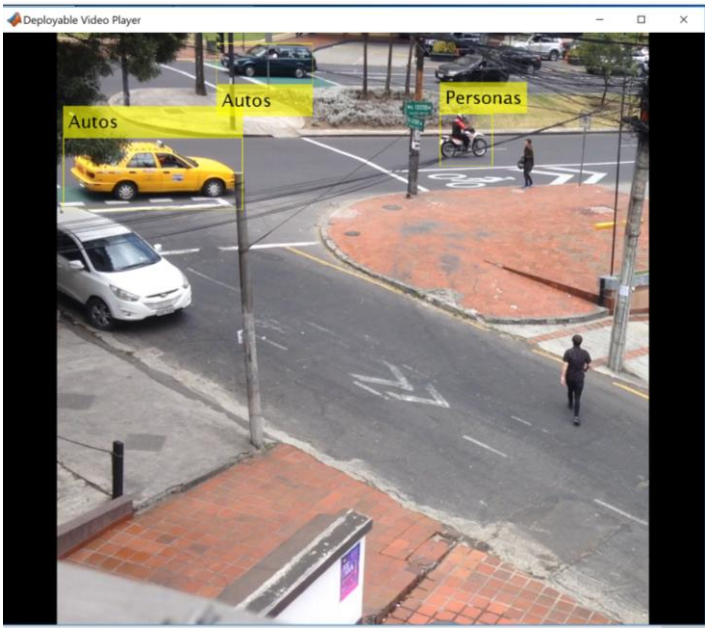


Figura 46. Frames de video para Prueba 1.

Figura 47. Frames de video para Prueba 1. Resultado

La calidad de video que se usó en este video fue de 1090 x 1080 pixeles. Para poder comparar el clasificador con videos de calidad baja se usó un video de 320 x 240 pixeles como se observa en las figuras 48, 49 y 50 y se obtuvo el siguiente resultado:

Figura 48. Carga de la Red entrenada en los frames de video para Prueba 1. Video de 320 x 240 pixeles.

Figura 49. Resultado Prueba 1 en video de baja calidad.

Figura 50. Resultado Prueba 1 video de baja calidad.

Estos dos videos se tomaron ya que el fondo de los mismos se encontraba estables, es decir no tenía movimiento. Las cámaras son fijas y no tienen la posibilidad de hacer un alejamiento o acercamiento al video, ni tampoco hacer un movimiento. La red por lo tanto podía detectar y clasificar los objetos sin mayor inconveniente. Debido a esto se realizó una prueba con dos videos

adicionales. En estos dos videos se tomó en cuenta que la cámara con que fueron grabados sea fija, pero pueda hacer un zoom al video o pueda tener movimiento y que el fondo no sea el mismo. Con esto se obtuvo los siguientes resultados.

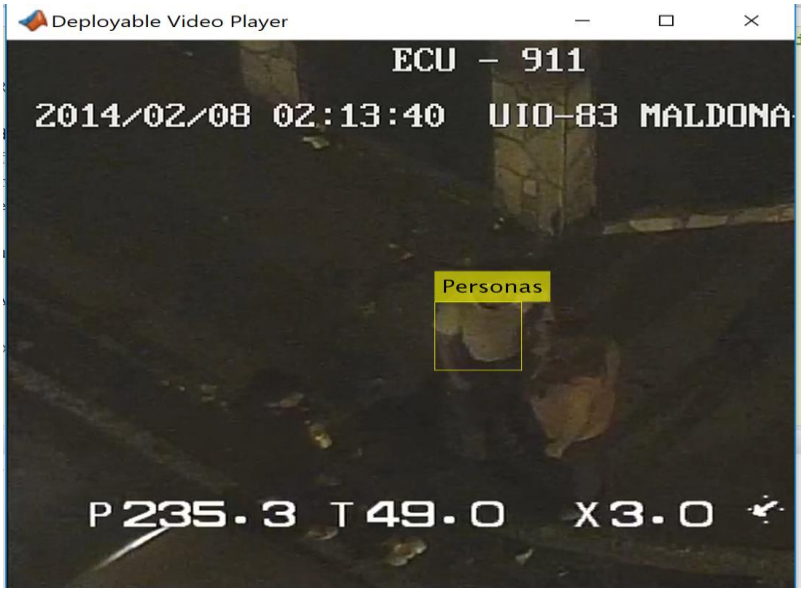


Figura 51. Resultado 1 de prueba 1 con video con fondo en movimiento. Video 1.

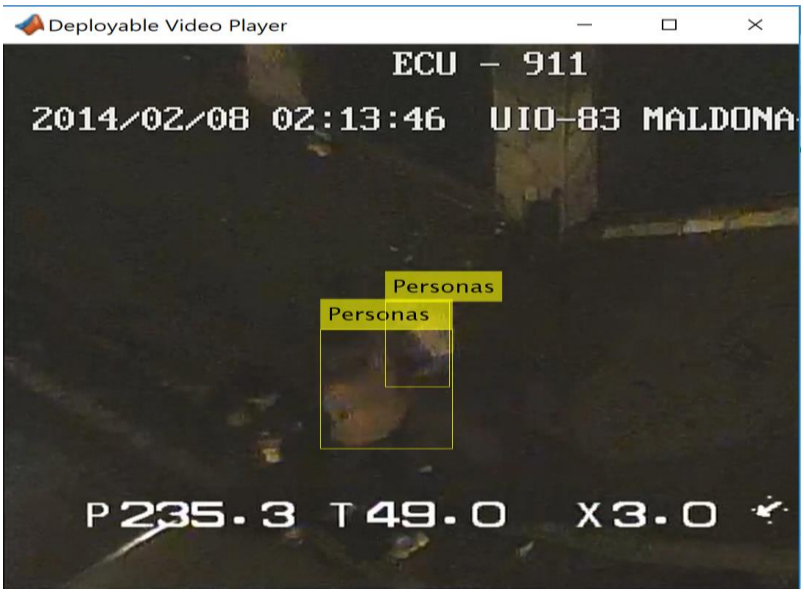


Figura 52. Resultado 2 de prueba 1 con video con fondo en movimiento. Video 1.

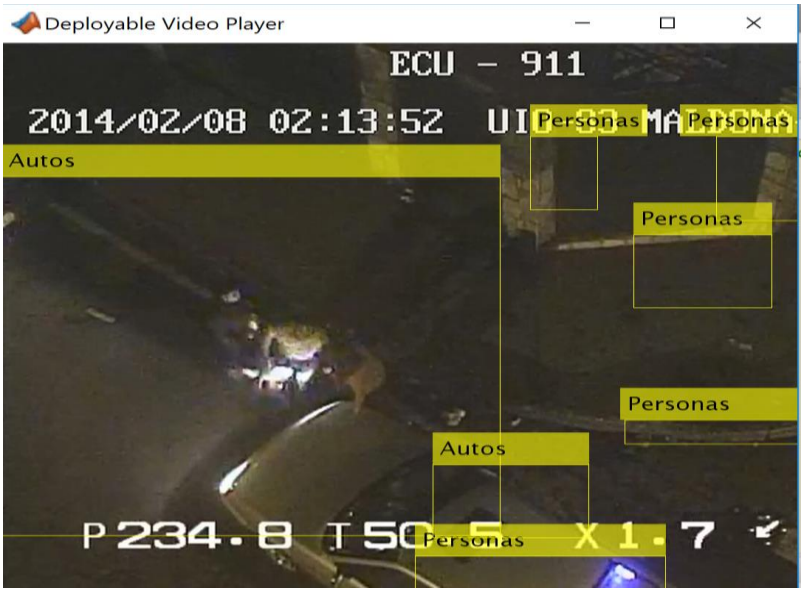


Figura 53. Resultado 3 de prueba 1 con video con fondo en movimiento. Video 1.

Figura 54. Resultado 1 de prueba 1 con video con fondo en movimiento. Video 2.

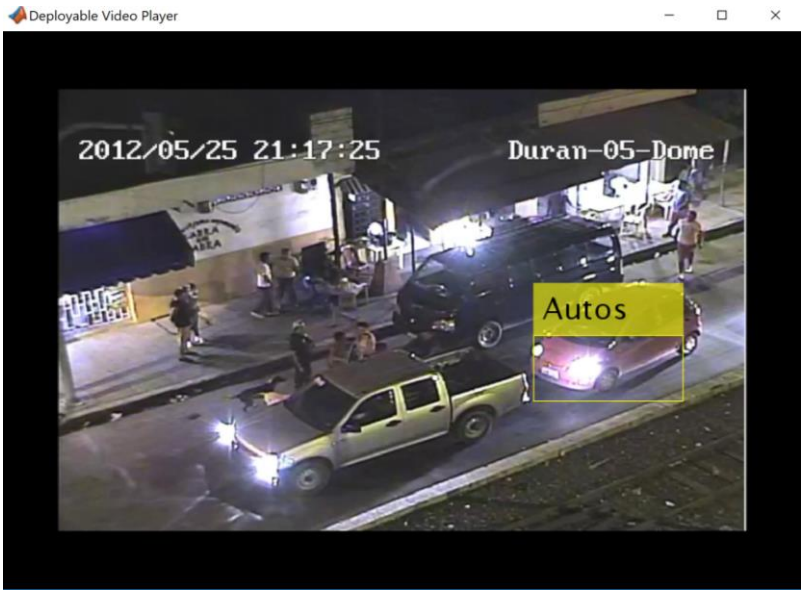


Figura 55. Resultado 2 de prueba 1 con video con fondo en movimiento. Video 2.

5.1.1.2 Análisis de Prueba 1

En la primera prueba realizada con los parámetros antes definidos se observa en la tabla 5 que el tiempo aproximado de entrenamiento de la red fue de 3193 segundos casi una hora en la que se llevó a cabo el entrenamiento de la red. Según los resultados obtenidas se puede llegar a la conclusión que el porcentaje de error que va a tener esta red es nulo, ya que se obtuvo un *Mini-Batch Accuracy* del 100% como se aprecia en la columna de dicho parámetro en la tabla 5. Ahora bien, para el caso en el cual se entrenó a la red únicamente con ayuda del CPU se obtuvo un tiempo aproximado de entrenamiento de 4333 segundos como se observa en la tabla 6, más de una hora. En cuanto al *Mini-Batch Accuracy* se obtuvo un promedio de 87,1% sumando todos los valores de la columna de dicho parámetro y dividiendo para el mismo valor, estos valores se muestran en la tabla 6, lo que quiere decir que la efectividad del clasificador de la red es alta, pero no al 100%. En comparación usando el GPU al momento de entrenar a la red se la hizo en 3193 segundos y usando únicamente el CPU el entrenamiento demoró 4333

segundos por lo que se puede concluir que los tiempos de entrenamiento se reducen notablemente utilizando la GPU.

Respecto a la prueba de las dos diferentes calidades de video se obtuvo un mejor rendimiento en la de más alta calidad en este caso el video de 1090 x 1080 pixeles en ambos casos cuando el video tenía fondo fijo y con movimiento. El clasificador tuvo un porcentaje mucho mayor al momento de clasificar las imágenes. Cuando se probó con el video de 320 x 240 pixeles había momentos en los que el clasificador no distinguía entre personas y autos como se observa en la figura 50. Esto debido a que las imágenes que están en los frames de video no se las ve con claridad.

En la prueba con los videos en el que el fondo no era estático el resultado no fue muy favorable, debido a que el clasificador únicamente funcionaba cuando la cámara se encontraba estática y sin realizar ningún movimiento. Cuando comenzaba a moverse; el clasificador no distinguía entre imágenes y únicamente hacia recuadros a imágenes en los *frames* que tenían algunas características parecidas a las de autos o personas.

5.1.2 Prueba 2

Tabla 7
Parámetros de entrenamiento de la Red – Prueba 2

Parámetro	Valor
miniBatchSize	32
maxEpochs	50
Learning Rate	0.001
Regularización	0.005

Los datos de la tabla 7 se utilizaron para la prueba 2. Además, que el número de imágenes que se usaron para la misma se redujeron de 300 por carpeta a 200 por carpeta. Esto con el fin de verificar si el clasificador va a tener un porcentaje de aciertos menor a la de a prueba anterior.

Tabla 8
Resultados Prueba 2 con el procesamiento en paralelo

Épocas	Iter.	Tiempo (s)	Mini-batch Loss	Mini-Batch Accuracy (%)	Base Learning Rate
1	1	23	0,43	56,23	0,001
6	50	54	0,4	68,29	0,001
12	100	132	0,63	89,43	0,001
17	150	234	-0,012	50,21	0,001
23	200	321	0,3	88,32	0,001
34	300	652	0,2	92,2	0,001
39	350	743	-0,06	85,38	0,001
45	400	902	0,3	100	0,001
50	450	1021	0,09	87,54	0,001

A continuación, se realizó el entrenamiento de la misma red con los mismos parámetros, pero esta vez no se utilizó el procesamiento en paralelo.

Tabla 9
Resultados Prueba 2 con el procesamiento de la CPU

Épocas	Iter.	Tiempo (s)	Mini-batch Loss	Mini-Batch Accuracy (%)	Base Learning Rate
1	1	45	0,5	63,4	0,001
6	50	123	0,76	45,65	0,001
12	100	276	0,04	73,5	0,001
17	150	349	0,05	80,43	0,001
23	200	508	0,2	86,32	0,001
28	250	723	0,67	89,32	0,001
34	300	956	0,03	76,21	0,001
39	350	1102	0,3	83,21	0,001
45	400	1389	0,03	73,46	0,001
50	450	1563	0,13	81,49	0,001

5.1.2.1 Prueba del Clasificador en los *frames* de videos

Para este caso de igual manera que en la prueba 1 se utilizaron dos videos uno de calidad de 1090 x 1080 pixeles y el otro de 320 x 240 pixeles.

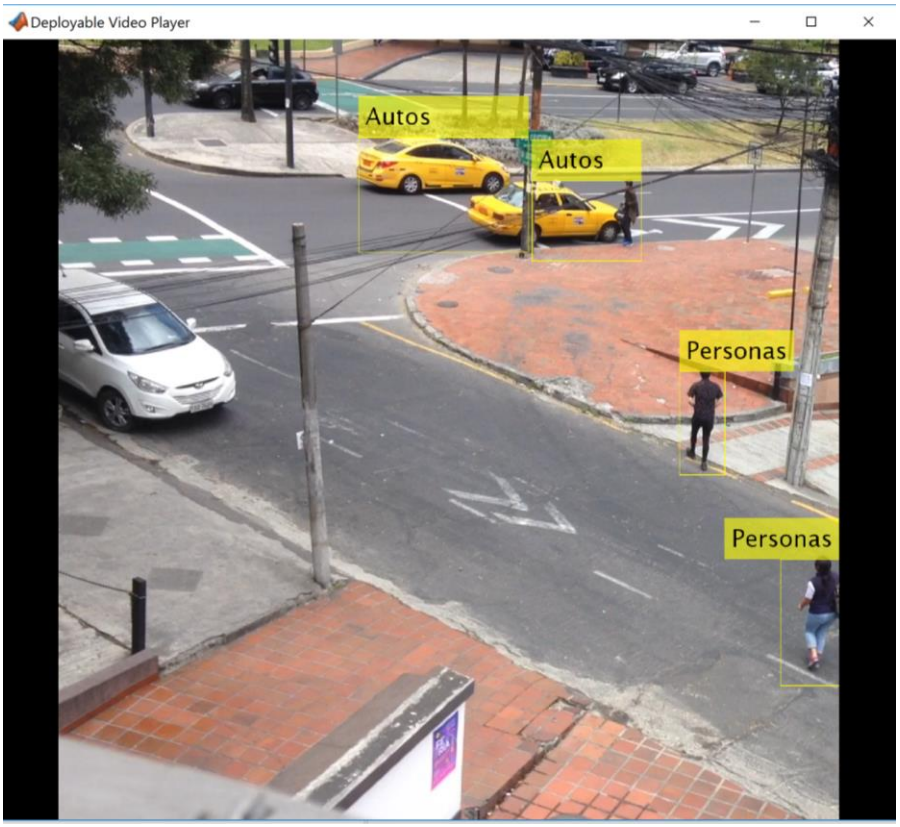


Figura 56. Carga de la Red entrenada en los frames de video para Prueba 2. Video de 1090 x 1080 pixeles.

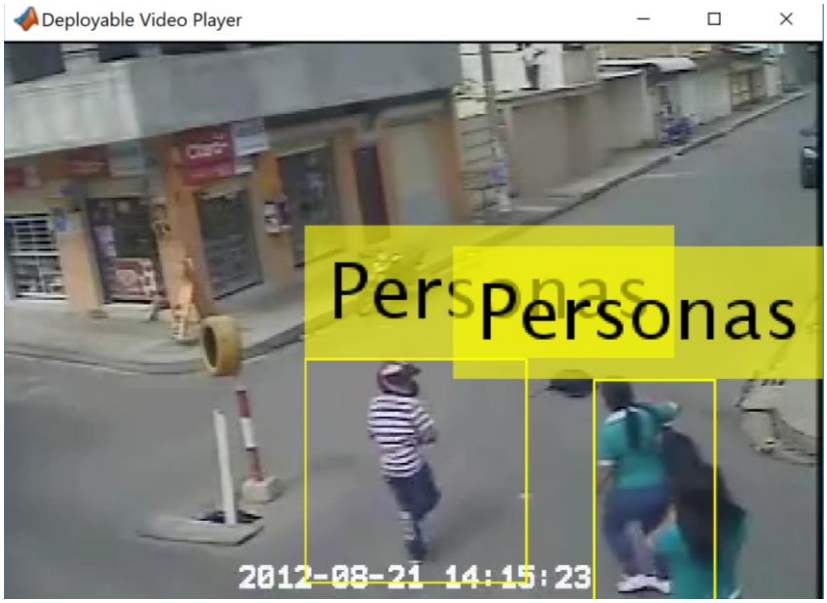


Figura 57. Carga de la Red entrenada en los frames de video para Prueba 2. Video de 320 x 240 pixeles.

5.1.2.2 Análisis de Prueba 2

En esta segunda prueba realizada con los parámetros que se observan en la tabla 7. El tiempo aproximado de entrenamiento de la red fue de 1021 segundos, un tiempo mucho menor al que se obtuvo en la primera prueba para el entrenamiento de la red. Según los resultados obtenidos se puede llegar a la conclusión que el porcentaje de error que va a tener esta red es aproximadamente de 79% de efectividad, para calcular este porcentaje se tomaron los valores de la columna de *Mini-Batch Accuracy* de la tabla 8, se sumaron todos los valores y se dividió para el mismo número. Este porcentaje no fue tan alto con respecto al de la prueba 1 debido a que se disminuyó las imágenes en la entrada y el número de iteraciones. Para el caso en el cual se entrenó a la red únicamente con ayuda del CPU se obtuvo un tiempo de entrenamiento de 1563 segundos, más que cuando se utiliza procesamiento en paralelo. En cuanto al Mini-Batch Accuracy se obtuvo un promedio de 75%, para calcular este porcentaje se tomaron los valores de la columna de *Mini-Batch Accuracy* de la tabla 9, se sumaron todos los valores y se dividió para el mismo número, lo que quiere decir que la efectividad del clasificador de la

red es moderada. Al momento de cargar la red en los *frames* de video, se obtuvo prácticamente los mismos parámetros que cuando se lo realizó en la prueba 1.

En cuanto a la calidad de los videos de la misma manera el clasificador tuvo mayor efectividad con el video de 1090 x 1080 pixeles ya que clasificaba a autos y personas sin problema alguno. Esto debido a la claridad de las imágenes.

De la misma manera que en la prueba 1, se tomaron los mismos videos los cuales no tenían un fondo estático, dando así exactamente los mismos resultados al momento de usar el clasificador. No clasificaba en ciertas ocasiones cuando el video tenía movimiento en el fondo.

5.1.3 Prueba 3

En esta última prueba se utilizó los valores que se muestran en la tabla 10.

Tabla 10
Parámetros de entrenamiento de la Red – Prueba 3

Parámetro	Valor
miniBatchSize	60
maxEpochs	150
Learning Rate	0.0001
Regularización	0.005

El número de imágenes que se usaron para la misma se aumentaron de 200 a 400 imágenes por cada categoría. Para de esta manera verificar el aumento del porcentaje de clasificación de la Red. Al aumentar el número de

imágenes para aumentar la red y aumentar el número de épocas el número de iteraciones que tendrá la red al momento del entrenamiento.

Tabla 11
Resultados Prueba 3 con el procesamiento en paralelo

Épocas	Iter.	Tiempo (s)	Mini-batch Loss	Mini-Batch Accuracy (%)	Base Learning Rate
1	1	45	1	74,21	0,0001
6	50	142	0	100	0,0001
12	100	354	0	100	0,0001
17	150	453	0	100	0,0001
23	200	731	0	100	0,0001
28	250	940	0	100	0,0001
34	300	1100	0	100	0,0001
39	350	1280	0	100	0,0001
45	400	1459	0	100	0,0001
50	450	1605	0	100	0,0001
56	500	1893	0	100	0,0001
62	550	2100	0	100	0,0001
67	600	2300	0	100	0,0001
73	650	2483	0	100	0,0001
78	700	2750	0	100	0,0001
84	750	2920	0	100	0,0001
89	800	3156	0	100	0,0001
95	850	3321	0	100	0,0001
100	900	3465	0	100	0,0001
106	950	3685	0	100	0,0001
111	1000	3883	0	100	0,0001
117	1050	4080	0	100	0,0001
122	1100	4278	0	100	0,0001
128	1150	4475	0	100	0,0001
134	1200	4673	0	100	0,0001
139	1250	4870	0	100	0,0001
145	1300	5068	0	100	0,0001
150	1350	5265	0	100	0,0001

A continuación, se realizó el entrenamiento de la misma red con los parámetros de la tabla 10, pero esta vez no se utilizó el procesamiento en paralelo.

Tabla 12
Resultados Prueba 3 con el procesamiento de la CPU

Épocas	Iteración	Tiempo (s)	Mini-batch Loss	Mini-Batch Accuracy (%)	Base Learning Rate
1	1	62	13,24	54,25	0,0001
6	50	230	0	100	0,0001
12	100	425	0	100	0,0001
17	150	597	0	100	0,0001
23	200	879	0	100	0,0001
28	250	1024	0	100	0,0001
34	300	1238	0	100	0,0001
39	350	1490	0	87,32	0,0001
45	400	1750	0	100	0,0001
50	450	1945	0	100	0,0001
56	500	2176	0	100	0,0001
62	550	2359	0	100	0,0001
67	600	2573	0	100	0,0001
73	650	2787	0	63,45	0,0001
78	700	3001	0	100	0,0001
84	750	3215	0	100	0,0001
89	800	3429	0	75,43	0,0001
95	850	3643	0	100	0,0001
100	900	3857	0	100	0,0001
106	950	4071	0	100	0,0001
111	1000	4286	0	100	0,0001
117	1050	4500	0	100	0,0001
122	1100	4714	0	100	0,0001
128	1150	4928	0	100	0,0001
134	1200	5142	0	100	0,0001
139	1250	5356	0	100	0,0001
145	1300	5570	0	100	0,0001
150	1350	5784	0	100	0,0001

5.1.3.1 Prueba del Clasificador en los *frames* de videos

Para la prueba 3 de igual manera se usó dos videos con diferentes calidades uno de 1090 x 1080 pixeles y el otro de 320 x 240 pixeles.

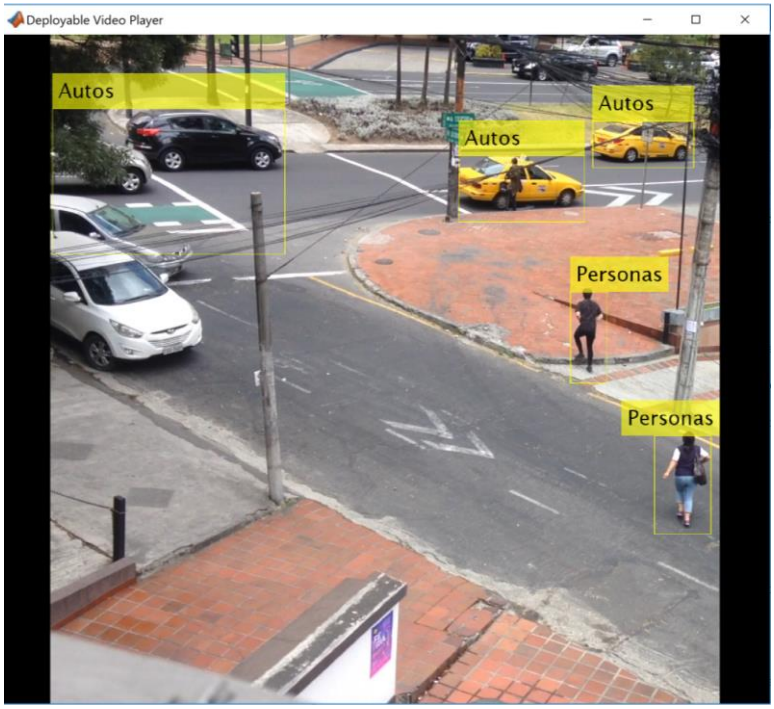


Figura 58. Carga de la Red entrenada en los frames de video para Prueba 3. Video 1090 x 1080 pixeles.

Figura 59. Carga de la Red entrenada en los frames de video para Prueba 3. Video 320 x 240 pixeles.

5.1.3.2 Análisis de Prueba 3

Para esta última prueba se observa en la tabla 11 que el tiempo aproximado de entrenamiento de la red fue de 5265 segundos, un tiempo mayor que el de las dos anteriores pruebas, esto se debe al número de imágenes aumentadas y el número de iteraciones. Según los resultados obtenidos se puede llegar a la conclusión que el porcentaje de error que va a tener esta red es prácticamente nulo debido a que en el Mini-Batch Accuracy se obtuvo una eficiencia de prácticamente 100% para calcular este porcentaje se tomaron los valores de la columna de *Mini-Batch Accuracy* de la tabla 11, se sumaron todos los valores y se dividió para el mismo número. Este porcentaje se obtuvo debido al aumento de imágenes que entran a la red. Para el caso en el cual se entrenó a la red únicamente con ayuda del CPU se obtuvo un tiempo aproximado de entrenamiento de 96 minutos, más que cuando se utiliza procesamiento en paralelo. En cuanto al Mini-Batch Accuracy se obtuvo un promedio de 97%, para calcular este porcentaje se tomaron los valores de la columna de *Mini-Batch Accuracy* de la tabla 12, se sumaron todos los valores y se dividió para el mismo número. Lo que quiere decir que la efectividad del clasificador de la red es alta. De esta manera en el caso de las tres pruebas el procesamiento en paralelo siempre fue superior.

En cuanto a la calidad de los videos de la misma manera que en las dos pruebas anteriores el clasificador tuvo mayor efectividad con el video de 1090 x 1080 pixeles ya que clasificaba a autos y personas sin problema alguno. Esto debido a la claridad de las imágenes.

De la misma manera que en la prueba 1 y 2, se tomaron los mismos videos los cuales no tenían un fondo estático, dando así exactamente los mismos resultados al momento de usar el clasificador. No clasificaba en ciertas ocasiones cuando el video tenía movimiento en el fondo.

CAPÍTULO 6

CONCLUSIONES Y RECOMENDACIONES

6.1 Conclusiones

- Se desarrolló un clasificador de dos tipos de imágenes (Autos y Personas) con ayuda de las redes neuronales convolucionales (CNN) que son usadas para este tipo de aplicaciones con imágenes. El programa se lo realizó en el software MATLAB versión R2017a. Para esto se entreno nuevamente a la red AlexNet modificando sus últimas capas para que clasifique únicamente autos y personas.

- Para el procesamiento al momento del entrenamiento de la Red Neuronal se utilizó la tarjeta gráfica NVIDIA GEFORCE GTX960M, previamente instalada en el computador. Con ayuda de esta GPU se logró acelerar el proceso de entrenamiento de la red en un 20%, debido a su capacidad computacional.

- El clasificador funcionó con videos que tengan un fondo fijo, es decir que no se esté moviendo, debido a que la red no clasifica cuando todo el cuadro o *frames* de video están en movimiento. Para ello se utilizó funciones que quitaron el fondo de las tramas de video y de esta manera la red pudo reconocer únicamente a los objetos que se encontraban en movimiento para su posterior clasificación.

- Existen varios tipos de clasificadores de imágenes que existen en la actualidad. Pero al momento de clasificar de mejor manera las Redes Neuronales Convolucionales son la mejor opción debido a que es más

eficiente ya que no tienen que la necesidad de estar extrayendo manualmente los descriptores de las imágenes, sino que tienen un aprendizaje de inicio a fin como una neurona interconectada.

- En las 3 pruebas realizadas en el entrenamiento de la Red, se obtuvieron valores similares. Las redes entrenadas tuvieron una efectividad de más del 80% en la mayoría de los casos. Debido a esto al momento de evaluar el desempeño del clasificador con los *frames* de video se obtuvieron resultados casi iguales en los tres casos. Lo que quiere decir que las Redes Neuronales son uno de los sistemas más efectivos al momento de realizar este tipo de operaciones.

- Se concluyó que mientras más imágenes se metan a la red para su entrenamiento mayor va a ser su eficiencia, pero el tiempo de entrenamiento va a ser mucho mayor.

- Se tomó dos videos en los cuales el fondo tenía movimiento es decir no era un fondo estable, y se realizó las pruebas con el clasificador. Dando así un resultado no favorable. Debido a que el clasificador únicamente fue eficaz cuando el fondo se encontraba fijo. Mientras que cuando tenía movimiento el clasificador lo que hizo fue confundirse y asignar etiquetas de autos y personas a cualquier objeto que tenía algún rasgo parecido a la de los dos objetos.

6.2 Recomendaciones

- Si se requiere utilizar las redes neuronales especialmente las Redes Neuronales Convolucionales en MATLAB, se recomienda utilizar versiones más actuales que el MATLAB R2016a, debido que en esta versión y versiones anteriores no se logró realizar el entrenamiento de la red ya que genera conflictos y errores. En el caso de este proyecto

se realizó en diferentes computadores teniendo así el mismo resultado con todos al momento de utilizar el MATLAB R2016a.

- Para el uso de la GPU conjuntamente con MATLAB es recomendable instalar todos los drivers y actualizaciones que estén disponibles para el GPU y de esta manera poder trabajar con un conjunto de herramientas al mismo tiempo. En el caso de este proyecto al momento de entrenar las redes neuronales no funcionaba debido a que los drivers de la GPU se encontraban desactualizados.

- Al momento de modificar las capas de las redes Neuronales tener mucho cuidado con las primeras capas, debido a que en ellas se aprende la información básica que servirá para después y que las capas superiores aprendan de ella.

6.3 Trabajos Futuros

Como una propuesta de trabajos futuros se tiene:

- Realizar un programa para clasificar más de dos categorías en *frames* de video, pero esta vez que el fondo del video se encuentre en movimiento. Con ayuda de una GPU de mucho más alto rendimiento debido a que las redes requieren de un alto proceso computacional.

- Para el campo de los sistemas de seguridad más específicamente los sistemas de videovigilancia se propone realizar una clasificación de rostros en *frames* de video utilizando este tipo de redes neuronales con la ayuda de una base de datos previamente cargada.

- Los sistemas de seguridad tienen un sin número de aplicaciones que se podrían mejorar o inventar con ayuda de las redes neuronales. Una

propuesta es realizar una detección y clasificación de personas y automóviles que entren en determinada área y dependiendo si el área es restringida dar seguimiento a los mismo, para de esta manera de alguna forma emitir una señal que pueda activar automáticamente una alarma y así alertar a las autoridades.

REFERENCIAS BIBLIOGRÁFICAS

Alex Krizhevsky, I. S. (2012). *ImageNet Classification with Deep Convolutional Neural Networks.* Toronto: University of Toronto.

Antonio J. Serrano, E. S. (2010). *Redes Neuronales Artificiales.* Valencia: Universidad de Valencia.

Ballesteros, A. (2008). *Software Neural Networks.* Malaga: University of Malaga.

Deshpande, A. (2016). *A Beginner's Guide to Understanding Convolutional Neural Networks.* Los Angeles: UCLA.

García, L. (22 de Junio de 2017). Deep Learning for Computer Vision with Matlab. (A. Amores, Entrevistador)

García, P. P. (2013). *Reconocimiento de Imágenes utilizando Redes Neuronales Artificiales.* Madrid: Universidad Complutense de Madrid.

García, V. G. (2010). *Estimación y clasificación de daños en materiales utilizando modelos AR y redes neuronales para la evaluación no destructiva con ultrasonidos.* Granada: Universidad de Granada.

González, G. F. (2009). *Redes Neuronales aplicadas al análisis de imágenes para el desarrollo de un prototipo de un sistema de seguridad.* Pereira: Universidad Tecnológica de Pereira.

Jerry A. Fodor, Z. W. (1988). *Connectionism and Cognitive Architecture: A Critical Analysis.* New Jersey: Rutgers University.

López, R. (2017). *Introducción al Deep Learning.* Buenos Aires: IAAR Capacitación.

Luis A. Palacios-Sánchez, (2006). *Clasficiador Genérico de Objetos en Imágenes ETM+.* Texcoco, México: Agrociencia, Vol.40, núm. 5.

Mihaich, F. (2014). *Aplicación de redes Neuronales en la clasificación de imágenes.* Córdoba: Universidad Nacional de Córdoba.

Moeslund, T. B. (2012). *Introduction to Video and Image Processing Building Real Systems and Applications .* London: Springer-Verlag London.

Moreno, J. J. (2002). *Redes Neuronales Artificiales aplicadas al Análisis de Datos.* Mallorca: Univesitat de Les Illes Balears.

Raquel Flórez López, J. M. (2008). *Las redes neuronales artificiales; fundamentos teóricos y aplicaciones prácticas.* Oleiros: Ontology Engineering Group.

Satué, M. G. (2013). *Reconocimiento de señales de tráfico para un sistema de ayuda a la conducción.* Sevilla: Universidad de Sevilla.

Tepán, E. C. (2013). *Estudio de los principales tipos de redes neuronales y las herramientas para su aplicación.* Cuenca: Universidad Politécnica Salesiana.