PROYECTO FINAL DE INGENIERÍA

Reconocimiento de objetos en videos en tiempo real

De Berniger Lacoste, Alan Nicolas – LU1010192
Ingeniería en Informática

Tutor/es:
Cuadrado Estrebou, María Fernanda, UADE

Noviembre 30, 2016

UNIVERSIDAD ARGENTINA DE LA EMPRESA
FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS

Resumen

El presente proyecto consiste en la investigación y desarrollo de un sistema capaz de aprender a identificar objetos en videos en tiempo real. Está formado principalmente por dos módulos. El módulo de **aprendizaje** el cual tiene como objetivo aprender cómo es un objeto identificando aquellas características que lo identifican, y el módulo de **reconocimiento** el cual se encarga de reconocer los distintos objetos que forman parte de una imagen o un video dado. Para lograr esto, el sistema está inspirado en la forma en la que los mamíferos realizan esta tarea. Utiliza una red neuronal artificial que aprende a identificar las características que definen a un objeto basándose en distintas imágenes del mismo. Estas imágenes son entregadas a la red neuronal en forma de estímulos que activan a las distintas neuronas que forman la capa de entrada de la red. Estos estímulos son propagados a través de la red utilizando las conexiones sinápticas que existen entre las neuronas estimulándose unas a otras según el peso con la que fue configurada esa conexión. Al final de la red, existe una neurona por cada objeto que la red puede reconocer. Estas neuronas son las neuronas de salida, y son las encargadas de dar una probabilidad de reconocimiento por cada objeto. En la fase de entrenamiento se le da a la red las imágenes y se le indica a que objeto corresponde cada una para que la red vaya ajustando los pesos de las conexiones según la importancia de cada neurona de entrada (Identificación de características).

Este sistema tiene diversas aplicaciones como puede ser la clasificación automática de archivos personales ya sean fotos o videos, conversión de textos escritos en formato de imagen a formato de texto, análisis forense de video cuando se requiere identificar un objeto determinado que aparece en una escena investigada o autenticación biométrica por reconocimiento facial. Este proyecto se enfoca en el sistema de reconocimiento de objetos en cámaras de seguridad tanto para objetos que estén estáticos como para objetos que se encuentran en movimiento.

Abstract

The objective of this project is the investigation and development of a system which is able to learn to identify objects in videos in real time. It consists mainly of two modules. The learning module has as purpose to learn how an object is identifying the properties that identifies it and the recognizing module which recognizes the objects that appears in an image or a video. To achieve this, the system is inspired in the mammalian visual system. It implements an artificial neural network which learns to identify the properties that identify an object based on different images of it. These images are delivered to the neural network in the form of stimuli that activate the different neurons that are part of the input layer. These stimuli are propagated through the network using the synaptic connections that exists between the neurons stimulating each other according to the weight with which that connection was configured. At the end of the network, there is one neuron for each object that the network can recognize. These neurons are the output neurons, and are responsible for giving a probability of recognition for each object. In the training phase the images are given to the network and it is indicated to which object corresponds each one so that the network will adjust the weights of the connections according to the importance of each input neuron (Properties identification).

This system has several applications such as the automatic classification of personal files, such as photos or videos, conversion of texts written in image format to text format, forensic video analysis when it is required to identify a specific object that appears in a scene investigated or facial recognition biometric authentication. This project focuses on the object recognition system in security cameras for both static and moving

Introducción

Objetivos

Se construirá un sistema capaz de reconocer objetos dentro de imágenes o en videos en tiempo real. Este sistema podrá ser de utilidad para el análisis de videos de cámaras de vigilancia donde se requiera la detección automática de determinados objetos como puede ser un vehículo secuestrado.

El desarrollo efectuado, y los algoritmos investigados para llevar a cabo el mismo, permitirán el análisis automático de información en videos, lo cual permite obtener información de fuentes existentes sin la necesidad del análisis manual, acelerando la obtención de esa información en tiempo real, o a efectos de investigación sobre videos grabados.

El sistema desarrollado a partir de la investigación será construido de manera tal que sea fácilmente configurable por cualquier usuario pero que a su vez un usuario experto pueda ajustar de manera más precisa los parámetros que intervienen en el proceso de entrenamiento de identificación de objetos con el fin de lograr una mayor efectividad de reconocimiento.

Alcance

En este trabajo se definió un algoritmo para análisis de objetos sobre imágenes y se construyó un prototipo capaz de aprender cómo identificar un determinado objeto para luego poder identificarlo en un video en tiempo real.

Este prototipo está compuesto por dos módulos. Uno de los módulos es el encargado de recibir información del usuario acerca de cómo identificar un objeto determinado, mientras que el otro se encarga de, a partir de lo aprendido, encontrar el objeto a identificar en el video.

El trabajo consta en la investigación y diseño de los algoritmos requeridos para llevar a cabo lo mencionado anteriormente, desarrollando un prototipo conceptual con el objetivo de ser utilizado para realizar pruebas y comprobaciones de los algoritmos.

Los algoritmos están descriptos por medio de pseudocódigo, y el prototipo está desarrollado en C#, entregando documentación de diseño conceptual y funcional del mismo, incluyendo una breve descripción del funcionamiento del prototipo.

Estado del arte

Actualmente en el mercado existen diferentes sistemas denominados analíticas que realizan la función de análisis de video. Estos sistemas realizan distintas tareas de análisis como el reconocimiento de placas, de rostros y de comportamientos. Algunas de las empresas que comercializan sistemas de reconocimiento de placas son: IIS (Inteligent Security Systems), Genetec, Vivotek, netHumans, tsn (Security & Telecom). Algunas de las que realizan reconocimiento de rostros son: Aware (Nexa | Face), Luxand.

Los sistemas de análisis de video pueden correr directamente sobre las cámaras de video como el Bosch o pueden correr en un servidor recibiendo el video desde una cámara en vivo o video grabado.

En la actualidad tanto en el ámbito público como privado aumenta la utilización de cámaras como medida de seguridad, llevando a muchas situaciones en que se disponen de minutos y minutos de videos que no pueden ser analizados en forma manual por un humano, por lo tanto, las herramientas de detección de patrones o comportamientos son cada vez más requeridas para colaborar en el análisis de esas imágenes que disponen las empresas y/o instituciones

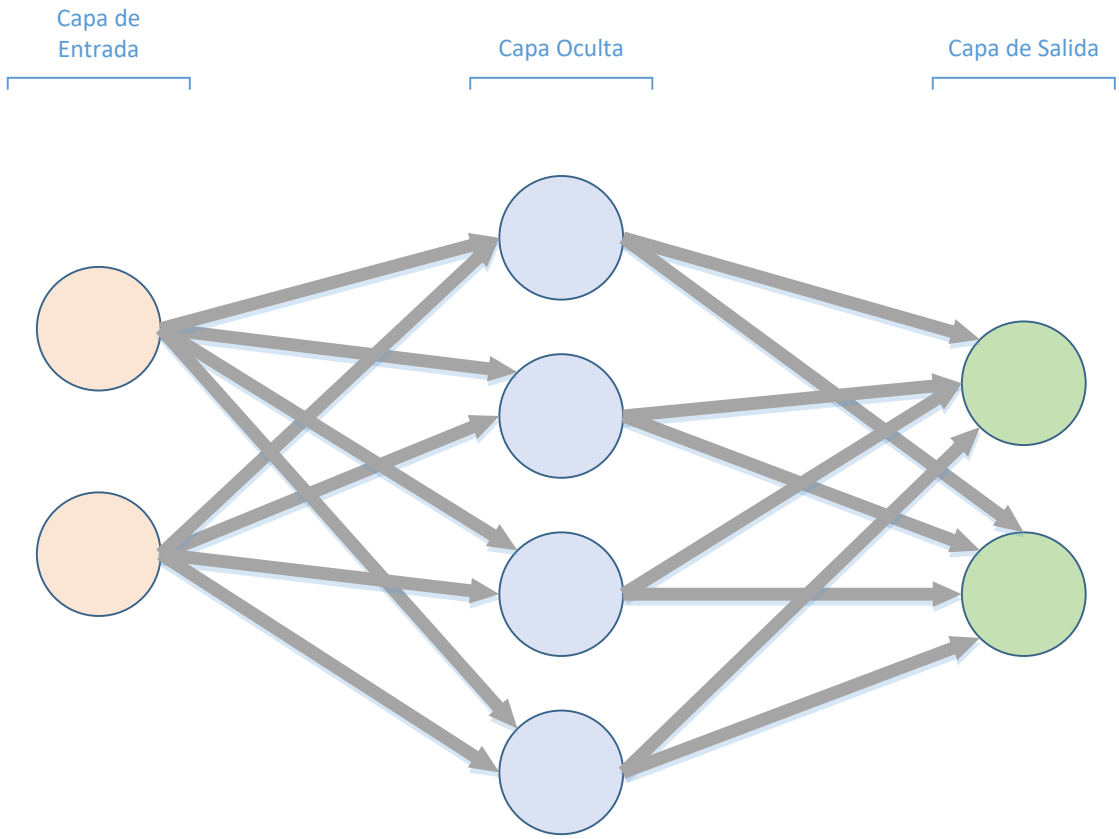
Desarrollo

Para lograr el reconocimiento de los distintos objetos definidos por el usuario, el sistema crea una red neuronal artificial con una arquitectura específica para lograr esta tarea y ajusta los pesos de las conexiones sinápticas que existen entre sus neuronas de acuerdo a las imágenes de los objetos que entrega el usuario. Las redes neuronales artificiales son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en la que funciona el sistema nervioso biológico. Dentro de la red, las neuronas se agrupan en capas. La primera de ellas es la capa de entrada, la cual contiene las neuronas que reciben los datos de entrada a la red, en este caso la imagen de un objeto a entrenar o uno de los fotogramas del video a reconocer. A continuación, las neuronas se organizan en 1 o más capas ocultas. Y por último una capa de salida, que tendrá las neuronas que darán el resultado según la entrada.

Una red neuronal tiene dos fases: La fase de entrenamiento, en la cual se le entregan imágenes de entrenamiento indicándole a que objeto corresponde cada imagen para que se ajusten los pesos de las conexiones sinápticas. Y la fase de reconocimiento, en la cual se entrega una imagen a la red y ésta propaga los valores de entrada a través de la red para dar como resultado la probabilidad de reconocimiento de cada uno de los objetos posibles a reconocer por la misma.

Red Neuronal

Una red neuronal es un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso biológico. Está formada por neuronas que se agrupan en capas y se conectan entre sí a través de conexiones sinápticas (Ver Anexo IV – Neurona). Las neuronas que forman parte de la capa de entrada son aquellas que reciben los datos de entrada a la red, como pueden ser las intensidades de color de cada pixel de una imagen. Las neuronas de las capas ocultas son aquellas que buscaran patrones en los datos de entrada. Y, por último, las neuronas de la capa de salida son aquellas que indicaran el resultado del análisis de los datos entrada. Para el caso del reconocimiento de una imagen, existirá una neurona de salida por cada objeto que puede reconocer la red. Cada neurona de salida producirá un valor de activación que indicará la probabilidad de que la imagen de entrada corresponda al objeto que la neurona está representando.



Entrenamiento de la red neuronal

El entrenamiento de la red neuronal consiste en entregarle a la red imágenes de los objetos que se quiere reconocer indicándole a que objeto corresponde cada imagen. A esta forma de entrenamiento se le llama "Aprendizaje supervisado" [1] ya que la red recibe las imágenes clasificadas por objeto. Cuando se entrena a la red por primera vez, lo primero que se debe hacer es inicializar los pesos de las conexiones sinápticas con valores aleatorios. Una vez que los pesos fueron inicializados, se ingresa una imagen a la red. Los datos de la imagen son recibidos por las neuronas de la capa de entrada y propagan estos valores a través de las conexiones sinápticas entre las neuronas de la red. Cuando la imagen llega a la última capa, se activarán las neuronas de esta capa con distintos valores. Este proceso se realiza con cada una de las imágenes que se encuentran en los sets de entrenamiento de cada objeto. Lo que se busca en esta etapa es configurar los valores de los pesos y los bias de la red que logren la mayor efectividad en el reconocimiento de las imágenes que se encuentran en el set de pruebas de cada objeto. Para lograr esto se utiliza el algoritmo "Gradient descent" [2]. Este algoritmo nos permite, dada una función de costo que mide el error de reconocimiento de la red, encontrar el conjunto de pesos y bias de la red que minimiza esta función. Anexo VII - Gradient descent. Para minimizar esta función se calculan las derivadas parciales respecto a cada uno de los pesos y a cada uno de los bias. Para realizar el cálculo de estas derivadas se utiliza el algoritmo "Backpropagation" [3]. Anexo VIII - Backpropagation. Se utilizará una variación del algoritmo "Gradient descent" llamado "Stochastic gradient descent". Esta variación consiste en actualizar los pesos y los bias de la red tomando un subconjunto del set de imágenes de entrenamiento. A éste subconjunto se lo llama "mini-batch". Se repite este ciclo hasta haber tomado todas las imágenes del set de entrenamiento. Este ciclo es llamado "Epoch".

[1] **(10) Russell, Ingrid. 1996.** Neural Networks Module
[2] **(8) Nielsen, Michael. 2015.** Neural Networks and Deep Learning. Capitulo 1 – Learning with gradient descent
[3] **(8) Nielsen, Michael. 2015.** Neural Networks and Deep Learning. Capitulo 2

Arquitectura de la red neuronal

La red neuronal que se va a utilizar para el aprendizaje y reconocimiento de los objetos entregados por el usuario es una "Red neuronal convolucional" [4]. Ver Anexo IX – Red neuronal convolucional

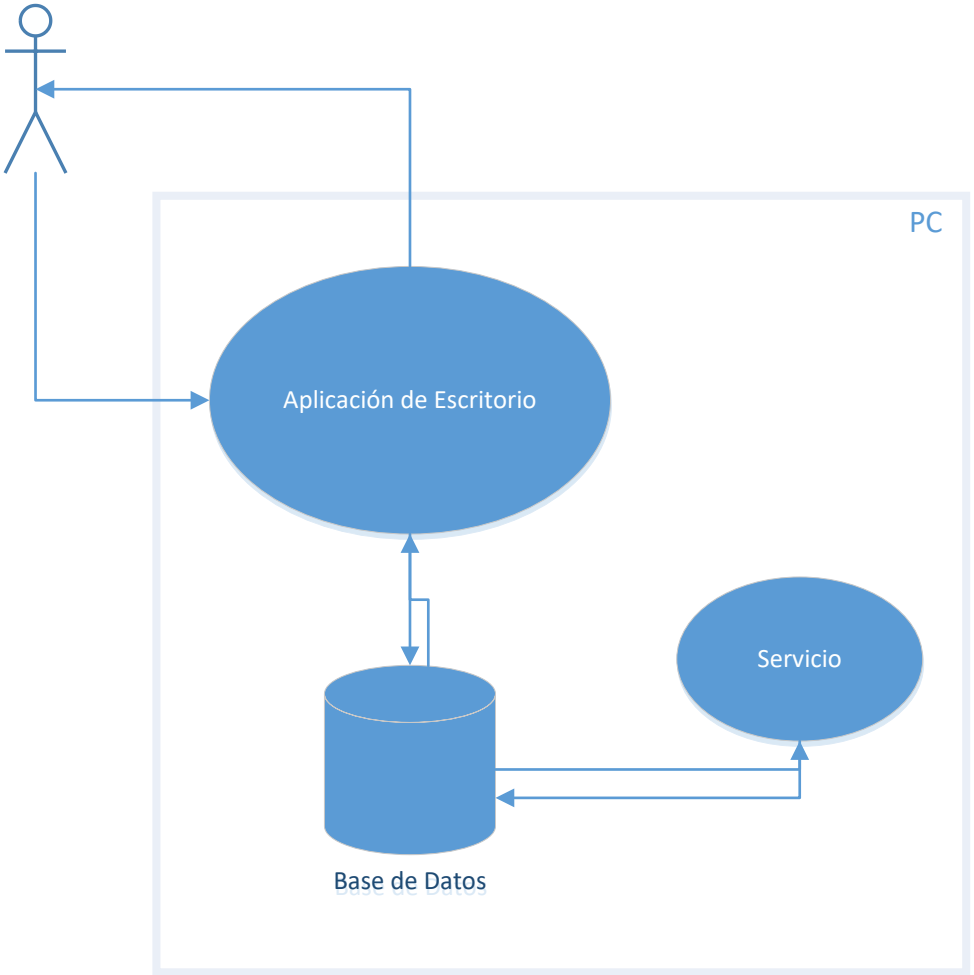
Reconocimiento de un video

El sistema desarrollado esta optimizado para el reconocimiento de objetos en videos de cámaras de seguridad. En primer lugar, toma el primer fotograma del video y reconoce los objetos que intervienen en esta imagen. Este primer reconocimiento sirve para detectar los objetos que se encuentran estáticos en el video. Para lograr esto, se utiliza un algoritmo llamado "Canny" el cual detecta los bordes en la imagen para poder segmentar la misma. Ver Anexo II - Canny. Cada una de estas sub-imágenes resultantes se envían a la red neuronal entrenada para ser reconocidas. Una vez finalizado este proceso, el sistema monitorea los distintos fotogramas del video buscando cambios que indiquen la existencia de un objeto en movimiento. Una vez encontrado el movimiento en una región de la imagen, se toma una sub-imagen de esta región y se la envía a la red neuronal para ser reconocida.

[4] **(3) Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, Awni Hannun, Brody Huval, Tao Wang, Sameep Tandon.** UFLDL Tutorial – Convolutional Neural Network

Arquitectura

El sistema está compuesto por una aplicación de escritorio desarrollada en C#.net, una base de datos de SQL Server y un servicio de Windows. La aplicación de escritorio permite al usuario crear los objetos a reconocer asociándole las imágenes de entrenamiento y de prueba, crear redes neuronales indicando que objetos debe reconocer y crearle entrenamientos para que el servicio pueda entrenar la red. Una vez entrenada la red, el usuario puede cargar imágenes o videos para que el sistema reconozca que objetos intervienen. En la base de datos, la aplicación almacena los objetos con sus imágenes asociadas, los entrenamientos creados y las redes neuronales que el usuario entrena.



Funcionalidades

El sistema desarrollado permite al usuario definir objetos con un nombre especifico e imágenes que lo identifquen, así como también categorizarlos para establecer relaciones jerárquicas entre los mismos. Una vez definidos los objetos, el sistema permite crear redes neuronales seleccionando distintos objetos y asignarles entrenamientos con distintos parámetros. Una vez creada una red neuronal, el sistema permite cargar una imagen o un video y reconoce los objetos que intervienen. El detalle funcional del sistema se encuentra en el Anexo X – Casos de uso

Diagrama de contexto

En la fase de entrenamiento, el usuario envía al sistema la definición de los objetos que desea que el sistema reconozca. La definición de un objeto está compuesta por el nombre del objeto con el cual se lo va a identificar, un conjunto de imágenes del objeto que se utilizaran para entrenar la red neuronal del sistema, y otro conjunto de imágenes del mismo que se utilizaran para probar la efectividad de este proceso de entrenamiento.

En la fase de reconocimiento, el usuario envía al sistema la imagen o el video que quiere que el sistema reconozca. El sistema envía esta información a la red neuronal entrenada y devuelve al usuario los objetos reconocidos.

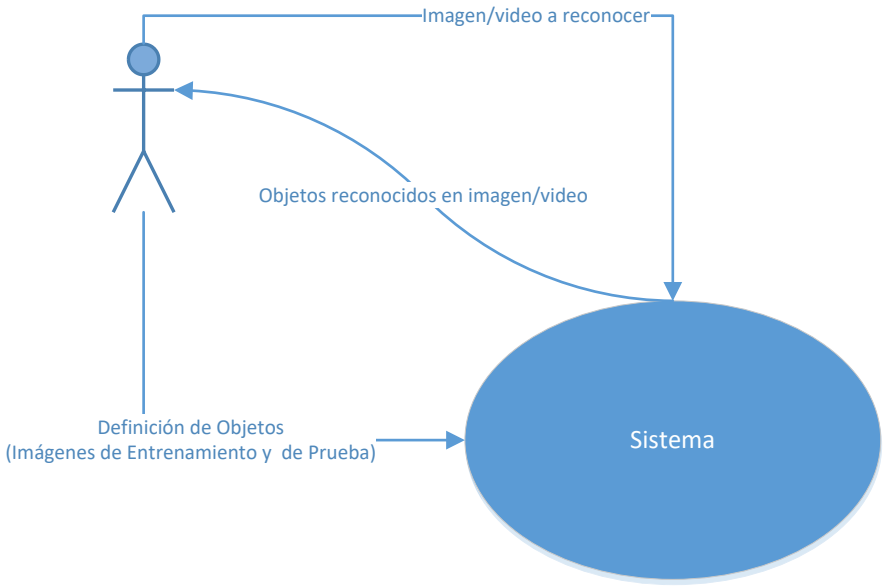
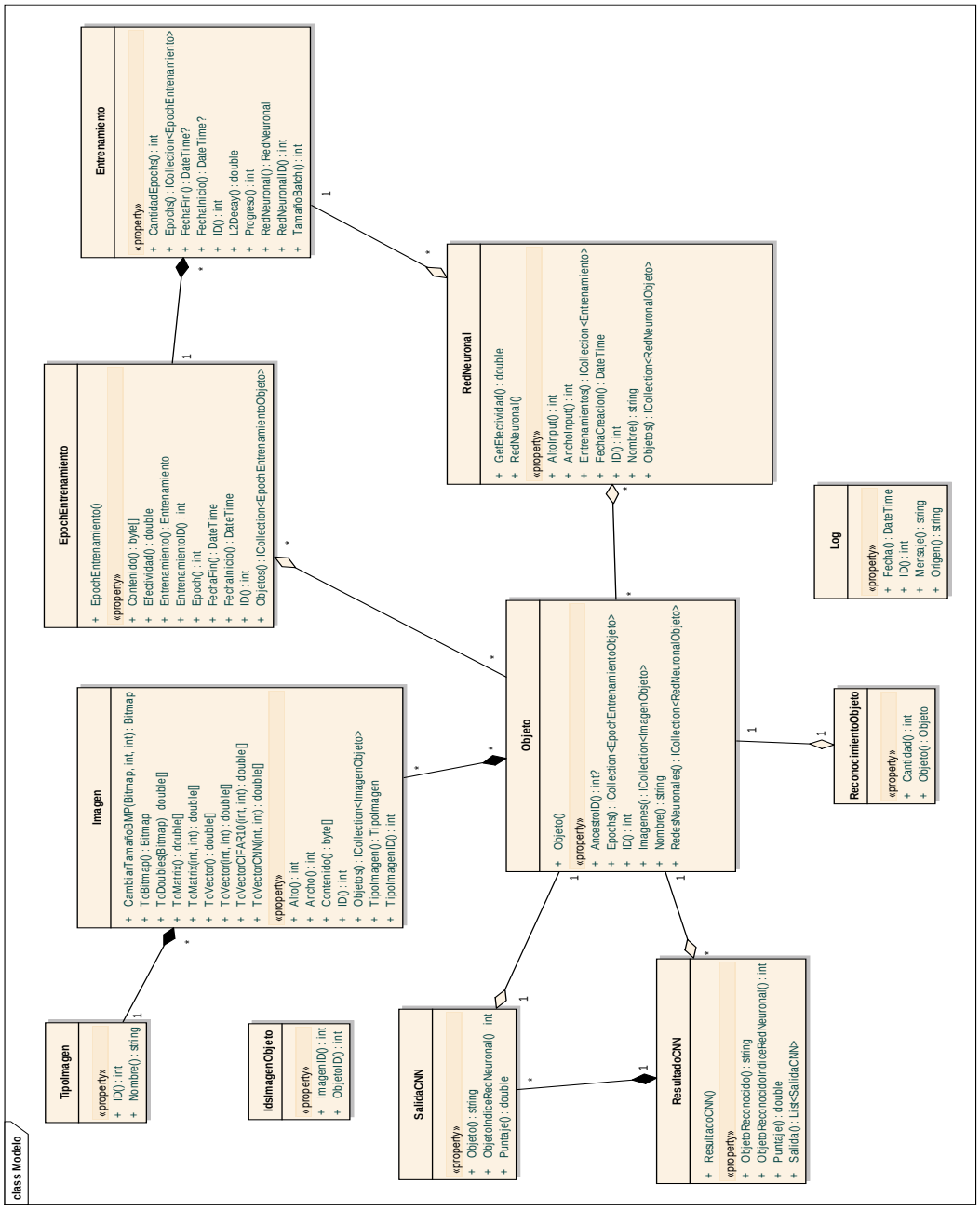
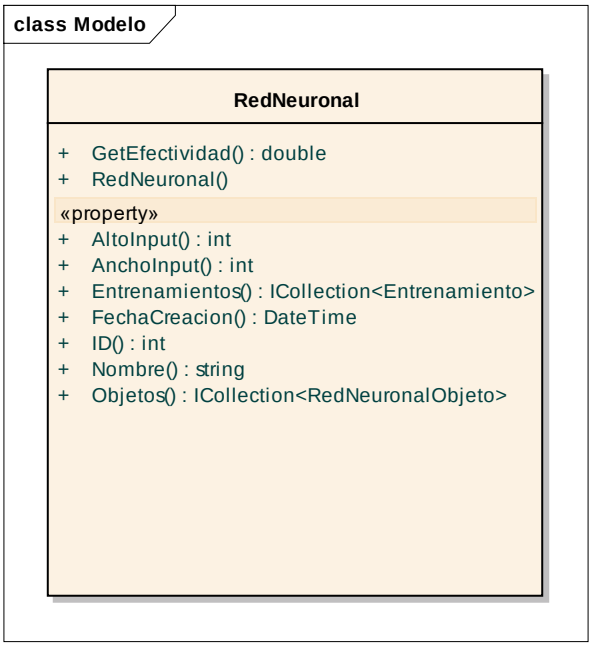


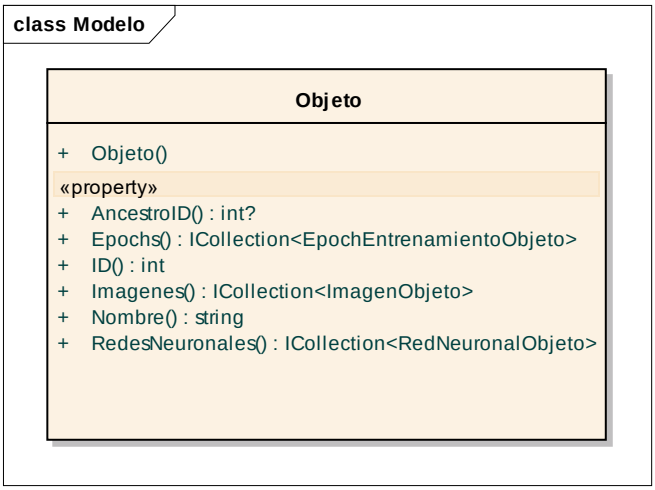
Diagrama de clases

Las clases que componen el sistema representan los objetos creados por el usuario con sus imágenes de entrenamiento y de prueba, y las redes neuronales entrenadas que son utilizadas en la fase de reconocimiento.

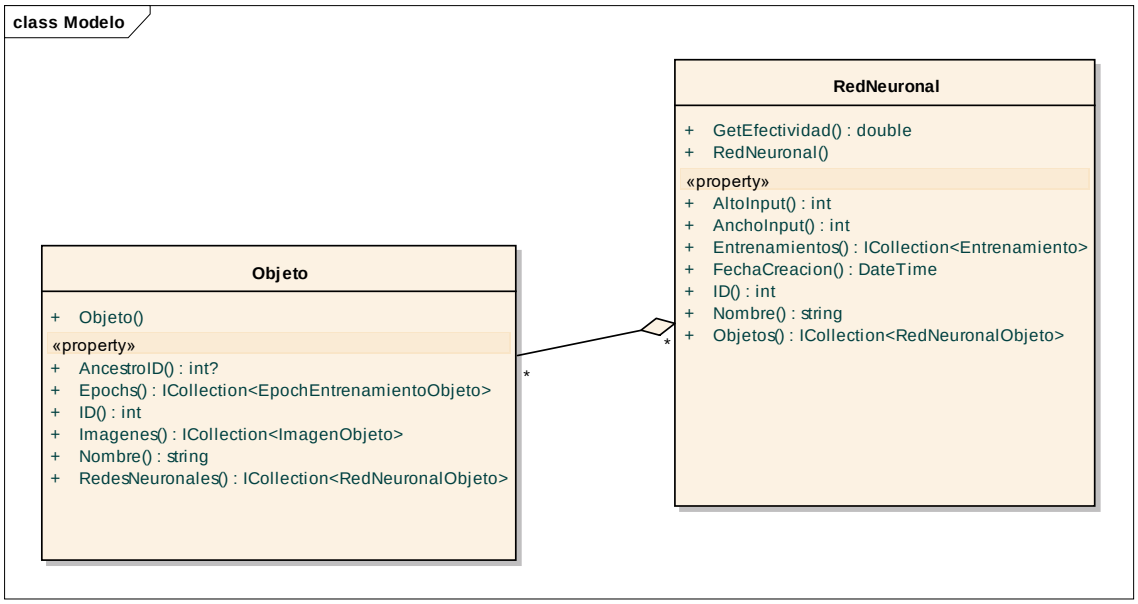


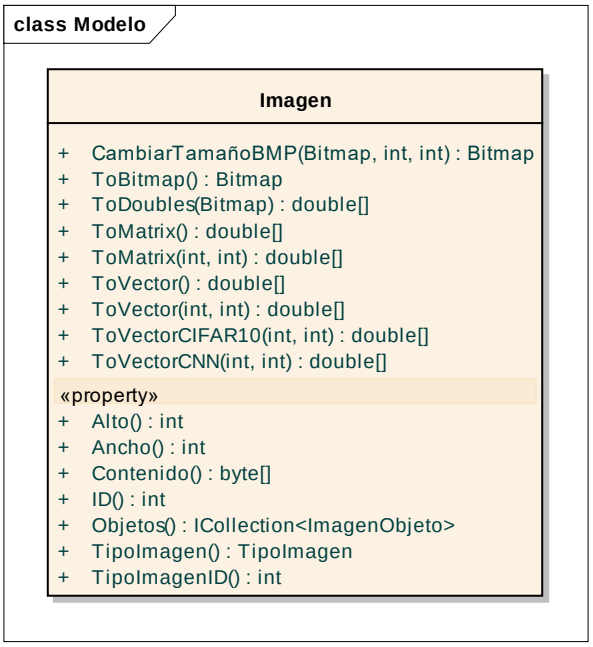


La clase “Red neuronal” es la que representa las redes neuronales que se crean en el sistema. Tiene como propiedades el ancho y el alto de las imágenes de entrada a la red (“AnchoInput” y “AltoInput”), los entrenamientos que se crearon para esta red, la fecha en la que se creo, el nombre, y los objetos que es capaz de reconocer. Las propiedades “AltoInput”, “AnchoInput” y “Objetos” son las que van a definir la estructura de la red. De acuerdo a los valores que tomen estas propiedades se definirá la cantidad de neuronas que formaran la capa de entrada y la cantidad de neuronas que formaran la capa de salida. El método “GetEfectividad” recorre todos los entrenamientos de la red y recupera la mayor efectividad de reconocimiento.

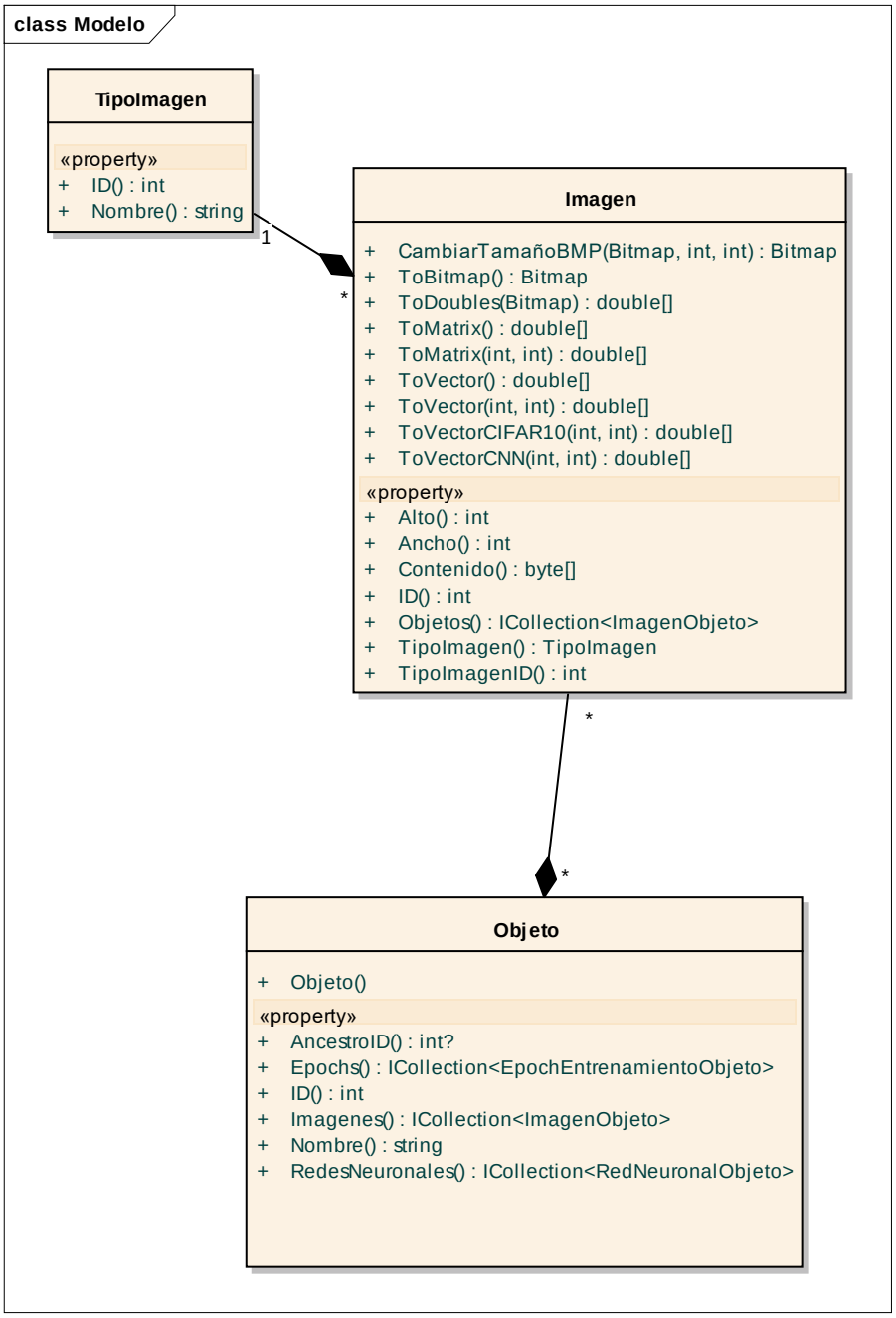


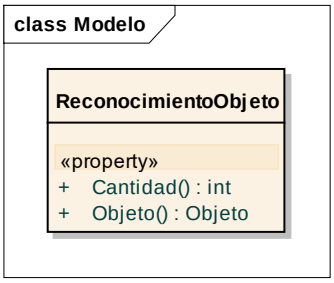
La clase “Objeto” es la que representa los objetos que se crean en el sistema. Los campos “ID” y “Nombre” son los que identifican al objeto. A un objeto se le pueden asociar distintas imágenes ya sean de entrenamiento o de prueba. La propiedad AncestroID define de qué objeto hereda este objeto. La propiedad “RedesNeuronales” representa todas las redes neuronales de las cuales este objeto forma parte.



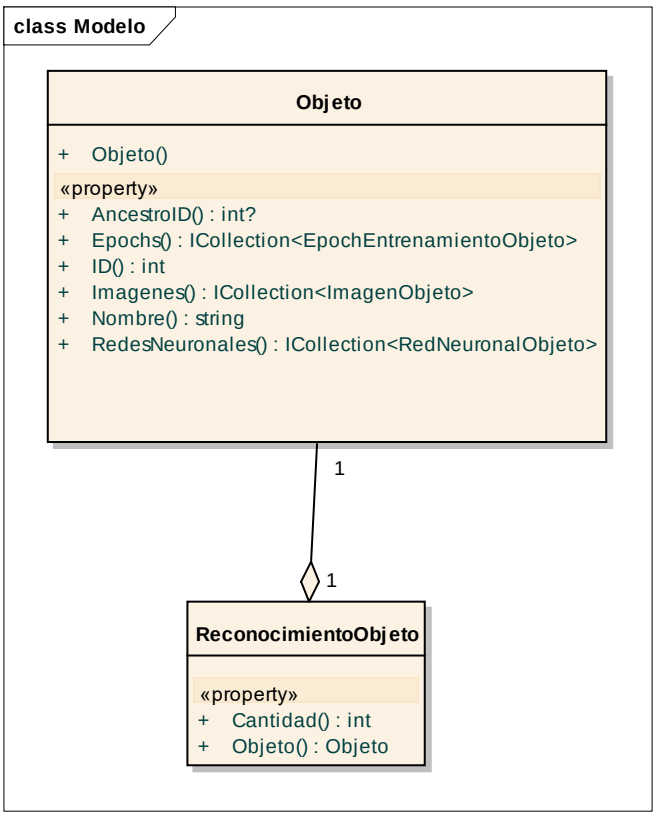


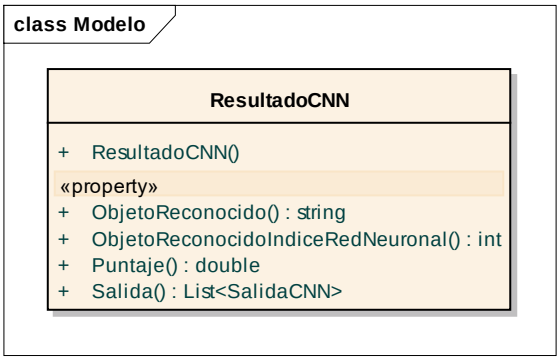
La clase imagen representa las imágenes que se cargan en el sistema. Las propiedades “Alto” y “Ancho” indican cual es el tamaño de la imagen y en la propiedad “Contenido” se encuentran la imagen en formato binario. La propiedad “TipoImagen” indica si la imagen es de entrenamiento o de prueba. En la propiedad “Objetos” se encuentran todos los objetos a los cuales está asociada la imagen.


class Modelo
TipoImagen
«property»
+ ID() : int
+ Nombre() : string
1
*
Imagen
+ CambiarTamañoBMP(Bitmap, int, int) : Bitmap
+ ToBitmap() : Bitmap
+ ToDoubles(Bitmap) : double[]
+ ToMatrix() : double[]
+ ToMatrix(int, int) : double[]
+ ToVector() : double[]
+ ToVector(int, int) : double[]
+ ToVectorCIFAR10(int, int) : double[]
+ ToVectorCNN(int, int) : double[]
«property»
+ Alto() : int
+ Ancho() : int
+ Contenido() : byte[]
+ ID() : int
+ Objetos() : ICollection<ImagenObjeto>
+ TipoImagen() : TipoImagen
+ TipoImagenID() : int
*
*
Objeto
+ Objeto()
«property»
+ AncestroID() : int?
+ Epochs() : ICollection<EpochEntrenamientoObjeto>
+ ID() : int
+ Imagenes() : ICollection<ImagenObjeto>
+ Nombre() : string
+ RedesNeuronales() : ICollection<RedNeuronalObjeto>

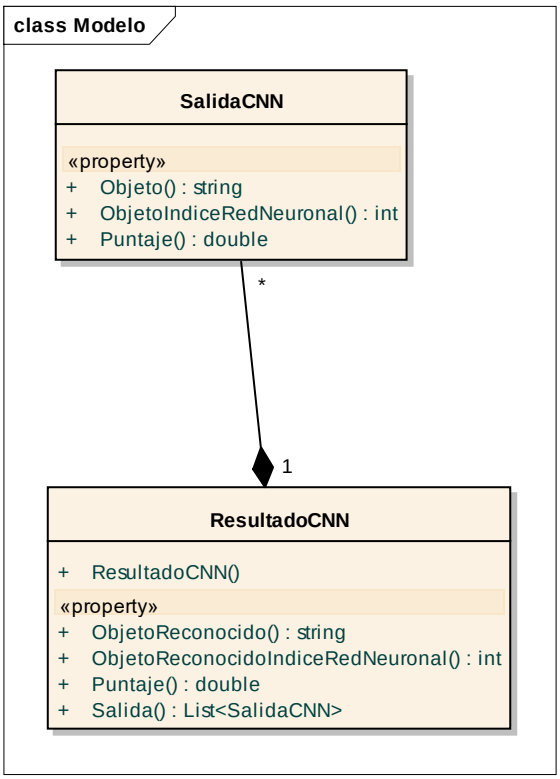


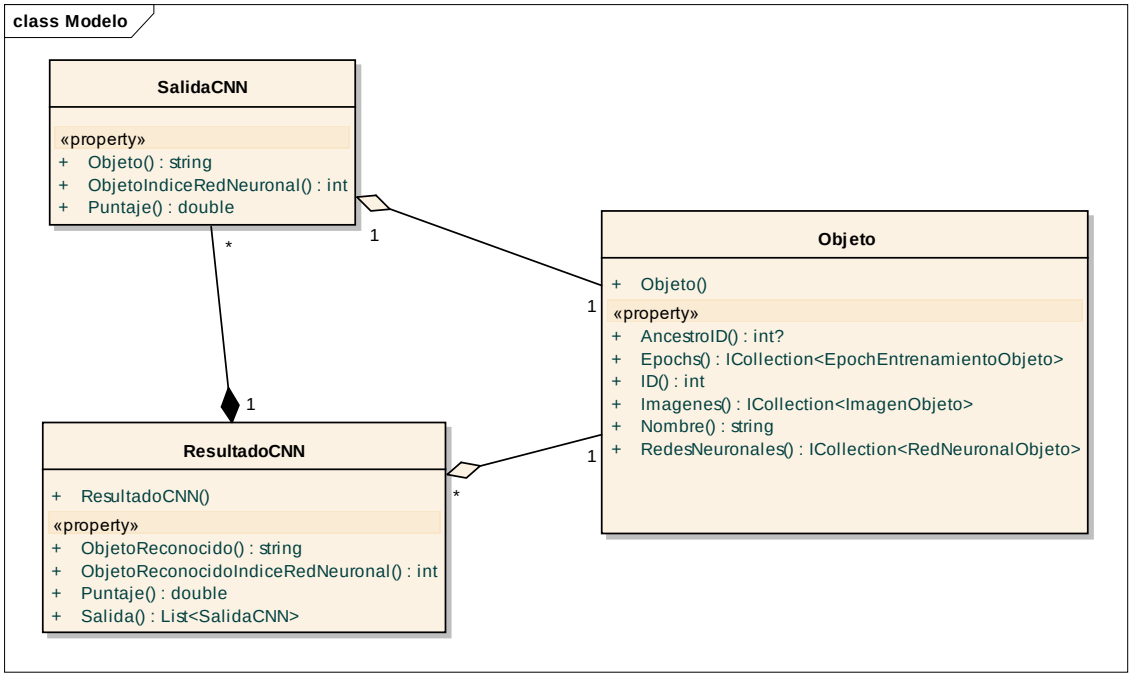
La clase “ReconocimientoObjeto” representa el resultado del reconocimiento de una imagen. Se utiliza para indicar cuantas veces aparece un objeto en una imagen. En la fase de reconocimiento se utiliza un listado de instancias de esta clase para dar el resultado final al usuario.

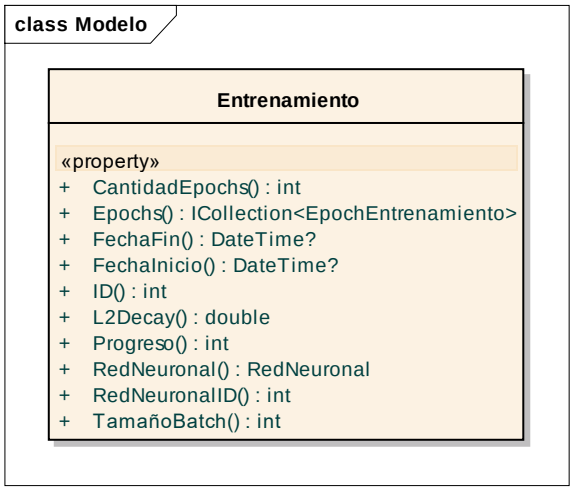




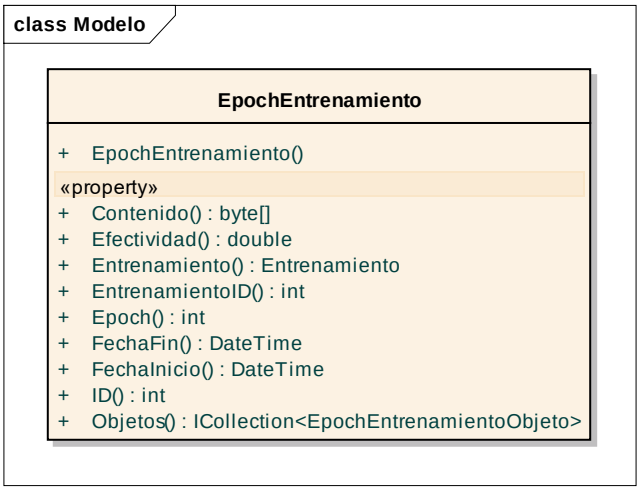
La clase “ResultadoCNN” representa el resultado de propagar una imagen a través de una red neuronal. La propiedad “ObjetoReconocido” representa al objeto con mayor probabilidad de reconocimiento (Mayor puntaje). La propiedad “ObjetoReconocidoIndiceRedNeuronal” indica cual es el índice del objeto en la salida de la red neuronal creada. Esta clase tiene una lista con el puntaje de cada objeto como resultado de la propagación de una imagen a través de la red neuronal



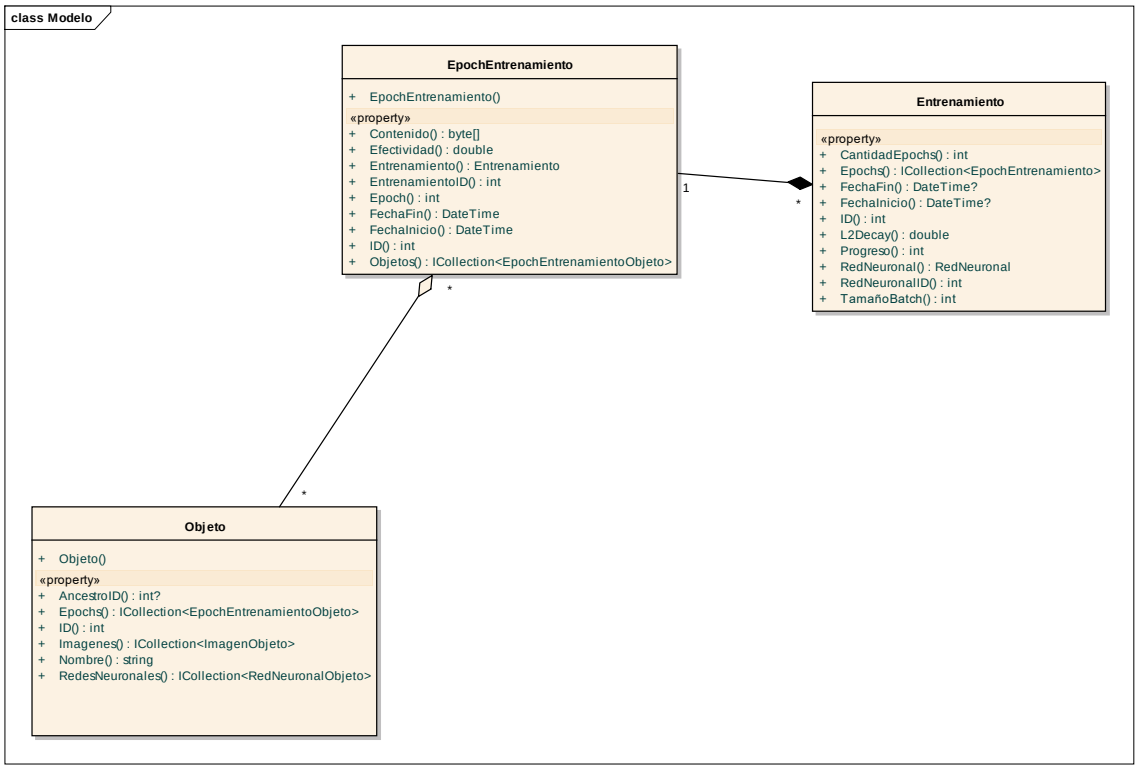

class Modelo
SalidaCNN
«property»
+ Objeto() : string
+ ObjetoIndiceRedNeuronal() : int
+ Puntaje() : double
Objeto
+ Objeto()
«property»
+ AncestroID() : int?
+ Epochs() : ICollection<EpochEntrenamientoObjeto>
+ ID() : int
+ Imagenes() : ICollection<ImagenObjeto>
+ Nombre() : string
+ RedesNeuronales() : ICollection<RedNeuronalObjeto>
ResultadoCNN
+ ResultadoCNN()
«property»
+ ObjetoReconocido() : string
+ ObjetoReconocidoIndiceRedNeuronal() : int
+ Puntaje() : double
+ Salida() : List<SalidaCNN>
1
*
1
1
1
*



La clase “Entrenamiento” representa una configuración de entrenamiento para una red neuronal. La propiedad “CantidadEpochs” indica cuantas iteraciones de entrenamiento se deben efectuar. Una iteración de entrenamiento consiste en mostrarle todas las imágenes de entrenamiento a la red neuronal. Debido a que se usa el método de optimización “gradient descent” que busca la configuración de pesos y bias que minimiza la función de costo, es necesario entrenar la red más de 1 vez para tratar de encontrar el menor costo. En la propiedad “Epochs” se almacenan los resultados de cada iteración de entrenamiento.



La clase “EpochEntrenamiento” representa una iteración de una configuración de entrenamiento de una red neuronal. En la propiedad “Contenido” se almacena la red neuronal (con su configuración de pesos y bias) luego de la iteración de entrenamiento. En la propiedad “Efectividad” se guarda el porcentaje de efectividad con el cual está reconociendo los objetos. En la propiedad “Objetos” se guarda la efectividad con la que está reconociendo cada objeto.



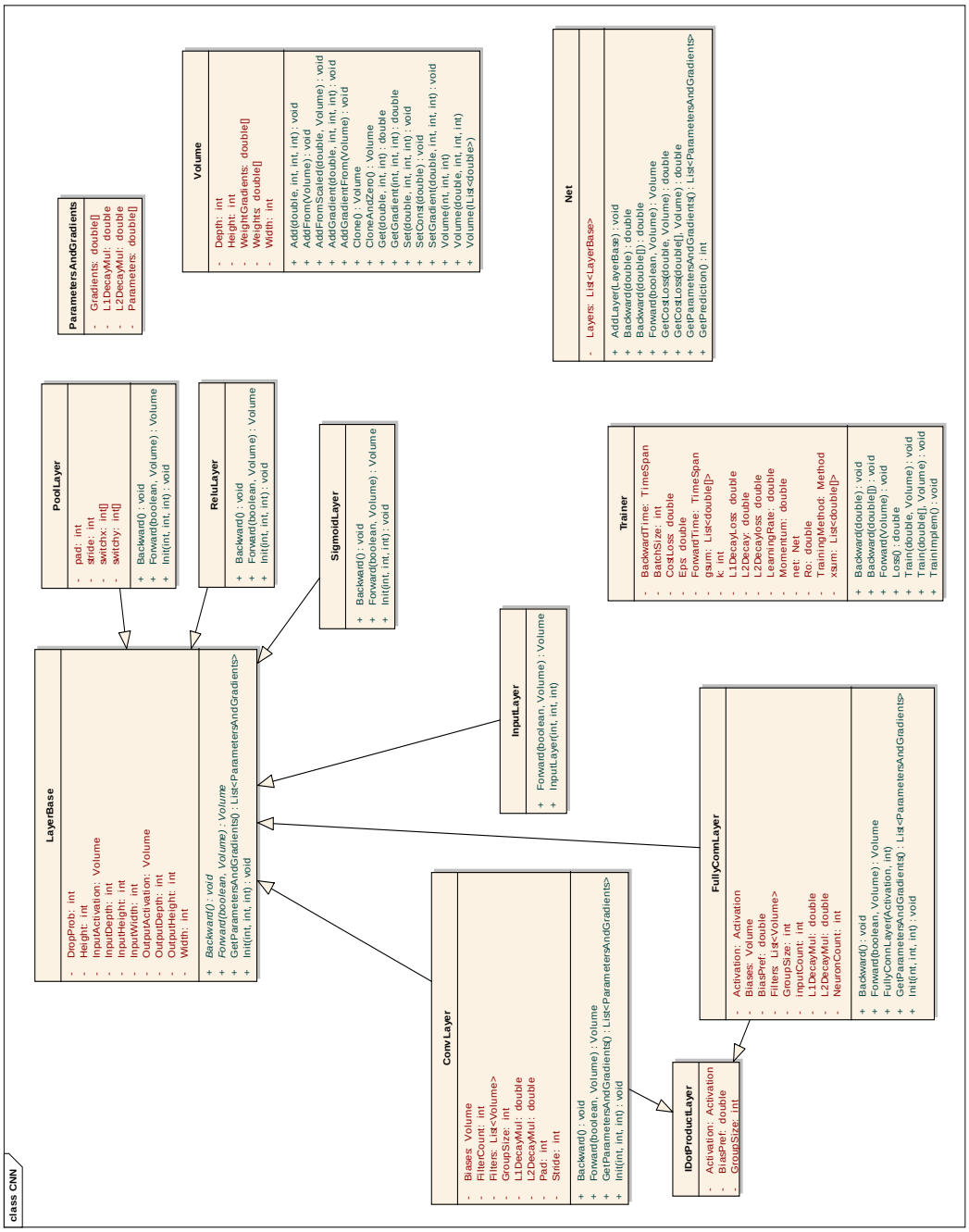
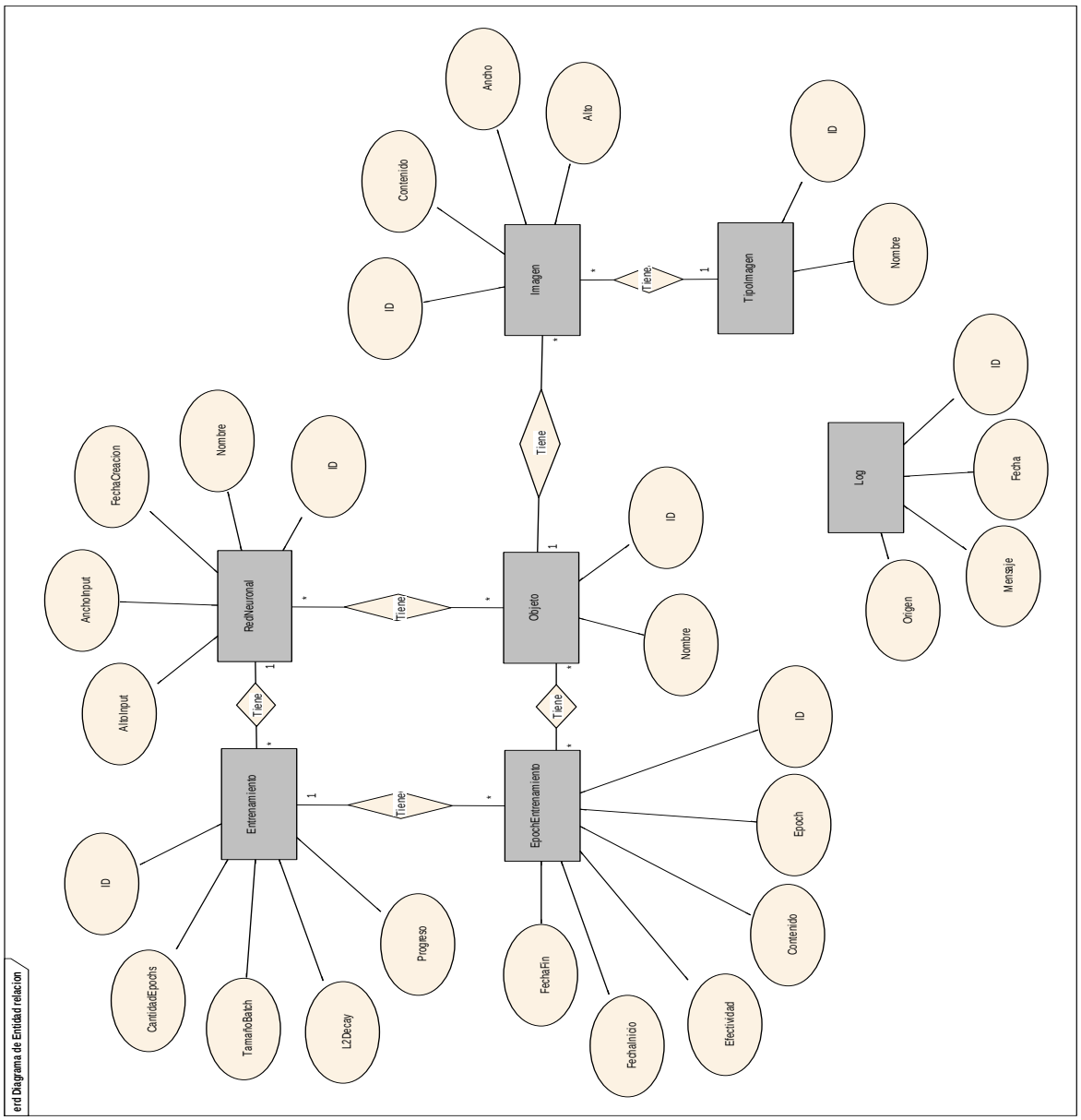
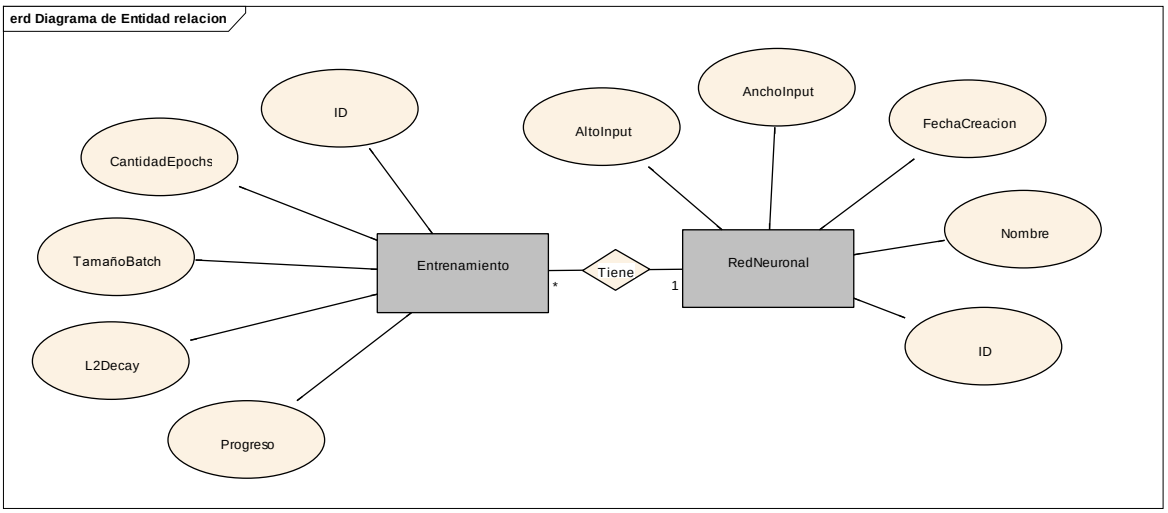

class CNN
ParametersAndGradients
- Gradients: double[]
- L1DecayMul: double
- L2DecayMul: double
- Parameters: double[]
Volume
- Depth: int
- Height: int
- WeightGradients: double[]
- Weights: double[]
- Width: int
+ Add(double, int, int, int) : void
+ AddFrom(Volume) : void
+ AddFromScaled(double, Volume) : void
+ AddGradient(double, int, int, int) : void
+ AddGradientFrom(Volume) : void
+ Clone() : Volume
+ CloneAndZero() : Volume
+ Get(double, int, int) : double
+ GetGradient(int, int, int) : double
+ Set(double, int, int, int) : void
+ SetConst(double) : void
+ SetGradient(double, int, int, int) : void
+ Volume(int, int, int)
+ Volume(double, int, int, int)
+ Volume(List<double>)
Net
- Layers: List<LayerBase>
+ AddLayer(LayerBase) : void
+ Backward(double) : double
+ Backward(double[]) : double
+ Forward(boolean, Volume) : Volume
+ GetCostLoss(double, Volume) : double
+ GetCostLoss(double[], Volume) : double
+ GetParametersAndGradients() : List<ParametersAndGradients>
+ GetPrediction() : int
PoolLayer
- pad: int
- stride: int
- switchx: int[]
- switchy: int[]
+ Backward() : void
+ Forward(boolean, Volume) : Volume
+ Init(int, int, int) : void
ReluLayer
+ Backward() : void
+ Forward(boolean, Volume) : Volume
+ Init(int, int, int) : void
SigmoidLayer
+ Backward() : void
+ Forward(boolean, Volume) : Volume
+ Init(int, int, int) : void
Trainer
- BackwardTime: TimeSpan
- BatchSize: int
- CostLoss: double
- Eps: double
- ForwardTime: TimeSpan
- gsum: List<double[]>
- k: int
- L1DecayLoss: double
- L2Decay: double
- L2DecayLoss: double
- LearningRate: double
- Momentum: double
- net: Net
- Ro: double
- TrainingMethod: Method
- xsum: List<double[]>
+ Backward(double) : void
+ Backward(double[]) : void
+ Forward(Volume) : void
+ Loss() : double
+ Train(double, Volume) : void
+ Train(double[], Volume) : void
+ TrainImplem() : void
LayerBase
- DropProb: int
- Height: int
- InputActivation: Volume
- InputDepth: int
- InputHeight: int
- InputWidth: int
- OutputActivation: Volume
- OutputDepth: int
- OutputHeight: int
- Width: int
+ Backward() : void
+ Forward(boolean, Volume) : Volume
+ GetParametersAndGradients() : List<ParametersAndGradients>
+ Init(int, int, int) : void
InputLayer
+ Forward(boolean, Volume) : Volume
+ InputLayer(int, int, int)
FullyConnLayer
- Activation: Activation
- Biases: Volume
- BiasPref: double
- Filters: List<Volume>
- GroupSize: int
- InputCount: int
- L1DecayMul: double
- L2DecayMul: double
- NeuronCount: int
+ Backward() : void
+ Forward(boolean, Volume) : Volume
+ FullyConnLayer(Activation, int)
+ GetParametersAndGradients() : List<ParametersAndGradients>
+ Init(int, int, int) : void
ConvLayer
- Biases: Volume
- FilterCount: int
- Filters: List<Volume>
- GroupSize: int
- L1DecayMul: double
- L2DecayMul: double
- Pad: int
- Stride: int
+ Backward() : void
+ Forward(boolean, Volume) : Volume
+ GetParametersAndGradients() : List<ParametersAndGradients>
+ Init(int, int, int) : void
IDotProductLayer
- Activation: Activation
- BiasPref: double
- GroupSize: int

Diagrama de entidad relación





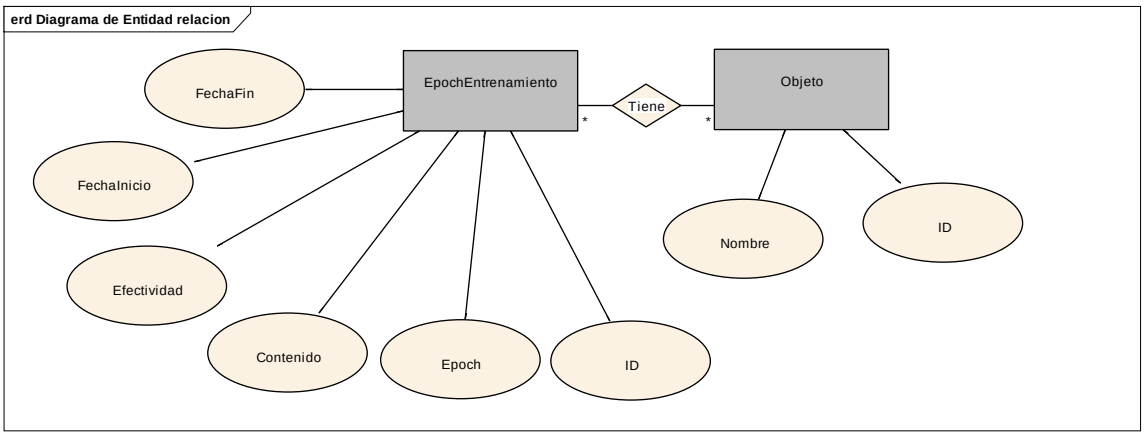
La entidad “RedNeuronal” representa la identificación y arquitectura de una red neuronal creada por el sistema. Se pueden crear muchas entidades de “Entrenamiento” relacionadas a una misma red neuronal. Esto permite entrenar una red neuronal con distintos parámetros de entrenamiento para lograr encontrar aquel que logre una mayor efectividad.

RedNeuronal

- **ID**: Valor auto numérico que identifica unívocamente a cada red neuronal
- **Nombre**: Nombre descriptivo de la red neuronal. Este nombre es establecido por el usuario
- **FechaCreacion**: Fecha en la que se creó la red neuronal
- **AnchoInput**: Establece el ancho de las imágenes que acepta como entrada la red neuronal
- **AltoInput**: Establece el alto de las imágenes que acepto como entrada la red neuronal

Entrenamiento

- **ID:** Valor auto numérico que identifica unívocamente a cada entrenamiento
- **CantidadEpochs:** Cantidad de epochs (ciclos de entrenamiento) que se realizaran durante el entrenamiento.
- **TamañoBatch:** Cantidad de imágenes de entrenamiento que la red neuronal propagara juntas hacia adelante y hacia atrás de la misma.
- **L2Decay:** Indica el grado de penalización por el excesivo crecimiento de un peso de la red.
- **Progreso:** Indica el porcentaje de avance del entrenamiento



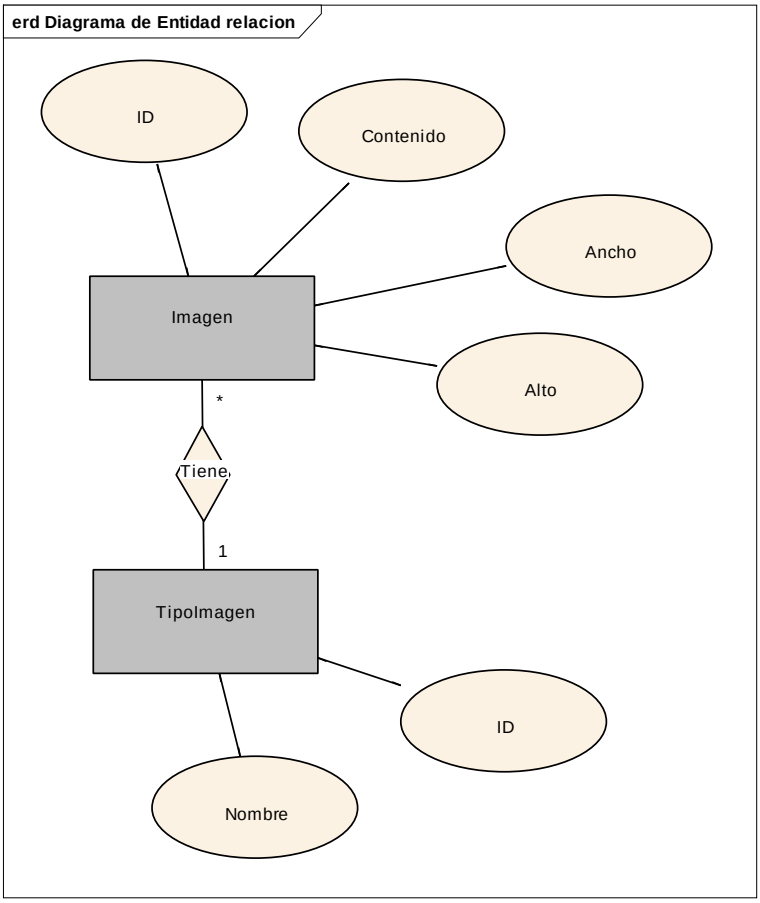
La entidad “Objeto” representa un objeto que pueda ser asociado a una red neuronal para que ésta identifique sus características y pueda reconocerlo en imágenes o videos. La entidad “EpochEntrenamiento” representa el estado de una red neuronal luego de haber pasado por un epoch de entrenamiento. Un “EpochEntrenamiento” se asocia con entidades “Objeto” para indicar el porcentaje de efectividad de reconocimiento de cada uno de los objetos.

EpochEntrenamiento

- **ID:** Valor auto numérico que identifica unívocamente a cada epoch
- **Epoch:** Numero de epoch que sirve de identificación dentro de un entrenamiento.
- **Contenido:** Red neuronal en formato binario
- **Efectividad:** Efectividad general de la red neuronal en este epoch de entrenamiento
- **FechaInicio:** Fecha en la que se inició el epoch de entrenamiento
- **FechaFin:** Fecha en la que finalizó el epoch de entrenamiento

Objeto

- **ID:** Valor auto numérico que identifica unívocamente a cada objeto
- **Nombre:** Nombre descriptivo del objeto



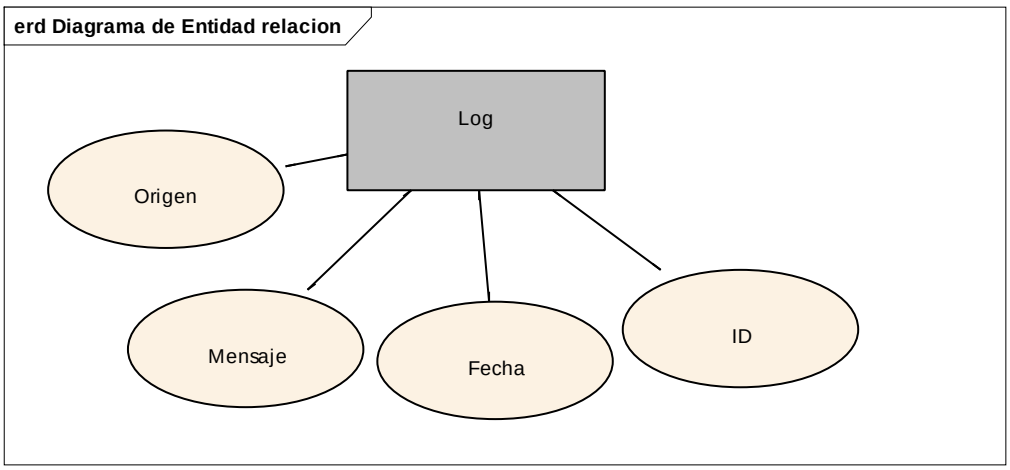
La entidad “Imagen” representa una imagen que puede ser entregada a una red neuronal durante la fase de entrenamiento o durante la fase de reconocimiento. La entidad “TipoImagen” indica si una imagen es de prueba o de entrenamiento.

Imagen

- **ID:** Valor auto numérico que identifica unívocamente a cada imagen
- **Contenido:** Imagen en formato binario
- **Ancho:** Ancho de la imagen
- **Alto:** Alto de la imagen

TipoImagen

- **ID:** Valor auto numérico que identifica unívocamente a cada tipo de imagen
- **Nombre:** Nombre descriptivo del tipo de imagen

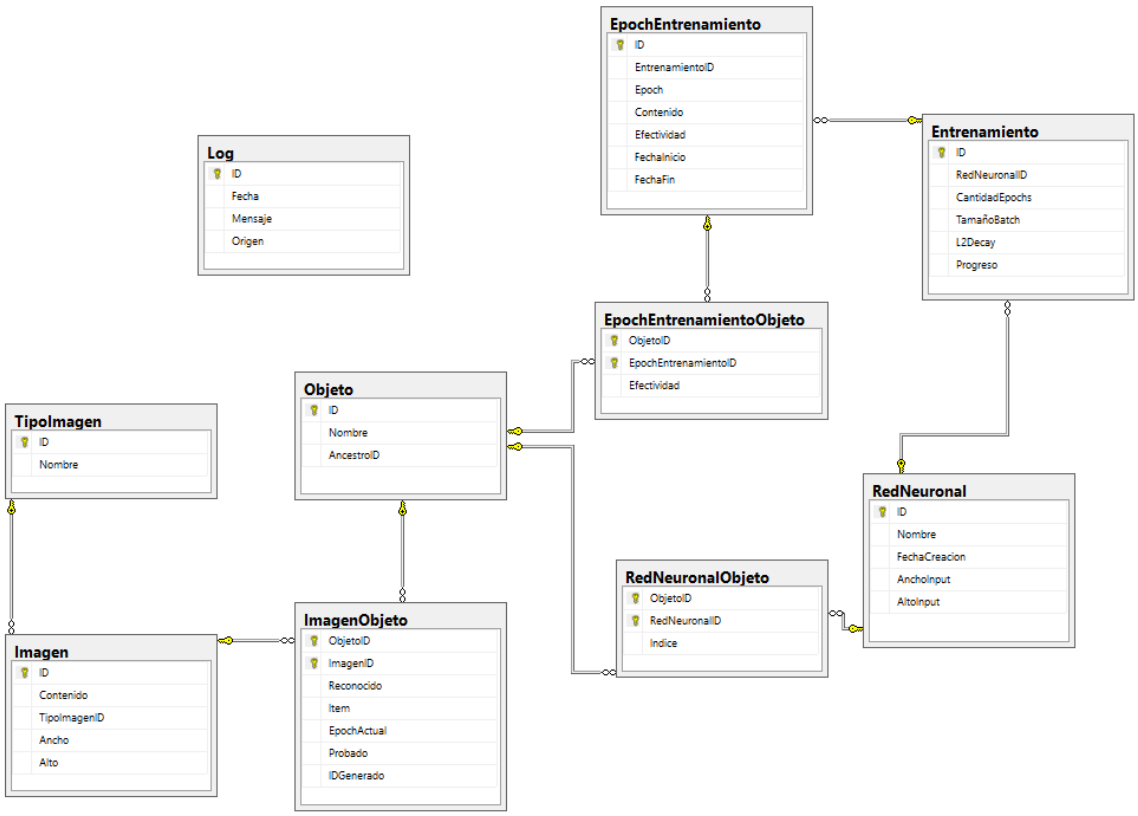


La entidad “Log” representa un mensaje de información de proceso que registra el sistema

Log

- **ID:** Valor auto numérico que identifica unívocamente a cada log
- **Origen:** Indica que proceso generó el log
- **Mensaje:** Mensaje registrado por el sistema
- **Fecha:** Fecha en la cual se emitió el log

Modelo relacional



Diccionario de datos

Tabla	Campo	Tipo de Dato	Descripción
Imagen	ID	Entero (Auto numérico)	Valor auto numérico que identifica unívocamente a cada imagen
	Contenido	Binario	Imagen en formato binario
	TipoImagenID	Entero	ID del tipo de imagen
	Ancho	Entero	Ancho de la imagen
	Alto	Entero	Alto de la Imagen
TipoImagen	ID	Entero (Auto numérico)	Valor auto numérico que identifica unívocamente a cada tipo de imagen
	Nombre	Alfa numérico	Nombre descriptivo del tipo de imagen
Objeto	ID	Entero (Auto numérico)	Valor auto numérico que identifica unívocamente a un objeto
	Nombre	Alfa numérico	Nombre descriptivo del objeto
	AncestroID	Entero	ID del objeto ancestro
ImagenObjeto	ObjetoID	Entero	ID del objeto
	ImagenID	Entero	ID de la imagen
	Reconocido	Booleano	Indica si la imagen que representa a un determinado objeto fue reconocida
	Item	Numérico	Representación de la relación Imagen-Objeto en un formato que la red neuronal puede procesar.
	EpochActual	Numérico	Indica cual fue el último epoch de entrenamiento por el cual paso la imagen
	Probado	Booleano	Indica si esta imagen fue probada por la red neuronal
	IDGenerado	Numérico	ID generado para facilitar la identificación durante el proceso de entrenamiento de una red neuronal
RedNeuronal	ID	Entero (Auto numérico)	Valor auto numérico que identifica unívocamente a una red neuronal
	Nombre	Alfa numérico	Nombre descriptivo de una red neuronal
	FechaCreacion	Fecha	Fecha en la que se creó la red neuronal
	AnchoInput	Entero	Ancho de las imágenes de entrada que soporta la red neuronal
	AltoInput	Entero	Alto de las imágenes de

			entrada que soporta la red neuronal
RedNeuronalObjeto	ObjetoID	Entero	ID del objeto
	RedNeuronalID	Entero	ID de la red neuronal
	Indice	Entero	Número que identifica cual es la neurona de salida que representa el objeto
Entrenamiento	ID	Entero (Auto numérico)	Valor auto numérico que identifica unívocamente a un entrenamiento
	RedNeuronalID	Entero	ID de la red neuronal
	CantidadEpochs	Entero	Cantidad de epochs (ciclos de entrenamiento) que se realizaran durante el entrenamiento
	TamañoBatch	Entero	Cantidad de imágenes de entrenamiento que la red neuronal propagara juntas hacia adelante y hacia atrás de la misma
	L2Decay	Decimal	Indica el grado de penalización por el excesivo crecimiento de un peso de la red
	Progreso	Decimal	Indica el porcentaje de avance del entrenamiento
EpochEntrenamientoObjeto	ObjetoID	Entero	ID del objeto
	EpochEntrenamientoID	Entero	ID del epoch de entrenamiento
	Efectividad	Decimal	Efectividad de reconocimiento del objeto en un epoch
EpochEntrenamiento	ID	Entero (Auto numérico)	Valor auto numérico que identifica unívocamente a un epoch de un entrenamiento
	EntrenamientoID	Entero	ID del entrenamiento
	Epoch	Entero	Numero de epoch que sirve de identificación dentro de un entrenamiento
	Contenido	Binario	Red neuronal en formato binario
	Efectividad	Decimal	Efectividad general de la red neuronal en este epoch de entrenamiento
	FechaInicio	Fecha	Fecha en la que se inició el epoch de un entrenamiento
	FechaFin	Fecha	Fecha en la que finalizó el epoch de un entrenamiento
Log	ID	Entero (Auto numérico)	Valor auto numérico que identifica unívocamente a un log

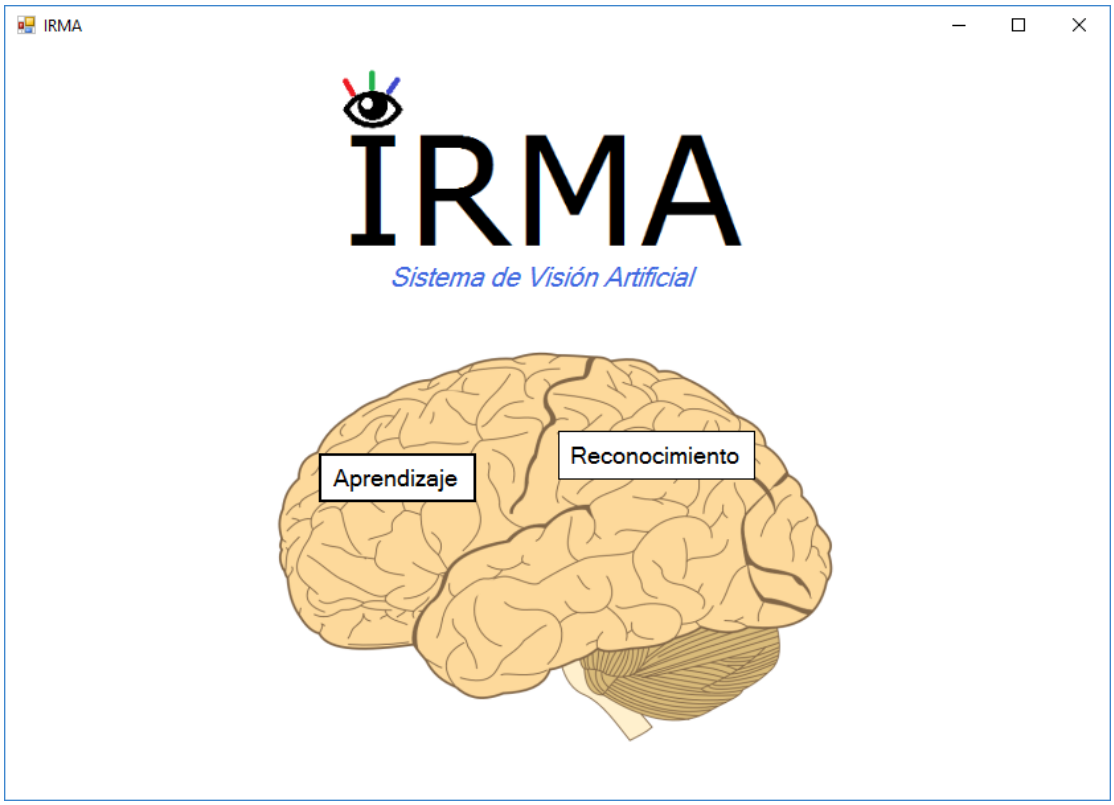
	Fecha	Fecha	Fecha en la cual se emitió el log
	Mensaje	Alfa numérico	Mensaje registrado por el sistema
	Origen	Alfa numérico	Indica que proceso genero el log

Aplicación

El objetivo de esta sección es mostrar las principales pantallas funcionales de la aplicación desarrollada.

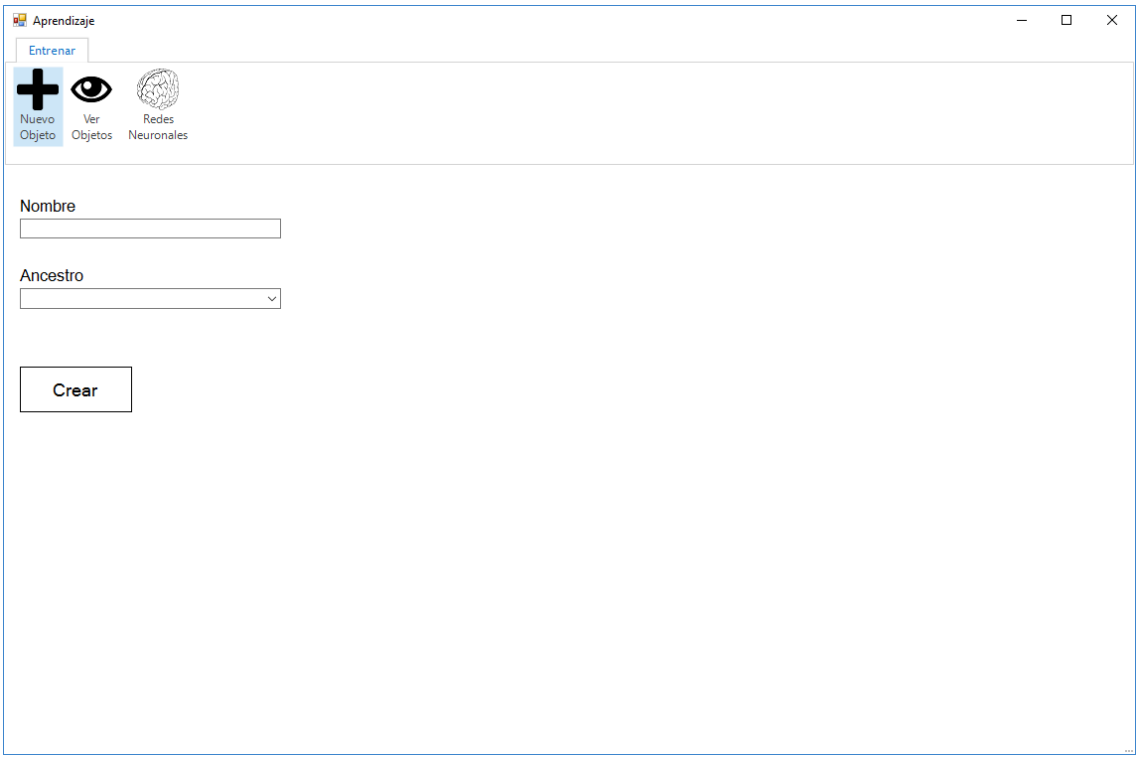
Inicio

Al ingresar a la aplicación se muestra una pantalla con 2 botones. Uno de los botones nos permite ingresar al módulo de aprendizaje donde se pueden definir los objetos y entrenar las redes neuronales. Y el otro botón nos permite ingresar al módulo de reconocimiento donde podemos cargar una imagen o video para que sea reconocido.



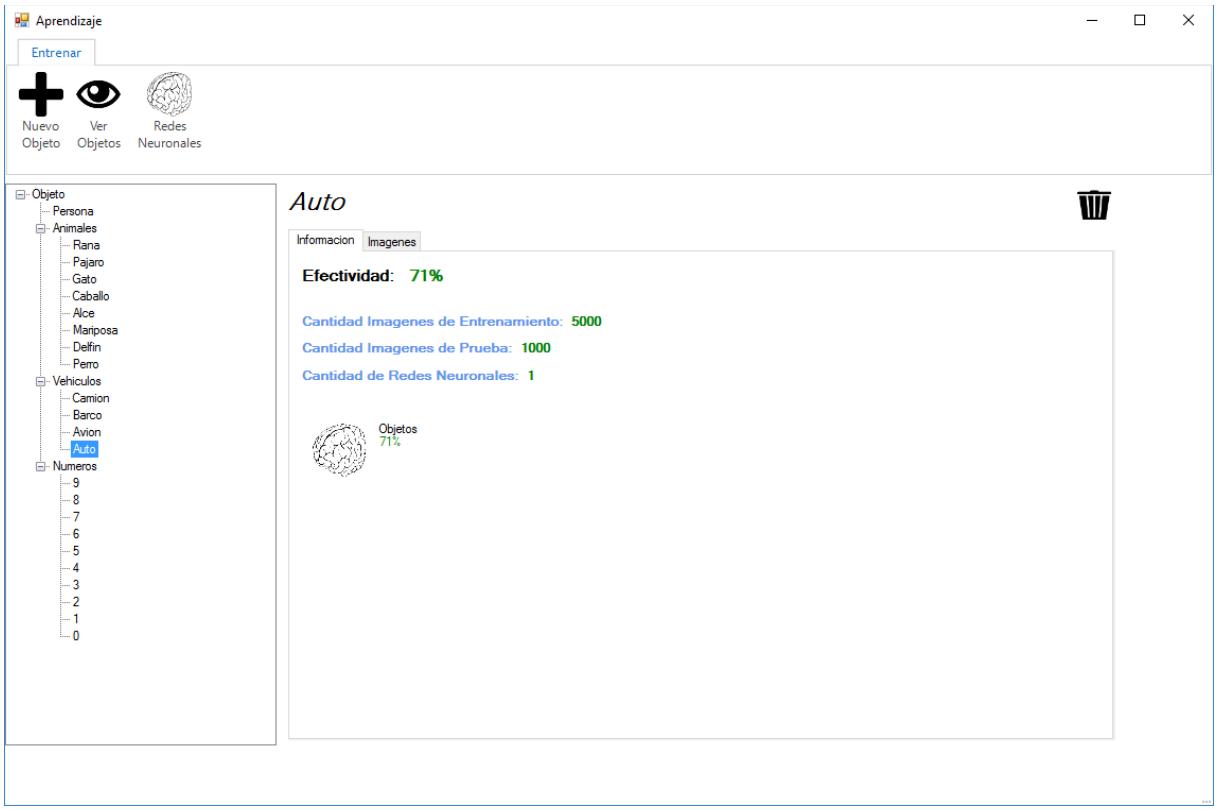
Crear objeto

Al hacer click en el botón “Nuevo objeto” que se encuentra en el módulo de aprendizaje, el sistema nos muestra una pantalla donde podemos crear un nuevo objeto definiendo su nombre y de que objeto hereda.

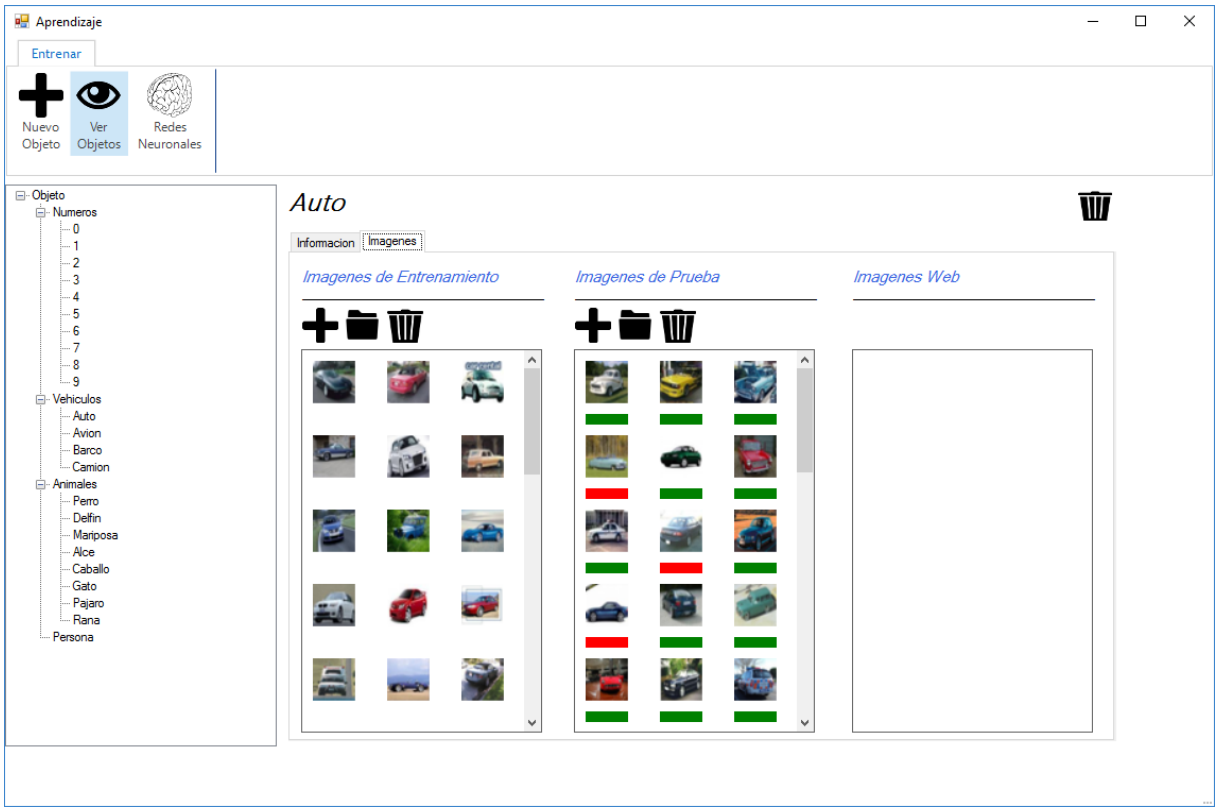


Ver objetos

Al hacer click en el botón “Ver objetos” que se encuentra en el módulo de aprendizaje, el sistema nos muestra una pantalla donde podemos ver todos los objetos creados por el usuario ordenados de acuerdo a la herencia entre los mismos. Al hacer click en un objeto, el sistema nos muestra las redes neuronales que pueden reconocerlo, la mayor efectividad de reconocimiento, la cantidad de imágenes de entrenamiento y de prueba y la cantidad de redes neuronales.

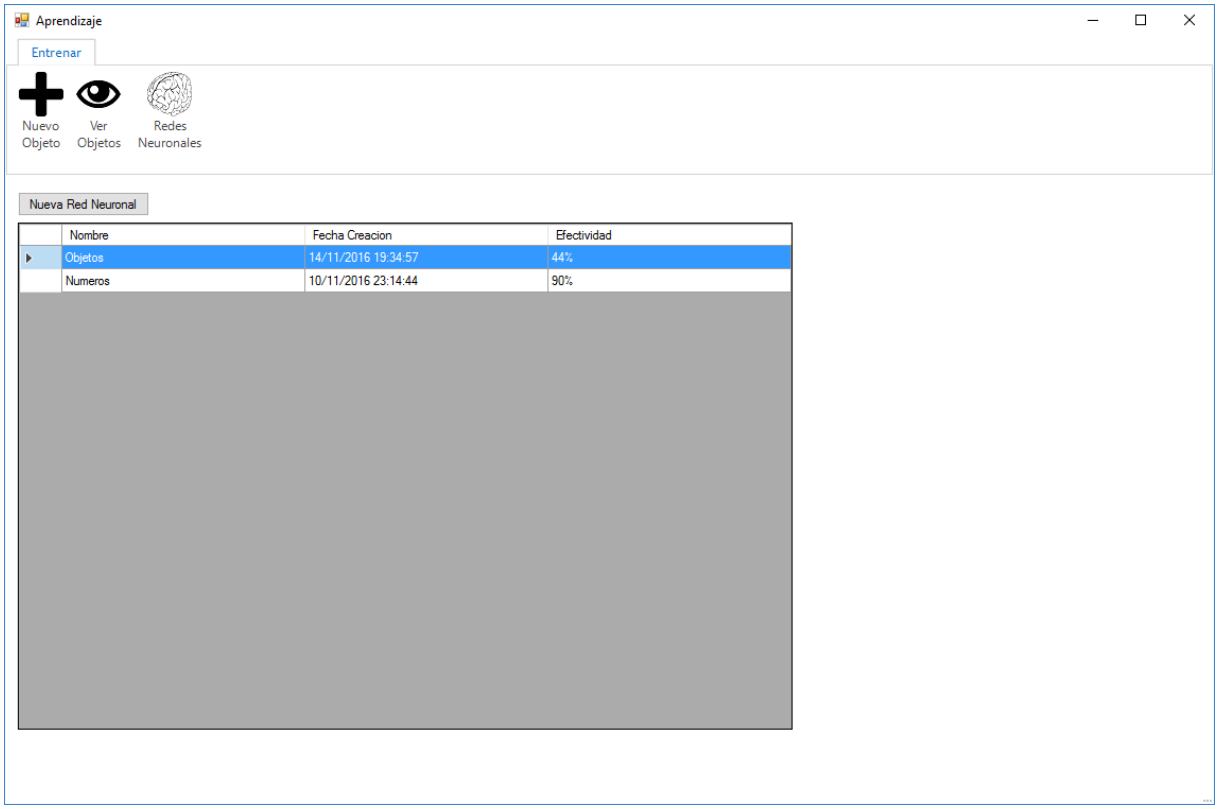


Al hacer click en la pestaña “Imágenes”, el sistema nos muestra las imágenes de entrenamiento y de prueba subidas. En esta pantalla se pueden agregar nuevas imágenes, ya sea individualmente o seleccionando una carpeta del sistema de archivos.



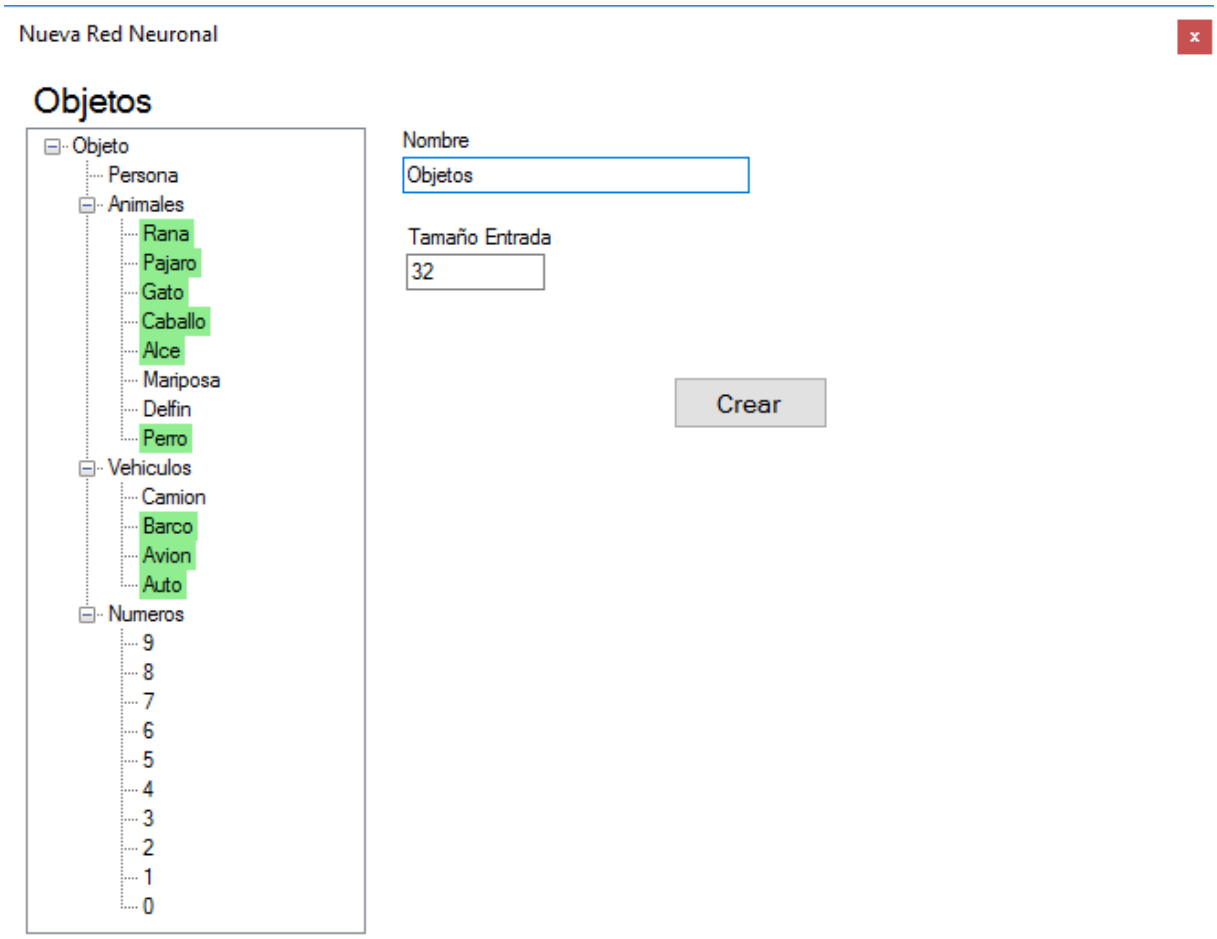
Redes neuronales

Al hacer click en el botón “Redes neuronales” que se encuentra en el módulo de aprendizaje, el sistema mostrara un listado de las redes neuronales creadas por el usuario indicando nombre, fecha de creación y efectividad de las mismas.



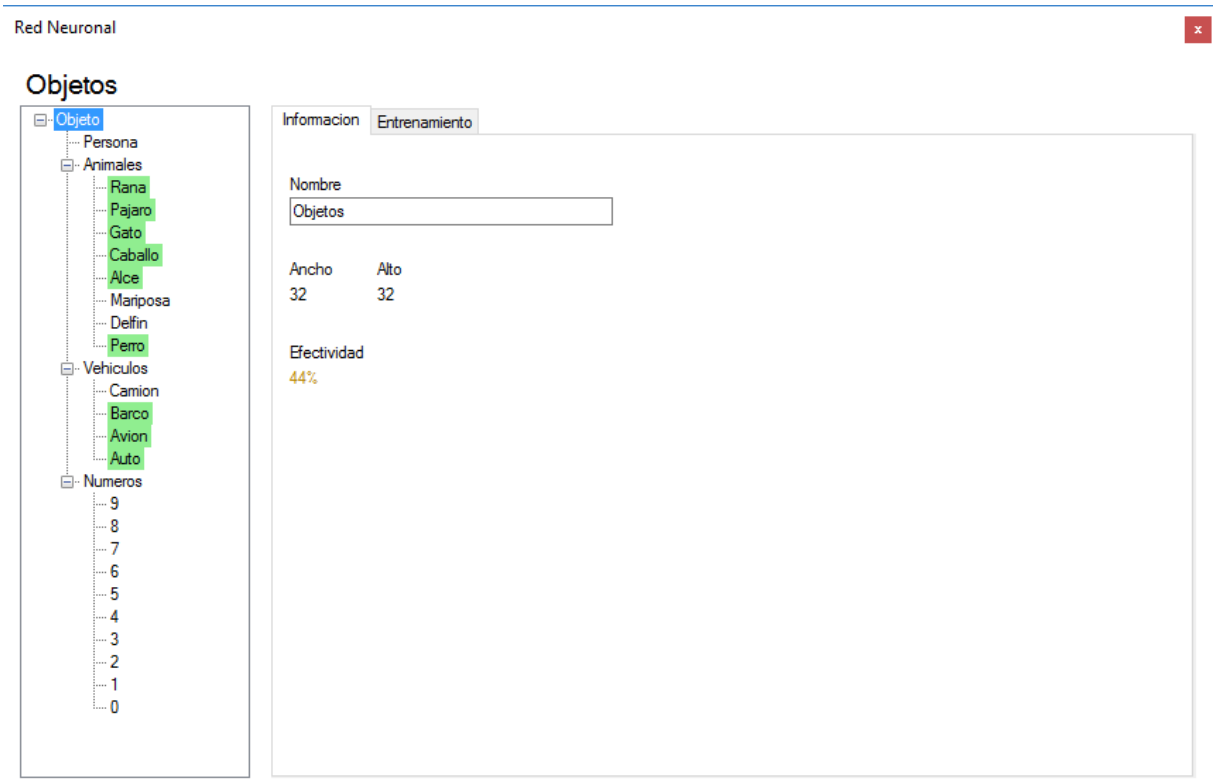
Crear red neuronal

Al hacer click en el botón “Nueva red neuronal” que se encuentra en la pantalla de “Redes neuronales”, el sistema nos pide que ingresemos el nombre de la nueva red neuronal, los objetos que tendrá que reconocer y el tamaño de las imágenes que podrá procesar.



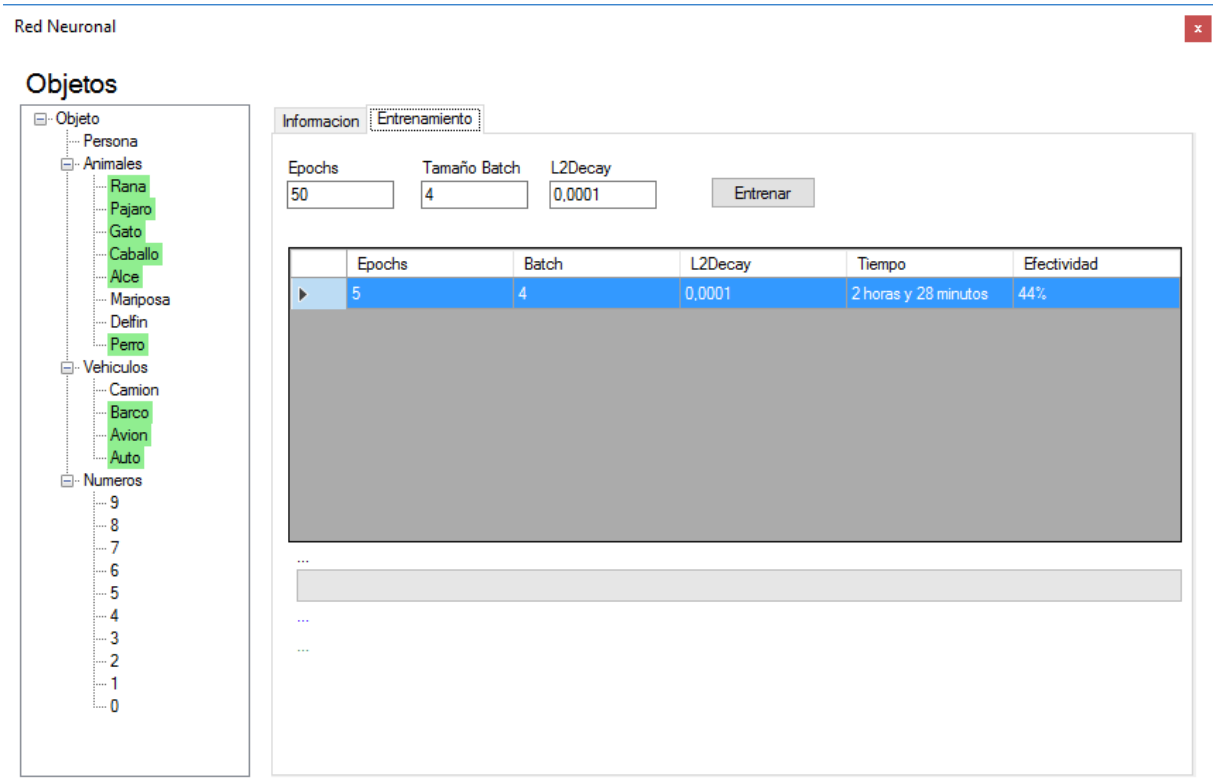
Red neuronal

Al hacer doble click en alguna de las redes neuronales de la lista de la pantalla “Redes neuronales”, el sistema mostrara el nombre de la misma, los objetos que puede reconocer, el tamaño de las imágenes que puede procesar y la efectividad de reconocimiento.



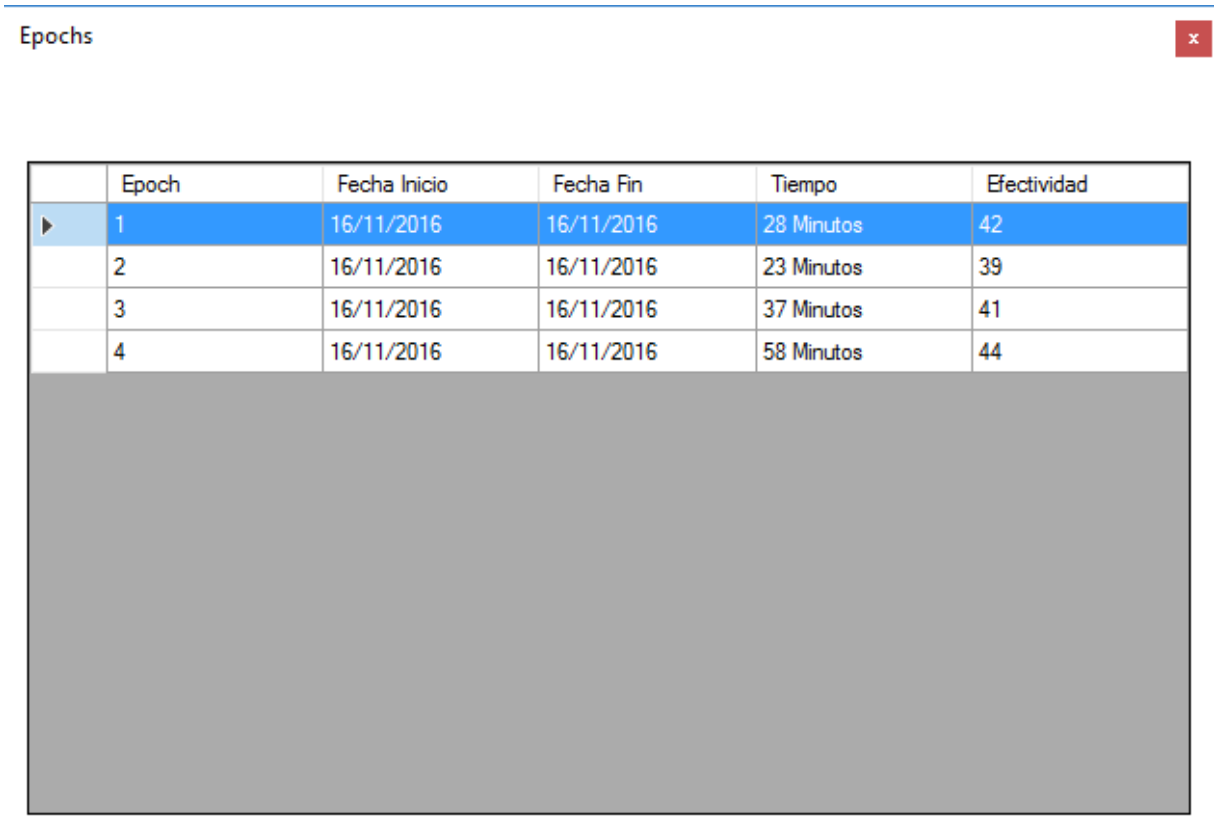
Entrenamientos

Al hacer click en la pestaña “Entrenamientos”, el sistema nos pedirá que ingresemos los parámetros con los cuales queremos entrenar a la red. Una vez que ingresamos los parametros, podemos hacer click en el botón “Entrenar” para que el sistema comience a entrenar la red neuronal.



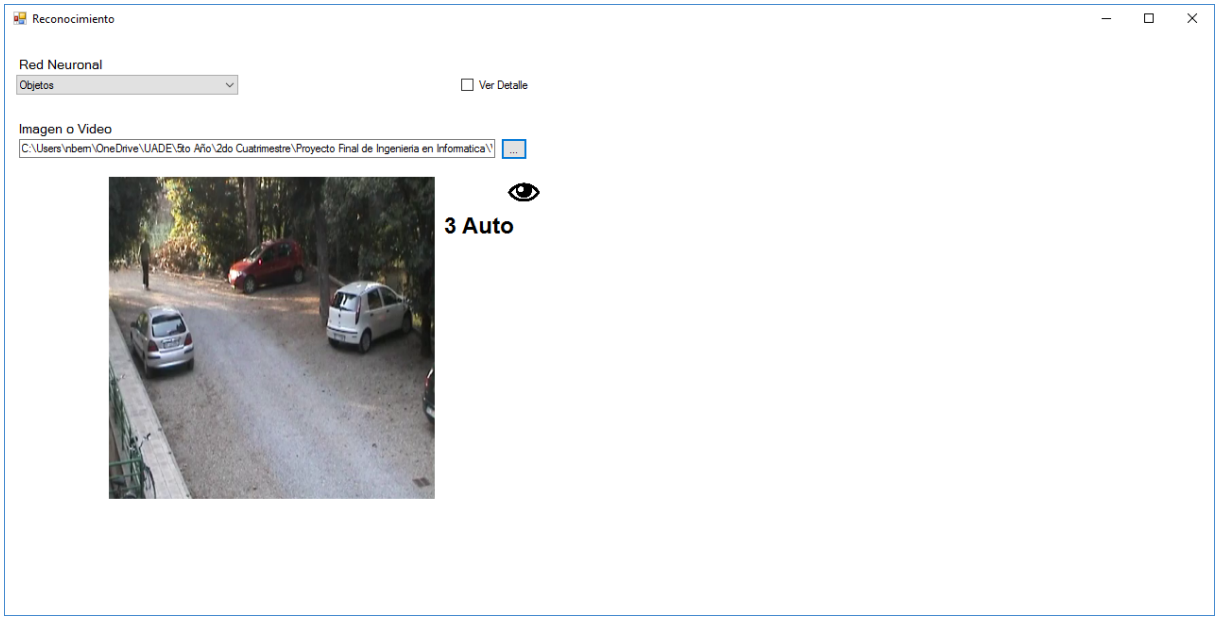
Epochs de Entrenamiento

Al hacer doble click en alguno de los entrenamientos creados, el sistema mostrara los epochs de ese entrenamiento, indicando número de epoch, fecha en la que se inició, fecha en la que finalizó, tiempo que tardo en completarse y efectividad de reconocimiento.



Reconocimiento

Al hacer click en el boton “Reconocimiento” que se encuentra en la pantalla inicial, el sistema nos pide que seleccionemos que red neuronal queremos usar para el reconocimiento y la imagen o el video que queremos reconocer.



Conclusiones

Luego de la investigación de distintos algoritmos de reconocimiento de patrones y segmentación de imágenes, se logró construir una aplicación capaz de identificar objetos en videos en tiempo real. La efectividad en la detección de estos objetos depende en gran medida de la cantidad de imágenes de entrenamiento que se le entreguen a la aplicación para entrenar la red neuronal que finalmente se encargara de realizar el reconocimiento de objetos en los videos entregados por el usuario. Por otro lado, se encontró que la efectividad final no solo depende de la cantidad de imágenes sino también de la calidad de las mismas. Las imágenes que aportan a que se logre una mayor efectividad son aquellas que contienen al objeto que representan como elemento central de la misma con la menor cantidad de otros objetos que puedan formar parte del entorno. En la etapa de reconocimiento de los videos, para el caso de los objetos que se encuentren estáticos en el mismo, la efectividad se puede ver afectada por el contraste entre estos y el entorno. Cuanto mayor sea el contraste mejor funcionara el algoritmo de segmentación. Para el caso de los objetos que se encuentren en movimiento, la segmentación se realizara con mayor facilidad gracias a la comparación de fotogramas.

La aplicación construida está adaptada principalmente a la detección de objetos en videos provenientes de cámaras de seguridad donde normalmente la cámara permanece estática o realiza leves movimientos. Sin embargo, realizando algunas modificaciones en la forma en la cual se realiza el proceso de segmentación en los fotogramas, se puede adaptar a otros tipos de videos.

A partir de esta investigación, se pueden realizar futuros trabajos para la clasificación de imágenes o videos por el color que predomina. También se podría realizar una clasificación por tipo teniendo en cuenta que clases de objetos intervienen en el mismo o los comportamientos detectados.

Anexos

Anexo I - Regla de la cadena

Dada una función $f(x)$

$f(x) = \cos^3 x$

Si llamamos a la función $\cos x$ como $u(x)$ y a la potencia como v

$f(x) = v(u(x))$

La derivada de esta función será

$$f'(x) = v'\big(u(x)\big).u'(x) = \frac{\partial v}{\partial u}.\frac{\partial u}{\partial x} = \frac{\partial \cos^3 x}{\partial cosx}.\frac{\partial cosx}{\partial x} = -3\sin x\ .\cos^2 x$$

Ejemplo:

$f(x) = \sin^2 x$

$$\frac{\partial f}{\partial x} = \frac{\partial \sin^2 x}{\partial sinx}.\frac{\partial sinx}{\partial x}$$

$$\frac{\partial f}{\partial x} = 2\sin x\ .\cos x$$

Anexo II - Producto convolucion

El producto convolucion se utiliza para aplicar una transformación a una imagen. Se utiliza una matriz llamada Kernel la cual recorre cada pixel de la imagen. Al tomar un pixel de la imagen (pixel inicial), éste y sus pixeles circundantes son multiplicados por cada valor correspondiente en el Kernel. Los resultados de estas multiplicaciones se suman y este nuevo resultado es reemplazado por el valor del pixel inicial.

Ejemplo de cálculo de un pixel

11	4	5	2
16	15	21	3
18	1	3	4
6	9	8	7

Imagen

4	5	1	2
1	3	2	3
6	2	1	4
6	9	8	7

Kernel sobre pixel inicial

11	4	5	2
16	285	21	3
18	1	3	4
6	9	8	7

Resultado

$$Resultado = (11 * 4) + (4 * 5) + (5 * 1) + (16 * 1) + (15 * 3) + (21 * 2) + (18 * 6) + (1 * 2) + (3 * 1) = 285$$

Anexo III - Canny

El algoritmo de canny [5] se utiliza para la detección de bordes en una imagen. La primera etapa del algoritmo consiste en aplicar un filtro gaussiano a la imagen para reducir el ruido presente en la misma. Para aplicar el filtro se utiliza una matriz llamada Kernel con la cual se realiza la operación de convolucion sobre la imagen (Ver Anexo II – Producto convolucion).

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Ejemplo de filtro Gaussiano (Kernel)

La segunda etapa consiste en aplicar el operador de Sobel a la imagen. Este operador se aplica para encontrar la magnitud del gradiente en cada pixel. Para aplicar este operador se utilizan 2 kernels.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_x \qquad\qquad G_y$$

Uno de los kernels es igual al otro rotado 90º. Para encontrar la magnitud absoluta del gradiente se combinan los resultados de aplicar los 2 kernels.

$$|G| = \sqrt{G_x^2 + G_y^2}$$

Para mejorar el rendimiento se suele utilizar la siguiente aproximación.

$$|G| = |G_x| + |G_y|$$

[5] **(5) Kim, Daniel. 2013.** Sobel Operator and Canny Edge Detector ECE 480 Fall 2013

Se calcula la orientación del gradiente de la siguiente forma.

$$\boldsymbol{\theta = \arctan\left(\frac{G_y}{G_x}\right)}$$

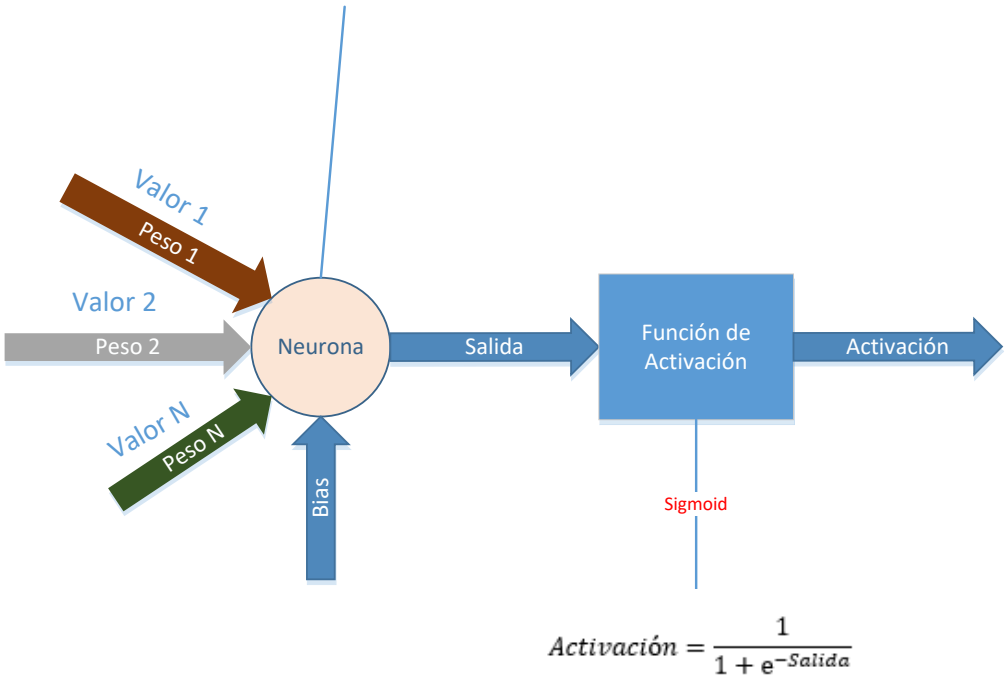
Una vez calculado el gradiente de la imagen, se determina si cada magnitud del gradiente corresponde a un máximo local en la dirección del gradiente.

- Si el ángulo del gradiente es 0° (Dirección Norte-Sur), se considera que el punto está en un borde si la magnitud del gradiente en ese punto es mayor que las magnitudes en las direcciones este-oeste.
- Si el ángulo del gradiente es 90° (Dirección Este-Oeste), se considera que el punto está en un borde si la magnitud del gradiente en ese punto es mayor que las magnitudes en las direcciones norte-sur.
- Si el ángulo del gradiente es 135° (Dirección Noreste y Sudoeste), se considera que el punto está en un borde si la magnitud del gradiente en ese punto es mayor que las magnitudes en las direcciones noroeste y sudeste.
- Si el ángulo del gradiente es 45° (Dirección noroeste y sudeste), se considera que el punto está en un borde si la magnitud del gradiente en ese punto es mayor que las magnitudes en las direcciones noroeste y sudeste.

Luego de esta etapa se logra obtener los puntos que corresponden a los bordes. El último paso consiste en quedarse con los puntos de mayor intensidad. Se asume que los bordes importantes corresponden a líneas continuas. En esta última etapa se utiliza un umbral mínimo y máximo para determinar qué puntos se dejarán y cuales se descartarán.

Anexo IV - Neurona

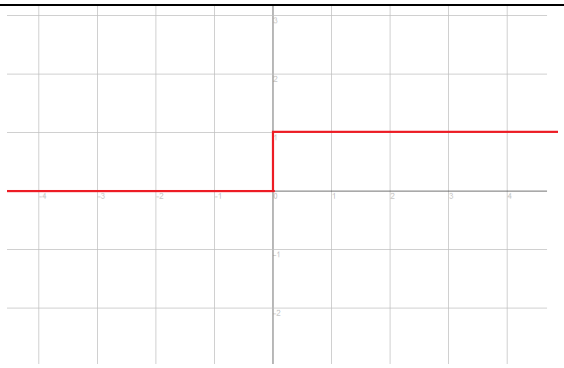
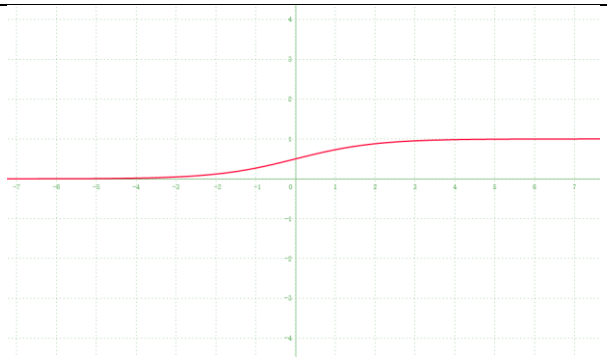
$$Salida = \left(\sum_{n=1}^{n=N} Peso_n \,.\, Valor_n\right) + Bias$$



$$Activación = \frac{1}{1 + e^{-Salida}}$$

A excepción de las neuronas que se encuentran en la capa de entrada, cada neurona de la red tiene como entrada la salida de una o más neuronas. La conexión entre una neurona y otra se llama **sinapsis.** La salida de una neurona se calcula como la sumatoria de los valores de entrada multiplicados por el peso asociado a esa entrada más un valor llamado **bias**. Este valor de salida se pasa como parámetro a una **función de activación** la cual puede devolver un valor discreto (0,1) o un valor continuo. De esta forma, se activarán distintas neuronas de acuerdo a la imagen que se le entregue a la red y a los pesos de las distintas conexiones sinápticas.

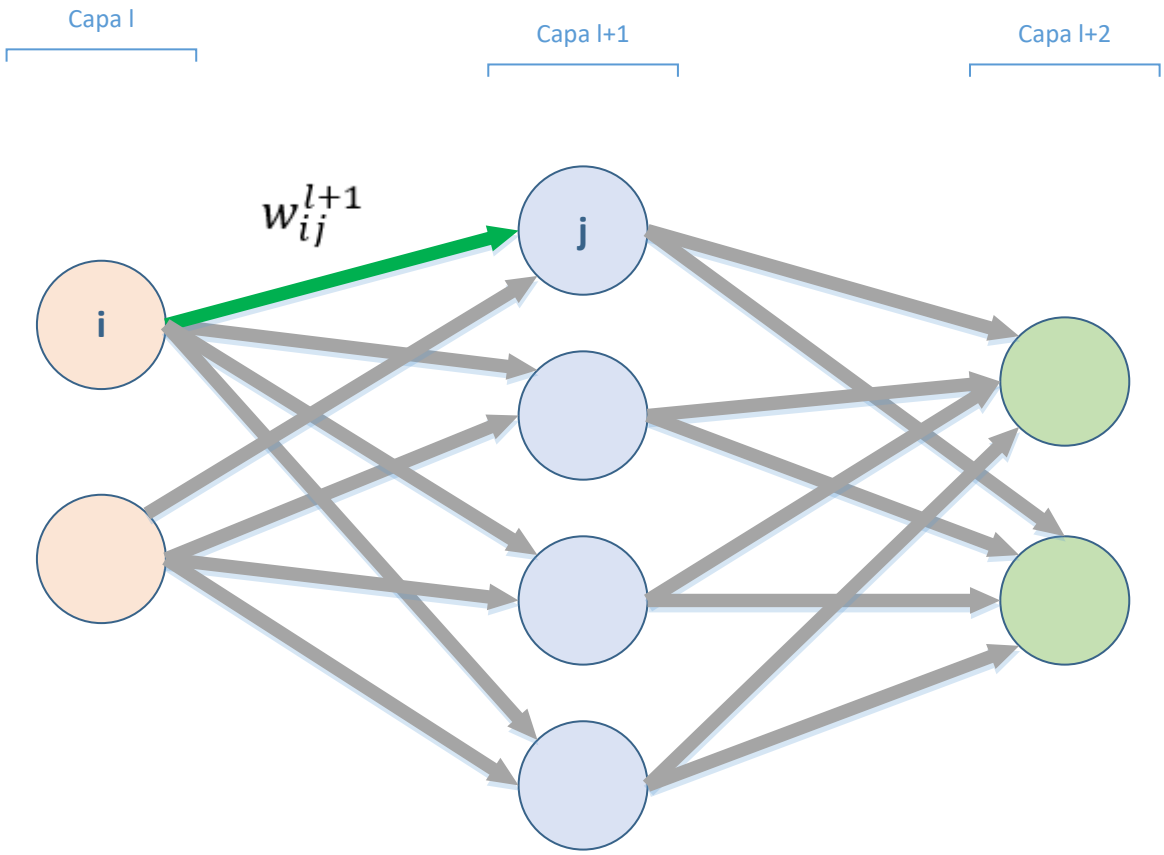
Anexo V - Funciones de activación

Binaria		$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$
Sigmoid		$f(x) = \frac{1}{1 + e^{-x}}$

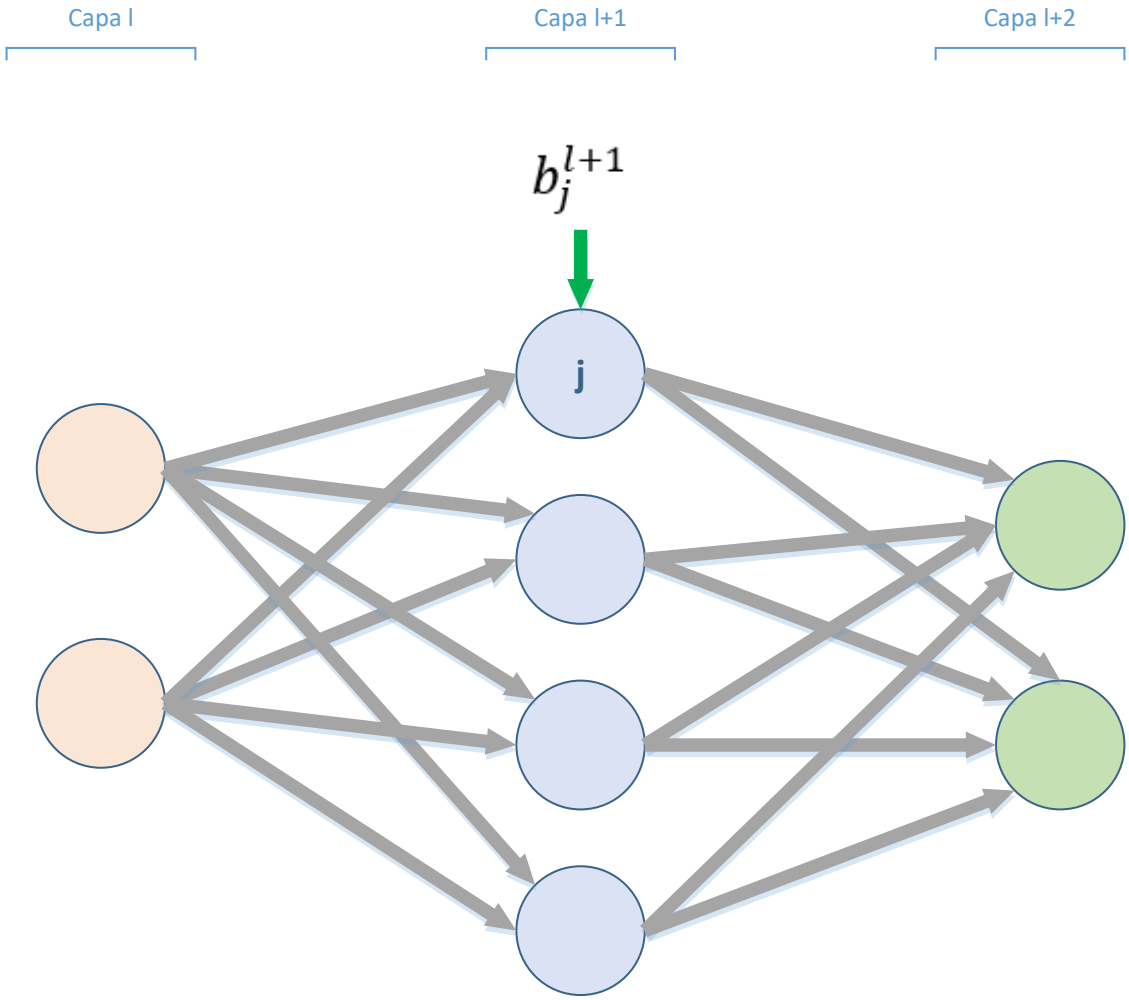
El problema de utilizar una función de activación que devuelva valores discretos, como la función binaria, es que cualquier cambio en los datos provocara cambios bruscos en la salida. En cambio, una función de activación como la función sigmoid provocara pequeños cambios.

Anexo VI - Notación

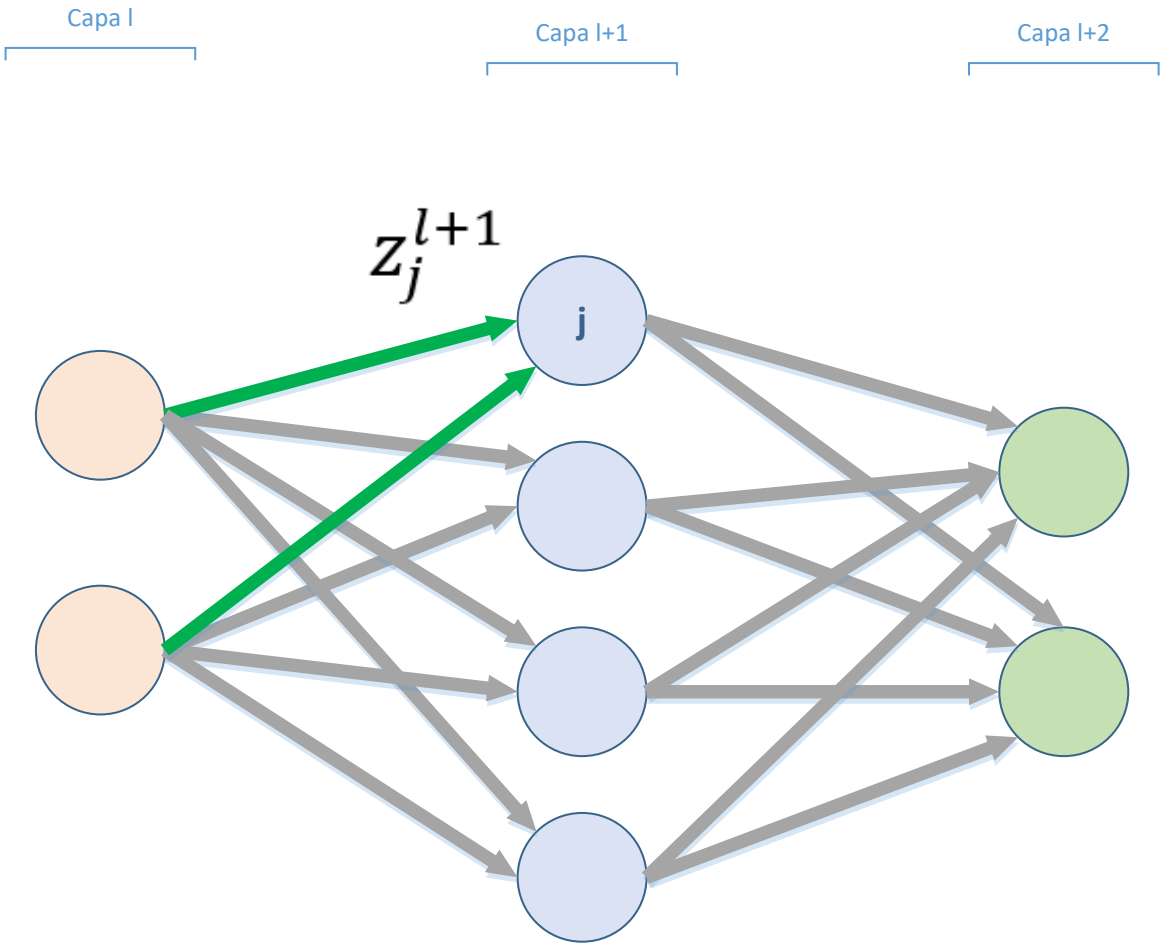
Al peso de la conexión sináptica entre la neurona **i** de la capa l y la neurona **j** de la capa l+1 se lo llama w_{ij}^{l+1}



Al bias de la neurona **j** de la capa l+1 se lo llama b_j^{l+1}



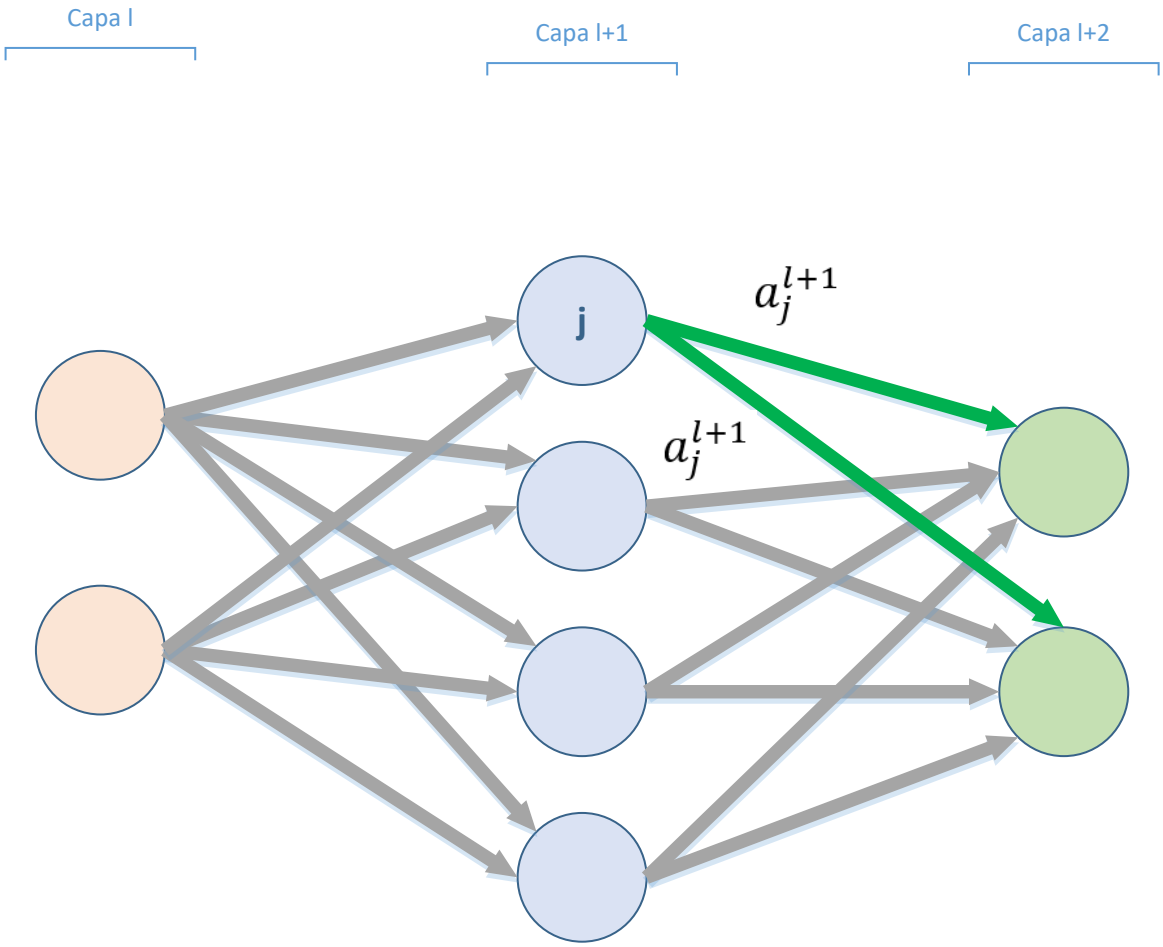
A la entrada de una neurona **j** de la capa l+1 se la llama z_j^{l+1}



$$z_j^{l+1} = \sum_i w_{ij}^{l+1} \cdot a_i^l + b_j^{l+1}$$

$$z^{l+1} = (w^{l+1})^T \, a^l + b^{l+1}$$

A la activación de una neurona **j** en la capa l+1 se la llama a_j^{l+1}

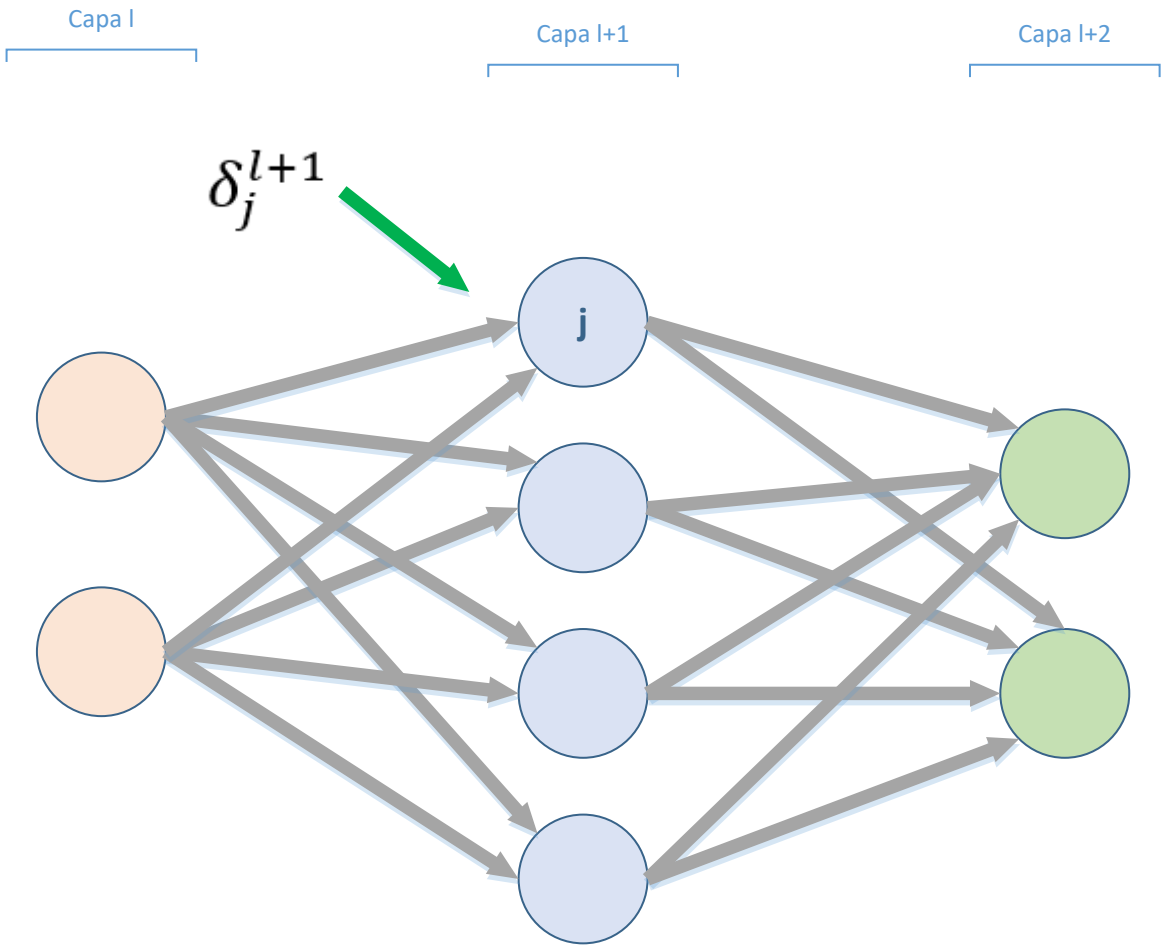


$$a_j^l = \sigma(z_j^l)$$

$$a^l = \sigma(z^l)$$

Siendo σ la función de activación

Al error introducido en una neurona **j** de la capa l+1 utilizado en el algoritmo "Backpropagation" se lo llama δ_j^{l+1}



Anexo VII - Gradient descent

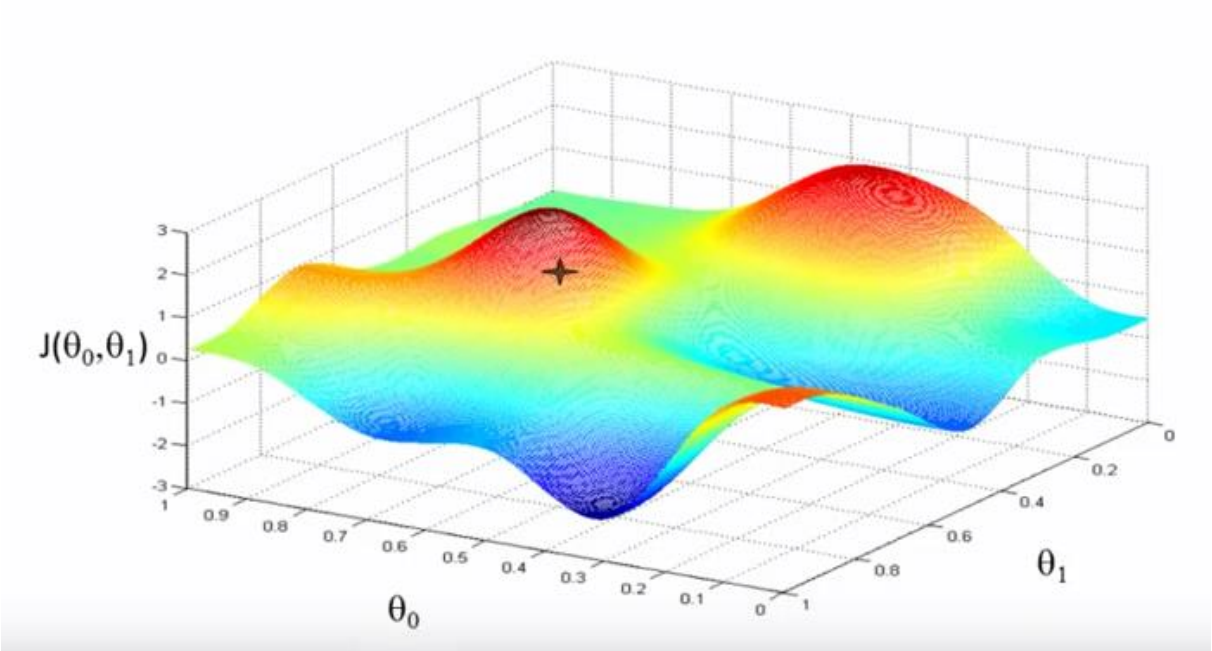
Cuando se crea la red, se inicializan los pesos de las conexiones de manera aleatoria. Estos pesos son ajustados en la fase de entrenamiento tratando de minimizar una función de costo como puede ser la función de costo cuadrática

$$C(w,b) = \frac{1}{2n}\sum_{x}\|y(x) - a\|^2$$

Donde **w** es la colección de pesos de toda la red. **b** son todos los bias de la red. **n** es el número total de entradas de entrenamiento. **a** es el vector de salidas de la red cuando x es la entrada. **y(x)** es un vector que representa la salida deseada respecto a las entradas. La función de costo nos indica la efectividad de reconocimiento de la red de las imágenes de entrenamiento. Esta función tiene como variables los pesos y los bias de las conexiones sinápticas de la red y su resultado será siempre positivo. Se busca encontrar el conjunto de pesos y de bias que minimicen esta función de costo. La razón por la cual se utiliza una función de costo para medir la efectividad de la red neuronal en lugar de simplemente contar la cantidad de imágenes reconocidas se debe a que pequeños cambios en los pesos o en los bias pueden mejorar la efectividad de la red sin que cambie la cantidad de imágenes reconocidas del set de entrenamiento.

Para minimizar esta función se utiliza el método de optimización llamado "Gradient Descent". Este método consiste en encontrar cuales son los pequeños cambios que tenemos que hacer en los parámetros para lograr que la función decremente. Para lograr esto, se calculan las derivadas parciales de la función respecto a cada una de las variables y se procede a actualizar los parámetros de la función en el sentido en el cual la función decrementa utilizando una tasa de aprendizaje. De esta forma se intenta llegar al mínimo global de la función que establece los valores que deben tomar los parámetros (pesos y bias) para que la función de costo tome el mínimo valor. La tasa de aprendizaje establece que tan rápido se va a avanzar en tal sentido. Si la tasa fijada es muy grande podemos pasar de largo el mínimo global debiendo dar por terminado el método en algún otro valor. Por el contrario, si la tasa de aprendizaje es muy chica, puede que el método le tome demasiado tiempo llegar a

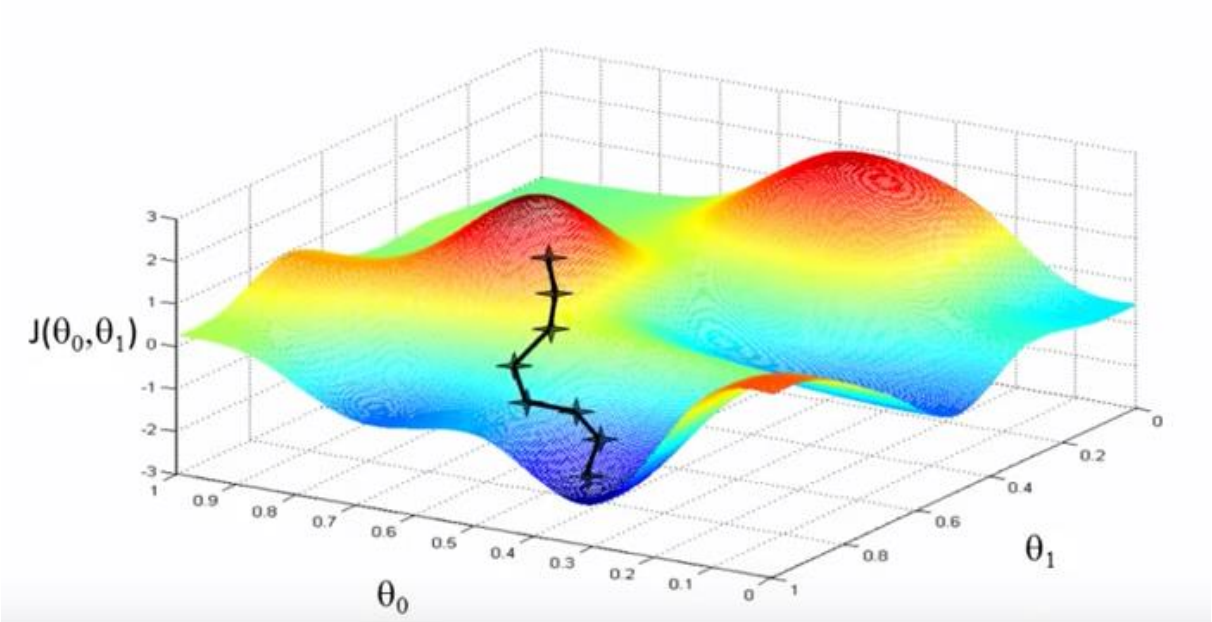
encontrar el mínimo global. Debido a que la función de costo para la red neuronal posee muchas variables (Cada peso de las conexiones sinápticas y bias de cada neurona) se procederá a explicar el algoritmo con una función de solo 2 variables.



Supongamos una función J de 2 variables (θ_0, θ_1). El algoritmo consiste en ir buscando valores de θ_0 y θ_1 que hagan que el resultado de la función sea cada vez menor hasta encontrar el mínimo. Lo primero que se debe hacer es tomar un punto cualquiera de la función y calcular las derivadas parciales respecto a cada una de las variables en este punto. La derivada parcial respecto a una variable nos dará como información si la función está creciendo o decreciendo en esa dimensión. Con esta información se procede a actualizar el valor de la variable de manera tal que la función tome un valor cada vez menor en cada iteración. El valor de la derivada y el parámetro α llamado "Tasa de aprendizaje" determinaran que tan rápido se avanza hacia el mínimo de la función.

$$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$



La elección de la tasa de aprendizaje es muy importante para el correcto funcionamiento de algoritmo. Si el valor de la misma es muy bajo, la actualización de los valores de las variables será muy pequeña en cada iteración lo que resultará en un lento funcionamiento del algoritmo. Pero si por otro lado el valor de la tasa de aprendizaje es muy alto, las variables tomaran valores muy lejanos entre cada iteración lo que puede resultar en no pasar por el mínimo quedando éste entre 2 iteraciones.

El algoritmo "Gradient descent" aplicado a la función de costo de la red neuronal quedaría de la siguiente forma:

Por cada peso:

$$w_k := w_k - \alpha \frac{\partial C}{\partial w_k}$$

Por cada bias:

$$b_i := b_i - \alpha \frac{\partial C}{\partial b_i}$$

Debido a que este algoritmo tardaría mucho en ejecutarse debido a la cantidad de parámetros sobre los cuales tiene que derivar la función, se utiliza una variación de este

algoritmo llamado “Stochastic gradient descent”. Este algoritmo consiste en hacer los cálculos usando solo un grupo de las imágenes de entrenamiento elegidas al azar. A este grupo de imágenes se lo llama “mini-batch”. Una vez finalizado los cálculos con ese mini-batch se toma otro “mini-batch” hasta que ya no queden imágenes de entrenamiento. A esto se lo llama “epoch”. El entrenamiento de una red neuronal es llamado “online” cuando el tamaño del “mini-batch” es de 1.

Anexo VIII - Backpropagation

Para calcular las derivadas parciales respecto a cada uno de los pesos y bias de la función de costo, se utiliza un algoritmo llamado "Backpropagation" [6].

La función de costo puede escribirse como:

$$C = \frac{1}{n}\sum_{x} C_x$$

Donde:

$$C_x = \frac{1}{2}\|y - a^L\|^2$$

El algoritmo consiste en tomar una neurona de la red y agregarle un valor llamado "error" (δ) al valor de entrada de la misma.

$$C = \frac{1}{2}\sum_{i}\left(y_i^L - a_i^L\right)^2$$

$$\frac{\partial C}{\partial z_j^l} = \delta_j^l$$

δ_j^l es la variación de la función de costo respecto a una neurona

Primero se calcula el δ de la última capa llamada L y luego se propaga hacia atrás para calcular los δ de las neuronas de las capas anteriores

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}$$

[6] **(8) Nielsen, Michael. 2015.** Neural Networks and Deep Learning. Chapter 2 – How the backpropagation algorithm works

Dada una neurona j, la derivada de la función de costo se vuelve 0 en todos los términos de la sumatoria excepto cuando $i = j$

$$\frac{\partial C}{\partial a_j^L} = \frac{2}{2} . \left(y_j^L - a_j^L\right). (0 - 1) = (a_j^L - y_j^L)$$

$$\frac{\partial a_j^L}{\partial z_j^L} = \sigma'(z_j^L)$$

$$\delta_j^L = \left(a_j^L - y_j^L\right). \sigma'(z_j^L)$$

$$\boxed{\delta^L = (a^L - y^L) \odot \sigma'(z^L)}$$

Para cualquier capa l, el error de una neurona j se calcula como la sumatoria de las variaciones de la función de costo con respecto a cada una de las neuronas de la capa siguiente multiplicado por la variación de las entradas de las neuronas de la capa siguiente con respecto a la neurona j:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} . \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

$\frac{\partial C}{\partial z_k^{l+1}}$ representa como las neuronas de la capa siguiente afectan a la función de costo.

$\frac{\partial z_k^{l+1}}{\partial z_j^l}$ representa como las neuronas de la capa siguiente varían en función de la neurona j.

$$\frac{\partial C}{\partial z_k^{l+1}} = \delta_k^{l+1}$$

$$z_k^{l+1} = \sum_i w_{ik}^{l+1}.a_i^l + b_k^{l+1}$$

$$z_k^{l+1} = \sum_i w_{ik}^{l+1}.\sigma(z_i^l) + b_k^{l+1}$$

$\frac{\partial z_k^{l+1}}{\partial z_j^l}$ será 0 en todos los términos de la sumatoria excepto cuando i = j

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \frac{\partial (w_{jk}^{l+1}.\sigma(z_j^l) + b_k^{l+1})}{\partial z_j^l}$$

$$\frac{\partial z^{l+1}}{\partial z_j^l} = w_{jk}^{l+1}.\sigma'(z_j^l)$$

$$\delta_j^l = \sum_k \delta_k^{l+1}.w_{jk}^{l+1}.\sigma'(z_j^l)$$

$$\boxed{\delta^l = w^{l+1}\delta^{l+1} \odot \sigma'(z_j^l)}$$

$$\frac{\partial C}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_j^l}.\frac{\partial z_j^l}{\partial w_{ij}}$$

$$\frac{\partial c}{\partial w_{ij}^l} = \delta_j^l.\frac{\partial z^l}{\partial w_{ij}}$$

$$\frac{\partial C}{\partial w_{ij}^l} = \delta_j^l.(S + a_i^{l-1}.w_{ij}^l)$$

$$\frac{\partial C}{\partial w_{ij}^l} = \delta_j^l.(0 + a_i^{l-1}.1)$$

$$\frac{\partial C}{\partial w_{ij}^l} = \delta_j^l.a_i^{l-1}$$

$$\boxed{\frac{\partial C}{\partial w^l} = a^{l-1}(\delta^l)^T}$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l . 1$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\boxed{\frac{\partial C}{\partial b^l} = \delta^l}$$

Anexo IX – Red neuronal convolucional

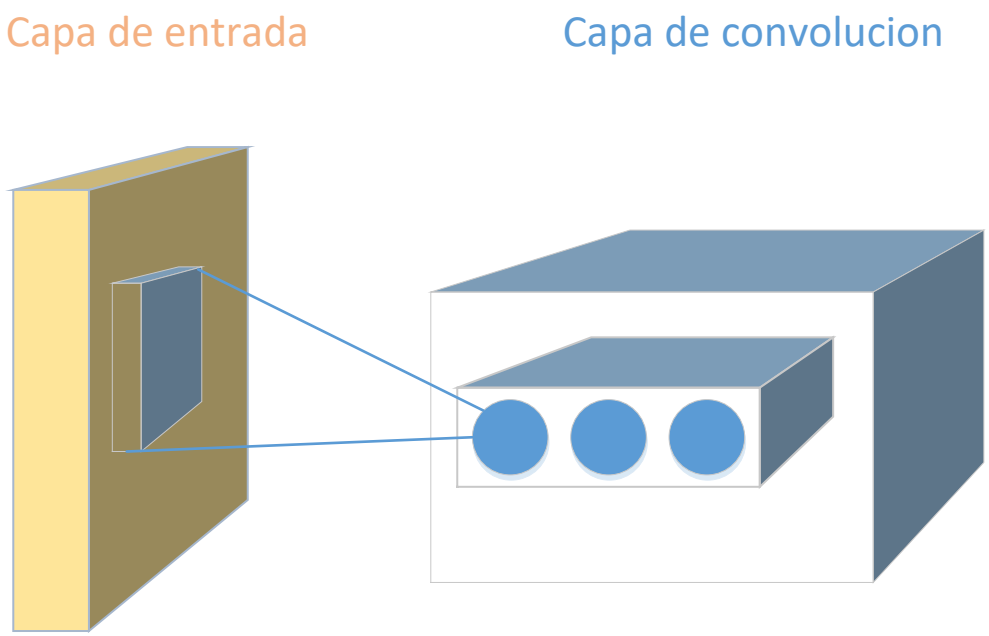
Una red neuronal convolucional[7] es un tipo de red neuronal que tiene una estructura más compleja que las redes neuronales convencionales. Este tipo de redes es normalmente utilizada para aplicaciones de visión artificial debido a las ventajas que ofrece en cuanto a reducción en cantidad de cálculos y eficacia al momento de reconocer imágenes que difieren en gran medida con las imágenes de entrenamiento.

Al momento de entrenar una red neuronal de este tipo, la imagen de entrenamiento va reduciendo su tamaño a medida que va pasando por las distintas capas de la red. De esta forma, la red neuronal puede detectar características de la imagen que no estén fuertemente ligadas al tamaño de la misma. Por otro lado, distintos grupos de neuronas se vinculan a la imagen aplicando distintos tipos de filtros para lograr detectar las características que identifican unívocamente a un objeto más allá de las distintas transformaciones que puedan sufrir las imágenes del mismo.

Las redes neuronales convolucionales, al igual que las redes neuronales convencionales, agrupan sus neuronas en distintas capas (Capa de entrada, capas ocultas y capas de salida). La capa de entrada y salida tienen la misma función que en las redes neuronales convencionales. La de entrada recibe la imagen que se quiere que la red neuronal entrene o reconozca y la de salida etiqueta la imagen en la fase de entrenamiento y muestra el resultado final del proceso en la etapa de reconocimiento. Las capas ocultas se dividen en distintos tipos. El primer tipo que aparece en este tipo de redes es la capa de convolucion. Esta capa distribuye sus neuronas en 3 dimensiones donde cada eje de profundidad de neuronas recorre la imagen de entrada con un filtro distinto. A cada eje de profundidad de neuronas se lo llama “Mapa de activación”. Cada neurona de esta capa se conecta a una región de neuronas de la capa de entrada a través de una matriz de valores (Filtro). De esta forma, cada neurona aprende características de la imagen teniendo en cuenta regiones de la misma. Esta capa tiene distintos parámetros como la profundidad de neuronas, o la separación entre la región de neuronas de la capa de entrada a la que se conectan las distintas neuronas de los mapas de activación y la región de neuronas a la que se conectan las neuronas adyacentes a estas. Al momento de entrenar la red, la actualización de los pesos y los bias de esta capa se

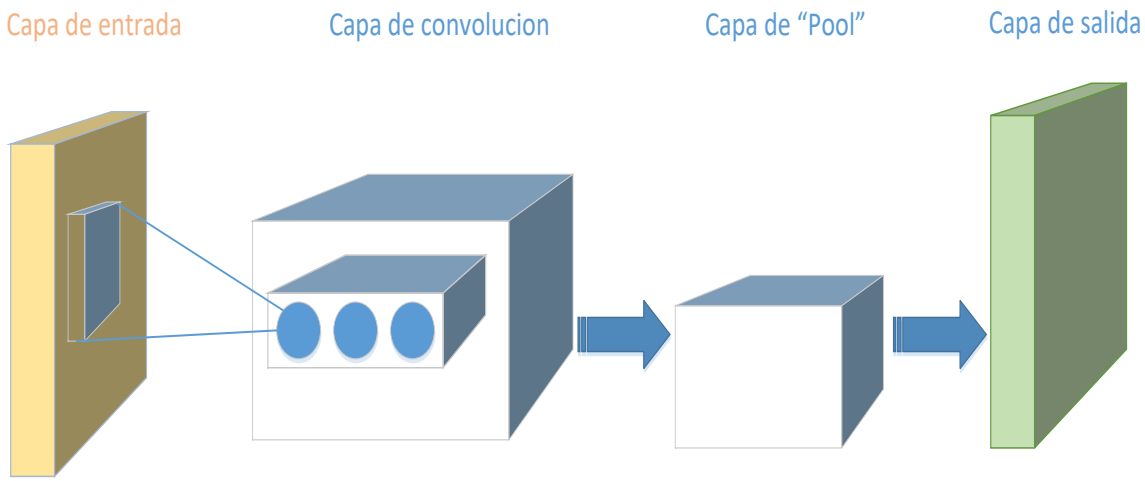
[7] **(3) Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, Awni Hannun, Brody Huval, Tao Wang, Sameep Tandon.** UFLDL Tutorial – Convolutional Neural Network

realizan por grupos ya que todas las neuronas de un mapa de activación comparten tanto los pesos como los bias pero no sobre qué región de la capa de entrada se conectan.

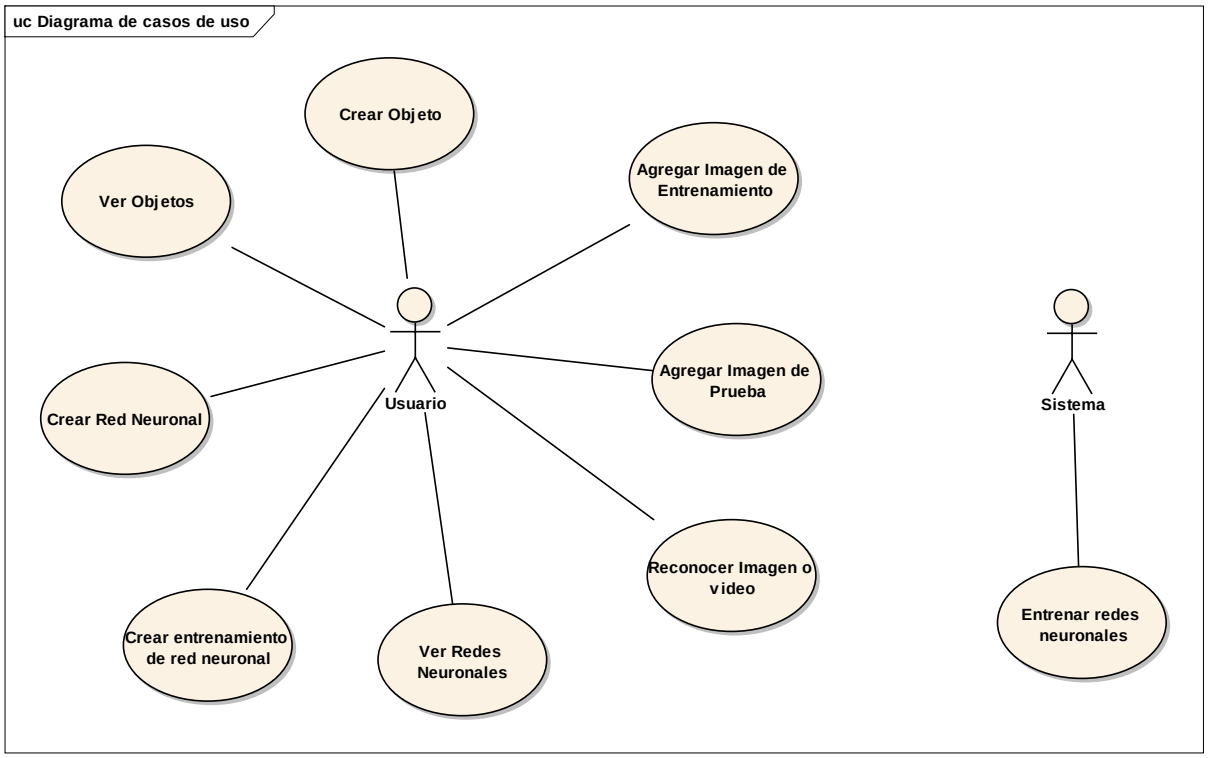


Otra de las capas que forman parte de la arquitectura de las redes neuronales convolucionales es la capa de “Pool”. Esta capa tiene como función reducir el tamaño de la imagen para entregarla a una nueva capa de convolucion. Gracias a esta capa se logran detectar características de la imagen que van a permanecer aunque cambie su tamaño. Para reducir el tamaño de la imagen se suele utilizar la técnica de “max-pooling”. Esta técnica consiste en tomar el mayor valor de un grupo de pixeles y pasarlo como salida en representación de este grupo.

Las redes neuronales convolucionales tiene una capa de entrada, una capa de salida y suelen tener varias capas de convolucion y de “Pool”.



Anexo X - Casos de Uso



Crear objeto

Caso de Uso ID:	CU001		
Caso de Uso Nombre:	Crear objeto		
Creado por:	Nicolas	Última actualización por:	Nicolas
Fecha Creación:		Fecha última actualización:	

Actor:	Usuario
Descripción:	El usuario
Precondiciones:	
Postcondiciones:	Se crea un objeto en el sistema al cual se le pueden añadir imágenes de entrenamiento y de prueba y se lo puede incluir en una red neuronal
Prioridad:	Alta
Frecuencia de uso:	
Flujo Normal	1 El usuario ingresa el nombre del objeto a crear 2 El usuario selecciona el objeto del cual hereda el objeto a crear 3 El sistema crea el objeto con el nombre indicado y establece la relación de herencia
Flujos Alternativos	
Excepciones	
Includes:	NA
Extends	NA
Requerimientos No Funcionales:	NA
Notas :	NA

Ver objetos

Caso de Uso ID:	CU002		
Caso de Uso Nombre:	Ver objetos		
Creado por:	Nicolas	Última actualización por:	Nicolas
Fecha Creación:		Fecha última actualización:	

Actor:	Usuario
Descripción:	*El usuario ve todos los objetos creados organizados en forma de árbol para representar la herencia que existe entre los mismo*
Precondiciones:	Debe haber al menos 1 objeto creado
Postcondiciones:	Se muestran todos los objetos creados
Prioridad:	Alta
Frecuencia de uso:	
Flujo Normal	1 El sistema muestra un listado de todos los objetos creados organizados en forma de arbol
Flujos Alternativos	
Excepciones	
Includes:	NA
Extends	NA
Requerimientos No Funcionales:	NA
Notas :	NA

Agregar imagen de entrenamiento

Caso de Uso ID:	CU003		
Caso de Uso Nombre:	Agregar imagen de entrenamiento		
Creado por:	Nicolas	Última actualización por:	Nicolas
Fecha Creación:		Fecha última actualización:	

Actor:	Usuario
Descripción:	*El usuario agrega una imagen al set de imágenes de entrenamiento de un objeto*
Precondiciones:	Debe haber al menos 1 objeto creado
Postcondiciones:	El objeto tiene una nueva imagen de entrenamiento
Prioridad:	Alta
Frecuencia de uso:	
Flujo Normal	1. El Usuario selecciona un objeto 2. El usuario busca una imagen para agregar al set de imágenes de entrenamiento del objeto 3. El sistema agrega la imagen al set de imágenes de entrenamiento del objeto
Flujos Alternativos	
Excepciones	
Includes:	NA
Extends	NA
Requerimientos No Funcionales:	NA
Notas :	NA

Agregar imagen de prueba

Caso de Uso ID:	CU004		
Caso de Uso Nombre:	Agregar imagen de prueba		
Creado por:	Nicolas	Última actualización por:	Nicolas
Fecha Creación:		Fecha última actualización:	

Actor:	Usuario
Descripción:	*El usuario agrega una imagen al set de imágenes de prueba de un objeto*
Precondiciones:	Debe haber al menos 1 objeto creado
Postcondiciones:	El objeto tiene una nueva imagen de prueba
Prioridad:	Alta
Frecuencia de uso:	
Flujo Normal	1. El Usuario selecciona un objeto 2. El usuario busca una imagen para agregar al set de imágenes de prueba del objeto 3. El sistema agrega la imagen al set de imágenes de prueba del objeto
Flujos Alternativos	
Excepciones	
Includes:	NA
Extends	NA
Requerimientos No Funcionales:	NA
Notas :	NA

Ver redes neuronales

Caso de Uso ID:	CU005		
Caso de Uso Nombre:	Ver redes neuronales		
Creado por:	Nicolas	Última actualización por:	Nicolas
Fecha Creación:		Fecha última actualización:	

Actor:	Usuario
Descripción:	*El usuario ve las redes neuronales creadas*
Precondiciones:	
Postcondiciones:	
Prioridad:	Alta
Frecuencia de uso:	
Flujo Normal	1. El sistema muestra un listado de las redes neuronales creadas
Flujos Alternativos	
Excepciones	
Includes:	NA
Extends	NA
Requerimientos No Funcionales:	NA
Notas :	NA

Crear red neuronal

Caso de Uso ID:	CU006		
Caso de Uso Nombre:	Crear red neuronal		
Creado por:	Nicolas	Última actualización por:	Nicolas
Fecha Creación:		Fecha última actualización:	

Actor:	Usuario
Descripción:	*El usuario crea una nueva red neuronal, indicando que objetos puede reconocer y cuál es el tamaño de imágenes que aceptara ya sea para el proceso de entrenamiento, de prueba o de reconocimiento*
Precondiciones:	Debe haber al menos 1 objeto creado
Postcondiciones:	El sistema tiene una nueva red neuronal
Prioridad:	Alta
Frecuencia de uso:	
Flujo Normal	1. El sistema solicita que se ingrese el nombre de la nueva red neuronal. 2. El sistema muestra un listado de los objetos creados 3. El sistema solicita que se ingrese el tamaño de las imágenes que aceptara la nueva red neuronal 4. El usuario ingresa el nombre de la red neuronal 5. El usuario marca los objetos que podrá reconocer la nueva red neuronal 6. El usuario indica el tamaño de las imágenes que aceptara la nueva red neuronal
Flujos Alternativos	
Excepciones	
Includes:	NA
Extends	NA
Requerimientos No Funcionales:	NA
Notas :	NA

Crear entrenamiento de red neuronal

Caso de Uso ID:	CU007		
Caso de Uso Nombre:	Crear entrenamiento de red neuronal		
Creado por:	Nicolas	Última actualización por:	Nicolas
Fecha Creación:		Fecha última actualización:	

Actor:	Usuario
Descripción:	*El usuario crea un nuevo entrenamiento para una red neuronal creada.*
Precondiciones:	Debe haber al menos 1 red neuronal creada
Postcondiciones:	Una red neuronal tiene 1 nuevo entrenamiento
Prioridad:	Alta
Frecuencia de uso:	
Flujo Normal	1. El sistema solicita que se ingrese la cantidad de epochs del nuevo entrenamiento 2. El usuario ingresa la cantidad de epochs del nuevo entrenamiento 3. El sistema solicita que se ingrese el tamaño del batch de entrenamiento 4. El usuario ingresa el tamaño del batch de entrenamiento 5. El sistema crea un nuevo entrenamiento
Flujos Alternativos	
Excepciones	
Includes:	NA
Extends	NA
Requerimientos No Funcionales:	NA
Notas :	NA

Reconocer imagen o video

Caso de Uso ID:	CU008		
Caso de Uso Nombre:	Reconocer imagen o video		
Creado por:	Nicolas	Última actualización por:	Nicolas
Fecha Creación:		Fecha última actualización:	

Actor:	Usuario
Descripción:	*El usuario ingresa al sistema una imagen o un video. El sistema realiza un proceso de segmentación de la imagen o de un fotograma del video para detectar las partes individuales de la imagen que con mayor probabilidad sean objetos a ser reconocidos por la red neuronal*
Precondiciones:	Debe haber al menos 1 objeto creado Debe haber al menos 1 red neuronal creada
Postcondiciones:	Imagen reconocida por el sistema
Prioridad:	Alta
Frecuencia de uso:	
Flujo Normal	1. El sistema solicita que se seleccione la red neuronal que se va a utilizar para reconocer la imagen o el video 2. El sistema solicita que se ingrese una imagen o un video 3. El usuario ingresa una imagen o un video 4. El sistema crea un nuevo entrenamiento
Flujos Alternativos	5. El usuario adjunta un tipo de archivo no valido 6. El sistema solicita un tipo de archivo valido (Imagen o video)
Excepciones	
Includes:	NA
Extends	NA
Requerimientos No Funcionales:	NA
Notas :	NA

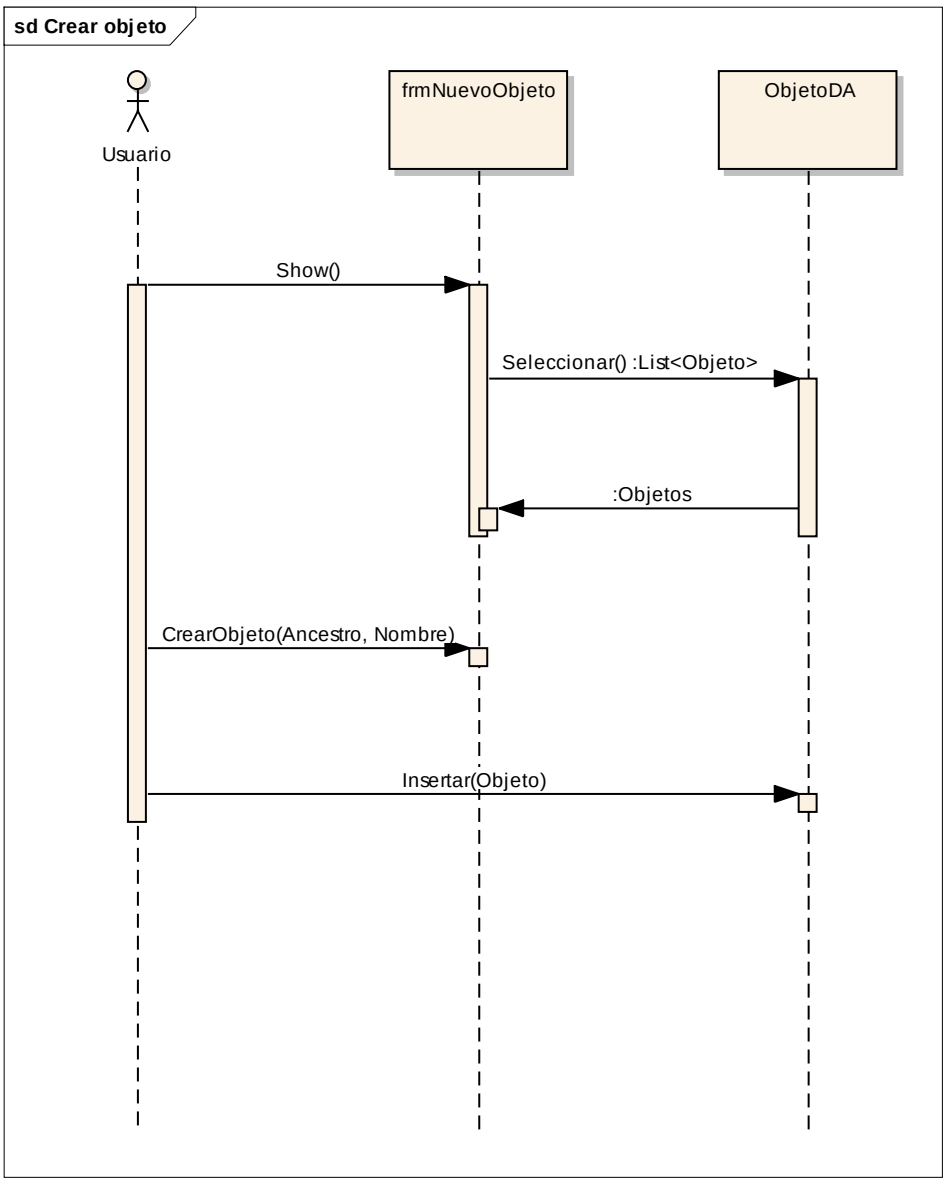
Entrenar redes neuronales

Caso de Uso ID:	CU009		
Caso de Uso Nombre:	Entrenar redes neuronales		
Creado por:	Nicolas	Última actualización por:	Nicolas
Fecha Creación:		Fecha última actualización:	

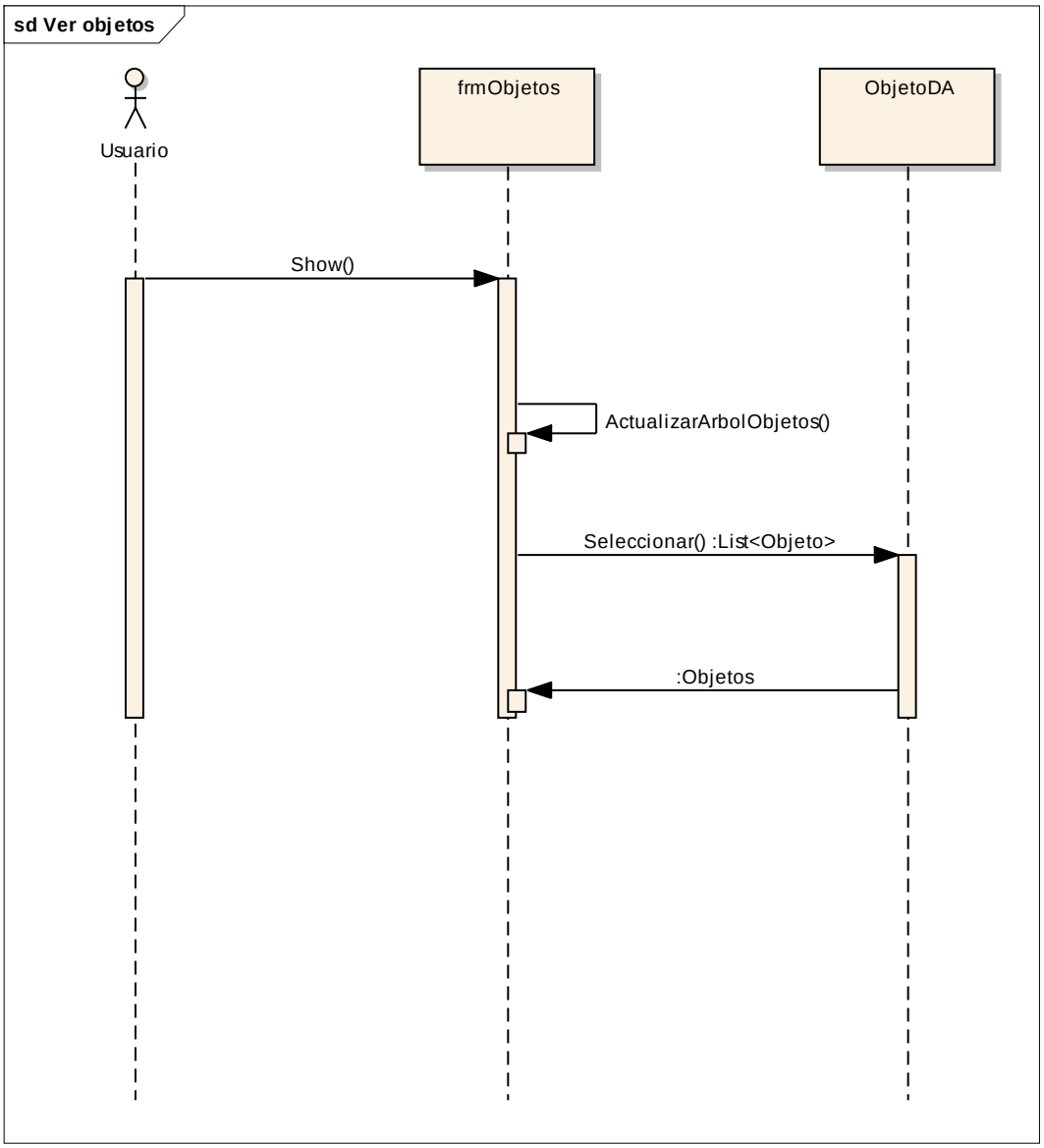
Actor:	Sistema
Descripción:	*El sistema entrena las redes neuronales según los entrenamientos creados por el usuario*
Precondiciones:	Debe haber al menos 1 red neuronal creada Debe haber al menos 1 entrenamiento de una red neuronal creado
Postcondiciones:	Se realizaron los entrenamientos creados por el usuario
Prioridad:	Alta
Frecuencia de uso:	
Flujo Normal	1. El sistema selecciona los entrenamientos que tienen epochs pendientes Por cada entrenamiento 2. Si el entrenamiento no tiene epochs existentes 2.1 El sistema crea una nueva red neuronal y entrena el primer epoch 3. Si el entrenamiento tiene epochs existentes 3.1 El sistema obtiene la red neuronal del ultimo epoch y la re entrena
Flujos Alternativos	
Excepciones	
Includes:	NA
Extends	NA
Requerimientos No Funcionales:	NA
Notas :	NA

Anexo XI - Diagramas de secuencia

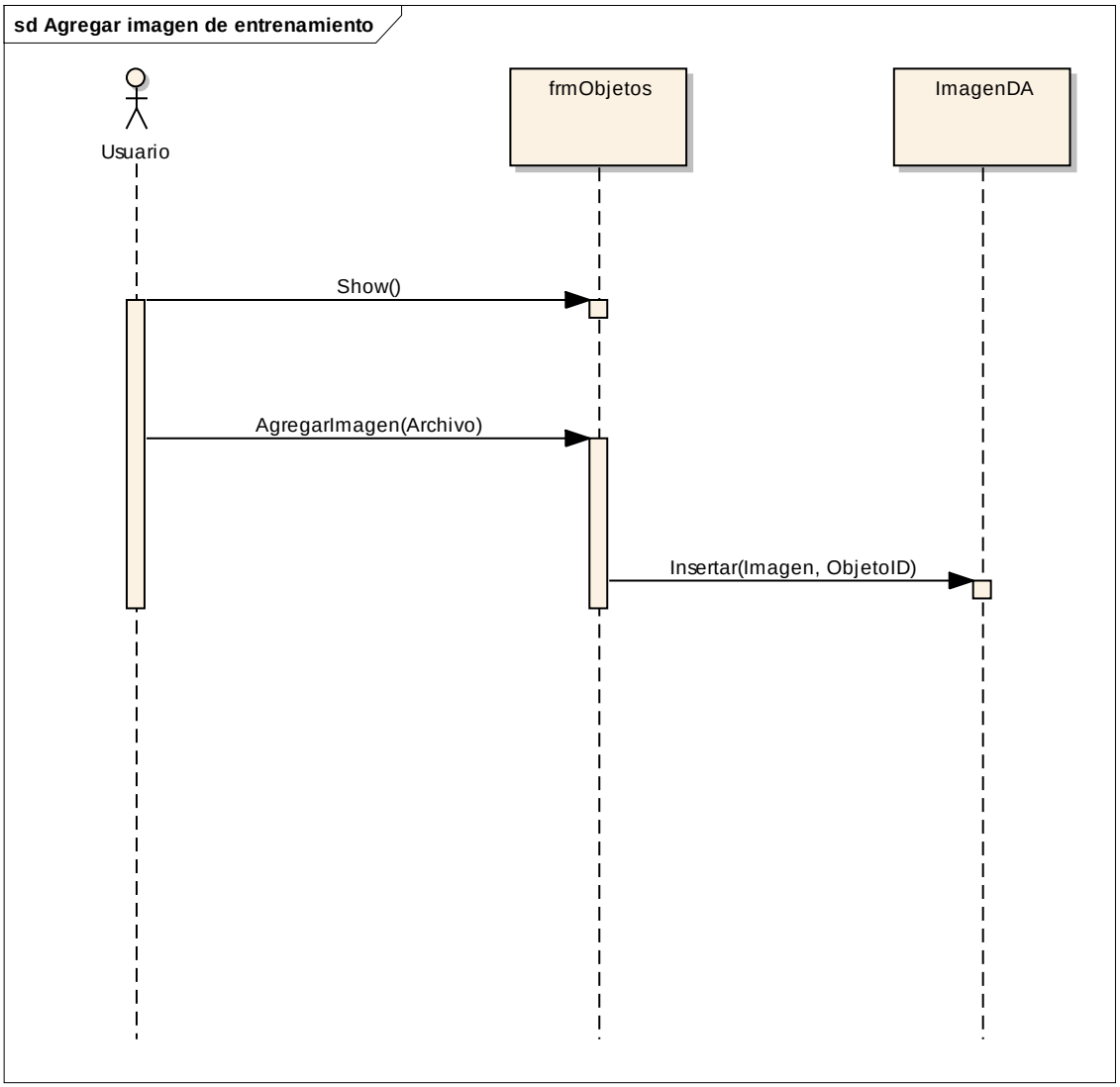
Crear objeto



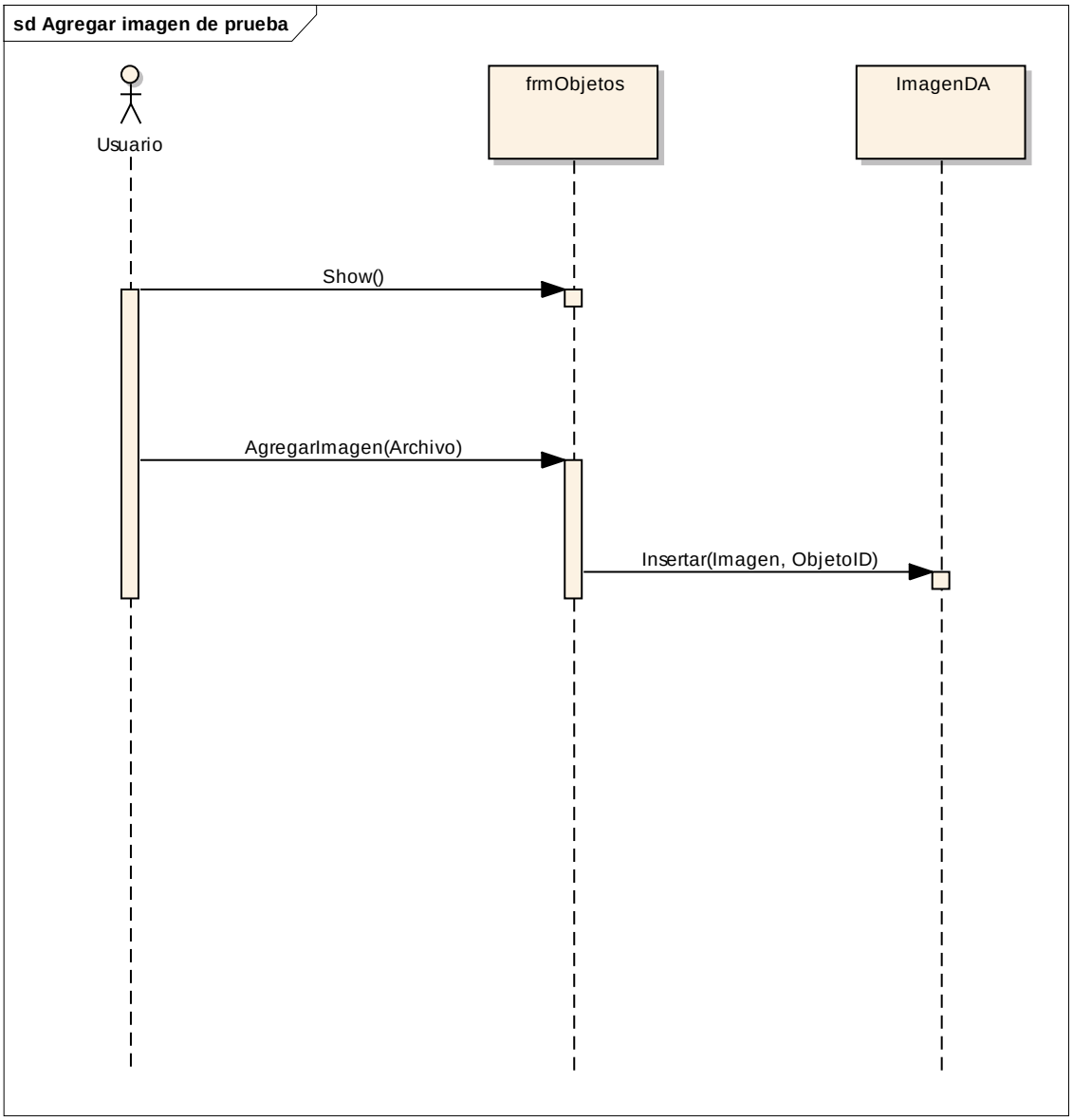
Ver objetos


sd Ver objetos
Usuario
frmObjetos
ObjetoDA
Show()
ActualizarArbolObjetos()
Seleccionar() :List<Objeto>
:Objetos

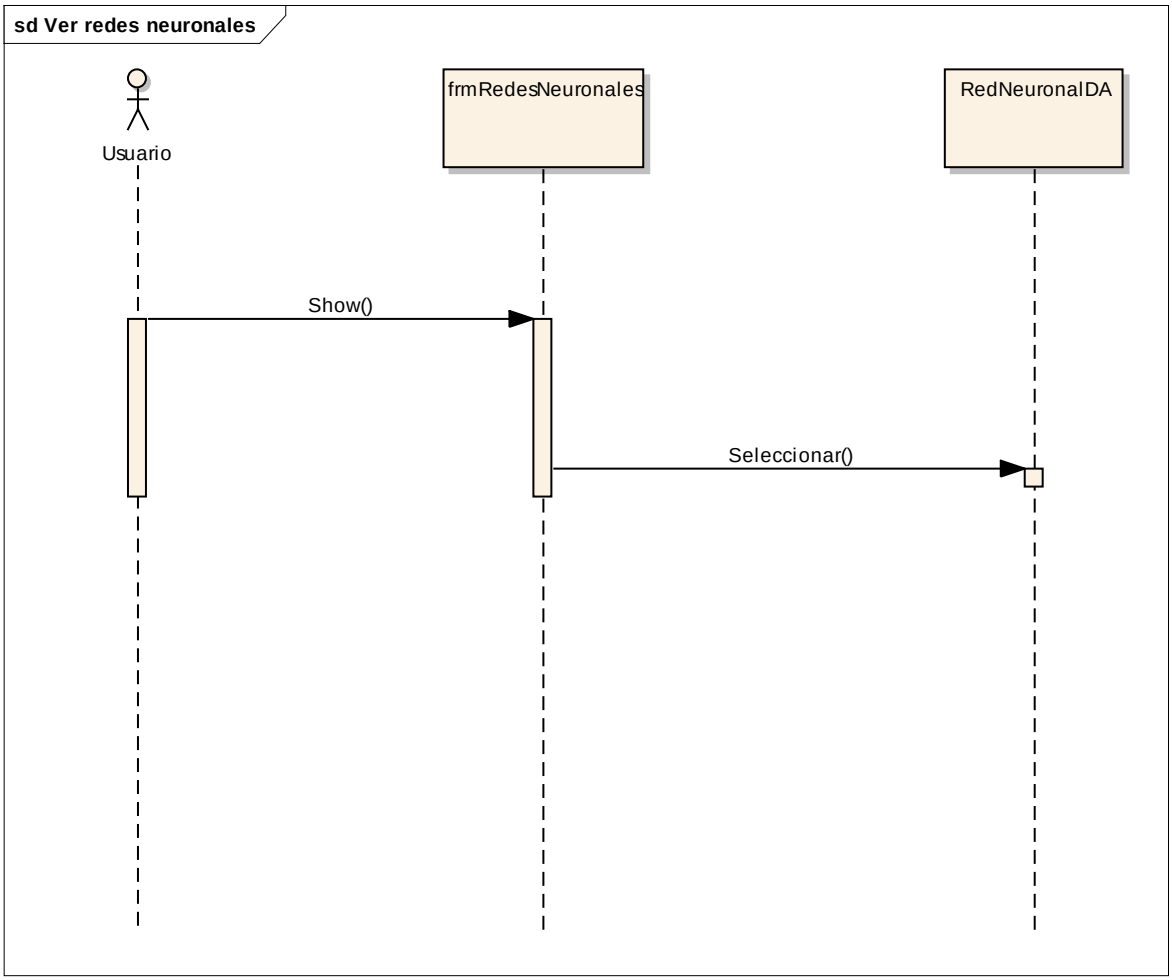
Agregar imagen de entrenamiento



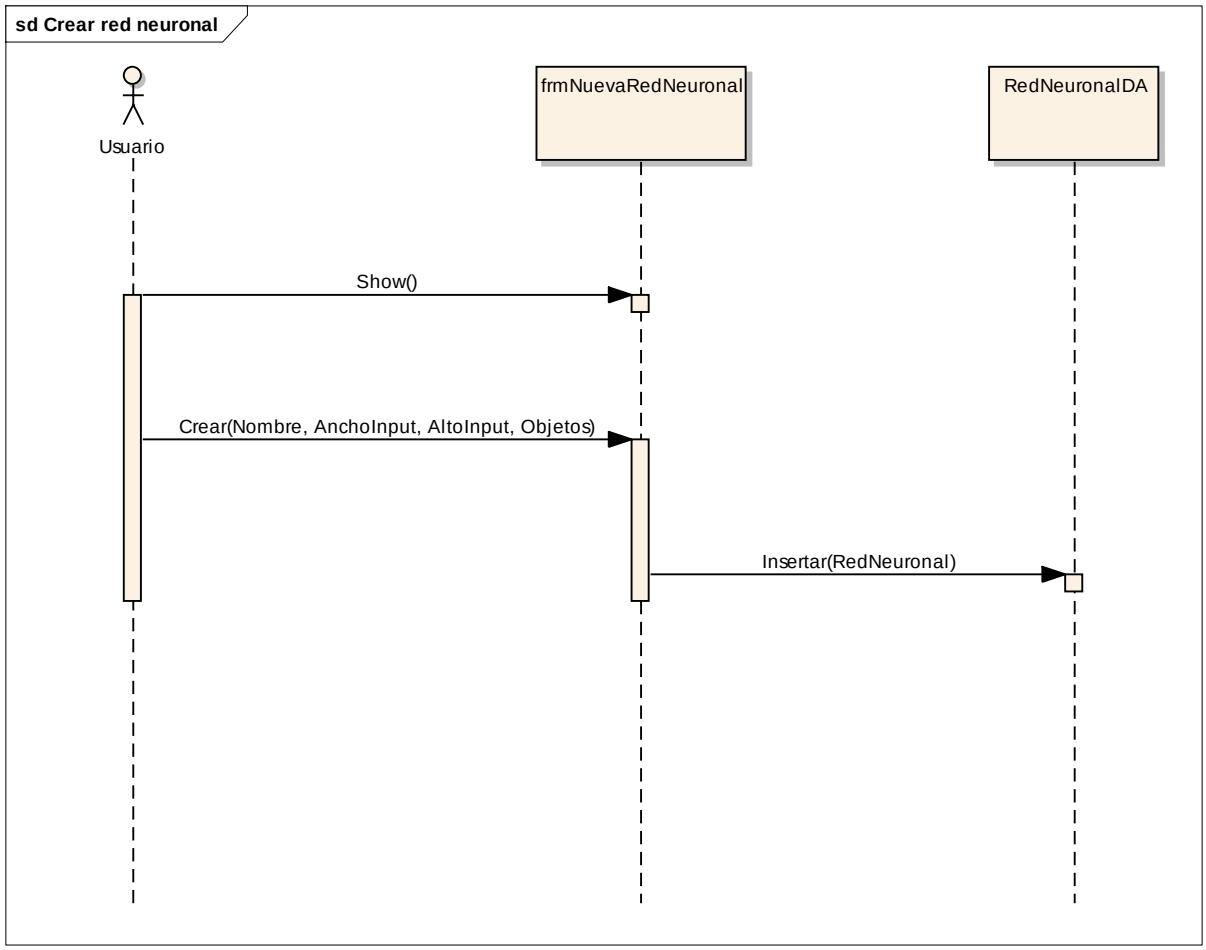
Agregar imagen de prueba



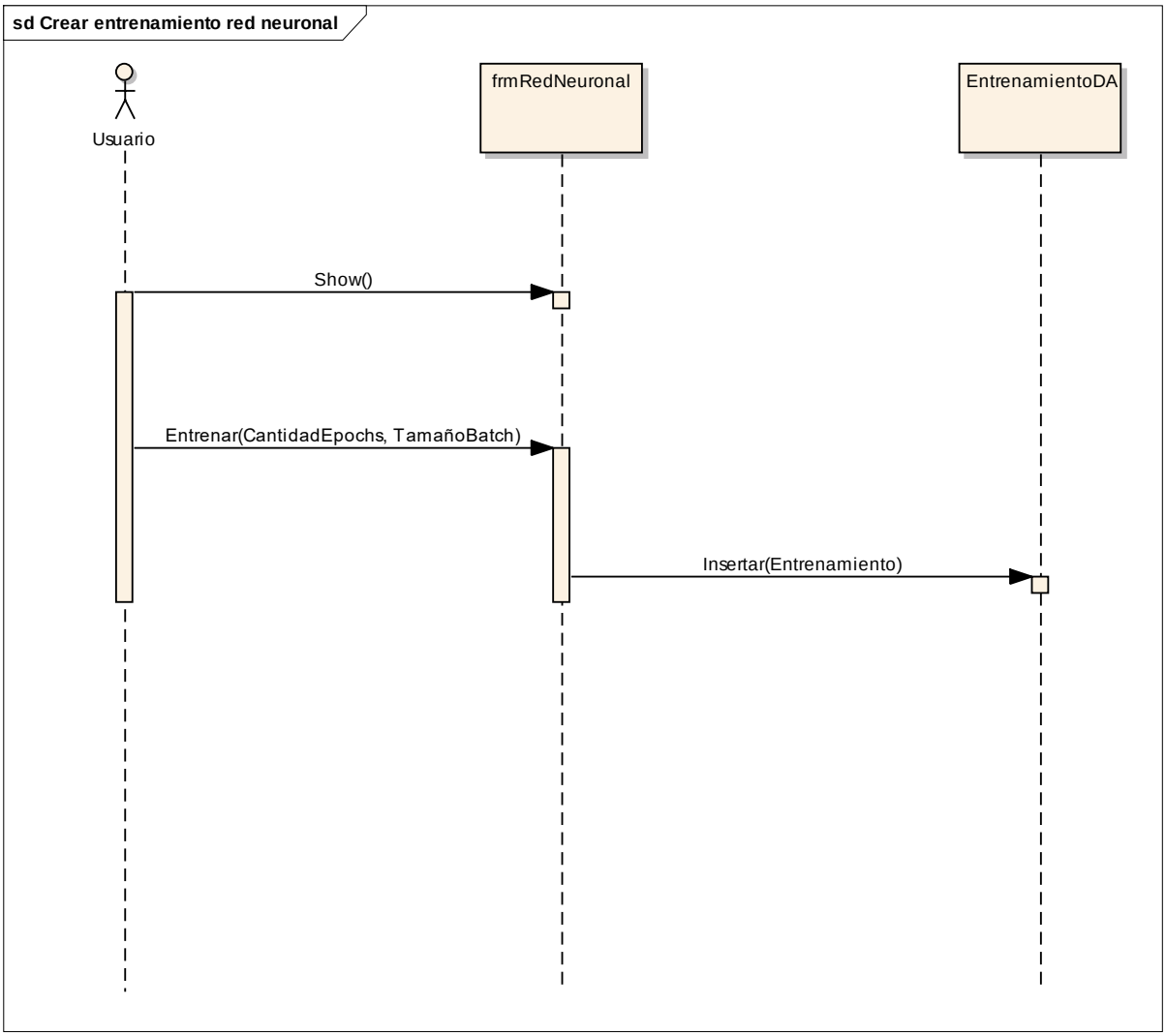
Ver redes neuronales



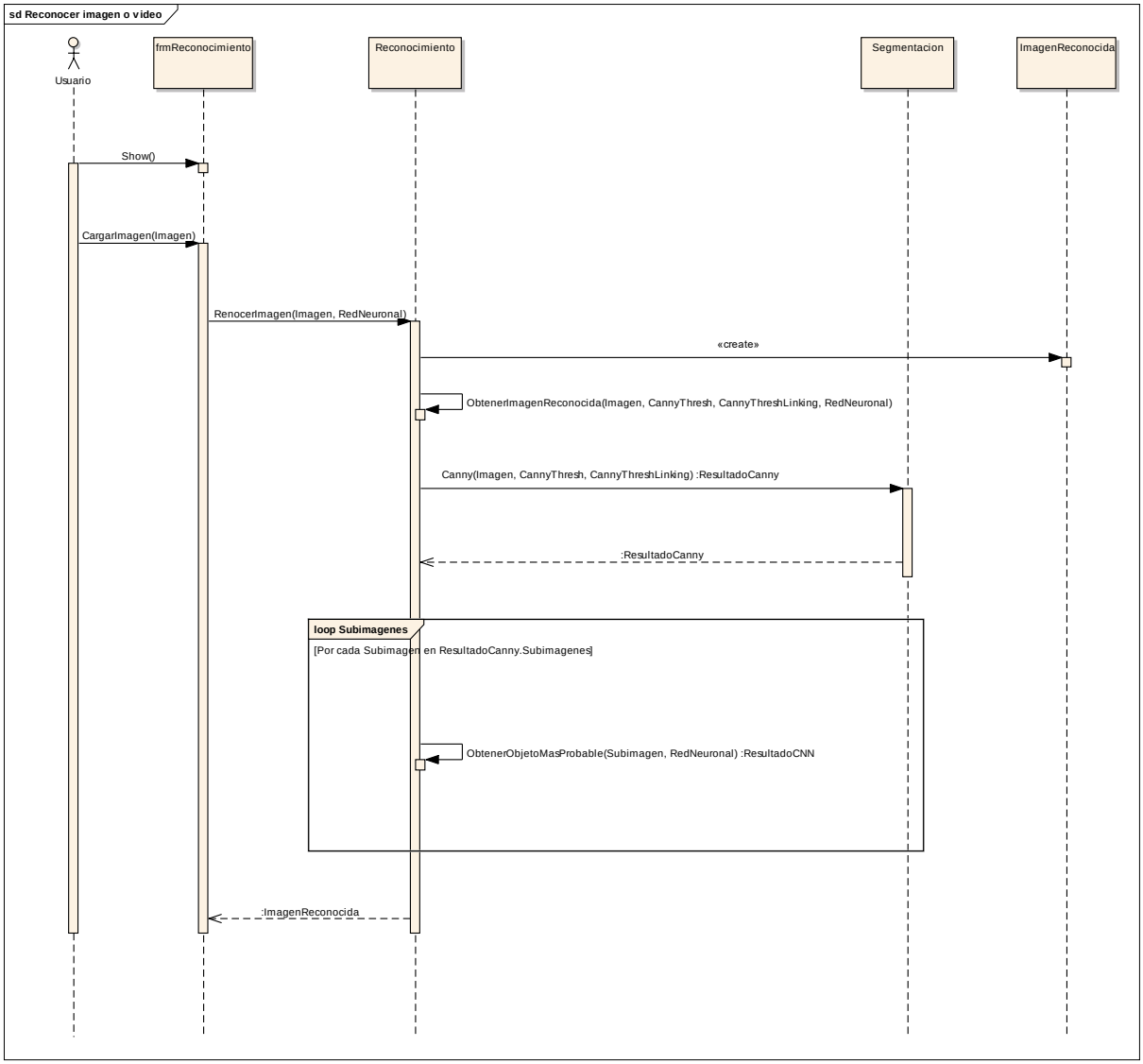
Crear red neuronal



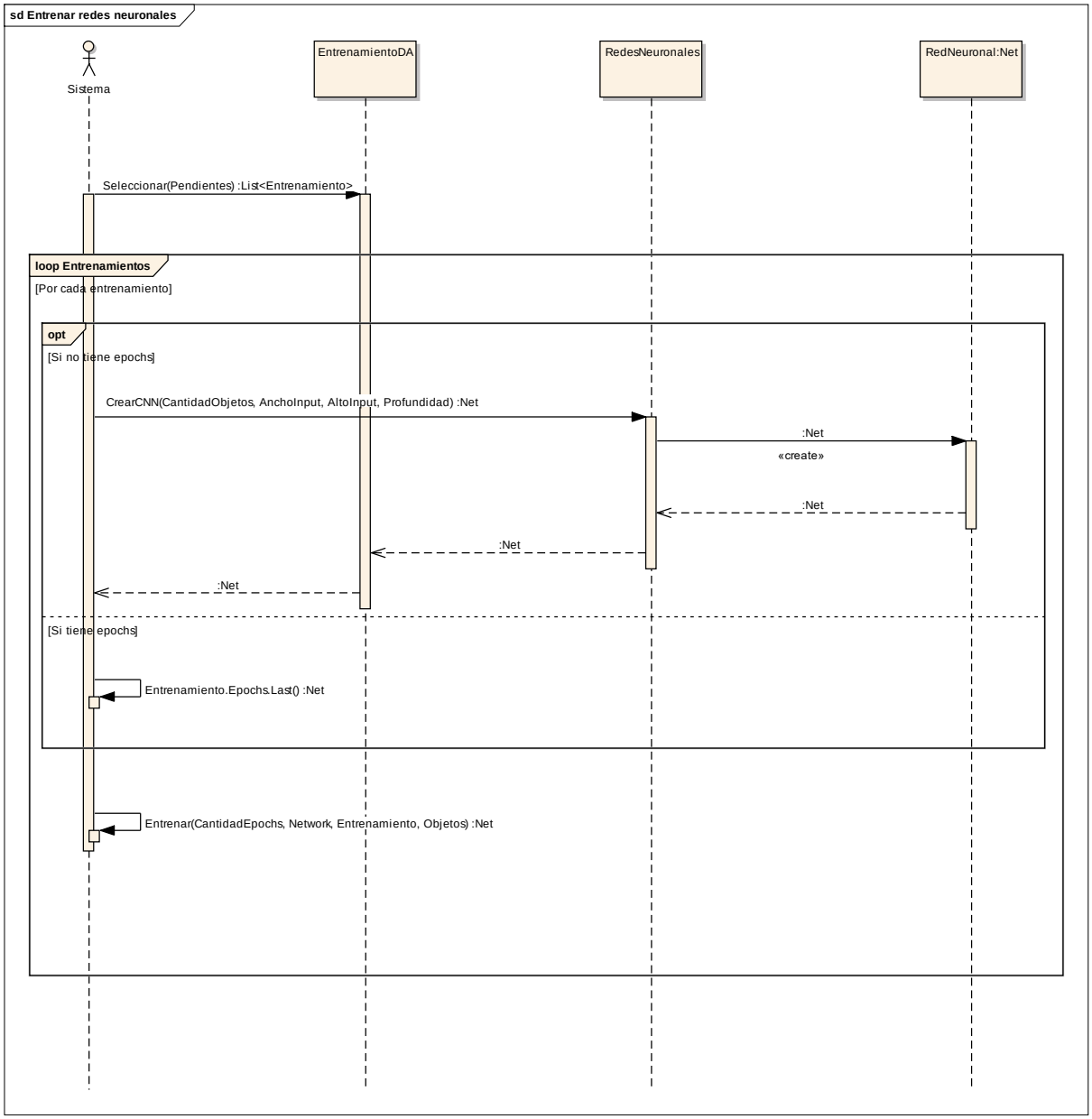
Crear entrenamiento de red neuronal



Reconocer imagen o video



Entrenar redes neuronales



Bibliografía

(**1) Aaron Hertzmann, David J. Fleet, Marcus Brubaker.** CSCC11: Introduction to Machine Learning and Data Mining. [Online] http://www.cs.toronto.edu/~g8acai/teaching/C11/Handouts/GradientDescent.pdf.
(2) Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR. [Online] http://www.cs.utoronto.ca/~kriz/cifar.html.
(3) Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, Awni Hannun, Brody Huval, Tao Wang, Sameep Tandon. UFLDL Tutorial. [Online] http://ufldl.stanford.edu/tutorial/.
(4) EmguCV. EmguCV. [Online] http://www.emgu.com/wiki/index.php/Main_Page.
Green, Bill. 2002. [Online] 2002. http://dasl.unlv.edu/daslDrexel/alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html.
(5) Kim, Daniel. 2013. Sobel Operator and Canny Edge Detector ECE 480 Fall 2013. [Online] 2013. http://www.egr.msu.edu/classes/ece480/capstone/fall13/group04/docs/danapp.pdf.
(6) L. Ballan, M. Bertini, A. Del Bimbo, L. Seidenari, and G. Serra. 2009. Effective Codebooks for Human Action Categorization. [Online] ICCV International Workshop on Video-oriented Object and Event Classification (VOEC), 2009.
(7) Leverington, David. 2009. A Basic Introduction to Feedforward Backpropagation Neural Networks. [Online] 2009. http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html.
(8) Nielsen, Michael. 2015. Neural Networks and Deep Learning. [Online] Determination Press, 2015. [Cited: 07 20, 2015.] http://neuralnetworksanddeeplearning.com/.
(9) OpenCV. OpenCV. [Online] http://docs.opencv.org/master/index.html.
(10) Russell, Ingrid. 1996. Neural Networks Module. [Online] Collegiate Microcomp uter Journal, 1996. http://uhaweb.hartford.edu/compsci/neural-networks-Learning.html.