

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

**DOCTORADO EN TECNOLOGÍAS INFORMÁTICAS
AVANZADAS**

TESIS DOCTORAL

Sistema para la identificación de eventos anómalos en el hogar
Home Monitor

Luis Gabriel Rojas Albarracín

mayo, 2020

**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

<Primera línea Depto. Director>
<Segunda línea Depto. Director>

TESIS DOCTORAL

**Sistema para la identificación de eventos anómalos en el
hogar
Home Monitor**

Autor: Luis Gabriel Rojas Albarracín

Tutor: Antonio Fernandez-Caballero

Co-Tutor: María Teresa López Bonal

mayo, 2020

TRIBUNAL:

Presidente: _____

Vocal: _____

Secretario: _____

FECHA DE DEFENSA: _____

CALIFICACIÓN: _____

PRESIDENTE

VOCAL

SECRETARIO

Fdo.:

Fdo.:

Fdo.:

*A mi familia y a Jenny
Por tolerarme durante el desarrollo de este trabajo.
Y a Yova
que por su inicial y durante el largo recorrido.*

Resumen

Abstract

AGRADECIMIENTOS

Luis Gabriel Rojas Albarracín
Albacete, 2020

ÍNDICE GENERAL

Resumen	vii
Agradecimientos	xI
Índice de figuras	xvII
Índice de tablas	xix
Índice de listados	xxI
Índice de algoritmos	xxIII
1. Planteamiento del problema	1
1.1. Introducción	1
1.2. Motivación / Justificación	2
1.3. Hipótesis	4
1.4. Objetivos	5
1.5. Alcance	6
2. Marco referencial	7
2.1. Marco conceptual	7
2.1.1. Inteligencia artificial	7
2.1.2. Aprendizaje automático (Machine learning)	9
2.1.3. Aprendizaje supervisado	10
2.1.4. Aprendizaje no supervisado	10
2.1.5. Aprendizaje reforzado	11
2.1.6. Aprendizaje profundo (Deep learning)	12
2.1.7. Redes neuronales (NN)	12
2.1.8. Redes convolucionales (CNN)	17
2.1.9. Redes recurrentes (RNN)	20
2.1.10. Redes residuales (ResNet)	23
2.1.11. Procesamiento de imágenes digitales	24
2.1.12. Procesamiento del lenguaje natural (NLP)	25
2.1.13. Sobre entrenamiento (Overfitting)	26
2.1.14. Transferencia de aprendizaje	26
2.2. Algoritmos	27
2.2.1. Funciones de activación	27
2.2.2. Funciones de pérdida	30
2.2.3. Descenso de gradiente	32
2.2.4. Propagación hacia adelante (Forward propagation)	36
2.2.5. Propagación hacia atrás (Back propagation)	37

2.2.6. Función de clasificación Softmax	37
2.3. Arquitectura de software	37
2.3.1. Patrón de desarrollo de software	38
2.3.2. Instancia única (Patrón)	38
2.3.3. Método factoría (Patrón)	38
2.3.4. Iterador (Patrón)	39
2.3.5. Comando (Patrón)	39
2.3.6. Composición (Patrón)	39
2.3.7. Método plantilla (Patrón)	39
2.3.8. Pila de datos (Pool)	40
2.3.9. Hilo	40
2.3.10. Servicio REST	40
2.4. Conjuntos de datos	41
2.4.1. Definición de los datos y fuentes necesarias	41
2.4.2. Recopilación de datos	41
2.4.3. Formateo de datos	42
2.4.4. Limpieza y reducción de datos	42
2.4.5. Normalización de datos	42
2.4.6. Aumento de datos	43
2.4.7. División del conjunto de datos	43
2.4.8. Sesgo de datos	44
2.4.9. ImageNet (Data set)	46
2.4.10. COCO (Data set)	46
2.4.11. MPII Human Pose (Data set)	47
2.4.12. HumanEva (Data set)	47
2.4.13. FLIC (Data set)	48
2.4.14. DMLSmartActions (Data set)	48
2.5. Software y librerías	48
2.5.1. Python	48
2.5.2. JSON	48
2.5.3. Yaml	48
2.5.4. Tensorflow	48
2.5.5. Keras	49
2.5.6. Flask	49
2.5.7. Logging	49
2.5.8. Microsoft Kinect	49
2.5.9. Web Sockets	50
2.6. Antecedentes	50
2.6.1. Detección de objetos en imágenes	50
2.6.2. Localización de personas en una imagen	52
2.6.3. Identificación de posturas	54
2.6.4. Reconocimiento de actividad humanas (HAR)	57
2.6.5. Identificación de eventos anómalos	62
3. Arquitectura el sistema	65
3.1. Diseño arquitectónico general	66
3.2. Pila de datos	68
3.2.1. Estructura de datos	68
3.2.2. Transmisión y vigencia de datos	70
3.3. Estructura básica de componentes extensibles	75
3.4. Sección A - Entrada de datos	77
3.4.1. cargador de controladores - LoaderOfController	77

3.4.2. Controlador de dispositivos - DeviceController	78
3.5. Sección B - Reconocimiento de eventos simples	81
3.5.1. Cargador de reconocedores - LoaderOfRecognizer	81
3.5.2. reconocedor de actividades - ActivityRecongnizer	82
3.6. Sección C - Analizador de eventos y notificación de alertas	85
3.6.1. Cargador de Analizadores - LoaderAnalyzer	86
3.6.2. Analizador de eventos - EventAnalyzer	87
3.6.3. Canales de notificación - CommChannel	89
3.6.4. Estructura de mensajes de notificación - Message	91
3.7. Modelo integrado del sistema	92
3.7.1. Estructura de directorios	92
3.7.2. Modelo de clases	94
3.7.3. Modelo de despliegue	94
3.8. Modelo para la construcción de un componente controlador de dispositivos	95
3.8.1. Plantilla Controlador de dispositivos	95
3.8.2. Implementación de un nuevo controlador de dispositivos	99
3.9. Modelo para la construcción de un componente de Reconocedor de actividades	101
3.9.1. Plantilla Reconocedor de actividades	101
3.9.2. Implementación de un nuevo Reconocedor de actividades	104
3.10. Modelo para la construcción de un componente analizador de eventos	106
3.10.1. Plantilla analizador	106
3.10.2. Implementación de un nuevo analizador de eventos	109
3.11. Modelo para la construcción de un componente de notificación	111
3.11.1. Plantilla notificador	111
3.11.2. Implementación de un nuevo notificador	113
3.12. Creación de conjuntos de datos para entrenamientos y pruebas	115
3.12.1. Selección de datos	115
3.12.2. División del conjunto de datos: Entrenamiento, Validación y Pruebas	115
3.12.3. Aumento de datos	115
3.13. Funcionalidades auxiliares	116
3.13.1. Utilidades varias - Misc.py	116
3.13.2. Segmentación de los datos - DataSplit.py	117
3.13.3. Segmentación de los datos - DataAugmented.py	117
4. Construcción del sistema	119
4.1. Módulo para incluir las cámaras web como un tipo sensor del sistema	121
4.1.1. Extracción de la persona	123
4.1.2. Extracción del esqueleto	123
4.2. Construcción de módulo para el reconocimiento de infartos	123
4.2.1. Detección de ataque cardíaco en imágenes en color utilizando redes neuronales convolucionales	124
4.2.2. Use of skeletons and facial expressions to identify heart attacks	126
4.3. Construcción del módulo para el reconocimiento de caídas	128
4.4. Construcción del módulo para envío de alertas	129
4.4.1. Módulo para la notificación usando correo electrónico - MailChannel	129
4.5. Construcción del módulo para análisis de eventos complejos	130

ÍNDICE DE FIGURAS

2.1.	Modelo general de una red neuronal	14
2.2.	Modelo del Perceptrón	14
2.3.	Modelo del Perceptrón multi capa	15
2.4.	Listado de arquitecturas de redes neuronales	16
2.5.	Arquitectura tradicional de una red convolucional.	18
2.6.	Pasos de una capa de convolución	18
2.7.	Capas de relleno para convolución	19
2.8.	Diseño general de una RNN	20
2.9.	RNN desenrollada	21
2.10.	Esquema de un bloque de LSTM	22
2.11.	Diseño general de la arquitectura ResNet	23
2.12.	Imagen digital	24
2.13.	Sobre entrenamiento	26
2.14.	Representación ecuación Sigmoidal	28
2.15.	Representación ecuación Tangente hiperbólica	29
2.16.	Representación ecuación ReLU	30
2.17.	Gráfica de la función $f(x) = x^2$	33
2.18.	Gráfica de la función $z = x^2 - y^2$	33
2.19.	Mínimos locales	34
2.20.	Modelo del Perceptrón multi capa	36
2.21.	Pasos para la creación de conjuntos de datos.	41
2.22.	División del conjunto de datos.	44
2.23.	Conjunto de datos sin sesgo.	45
2.24.	Ejemplo de imágenes del COCO keypoint Challenge	47
2.25.	MPII Human Pose	47
2.26.	Microsoft Kinect	50
2.27.	Extracción del fondo usando croma	53
2.28.	Extracción del fondo usando Kinect	53
2.29.	Extracción del fondo usando Mask RCNN	54
2.30.	Identificación de postura con mapas de calor	56
2.31.	Arquitectura de relojes de arena	57
2.32.	Análisis secuencial para HAR	59
2.33.	Análisis por volumen para HAR	60
3.1.	Diagrama general del sistema	66
3.2.	Flujo general de información del sistema	70
3.3.	Diagrama de la pila de datos	73
3.4.	Estructura general de un componente extensible	75
3.5.	Ciclo de vida de la entrada de datos	77
3.6.	Ciclo de vida del proceso de reconocimiento de actividades	81

3.7. Ciclo de vida del análisis de eventos	85
3.8. Ciclo de vida de una notificación	90
3.9. Árbol de directorios	92
3.10. Modelo de clases del sistema	94
3.11. Modelo de despliegue del sistema	95
3.12. Archivos de un componente controlador	96
3.13. Archivos de un nuevo controlador	99
3.14. Archivos de un reconocedor de actividades	101
3.15. Archivos de un nuevo clasificador HAR	105
3.16. Archivos de un analizador de eventos	106
3.17. Archivos de un nuevo analizado de eventos	109
3.18. Archivos de un notificador	111
3.19. Archivos de un nuevo notificador	113
4.1. Modelo de clases de CamController	121
4.2. Captura y transformaciones del CamController	123
4.3. Modelo de clases de ColorInfarctRecognizer	125
4.4. Red neuronal usada por ColorInfarctRecognizer	126
4.5. Flujo de reconocimiento de infartos del módulo SkeletonInfarctRecognizer	127
4.6. Modelo de clases de SkeletonInfarctRecognizer	128
4.7. Modelo de clases de MailChannel	129

ÍNDICE DE TABLAS

3.1.	Atributos de datos a transmitir	69
3.2.	Parámetros para consultar datos de la pila	72
3.3.	Métodos adicionales del la pila	74
3.4.	Atributos del mensaje a transmitir por los canales de notificación	91
3.5.	Archivo de configuración controlador	96
3.6.	Archivo de configuración clasificador	102
3.7.	Archivo de configuración analizador	107
3.8.	Archivo de configuración del componente notificador	112
3.9.	Métodos de la clase Misc	116
4.1.	Archivo de configuración de CamController	122
4.2.	Archivo de configuración de CamController	122
4.3.	Archivo de configuración de ColorInfarctRecognizer	125
4.4.	Archivo de configuración de SkeletonInfarctRecognizer	128

ÍNDICE DE LISTADOS

3.1.	Firma del método “ <i>getJSON</i> ” de la clase Data	68
3.2.	Firma del método “ <i>parse</i> ” de la clase Data.	68
3.3.	Estructura de datos en JSON.	69
3.4.	Firma del método “ <i>valueOf</i> ”de la clase Data.	69
3.5.	Firma del método “ <i>serialize</i> ”de la clase Data.	70
3.6.	Firma del método “ <i>deserialize</i> ”de la clase Data.	70
3.7.	Firma del método “ <i>sendData</i> ”de la clase CommPool	70
3.8.	Firma del método “ <i>receive</i> ”de la clase CommPool	70
3.9.	Firma del método “ <i>log</i> ”.	71
3.10.	Ejemplo de parámetros para enviar datos la pila.	71
3.11.	Ejemplo de parámetros para consultar la pila.	72
3.12.	Firma del método “ <i>sendCommand</i> ”.	74
3.13.	Firma del método “ <i>getTime</i> ”.	74
3.14.	Firma del método “ <i>getTimeDiff</i> ”.	74
3.15.	Firma del método “ <i>isLive</i> ”.	74
3.16.	Firma del método “ <i>count</i> ”	74
3.17.	Firma del método “ <i>checkConnection</i> ”.	76
3.18.	Firma del método “ <i>log</i> ”.	76
3.19.	Firma del método “ <i>start</i> ”.	79
3.20.	Firma del método <i>preLoad</i> de DeviceController.	79
3.21.	Firma del método “ <i>getDeviceList</i> ”.	79
3.22.	Firma del método “ <i>initializeDevice</i> ”.	79
3.23.	Firma del método “ <i>getData</i> ”.	80
3.24.	Firma del método “ <i>send</i> ”.	80
3.25.	Firma del método “ <i>check</i> ”	80
3.26.	Firma del método “ <i>showData</i> ”.	80
3.27.	Firma del método “ <i>simulateData</i> ”.	80
3.28.	Firma del método “ <i>stop</i> ”.	80
3.29.	Firma del método “ <i>start</i> ” de la clase ActivityRecognizer.	83
3.30.	Firma del método “ <i>preLoad</i> ” de la clase ActivityRecognizer.	83
3.31.	Firma del método “ <i>loadModel</i> ” de la clase ActivityRecognizer.	83
3.32.	Firma del método “ <i>loaded</i> ” de la clase ActivityRecognizer.	83
3.33.	Firma del método “ <i>bring</i> ” de la clase ActivityRecognizer.	84
3.34.	Firma del método “ <i>predict</i> ” de la clase ActivityRecognizer.	84
3.35.	Firma del método “ <i>send</i> ” de la clase ActivityRecognizer.	84
3.36.	Firma del método “ <i>check</i> ” de la clase ActivityRecognizer.	84
3.37.	Firma del método “ <i>showData</i> ” de la clase ActivityRecognizer.	84
3.38.	Firma del método “ <i>simulateData</i> ”.	84
3.39.	Firma del método “ <i>stop</i> ”.	84
3.40.	Firma del método “ <i>start</i> ”de la clase EventAnalyzer.	87
3.41.	Firma del método “ <i>preLoad</i> ”de la clase EventAnalyzer.	87

3.42. Firma del método " <i>loadModel</i> " de la clase EventAnalyzer.	88
3.43. Firma del método " <i>loadChannels</i> " de la clase EventAnalyzer.	88
3.44. Firma del método " <i>loaded</i> " de la clase EventAnalyzer.	88
3.45. Firma del método " <i>bring</i> " de la clase EventAnalyzer.	88
3.46. Firma del método " <i>analyze</i> " de la clase EventAnalyzer.	88
3.47. Firma del método " <i>notify</i> " de la clase EventAnalyzer.	88
3.48. Firma del método " <i>check</i> " de la clase ActivityRecognizer.	88
3.49. Firma del método " <i>showData</i> " de la clase EventAnalyzer.	89
3.50. Firma del método " <i>simulateData</i> ".	89
3.51. Firma del método " <i>stop</i> " de la clase EventAnalyzer.	89
3.52. Firma del método " <i>start</i> " de la clase NotificationChannel.	90
3.53. Firma del método " <i>loaded</i> " de la clase NotificationChannel.	90
3.54. Firma del método " <i>notify</i> " de la clase NotificationChannel.	90
3.55. Firma del método " <i>preNotify</i> " de la clase NotificationChannel.	90
3.56. Firma del método " <i>tryNotify</i> " de la clase NotificationChannel.	90
3.57. Estructura del archivo de configuración de dispositivos.	96
3.58. Plantilla para un nuevo controlador de dispositivos.	97
3.59. Implementación del método 'getData' en un nuevo controlador.	99
3.60. Plantilla para un nuevo Reconocedor de actividades.	101
3.61. Implementación del método 'predict' en un nuevo clasificador.	105
3.62. Plantilla para un nuevo Analizador de Eventos.	106
3.63. Implementación del método 'analyze' en un nuevo analizador.	110
3.64. Plantilla para un nuevo Canal de notificaciones.	111
3.65. Implementación del método 'send' en un nuevo componente notificador.	113
4.1. Cámaras disponibles.	121
4.2. Predicción de infarto del módulo ColorInfarctRecognizer.	125
4.3. Predicción de infarto del módulo SkeletonInfarctRecognizer.	127

ÍNDICE DE ALGORITMOS

CAPÍTULO 1

PLANTEAMIENTO DEL PROBLEMA

1.1. INTRODUCCIÓN

Según un estudio del banco mundial, la edad promedio de a nivel mundial va en aumento, en algunos países (por ejemplo Italia, Grecia o Japón) la población mayor a 65 años supera incluso el 20 %, y el porcentaje tiende a aumentar [WorldBank2019]. En relación directa a esto, el número de hogares constituidos por más de una persona va en descenso, teniendo países como Argentina y Uruguay donde las familias unipersonales superan incluso el porcentaje de familias monoparentales [Arriagada2007].

Adicional al aumento de la edad promedio, sobre todo en países más desarrollados, también se ha proliferado el hecho de encontrar más ancianos viviendo solos y en muchos casos desconectados de amigos o familiares. ‘Kodokushi’ es un término utilizado en Japón para nombrar un fenómeno que apareció hacia los años 80, y que ha venido en aumento desde entonces, como es la muerte solitaria. Este fenómeno se hizo noticia en Japón en el año 2000, cuando un hombre de 69 años fue encontrado muerto en su casa, después de 3 años de fallecer, sin que nadie sintiera su ausencia, como reporta el New York Times [Onishi2017]. Incluso se ha reportado que bajo esta condición, solo en Japón, en el 2009 murieron 32.000 personas [Allison2014]. En occidente las cosas no distan mucho, en Barcelona los bomberos encuentran cada año más de 100 personas muertas en soledad [Sanchez2016]. Incluso para Inglaterra ya es un tema de estado, pues se estima que la mitad de las personas de 75 o más, viven en soledad, por lo que a inicios de 2018 se creó el ministerio de la soledad, según indica la BBC [BBC2018].

Simultáneamente, el desarrollo actual ha permitido que la tecnología sea omnipresente, entonces, ¿por qué no aprovechar esa facilidad y monitorizar cualquier evento anómalo y perjudicial para las personas en su hogar?

1.2. MOTIVACIÓN / JUSTIFICACIÓN

Muchas personas ancianas en capacidad de elegir prefieren vivir solas en sus casa que hospedarse en un asilo de ancianos. Sin embargo, es evidente que una persona de mayor edad y que vive sola, se encuentra en un estado de menor protección; comparado con personas acompañadas, frente a problemas físicos o emocionales.

Por otro lado, no es extraño que dentro del propio hogar ocurran eventos puntuales que pueden desencadenar problemas más graves o incluso la muerte de la persona, tal es el caso de una caída o de un ataque cardiaco. Por tal razón, la identificación de estados perjudiciales para el bienestar de la persona, mediante sistemas automatizados, ha adquirido gran relevancia en diferentes campos de investigación [Fragapanagos2005], y especialmente cuando se trata de personas que viven solas y son de edad avanzada. Todo esto presenta tres problemáticas principales:

No existen herramientas automatizadas que reconozcan eventos perjudiciales de una persona anciana y a su vez puedan alertar sobre posibles urgencias y la causa de las mismas. Revisar este párrafo pues si existen. Es muy costoso contar con personal, que esté de tiempo completo supervisando la actividad de personas mayores. El promedio de edad en los países desarrollados o en vía de desarrollo va en aumento.

Así pues, contar con un mecanismo que permita hacer una supervisión de la persona y, además, que puedan de alguna forma mitigar el impacto causado por situaciones anómalas y perjudiciales, se ha convertido en el foco de atención para gobiernos y para instituciones de investigación. Tal es el caso de “Horizonte2020”, un programa de la unión europea para el desarrollo de la investigación y que dentro de sus objetivos cuenta con uno específico para la “Salud, cambio demográfico y bienestar” [Horizonte2020].

El presente trabajo presenta dos aportes principales con vistas a solventar las problemáticas presentadas. En primera instancia, un marco de trabajo en el área de reconocimiento de actividad humana (HAR), para el desarrollo de módulos, con capacidades específicas dentro del HAR (esto se llama lóbulo en el cerebro), con solo entradas y salidas con una estructura estandarizada. Con esto se consigue, soluciones altamente reutilizables, pero manteniendo el método usado para el reconocimiento, independiente y, por tanto, flexible para cada módulo. Esto permitirá construir un sistema altamente escalable, el cual tiene a modo de sistema nervioso, la capacidad de conectar los diferentes lóbulos; donde la estandarización de entradas y salidas permitirá, no solo que el sistema central pueda recibir información de cada lóbulo sino también, compartirla entre ellos. Mejorar y profundizar.

El segundo valor aportado, es un sistema que permite integrar el análisis / estímulo recibido desde diferentes lóbulos dentro de un solo entorno. Al permitir integrar módulos independientes, se podrá combinar investigaciones con diferentes capacidades en el área de reconocimiento de actividad humana, como son: la detección de infarto, de caídas u otras actividades, en un solo conjunto de eventos reconocidos para así, identificar supra estados que caracterizan un evento que supera la simple etiqueta de “se cayó” o “está caminando”. Emergiendo información contextual generada al combinar la información entregada por cada módulo HAR para; mediante el análisis de todos los eventos detectados y recibidos en conjunto, identificar si un comportamiento es anómalo y, dentro de esta misma anormalidad, identificar si debe ser considerado como peligroso o perjudicial y, por tanto, deba realizar una acción como enviar un mensaje de alerta a algún organismo encargado.

Cabe aclarar que, si bien es cierto que se mencionó a las personas mayores como un grupo especialmente vulnerable, sobre todo cuando no se encuentran acompañados, este sistema podrá beneficiar a cualquier persona, sin importar su edad, pues genera acciones que, en caso de emergencia, puedan alertar a sistemas de salud, de seguridad o personas cercanas y, así, recibir ayuda de forma oportuna.

El enfoque del presente trabajo abre la puerta para el desarrollo de sistemas de inteligencia artificial altamente escalables, permitiendo la integración de métodos, lo cual permite a su vez la colaboración de una comunidad en la construcción de un sistema mucho más potente que el desarrollado en el presente trabajo como evidencia de sus capacidades.

Además de la integración compartida de código y métodos, también permitirá centralizar la construcción de bancos de entrenamientos para que personas con conocimientos en el área de la inteligencia artificial pueda construir sus propios sistemas de aprendizaje y, a su vez, los pueda compartir con la misma comunidad. (Aún no tan seguro para este proyecto pues puede extender el alcance demasiado. Se debe analizar la forma más conveniente de lograrlo).

1.3. HIPÓTESIS

- **H0:** Es posible, mediante el análisis de imágenes en vídeo determinar algún evento individual que afecte la calidad de vida de una persona.
- **H1:** Es posible, mediante la combinación de sistemas de inteligencia artificial, entrenados de forma independiente, identificar eventos complejos en la actividad humana.

1.4. OBJETIVOS

Objetivo General

Construir un sistema que permita integrar dentro de un único marco de trabajo métodos y algoritmos, con capacidades individuales en el reconocimiento de actividades humanas, para identificar supra eventos que puedan surgir de la combinación de las inferencias realizadas por cada subsistema.

Objetivos específicos

Identificar algoritmos para la identificación y extracción de personas de una imagen o un vídeo.

Crear un modelo y la plantilla de trabajo correspondiente, para la construcción de módulos aplicable al reconocimiento de actividades humanas.

Construir algoritmos que permitan identificar eventos puntuales en actividades humanas.

Diseñar y construir un software que permita la integración de los resultados obtenidos de forma independientes por los algoritmos de identificación en cada una de las cámaras que formen parte del sistema.

Diseñar una plataforma física multi-cámara que sirva para la monitorizar la actividad de las personas en su hogar.

1.5. ALCANCE

El proyecto se limita a la identificación de 2 eventos específicos, como modo de demostración del método desarrollado en este trabajo.

El sistema de identificación de eventos anómalos principal hará uso solo de los módulos de identificación de eventos creados en este mismo trabajo, para con ellos identificar eventos anómalos y ejecutar una acción de notificación por un medio de comunicación tradicional.

Aunque la arquitectura del sistema permite, por su diseño, sensores de diferente naturaleza como, micrófonos, acelerómetros u otros, el sistema principal y módulos adicionales solo será validado usando cámaras RGB tradicionales.

CAPÍTULO 2

MARCO REFERENCIAL

2.1. MARCO CONCEPTUAL

2.1.1. Inteligencia artificial

“El estudio de cómo producir máquinas que tengan algunas de las cualidades que tiene la mente humana, como la capacidad de comprender el lenguaje, reconocer imágenes, resolver problemas y aprender.” [CambridgeDicAI].

La inteligencia artificial (I.A.) es una de esas ciencias que ha acompañado a la humanidad por un largo tiempo, o al menos el deseo de ella y ciertamente las herramientas requeridas que pueden datar de milenios. La I.A. une conceptos con origen tan distantes en el tiempo como la lógica de los antiguos griegos, o la moderna computación. También materias que en principio parecen de aplicación tan lejana como la filosofía y la neurociencia, además de muchas otras como la algoritmia, la lingüística, las ciencias de la información y hasta la evolución natural. Todo en un mezcla maravillosamente engranada como una de las más grandes proezas de la humanidad.

Hoy en día se reconoce la lógica silogística de Aristóteles como uno de los primeros intentos de identificar bases cómo funciona el cerebro humano de manera estructurada y por medio de un conjunto de reglas, silogismos, que describen una parte del funcionamiento de la mente humana. Muchos más avances en el entendimiento de la mente fueron necesarios, entre ellos la propuesta de René Descartes, para explicar a los animales como máquinas complejas.

Pero también fue necesario del desarrollo de máquinas como la primera calculadora creada por Blaise Pascal, posteriormente mejorada por Leibniz. Así mismo la creación de autómatas muy sofisticados para cada época, siempre han sido relacionados con la inteligencia, en el siglo XVIII el relojero Pierre Jaquet-Droz creó algunos que aun hoy impresionan por su nivel de detalle y precisión como el escritor, el dibujante o el músico, actualmente expuestos en el museo de arte e historia de Neuchâtel en suiza.

Sin duda una de las bases fundamentales de los sistemas de computación actual desde el punto de vista matemático es el aporte que hizo George Boole con el establecimiento de la lógica proposicional o lógica booleana, pocas décadas después extendida por Gottlob Frege.

El siglo XX fue sin duda, el siglo en que todos los avances tanto filosóficos, matemáticos y técnicos se juntaron para dar nacimiento a lo que actualmente se conoce como inteligencia artificial, si bien es cierto que aun no existe una definición completamente uniforme.

Hacia la década de los veinte del siglo pasado el escritor Gottlob Frege acuñó el término Robot para referirse a autómatas pseudo inteligentes que reemplazarían el al ser humano en el trabajo. Término que hoy se usa muy frecuentemente en el campo de la informática, y de la automatización en general, y al que la ciencia ficción suele atribuir inteligencia similar sino superior a la humana.

En el año 1936 Alan Turing estableció las bases teóricas de la computación, lo que posteriormente llevó al concepto de máquina de Turing. Una máquina que conceptualmente podría realizar operaciones a partir de instrucciones escritas en una cinta, lo que terminaría por convertirse en el precursor de los ordenadores actuales.

Generalmente el trabajo de Warren McCulloch y Walter Pitts en 1943 [**McCulloch1943**] es considerado como el primero en el campo de la inteligencia artificial. Ellos basaron su trabajo en fisiología básica, la función de las neuronas en el cerebro y la lógica proposicional.

Antes de que las definiciones sobre qué es inteligencia artificial, se hicieran más o menos homogéneas, el gran Allan Turing planteó un problema que serviría como referente para poder establecer si una máquina era inteligente o no. En 1950 Turing en su artículo 'Computing Machinery and Intelligence', publicó el conocido test de Turing [**Turing1950**] que, en términos generales, planteaba la posibilidad de establecer una conversación, usando un canal indirecto, entre una máquina y una persona sin que ésta última pudiera discernir si su interlocutor era también un ser humano o no.

En 1958, John McCarthy inventa LISP, el primer lenguaje para inteligencia artificial. Ese mismo año Rosenblatt presentó el 'Perceptrón' [**Rosenblatt1958**], un concepto inspirado en las redes neuronales del cerebro animal.

La pregunta fundamental planteada por Turing: ¿puede pensar una máquina? ha llevado a diferentes definiciones sobre qué es el pensamiento y qué es la inteligencia.

"La automatización de actividades que asociamos con el pensamiento humano, actividades como la toma de decisiones, la resolución de problemas, el aprendizaje..." [**Bellman1978**].

Durante la segunda mitad del siglo XX se propusieron diferentes técnicas y algoritmos en el campo de la inteligencia artificial: Sistemas expertos, algoritmos genéticos, entre otros y también se crearon lenguajes de programación como Lisp y Prolog. Así mismo nuevas definiciones sobre qué es la I.A. han sido planteadas.

"La teoría y el desarrollo de sistemas informáticos capaces de realizar tareas que normalmente requieren inteligencia humana, como la percepción visual, el reconocimiento de voz, la toma de decisiones y la traducción entre idiomas." [**OxfordDicAI**].

"La inteligencia artificial (IA) es una disciplina académica relacionada con la teoría de la computación cuyo objetivo es emular algunas de las facultades intelectuales humanas en sistemas artificiales." [**benitez2014AI**]

Por su parte, Stuart Russell y Peter Norvig dividen la inteligencia artificial en seis disciplinas [**Russell2009**]:

- **procesamiento del lenguaje natural** para permitirle comunicarse con éxito.
- **Representación del conocimiento** para almacenar lo que sabe o escucha.
- **Razonamiento automatizado** para usar la información almacenada para responder preguntas y sacar nuevas conclusiones.
- **Aprendizaje automático** para adaptarse a nuevas circunstancias y detectar y extraer patrones.
- **Visión por computadora** para percibir objetos, y
- **robótica** para manipular objetos y moverse.

A partir de las diferentes definiciones es posible enmarcar a la inteligencia artificial como la capacidad que se brinda a un ordenador u otra máquina para realizar acciones que requieren inteligencia. Entre estas acciones están la capacidad de tomar decisiones basadas en la experiencia

aun con información insuficiente o conflictiva, la capacidad de comprender el lenguaje humano y la capacidad de reconocer patrones en un conjunto de datos dado.

2.1.2. Aprendizaje automático (Machine learning)

“Campo de estudio que provee a los ordenadores la habilidad de aprender algo sin ser explícitamente programado.” [Samuel1959]. (Traducción propia).

- **¿Qué es aprender?** - “**Aprendizaje** es un término muy general que denota la forma, o formas, en la cual un animal (o una máquina) aumenta su conocimiento y mejora sus capacidades de actuación (‘performance’) en un entorno. De esta manera, el proceso de aprendizaje puede ser visto como un generador de cambios en el sistema que aprende - que por otra parte ocurren lentamente, adaptativamente - y que pueden ser revocados o ampliados.” [Moreno1994aprendizaje].

El Aprendizaje automático (en inglés Machine learning) también conocido como aprendizaje de máquina, es una rama de la Inteligencia artificial que conociste, de forma simplificada, en la creación de algoritmos que permiten dotar a los ordenadores con la capacidad de realizar tareas para las cuales no fueron explícitamente programados, es decir, aprenden de forma automática.

En un sentido más generalista para el campo del aprendizaje automático, es válido decir que un sistema debe aprender a separar la señal del ruido en los datos, de forma que se construya un engrama o modelo que se ajuste más acertadamente al patrón en los datos.

Los modelos generados en el aprendizaje automático son esencialmente paramétricos y el conocimiento se sustenta en el ajuste del valor de los parámetros. Es en la etapa de entrenamiento donde se ajustan los parámetros del modelo y es justo aquí donde la asociación o generalización de un comportamiento (salida) se asocia con el estímulo recibido (entrada).

Así pues, en este contexto, el aprendizaje automático consiste en la identificación de patrones a partir de una muestra de datos de entrenamiento. Durante la etapa de entrenamiento, los sistemas que utilizan técnicas de aprendizaje automático, son capaces de tomar los datos y extraer de ellos las características intrínsecas de los mismos, de forma que puedan predecir comportamientos futuros para datos no observados mientras se realiza el entrenamiento. De igual forma, el Machine learning implica mejoramiento del sistema sin la intervención humana.

“Se puede decir que un programa aprende a partir de la experiencia ‘E’, respecto a una tarea ‘T’ y alguna medida de rendimiento ‘P’, si el rendimiento en ‘T’, medido por ‘P’, mejora con experiencia ‘E’.” [Mitchell1997]. (Traducción propia).

El Aprendizaje automático se encuentra presente hoy día en todos los sistemas de entretenimiento o venta en línea, de relevancia mundial, como pueden ser las recomendaciones de productos en la tienda de Amazon o de contenido en la plataforma Netflix. Pero incluso en los asistentes virtuales de los teléfonos móviles u otros como el dispositivo Google Home, Alexa o Siri utilizan las técnicas de Machine learning para que el comportamiento de estos asistentes sea cada vez más cercano a la persona que los usa, así por ejemplo, en el reconocimiento de comandos por voz puede ir aprendiendo la forma en que una determinada persona pronuncia algunas palabras y incluso la intención o emoción como las dice.

En conclusión se podría decir que Machine learning es un maestro del reconocimiento de patrones que, a partir de un conjunto de datos es capaz de inferir las reglas que intrínsecas, para que un ordenador aprenda de los mismos, sin la necesidad de que un humano defina y programe dichas reglas de forma explícita en un software, dotándolo con esto, además, de la capacidad de adaptarse a los propios datos e incluso cambiar con el tiempo sin intervención humana.

2.1.3. Aprendizaje supervisado

“En el aprendizaje supervisado, el objetivo es aprender un mapeo de la entrada a una salida cuyos valores correctos son proporcionados por un supervisor.” [alpaydin2014ML] (Traducción propia).

El aprendizaje supervisado es una técnica dentro del aprendizaje automático que permite deducir la función que mejor se ajusta a una muestra de vectores emparejados Entrada-Salida, es decir, ajustar los parámetros de un modelo de forma que éste permita tomar la entrada, ejecutar el conjunto de operaciones dentro del sistema y obtener valores que se acerquen lo más posible a la salida esperada.

En otras palabras, cuando se entrena un sistema, usando la técnica aprendizaje supervisado, se tiene un conjunto de datos de ejemplo (ver sección 2.4 para mayor detalle sobre Datasets) $x[0, 1, \dots, n]$ correspondientes a las entradas al sistema y un conjunto de salidas correspondientes $y[0, 1, \dots, n]$ donde cada entrada del vector x se asocia con una salida del vector y . El objetivo del aprendizaje supervisado es encontrar la función que permita generalizar los datos (modelo), de forma que, en un futuro, se pueda predecir el valor de salida correspondiente a cualquier entrada sin que haya visto dicha entrada durante el entrenamiento.

Debido a la naturaleza misma de este algoritmo, es necesario contar con un conjunto de datos suficientemente amplio para establecer la generalización, además la correcta etiquetación de cada dato de entrada con su salida correspondiente, de no ser así, se podrían generar un modelo de predicción poco confiable o que el aprendizaje nunca se dé. Es por esto que la preparación de los datos, constituye uno de los pasos fundamentales de este tipo de aprendizaje, siendo incluso el que consuma gran parte del tiempo en la construcción del sistema.

En el aprendizaje supervisado se suelen separar los modelos generados en dos categorías, dependiendo el tipo de salida obtenida: por un lado están los modelos de clasificación, que permiten obtener un valor que pertenece a un conjunto de posibles valores. Y por otro lado, los modelos de regresión, cuya salida es un valor de un espacio continuo.

Algunos de los ejemplos más notorios dentro de la Inteligencia artificial de métodos de aprendizaje supervisado son: Regresión logística, SVM, Redes neuronales, Árboles de decisión entre otros.

2.1.4. Aprendizaje no supervisado

“En el aprendizaje no supervisado, no existe tal supervisor y solo tenemos datos de entrada. El objetivo es encontrar las regularidades en la entrada.” [alpaydin2014ML] (traducción propia).

El aprendizaje no supervisado es un método de entrenamiento dentro del campo del aprendizaje automático que permite, a partir de un conjunto de datos sin etiquetas a priori, identificar las características de los elementos de la muestra, es decir, sin saber cual es la salida esperada, generar un modelo de densidad que permita caracterizar o agrupar los datos.

Un uso habitual de este tipo de métodos es la segmentación de conjuntos de datos por atributos compartidos, este uso está muy extendido dentro de los denominados ‘recomendadores’, algoritmos que permiten, por ejemplo, encontrar usuarios con gusto similares y así poder recomendar productos o contenidos que también han sido adquiridos por otros miembros de la misma categoría. Un ejemplo de este tipo de métodos es el algoritmo K-means que permite crear agrupaciones a partir de la cercanía de las características de los datos.

Otro uso habitual es detección de anomalías que no encajan en ningún grupo, por ejemplo la identificación de células cancerosas [kakushadze2017]. Pero también pueden ser usadas para la simplificación de conjuntos de datos, agrupando elementos con atributos similares, esto tiene especial utilidad, cuando el coste de preparar los datos (etiquetar) es muy alto o se desconoce la salida esperada.

Por tanto, se puede decir que esta técnica busca identificar la estructura intrínseca de los datos. En el aprendizaje no supervisado se suelen separar los modelos generados en dos categorías, dependiendo el tipo de salida obtenida: agrupación y reducción de la dimensionalidad.

2.1.5. Aprendizaje reforzado

“Un proceso de prueba y error basado en recompensas, en el que un sistema aprende a interactuar con un entorno complejo para lograr resultados gratificantes, se conoce en lenguaje de aprendizaje automático como aprendizaje de refuerzo. En el aprendizaje por refuerzo, el proceso de prueba y error es impulsado por la necesidad de maximizar las recompensas esperadas a lo largo del tiempo.” [Aggarwal2018] (traducción propia).

En I.A. se espera que un sistema que cuenta con aprendizaje automático 'aprenda' a tomar las decisiones correctas, incluso en situaciones cambiantes. Por ejemplo, cuando se trata de vehículos autónomos, en estos casos es natural suponer que no se puede contar, a priori, con la secuencia de acciones correctas a realizar debido principalmente a la gran cantidad de variaciones que el propio entorno puede tener. Con esto es difícil modelar las acciones del sistema debido a que debe ser una secuencia de acciones dinámicas.

“La suposición clave aquí es que estos sistemas son demasiado complejos para modelar explícitamente, pero son lo suficientemente simples como para evaluarlos, de modo que valor de recompensa se puede asignar para cada acción.” [Aggarwal2018] (traducción propia).

En forma general se puede decir que el aprendizaje reforzado consiste en un mecanismo en el que un agente va adaptando su comportamiento tras cada acción que realiza, mediante la recompensa (o castigo). Es decir, si una acción en concreto es beneficiosa dentro del objetivo general se otorga un puntaje, de esta forma, se va construyendo la política de acción. Un ejemplo de este tipo de aprendizaje es algoritmo Q-learning.

“Un buen ejemplo es un videojuego, donde un solo movimiento por sí solo no es tan importante; sino que es la secuencia de movimientos correctos lo que lo hace bueno. Un movimiento es bueno si es parte de una buena política de juego.” [alpaydin2014ML] (Traducción propia).

Los videojuegos durante los últimos años se han convertido en los entornos de pruebas para este tipo de algoritmos, pues presenta una gran variabilidad en un ambiente controlado y que justamente permite probar la premisa de 'Aprendizaje por recompensa'. Considerando el caso del clásico juego 'Pacman' en el el personaje central en cada momento del juego solo puede hacer una de cuatro posibles acciones, avanzar a la izquierda, a la derecha, arriba o abajo. Sin embargo, de la acción de realice en cada momento dependerá que reciba más punto o incluso pierda la partida. Al tomar en consideración los posibles estados en los que el personaje central esté en una de las posibles posiciones dentro del tablero de juego y los 'fantasmas' estén cada uno en su posición, genera una combinación de posibilidades tan grande que hace inviable definir las acciones a tomar en cada estado. Es aquí donde el aprendizaje por refuerzo permite identificar la mejor acción para cada estado, dependiendo la recompensa que reciba al realizar el movimiento.

Es importante considerar que este tipo de algoritmos no indican una 'verdad absoluta', en cuanto que la secuencia de movimiento es la mejor posible en un entorno, simplemente es una lo suficientemente buena para cumplir el objetivo.

2.1.6. Aprendizaje profundo (Deep learning)

Aprendizaje profundo (en inglés, Deep learning) es un conjunto de algoritmos dentro de la rama de Aprendizaje automático que busca la generación de modelos que permitan inferencias de alto nivel a partir de grandes conjuntos de datos. Para esto hace uso de arquitecturas computacionales de múltiples capas, que realizan transformaciones no lineales y de forma iterativa a los datos de entrada.

“La idea detrás de las arquitecturas profundas es que pueden aprovechar mejor las regularidades repetidas en los patrones de datos para reducir el número de unidades computacionales y, por lo tanto, generalizar el aprendizaje incluso a áreas del espacio de datos donde no hay ejemplos.” [Aggarwal2018]. (Traducción propia).

El concepto de ‘profundo’ se debe esencialmente a la cantidad de transformaciones aplicadas, respecto de otros métodos de aprendizaje. Además, cada transformación cuenta con umbrales y pesos que son modificados durante el entrenamiento. Aunque no hay una definición clara de cuántas transformaciones son requeridas para poder ser llamado profundo, algunos de los modelos actuales dentro de esta categoría pueden superar el millón de parámetros entrenables.

“Un MLP (*Perceptrón multicapa*) con una sola capa oculta tiene una capacidad limitada, mientras que el uso de un MLP con múltiples capas ocultas puede aprender funciones más complicadas de la entrada. Esa es la idea detrás de las redes neuronales profundas donde, a partir de la entrada sin procesar, cada capa oculta combina los valores en su capa anterior y aprende funciones más complicadas.” [alpaydin2014ML]. (Traducción propia).

La profundidad misma, es decir, el número de transformación trae consigo un problema implícito, éste es el desvanecimiento gradual de la relevancia de los datos de entrada, pues su transformación acumulada es tan alta, que el impacto de los parámetros entrenables es mayor que el de los propios datos de entrada, consiguiendo, con esto, que la relevancia de la entrada se pierda a medida que se aumenta la profundidad, convirtiéndose en un sistema esencialmente determinista. Para subsanar el problema del desvanecimiento gradual, se han planteado diferentes estrategias (ver 2.2.3) y también se han definido arquitecturas como las redes residuales [Kaiming2015] (ver 2.1.10).

2.1.7. Redes neuronales (NN)

Breve historia

En el año 1943 Warren McCulloch y Walter Pitts presentaron un primer modelo para la creación de redes neuronales artificial [McCulloch1943], partiendo de tres fuentes: El conocimiento existente sobre el funcionamiento de las neuronas del cerebro y su fisiología, el análisis formal de la lógica proposicional de Russell y Whitehead y la teoría de la computación de Turing. Con esto presentaron un modelo constituido a su vez por modelos de neuronas artificiales, de forma que cada una de ellas pudiera ser representada como ‘activada’ o ‘desactivada’ a partir de los ‘estímulos’ (entrada / inputs) recibidos, siendo un modelo equivalente a las neuronas naturales. Además mostraron que cualquier función podría computarse usando la red de neuronas interconectadas ya que es posible con las estructuras de red propuestas, implementar las compuertas lógicas tradicionales: and, or, nor o cualquier otra. Este modelo sentó las bases para los futuros desarrollos de redes para la inteligencia artificial.

“Se puede crear una red neuronal artificial simulando un modelo de una red de neuronas en un computador. Al aplicar algoritmos que imitan los procesos de las neuronas reales, es posible hacer que la red ‘aprenda’ a resolver muchos tipos de problemas.” [Krogh2008].

Más adelante en 1949 Donald Hebb, publicó una hipótesis sobre los mecanismos en que las

neuronas interactúan para generar lo que se conoce como aprendizaje. Hebb a partir de sus estudios (entre los que se encuentran el estudio de la sinapsis basado en el famoso experimento de Pavlov) entendió, entre otras cosas, que una neurona no era capaz, por si sola, de excitar a otra, pero que el acto conjunto de varias si podría desencadenar en la activación o desactivación. Además plantea el modelo que define la plasticidad del cerebro:

“Si las entradas a un sistema producen el mismo patrón de actividad repetidamente, el conjunto de elementos activos que constituyen ese modelo llegarán a tener una interasociación más fuerte. Es decir, cada elemento tenderá a activar a otros elementos y (con pesos negativos) a inactivar a los elementos que no formen parte del patrón. En otras palabras, el patrón en su conjunto se convertirá en ‘auto-asociado’. Se puede decir que un aprendizaje (auto-asociado) es un patrón de engrama.” [Hebb1949].

De la hipótesis de Hebb se desprendió uno de los conceptos más tradicionales en el campo de la inteligencia artificial, el aprendizaje no supervisado. En poco tiempo algunos investigadores empezaron a aplicar las ideas de Hebb en el ámbito computacional. En 1954 Farley y Clark [Farley1954] construyeron un experimento para simular una red con los conceptos de Hebb usando máquinas computacionales. En 1958 Rosenblatt presentó el ‘Perceptrón’ [Rosenblatt1958], un algoritmo para el reconocimiento de patrones, basado en sus ideas de como la información acerca del mundo físico es almacenada e interpretada y como esa información influencia el reconocimiento de las cosas.

A pesar de los avances alcanzados, por al menos una década, el desarrollo del campo de las redes neuronales se estancó, en gran medida por dos factores, en primera instancia por falta del poder computacional y en segunda instancia, por falta de algoritmos que permitieran el ‘entrenamiento’ del Perceptrón. Fue hasta que Paul Werbos presentó en 1975 su algoritmo de propagación hacia atrás [Werbos1975] (Back propagation en inglés), que se logró una forma eficaz para el entrenamiento de las redes. El éxito de la propuesta de Werbos radica en utilizar la diferencia entre el resultado producido y el resultado deseado para cambiar los “pesos” de las conexiones entre las neuronas artificiales (Vea la sección 2.2.5, para un mayor detalle de este algoritmo.).

A pesar de la mejora en los algoritmos, el problema de poder computación siguió vigente hasta mediados de la década de los 2000, cuando se empezaron a alcanzar capacidades de computo más acordes a las necesidades, así mismo se pudo desarrollar redes cada vez más grandes. Y dada la naturaleza altamente paralelizable en el procesamiento de las redes neuronales, un nuevo avance en el poder de computo se logró tras la inclusión de las tarjetas de vídeo (GPU) en el procesamiento.

Actualmente las redes neuronales están siendo usados por investigadores de diferentes áreas, no solo donde los algoritmos tradicionales son poco efectivos, sino también en la solución de nuevos problemas como la detección de exo-planetas [NasaExoplanet2017].

Definición técnica

“Una red neuronal artificial (ANN) es un modelo computacional que está ligeramente inspirado en el cerebro humano, ya que consiste en una red interconectada de unidades de procesamiento simples (neuronas artificiales) que aprenden de la experiencia al modificar sus conexiones.” [VanGerven2017].

De forma general se puede decir que las redes neuronales son un modelo parámetrico, es decir, la representación matemática de un proceso o fenómeno con el cual se puedan realizar predicciones. En otras palabras, a partir de un conjunto x de valores de entrada es capaz de inferir un conjunto y de valores de salida.

De forma esquemática, las redes neuronales artificiales las neuronas están dispuestas en capas y, a su vez, cada capa está compuesta por nodos (neuronas), donde los parámetros de entrada de una capa corresponden con los valores de salida de la capa inmediatamente anterior, (algunas arquitecturas

pueden diferir en algunas partes de esta premisa, ver sección 2.1.7). Según su ubicación en la red, cada capa pertenece a una de los siguientes grupos: entrada, salida o capas ocultas, como se ve en la figura 2.1.

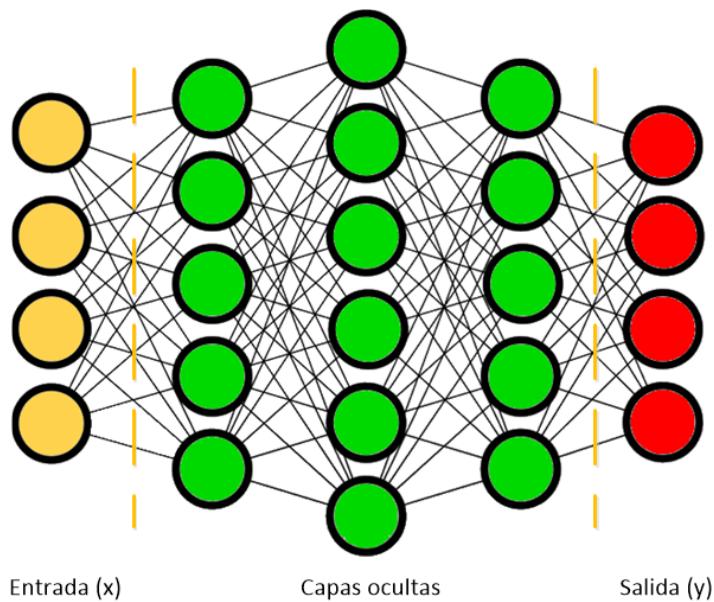


Figura 2.1: Modelo general de una red neuronal.

Antes de presentar el como se transmite o fluyen los datos en la red desde la entrada hasta la predicción, considere cada nodo o neurona de la red como el resultado de la ecuación 2.1.

$$y = f(b + \sum_{i=0}^n w_i x_i) \quad (2.1)$$

Donde y es el valor de salida de la neurona, f es una función de activación (vea 2.2.1 para mayor detalle), x el conjunto valores de entrada, w el conjunto de parámetros que ajustan el peso o importancia que tiene cada valor del conjunto x . Finalmente, b es un valor de sesgo que se aplica a la neurona misma, este sesgo se suele considerar como la conexión con otra neurona cuya salida siempre será 1.

La red neuronal más simple es el Perceptrón (ver figura 2.2). Ésta es simplemente una red de un única capa, esto se considera así, pues a los valores que recibe la capa de entrada no se aplica ningún ajuste sino que son transmitidos directamente a la siguiente capa.

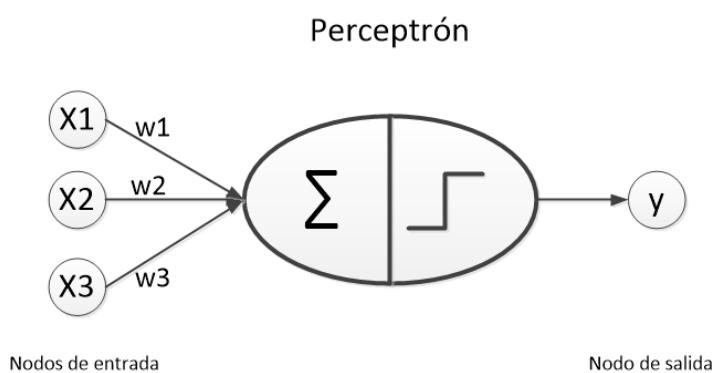


Figura 2.2: Modelo del Perceptrón.

A pesar de que un Perceptrón cuenta con todas las características fundamentales de una red neuronal, su limitada cantidad de parámetros no permite su aplicación a problemas con más dimensiones como es el caso de la clásica compuerta *Xor*, en su lugar se suelen usar redes con un número mucho más amplio de capas y de neuronas como se muestra en la figura 2.3, esto es conocido como el 'Perceptrón multicapa'.

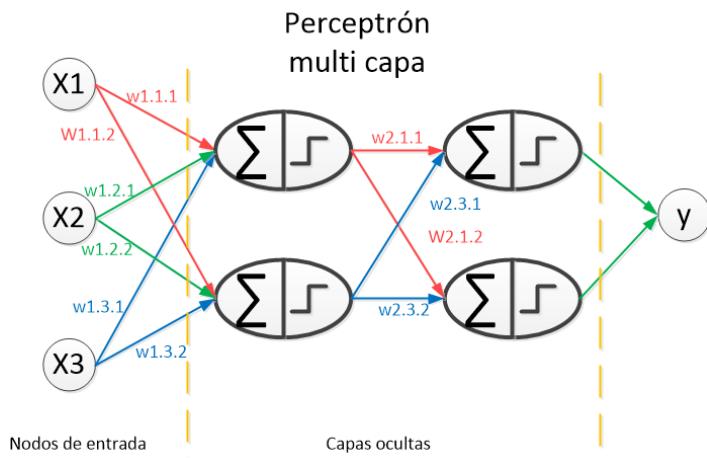


Figura 2.3: Modelo del Perceptrón multi capa.

Flujo de datos en la red

El flujo de datos de la red consiste en tomar los valores de entrada x e ir pasándolos de una capa c a la siguiente $c + 1$ hasta llegar a la salida y , aplicando para cada neurona la ecuación 2.1, para eso, las redes neuronales hacen uso del algoritmo de 'propagación hacia adelante' (puede ver este algoritmo en detalle en la sección 2.2.4).

Por otro lado, el objetivo fundamental de las redes neuronales, como de cualquier otro algoritmo de aprendizaje automático es reducir el error o lo también se conoce como función de pérdida (remitase a la sección 2.2.2 para mayor detalle sobre las funciones de pérdida). Para reducir el error se ajustan los parámetros w y b es decir, se cambia la influencia de cada entrada respecto a la diferencia entre el valor de salida obtenido y el valor esperado. Cuando se habla de valor esperado, se remite específicamente al conjunto de valores de entrenamiento de la red (ver sección 2.4 sobre conjuntos de datos de entrenamiento).

Al cambiar los parámetros de cada neurona es justamente cuando ocurre lo que se conoce como 'aprendizaje'. Considere lo siguiente, cuando una red neuronal se crea los valores de w y b son asignados con valores aleatorios, esto es porque los pesos o influencia de cada x no se conocen a priori y, por tanto, la salida obtenida también es un valor que no coincidirá con lo esperado.

Para que la red vaya aprendiendo es decir, que los parámetros se vayan ajustando de forma que la salida obtenida se acerque cada vez más a la esperada, se realiza el proceso conocido como entrenamiento, el cual se apoya de dos algoritmos principales: el 'descenso del gradiente' (explicado en la sección 2.26) y la 'propagación hacia atrás' (Explicado en la sección 2.2.5).

El 'descenso del gradiente', se encarga de encontrar el impacto del error acorde a la función de pérdida utilizada, para encontrar un valor menor o que generé un menor error. Mientras que el algoritmos de 'propagación hacia atrás' se encarga de ir propagando el ajuste a los parámetros, acorde al resultado del descenso del gradiente, desde la capa de salida (la que tiene el valor final obtenido en la red), hacia las capas anteriores de forma iterativa. De esta forma se identifica el error y el impacto causado en cada nodo para generar dicho error y se va ajustando durante los múltiples ciclos en que se repite la 'propagación hacia atrás' durante el entrenamiento.

Arquitecturas de redes neuronales

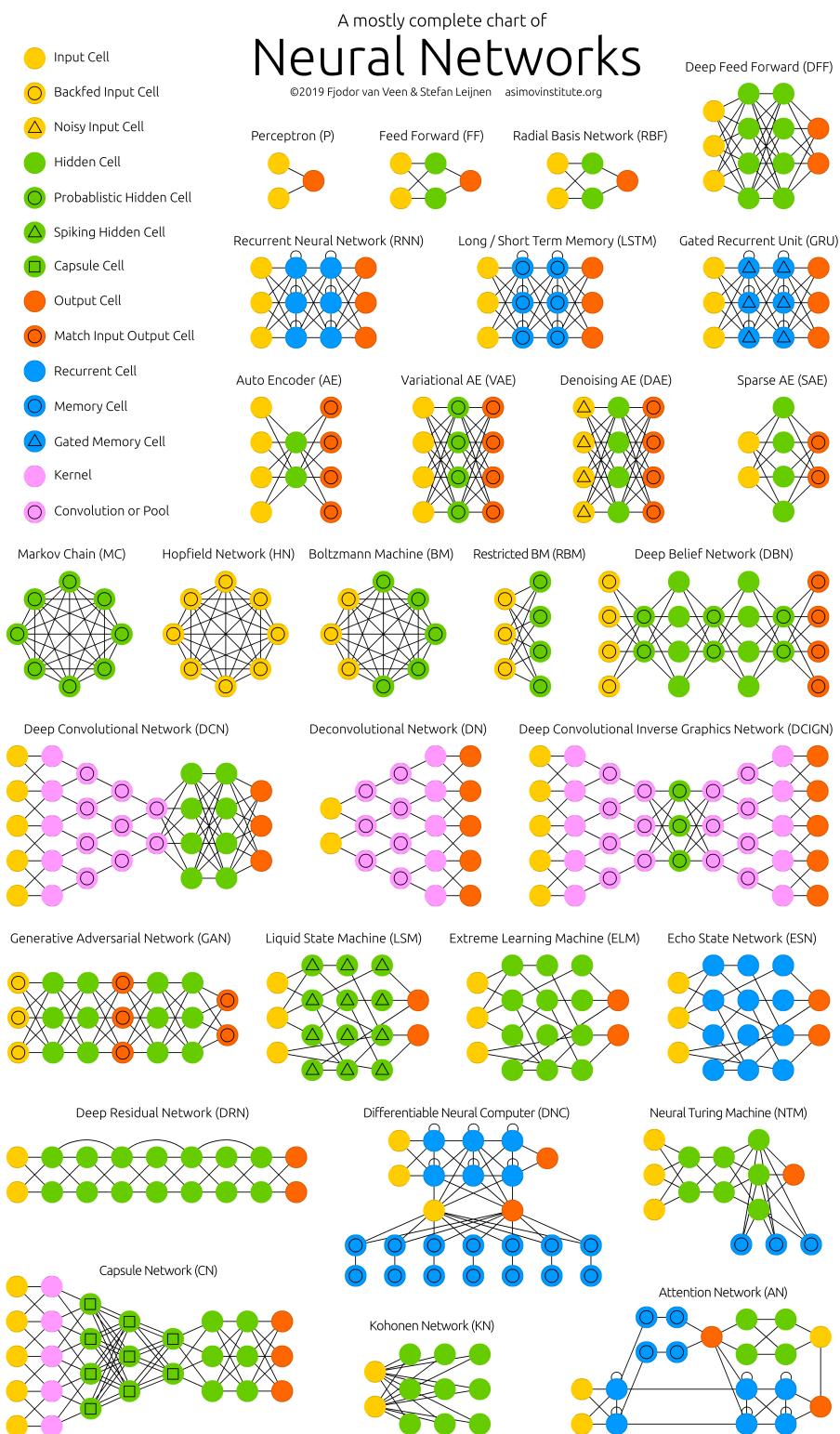


Figura 2.4: Listado de arquitecturas de redes neuronales.

La arquitectura de una red neuronal artificial es la forma o configuración como se organizan las neuronas en su interior, aunque también la forma en que el aprendizaje va llevarse a cabo. A lo largo del desarrollo de las redes neuronales se han propuesto diversas configuraciones, cada una con

características propias y que presentan un ventajas según el campo utilizado pero también la forma en que aprenden. Dependiendo del problema puede ser más efectiva una u otra arquitectura de red.

Debido al éxito de algunas de las arquitecturas de las redes neuronales, varias de ellas han perdido el prefijo de arquitectura y se suele atribuir un nombre independiente, es decir como un tipo de red propia, sin embargo todas ellas siguen mantenido una estructura que incluye o depende del diseño original presentado anteriormente. Algunas de las arquitecturas más exitosas en las ultimas décadas son: las redes convolucionales (ver sección 2.1.8), las redes recurrentes (ver sección 2.1.9) y las redes residuales (ver sección 2.1.10), sin embargo, existe un conjunto muy amplio de arquitecturas, la figura 2.4 una recopilación de arquitecturas presentada por 'The Asimov Institute' [NNZoo2019].

2.1.8. Redes convolucionales (CNN)

"Las redes neuronales convolucionales han sido históricamente las más exitosas de todos los tipos de redes neuronales. Se utilizan ampliamente para el reconocimiento de imágenes, detección / localización de objetos e incluso procesamiento de texto." [Aggarwal2018] (Traducción propia).

Las redes convolucionales son un tipo de arquitectura de red neuronal. Puede ser considerada una adaptación a las redes neuronales tradicionales a las cuales se les adicionan capas de convolución. Las capas de convolución, son un modelo inspirado en el descubrimiento de Hubel y Wiesel sobre las células encargadas de la visión en gatos [Hubel1962], donde se hace inicialmente se identifican patrones simples como bordes y a medida que se avanza en las capas se van identificando características cada vez más complejas, a medida que se procesa la imagen.

La primera implementación de este modelo en el campo de la visión por computador fue hecha por Fukushima y Miyake en el trabajo que presentan el Neocognitron [Fukushima1982], mostrando la capacidad de modelo para tolerar variaciones. Sin embargo, solo fue hasta el trabajo de LeCun [LeCun1998], en los años 90, que se integró con el algoritmo de 'propagación hacia atrás' pudiendo automatizar el aprendizaje de este tipo de capas.

Las redes convolucionales han demostrado una superioridad sobre las redes tradicionales en cuanto a las exactitud de sus predicciones, principalmente en el campo de la identificación de patrones en imágenes, debido principalmente a una mayor tolerancia a variaciones. LeCun [LeCun1998] demostró la eficacia de las redes convolucionales en el campo de la visión artificial tomando como caso de estudio la identificación de números escritos a mano. Para esto tomó un conjunto de imágenes de 28 x 28 píxeles en escala de grises.

Es importante considerar que el uso de imágenes hace el número de parámetros crecer muy rápido por ejemplo, para una imagen de 256x256 en escala de grises, se tendría en una capa tradicional 65.536 parámetros, o 3 veces ese valor si fuera una imagen RGB, para reducir la cantidad de parámetros es habitual poner capas con funciones de reducción como 'Max-pooling'.

Una ventaja de las redes convolucionales sobre las tradicionales, consiste en que la entrada de una red tradicional debe mantener características homogéneas, es decir, mantener una disposición de los elementos en cada imagen con muy poca variación, lo cual normalmente no es fácil, un ejemplo de ello es el caso tomado por Lecun, donde es necesario superar las variaciones que tiene cada persona en su forma de escribir.

Las capas convolucionales se apoyan en que las imágenes tienen una rígida estructura en 2D, lo cual significa una fuerte correlación local de los datos, en otras palabras, el valor de un píxel depende y tiene estrecha relación con el valor de sus vecinos. Esto presenta una ventaja significativa, pues permite reconocer características simples, como bordes o esquinas, antes de intentar identificar características espaciales, lo que les permite su gran tolerancia a variaciones de desplazamiento, escala o distorsión.

Arquitectura habitual de una red convolucional

Normalmente una red convolucional está compuesta por varias capas de convolución a la entrada de la red, intercaladas con capas de reducción y posteriormente varias capas de red neuronal tradicional como se ve en la figura 2.5.

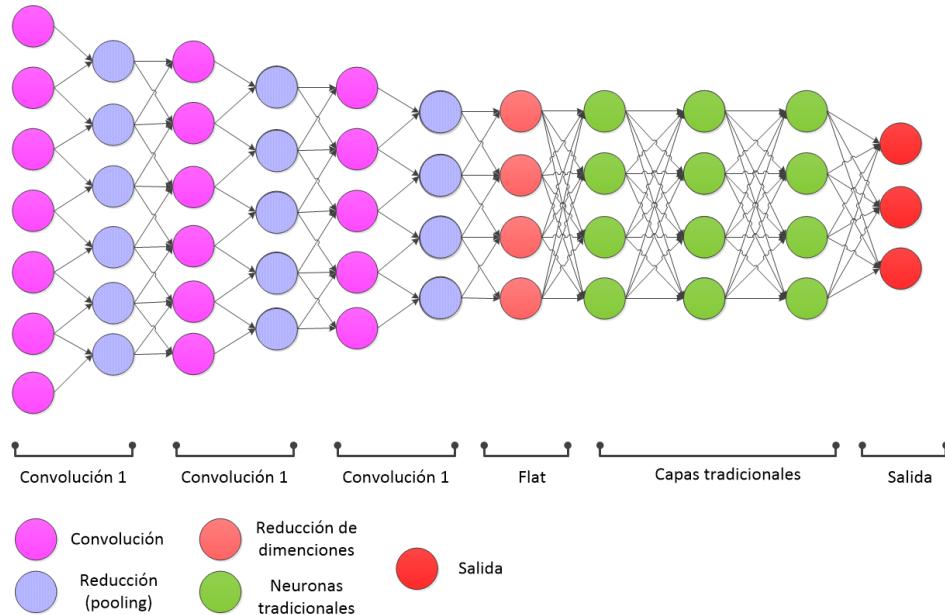


Figura 2.5: Arquitectura tradicional de una red convolucional.

La convolución

El proceso realizado en una capa convolucional consiste en realizar una operación de producto punto entre una matriz llamado kernel y las submatrices de la imagen cuyo centro es cada uno de los píxeles de la imagen, haciendo un desplazamiento de izquierda a derecha y de arriba a abajo de la imagen, como muestra la imagen 2.6, una vez realizada la multiplicación los valores obtenidos son sumados y pasado por una función de corrección no lineal (Típicamente ReLU, ver funciones de activación en la sección 2.2.1).

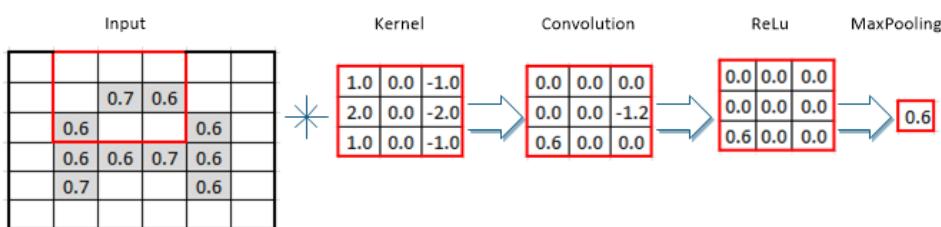


Figura 2.6: Pasos de una capa de convolución.

Relleno (Padding)

Es importante considerar que el desplazamiento de la matriz Kernel puede llevar a dos casos: primero, si el tamaño de las dimensiones del kernel es compatible con el tamaño de la matriz de entrada, es decir, cada dimensión de la matriz de entrada es múltiplo de su correspondiente en el kernel y, por tanto, el kernel podrá ir de inicio a fin sin que sobren elementos en una u otra. El problema de esto es que el kernel solo llegaría hasta el punto donde no exceda los bordes con lo que, en caso de una imagen se estaría perdiendo información de los bordes al no poder operar el kernel completamente.

El segundo caso ocurre cuando las dimensiones de la matriz de entrada y del kernel no son compatibles, haciendo que queden elementos sobrantes. En el caso de una imagen podrían quedar píxeles sin analizar.

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	2	3	5	7	0	0	0
0	0	8	5	3	2	1	0	0	0
0	0	2	4	6	4	2	0	0	0
0	0	9	7	5	7	9	0	0	0
0	0	9	6	3	2	1	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figura 2.7: Capas de relleno para convolución.

Ninguno de las dos pérdidas de información son deseables. Este problema se suele solventar con la adición de ceros antes y después de los límites de la matriz de entrada (ver fig. 2.7), de forma que ambos problemas son solucionados, por un lado se asegura tener dimensiones compatibles entre las matrices y, además, extiende los límites para que el kernel pueda operar en su totalidad con todos los valores de la matriz de entrada. La adición de ceros no representa ninguna alteración a los datos de la matriz de entrada, debido a que al realizar el producto punto las operaciones generadas por el relleno no afectan el resultado.

Paso (Stride)

Un concepto adicional de las redes convolucionales es el **paso**. En lo mencionado anteriormente se describe un desplazamiento del kernel en el que solo se avanza un píxel antes de realizar la operación de convolución, sin embargo, esto también puede ser alterado para que dé dos o más saltos, aunque no se suele utilizar un número muy alto para el desplazamiento, ya que se pierde granularidad. La ventaja, por otro lado, es que al aumentar el tamaño del paso se puede reducir el número de operaciones y de valores resultantes para operar en la capa siguiente.

Reducción (Pooling)

Las capas de reducción (Pooling) como el caso de Max-pooling permiten “quitar resolución” a la imagen, y así, tener menos parámetros que entrenar tras cada capa, permitiendo redes de mayor profundidad, es decir, más capas sin aumentar excesivamente las necesidades de cómputo, lo que a su vez otorga una mayor capacidad de aprendizaje a la red.

El principio fundamental de este tipo de capas se conoce como sub muestreo (Sub-sampling), el cual toma solo algunos valores de un conjunto, por ejemplo, el Max-pooling toma submatrices y de estas extrae el mayor valor, de forma que luego de esta capa se tendrá una matriz de salida de tamaño reducido y con los máximos de cada sub matriz. Reduciendo enormemente los parámetros de entrada para la siguiente capa. Tal como se ve en el último paso de la imagen 2.6.

Entrenamiento de una red convolucional

El entrenamiento de aplicado a una red convolucional es esencialmente igual realizado en una red tradicional. Se hace uso del algoritmo Back propagation, aunque también puede ser vista como la

transpuesta de la convolución aplicada durante la etapa de predicción realizada con el algoritmo Forward propagation. Ambos, Forward Propagation y Back propagation son explicados en las secciones 2.2.4 y 2.2.5 respectivamente.

2.1.9. Redes recurrentes (RNN)

El análisis de imágenes puede ser hecho de forma integral ya que no es necesario tener otras imágenes para, por ejemplo, identificar objetos en ella, si bien es cierto que la existe una fuerte correlación entre los datos de la imagen (los píxeles), la información en su totalidad está contenida en la imagen, de esta forma un red neuronal puede recibir en un solo momento todos los datos necesarios para el análisis.

A diferencia del análisis integral de las imágenes, tanto en la naturaleza como en el día a día de las personas, existen situaciones en que una acción depende del análisis de una secuencia de datos que tienen una estrecha relación unos de otros pero que ocurren en momentos diferentes. En otras palabras que un único momento no permite el correcto análisis del sistema. Algunos ejemplos de estos tipos de secuencias de datos son:

- **Las secuencias de tiempo:** un ejemplo de este tipo de secuencias es el análisis del clima, donde un estado $n + 1$ está estrechamente relacionado con el estado n y éste a su vez depende del estado $n - 1$. Este puede ser el caso del reconocimiento del habla como los sistemas que convierten voz a texto.
- **Los textos:** Los textos son un ejemplo común de dependencia de estados, que una red neuronal tradicional no puede abordar correctamente. Esto se debe, no solo a la dependencia entre las palabras sino, además, porque no se conoce a priori la longitud de la frase a analizar. Cuando se procesa un texto se podría pasar toda la frase a la red neuronal, sin embargo, el problema aquí radica en que no es posible definir el tamaño de la entrada, podría crearse una red que reciba, por ejemplo, diez palabras pero si la frase es más larga que eso se estaría perdiendo información y, por otro lado, si la frase es más corta quedarían entradas sin asignar.
- **Información genética:** El ADN de los seres vivos en nuestro planeta está compuesto por solo cuatro bases nitrogenadas, sin embargo, la combinación, el orden y la posición de cada una permiten codificar todas las instrucciones de generación de proteínas en las células, por tanto, es claro que el análisis de un solo par de bases no es suficiente, e incluso el de un único gen.

Considerando lo anterior se hace evidente que es necesario contar con una arquitectura de red que permita recibir un número de parámetros variable y que además mantenga la correlación entre ellos a lo largo del tiempo.

Las redes neuronales recurrentes (RNN), son una arquitectura que contiene ciclos de forma que la salida de una neurona en un tiempo (Y_t) se use como entrada en un momento posterior (X_{t+1}), generando con esto bucles de retroalimentación. La figura 2.8 muestra el diseño general de este tipo de redes.

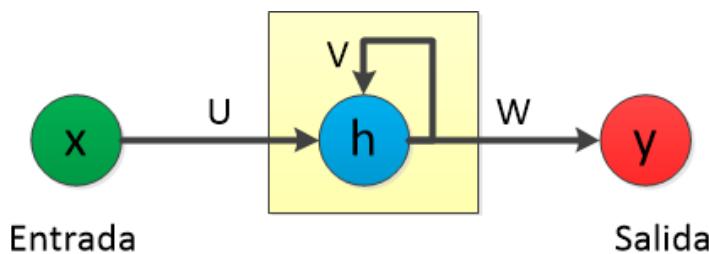


Figura 2.8: Diseño general de una RNN.

El concepto de redes recurrentes apareció en los años 80s con los trabajos John J. Hopfweld [Hopfield1982RNN], posteriormente Jeffrey Elman presentó un modelo más generalista [Elman1990RNN].

El principal impacto de las RNN consiste en que al almacenar información para utilizarla en momentos posteriores, a cuando se generó, con esto es posible decir que tiene memoria. Permitiendo con esto, el análisis de series de datos temporales.

Entrenamiento de una RNN

Una RNN puede ser descrita como se muestra en la ecuación 2.2:

$$y_t = f(Wx_t + Uy_{t-1} + b) \quad (2.2)$$

Siendo $x = [x_1, x_2, x_n]$ la secuencia de entrada proveniente de una capa anterior, W el peso aplicado a cada elemento de x . y_{t-1} corresponde a la salida obtenida en un tiempo anterior y U la matriz de pesos con los que se opera la salida del tiempo anterior. Finalmente, b corresponde al sesgo propio de la neurona.

En la etapa de entrenamiento de una RNN se debe actualizar los pesos tanto en W y b como en U . Para esto se hace uso del algoritmo de Propagación hacia atrás (en inglés Back propagation) (ver sección 2.2.5 para más detalle sobre Back propagation).

El método de Back propagation aplicado a las RNN cuenta con una variación por lo que se suele llamar propagación hacia atrás a través del tiempo (BPTT por sus siglas en inglés)[**Werbos1990BPTT**]. Esté método incluye el concepto de desenrollar la red. Entendiendo que la recurrencia de este tipo de redes, se puede considerar como una red tradicional envuelta sobre sí misma y, de igual forma la dependencia de una capa respecto a las anteriores, como se ve en la figura 2.9.

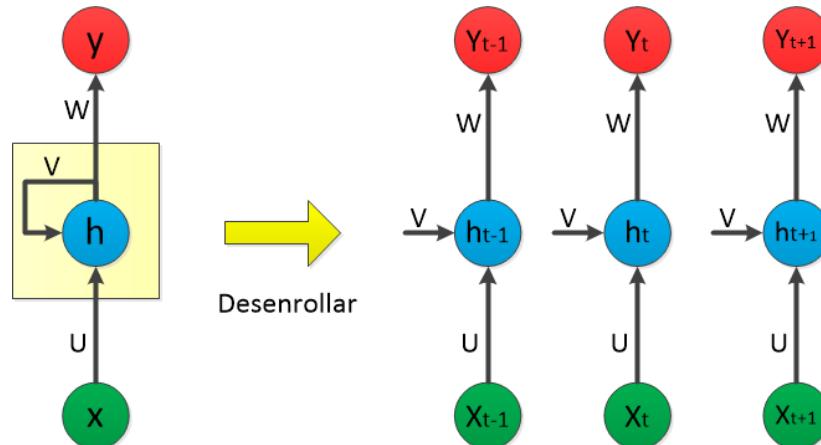


Figura 2.9: RNN desenrollada.

Así pues, al desenrollar la red, el entrenamiento puede realizarse de la misma forma que una red tradicional, tomando cada bucle de tiempo almacenado como una capa más.

Variación LSTM

Con lo mencionado en el entrenamiento de una RNN es entendible las implicaciones del desdoblamiento, en primer lugar, la gran cantidad de capas adicionales que se agregan, lo que además del proceso mismo de desenrollar la red aumentan los requerimientos computacionales, en consecuencia aumenta el tiempo de entrenamiento.

La segunda implicación, inherente a las redes y que se hace cada vez más notorio con el aumento de capas, es la explosión o el desvanecimiento del gradiente [**Hochreiter2001gradient**] (ver la sección 2.2.3 para más detalle de este concepto), impidiendo el cambio de los pesos de las conexiones entre las neuronas eficazmente.

Para solucionar el problema de desvanecimiento, en 1997 Sepp Hochreiter y Jürgen Schmidhuber presentaron un tipo de red recurrente llamada Long Short Term Memory (LSTM) [Hochreiter1997].

Las LSTM se apoyan de una estructura llamada Células de memoria. Las Células de memoria contienen una estructura que consta de tres puertas: *de olvido*, *de entrada* y *de salida*, que permiten gestionar lo que es información importante y se debe conservar y lo que debe ser 'olvidado', de esta forma mantiene durante largos períodos (momentos), fragmentos de información. El esquema general de las Células de memorias se muestra en la figura 2.10.

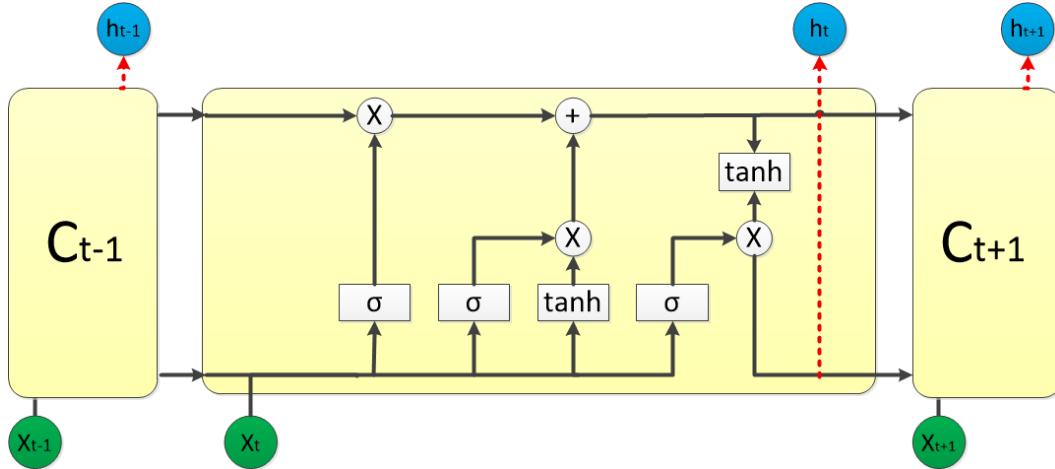


Figura 2.10: Esquema de un bloque de LSTM.

Puerta de olvido: Esta puerta se encarga de decidir que debe salir de la memoria debido a que no es 'relevante'. Esto es necesario para optimizar el rendimiento de la red. Para decidir que debe ser borrado se aplica la ecuación 2.3:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (2.3)$$

siendo f_t el resultado de aplicar la función Sigmoid a W_f los pesos para el conjunto de entrada x en el momento actual t y los valores del estado anterior h_{t-1} . Además, considerando el sesgo b para la información a olvidar. De esta forma, si f_t es cero deberá ser eliminada de la memoria.

Puerta de entrada: Esta puerta se encarga de decidir que información nueva debe ser recordada y se realiza contemplando dos acciones, en primer lugar se decide que información debe ser actualizada (ver ecuación 2.4) y cual debe ser incluida (ver ecuación 2.5)

$$i_t = \sigma(W_i \cdot [h_{t-1}] + b_i) \quad (2.4)$$

Siendo i_t la información que debe ser actualizada, W el vector de pesos de la información, b el sesgo propio de la información de entrada y h_{t-1} los valores para un estado anterior.

$$C_t = \tanh W_c \cdot [h_{t-1}, x_t] + b_c \quad (2.5)$$

Siendo C_t la información candidata a adicionar a la memoria, calculada por la aplicación de la función tanh a la multiplicación de la información candidata por sus pesos W más el sesgo b de la información candidata.

Puerta de salida: La puerta de salida se encarga de decidir cuando se entrega un resultado, para cada momento, debido a que no toda la información almacenada es necesaria en cada momento. Para decidir si una información es relevante en un determinado momento t , se aplica la ecuación 2.6.

$$o_t = \sigma(W_o \cdot [h_{t-1}] + b_o) \quad (2.6)$$

Donde o_t es la información relevante para el momento t , siendo W_o los pesos para la información de salida multiplicados por los valores del estado anterior h_{t-1} y b el sesgo propio de la información de salida.

Finalmente, se decide si un dato se generará como salida utilizando la ecuación 2.7.

$$h_t = o_t * \tanh c_t \quad (2.7)$$

2.1.10. Redes residuales (ResNet)

Si bien es cierto que el aumento de la profundidad (cantidad de capas) en las redes neuronales ha permitido, en las últimas décadas, alcanzar hitos que antes eran solo un deseo poco factible por muchos investigadores en inteligencia artificial, el aumento mismo de la profundidad ha traído consigo un problema inherente el desvanecimiento o fuga del gradiente (ver sección 2.2.3 para mayor detalle sobre este problema) lo que afecta el aprendizaje de la red.

Las redes residuales (ResNet) son un tipo de arquitectura que se enfoca en el problema del gradiente de fuga que ocurre en redes con muchas capas. En esta línea Kaiming He et al. [Kaiming2015] presentaron un trabajo que demuestra la efectividad, incluso, en redes de 152 capas, siendo además ganadores del ILSVRC 2015.

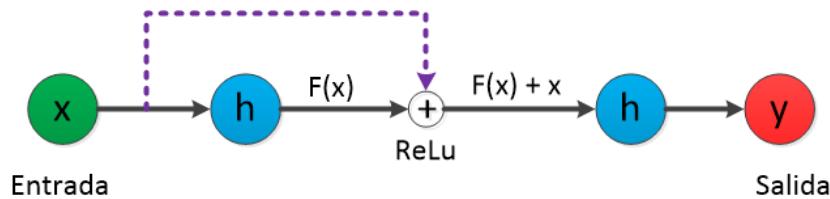


Figura 2.11: Diseño general de la arquitectura ResNet.

El principio fundamental de las ResNet consiste en agregar conexiones que saltan capas, como muestra la figura 2.11, de esta forma las activaciones de algunas capas siguen generando peso en capas posteriores. Aunque esta idea fue abordada anteriormente en el diseño de redes [Schraudolph1998], el trabajo presentado por Kaiming He et al. [Kaiming2015] ha sido uno de los más representativos; de este tipo de arquitectura, en los últimos años. Matemáticamente expresan su arquitectura con la Ecuación 2.8:

$$y = F(x, Wi) + x \quad (2.8)$$

Siendo x e y son los vectores de entrada y salida de las capas consideradas. La función $F(x, Wi)$ representa la asignación residual que se debe aprender. Es importante resaltar que el sesgo ha sido omitido por simplicidad de las anotaciones pero es considerado en su implementación, como en cualquier red tradicional.

Los experimentos en [Kaiming2015] demuestran que la omisión de capas acelera el aprendizaje al reducir el impacto de los gradientes que desaparecen, ya que hay menos capas para propagarse. Esto tiene cierta similitud con las conexiones del Cortex cerebral, donde las neuronas de la capa cortical VI reciben información de la capa I, omitiendo las capas intermedias.

2.1.11. Procesamiento de imágenes digitales

Una imagen digital puede definirse como una función $f(x, y)$ aplicada a una matriz de elementos (comúnmente conocidos como píxeles, o puntos elementales de una imagen), donde x e y corresponden con las coordenadas de cada elemento en la matriz. Además f corresponde a la intensidad en el punto (x, y) . El resultado de f es un valor discreto y perteneciente a un conjunto finito, por ejemplo un valor entero entre 0 y 255. En otras palabras, una imagen digital es una matriz de píxeles donde cada píxel corresponde con el 'color' en la coordenada en la que esté como se muestra en la figura 2.12.

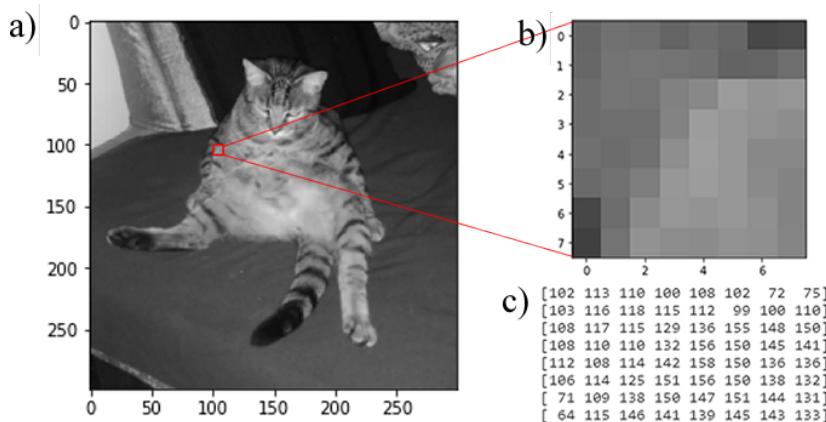


Figura 2.12: Imagen digital. a) Imagen en escala de grises de tamaño 300 x 300 píxeles. b) Fragmento de imagen de tamaño 8 x 8 píxeles. c) Matriz de valores del segmento de la imagen extraído, cada número corresponde al valor del píxel en escala de grises en un rango de 0 (negro) a 255 (blanco).

Esencialmente es posible agrupar las imágenes en tres tipos, dependiendo la cantidad de información que almacena cada píxel: Binarias, Escala de grises y Color.

- **Binarias:** Corresponden a imágenes donde el conjunto de valores posibles que puede tomar un píxel se limita a cero (0) o uno (1).
- **Escala de grises:** En este caso, el píxel cuenta con un rango más amplio de valores por ejemplo de 0 a 16, o 0 a 255. Este tipo de imágenes solo permite la representación de la imagen en diferentes todos de luz u oscuridad, yendo del valor más oscuro en cero al más claro en el máximo valor del rango usado.
- **Color:** Las imágenes de color almacenan más información pues cada píxel se corresponde, a diferencia de los anteriores, en una tupla que combina varias características de la luz o el color, siendo RGB el modelo más común.

RGB es un modelo aditivo de color, es decir, que el resultado se enmarca en la suma de tres colores rojo (R por su nombre en inglés), verde (G por su nombre en inglés) y azul (B por su nombre en inglés). Bajo el modelo de RGB, cada uno de las tres componentes del píxel pueden tomar un valor entero entre 0 y 255, variando desde el negro (0,0,0) al blanco (255,255,255).

El uso de una tupla en lugar de un valor escalar, aumenta enormemente la cantidad de variaciones posibles, permitiendo bajo el modelo RGB más de 16 millones de colores diferentes.

"El análisis de imagen digital está relacionado con la descripción y el reconocimiento del contenido de la imagen digital. Su entrada es una imagen digital y su salida es una descripción simbólica de la imagen. En muchos casos, las técnicas de análisis de imágenes digitales simulan la función de visión humana.". [Pitas2000] (Traducción propia).

El procesamiento de imágenes digitales es el conjunto de técnicas y algoritmos que busca interpretar la información que se encuentra contenida en imágenes digitales. Dentro de los objetivos más comunes están: el mejoramiento de la calidad de la imagen, transformación de la imagen, la identificación o localización de patrones u objetos y la clasificación de las imágenes.

El uso de técnicas de inteligencia artificial para el procesamiento de imágenes digitales, conocido como de forma general como visión por computador ha traído técnicas de procesamiento de imágenes como el uso de redes convolucionales (ver 2.1.8) y, sumado al aumento en las potencia de cómputo, esta área ha tenido un amplio crecimiento en las últimas dos décadas [Qin2020, Bora2020, Rojas2019heart], pues su aplicabilidad excede el ámbito netamente científico, siendo de importancia importancia para el sector industrial, la seguridad y la salud.

2.1.12. Procesamiento del lenguaje natural (NLP)

“El procesamiento del lenguaje natural es una técnica en la que la máquina puede volverse más humana y, por lo tanto, reduce la distancia entre el ser humano y la máquina. En sentido simple, la NLP hace que los humanos se comuniquen fácilmente con la máquina.” [Surabhi2013] (Traducción propia).

Entender el lenguaje natural de los humanos es un gran reto para la ciencia, debido a que no solo consiste en identificar las palabras por separado sino también el contexto. El procesamiento del lenguaje natural (NLP por sus siglas en inglés) abarca las áreas de lingüística, informática, ingeniería de la información e inteligencia artificial y busca, como principal objetivo, simplificar la interacción que tienen los humanos con las máquinas mediante el entendimiento del lenguaje que las personas usan de forma natural.

Además de la interacción Hombre-Máquina propiamente dicho, el NLP encuentra aplicación en muchos otros campos de uso cotidiano, por ejemplo: Identificación de correo Spam, La detección de emociones o sentimientos, la generación de textos o de resúmenes y la traducción de textos a diferentes idiomas manteniendo la coherencia del mismo.

Como se mencionó, en NLP no basta con identificar palabras por separado sino que debe comprender frases enteras, párrafos o textos de mayor longitud, dando un contexto global, de forma que se pueda obtener resultados coherentes en cualquiera que sea su aplicación. Por ejemplo, en el caso de una comunicación fluida deberá poder identificar instituciones, personas, acciones, lugares y la interacción entre todas ellas. Para lograr esto es necesario contemplar algunas tareas de pre-procesamiento, por ejemplo:

- **Normalización:** Existen ocasiones en que los textos pueden contener errores de digitación o bien que una misma cosa es nombrada de diferentes formas, por ejemplo: “Estados Unidos”, “USA”, “EEUU”, siendo todas formas de referirse al país norte americano. Al aplicar la normalización todas las apariciones de los textos puede ser reemplazadas por un inicio valor.
- **Tokenización:** Consiste en separar las frases en grupos de N palabras llamados N-Gramas. Cuando $N = 1$ hará una separación palabra a palabra, por ejemplo en la frase: “Sueñan los robots con ovejas mecánicas” se separaría en el vector $v=[\text{'Sueñan'}, \text{'los'}, \text{'robots'}, \text{'con'}, \text{'ovejas'}, \text{'mecánicas'}]$. Esto permite llevar los textos al campo numérico dando un valor a cada palabra.
- **Numeración de palabras:** Uno de los pasos finales de la preparación de los textos para ser analizados consiste en dar un valor a cada N-Grama, de forma que pueda ser utilizado como la entrada de una NN o cualquier otro método numérico.

Una vez preparados los textos, pueden aplicarse las técnicas que permiten ‘entender’ el lenguaje como tal, siendo el uso de RNN (ver 2.1.9) el enfoque ha dado mejores resultados en los últimos años,

dentro del NLP debido a su capacidad de procesar información distribuida en el tiempo, como es el lenguaje natural.

Actualmente el NLP sigue siendo un campo muy activo con usos prácticos como la generación de noticias [Grover2020], la identificación de eventos adversos[Young2019NLP], o la interacción con asistentes virtuales como Alexa de Amazon, el Google Home.

2.1.13. Sobre entrenamiento (Overfitting)

Dentro del campo del aprendizaje automático, el sobre entrenamiento hace se refiere a la situación en que los pesos (variables) del modelo generado durante el entrenamiento, están fuertemente ajustados a los datos usados para entrenar el modelo, de forma que, al ingresar los datos de prueba o cualquier dato no visto durante el entrenamiento, el sistema falla en sus predicciones. En otras palabras, cuando el entrenamiento da tasas de precisión muy altas pero en la etapa de pruebas da tasas de precisión son significativamente menores.

El sobre entrenamiento puede ser causado por dos factores principales:

- **Conjunto de entrenamiento pobre:** esto es cuando la muestra de datos seleccionados para realizar el entrenamiento consta de muy pocos ejemplos, cuando es excesivamente homogénea o ambos.

Es importante, al momento de definir la muestra de entrenamiento, que ésta contenga un numero amplio de ejemplo y que además contemple suficiente variación en ellos para contemplar la mayor cantidad de posibilidades o combinaciones de los datos, por ejemplo: si se busca construir un sistema que identifique números escritos a mano, puede no ser suficiente con tomas unos pocos ejemplos de una sola persona, porque en tal caso, el sistema aprenderá a identificar esos pocos ejemplos o solo los números escritos por esa persona, incluso podría fallar si la persona varía un poco la forma de escribir. Para mayor detalle en la construcción de conjuntos de datos vea la sección 2.4.

- **Modelos con un número alto de parámetros:** Uno de los factores que ha contribuido más al éxito de las redes neuronales actuales es su profundidad, sin embargo y contradictoriamente, es esto el generador de la poca precisión en algunos modelos para datos no vistos respecto a los usados en el entrenamiento. La razón de esto se debe a que un mayor número de parámetros permite que el modelo se ajuste de forma muy fuerte a los datos ingresados ya que cuenta con una mayor flexibilidad para ello, como se muestra en la figura 2.13, lo que lleva a menor precisión con datos nuevos.

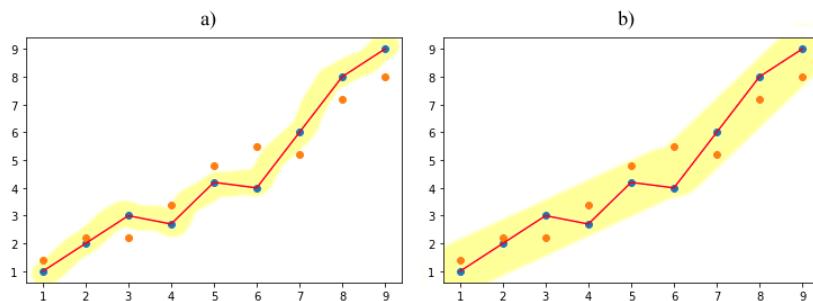


Figura 2.13: Sobre entrenamiento. a) Modelo con sobre entrenamiento por exceso de parámetros. b) Imagen sin sobre entrenamiento y pocos parámetros.

2.1.14. Transferencia de aprendizaje

La transferencia de aprendizaje se refiere al uso de modelos pre-entrenados para entrenar un modelo con objetivos diferentes al original, en lugar de partir con valores aleatorios en los parámetros del

nuevo modelo.

“En algunos dominios, como la bioinformática y la robótica, es muy difícil construir un conjunto de datos bien anotado a gran escala debido al costo de la adquisición y etiquetación de los datos, lo que limita su desarrollo.” [Tan2018TransferLearning] (Traducción propia).

La transferencia de aprendizaje reduce el problema de datos limitados, lo que podría conducir a sesgos (ver 2.4.8) o sobre entrenamiento (ver 2.1.13). La idea fundamental es tomar el aprendizaje (pesos) que tiene un modelo para ajustarlos a nuevos datos. Por ejemplo se podría utilizar un modelo entrenado para encontrar coches en imágenes y a partir del conocimiento que ya tiene, reentrenarlo para que encuentre aviones, pues parecer que las características entre ambos distan mucho, sin embargo, para que la IA pueda identificar coches debe pasar por algunas etapas en su procesamiento, como son, identificar bordes, encontrar que unos objetos están unidos a otros y finalmente lograr hacer una composición a gran escala del objeto. Aun cuando un avión tenga forma diferente a un coche, el hecho de que el modelo pueda entender características básicas de los objetos, permite que ese aprendizaje pueda ser usado y ajusta “rápidamente” para que aprenda nuevas cosas.

Un conveniente con el uso de esta estrategia consiste en que, si bien puede ayudar con conjuntos de datos (ver 2.4) de tamaño reducido, justamente si la diferencia en tamaño entre el conjuntos de datos original y el nuevo utilizado para el reentrenamiento es muy alta, podría pasar que nunca llegará a cambiar de forma significativa los pesos originales o al menos que el tiempo necesario sea excesivamente alto.

Por otro lado, es importante tener en cuenta que transferencia de aprendizaje no significa aprender nuevas cosas, por el contrario, lo que se consigue con esta estrategia es que el modelo empiece a “olvidar” su aprendizaje previo y éste se transforme en nuevo aprendizaje. Sin embargo es una estrategia a tener en cuenta, sobre todo para investigaciones nuevas donde la adquisición de los datos es muy costosa o de difícil consecución.

2.2. ALGORITMOS

2.2.1. Funciones de activación

A nivel biológico cada neurona presenta comportamiento muy interesantes, además de sus procesos metabólicos para mantenerse viva, es capaz de gestionar un potencial de voltaje mediante la apertura o cierre de ‘compuertas’ que permiten el paso de estímulos desde o hacia el exterior. Sin embargo, lo más interesante ocurre cuando cada neurona se conecta con otras, formando un circuito o red que permite, a partir de una información entrante generar una salida acorde. En otras palabras, Cuando las neuronas del cerebro reciben un estímulo electroquímico, se encargan de propagar la señal, a lo largo de la red que componen.

Un sistema estático y rígido no puede transmitir información, por lo que se hace necesario que haya diferencias de potencia que puedan cambiar en el tiempo y en el espacio, de forma que puedan transmitirse a lo largo de toda la red neuronal. “Estos cambios de potencial espacio temporales se denominan potenciales de acción y son las verdaderas unidades de información del sistema nervioso.” [Pastor2000Neurons]

Sin embargo, el comportamiento neuronal no consiste simplemente en retransmitir la señal recibida, sino que esta señal es transformada a causa del potencial de acción de la célula. La diferencia del potencial de acción puede causar excitación o inhibición [tamorri2004neurociencias], haciendo que la señal de entrada se procesada a lo largo de la red, permitiendo o no el paso del mensaje. Esto es muy importante ya que el potencial de acción determina la influencia que puede tener cada neurona en el sistema dependiendo del estímulo recibido, dotando al cerebro de una no linealidad.

En el contexto de las redes neuronales artificiales, una neurona debe calcular su ‘peso’ o influencia

sobre la red, tomando los valores de entrada y transmitirlo a las neuronas de la capa siguiente. Para asegurar que cada neurona se diferencia de las demás y, además, no se obtenga un valor 'directo' en la salida respecto a la entrada, se usan ecuaciones no lineales que transforman los valores de entrada, dando un mayor 'peso' o no a cada neurona, en forma semejante al potencial de acción. Estas ecuaciones se conocen como funciones de activación; entre las más usadas se encuentran la **Sigmoide**, la **Tangente hiperbólica** y la **ReLU**.

Sigmoide

La Sigmoide o sigmoidea también conocida como función logística es una función matemática no lineal, definida como se muestra en la ecuación 2.9, permite representar la evolución temporal de un sistema desde sus niveles más bajos de energía hasta su saturación, presentando una casi estabilidad hacia los extremos y una fuerte aceleración en el centro de los posibles valores de entrada como se muestra en la figura 2.14.

$$y = \frac{1}{1 + e^{-x}} \quad (2.9)$$

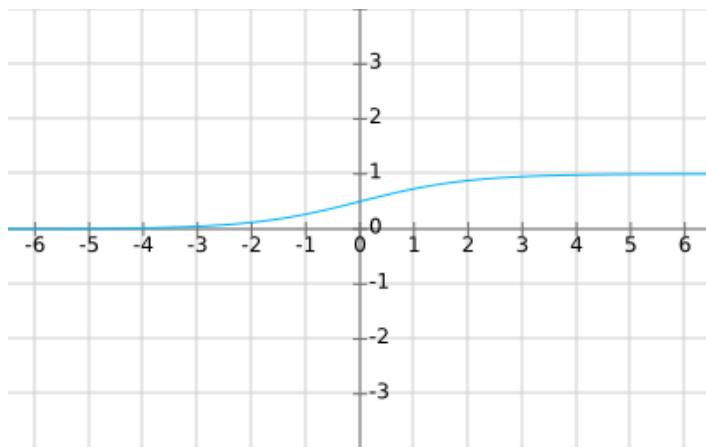


Figura 2.14: Representación ecuación Sigmoide.

Dentro de sus principales propiedades que la hacen útil y muy usada en el campo de las redes neuronales artificiales están:

- Tiene una derivada simple, lo que reduce las necesidades computacionales, como se ve en la ecuación 2.10.

$$y' = y(1 - y) \quad (2.10)$$

- Tiene dos asíntotas horizontales, por lo que la salida obtenida siempre va a ser un valor real entre cero y uno, como se ve en las ecuaciones 2.11 y 2.12. Donde la mitad de los valores de resultados posibles es negativa y la otra mitad positiva. Esto especialmente aprovechado por los clasificadores binarios.

$$\lim_{x \rightarrow -\infty} \frac{1}{1 + e^{-x}} = 0 \quad (2.11)$$

$$\lim_{x \rightarrow +\infty} \frac{1}{1 + e^{-x}} = 1 \quad (2.12)$$

- Su primera derivada es no negativa, como se muestra en la ecuación 2.13.

$$y' = \frac{dy}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} \quad (2.13)$$

- Además, a partir de un muy amplio rango de valores de entrada es posible obtener un número más limitado de salidas.

TanH

La Tangente hiperbólica es una función de activación usada en redes neuronales. Matemáticamente, se define como se ve en la ecuación 2.14. El comportamiento de esta función presenta características similares a la función Sigmoid como se muestra de la figura 2.15, sin embargo, la Tangente hiperbólica permite un rango de valores de resultado más amplio pues sus cotas se extienden de -1 a 1.

$$y = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.14)$$

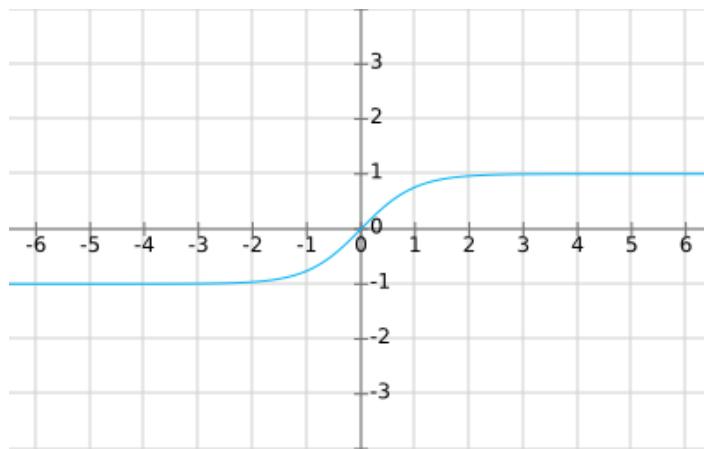


Figura 2.15: Representación ecuación Tangente hiperbólica.

Algunas propiedades que permiten su fácil uso dentro de las redes neuronales artificiales son:

- Derivada simple, se puede calcular a partir de la constante e , como se ve en la ecuación 2.15

$$y = \tanh(x) = \frac{1}{\cosh^2(x)} = \frac{1}{(e^x + e^{-x})^2} \quad (2.15)$$

- Tiene dos asíntotas horizontales, por lo que la salida obtenida siempre va a ser un valor real entre menos uno (-1) y uno (1), como se ve en las ecuaciones 2.16 y 2.17. Donde la mitad de los valores de resultados posibles es negativa y la otra mitad positiva. Esto especialmente aprovechado por los clasificadores binarios.

$$\lim_{x \rightarrow -\infty} \frac{1}{(e^x + e^{-x})^2} = -1 \quad (2.16)$$

$$\lim_{x \rightarrow +\infty} \frac{1}{(e^x + e^{-x})^2} = 1 \quad (2.17)$$

- Además, a partir de un muy amplio rango de valores de entrada es posible obtener un número más limitado de salidas.

ReLU

ReLU (rectified linear unit) unidad lineal rectificada, es un tipo de función de activación comúnmente usada en redes neuronales artificiales, matemáticamente se define como se ve en la ecuación 2.18. Dado que es una ecuación no lineal muy simple permite una implementación igualmente simple

en cualquier lenguaje de programación. La figura 2.16 muestra la representación gráfica de esta ecuación.

$$y = \max(0, x) \quad (2.18)$$

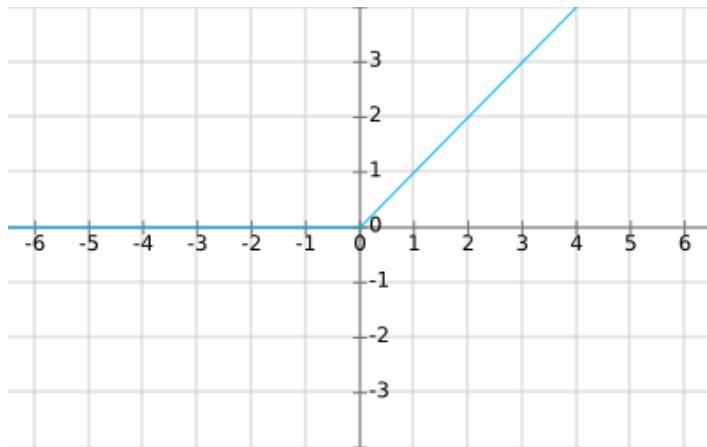


Figura 2.16: Representación ecuación ReLU.

La ecuación de ReLU equivale a decir que todos los valores negativos se reemplazaran por cero, en tanto que, los positivos mantendrán su valor original. Sus principales ventajas son: bajo coste computacional, por tanto ayuda a reducir el tiempo que puede llevar el entrenamiento, en comparación con otras funciones de activación. Converge más rápido. No tiene el problema de gradiente de fuga que sufren otras funciones de activación como sigmoide o tanh.

Por otro lado, para el calculo de la derivada de la función, requerida durante la etapa de entrenamiento de la red neuronal, basta con poner un condicional como se muestra en la ecuación 2.19.

$$y' = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{para mayores a cero.} \end{cases} \quad (2.19)$$

Puesto que ReLU es cero para todas las entradas negativas, es probable que alguna unidad no se active en absoluto lo que suele ser deseado en redes neuronales. Sin embargo, esta gran cualidad también puede ser su gran desventaja en algunos casos. Ya que puede llevar a la 'muerte' de las neuronas. Una neurona ReLU está 'muerta' si está atrapada en el lado negativo y siempre emite 0. Debido a que la pendiente de ReLU en el rango negativo también es 0, una vez que una neurona se vuelve negativa, es poco probable que se recupere. En tal caso, las neuronas dejan de tener un rol en la discriminación de la entrada y son esencialmente inútiles.

2.2.2. Funciones de pérdida

La razón de entrenar modelos en inteligencia artificial es encontrar valores para los parámetros del modelo que representen más acertadamente el peso o impacto de cada uno dentro de un conjunto de datos de entrenamiento, de forma que las predicciones estén cada vez más cercanas al valor esperado en la salida del sistema. En otras palabras reducir el error.

Dicho lo anterior, y antes de poder reducir el error que tiene un modelo de inteligencia artificial es necesario calcular dicho error, para esto posible definir una función de pérdida $l(x)$ que mida que tan malos o lejanos son los valores obtenidos en una predicción de un modelo de inteligencia artificial.

Existen varias funciones de pérdida como el error cuadrático medio o entropía cruzada, cada una con características particulares y, por tanto, su uso o selección puede depender de diversos factores como el tipo de algoritmo seleccionado o incluso la naturaleza de los datos de entrenamiento.

Error Absoluto Medio (MAE) / Pérdida L1

Una forma simple de estimar el error entre los valores obtenidos y los esperados, consiste simplemente en la suma de los valores absolutos de la diferencia entre los valores obtenidos y los esperados. La ecuación 2.20 muestra la expresión matemática de este método.

$$MAE = \frac{\sum_{i=0}^n |y_i - y'_i|}{n} \quad (2.20)$$

Siendo y el vector de valores esperados durante el entrenamiento y y' el vector equivalente obtenido.

Error cuadrático medio (MSE) / Pérdida L2

El método de error cuadrático medio permite estimar el error entre los valores obtenidos y los esperados, a partir de la suma de los valores de la diferencia entre los valores obtenidos y los esperados elevada al cuadrado. Esto conlleva una penalización más marcada entre errores grandes en comparación con errores pequeños. La ecuación 2.21 muestra la expresión matemática de este método.

$$MSE = \frac{\sum_{i=0}^n (y_i - y'_i)^2}{n} \quad (2.21)$$

Siendo y el vector de valores esperados durante el entrenamiento y y' el vector equivalente obtenido.

Entropía cruzada

El método de entropía cruzada aplicado a problemas de optimización combinatoria fue explorado por Rubinstein [Rubinstein1997CrossEntropy], aplicándolo inicialmente a redes estocásticas complejas y posteriormente en la estimación de la probabilidad de eventos raros. Se basa en el principio de máxima entropía propuesto por Jaynes [Jaynes1957InformationTheory] y cuya exactitud fue demostrada por Shore y Johnson [Shore1980].

La entropía cruzada puede utilizarse para medir la precisión de los valores pronosticados (el error), matemática (ver ecuación 2.22) y conceptualmente se relaciona con la estimación por máxima verosimilitud. En otras palabras la suma negativa de la multiplicación de los valores del vector esperado por los logaritmos de los valores del vector de resultado.

$$CEL = - \sum_{i=0}^n y_i \log(y'_i) \quad (2.22)$$

Siendo y el vector de valores esperados durante el entrenamiento y y' el vector equivalente obtenido.

Pérdida de bisagra / Pérdida SVM

La función de pérdida de bisagra (hinge loss en inglés), es usada en la clasificación de “margen máximo” en especial dentro de SVM. Permite saber el grado de certeza de la predicción a partir de los mismos valores predichos. La ecuación 2.23 muestra la expresión matemática de este método.

$$\sum_{j \neq i}^n \max(0, y_j - y_i + 1) \quad (2.23)$$

Siendo y_j el valor obtenido en una de las posiciones del vector de resultados y y_i el valor obtenido en cada una de las posiciones del vector de resultados diferente de y_j .

Error de sesgo Medio (MBE)

Aunque este método es quizá el menos común, en el campo de aprendizaje automático ya que su precisión es inferior a la de otros métodos, permite determinar si el modelo tiene un sesgo positivo o negativo. La ecuación 2.24 muestra la expresión matemática de este método.

$$MBE = \frac{\sum_{i=0}^n (y_i - y'_i)}{n} \quad (2.24)$$

Siendo y el vector de valores esperados durante el entrenamiento y y' el vector equivalente obtenido.

2.2.3. Descenso de gradiente

Reducir el tamaño del error entre lo que predice un sistema de inteligencia artificial y el valor esperado es quizá la premisa más importante durante la etapa de entrenamiento. Aquí es importante entender qué es el tamaño del error y cómo se mide ese error. De forma general, el error en la predicción es la diferencia entre el valor esperado y el predicho por el sistema IA (vea la sección 2.2.2 para profundizar en el concepto de funciones de pérdida). Entré más pequeño sea el tamaño del error, significa un mejor ajuste del sistema a los datos de entrenamiento (Ver sección 2.4 Referencia a creación de dataset para más detalle sobre conjuntos de datos), es decir una mejor caracterización de los datos. Es importante mencionar que un sobre ajuste en el entrenamiento puede ser problemático durante la ejecución con conjuntos de datos diferentes a los de entrenamiento, pero este tema es tratado en la sección 2.1.13.

Durante el entrenamiento de una red neuronal artificial se utiliza el algoritmo de propagación hacia atrás o Back Propagation en inglés (El algoritmo de Back propagation se explica en detalle en la sección 2.2.5), este algoritmo debe calcular de forma iterativa los valores para cada neurona que minimicen el error general. Sin embargo, encontrar el valor mínimo para algunas funciones puede ser sencillo, como el caso presentado en la figura 2.17, pero no todas las funciones tienen una forma tan clara para identificarlo, como el caso de la figura 2.18.

El mínimo de una función puede ser calculado de forma directa tomando su derivada e igualándola a cero. Sin embargo, eso solo es posible si la función tiene forma cerrada, lo cual no siempre es posible con las funciones de estimación de error en redes neuronales.

Un segundo método consiste en ubicarse en cualquier punto de la función y avanzar de forma iterativa, buscando en cada paso estar en un punto inferior tratando de descender al punto más bajo. Para entender mejor considérese el siguiente ejercicio mental: Si una persona se encuentra en un punto no definido de una montaña, con los ojos vendados y debe encontrar la ruta que lo lleve a la parte baja de la montaña sin ver. En forma general la persona podrá moverse en cuatro direcciones (o 360 si desea comprobar la inclinación del terreno circundante cada grado de giro), cada vez que

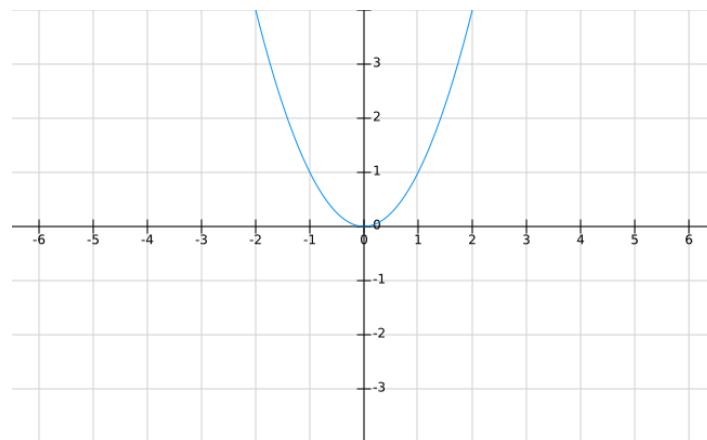


Figura 2.17: Gráfica de la función $f(x) = x^2$.

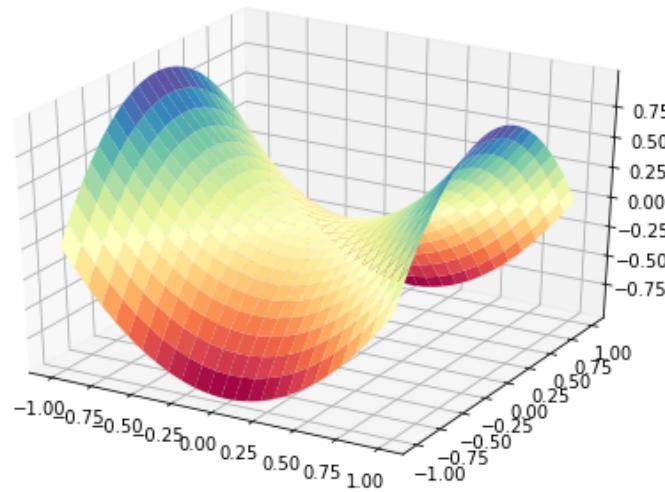


Figura 2.18: Gráfica de la función $z = x^2 - y^2$.

la persona identifique la dirección que le lleva más abajo respecto a las otras dará un paso, y este proceso lo repetirá de forma indefinida hasta que encuentre punto en el que todas las direcciones estén al mismo nivel. Este método es conocido como descenso de la colina o hill climbing.

El principal inconveniente en la implementación del ejercicio mental expuesto, es que se debe realizar los cálculos para las cuatro direcciones antes de poder definir la dirección a avanzar, una forma de reducir esos costos computacionales consiste en el uso del gradiente. El gradiente es la generalización vectorial de la derivada, es decir, un vector con 'N' dimensiones y sobre el que se calcula para cada dimensión su derivada parcial, como muestra en la ecuación 2.25, obteniendo de forma general la pendiente en un determinado punto.

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right] \quad (2.25)$$

Siendo ∇f_x el vector que contiene la información de cuanto crece la función en un punto específico x por cada dimensión de la función de forma independiente.

A partir de la pendiente obtenida con uso del gradiente, es posible saber si un punto en una función sube o baja y, usando este mismo razonamiento, bastará con ir en contra de la pendiente en cada paso para encontrar cada vez un punto menor. Es justo aquí donde entra en juego el uso del algoritmo del descenso de gradiente en redes neuronales artificiales.

El descenso de gradiente fue originalmente propuesto por Cauchy en 1847 [Cauchy1847], es un algoritmo de optimización iterativa, que se encarga de buscar el mínimo local yendo en contra de la pendiente dando pequeños pasos, usando información de la primera derivada y se ha convertido en uno de los pilares del aprendizaje automático, ya que durante la etapa de entrenamiento, permite reducir el valor del error entre la salida obtenida por la red y el valor esperado sin un excesivo costo computacional.

Ahora bien, ir en contra de la derivada para definir la dirección de avance es solo una parte del razonamiento del descenso del gradiente, otro punto importantes es cuál debería ser la longitud del paso, es decir, qué tanto avanzar en cada paso para no pasar de largo y tener que regresar o inclusive nunca llegar. Para esto se agrega un parámetro α adicional llamado tasa de aprendizaje o learning rate (ver sección 2.2.3, para mayor detalle sobre tasa de aprendizaje), como se muestra en la ecuación 2.26.

$$-\alpha \nabla f \quad (2.26)$$

Siendo α un valor pequeño, por ejemplo 0.3 y ∇f el vector del gradiente. Es importante considerar que α puede cambiar tras cada paso, y pueden existir varias criterios para ello, como incrementar su valor si el gradiente es grande y reducirlo en caso contrario, o simplemente ir reduciéndolo paulatinamente a medida que se realizan más iteraciones.

Así pues, teniendo la dirección y el tamaño del paso a dar es posible calcular, dada una posición x la siguiente posición \hat{x} .

Finalmente, es necesario identificar si el algoritmo ha llegado a su fin, es decir, si se llegó al mínimo de la función. Una forma de hacerlo es calcular la diferencia entre x y \hat{x} , si es menor que un umbral se puede dar por finalizado. También se puede simplemente realizar un número específico de iteraciones.

Una consideración especial, sin embargo, es que este tipo de métodos iterativos, no siempre permiten llegar de forma inexorable a un mínimo global de la función, sino más bien a mínimos locales, estos son puntos mínimos respecto a los valores a su alrededor como se muestra en la gráfica 2.19. Aunque también cabe destacar que en el área del aprendizaje automático existen estrategias para intentar sortear los mínimos locales, como puede ser realizar varios entrenamientos con valores iniciales aleatorios e ir variando la tasa de aprendizaje.

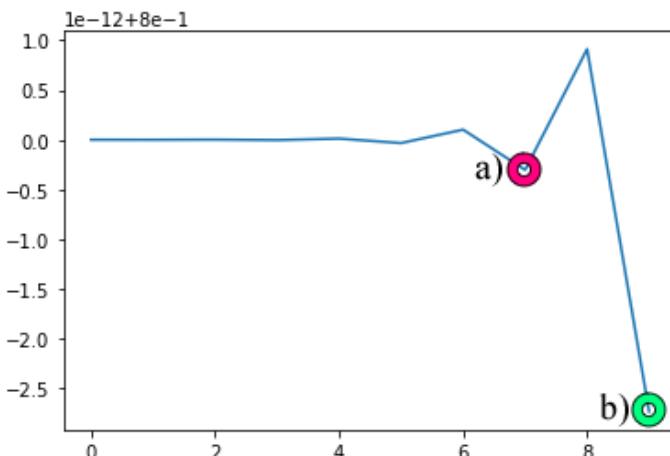


Figura 2.19: Mínimos locales. el punto a) corresponde a un mínimo local en la función, en tanto que b) corresponde al valor mínimo real de toda la función.

Tasa de aprendizaje

El concepto de tasa de aprendizaje consiste en un hiper parámetro de las redes neuronales u otros modelos de aprendizaje supervisado, que permite definir que tan rápido aprende, en otros términos, que tan grande puede ser el cambio que se aplica a las variables de un modelo que se pretende ajustar.

Las principales consideraciones de este parámetro radican en que si se asigna un valor muy pequeño, así mismo serán las actualizaciones a las variables del modelo, por lo que se requerirá de muchas iteraciones para que los cambios sean perceptibles, es decir aprenderá muy lentamente.

Por otro lado, si la tasa de aprendizaje es muy alta, esto podría causar que las actualizaciones a las variables del modelo estén saltando de un lado al otro rodeando el error mínimo haciendo que, incluso, nunca se converja en una solución óptima.

Ambos casos, una tasa muy alta o muy baja pueden llevar al problema de desvanecimiento o a la explosión del gradiente (ver sección 2.2.3 para mayor detalle).

Explosión y desvanecimiento del gradiente

De forma general el descenso del gradiente permite calcular el cambio que debe hacerse en las variables de un modelo (pesos), realizando esto iterativamente este cambio (gradiente) va impactando en todo el modelo. Sin embargo, esta misma iteración o llamado recursivo trae con sigo un problema y es llevar el gradiente a un extremo superior muy alto o a un extremo inferior muy pequeño, esto se conoce como explosión y el desvanecimiento del gradiente.

La explosión y el desvanecimiento del gradiente (en inglés Exploding Gradients y Vanishing Gradients respectivamente) son un problema que afecta en especial a las redes muy profundas (con muchas capas) como el caso de las redes neuronales recurrentes (ver 2.1.9).

La explosión del gradiente ocurre cuando en el modelo se genera una importancia exageradamente alta a los pesos, sin razón aparente, con lo que las entradas pierden importancia para el modelo. Aunque es un problema que no se puede predecir a priori es fácil de resolver con el truncamiento de los gradientes, evitando que sobrepasen un determinado umbral.

Por otro lado, el desvanecimiento del gradiente o gradiente desvanecido ocurre cuando los cambios del gradiente son tan pequeños que el modelo deja sencillamente de aprender. Aunque es un problema mucho más difícil de resolver se han desarrollado mecanismos como de LSTM (ver sección 2.10) para el caso de las redes recurrentes o el uso de Redes Residuales para otro tipo de redes.

La naturaleza de este problema está, como se mencionó, en la repetición de la operación y la constitución misma de la regla de la cadena usada por el algoritmo de propagación hacia atrás (ver sección 2.2.5). Supóngase una red de t capas, cuando t es un número grande, la operación iterativa aplicada a una función de activación f para obtener el gradiente, es decir la deriva de dicha función, sería como se muestra en la ecuación 2.27.

$$f'_i(x) = \prod_{i=t-1}^1 f_i(x) * f'_{i+1}(x) \quad (2.27)$$

Es evidente que si al tener multiplicaciones sucesivas si se cae en un número grande éste se irá incrementando iterativamente y por otro lado los números pequeños se irán reduciendo sucesivamente. De aquí se identifica la importancia de la selección de la función de aprendizaje más adecuada para cada problema.

2.2.4. Propagación hacia adelante (Forward propagation)

En el campo de las redes neuronales artificiales se les suele mencionar como un modelo inspirado en las las redes neuronales biológicas y, aunque dista mucho el entendimiento de las segundas permiten inferir el comportamiento esperado de las primeras.

Una neurona biológica en forma general la componen tres partes: las dendritas, el cuerpo y los axones. Por medio de la sinapsis, la neurona recibe en las dendritas los mensajes de otras neuronas que la preceden, posteriormente en el cuerpo realiza la “suma” de todas las señales de excitación o inhibición, y si la señal de excitación es mayor transmite un un mensaje en forma de diferencia de potencial, a otras neuronas, usando su axón.

En el campo de las redes neuronales artificiales es posible caracterizar a las neuronas por dos funciones: la función base (ver ecuación 2.28) y la función de activación (ver 2.2.1). Además, cada neurona está conectada a las neuronas de otras capas y asociado a cada conexión existen valores que determinan el efecto que tiene la neurona sobre las siguientes, esto se conoce como peso sináptico y se denomina como w_{ij} , siendo w el efecto que la neurona i sobre la neurona j estando i y j en capas sucesivas.

$$u_i(w, x) = b + \sum_{j=0}^n w_{ij} * x_j \quad (2.28)$$

El algoritmo de propagación hacia adelante (Forward propagation en inglés) se utiliza para replicar el comportamiento de cada neurona biológica a lo largo de toda la red, esto es, ir transmitiendo las señales de las capas anteriores a las siguientes. Para esto el algoritmo toma las entradas a la neurona les aplica la función base (ecuación 2.28), posteriormente, el resultado de la función base es pasado a la función de activación.

De esta forma, los valores de entrada $x[...]$, sus pesos w y el sesgo propio de la neurona b , generan una salida $y[...]$ que se convertirán en la entrada de las neuronas de la siguiente capa, repitiendo este proceso desde la primera hasta ultima capa de la red, como se muestra en la figura 2.20.

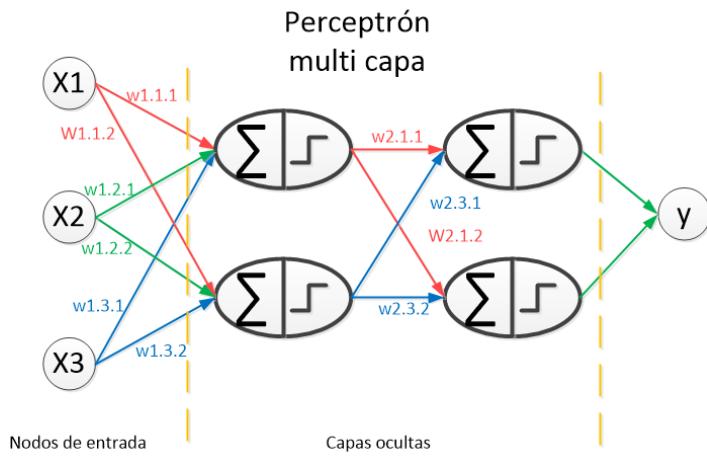


Figura 2.20: Modelo del Perceptrón multi capa.

El resultado de la capa final será la inferencia o predicción que es capaz de realizar un modelo para un conjunto de datos de entrada. Para que los resultados sean los esperados los pesos sinápticos w y los sesgos b deberán ser actualizados durante la etapa de entrenamiento y para ello se hace uso de los algoritmos de descenso de gradiente(ver sección 2.2.3) y de propagación hacia atrás (ver sección 2.2.5).

2.2.5. Propagación hacia atrás (Back propagation)

Dentro del campo aprendizaje automático se busca establecer las relaciones entre un conjunto de valores de entrada $x = [x_1, x_2, x_n]$ y un conjunto de salida $y = [y_1, y_2, y_m]$, en otras palabras que a partir de una entrada x se pueda tener una salida conocida y . Para lograr esto se define un modelo que contemple las n dimensiones de entrada de cada neurona para cada elemento de y , por ejemplo, es posible definir el modelo de la forma: $y = wx + b$.

Dado que x es un valor invariable es posible ajustar los valores de w y de b de forma que para cualquier valor x de entrada se aproxime a los valores de salida y correspondientes. El valor que debe ser ajustado se logra haciendo uso del algoritmo del descenso del gradiente (ver sección 2.2.3), sin embargo es necesario que el ajuste de los parámetros del modelo se ajuste en cada neurona de la red.

El algoritmo de propagación hacia atrás o retropropagación (Back propagation en inglés) fue propuesto por Paul Werbos [[Werbos1975](#)] en 1975 y representó un hito en el avance del desarrollo de las redes neuronales, pues fue el primero que permitió un entrenamiento eficaz, logrando que éstas realmente 'aprendieran' basados en la experiencia.

La propagación hacia atrás es un método iterativo y recursivo que permite ir actualizando los 'pesos' de las neuronas, apoyándose en el descenso de gradiente y explota la conocida regla de la cadena.

El proceso general de algoritmo de propagación hacia atrás consiste en aplicar a todas las neuronas de la última capa (capa de salida) el algoritmo del descenso de gradiente, para identificar el cambio que debe hacerse en los pesos (w y b). Una vez estimado el error en la última capa, dicho valor será multiplicado por el resultado de la aplicación de la derivada parcial a los pesos de las neuronas conectadas en la capa inmediatamente anterior.

El proceso de ajustar los pesos, se realiza desde la última capa hasta llegar a la primera y se repetirá durante miles o millones de veces (dependiendo del problema) durante la etapa de entrenamiento, hasta que los pesos w y b de todas las neuronas y sus conexiones permitan que dada una entrada x se obtenga una salida y esperada, es decir, aprenda.

2.2.6. Función de clasificación Softmax

[Softmax](#)

2.3. ARQUITECTURA DE SOFTWARE

La arquitectura de software es la forma en que se describen los componentes de un sistema informático, las estructuras, los procesos que realizan y la manera en que interactúan, antes de la construcción del sistema. Por tanto es una tarea de diseño, desde el más alto nivel.

Peter Freeman en [[Freeman1980](#)], define el diseño como “*una actividad que tiene que ver con la toma de decisiones importantes, con frecuencia de naturaleza estructural. Comparte con la programación el objetivo de abstraer una representación de la información y de las secuencias de procesamiento, pero en los extremos el grado de detalle es muy distinto*”.

Por su parte Presman en [[Pressman2010](#)], define la arquitectura como “*Cuando se piensa en la arquitectura de una construcción, llegan a la mente muchos atributos distintos. En el nivel más sencillo, se considera la forma general de la estructura física. Pero, en realidad, la arquitectura es mucho más que eso. Es la manera en la que los distintos componentes del edificio se integran para formar un todo cohesivo. Es la forma en la que la construcción se adapta a su ambiente y se integra a los demás edificios en la vecindad. Es el grado en el que el edificio cumple con su propósito y en el que satisface las necesidades del propietario.*”. considerando además que es una tarea de diseño agrega: “*La arquitectura no es el*

software operativo. Es una representación que permite 1) analizar la efectividad del diseño para cumplir los requerimientos establecidos, 2) considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y 3) reducir los riesgos asociados con la construcción del software.”

La importancia de esta actividad radica entonces, en que en ella se plasman todas las decisiones importantes del sistema y como se mencionó la forma en que interactúan todas las partes involucradas en el mismo. De esta forma, y haciendo un símil con la construcción de edificios, de una correcta arquitectura puede depender, en gran medida, la estabilidad del sistema y, para el caso de sistemas informáticos la futura ampliación evitando la reconstrucción de partes importantes.

Debido a su importancia, esta tarea suele ser ejecutada por personas experimentadas en el campo del desarrollo de software, usualmente conocidos como ‘arquitectos de software’. Esta persona será la responsable de definir tanto las estructuras como mecanismos para la implementación de los diferentes artefactos del software. Para ello cuenta con la definición de soluciones generalizadas a problemas habituales, esto será explicado más adelante en la sección de patrones de desarrollo de software.

2.3.1. Patrón de desarrollo de software

El concepto de padrón se atribuye al arquitecto Christopher Alexander, quien en su publicación de 1977 [Alexander1977]: “*Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe el núcleo de la solución a ese problema, de tal manera que puede usar esta solución un millón de veces, sin hacerlo dos veces de la misma manera.*”

Prontamente, los especialistas en software se dieron cuenta que las ideas de Alexander, aun cuando estaban establecidas y orientadas hacia la construcción de edificios o similares, podrían ser extrapolables al desarrollo de software. Así Gamma et al. en su publicación de 1994 [Gamma1994], sentaron las bases en la definición de los patrones que permiten generalizar la mayoría de los problemas a los que se enfrenta un diseñador de software y brinda una orientación sobre como debería ser la correcta implementación.

2.3.2. Instancia única (Patrón)

“*Asegúrese de que una clase solo tenga una instancia y proporcione un punto de acceso global a ella.*” [Gamma1994]. (Traducción propia).

Para algunos sistemas es de suma importancia que algunas clases solo tengan una única instancia, por ejemplo, para compartir información o recursos entre varios objetos o para que las modificaciones hechas por un objeto sea accesible de forma directa por otros. Un ejemplo de esto pueden ser el acceso a repositorios de datos como bases de datos o pilas de datos.

Adicionalmente, debido a instancia única se puede aplicar un acceso controlado sobre la forma en que puede ser accedida y los requerimientos de acceso.

2.3.3. Método factoría (Patrón)

“*Define una interfaz para crear un objeto, pero permita que las subclases decidan qué clase instanciar* ‘Método factoría’ *permite que una clase difiera la creación de instancias en subclases.*” [Gamma1994]. (Traducción propia).

“*Los métodos de fábrica eliminan la necesidad de vincular clases específicas de la aplicación en el código. El código solo se ocupa de la interfaz de la clase; por lo tanto, puede funcionar con cualquier Clase definida por el usuario.*” además, “... *El Método de Fábrica le da a las subclases un gancho para proporcionar una versión extendida de un objeto.*” [Gamma1994]. (Traducción propia).

Con lo anterior es posible crear objetos de forma independiente para que las subclases puedan sobre escribir los métodos de la clase que heredan si fuera necesario.

2.3.4. Iterador (Patrón)

“Proporcione una forma de acceder a los elementos de un objeto agregado secuencialmente sin exponer su representación subyacente.” [Gamma1994]. (Traducción propia).

Gracias a este patrón es posible acceder a los elementos de un objeto agregado, como una lista, sin exponer su estructura interna.

2.3.5. Comando (Patrón)

“Encapsula una solicitud como un objeto, lo que te permite parametrizar clientes con diferentes solicitudes, solicitudes de cola o registro, y admite operaciones que no se pueden deshacer.” [Gamma1994]. (Traducción propia).

“A veces es necesario emitir solicitudes a objetos sin saber nada sobre la operación que se solicita o el receptor de la solicitud. Por ejemplo, los kits de herramientas de la interfaz de usuario incluyen objetos como botones y menús que realizan una solicitud en respuesta a la entrada del usuario. Pero el juego de herramientas no puede implementar la solicitud explícitamente en el botón o menú, porque solo las aplicaciones que usan el juego de herramientas saben qué se debe hacer en cada objeto. Los diseñadores de kits de herramientas, no tienen forma de conocer al receptor de la solicitud o las operaciones que la llevarán a cabo.” [Gamma1994]. (Traducción propia).

Esto permite poner en cola y ejecutar solicitudes en diferentes momentos, lo que es además es útil para la interacción con pilas o con servicios que provean funcionalidades que no necesariamente impacten de forma directa en el cliente, ya que su naturaleza desacopla el objeto que hace la solicitud y la ejecución de la misma.

La naturaleza desacoplable brinda, además de lo mencionado, la flexibilidad de adicionar métodos o comandos, sin alterar el cliente y la forma en que éste interactúa, dando al sistema una capacidad mayor de escalabilidad.

2.3.6. Composición (Patrón)

Consiste en la “*Composición de objetos en estructuras de árbol para representar jerarquías tanto de sus partes como de estructura completa. La composición permite a los clientes tratar objetos individuales y sus composiciones de objetos de manera uniforme.*” [Gamma1994]. (Traducción propia).

“El patrón compuesto define jerarquías de clase que consisten en objetos primitivos y objetos compuestos. Los objetos primitivos se pueden componer en objetos más complejos, que a su vez se pueden componer de otros, y así sucesivamente de forma recursiva.” [Gamma1994]. (Traducción propia).

2.3.7. Método plantilla (Patrón)

“Consiste en definir el esqueleto de un algoritmo en una operación, aplazando algunos pasos a subclases. El Método de plantilla permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar la estructura del algoritmo.” [Gamma1994]. (Traducción propia).

Una de sus funciones es “*implementar las partes invariantes de un algoritmo una vez y dejarlo en manos de subclases para implementar el comportamiento que puede variar.*” [Gamma1994]. (Traducción propia).

“El método plantilla es una técnica fundamental para la reutilización de código. Tiene una gran importancia en las librerías de clases, porque son los medios para la generalización del comportamiento

común en las clases de la biblioteca.” [Gamma1994]. (Traducción propia).

Este patrón suele combinarse con el patrón de método de factoría, pues la generalización que brinda una plantilla facilita la creación de objetos por parte de la factoría.

2.3.8. Pila de datos (Pool)

“Mejorar el rendimiento y el uso de la memoria reutilizando objetos de un grupo fijo en lugar de asignarlos y liberarlos individualmente.” [Nystrom2014]. (Traducción propia).

En ciencias de la computación, una pila es un conjunto de objetos con características en común. El uso de pilas de datos u objetos es especialmente útil cuando se necesita el manejo y reuso de grandes cantidades de objetos, de esta forma, cuando cliente solicita un recurso a la pila este es entregado, sin embargo, cuando el cliente termina de usarlo, el recurso es mantenido en lugar de liberarse y perderse, lo cual mejora la eficiencia del sistema.

A nivel de programación una pila es representado como un arreglo o una lista de un tipo de dato.

Las permanencia de los datos en la lista dependerá del problema a tratar y, de igual forma la forma en que se almacenan o como se accede a cada elemento. Siendo posible almacenar y entregar los datos en la forma en que llegan, en forma inversa, o basado en características inherentes a los datos como el estado o un cualquier otro atributo.

2.3.9. Hilo

“Un hilo es un flujo de control independiente dentro de un proceso, compuesto de un contexto (que incluye un conjunto de registros) y una secuencia de instrucciones para ejecutar. Por flujo de control independiente, nos referimos a una ruta de ejecución a través del programa.” [Wadleigh2000]. (Traducción propia).

Un hilo o subprocesso en e campo de la computación es un el paquete de instrucciones procesamiento más pequeña que se puede realizar en un sistema operativo. Es el modelo de programación que permite que los procesos (o subprocessos) se ejecuten simultáneamente. Solo programa con capacidades de subprocessamiento múltiple permite que subprocessos individuales (o hilos) se ejecuten aparentemente al mismo tiempo.

Una de las principales ventajas es que si un proceso se divide en múltiples subprocessos, y cada función de subprocesso se considera como un trabajo, entonces aumenta el número de trabajos completados por unidad de tiempo, lo que aumenta el rendimiento del sistema.

2.3.10. Servicio REST

En el campo de la arquitectura de software, el concepto de servicio hace referencia a una o varias piezas de software que permiten el acceso a diferentes recursos a través de una interfaz claramente establecida.

REST (REpresentational State Transfer) es un modelo arquitectónico para desarrollar servicios web, descrito por Roy Fielding en el año 2000 [Fielding2000]. Los servicios REST se basa en el estándar de transferencia de hipertexto (HTTP) y hace uso de los métodos que lo componen. En los últimos años ha ganado gran popularidad en el campo del desarrollo de sistemas de información, debido a su simplicidad y velocidad.

“Un servicio REST se define como una agregación de diferentes recursos a los que se puede acceder desde un URI base. Un recurso representa una entidad del mundo real cuyo estado está expuesto y se puede cambiar accediendo a un URI (identificador universal de recursos).” [Pautasso2008]. (Traducción propia).

2.4. CONJUNTOS DE DATOS

El conjunto de datos ('Data set' en inglés) es una agrupación de elementos con características similares de los cuales se desea que una Inteligencia Artificial reconozca las características intrínsecas a los elementos para, posteriormente, poder aplicar las inferencias aprendidas por el sistema a información nueva, sea esta, imágenes, textos, acciones o cualquier otro.

Dentro del campo del aprendizaje automático, la creación de un conjunto de datos es en sí misma una área de estudio, debido a la importancia fundamental de ellos. Sin datos no es posible hacer que un sistema "aprenda". A causa de esto, en cualquier proyecto de Inteligencia Artificial una de las actividades que demandan una gran cantidad de esfuerzo y tiempo es la preparación de un conjunto de datos.

Cuando se prepara un conjunto de datos para entrenar una red neuronal, es necesario contar con una cantidad suficientemente amplia para evitar o mitigar el sobre entrenamiento (ver 2.1.13). Sin embargo, el tamaño no es la única consideración, la calidad de los datos es fundamental para contar con un buen Data set.

En muchas ocasiones para lograr la calidad de los datos es necesario realizar algunas etapas de preparación antes de que los sean aceptables para un entrenamiento. Considérese la figura 2.21 para establecer las etapas necesarias para la constitución de un Conjunto de datos para un sistema de aprendizaje automático.

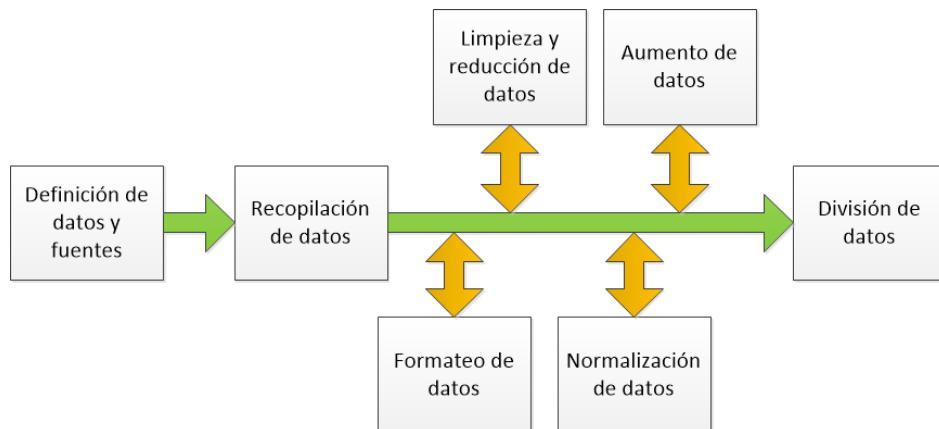


Figura 2.21: Pasos para la creación de conjuntos de datos.

Es importante mencionar que no todas las etapas son obligatorias, éstas dependerán de la calidad, la fuente y el formato originales de los datos.

2.4.1. Definición de los datos y fuentes necesarias

El primer paso en la preparación de un Data set consiste en definir qué se desea que el sistema aprenda, por ejemplo: identificar identificar evento anómalos en un entorno definido. A partir de esto es necesario establecer que posibles fuentes de información se tienen, por ejemplo: Imágenes, secuencias de vídeo, acelerómetros, o cualquier otro mecanismo del cual se puedan obtener los datos.

Junto con esto se identifica la estructura que tiene los datos en la fuente, por ejemplo: una matriz de píxeles, un flujo de números en el tiempo o cualquier otro, identificando completamente qué dato contiene cada variable obtenida de la fuente.

2.4.2. Recopilación de datos

La calidad y cantidad de los datos de entrenamiento puede marcar el éxito del entrenamiento. La recopilación de datos consiste en acumular datos de la mayor cantidad de fuentes posibles que

cumplan con la definición hecha anteriormente. Una consideración fundamental en aprendizaje automático es contar con una cantidad amplia de datos. Por eso, el proceso de recopilación de datos debe ser hecho de forma que se tenga un conjunto amplio, por ejemplo, para el análisis de imágenes usando redes convolucionales se suelen considerar decenas de miles, cientos de miles o incluso millones de imágenes. De igual forma, si la información a utilizar como conjunto de entrenamiento depende del tiempo, es necesario contar con ventanas de tiempo suficientemente amplias que permita tener una muestra amplia.

2.4.3. Formateo de datos

Este paso busca hacer que todos los datos que recibirá el sistema para su entrenamiento sean consistentes en su estructura y en su valor. Por ejemplo, si una de las fuentes entrega un valor en dólares (USD\$1) y otra en euros (EUR\$1.00) es necesario definir un único formato y realizar la conversión a la moneda definida. O si se usan imágenes a escala de grises de un tamaño de 300x300 píxeles, es necesario llevarlas todas, sin importar la fuente, a escala de grises y al tamaño definido.

Otro tipo de acción puede ser la **descomposición de los datos**, en caso de que la fuente genere datos complejos, éstos pueden ser descompuestos en partes más simples, por ejemplo: un valor de año mes día, puede ser separado en tres variables diferentes.

2.4.4. Limpieza y reducción de datos

En este paso se busca asegurar que toda la información a utilizar tiene aporte significativo durante el entrenamiento, para así, no aumentar excesivamente los recursos computacionales necesarios.

Según lo mencionado, es beneficioso tener un conjunto amplio de datos, sin embargo, es necesario que asegurar que los datos tiene los atributos necesarios según lo definido, por ejemplo, si se busca identificar la postura de una persona en una imagen, puede ser de utilidad la eliminación de fotos donde haya más de una persona, pero también puede ser considerada la eliminación del fondo.

En este paso, también se busca completar los datos, hasta donde sea posible, siguiendo con el caso de identificación de la postura de una persona en una imagen, puede ocurrir que no esté completa o al menos no tenga todos los puntos necesarios, en este caso en lugar de simplemente eliminarla podría ser aceptable completarla, esto es, proyectar extremidades ocluidas o cortadas de la foto usando las proporciones del cuerpo.

2.4.5. Normalización de datos

Un proceso normalización sobre los datos ayudan a asegurar la calidad de la información de forma que se reduzca el impacto negativo que pueden generar los valores anormales en el entrenamiento, un ejemplo de normalización de datos se demuestra en la ecuación 2.29.

$$x_i' = \frac{x_i - \min(x)}{\max(x) - \min(x)} \quad (2.29)$$

Donde \min representa el valor mínimo del conjunto x y \max el mayor valor. De esta forma cada valor es ponderado respecto a su conjunto de pertenencia.

Otra forma de normalización es 'discretizar' los datos. En algunos casos puede ser más eficiente trabajar con categorías discretas en lugar de valores continuos, por ejemplo, un sistema de recomendaciones basada en la edad podría tomar rangos de edades en lugar del valor de edad concreta.

2.4.6. Aumento de datos

La estrategia de aumento de datos es habitualmente utilizada cuando se dispone de muy pocos ejemplos para un entrenamiento [Rojas2019heart], ya que ayuda a reducir el sobre entrenamiento.

“La mejor manera de hacer que un modelo de aprendizaje automático se generalice mejor es entregarlo en más datos. Por supuesto, en la práctica, la cantidad de datos que tenemos es limitada. Una forma de solucionar este problema es crear datos falsos y agregarlos al conjunto de entrenamiento.” [Goodfellow2016]. (Traducción propia).

El aumento de datos se logra aplicando diferentes transformaciones a los datos originales [Fawzi2016DataAugmented], por ejemplo, cuando se trata de imágenes es posible, a partir de una imagen original generar otras aplicando: rotaciones, cambio de escala, cambio de tonos, trasladados entre otras. Gracias a esto se consigue una mejor generalización de los datos en la etapa de entrenamiento ya que las transformaciones no alteran la información intrínseca de los datos, pero aumentan la muestra, por ejemplo una imagen en espejo de un gato seguirá conteniendo un gato tanto en la original como en la transformada pero permitirá al sistema tener otro punto de vista del mismo objeto.

2.4.7. División del conjunto de datos

Para saber que tan bueno es un sistema de aprendizaje automático no basta con recopilar un conjunto amplio de datos, realizar el entrenamiento con todos ellos y estimar la precisión, de hecho esto es una mala práctica debido a que no permite identificar un ‘sobre entrenamiento’ (ver 2.1.13) o sesgos que se podrán evidenciar posteriormente en la puesta en producción del sistema.

La mejor opción para asegurar una tasa de precisión extrapolable a entornos fuera del conjunto de datos de entrenamiento; para sistemas de aprendizaje automático, es dividir el conjunto en tres partes: Entrenamiento, Validación y Pruebas. La figura 2.22 muestra dos momentos de división de los datos, en primer lugar una separación de los datos usados durante el tiempo de **entrenamiento** de los **datos de prueba**. Posteriormente una división adicional donde una parte de los datos de entrenamiento son tomados para **validación** y hacer los ajustes al modelo y verificaciones durante el proceso de entrenamiento.

- **Subconjunto de entrenamiento:** Corresponde a los datos que serán usados para el entrenamiento del modelo.
De este subconjunto dependerá lo que aprenda el sistema.
- **Subconjunto de validación:** Corresponde a los datos que serán usados durante la etapa de entrenamiento para refinar tanto el diseño del sistema como los parámetros.
Este subconjunto ayudará a reducir el sobre entrenamiento del modelo durante la etapa de entrenamiento.
- **Subconjunto de pruebas:** Corresponde a los datos que serán usados para comprobar la precisión del modelo fuera de la etapa de entrenamiento, es decir, el que permitirá identificar que tan bueno es el modelo fuera del aprendizaje.
La consideración más importante de este subconjunto es que los datos de éste nunca deben ser ‘vistos’ por el sistema durante la construcción y entrenamiento del sistema.

Aunque no existe un consenso sobre qué porcentaje debe ser asignado a cada parte, se suele dejar la mayor parte de los datos para el entrenamiento como tal, y es frecuente usar una distribución de 70-15-15 [Valipour2016, Hemmat2015, Turabieh2019, Rojas2019heart]: 70 % para entrenamiento, 15 % para validación y 15 % para pruebas.

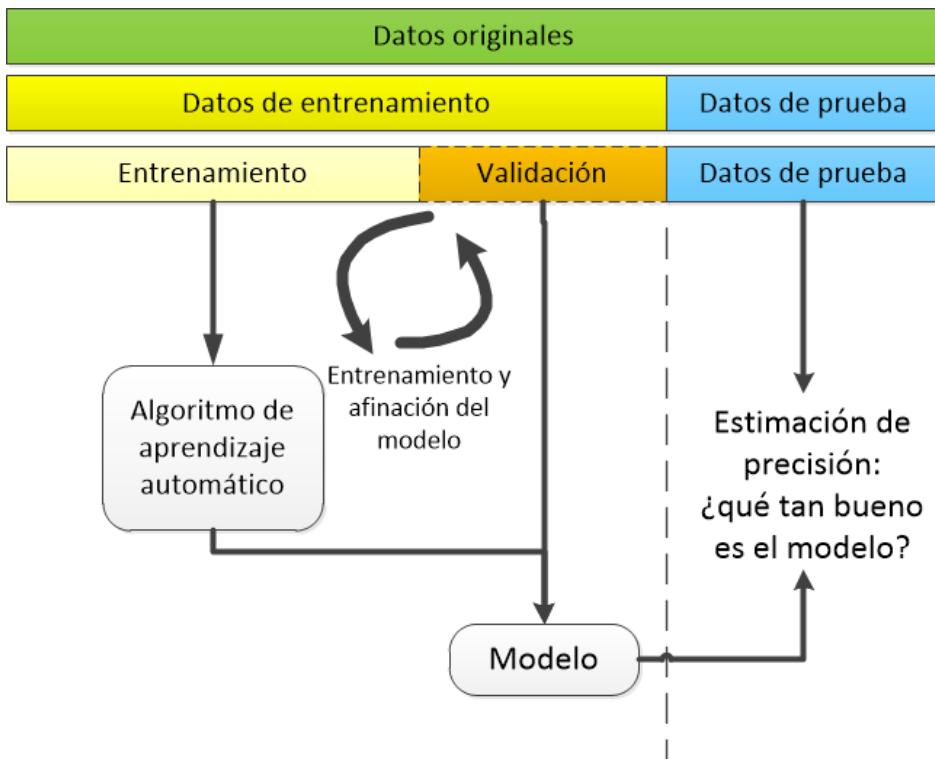


Figura 2.22: División del conjunto de datos.

2.4.8. Sesgo de datos

El contar con un gran número de datos no es suficiente para decir que se cuenta con un buen conjunto de datos para el entrenamiento, pues dentro de éstos puede existir un problema oculto conocido como “Sesgo de datos”.

“Ningún sistema de inteligencia artificial tiene intencionalidad, pero las decisiones que aprende están basadas en los datos con los cuales ha sido entrenado, y si esos datos están sesgados (intencionadamente o no), el algoritmo decidirá sesgado, y ese sesgo puede tener consecuencias muy drásticas que afecten a la vida de las personas”, Ramón López de Mántaras (Director del Instituto de Investigación en Inteligencia Artificial del Consejo Superior de Investigaciones Científicas (CSIC)) [Vanguardia2017Sesgo].

El sesgo en los datos es una posibilidad de cualquier conjunto de datos, en mayor o menor medida, que se da en muchos casos de forma no intencional, por una limitada muestra, pero también puede ser dada por la fuente misma de los datos. El sesgo hace referencia al hecho de que los datos pueden contener características sobre los cuales se puede dar más importancia durante el entrenamiento de una Inteligencia Artificial y de las que el propio creador de la IA puede no tener conocimiento.

El sesgo en los datos es un problema que impacta a los sistemas de aprendizaje automático debido a la dependencia de los datos para el entrenamiento. Debido a esto, en los últimos años ha aparecido una creciente preocupación por el impacto de la Inteligencia Artificial en la vida diaria, no por el potencial mismo de esta tecnología, sino por los sesgos que este tipo de sistemas puedan tener [Buolamwini2018Sesgo]. Sesgos como: decisiones injustificadamente influenciadas por la raza o el género de la persona.

Los sesgos se pueden presentar en cualquier conjunto de datos, sin importar su formato u origen, por ejemplo, se puede presentar cuando se construye un Data set para identificar rostros humanos en el cual la mayor cantidad de fotos corresponde a personas caucásicas de género masculino y una escasa muestra de mujeres de origen africano. Este tipo de situaciones no necesariamente es intencional y puede deberse a la facilidad de conseguir los datos, en este caso imágenes, con las

condiciones necesarias para el entrenamiento (fondo, iluminación, etc) de personas de regiones diferentes a la del investigador. También es habitual que cada pueblo haga herramientas para sus propias necesidades, así pues consciente o inconscientemente los creadores de este tipo de soluciones basadas en Inteligencia Artificial las construyen para sus contextos cercanos y, considerando que la mayor parte de investigaciones y de empresas que trabajan en el desarrollo de estas tecnologías se encuentran en el norte de América, Europa y Asia, es entendible que inicialmente sus propios Data set estén sesgados a las características dominantes de su población.

Otro sesgo que encontrado en los datos, que si bien al mirar hacia atrás pudo ser previsible, solo se notó su impacto bajo la mirada moderna de equidad, fue el presentado por Bolukbasi et al. [Bolukbasi2016Sesgo] cuando se usa al modelo Word2Vec construido por investigadores de la empresa Google. Word2Vec es un modelo entrenado usando un corpus muy extenso de textos para inferir, entre otras cosas, la cercanía contextual entre palabras, de forma que la relación entre gato y animal es mayor que entre gato y televisor [Mikolov2013Word2Vec]. Bolukbasi et al utilizaron Word2Vec para construir un sistema de analogías de forma que el sistema completara la palabra faltante, lo que mostraron Bolukbasi et al. es un sesgo de género con el ejemplo: "Hombre es a programador como mujer es a X" donde la "X" fue reemplazada por ".ma de casa" [Bolukbasi2016Sesgo].

Para cualquier investigador es evidente que el sesgo existente en Word2Vec no es causado intencionalmente por los investigadores de Google, sino que es un sesgo existente en la misma sociedad generadora de los textos, y como se mencionó anteriormente es entendible el origen de estos sesgos, sin embargo, el ser entendible no significa que sea aceptable, pues aunque los sesgos como decidir enviar a prisión a una persona, por variables diferentes a su comportamiento, que unos sistemas funcionen mejor en función de su raza o que el género defina las actividades de una persona pueden no ser socialmente aceptables.

Aunque los sesgos mostrados pueden ser los de mayor impacto social en la actualidad, el sesgo de los datos afecta de forma negativa a cualquier sistema de aprendizaje automático, puede un sistema de IA entrenado para que reconozca una determinada enfermedad entrenarla con, por ejemplo, imágenes de mujeres con dicha enfermedad, llegar a inferir que los hombres no la padecen. O si se entrena a un asistente virtual para reconocer comandos de voz con un conjunto de datos constituido solo por archivos de audio de personas de España, podría tener una tasa de error muy alto cuando lo use una persona de origen americano, aun cuando se use el mismo idioma, pues la IA puede desarrollar un sesgo por la forma de pronunciar las palabras.

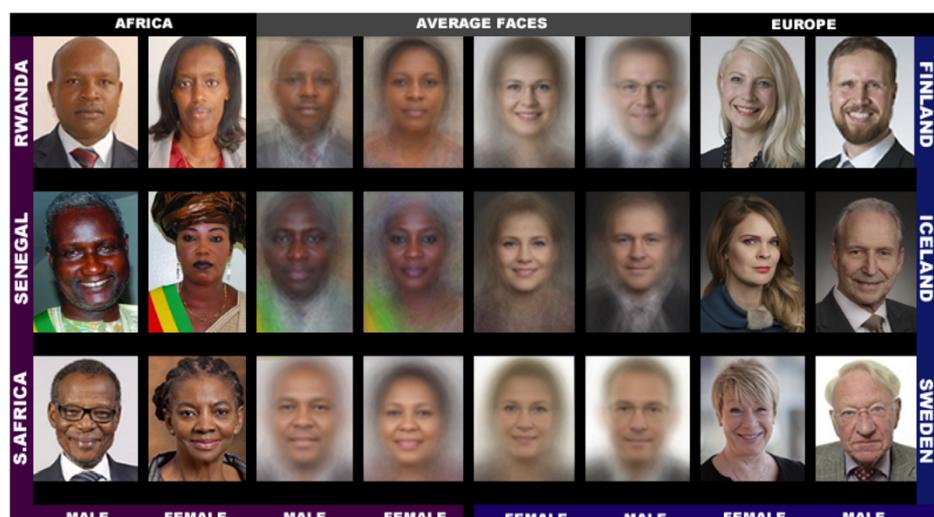


Figura 2.23: Conjunto de datos sin sesgo. Imagen tomada de [Buolamwini2018Sesgo] donde se presenta una distribución bien balanceada entre rostros de personas de diferentes colores de piel y género.

Para reducir el impacto del sesgo de datos es necesario no solo un conjunto de datos más amplio sino también más heterogéneo en su muestra, como el caso del proyecto <http://gendershades.org/> liderado por Buolamwini y Gebru [**Buolamwini2018Sesgo**] donde muestran un conjunto más balanceado entre razas, colores de piel y géneros (ver figura 2.23), de forma que el sistema entrenado no tenga ninguna inclinación especial e injustificada por ninguna de las características fenotípicas y su aprendizaje esté mejor orientado a las características que se desean en la identificación de rostros.

2.4.9. ImageNet (Data set)

El data set ImageNet es un conjunto compuesto por más de 14 millones de imágenes de alta resolución publicado y compartido en el año 2009 [**ImageNet2009**], creado por Deng et al.

“ImageNet es un conjunto de datos de imágenes organizado de acuerdo con la jerarquía de WordNet. Cada concepto significativo en WordNet, posiblemente descrito por múltiples palabras o frases de palabras, se denomina conjunto de sinónimos o synsets. Hay más de 100,000 synsets en WordNet, la mayoría de ellos son sustantivos (80,000+). En ImageNet, nuestro objetivo es proporcionar un promedio de 1000 imágenes para ilustrar cada synset. Las imágenes de cada concepto tienen control de calidad y anotación humana.” [**Imagenet2020Home**] (Traducción propia).

El conjunto Imagenet más que un repositorio de imágenes provee un listado de URLs de Internet donde se localizan las imágenes. Adicionalmente, para cada imagen se etiqueta diferentes objetos en ellas, hasta un total de 20 mil en todo el conjunto, donde se pueden encontrar etiquetas para: Aviones, gatos, espejos de coches, entre muchos otros.

Competencia para el reconocimiento visual

Adicional al data set y en apoyado en éste se creó el “ImageNet Large Scale Visual Recognition Challenge [ILSVRC]” [**Imagenet-ILSVRC15**], una competición para impulsar el desarrollo de las tecnologías en el campo de la identificación de objetos. Gracias a esto se han desarrollado modelos de identificación de objetos como Inception V3” [**InceptionV32016**], creado y compartido por la empresa Google.

2.4.10. COCO (Data set)

Objetos comunes en contexto (En inglés Common Objects in Context - COCO) es un data set con una gran cantidad de imágenes (más de 300 mil) [**COCO2014**] que cuenta con información detallada de los objetos comunes en ellas, de forma que puede ser usado para el entrenamiento de modelos para detección, segmentación o etiquetado de objetos. Los objetos fueron etiquetados utilizando segmentaciones por instancia para ayudar en la localización precisa de objetos. Dentro de sus características principales se encuentran:

- Más de 200 mil imágenes etiquetadas.
- 2.5 millones de instancias de objetos segmentados en todo el conjunto.
- 80 tipos de categorías de objetos.
- 250 mil imágenes de personas con puntos clave (articulaciones principales, cabeza, entre otras que permiten formar un esqueleto) marcadas.

Competencia COCO para generación de esqueletos de personas en imágenes

A partir de las 250 mil imágenes de personas y los 1.7 millones de puntos clave, se creó una competencia (“COCO Keypoints Challenge” [**Coco2020KeyPointsHome**]) para impulsar el desarrollo de investigaciones de métodos que permitan la localización de personas y la generación de un esqueleto caracterice la postura de las personas. Como se muestra en la figura 2.24, este data set contiene puntos etiquetados para las articulaciones principales, el cuello y el centro de la cabeza en imágenes una o más personas en situaciones y posturas cotidianas.



Figura 2.24: Ejemplo de imágenes del COCO keypoint Challenge [Coco2020KeyPointsHome].

2.4.11. MPII Human Pose (Data set)

Para impulsar la investigación en la estimación de la postura humana en imágenes en el año 2014, Andriluka et al. presentaron el conjunto de datos “MPII Human Pose” [MPII2014].

Este conjunto de datos está compuesto por más de 25 mil imágenes y dentro del conjunto más de 40 mil personas con las articulaciones principales etiquetadas. El conjunto también tiene etiquetas para partes del cuerpo ocultas y la orientación de la cabeza.

El data set fue construido a partir de videos de Youtube, considerando varios frames, los cuales cuentan también con un marcado de orden de secuencia de cada uno. El conjunto de datos agrupa un total de 410 actividades diferentes, organizadas como se muestra en la figura 2.25.

Activity Categories	Activities	Images
bicycling	animal care, household animals, general (0) - 10	
conditioning exercise	breastfeeding, sitting or reclining (34) - 364	
dancing	building a fire inside (0) - 216	
fishing and hunting	butchering animals, small (0) - 165	
home activities	carrying groceries upstairs (4) - 368	
home repair	child care (104) - 870	
inactivity quiet/light	cleaning windows, washing windows, general (52) - 465	
lawn and garden	cleaning, general (103) - 989	
miscellaneous	cleaning, sweeping floor or carpet (37) - 863	
music playing	cooking or food preparation (236) - 865	
occupation	cutting and smoking fish, drying fish or meat (41) - 637	
religious activities	dusting or polishing furniture, general (29) - 222	
running	elder care (35) - 872	
self care	feeding household animals (16) - 198	
sports	food shopping with or without a grocery cart, sta... (47) - 199	
transportation	ironing (53)	
volunteer activities	kitchen activity, general (e.g., cooking, washin... (63) - 73	
walking	knitting, sewing (38) - 866	
water activities	laundry (61) - 867	
winter activities	making bed, changing linens (77) - 756	
	maple syrup/sugar bushing (including carrying b... (22)	
	mopping, standing, light effort (72) - 341	
	moving furniture, household items, carrying boxes (70) - 31	
	moving household items upstairs, carrying boxes o... (31)	
	moving, lifting light loads (0) - 170	
	non-food shopping, with or without a cart, standi... (53) - 46	
	organizing room (20) - 244	
	playing with animals (129) - 871	
	playing with children (97) - 869	
	polishing floors, standing, walking slowly, using... (44) - 34	
	putting away groceries (e.g. carrying groceries, ... (13) - 4	
	scrubbing floors (42) - 868	

Figura 2.25: MPII Human Pose [MPII2014].

2.4.12. HumanEva (Data set)

HumanEva es un data set de imágenes de actividades humanas hechas en laboratorio [Sigal2010HumanEva, HumanEva2020Home], es decir, fueron capturadas en un ambiente controlado (objetos e iluminación). Para la captura se utilizaron siete cámaras sincronizadas, cuatro de escala de grises y tres de color, de forma que el conjunto de datos contiene la información 3d de las personas filmadas.

El conjunto de datos se realizó con la participación de cuatro actores mientras realizaban diferentes actividades comunes (caminar, trotar, lanzar y atrapar una pelota, boxeo y algunas combinaciones). En total 40 mil fotogramas sincronizados.

2.4.13. FLIC (Data set)

FLIC es un conjunto de imágenes (con alrededor de cinco mil imágenes), generado a partir de 30 películas populares de cine de Hollywood [FLIC2013]. El data set tiene diez etiquetas para las articulaciones superiores de las personas en las imágenes.

2.4.14. DMLSmartActions (Data set)

DMLSmartActions DMLSmartActions dataset: Dataset for human actions in smart environments, (Jan. 2019), University of British Columbia, [Online]. Available: <http://dml.ece.ubc.ca/data/smartaCTION/>.

UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild

SBU Kinect Interaction Dataset SBU Kinect Interaction Dataset is created by Yun and Honorio et al. [38] using Microsoft Kinect for two-person interactions. There are 8 categories including 7 participants and 21 groups of interactive actions. The dataset contains 282 skeletal sequences and 6822 frames and 15 joints. <https://ieeexplore.ieee.org/document/6239234>

2.5. SOFTWARE Y LIBRERÍAS

2.5.1. Python

“Python es un lenguaje de programación interpretado, orientado a objetos y de alto nivel con semántica dinámica...”, “...Python admite módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización de código. El intérprete de Python y la extensa biblioteca estándar están disponibles en formato fuente o binario sin cargo para todas las plataformas principales, y se pueden distribuir libremente.” [Python2019].

Debido a flexibilidad, además de propiciar la facilidad en la legibilidad en el código, Python se ha convertido en el principal lenguaje utilizado en el desarrollo de librerías de índole científico en general y de inteligencia artificial en particular.

2.5.2. JSON

“JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos. Es fácil para los humanos leer y escribir. Es fácil para las máquinas analizar y generar. Se basa en un subconjunto del lenguaje de programación JavaScript Standard ECMA-262 3rd Edition - December 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son familiares para los programadores de la familia de lenguajes C, incluido C, C++, C#, Java, JavaScript, Perl, Python y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.” [Json2019].

2.5.3. Yaml

Yaml es un lenguaje de serialización de datos basado en Unicode, pensado para que sea de fácil interpretación por humanos. Está diseñado para representar los principales tipos de datos de los lenguajes de programación ágiles. debido a su naturaleza como archivo plano puede ser usado por cualquier lenguaje de programación y presta especial utilidad para el envío de mensajes por Internet o para archivos de configuración [Yaml2019].

2.5.4. Tensorflow

“TensorFlow es una plataforma de código abierto de extremo a extremo para el aprendizaje de máquina (ML). Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la

comunidad que permite a los investigadores impulsar el estado del arte en ML y los desarrolladores pueden construir y desplegar fácilmente aplicaciones potenciadas por ML.” [GoogleTF2019].

TensorFlow fue desarrollado originalmente por el equipo de Google Brain Team, dentro del departamento de investigación de Machine Intelligence, con el propósito de llevar a cabo el aprendizaje automático y la investigación de redes neuronales profundas. Desde su lanzamiento como software de código abierto, a finales del 2015, se ha convertido en una de las principales herramientas en la construcción de sistemas de inteligencia artificial.

2.5.5. Keras

“Keras es una API de redes neuronales de alto nivel, escrita en Python y capaz de ejecutarse sobre TensorFlow, CNTK o Theano. Fue desarrollado con un enfoque en permitir la experimentación rápida. Poder pasar de la idea al resultado con el menor retraso posible es clave para hacer una buena investigación.”

Keras permite la creación fácil y rápida de redes neuronales, incluyendo convolucionales y redes recurrentes, así como combinaciones de las dos y, además, su ejecución en GPU. Aunque Keras no se encarga de las funciones matemáticas involucradas en el entrenamiento y aprendizaje de las redes neuronales, su popularidad radica en que expone una capa que simplifica el uso de las librerías matemáticas como Tensorflow.

El desarrollo de Keras está respaldado principalmente por Google, y la API de Keras viene empaquetada en TensorFlow como tf.keras. Además, Microsoft mantiene el backend CNTK Keras. Amazon AWS mantiene la bifurcación Keras con soporte MXNet. Otras compañías contribuyentes incluyen NVIDIA, Uber y Apple (con CoreML).

2.5.6. Flask

Flask es una librería escrita en lenguaje Python, que permite de forma rápida la creación y exposición de servicios REST [Flask2019].

2.5.7. Logging

Una de las tareas más habituales en cualquier proyecto de software es el registro de los eventos que ocurren durante su ejecución. Normalmente se registran mensajes de errores o excepciones, pero también pueden ser almacenados mensajes como inicio de servicios u otros.

Dentro de la funcionalidades incluidas en el ambiente Python, existe la librería ‘Loggin’. “Este módulo define funciones y clases que implementan un sistema flexible de registro de eventos para aplicaciones y bibliotecas. El beneficio clave de tener la API de registro proporcionada por un módulo de biblioteca estándar es que todos los módulos de Python pueden participar en el registro, por lo que el registro de su aplicación puede incluir sus propios mensajes integrados con mensajes de módulos de terceros.” [Logging2019].

2.5.8. Microsoft Kinect

Kinect es un dispositivo desarrollado por Microsoft, que estuvo en el mercado desde el año 2010 y fue descontinuado en el año 2017. Kinect es una barra horizontal diseñado para ser colocado longitudinalmente por encima o por debajo de la pantalla de vídeo (Ver fig. 2.26).

El dispositivo cuenta con una cámara RGB, un sensor de profundidad, un micrófono de múltiples matrices y un procesador personalizado que ejecuta el software patentado, que proporciona captura de movimiento de todo el cuerpo en 3D reconocimiento facial y capacidades de reconocimiento de voz.



Figura 2.26: Microsoft Kinect.

Algunas de sus características más relevantes son: Iluminación infrarroja independiente (30 fps), Esqueleto de 25 puntos para un total de seis personas (cada persona tiene 25 articulaciones esqueléticas) Seguimiento del pulgar, seguimiento del final de la mano, gestos con las manos abiertas y cerradas. Alcance de operación de desde 0.5 metros cerca a 4.5 metros lejos [Kinect2019].

2.5.9. Web Sockets

“WebSockets es una tecnología avanzada que hace posible abrir una sesión de comunicación interactiva entre el navegador del usuario y un servidor. Con esta API, puede enviar mensajes a un servidor y recibir respuestas controladas por eventos sin tener que consultar al servidor para una respuesta.” [WebSockets2020].

2.6. ANTECEDENTES

2.6.1. Detección de objetos en imágenes

“La detección de objetos es una tarea importante de visión por computadora que se ocupa de detectar instancias de objetos visuales de una determinada clase (como humanos, animales o automóviles) en imágenes digitales... ”, para dar respuesta a “... ¿Qué objetos están y dónde están?” [zou2019]. (Traducción propia).

Durante las últimas dos décadas ha habido un amplio interés en el desarrollo de técnicas que permitan identificar objetos en imágenes. Aunque, las investigaciones en este campo se remontan más atrás, con trabajos como el presentado por Vaillant et al. haciendo uso de redes neuronales [Vaillant1994] o pocos años más tarde LeCunn et al. presentando el uso de capas convolucionales, dentro de las redes neuronales tradicionales [LeCun1998]. Aunque el uso de capas de convolución fueron muy limitadas en su momento, debido a los requerimientos de computo, hoy en día que se han convertido en pieza fundamental de los desarrollos más resientes en el la visión por computador.

Las propuestas para identificar objetos en imágenes pueden ser divididas en dos grupos, el primero está compuesto por los métodos que podrían llamarse tradicionales vigentes hasta el año 2014. Por otro lado, están los métodos basados en redes neuronales, los cuales mostraron su amplia superioridad respecto a los anteriores, destacando en el año 2014 el método denominado R-CNN [Girshick2014] que logró una precisión de 58.5 % en conjunto de prueba “The Pascal Visual Object Classes (VOC) challenge”[VOC2010].

Métodos tradicionales

Uno de los métodos de mayor popularidad, en detección de objetos, a principios de los años 2000 fue presentado por P. Viola y M. Jones [Viola2001, Viola2004], cuyo algoritmo, actualmente conocido como “VJ Detector” permitió la detección de rostros en tiempo real (15 cuadros por segundo) en una máquina con un procesador de 700MHz.

Según los propios autores, Viola y Jones, resaltan tres etapas y contribuciones clave de su trabajo.

En primer lugar la representación integral de la imagen (Integral Image), que permite identificar rápidamente las características utilizadas por el detector. El segundo es un clasificador basado en AdaBoost que a su vez permite la selección de unas pocas características críticas. Finalmente, la combinación en cascada de clasificadores, lo que permitió descartar rápidamente zonas como el fondo, dando mayor tiempo de computo a las zonas prometedoras.

En el año 2005, N. Dalal y B. Triggs, presentaron el método para la extracción de características conocido como “Histograma de gradientes orientados” (HOG - Histogram of Oriented Gradients) [Dalal2005]. El método de Dalal y Triggs se motivó en el problema de detección del peatón, y fue uno de los principales métodos para resolver este problema en su época.

Dalal y Triggs propusieron una forma en la cual generalizar las características de una imagen a partir de la variación de luminosidad de cada píxel o conjunto de píxeles, en otras palabras, la generación de flechas que indican la dirección en la que los vecinos de un píxel son más oscuros respecto el píxel observado. Posteriormente, esas flechas son pasadas a un clasificador SVM previamente entrenado que se encarga de identificar qué grupo de flechas se asemejan al conjunto extraído durante el entrenamiento.

Basado en el método de HOG, hacia 2008 Felzenszwalb et al. presentaron un modelo deformable basado en las partes (DPM - Deformable Part-based Model) de los objetos que componen. Convirtiéndose en el método con mejores resultados hasta antes de la aparición de los métodos basados en Deep learning. La estrategia de este método consistió en identificar las partes que componen un objeto, por ejemplo, en una persona, de igual forma que HOG, se identifica el rostro, los brazos, las piernas, etc. por separado, para posteriormente otorgar una etiqueta de conjunto. Este método tuvo gran éxito y fue mejorando hasta llegar a la versión 5 del mismo, en el año 2014, año en que fue superado ampliamente por los métodos basados en redes neuronales.

Métodos basados en deep learning

En el año 2014 Girshick et al. presentaron R-CNN [Girshick2014], el resurgimiento de las redes convolucionales como mecanismo para el procesamiento de imágenes. RCNN consiste en dos etapas o estados, en primer lugar extrae cajas candidatas a objetos, luego cada una de las cajas tomadas de la imagen original es escalada para ajustarla al tamaño de entrada de una red neuronal convolucional que previamente se entrena con el conjunto de datos de ImageNet [ImageNet2009], al finalizar el proceso un clasificador SVM discriminaba si un objeto específico se encontraba en la imagen procesada. La superioridad de este tipo de métodos se demostró con el conjunto de prueba del “PASCAL VOC Challenge” [VOC2010] donde superó a los mejores métodos hasta entonces, con hasta un 25 % más de precisión.

Aunque una mejora significativa en la precisión, RCNN presentaba un gran inconveniente para su implementación en dispositivos de bajo poder de computo, debido a los tiempos de procesamiento empleados, pues se generaba un gran número de cajas a procesar. El siguiente año a la presentación de RCNN, He et al. presentaron un método denominado SPP-Net que reduce el tiempo de procesamiento entre 24 y 101 veces [He2015]. Este método tiene la ventaja de que evita la repetición del procesamiento de las características de convolución, ya que calcula los mapas de características de la imagen completa solo una vez, adicionalmente, los detectores son entrenados sobre regiones arbitrarias de longitud fija.

El mismo 2015, nuevamente Girshick presentó Fast R-CNN [Girshick2015], una mejor a su propuesta inicial. Fast R-CNN logra que el tiempo de entrenamiento de la red VGG16[Simonyan2014VGG] sea 9 veces más rápida en comparación con R-CNN y hasta 213 veces más rápida en tiempo de prueba.

Los trabajos de SPP-Net y Fast R-CNN, expusieron como principal problema de R-CNN en términos de rendimiento, la sobrecarga en la estimación de las regiones propuestas. En 2015 Ren et

al. presentaron “Faster R-CNN”, una red de propuesta de región (RPN) que comparte características convolucionales con la red de detección, permitiendo así propuestas de región casi sin costo [Ren2015].

Hasta este momento, los métodos expuestos para la detección de objetos estaban basados en dos estados: en un primer momento se genera un conjunto de regiones candidatas, a partir de búsquedas selectivas como “Deep Mask” [Pinheiro2015], RPN [Ren2015] o “Edge boxes” [Zitnick2014]. Posteriormente, las regiones de interés (RoI) son pasadas por una red convolucional para determinar la pertenencia a una determinada clase. Si bien esta estrategia obtenía buenos resultados, seguía requiriendo alta carga de procesamiento.

Como respuesta y solución a los problemas de velocidad de los métodos de dos estados surgen los métodos de un estado. En el año 2016 Redmon et al. presentaron YOLO [Redmon2016] (You Only Look Once), un cambio significativo respecto a los métodos anteriores, aunque con resultados similares en su precisión, logra el procesamiento de 45 cuadros por segundo, lo que permite su implementación en sistemas que requieran procesamiento en tiempo real. Pero incluso una velocidad aun mayor, 155 cuadros por segundo en una versión reducida de su diseño, aunque con menor tasa de éxito. La filosofía del modelo propuesto en YOLO, consiste en la integración en una única red la detección de los cuadros que delimitan un objeto y los porcentajes de probabilidad de pertenencia a una determinada clase, lo que permitió a los autores optimizar significativamente la cantidad de procesamiento requerida.

También en 2016, Liu et al. presentaron SSD [Liu2016], el segundo detector de un solo estado. SSD Parte de la red VGG[Simonyan2014VGG] tradicionalmente utilizada para la clasificación de imágenes, pero elimina las capas de clasificación y agrega otras capas de convolución propias, logrando con un 74 % superar la precisión de YOLO de 63 % en el conjunto de prueba “Pascal VOC” [VOC2010].

En 2017 Lin et al. presentaron una propuesta que denominan pérdida focal (focal loss) [Lin2017] donde argumentan que la razón central del desequilibrio entre ambas clases se debe a que hay una descompensación entre lo que son los objetos a detectar y el fondo de la imagen, en otras palabras que durante el entrenamiento, las regiones identificadas como fondo, por su cantidad, pueden abrumar los escasos ejemplos positivos, por tanto, proponen modificar el método de estimación de perdida de entropía cruzada, disminuyendo la perdida que se da a los casos bien identificados, logrando tasas de éxito y de velocidad comparables a los mejores métodos de su momento, como demostración de lo propuesto, los autores crearon un detector simple llamado RetinaNet [Lin2017].

En 2018 Zhang et al. presentaron RefinaDet [Zhang2018] con el objetivo de integrar, aún más, la precisión de los modelos de dos etapas como R-CNN [Girshick2014] y la velocidad de los modelos de una etapa como SSD [Liu2016]. RefinaDet consiste en dos módulos interdependientes, donde el primero va eliminando los candidatos negativos, mientras que el otro módulo va realizando la clasificación sobre los restantes.

2.6.2. Localización de personas en una imagen

La localización de una persona en una imagen consiste en separar la persona de todos los demás elementos de una imagen, sin importar la postura, la forma, el color o incluso la vestimenta que use.

Tener cada persona que aparece en una imagen claramente diferenciada, facilita la realización de tareas de índole superior, como identificar una postura o incluso una emoción, ya que permite centrar el análisis en los píxeles que conforman la persona, lo que reduce la cantidad de datos a analizar.

Una técnica comúnmente utilizada en el cine, la televisión y fotografía, conocida como croma o uso de croma, permite la extracción de la persona usando un fondo de un color homogéneo, usualmente verde, como se muestra en la figura 2.27. Sin embargo, esta técnica no puede ser aplicada de forma

general, pues requiere de un ambiente controlado donde, además del fondo uniforme se controle la iluminación.



Figura 2.27: Extracción del fondo usando croma. Imagen de <https://www.youtube.com/watch?v=RH29BPabjs0>.

Una técnica diferente al uso de croma, para la extracción de la persona se basa en el uso de sensores de profundidad (RGB-D), siendo el “Microsoft Kinect” el dispositivo de este tipo el más popular durante la última década, en este campo de la investigación ([**Adhikari2017**, **Lin2016**, **Stone2015**, **Galna2014**]). Esto se debe, por un lado, a su bajo coste y por otro, a que el uso del dispositivo Kinect simplifica algunas tareas como la sustracción de fondo o la generación de esqueletos (la figura 2.28 muestra un ejemplo de la extracción realizada). Esto ha permitido que investigadores de muchas áreas lo hayan adoptado dentro de sus trabajos. Sin embargo, presenta una limitante muy importante, sobre todo cuando se desea usar en espacios abiertos o de gran amplitud, pues su precisión se reduce después de unos pocos metros de distancia [**Fan2017**] (cerca de 4.5 metros), además que su capacidad de detección y extracción de personas está limitadas a no más de seis, por lo que su uso en espacios abiertos o muy transitados es poco práctico.



Figura 2.28: Extracción del fondo usando Kinect. Imagen de Microsoft.

Un enfoque diferente a los mencionados, consiste en el análisis de las imágenes RGB tradicionales para localizar y extraer la persona. Para ello se valido del desarrollo de algoritmos de inteligencia artificial y especialmente redes neuronales convolucionales, lo que ha permitido el uso de dispositivos menos especializados, como cámaras web de computadora [**Lie2018**, **Fan2017**, **Ravi2016**]. Si bien

es cierto que, aun cuando el análisis de imágenes RGB simples imponen una mayor dificultad que los dos métodos anteriores, pues no se tiene el apoyo de una variable fija como en el caso del croma, o de información de profundidad con en el caso de los sensores RGB-D, tiene a su favor: un costo aun menor que el de los dispositivos Kinect, se ve menos afectado en ambientes no controlados y reducen los problemas de pérdida que se da con el aumento de distancia entre la cámara y la persona en espacios abierto y, finalmente, no tiene un número límite de personas que pueden ser detectadas en una imagen, estando solo sujeto al tamaño mismo de la imagen a analizar.

La detección, detección y extracción de personas en una imagen RGB tradicional puede estar enmarcado en un campo más amplio conocido como detección de objetos. La detección de objetos es un reto que se ha venido superando con rapidez, en los últimos años, esto se debe, en gran medida, al trabajo base R-CNN, presentado Girshick et al. [Girshick2014] y posteriormente “Faster R-CNN” presentado por Ren et al. en [Ren2015], trabajos que hacen uso de redes neuronales convolucionales muy profundas y de los cuales se extiende uno de los trabajos más destacados en la detección de objetos y dentro de ellos, de personas. “Mask R-CNN” [He2017] que fue presentado en el año 2017 por He et al., permite que, analizando sólo imágenes RGB sin información adicional puede generar una máscara o silueta que envuelve la persona, obteniendo resultados como los que se muestran en la figura 2.29, este trabajo además es un proyecto de código abierto y que puede ser descargado de https://github.com/matterport/Mask_RCNN.



Figura 2.29: Extracción del fondo usando Mask RCNN. Imagen de https://github.com/matterport/Mask_RCNN.

2.6.3. Identificación de posturas

La estimación de la postura humana es un proceso por el cual se pretende identificar la configuración de las partes del cuerpo de una persona [Poppe2007], es decir, la posición de cada parte respecto a las demás, de tal forma que dicha configuración pueda ser etiquetada. Normalmente los investigadores se han centrado en localizar las articulaciones principales, ya que con esto se puede generar una nube de puntos o bien reconstruir el esqueleto completo [Rojas-Albarracín2010]. Una vez identificadas las partes del cuerpo y unidas por líneas, este esqueleto podrá ser analizado mediante el cálculo de los ángulos resultantes como presenta [Rojas-Albarracín2009].

Poner imagen de esqueleto formato por puntos, puede ser vitruvio

El identificar la postura tiene utilidad en diferentes campos, siendo la salud y el ocio los más destacados. Por ejemplo, a nivel médico, identificar la postura de una persona podría permitir que se valore respecto a algunos óptimos de salud, es decir, si la postura de una persona puede afectar su

salud a corto, mediano o largo plazo. A este respecto, se han propuesto diferentes ideas que expresan que los desórdenes musculo-esqueléticos de una persona están, en muchos casos, relacionados con las acciones que realiza (movimientos, repeticiones, esfuerzo y postura) [Karsh2006]. Esto ha motivado la investigación para determinar la postura relacionados con algunos problemas de salud usando mecanismos informáticos [Balista2010, Bianch2006].

Aunque identificar las partes del cuerpo de una persona en entornos controlados, como puede ser un estudio de grabación, puede lograrse usando un sistema tradicional de marcas o sensores puestos sobre la persona, sin embargo, fuera de esos entornos tener un sistema basado en marcas o sensores sobre el sujeto a observar presenta dos problemas: primero, implica costos adicionales como trajes especiales o los mismos ambientes controlados, que deben tener una iluminación bien definida y fondos normalmente monocromáticos. El segundo problema es el posible rechazo para el uso continuado de estos mecanismos, además de que algunas personas por su edad o su capacidad de movilidad, puede requerir del acompañamiento o ayuda para su puesta. Por lo anterior, en la última década ha habido un gran número de propuestas para automatizar la tarea de identificación de la postura humana.

De igual forma a la detección de objetos, la identificación de la postura es un reto que se remonta a varias décadas y cuyas propuestas pueden ser divididas en dos tipos: los métodos tradicionales anteriores al año 2014 y los métodos basados en redes neuronales a partir del mismo año (destacando en las primeras la combinación de los métodos HOG y SVM [Aslan2020CnnVsHogSVM], ver ?? y 2.2.2 respectivamente).

Métodos tradicionales

En el año 2007 Ramanan et al. presentan una propuesta en la que se hace análisis de los contornos de las imágenes [Ramanan2007]. Los contornos son comparados con modelos deformables para estimar la probabilidad de la postura.

Posteriormente en el 2008, Ferrari, Jiménez y Zisserman presentan una propuesta [Ferrari2008] basada en el trabajo Dalal y Triggs [Dalal2005], usando HOG se genera un vector de datos que luego es clasificado usando SVM, la particularidad de este trabajo es que se aplicó a videos de series de televisión, con lo cual solo se estima la postura de la parte superior del cuerpo. Un trabajo bastante citado el mismo año 2008 fue presentado por Felzenszwalb, McAllester y Ramanan [Felzenszwalb2008], el cual también hace del mismo SVM y de HOG y apoyándose en un modelo de partes deformables.

Durante siguientes años la principal estrategia consistió en el uso del descriptor HOG + el clasificador SVM [Bourdev2010, Johnson2011, Ladicky2013], logrando algunos mejorar los resultados a partir de conjuntos de datos propios [Bourdev2010, Johnson2011, FLIC2013]. Sin embargo, en conjunto de imágenes de prueba públicas como Pascal VOC [VOC2010] los resultados no fueron demasiado significativos siendo el trabajo de Bourdev y Malik [Bourdev2010] uno de los mejores del año 2010, alcanzando una precisión de 36 %.

El trabajo con mayor precisión de las propuestas con métodos tradicionales fue hecha por Pishchulin et al. en el año 2013 [Pishchulin2013]. Este trabajo logró una precisión de 62.9 % en el conjunto de pruebas LSP [LSP2010]. Este trabajo hace uso de un modelo de estructuras pictóricas (PS) que representa la configuración del cuerpo como una colección de partes rígidas y un conjunto de conexiones parciales. Tomando estas conexiones en forma de árbol es posible realizar la inferencia de la postura de forma rápida. Esta propuesta además, hace uso del detector AdaBoost.

En el año 2014 Zhao et al. [Zhao2015] presentó una propuesta basada en varios de los algoritmos más populares en el campo de procesamiento de imágenes. La propuesta partió de la definición de una estructura de tres capas para determinar una variada gama de poses mediante una estructura de árbol. Las partes son identificadas por separadas para luego ir componiendo el cuerpo a partir de la relación de árbol que hay entre cada parte. Cada parte es estimada usando el algoritmo SVM

pre-entrenado, con los pesos generados por HOG, siguiendo la estrategia de aprendizaje supervisado. Posteriormente la relación de hijos a padres es estimada usando el algoritmo de K-means.

Métodos basados en deep learning

El 2014 se convirtió en el año que el uso de redes neuronales marcaron un antes y un después en el campo de la visión por computador. Numerosos trabajos haciendo uso de redes neuronales demostraron la superioridad de estas sobre los método tradicionales. Fue este mismo año que Toshev y Szegedy, pertenecientes al equipo de Google, presentaron DeepPose [Toshev2014], la primera propuesta para la estimación de la postura, usando redes neuronales profundas con la inclusión de capas de convolución, con lo que se alcanzó una precisión del 69 %, superando por siete puntos el trabajo de Pishchulin et al. [Pishchulin2013], el mejor resultado que existía en el estado del arte de ese momento.

Posteriormente, en el año 2015 Tomposon et al. implementaron el concepto de mapas de calor [Tompson2015], con lo que la posición de cada articulación principal, de una persona, se va refinando al pasar la imagen por varias capas de convolución y agrupación (pooling). Siendo el mapa de calor la zona de la imagen con mayor probabilidad de ser una articulación (ver figura 2.30). Tomposon et al. lograron en su propuesta [Tompson2015], una tasa de precisión del 80 % en el conjunto de imágenes de prueba MPII-Human-pose [MPII2014] (ver sección 2.4.11) y de 70 % en el conjunto de prueba FLIC [FLIC2013] (ver sección 2.4.13).

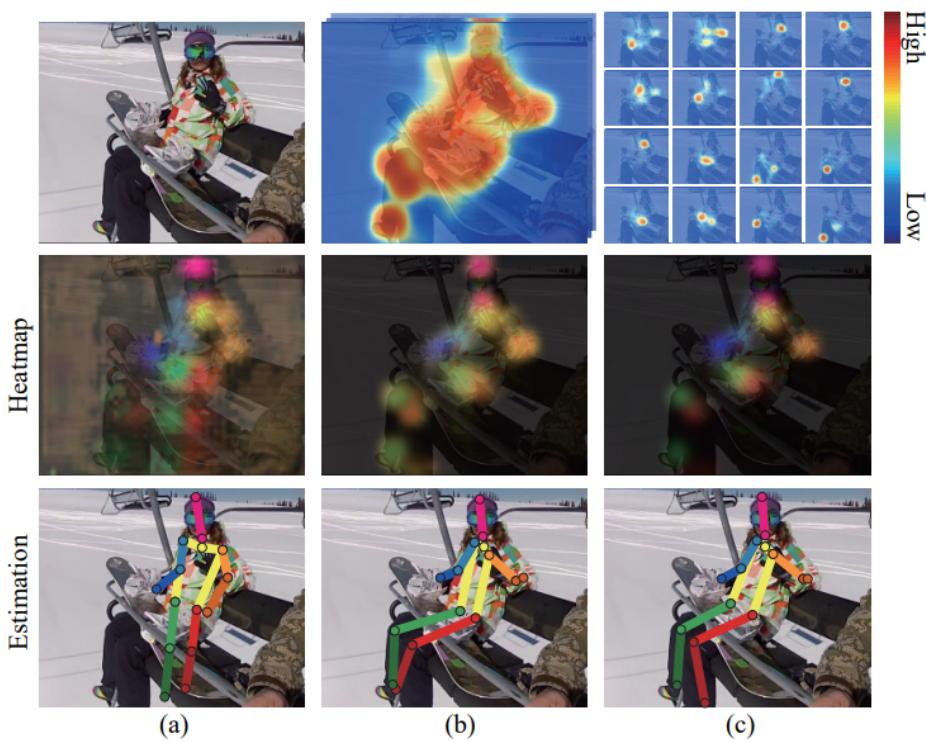


Figura 2.30: Identificación de postura con mapas de calor. Imagen de [Chu2017].

Una de las propuestas con mejores resultados para localizar las partes del cuerpo fue presentada por Newell, Yang y Deng [Newell2016] en el año 2016, alcanzando una tasa de acierto del 90.9 % en el conjunto de imágenes de prueba MPII-Human-pose [MPII2014] (ver sección 2.4.11). La estrategia de Newell et al., fue influenciada fuertemente por el trabajo de Tompsonon et al.[Tompson2015], aunque agregando, a su estrategia, una arquitectura denominada de 'relojes de arena' apilados (ver sección ??), en la cual combina varias capas convolucionales reduciendo la ventana de análisis y luego vuelta a aumentar para fortalecer las características predominantes (ver figura 2.31).

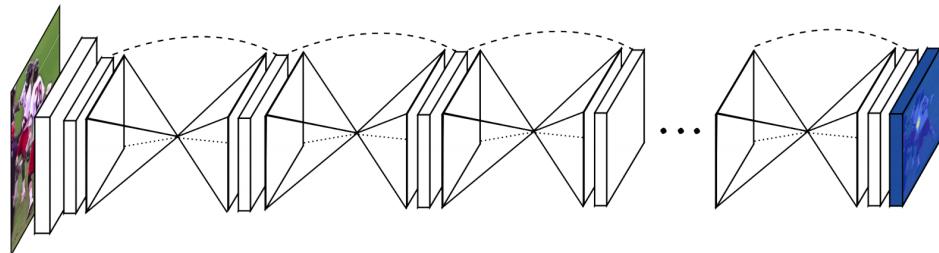


Figura 2.31: Arquitectura de relojes de arena. Imagen de [Newell2016].

En el año 2017 se presentaron varias propuestas en las que se planteaban diferentes arquitecturas. Cao et al. en [Cao2017] presentaron una propuesta que permite la identificación de las articulaciones de múltiples personas en tiempo real, con una precisión del 75 % en el conjunto de imágenes de prueba MPII-Human-pose [MPII2014]. Muy superior en la precisión, Yang et al. propuso una arquitectura de pirámides residuales (PRM - Pyramid Residual Module) [Yang2017]. Un concepto que combina el concepto de “relojes de aren” con la arquitectura de redes residuales, muy efectivas en redes profundas, logrando la extracción de las características a diferentes escalas. Con esta propuesta Yang et al. [Yang2017] lograron una precisión del 92 % en el conjunto de imágenes de prueba MPII-Human-pose [MPII2014].

Una novedosa estrategia fue utilizada por Chen et al., quienes hicieron uso de las redes generativas adversarias (ver ??) [Chen2017]. Chen et al. entrenaron su modelo para que aprendiera a discriminar las posturas que son biológicamente válidas de las que no lo son para generar resultados más precisos en la discriminación de los puntos correspondientes a las articulaciones, alcanzando una precisión del 92.1 % en el conjunto de imágenes de prueba MPII-Human-pose [MPII2014].

Combinando estrategias muy importantes y de reciente aparición, como las de relojes de arena”(ver ??) y redes residuales (ver 2.1.10) Chu et al. en [Chu2017] presentaron una propuesta en la que incorporan el concepto de zonas de atención o mapas de calor (similar al concepto de región de interés, ver ??). Con la generación de las zonas de atención en diferentes resoluciones logran eliminar fácilmente los posibles falsos positivos y alcanzando una precisión del 92.6 % en el conjunto de imágenes de prueba MPII-Human-pose [MPII2014].

Independientemente de las variaciones, las estrategias de relojes de arena y mapas de calor es la de mayor recurrencia en los trabajos recientes, para la identificación de la postura [Tompson2015, Newell2016, Ning2018, Luvizon2019], incluso integrándolas con modelos más sofisticados como las redes generativas adversarias [Chou2018]. Y principalmente a partir del año 2014 se ha evidenciado que el objetivo de mapear la postura de una persona a partir de imágenes, usando redes neuronales, ha permitido superar el 90 % de precisión en conjuntos de pruebas tradicionales, a pesar de las grandes variaciones y la restricciones que imponen la propia anatomía del cuerpo humano.

2.6.4. Reconocimiento de actividad humanas (HAR)

El reconocimiento de la actividad humana (HAR por sus siglas en inglés) consiste en reconocer la acción que está realizando o el evento que está teniendo una persona. Identificar la actividad humana va un paso más adelante con respecto a la identificación de las posturas, agregando una dimensión más al análisis, ya que no se limita a un estado estático sino que pretende poner una etiqueta a las posturas en un marco temporal.

Caminar, saltar, tener un infarto, caerse, son algunos de los sucesos que involucran a personas y que pueden ser etiquetados por las técnicas de HAR en bajo, mediano o alto nivel. Cuando se habla de de bajo nivel corresponde con acciones muy simples realizadas por una persona, por ejemplo, “Ponerse de pie”. Mediano nivel hace referencia a eventos que incluyen identificación de acciones que es difícil confundir con otras, aun cuando el movimiento que realiza la persona puede ser encontrado

en otros momentos, además incluye la interacción con objetos, como “tomar el cepillo de dientes” o “poner crema en el cepillo”, mientras que las etiquetas de alto nivel hacen referencia a eventos más complejos que en sí mismos están compuestos por un conjunto de etiquetas de mediano nivel, como puede ser el “lavarse los dientes”.

El reconocimiento de actividad humana tiene gran relevancia en diferentes campos de investigación debido a su aplicabilidad en diversos contextos de la vida diaria, como puede ser la aplicación en entornos vigilados, monitorización de pacientes [Cho2009Parkinson], espacios inteligentes saludables [Akula2018HarCnn, Nweke2019, mehr2019HarCnn], robótica e incluso la interacción misma entre una persona y un ordenador. En muchos de los entornos mencionados es fundamental el reconocimiento de la actividad de la persona, para tener etiquetas simples que permitan tomar decisiones específicas basadas en las acciones humanas, por ejemplo, en un sitio de cuidado de pacientes puede ser útil identificar acciones como: el paciente fue al baño, el paciente tomó su medicina, el paciente se cayó al suelo o cualquier otra que pueda afectar el cuidado de la persona.

Durante las últimas décadas se han presentado una gran variedad de propuestas [Jobanputra2019SurveyHar] para la identificación de diferentes tipos de acciones. Algunas de ellas se caracterizan por el uso de dispositivos vestibles[Gonzalez2015] o portables como teléfonos móviles que analizan datos obtenidos de diferentes tipos de sensores, principalmente de acelerómetros o giroscopios [Chen2015, Chen2016, Murad2017, Guan2017, Hur2018AcelerometerCNN].

Al uso de estos dispositivos ha permitido a los investigadores tener tasas de éxito altas, alcanzando una gran precisión a la hora de identificar actividades que pueden ser descritas de forma secuencial por los movimientos realizados por el cuerpo en su conjunto, tal es el caso de caminar, saltar, subir o bajar escaleras. Sin embargo, el uso de este tipo de dispositivos puede ser considerados como invasivos, puesto que debe ser portado por la persona a analizar, lo que en algunos casos es un inconveniente mayor al requerir la intervención de la persona misma. Adicionalmente, aumentan los costos cuando se piensa en soluciones de gran escala.

Por otro lado, el uso de dispositivos como cámara RGB tradicionales u otros sensores permite realizar el análisis de una forma diferente, basado en visión técnicas de visión artificial, como se muestra en a continuación.

Uso de visión artificial en HAR

Identificar la acción que está realizando una persona, de forma automática, ha sido un reto en el campo de la visión por computador principalmente por la gran variabilidad de los datos de entrada y, aunque de forma simple se puede pensar en el análisis de las tradicionales imágenes RGB, no son estas el único tipo utilizado, de hecho algunas de las de las primeras propuestas se apoyaron en el uso de cámaras infrarrojas [Sokolova2013] y, posteriormente, el sensor kinect de Microsoft (ver sección 2.26) se convirtió en uno de los dispositivos más usados en este campo debido a su bajo coste y precisión.

Una de las principales ventajas del uso de este tipo de soluciones es que los datos pueden ser analizados de diferentes formas, por ejemplo: usar silueta de la persona [Vishwakarma2015, Zeng2014, Balista2010], el esqueleto [Meshry2016] generado a partir de la identificación de las articulaciones o incluso la imagen en su totalidad [Rojas2019heart] lo que a su vez podría otorgar un contexto a la acción.

El principal inconveniente del uso de imágenes se centra en que la cantidad de datos a analizar aumenta las necesidades de recursos computacionales, lo que puede limitar su uso en algunos contextos que requieran respuesta en tiempo real, sin embargo, es oportuno mencionar que este problema cada vez es menos significativo debido al aumento del poder de cálculo disponible y a la reducción de precios de éste.

El proceso completo para el reconocimiento de actividades humanas usando técnicas de visión artificial puede ser dividido, de forma general, en tres paso: primero, Localización y extracción de la persona (ver sección 2.6.2). Segundo, identificación de características simples como pueden ser el área (de la silueta [Lin2016, Sokolova2013]) o la postura (ver sección 2.6.3). Tercero, combinación de características para identificar su similitud con acciones predefinidas.

Considerando los pasos mencionados, una de las principales decisiones al diseñar un sistema de HAR consiste en definir la forma en que la información será atomizada, es decir, definir la mínima unidad de información que puede ser utilizada en el tercer paso para identificar la acción. Por esta razón, dependiendo el objetivo final en el reconocimiento de la acción, se han presentado diferentes formas de abordar el problema, cada una con una fuerte dependencia a la forma de localización y extracción de una persona en imágenes. Esto conlleva la selección de mecanismo para la preparación de los datos a analizar, las formas más comunes son: Análisis secuencial y análisis de volúmenes.

- **Análisis secuencial:** Consiste en el análisis de vectores de características o etiquetas, en el primer caso puede tomarse como ejemplo un conjunto de articulaciones o el ángulo entre ellas. En el segundo caso puede tomarse el mismo conjunto para establecer etiquetas que caractericen posturas específicas como: “Estar de pie”, “Pie derecho adelante”, “Pie izquierdo adelante” o cualquier otro, de esta forma se tiene un vector a comparar con modelos predefinidos, como se ve en la figura 2.32.

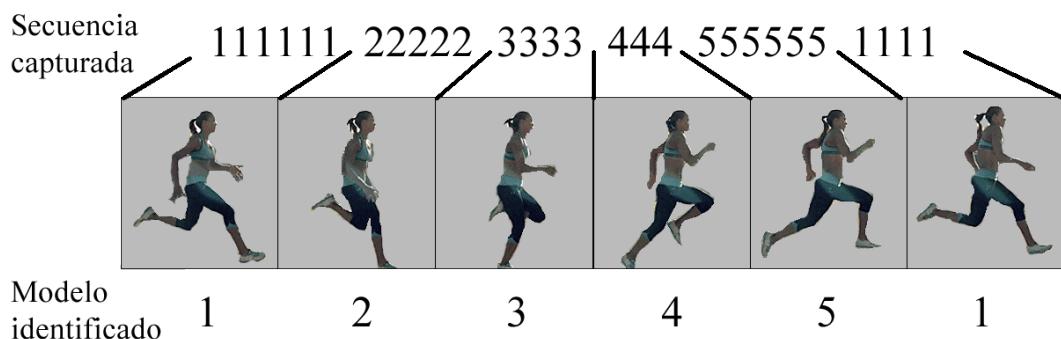


Figura 2.32: Análisis secuencial para HAR.

- **Análisis de volúmenes:** Consiste en tomar la secuencia de imágenes para generar volúmenes para, posteriormente, compararlo con plantillas predefinidas y, según el grado de similitud, identificar la acción.

Cuando se habla de volúmenes es necesario considerar 3 dimensiones (3-D), para el caso de secuencias de vídeo tradicionales las 3-D no hacen referencia a 3 dimensiones espaciales (XYZ) sino a 2 espaciales y una temporal (T). Esta ultima considerada como la posición en de un fotograma en la secuencia ordenada. Así pues, a las 2 dimensiones (XY) de un fotograma es adicionada la dimensión de tiempo (T), siendo un el sistema en conjunto de 3-D (XYT) o de espacio-temporal, como se muestra en la figura 2.33.

Al igual que en otras áreas de la visión por computador, el uso de las redes neuronales en la ultima década ha permitido un salto significativo respecto a las técnicas anteriores o tradicionales. A continuación, serán presentadas los métodos tradicionales que se usaron para el reconocimiento de actividad humana y los actuales basados en aprendizaje automático.

Métodos tradicionales para el reconocimiento de actividades humanas

Previo a la popularización de las redes neuronales, en especial las redes convolucionales, se presentaron propuestas con diferentes métodos para la inferencia de la acción realizada por una persona en imágenes, sin embargo, la mayoría de las propuestas se centró en el uso de la silueta para su análisis,

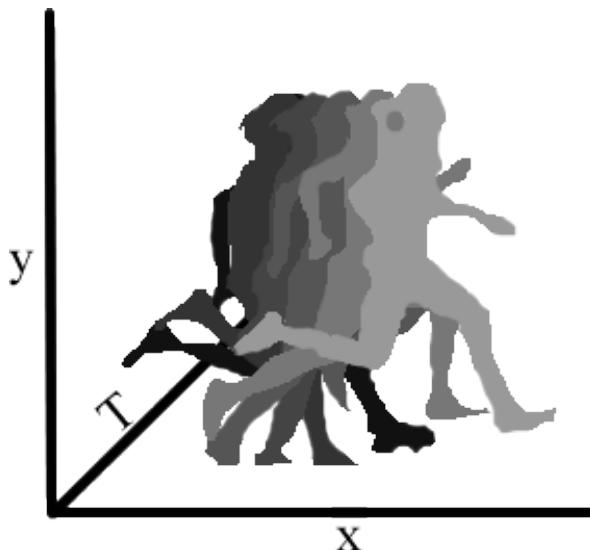


Figura 2.33: Análisis por volumen para HAR.

tal es el caso de la propuesta hecha por Cho et al. en [**Cho2009Parkinson**] donde combina los algoritmos de análisis de componente principal (PCA por sus siglas en inglés) y análisis discriminatorio lineal (LDA por sus siglas en inglés), para reconocer los patrones al andar, que pueden ayudar a diagnosticar la enfermedad de Parkinson en imágenes de las cuales se la silueta de binarizada de la persona.

Una propuesta más simple fue presentada por Balista, Soriano y Saloma en [**Balista2010**] donde propusieron un método para identificar anomalías en el caminar de una persona, que podrían estar relacionadas con problemas de salud. Para ello, el método propuesto hace un análisis simple de la silueta frontal de una persona extraída a partir de imágenes donde la persona vestía un traje blanco y el fondo era negro. El análisis consistió en un método propio donde la silueta fue dividida verticalmente, de forma que se pudiera identificar irregularidades al caminar, ya que los autores esperaban que en el caminar normal de una persona los dos lados se deberían comportar de forma similar a un espejo cuando se equiparan los movimientos de cada lado.

Como se mencionó antes, con la aparición del dispositivo Kinect de Microsoft (ver 2.5.8), varios autores presentaron propuestas en las que hacían uso de éste, especialmente por la generación de los esqueletos que entrega el dispositivo. Por ejemplo, en [**Sung2012HARKinect**] se toman los esqueletos y calculan los ángulos que forman las líneas entre las articulaciones relacionadas, también calculan las distancias euclídeas en tres dimensiones entre las articulaciones para identificar, por ejemplo si tiene una mano en el abdomen o cerca a la cabeza. En sus experimentos aplicaron su método a cinco entornos: Cocina, Oficina, baño, Sala de estar y dormitorio; para identificar doce diferentes actividades, entre ellas: hablar por teléfono, trabajar en el ordenador, lavado de dientes. Para la inferencia de la actividad toman hasta nueve fotogramas repartidos en tres segundos y aplican los Modelos de Markov de Entropía Máxima (MEMM por sus siglas en inglés) en dos capas: en una capa de nivel medio encuentra actividades de medio nivel como poner crema al cepillo y en una capa de alto nivel la actividad de alto nivel como lavado de dientes.

Meshry, Hussein y Torki en [**Meshry2016**] toman como base la idea de características locales ampliamente usado para la identificación de objetos. El concepto de características locales toma como un hecho que cada objeto tiene un conjunto único de características y que si éstas se encuentran juntas esto significa que el objeto está presente. Así mismo, caracterizan cada acción como un conjunto o bolsa de características de la acción. Los autores tomaron como entrada la información de profundidad y esqueletos entregada por el dispositivo Kinect para un análisis en 4D (XYZT). Cada característica a analizar fue formada por el cálculo de los ángulos del esqueleto obtenido en cada frame, teniendo así

una componente de tiempo en el modelo, para poder diferenciar en acciones simples que un solo frame no permite identificar, tal es el caso de “Ponerse de pie” o “Sentarse”. Para la identificación de la acción, desarrollaron un método propio que denominaron “Búsqueda lineal eficiente” (ELS por sus siglas en inglés) en el cual compara el vector o bolsa de características con acciones específicas.

Adicionalmente a las propuestas anteriores, el uso de SVM fue usado para clasificar las diferentes acciones. Sokolova et al. [Sokolova2013] plantean un problema de relevancia especialmente para personas ancianas: las caídas. La propuesta [Sokolova2013] se apoya de un modelo de lógica difusa combinado con SVM (ver 2.2.2). Para la identificación de las caídas se usan imágenes de cámaras de infrarrojo y un análisis geométrico de la región segmentada (región de interés, ver ??) que corresponde a la persona. Siendo para este trabajo el ROI, el mínimo rectángulo que puede contener la persona. Para análisis se toma la variación de la geometría (variación de la relación ancho y alto) y la dimensión de tiempo (en esta propuesta una caída se enmarca como un evento que ocurre entre 1 y 3 segundos), la estrategia de lógica difusa es tomada para dotar al sistema de un mayor grado de libertad que otras propuestas.

Vishwakarma y Kapoor [Vishwakarma2015] presentaron una propuesta para el análisis de siluetas divididas en celdas, con un método de análisis que combina SVM y k-NN para identificar, a partir de la cantidad de píxeles que corresponden a la persona en cada celda, diferentes acciones como: caminar, saltar, agacharse, entre otras similares.

Métodos de aprendizaje automático para el reconocimiento de actividades humanas

En la ultima década las redes neuronales se han convertido en un pilar en el area de la visión artificial y, más aún, con la inclusión de las redes convolucionales (ver 2.1.8) el campo del reconocimiento de actividades humanas ha tenido un significativo avance, sobre todo en los últimos tres años, como se evidencia en los trabajos mencionados a continuación.

Tal es el aporte de las técnicas de aprendizaje automático al campo de HAR que Merh y Polat en [mehr2019HarCnn] etiquetan los métodos tradicionales que no usan Deep Learning (ver 2.1.6 para mas detalle sobre Deep Learning) como artesanales, principalmente por la forma en que extraen las características de las imágenes a analizar y proponen un modelo que usa redes convolucionales para la identificación de actividades humanas dentro de hogares inteligentes. La red convolucional consta de cinco capas de convolución seguidas de 2 capas tradicionales y una softmax (ver 2.2.6) de salida, logrando una tasa de éxito del 82.41 % en el conjunto de datos (ver 2.4.14) y una tasa individual para cada una de las doce actividades que identifica de entre el 69.67 % (Escribir) y el 88.77 % (caminar).

Akula, Shah y Ghosh [Akula2018HarCnn] hacen uso de un modelo de red neuronal con dos capas de convolución, que analiza imágenes infrarrojas para la identificación de caídas de personas, detectando seis actividades de bajo nivel (Ponerse de pie, caminar, caer al suelo, caer sobre un escritorio, sentarse y sentarse frente a un escritorio). En este trabajo, al igual que la inmensa mayoría de propuestas basadas en capas de convolución, tras cada capa se agregan capas de muestreo (como max pooling, ver 2.1.8 para más detalle en redes neuronales convolucionales), y una capa de aplanamiento (flatten) antes de capas de neuronas tradicionales completamente conectadas. En esta propuesta los autores anuncian una tasa de éxito del 87.44 % sobre el Dataset propio.

Khaire, Kumar y Imran [Khaire2018HarCnn] también hacen uso de las CNN y proponen como principal enfoque el tomar como entrada toda la información entregada por las cámaras DGB-D, esto es tomar tanto la información de color, la información de movimiento captada a partir de la información de profundidad y el esqueleto de la persona, en tres redes diferentes que usan en modelo VGG[Simonyan2014VGG]. En su paso final, combinan el puntaje de las capas de Softmax en una sola salida, para estimar la posible acción detectada.

Las CNN han sido combinadas con otras estrategias, bien para obtener una precisión mayor o bien para acelerar los tiempos de respuesta, por ejemplo Mishra et al. en [Mishra2020HarCnnFuzzy]

propusieron un método que combina análisis difuso con una red CNN con el principal objetivo de acelerar el proceso de inferencia en vídeo. En este trabajo el proceso es dividido en dos etapas, en primera instancia se seleccionan los frames de interés usando un análisis rápido con un sistema de inferencia difusa, donde se los fotogramas son separados si se detecta efectivamente una persona, descartando imágenes donde solo existen objetos o animales. El segundo paso es pasar solo los fotogramas seleccionados a una red convolucional que permiten clasificar hasta ocho diferentes acciones, logrando una precisión, según reportan los autores, del 96 % y un retraso de, tan solo, dos frames.

Otro trabajo que combina las redes convolucionales con otros modelos fue presentado por Imran y Raman [**Imran2019HarCnnLstm**] donde además de las CNN hacen uso de redes recurrentes (ver 2.1.9). Para el primer caso se apoyan en la arquitectura ya conocida de Resnet (ver 2.1.10) con la que se elimina o reduce el problema desvanecimiento del gradiente (ver 2.2.3), mientras que para la segunda se basa en un modelo bidireccional de la arquitectura LSTM (BiLSTM) y como entrada toman imágenes de cámaras infrarrojas.

Dang, Yang y Yin [**Dang2020Har3dSkelleton**] también se apoyan de las CNN para la inferencia de actividades, sin embargo, proponen algunos cambios al modelo tradicional eliminando el uso de capas Dropout (usadas comúnmente para reducir el sobre entrenamiento, ver 2.1.13) y de Softmax para la clasificación final, con el fin de reducir los requerimientos computacionales y lograr, con esto, mayor eficiencia para identificación de acciones en tiempo real. En lugar de ello, después de la segunda capa de convolución realizan una transpuesta a los datos que son entregados a una tercera capa de convolución, al final para la clasificación se basan en el el método de aprendizaje amplio propuesto en [**Chen2018B1s**]. Una característica adicional del modelo propuesto en el trabajo [**Dang2020Har3dSkelleton**] es la división de la entrada de la red en dos “ramas” de igual diseño salvo por la estructura de los datos de entrada, una para el ingreso de la información espacial, donde la información del esqueleto es organizado de forma similar a una imagen a color pero donde la primera dimensión tiene el numero de frames a analizar, la segunda el número de articulaciones del esqueleto y la tercera las tres coordenadas de posición de la articulación. Por otro lado, la segunda rama de entrada contiene la información temporal, calculada por la resta de la posición entre las coordenadas de las articulaciones entre frames sucesivos.

Si bien es cierto que las CNN han sido el pilar en el campo del HAR en la última década, no es el único tipo de redes utilizadas: Zeng, Wang y Yang [**Zeng2014**] propusieron para el reconocimiento de personas caminando, analizar las variaciones del aspecto ancho-alto de imágenes binarizadas que corresponde con el rectángulo mínimo que contiene las silueta de una persona caminando, desde una perspectiva lateral. Para ello hace uso de redes de función de base radial (RBF por sus siglas en inglés). Esta propuesta se basa en el carácter cíclico o periódico del movimiento del caminar, que permite a la red RBF clasificar de forma directa las etapas del andar.

Una estrategia diferente a todo lo anterior donde hace uso de técnicas de visión artificial para el análisis de datos generados por un acelerómetro, fue presentada por Hur et al. en [**Hur2018AcelerometerCNN**]. En este trabajo en lugar del análisis habitual del flujo de datos entregado por el dispositivo, éste flujo es tomado en ventanas de tiempo y almacenado como imágenes, las cuales son analizadas por un red neuronal convolucional.

Finalmente, aunque debido a su naturaleza temporal las investigaciones se centran principalmente en el uso de vídeos o secuencia de imágenes, también es posible identificar acciones en imágenes simples [**Guo2014, Rojas2019heart**].

2.6.5. Identificación de eventos anómalos

La identificación de eventos anómalos consiste en determinar si; basado en el contexto que se desarrolla una acción, un hecho envuelve situaciones que no son habituales en un marco temporal.

De la mano con el creciente interés por crear mecanismos que permitan identificar; de forma automática, las acciones de las personas, también se ha desarrollado un interés por identificar y entender si estas son “normales”, en un entorno específico. Este enfoque tiene especial relevancia en ambientes públicos y donde la seguridad tiene mayor importancia, pero también en campo de la salud tanto para prevención como para reacción oportuna.

CAPÍTULO 3

ARQUITECTURA EL SISTEMA

En este capítulo serán presentados todas las partes del sistema diseñadas, así como la forma en que interactúan y la estructura de la información que se transmite entre ellos, en busca de un sistema que además de cumplir con el planteamiento fundamental de la identificación de eventos anómalos en el hogar sea también altamente escalable.

La escalabilidad es la capacidad de adaptarse para cambiar tanto el hardware como el software. En el primer caso, un sistema es altamente escalable si tiene un procesamiento distribuido ya que esto permite, por ejemplo, pasar de realizar sus procesos de una a N máquinas, para mejorar los recursos disponibles. En el segundo caso, un sistema es altamente escalable si tiene piezas de software construidas modularmente, como componentes desacoplables que permiten agregar nuevas funcionalidades, eliminar otras o cambiar otras sin afectar a todo el sistema.

Una arquitectura que sea altamente desacoplable es una premisa fundamental en el diseño de sistemas. Esta idea permite que cada componente del sistema pueda trabajar de forma independiente sin una comunicación directa, pero con estructuras fuertemente definidas para el intercambio de datos. Esto tiene varias ventajas, entre ellas, que cada componente pueda ser ejecutado en ordenadores independientes, como se muestra en modelo físico del sistema (ver Fig. 3.11). Además, el inicio o detención de uno no altera el comportamiento de los demás. Bajo esta arquitectura el sistema se podrá escalar, adicionando nuevos componentes sin necesidad de detener los que ya estén en ejecución.

“**Un componente**”, tal como se maneja en este trabajo, es una pieza de software formada por uno o más artefactos que en su conjunto cumplen una función concreta, que puede ser la captura de imágenes de una cámara, la identificación de un evento en una imagen u otras y que por su definición no dependen de ningún otro componente del sistema para funcionar, es decir, son enteramente independientes. Claro está que su independencia no impide el funcionamiento integrado con el sistema.

Para la construcción de este sistema se definieron cinco tipos de componentes: **controlador de dispositivos, identificador de actividades humanas, analizador de eventos, notificador y pila de datos**. El funcionamiento de cada uno de ellos será explicado más adelante. También es importante mencionar que pueden existir diversas instancias de un mismo tipo de componente, por ejemplo, podría existir un componente de identificación de actividad humana que detecte personas con infarto y otro componente del mismo tipo que detecte caídas.

Adicional al planteamiento de la arquitectura, para facilitar el desarrollo de nuevos componentes, en este trabajo se han construido plantillas de las cuales partir, para la construcción de nuevos componentes integrables al sistema. Las plantillas disponibles se presentarán más adelante en las secciones 3.8, 3.9, 3.10 y 3.11.

A continuación, será presentado el diseño general del sistema para luego ir presentando cada segmento de forma específica.

3.1. DISEÑO ARQUITECTÓNICO GENERAL

En esta sección será explicado de forma general la interacción de cada componente. La arquitectura del sistema consta de cuatro partes o secciones etiquetadas por su objetivo principal. Cada sección agrupa los componentes que permiten realizar la tarea correspondiente con su ubicación en el sistema. Las cuatro secciones del sistema son (ver fig. 3.1):

1. Entrada de datos (A - Verde).
2. Reconocimiento de eventos simples (B - Amarillo).
3. Analizador de eventos anómalos y emisión de alertas (C - Rojo).
4. pila de datos (Azul Claro).

De éstas, las tres primeras son partes extensibles del sistema, es decir, que pueden ser ampliadas para, por ejemplo, adicionar nuevos tipos de dispositivos o permitir identificar nuevos eventos. En tanto que la cuarta, la 'Pila de datos' hace parte del núcleo del sistema y se diferencia de las demás principalmente en su rigidez.

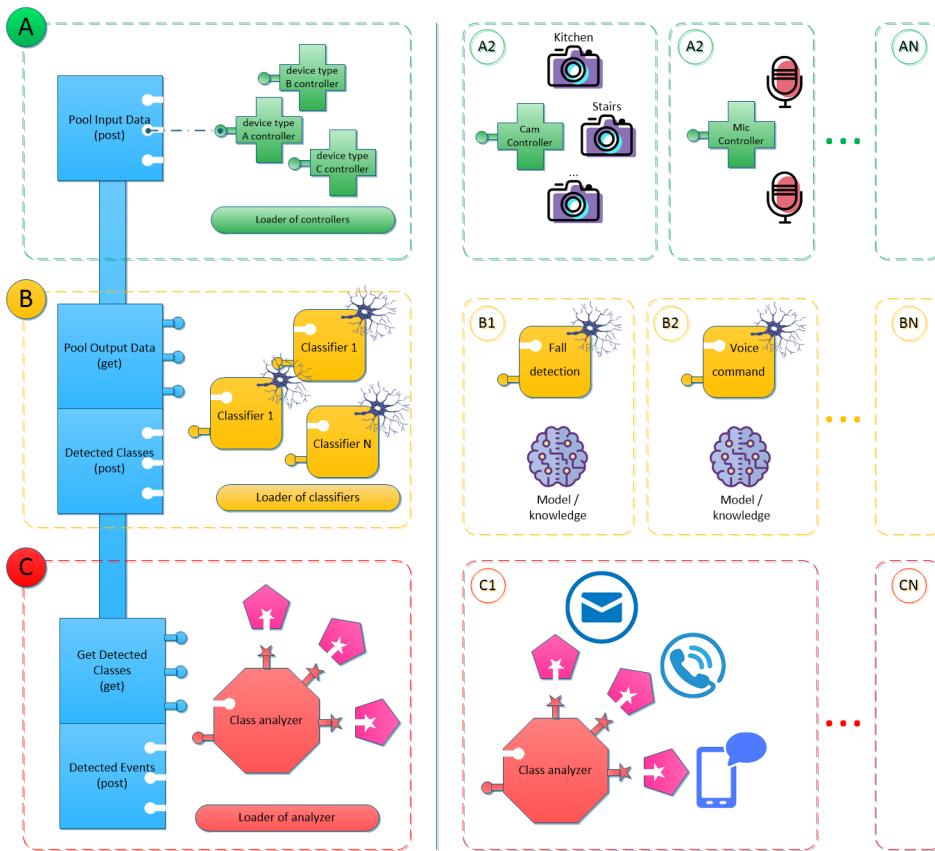


Figura 3.1: Diagrama general del sistema. Presenta las secciones en que se divide el sistema y los componentes que conforman cada sección. También presenta, la relación que hay entre cada componente y el sistema general para el intercambio de datos.

Sección A, encargada de capturar la información de entrada (por ejemplo imágenes de una cámara web) y entregarla a la pila de datos para que otros componentes puedan hacer uso de ella. Adicionalmente, en esta sección se incluyen etapas de preparación de los datos a analizar, esto es, dar una estructura estándar a los datos sin importar el origen, de esta forma se mantendrán estructuras claramente definidas para que los componentes posteriores puedan entenderlas. El funcionamiento detallado de esta sección será explicado más adelante en la sección 3.4.

Sección B, encargada de la interpretación de los datos entregados por la sección A, es aquí donde se realiza una primera identificación de eventos. La sección B consulta en la pila los datos entregados

por la sección A y los procesa de forma paralela, en búsqueda de eventos concretos. Al finalizar el proceso, los eventos detectados son entregados, a la pila de datos, como una lista de pequeñas frases con la información referente al tipo de evento, el momento y la fuente. El funcionamiento detallado de esta sección será explicado más adelante en la sección 3.5.

Sección C, encargada del discernimiento de las frases generadas de la sección B, identificando los eventos que deberán emitir alertas. A diferencia de los componentes de la sección B, que solo realizarán su procesamiento usando el entrenamiento previo, la sección C cuenta con un componente llamado “Analizador de eventos” el cual continúa su aprendizaje durante todo el funcionamiento del sistema. Dentro de esta sección se encuentran tanto los componentes de análisis de eventos como los notificadores, el funcionamiento detallado de esta sección será explicado más adelante 3.6.

La pila de datos es la columna vertebral del sistema. Se encargada de controlar toda la información que fluye en el sistema y ser el medio de comunicación, permitiendo a los demás componentes enviar y consultar la información, mediante la exposición de un API. Por tanto, la pila de datos es el único componente transversal a todo el sistema, como se muestra en la figura 3.1. El funcionamiento detallado de este componente será explicado más adelante en la sección 3.2.

3.2. PILA DE DATOS

Para lograr un procesamiento altamente distribuido y un software altamente desacoplado, es necesario que el sistema tenga una forma rígida para transmitir y compartir su información. La pila de datos es la columna vertebral del sistema, es a través de ésta que fluye toda la información del sistema.

La pila de datos consiste en una base de objetos de tipo *Data* (explicada más posteriormente en la sección Estructura de datos 3.2.1), donde se almacenará toda la información que fluye a lo largo del sistema. Así mismo, es la implementación del patrón de instancia única (ver 2.3.2), por lo que solo existirá una única pila de datos accesible por todos los módulos del sistema.

Las tareas de la pila de datos como pieza estructural del sistema van más allá que solo tener los datos. Este componente se encarga de almacenar y administrar el acceso a los datos que serán usados por los demás componentes. Para ello se apoya en la implementación de un servicio REST basado en el paquete de python: "flask"(ver 2.5.6) para proveer cuatro rutas: `"/api/tickets"`, `"/api/events"`, `"/api/alerts"` y `"/api/logs"`.

- `/api/tickets`: se encarga de recibir y transmitir la información entregada por los componentes de entrada de datos (Controllers, 3.4).
- `/api/events`: se encarga de recibir y transmitir la información generada por los componentes de reconocimiento de actividades (Classifier, 3.5).
- `/api/alerts`: se encarga de recibir y transmitir la información generada por los componentes analizadores de la actividades (Analyzers, 3.6).
- `/api/logs`: se encarga de recibir información sobre sucesos o errores dentro de los diferentes componentes del sistema, de forma que se pueda tener una bitácora de la salud de todo el sistema.

En relación con lo anterior, la pila de datos cuenta con tres clases principales, **Data**, **DataPool** y **CommPool**.

- **Data**: está encargada de definir la estructura básica de toda la información que fluye hacia o desde la pila de datos como se detallará en la sección 3.2.1.
- **DataPool**: se encarga de recibir, mantener, entregar y remover la información disponible en el sistema, como se presenta en sección 3.2.2.
- **CommPool**: es una clase reutilizable por los módulos que interactúan con el la pila de datos. Contiene la lógica necesaria para la comunicación entre módulos y pila de datos.

3.2.1. Estructura de datos

La definición de una estructura, aunque estática en los atributos que la componen, pero flexible en cuanto a la forma de la información de algunos atributos, permite la estabilidad y la escalabilidad. Por esto, la importancia subyacente de la clase **Data** donde se define la estructura de la entidad mínima de información de todo el sistema. La tabla 3.1 muestra los atributos que componen la clase **Data**.

Además de los atributos como tal, la clase **Data** cuenta con un método que permite serializar los datos (ver alg. 3.1), para ser transmitidos de forma plana sin importar la estructura interna. Utilizando la notación JSON como formateador los datos son transmitidos y almacenados (ver alg. 3.3). Así pues, cualquier dato puede ser serializado para transmitir invocando el método `getJson()`. Y en sentido inverso se puede crear el objeto a partir de una cadena de texto que contenga un JSON que permita representar la clase **Data**, invocando el método `parse(str)` (ver alg. 3.2)

Listado 3.1: Firma del método `"getJson"` de la clase Data.

```
def getJson(self):
```

Listado 3.2: Firma del método `"parse"` de la clase Data.

Tabla 3.1: Atributos de la estructura de los datos que viajan por el sistema.

Atributo	Descripción
id*	Identificador único del dato. Puede ser alterado al cargar a la pila (ver sección 3.2.2).
born*	Almacena la hora en que el objeto fue creado y, con esto poder controlar el tiempo de vida del dato (ver sección 3.2.2).
source_type	Identifica el tipo de dato almacenado, este puede ser CONTROLLER, CLASSIFIER, ANALYZER.
source_name	Nombre del componente o módulo que genera el dato, por ejemplo: camController, infarctRecognizer, ventAnalyzer (ver secciones 3.4.2, 3.5.2, 3.6.2).
source_item	Fuente primordial de la información, ejemplo: LivingRoom-Cam1.
package	Cuando se recibe un dato desde un dispositivo, este puede ser preprocesado antes de ser enviado a la pila de datos, por ejemplo una imagen puede ser convertida a grises, con lo cual se puede tener más de una versión del mismo dato. El atributo “package” almacena un identificador que permite agrupar varios datos provenientes de un mismo origen.
data	Este atributo contiene la información como tal, por ejemplo, si es una imagen contendrá la matriz de píxeles. No está restringido a una estructura rígida sino que cada componente de entrada podrá organizar los datos a conveniencia.
aux	Permite almacenar información adicional, como la forma en que se organizan los datos o cualquier otro dato a conveniencia.
state*	El estado permite controlar la vigencia del dato: ACTIVE o QUARANTINE.

(*) Indica que el valor de este atributo es autogenerado al momento de crear el dato.

```
def parse(self, str):
```

Listado 3.3: Estructura de datos en JSON.

```

1   {
2     'id'          : self.id,
3     'born'        : self.born,
4     'source_type' : self.source_type,
5     'source_name' : self.source_name,
6     'source_item' : self.source_item,
7     'package'     : self.package,
8     'data'         : self.data,
9     'aux'          : self.aux,
10    }
```

Adicionalmente, la clase **Data** provee tres métodos utilitarios: Algoritmos 3.4, 3.5 y 3.6.

- **valueOf** (alg. 3.4): Permite obtener el valor de un atributo a partir de su nombre.

Listado 3.4: Firma del método "valueOf" de la clase Data.

```
def valueOf(self, key):
```

- **serialize** (alg. 3.5): Permite que cualquier componente pueda, de forma simplificada, serializar los datos, sin importar su forma, para poder enviarlos al servicio de la pila de datos en forma de cadena de texto.

Listado 3.5: Firma del método "serialize" de la clase Data.

```
def serialize(self, data: Any):
```

- **deserialize** (alg. 3.6): Permite que cualquier componente pueda, de forma simplificada, convertir un cadena de texto recibida desde la pila de datos a su estructura original.

Listado 3.6: Firma del método "deserialize" de la clase Data.

```
def deserialize(self, data: str):
```

3.2.2. Transmisión y vigencia de datos

Una vez iniciado el servicio de la pila de datos (**DataPool**) se exponen los métodos (verbos) *post* y *get*, en las rutas “/api/tickets”, “/api/events” , “/api/alerts” y “/api/logs”, para el flujo de información de todo el sistema, como se ve en la figura 3.2. Cada una de las rutas es gestionada por una clase independiente y homónima a la ruta que gestiona.

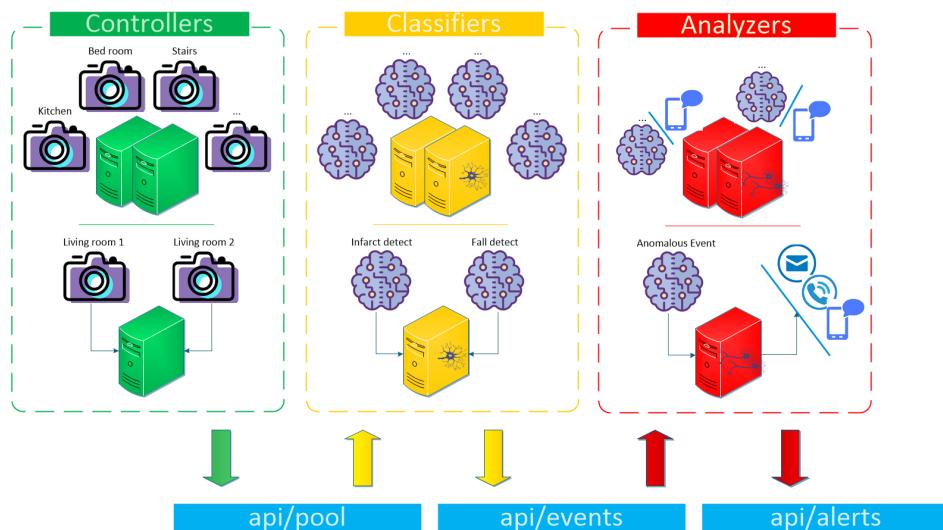


Figura 3.2: Flujo general de información del sistema. De izquierda a derecha se presenta como fluye la información desde su captura hasta la detección de eventos, pasando por las tres rutas definidas para el sistema.

Para unificar y facilitar la comunicación entre los módulos y la pila de datos, en este trabajo se desarrolló la clase **CommData**. Esta clase expone los métodos que se detallaran a continuación, para, además, agilizar el desarrollo de nuevos componentes:

- **send** (alg. 3.7): Permite enviar de forma simple un objeto data a la pila de datos. Siendo *data* el objeto a almacenar en la pila de datos. Es importante mencionar que el objeto recibido podría tener valor solamente en su atributo *data*, ya que los demás atributos, en caso de ir vacíos serán cargados automáticamente con los valores del archivo de configuración del componente.

Listado 3.7: Firma del método "sendData" de la clase **CommPool**.

```
def send(self, data: Data):
```

- **receive** (alg. 3.7): Permite enviar de forma simple un objeto data a la pila de datos. Siendo *data* el objeto con los criterios de filtrado, *limit* la cantidad de datos a retornar (-1 para no considerar este filtro) y *lastTime* una variable de tiempo que indica la ultima vez que se consulto la pila de datos (-1 para no considerar este filtro).

Listado 3.8: Firma del método "receive" de la clase **CommPool**.

```
def receive(self, data:Data, limit=-1, lastTime=-1):
```

- **log** (alg. 3.9): Permite que cualquier componente pueda, de forma simplificada, enviar mensajes a la bitácora del sistema, pudiendo ser estos Errores, Advertencias, Información o cualquiera de la lista de opciones disponible en por el numerador “*LogTypes*”. Siendo *data* un objeto con la misma estructura utilizan los componentes para la comunicación con la pila de datos, que en su atributo *data* contiene el mensaje a almacenar.

Listado 3.9: Firma del método "log".

```
def log(self, data:Data, type:):
```

Independientemente de la clase **CommPool**, la pila de datos como tal expone los servicios de comunicación con los diferentes componentes dos sentidos: **Recibir datos** y **Entregar datos**. El primer caso ocurre cuando los módulos envían información a la pila de datos y el segundo caso es cuando la consultan.

Recibir datos

Para que la pila de datos reciba la información se usa el método *post* de cada una de las rutas. Este método recibirá como parámetro un objeto que cumpla con la estructura de la clase **Data**. En el algoritmo 3.10 se presenta un ejemplo de la estructura que tendría la petición.

Listado 3.10: Ejemplo de parámetros para enviar datos la pila.

```
1 {
2     'package': '1001',
3     'source_type' : 'CONTROLLER', # Replaced on server
4     'source_name' : 'CamController',
5     'source_item' : 'LivingRoom1',
6     'data' : "...", # Data serialized
7     'aux' : '{"t":"image","ext":"png","W":"320","H":"240"}',
8 }
```

Cada vez que el método *post* es invocado, se crea un objeto con la estructura **Data** (ver sección 3.2.1).

Como se mencionó anteriormente, al momento de crear el objeto **Data** se asigna un identificador único dentro del sistema, además se adiciona la hora en que se creo y el estado. Adicionalmente, dependiendo la ruta utilizada, el atributo *source_type* será cambiado por el tipo asociado a cada una así:

- api/tickets =>CONTROLLER
- api/events =>RECOGNIZER
- api/alerts =>ANALYZER

Entregar datos

Todo dato recibido, siempre que éste no haya cumplido su tiempo de vida, estará disponible para ser consultado por cualquier componente. Para ello se expone el método *get* en cada ruta.

Para poder consultar información alojados en la pila de datos, es necesario entregar parámetros al método *get*. En este caso los parámetros serán usados como filtro de la información, por ejemplo si un componente de reconocimiento de eventos simples (ver sección 3.5), desea procesar solo los datos de un dispositivo específico debe indicarlo dentro de los parámetros enviados. La tabla 3.2 presenta los parámetros recibidos por el método y en el fragmento de código 3.11 se muestra un ejemplo.

Tabla 3.2: Parámetros para consultar datos de la pila.

Atributo	Descripción
id	identificador único del dato.
package	Retorna todos los datos que tienen un mismo origen.
source_type	Identifica el tipo de dato almacenado, este puede ser: CONTROLLER para la subruta api/tickets , RECOGNIZER para la subruta api/events y ANALYZER para la subruta api/alerts .
source_name	Nombre del componente o módulo que genera el dato. Puede ser una expresión regular, por ejemplo 'camController' para un módulo específico o 'CamController*' para los datos de cualquier controlador que inicie con 'CamController'.
source_item	Fuente primordial de la información. Puede ser una expresión regular, por ejemplo 'LivingRoom-Cam1' para un dispositivo específico o 'Living*' para los datos de cualquier dispositivo que cuyo nombre inicie con 'Living'.
limit	Limita la cantidad de elementos a retornar
lastTime	Es el tiempo de la ultima consulta, si se envía solo retornará datos creados posterior a ese momento

(*) Todos los atributos son opcionales.

Listado 3.11: Ejemplo de parámetros para consultar la pila.

```

1 {
2   'source_type': 'CONTROLLER',
3   'source_name': 'camController',
4   'source_item': 'LivingRoom-Cam1',
5   'limit'      : 1,
6   'lastTime'   : 1111,
7 }
```

Dado que todos los parámetros sonopcionales, el método puede ser invocado incluso sinparámetros, en tal caso no se aplicará ningún filtro y se retornarán todos los datos vigentes, es decir, que no se hayan etiquetados para eliminar (ver sección 3.2.2).

Para reducir la redundancia de procesamiento y permitir que cada componente funcione de forma asíncrona con los demás, este método además de entregar los datos vigentes y que cumplen con los criterios de filtrado, entrega una variable llamada "*timeQuery*" que contiene el momento (hora en la máquina donde se ejecuta el servicio) en que fue consultado. De esta forma siempre se puede solicitar solo los datos más recientes que la última consulta hecha. Así pues, los datos retornados son una lista de objetos con la estructura definida en la sección 3.2.1, más el "*timeQuery*".

Permanencia y eliminación de datos

La clase **DataPool** se encarga de la permanencia y gestión de los datos. Esta clase mantiene la base de datos con toda la información recibida por el sistema.

La pila de datos tiene una política de permanencia de datos basado en tiempos (ver fig. 3.3). Que mantiene el sistema en estado óptimo de forma constante.

Como se muestra en la figura 3.3, los datos recibidos se apilan y la enmarcan en ventanas de tiempo, para que los módulos los puedan consultar. La primera ventana de tiempo mantendrá los datos apilados por orden de llegada y los irá entregando por demanda, considerando la última vez que un componente consultó la pila.

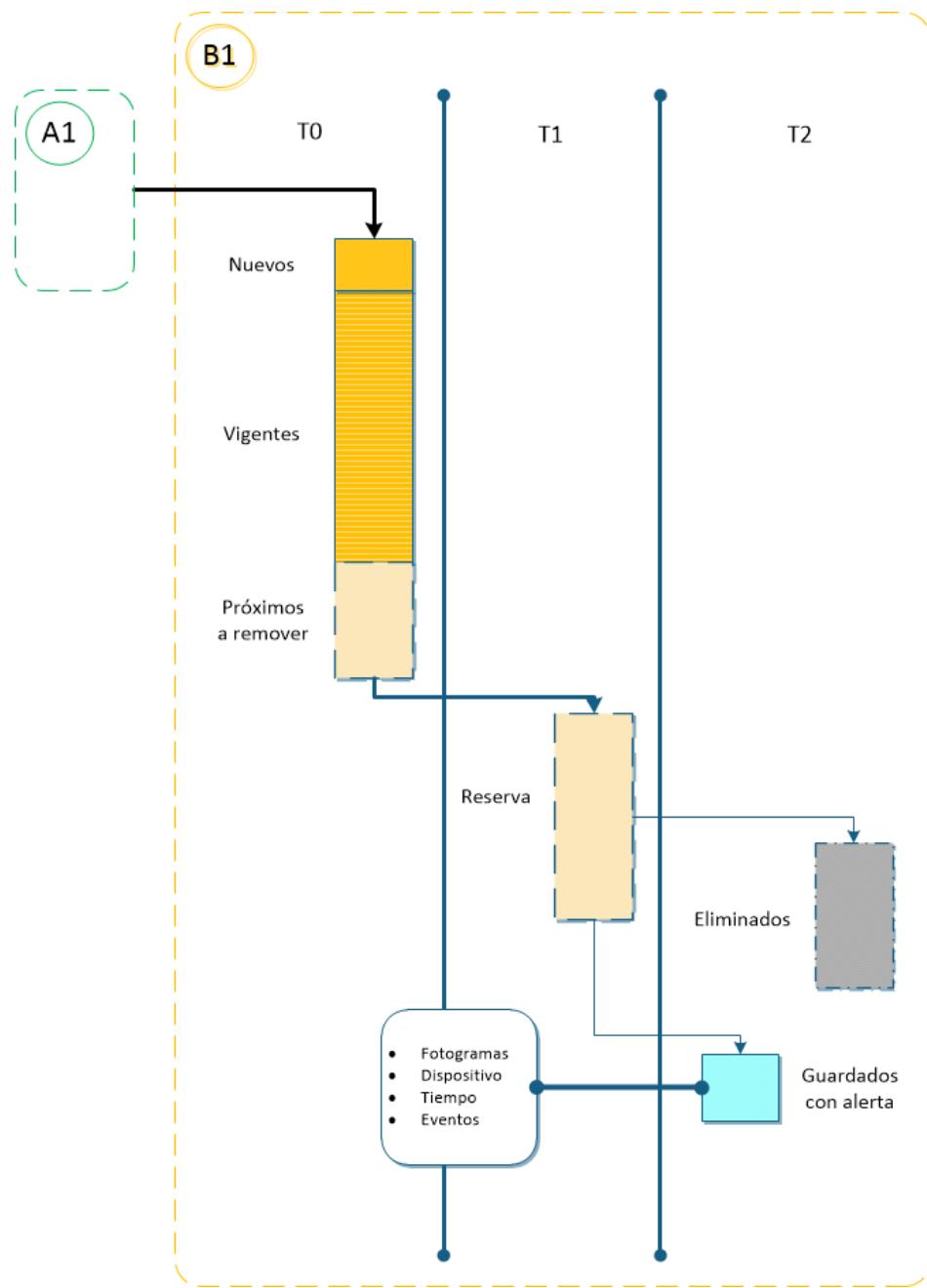


Figura 3.3: Diagrama de la pila de datos. Presenta los diferentes estadios por los que pasa un dato desde su creación hasta su eliminación del sistema.

Una vez cumplida la ventana de tiempo los datos serán removidos de la pila de datos vigentes, pero se mantendrán durante un periodo adicional en reserva, en caso de que una alerta sea reportada por el analizador de eventos, si es así se almacenará de forma permanente con la información complementaria de tiempo, dispositivo y eventos detectados. En caso de no generar ninguna alerta, los datos recibidos serán eliminados definitivamente.

Métodos adicionales de la pila de datos

Dentro de cada una de las subrutinas se adicionó un tercer verbo (**put**), que permite invocar algunos métodos adicionales de la pila de datos (ver patrón comando en 2.3.5). Para invocar este método es suficiente con enviar como parámetro la variable “*command*” con el nombre de la acción a ejecutar. La tabla 3.3 muestra los posibles valores que puede tomar “*command*”.

Tabla 3.3: Métodos adicionales de la pila.

Valor	Descripción
time	Retorna la hora actual del servidor.
isLive	Permite una rápida verificación del estado de la pila.
pop	Permite invocar la purga de los datos de la pila, eliminando los datos que ya no cumplen con la ventana de tiempo de vigencia (ver sección 3.2.2).
count	Entrega el número total de datos almacenados en la pila.

(*) Cualquier valor diferente dará como resultado un mensaje indicando el error.

Además de los presentados, la clase **CommPool**, tiene algunos métodos utilitarios que pueden ser invocados sin tener llamar el servicio directamente . Estos métodos son:

- **sendCommand** (alg. 3.12): Permite que cualquier componente pueda invocar de forma simplificada alguno de los métodos adicionales de la pila de datos (ver tab. 3.3).

Listado 3.12: Firma del método "sendCommand".

```
def sendCommand(self, command):
```

- **getTime** (alg. 3.13): Permite que cualquier componente pueda, de forma simplificada, consultar la hora actual de servidor.

Listado 3.13: Firma del método "getTime".

```
def getTime(self):
```

- **getTimeDiff** (alg. 3.14): Permite que cualquier componente pueda, de forma simplificada, consultar la diferencia (en milisegundos) entre el servidor y la máquina en que se está ejecutando el componente.

Listado 3.14: Firma del método "getTimeDiff".

```
def getTimeDiff(self):
```

- **isLive** (alg. 3.15): Permite que cualquier componente pueda, de forma simplificada, saber la disponibilidad del servicio de la pila de datos.

Listado 3.15: Firma del método "isLive".

```
def isLive(self):
```

- **count** (alg. 3.16): Permite que cualquier componente pueda, de forma simplificada, consultar el número de elementos almacenados en la pila de datos.

Listado 3.16: Firma del método "count".

```
def count(self):
```

3.3. ESTRUCTURA BÁSICA DE COMPONENTES EXTENSIBLES

Como se mencionó anteriormente, en las secciones A, B y C (ver figura 3.1) se ubican los componentes que por su naturaleza pueden ser extendidos en su funcionalidad, permitiendo, incluso, la ejecución de múltiples instancias simultáneamente.

Aun cuando cada uno de los componentes extensibles: Controlador de dispositivos, Clasificador de reconocimiento de actividad humana, Analizador de eventos o canales de notificación, tienen funciones claramente diferenciadas, todos fueron definidos bajo una misma estructura.

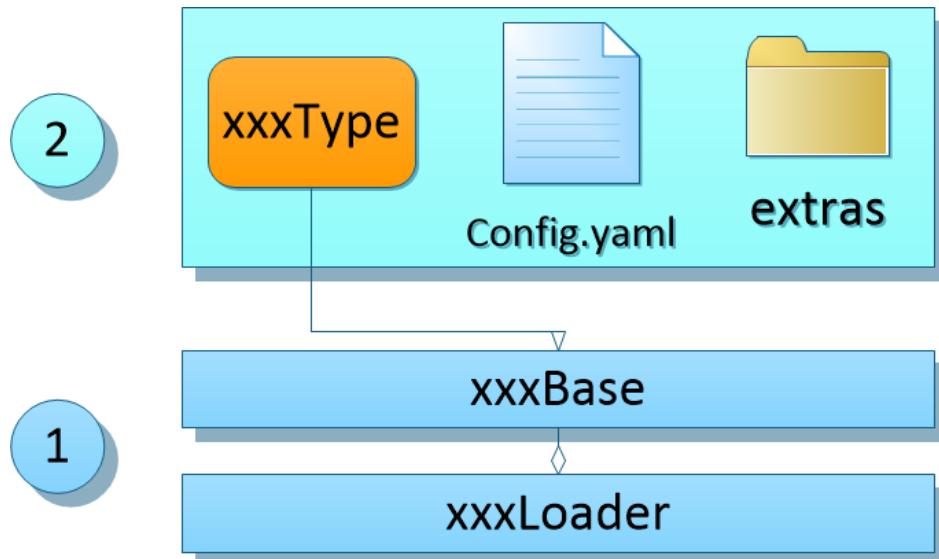


Figura 3.4: Estructura general de un componente extensible. 1) Piezas de código incluidas en el núcleo del sistema. 2) Piezas de software, configuración y adicionales exclusivas de cada componente

La figura 3.4 presenta las partes de un componente extensible, en la parte 1 se presentan dos piezas encargadas de la estructura y comportamiento general de cualquier componente y se encuentran en el núcleo del sistema. En primer lugar el cargador (Loader), encargado de identificar componentes disponibles en el sistema (que están en una ubicación definida, ver la sección de estructura de directorios para mayor claridad). El cargador es, en sí mismo, un servicio que constantemente está supervisando la ejecución de los componentes y en caso de ser necesario reinicia la ejecución del que corresponda.

La segunda pieza del componente incluida en el núcleo del sistema es la clase base. De igual forma que el cargador, existe una clase base para cada tipo componente, cuatro en total. La clase base esta construida como una clase abstracta que define la estructura de un componente y de esta forma generalizar la forma en que el cargador puede iniciar y monitorizar cada una. Además, cuenta con métodos genéricos implementados para la comunicación de componente con la Pila de datos.

En la parte 2 de la figura 3.4, se muestran las piezas específicas del componente, siendo obligatorios la clase tipo, que hereda y extiende la clase base, aportando funcionalidad específica al componente. También se encuentra en esta parte el archivo 'config.yaml' que define las variables necesarias para el correcto funcionamiento de cada componente, como pueden ser tamaños de capturas o cualquier otro necesario. Finalmente, una o varias carpetas no obligatorias, como puede ser 'model' para guardar archivos de entrenamientos previos o cualquier otro recurso que sea necesario para el componente.

Todas las clases tipo heredan de la clase "**Component**" que además provee funcionalidades de uso común, como son:

- **checkConnection** (alg. 3.17): Verifica la conexión del componente con la pila de datos.

Listado 3.17: Firma del método “*checkConnection*”.

```
def checkConnection(self):
```

- ***log*** (alg. 3.18): Permite el reporte de mensajes al registro de atendiaría. Este método recibe un objeto tipo **Data**, con lo cual se obtiene toda la información del componente generador del mensaje. El método también recibe el tipo de mensaje que puede ser de Error, Alerta o Información. Este método además de presentarlo en la forma tradicional de Python, también enviará el mensaje a la Pila de datos de forma que se pueda tener una bitácora centralizada, lo cual se hace útil cuando la ejecución de los diferentes componentes del sistema se hace de forma distribuida.

Listado 3.18: Firma del método “*log*”.

```
def log(self, data, type):
```

Además de los mencionados, cada componente contendrá métodos abstractos que deberán ser implementados por cada uno y que serán comentados más adelante. Pues, aunque todos los componentes tienen la misma estructura, existen particularidades no solo el tipo del componente sino que aun componentes del mismo tipo podrían tener sus necesidades particulares.

3.4. SECCIÓN A - ENTRADA DE DATOS

En esta sección se encuentran las piezas principales de la captura de información. Como se muestra en el diagrama general (fig. 3.1), la información a analizar se recibe por un medio físico. Para el desarrollo de este trabajo sólo se consideraron cámaras RGB, sin embargo, el diseño del sistema permite cualquier otra fuente.

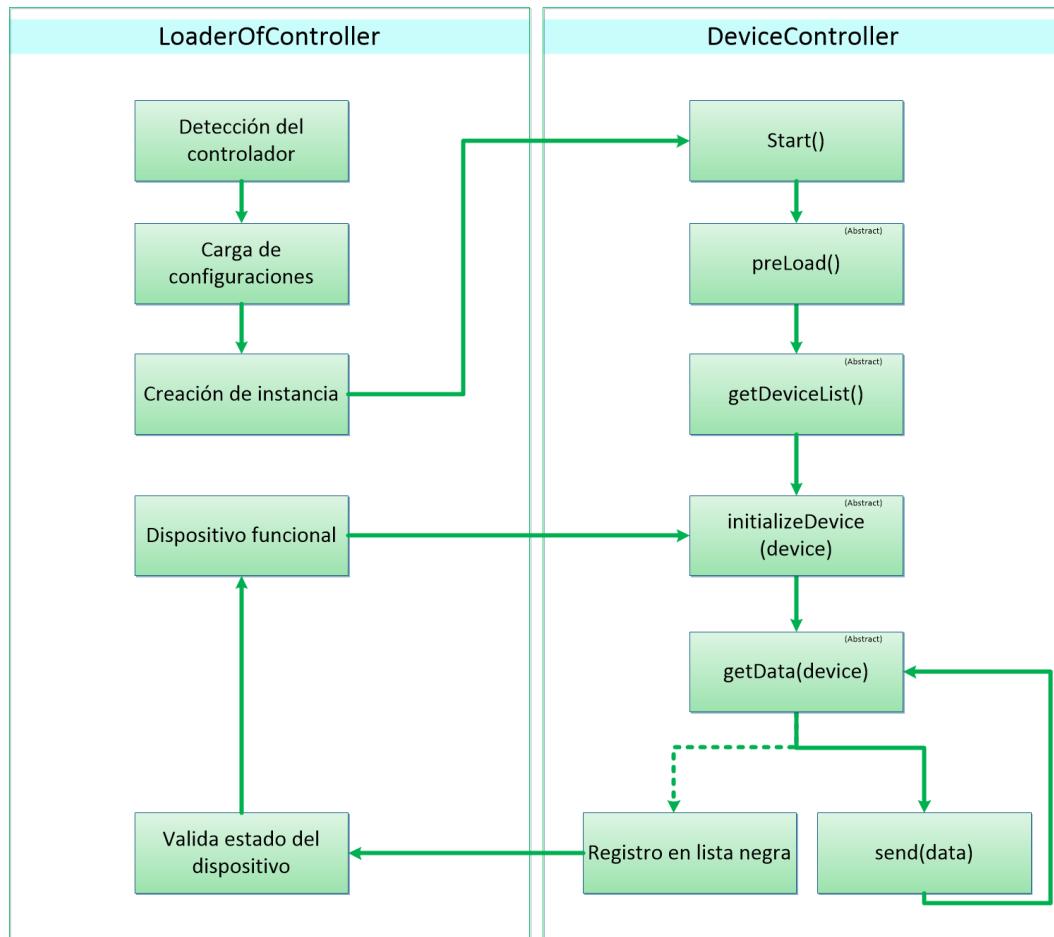


Figura 3.5: Ciclo de vida de la entrada de datos. A la izquierda las tareas realizadas por el cargador de controladores (LoaderOfController). A la derecha las tareas realizadas por el controlador del dispositivo (DeviceController).

Esta sección de sistema consta de dos artefactos principales: El cargador de controladores y los controladores de dispositivo (ver figura 3.5). Mediante la combinación de ambos se captura y organiza la información para que se entregue de forma homogénea a la pila de datos.

3.4.1. cargador de controladores - LoaderOfController

Esta clase se basa en definición del patrón “método factoría” y cumple tres funciones principales, primero identifica que controladores de dispositivos están disponibles, luego carga e inicia la captura de datos y por último se asegura que los controladores se mantengan en funcionamiento.

Detección de controladores de dispositivos

Cuando el cargador de controladores es inicializado recorre la carpeta “Controllers” en búsqueda de archivos con nombre “config.yaml”. De esta forma, toda sub carpeta dentro de “Controllers” que contenga un archivo de configuración, será considerado un posible componente a cargar.

Como se mostró en la sección 3.3 cada componente de entrada debe contar con un archivo “config.yaml”, el cual permitirá al cargador de controladores identificar características como la clase que debe ser inicializada para cargar el componente además de parámetros particulares.

Para tomarlo como un componente válido, el archivo de configuración debe cumplir con los criterios expuestos en la sección 3.3, asegurando además que el estado del mismo esté asignado como “Activo” (Enabled).

Adicionalmente, el archivo “config.yaml” contará con dos variables adicionales: ‘CHECKING_TIME’ y ‘SAMPLING’. La primera permite definir el tiempo de espera para verificar el funcionamiento un dispositivo, en caso de que se haya detectado un fallo. La segunda, permite definir la tasa de muestreo de un dispositivo, por ejemplo 30 cuadros por segundo.

Inicio de controladores

Gracias al uso del patrón de “método plantilla” (ver 2.3.7), el cargador de controladores es capaz de cargar de forma dinámica componentes que hayan sido creados en el momento que se desarrolla este sistema o posteriormente, incluso componentes que se pongan en la carpeta de “Controllers” después de haber iniciado el sistema, siempre que cumpla con las consideraciones que se presentan en la sección de detección de dispositivos.

Una vez cargado el controlador, creado una instancia del mismo, se invoca el método *start* implementado en la clase padre del componente (ver 3.4.2). En adelante, el componente será gestionado por la clase “DeviceController”, salvo que se presente algún error no controlado y éste se detenga, en cuyo caso el cargador de controladores intentará iniciar lo nuevamente.

Manteniendo dispositivos funcionando

La naturaleza del sistema impone que todos sus componentes mantengan un funcionamiento constante incluso con la posibilidad de fallos. Si un dispositivo físico, por ejemplo, una cámara, presenta algún error el cargador de controladores intentará volver a iniciar el dispositivo, sin afectar el funcionamiento de los otros componentes que se ejecutan en el sistema ya que cada controlador de dispositivos se mantiene aislado de los demás.

Para ello, cuando un controlador falla al obtener información de un dispositivo, lo registra en una lista negra. El cargador de controladores, cada 30 segundos recorre el listado de controladores activos (ver patrón iterador 2.3.4) revisando que dispositivos están en lista negra e intentando volver a recibir datos de ellos. En caso de tener éxito el propio Cargador de controladores retira al dispositivo de la lista negra y con ello el controlador volverá a consultar y a enviar datos a la pila.

Adicionalmente, este componente es capaz, en cualquier momento y de forma automática, de detectar cambios en el archivo de configuración, recargando el controlador del dispositivo con sus nuevas configuraciones.

3.4.2. Controlador de dispositivos - DeviceController

El controlador de dispositivos será el encargado de comunicarse directamente con los dispositivos físicos y, por tanto, podrá existir al menos un módulo de este tipo por cada tipo de dispositivo a usar, por ejemplo: un controlador para cámaras RGB, uno para sensores kinects, etc.

Es importante indicar que, debido al diseño del sistema, cada controlador de dispositivos podrá ser ejecutado en máquinas independientes, esto permite que varias instancias del componente ejecutándose al mismo tiempo, permitiendo por ejemplo, varias cámaras capturando imágenes en diferentes ordenadores y enviándolas concurrentemente a la pila de datos. Además, este trabajo cuenta una plantilla (sección 3.8), que se presentará en detalle más adelante, con la cual se podrán

construir otros controladores de dispositivos, de forma que el sistema pueda escalar y pueda tener compatibilidad con nuevos dispositivos o fuentes de datos.

El controlador de dispositivos se basa en la implementación de una clase abstracta (clase padre) que define las reglas para la construcción de un controlador para un dispositivo específico. De esta forma, cualquier controlador de un dispositivo debe heredar de la clase abstracta “DeviceController”, obligándole a implementar algunos métodos específicos, aunque también se cuenta con algunos previamente implementados. Los siguientes métodos serán presentados siguiendo el ciclo de vida de la captura de datos presentada en la figura 3.5.

- **start** (alg. 3.19): Inicia el controlador como tal. Inicia el registro de actividad en los log del sistema. Se encarga de invocar sucesivamente los demás métodos del ciclo de vida de la entrada de datos.

Un componente puede iniciar de dos formas: independiente o conectado, en el primer caso no se enviarán los datos la pila de datos sino que se presentarán directamente mediante el método **showData**. En caso de iniciar conectado, el método **start** comprueba la conectividad con la pila de datos.

Finalmente, este método inicia los mecanismos para estar en constante revisión del funcionamiento del dispositivo y en caso de fallo, pone el dispositivo en cuarentena hasta asegurar el funcionamiento del mismo, de esta forma se aislan los fallos de un dispositivo y que no afecte a los demás.

Listado 3.19: Firma del método “*start*”.

```
def start(self):
```

- **preLoad** (alg. 3.20): (abstracto) Método invocado justo de iniciar la captura de datos. Tiene el objetivo de cargar los recursos para la captura y pre procesamiento de los datos.

Listado 3.20: Firma del método *preLoad* de DeviceController.

```
@abc.abstractmethod
def preLoad(self):
```

- **getDeviceList** (alg. 3.21): (abstracto) Retorna el listado de dispositivos físicos disponibles, además cada dispositivo podrá tener una etiqueta para el reconocimiento, por ejemplo “LivingRoom1”, “Kitchen”, etc. Para ello se lee el archivo de configuración de componente (ver sección 3.8.1).

Listado 3.21: Firma del método “*getDeviceList*”.

```
def getDeviceList(self):
```

- **initializeDevice** (alg. 3.22): (abstracto) Método que activa cada dispositivo físico e inicia la captura datos.

Listado 3.22: Firma del método “*initializeDevice*”.

```
@abc.abstractmethod
def initializeDevice(self, device):
```

- **getData** (alg. 3.23): (abstracto) Este método Retorna los datos capturado del dispositivo físico (*idDevice*).

Este método retorna una lista de objetos de tipo **Data**, pues es posible realizar pre procesamiento a los datos capturados, antes de enviarlos a la pila de datos, con esto es posible obtener de una captura, por ejemplo, la imagen RGB original más una en grises o incluso cambios más sofisticados como la extracción de la persona del fondo de la imagen.

En caso de que a los datos capturados se aplique algún procesamiento adicional, todos contendrán un mismo valor en el atributo “package” de los objetos retornados. Además en el

atributo “source_name” tendrá el nombre del componente y separado por una barra (‘/’), el nombre que identifica el proceso realizado, por ejemplo, “CamController/Gray” para imágenes convertidas a escala de grises. Esto se tendrá en cuenta cuando algún componente consulte datos en la pila de datos.

Listado 3.23: Firma del método “*getData*”.

```
@abc.abstractmethod
def getData(self, device):
```

- **send** (alg. 3.24): Serializa los datos y envía a la pila de datos.

Listado 3.24: Firma del método “*send*”.

```
def send(self, controller, device, data, aux=None, ↵
        ↵ package=''):
```

- **check** (alg. 3.25): (abstracto) En caso de que el dispositivo físico tenga algún problema y se detenga, se intentará comprobar con esta función, que el dispositivo vuelva estar disponible y, en tal caso, volver a cargar datos.

Este método retornara un valor de “Verdadero” o “Falso” indicando si el dispositivo esta funcionando correctamente.

Listado 3.25: Firma del método “*check*”.

```
def check(self, device):
```

Además de los métodos mencionados, la clase “DeviceController” espera la implementación de algunos métodos adicionales.

- **showData** (alg. 3.26): (Abstracto) Este método es útil para presentar los datos capturados por el dispositivo, cuando el componente inicia de forma independiente. Este método será invocado en lugar del método **send**.

Listado 3.26: Firma del método “*showData*”.

```
@abc.abstractmethod
def showData(self, data):
```

- **simulateData** (alg. 3.27): (Abstracto) Este método es útil para simular la captura de los datos capturados por el dispositivo, cuando el componente inicia de forma independiente. Este método será invocado en lugar del método **getData**.

Listado 3.27: Firma del método “*simulateData*”.

```
@abc.abstractmethod
def simulateData(self):
```

- **stop** (alg. 3.28): (Abstracto) Detiene la captura de datos de todos los dispositivos administrados por el controlador.

Listado 3.28: Firma del método “*stop*”.

```
def stop(self):
```

3.5. SECCIÓN B - RECONOCIMIENTO DE EVENTOS SIMPLES

En esta sección se encuentran las piezas principales encargadas de procesar la información capturada por los controladores de dispositivos y realizar el reconocimiento de actividades humanas (HAR - Human Activity Recognition), haciendo uso de técnicas de inteligencia artificial (ver fig. 3.1).

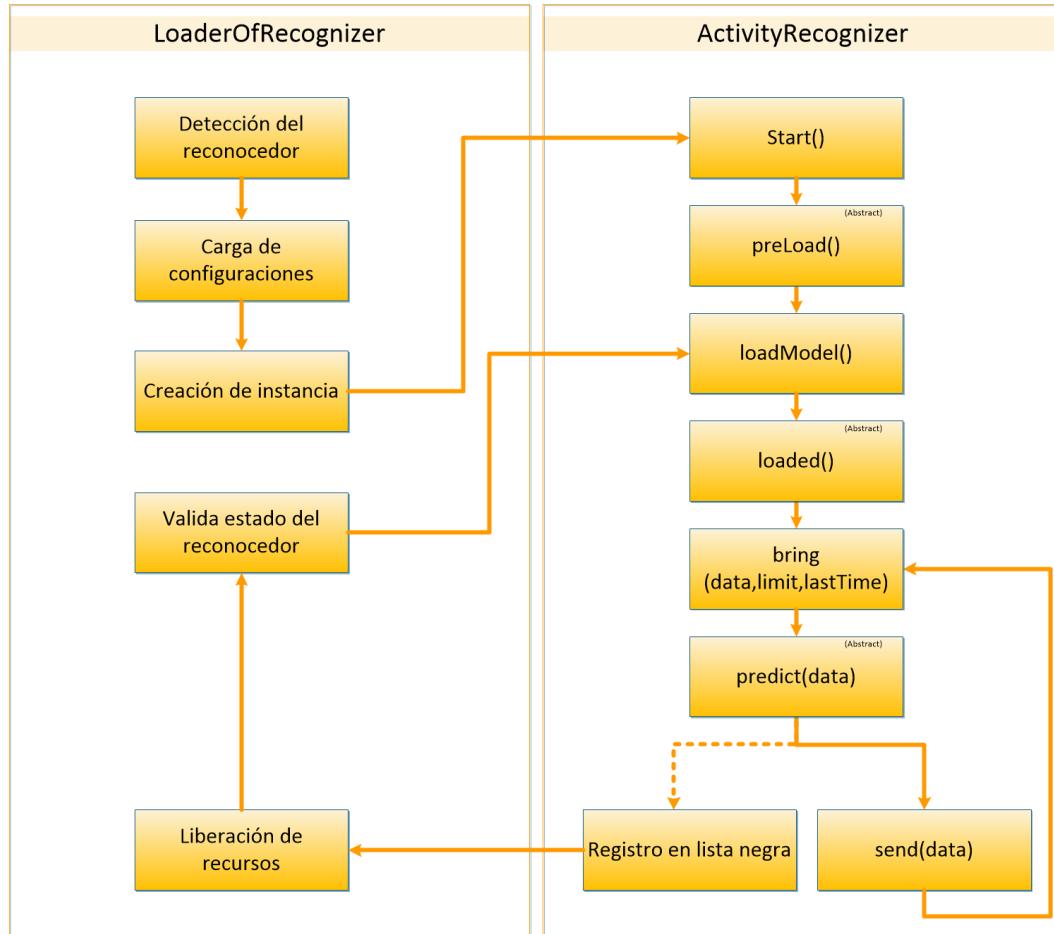


Figura 3.6: Ciclo de vida del proceso de reconocimiento de actividades. A la izquierda las tareas realizadas por el cargador de reconocedores (LoaderOfRecognizer). A la derecha las tareas realizadas por el reconocedor de actividades (ActivityRecognizer).

Esta sección del sistema consta de dos artefactos principales: el cargador de reconocedores y el reconocedor de actividades (ver figura 3.6). Mediante la combinación de ambos se logra la detección de eventos particulares, como puede ser una persona sufriendo un infarto.

3.5.1. Cargador de reconocedores - LoaderOfRecognizer

Esta clase se basa en definición del patrón “método factoría” y cumple tres funciones principales, primero identifica que reconocedores están disponibles para el reconocimiento de actividades, luego carga e inicia cada uno y por último se asegura que los se mantengan en funcionamiento.

Detección de reconocedores

Esta tarea se lleva acabo cuando el cargador de reconocedores es inicializado y recorre la carpeta “Recognizers” en búsqueda de archivos con nombre “config.yaml”. De esta forma, toda sub carpeta dentro de “Recognizers” que contenga un archivo de configuración, será considerado un posible componente a cargar.

Como se mostró en la sección 3.3 cada componente debe contar con un archivo “config.yaml”, el cual permitirá al cargador identificar características como la clase que debe ser inicializada para cargar el componente además de parámetros particulares.

Para tomarlo como un componente válido, el archivo de configuración debe cumplir con los criterios expuestos en la sección 3.3, asegurando además que el estado del mismo esté asignado como “Activo” (Enabled).

Adicional a las variables generales de configuración, los componentes Reconocedores, en el archivo “config.yaml”, cuentan con algunas variables adicionales:

- **CLASSES**: Lista de eventos u objetos que puede reconocer el componente.
- **MODEL**: Ruta al archivo modelo del entrenamiento o conocimiento de la Inteligencia artificial que permitirá la inferencia.
- **FILTER_NAME**: Nombre del controlador del cual se desea traer la información almacenada en la pila de datos
- **FILTER_ITEM**: Nombre del dispositivo que se desea consultar.
- **FILTER_LIMIT**: Número de datos a consultar. -1 en caso de no requerir un límite de datos.

Inicio de reconocedores

Gracias al uso del patrón de ’método plantilla’ (ver 2.3.7), el cargador de reconocedores es capaz de cargar de forma dinámica componentes que hayan sido creados en el momento que se desarrolla este sistema o posteriormente, incluso componentes que se pongan en la carpeta de “Recognizers” después de haber iniciado el sistema, siempre que cumpla con las características que se presentan en la sección de detección de reconocedores.

Una vez cargado el Reconocedor, creado una instancia del mismo, se invocan el método *start* implementado en la clase padre del componente (ver 3.5.2). En adelante, el componente será gestionado por la clase “ActivityRecognizer”, salvo que se presente algún error no controlado y éste se detenga, en cuyo caso el cargador de reconocedores intentará iniciararlo nuevamente.

Manteniendo reconocedores funcionando

La naturaleza del sistema impone que todos sus componentes mantengan un funcionamiento constante incluso con la posibilidad de fallos. Si un reconocedor que detecte, por ejemplo, “infartos” presenta algún error, el cargador intentará volver a iniciar reconocedor, sin afectar el funcionamiento de los otros componentes que se ejecutan en el sistema ya que cada reconocedor se mantiene tiene aislado de los demás.

Para ello, cuando un reconocedor, es registrado en una lista negra. El cargador de reconocedores, cada 30 segundos recorre el listado de reconocedores en lista negra (ver patrón iterador 2.3.4) e intenta volver a iniciararlo. En caso de tener éxito, el reconocedor es retirado de la lista negra y estará listo nuevamente para consultar y a enviar datos a la pila.

Adicionalmente, este componente es capaz, en cualquier momento y de forma automática, de detectar cambios en el archivo de configuración, recargando el reconocedor con sus nuevas configuraciones.

3.5.2. reconocedor de actividades - ActivityRecongnizer

Los componentes reconocedores, son los encargados de realizar la identificación de actividades humanas como tal. Estos componentes consultan y procesan, la información entregada por los dispositivos, para ellos pueden usar una o varias etapas de procesamiento, incluyendo técnicas de inteligencia artificial. Al finalizar el proceso informará, a la pila de datos, las actividades detectadas, además del identificador del dato en que fue detectado.

Cada componente Recognizer adherido al sistema cuenta con el aprendizaje necesario para las predicciones. Aunque este entrenamiento se hará al margen sistema, este trabajo provee tanto una plantilla para la creación de este tipo de componente que será presentado más adelante (sección 3.9), como conjuntos de datos y utilidades para realizar el entrenamiento de inteligencias artificiales, lo cual permitirá extender el sistema mediante la creación de nuevos componentes que identifiquen diferentes eventos o actividades humanas.

Debido al diseño modular del sistema, cada componente reconocedor podrá ser ejecutado en máquinas independientes lo cual, además, se hace recomendable para asegurar la suficiente capacidad de procesamiento, puesto que los algoritmos que contienen estos componentes pueden requerir de procesamiento exhaustivo.

El reconocedor de eventos se basa en la implementación de una clase abstracta (clase padre) que define las reglas para la construcción de un reconocedor de un evento específico. Cualquier reconocedor debe heredar de la clase abstracta “ActivityRecognizer”, obligándole a implementar algunos métodos específicos, aunque también se cuenta con algunos previamente implementados. Los siguientes métodos serán presentados siguiendo el ciclo de vida del proceso de reconocimiento de actividades en la figura 3.6.

- **start** (alg. 3.29): Inicia el módulo que realiza el reconocimiento de actividades con los datos alojados en la pila de datos.

Listado 3.29: Firma del método “*start*” de la clase ActivityRecognizer.

```
def start(self):
```

- **preLoad** (alg. 3.30): (Abstracto) Método invocado antes de iniciar el procesamiento. Tiene el objetivo de permitir cargar los recursos necesarios del componente. También puede cambiar el valor de cualquier parámetro de configuración antes de los procesos automáticos, como puede ser el cambio de la ruta de la base de conocimiento.

Listado 3.30: Firma del método “*preLoad*” de la clase ActivityRecognizer.

```
@abc.abstractmethod
def preLoad(self):
```

- **loadModel** (alg. 3.31): Una de las cualidades de los sistemas de inteligencia artificial es que los entrenamientos o base de conocimiento puede ser almacenada para utilizarla posteriormente. Esta función define el punto de carga de la base de conocimiento. La ruta de este conocimiento será obtenida del archivo de configuración del propio componente.

Listado 3.31: Firma del método “*loadModel*” de la clase ActivityRecognizer.

```
@abc.abstractmethod
def loadModel(self):
```

- **loaded** (alg. 3.32): (Abstracto) Método invocado después de la carga de la base de conocimiento y tiene como fin la preparación o ajuste de los recursos necesarios antes de la predicción. También puede cambiar el valor de cualquier parámetro de configuración antes de la predicción.

Listado 3.32: Firma del método “*loaded*” de la clase ActivityRecognizer.

```
@abc.abstractmethod
def loaded(self):
```

- **bring** (alg. 3.33): Consulta los datos disponibles en la pila. Recibe como parámetro los diferentes valores de filtrado que se adecuen a componente, esto es un objeto “Data”, cuántos datos traer (*'limit'*) y el tiempo base de consulta (*'lastTime'*) para traer los datos más recientes.

Listado 3.33: Firma del método “*bring*” de la clase ActivityRecognizer.

```
def bring(self, dataFilter:Data, limit=-1, lastTime=0):
```

- ***predict*** (alg. 3.34): (Abtracto) Realiza la identificación de un evento en el dato recibido (*'data'*) y envía las clases detectadas y el identificador del dato a la pila.

Listado 3.34: Firma del método “*predict*” de la clase ActivityRecognizer.

```
@abc.abstractmethod
def predict(self, data):
```

- ***send*** (alg. 3.35): Envía los datos de la detección (*'idData'*, *'classes'*) a la pila de datos.

Listado 3.35: Firma del método “*send*” de la clase ActivityRecognizer.

```
def send(self, data:Data):
```

- ***check*** (alg. 3.36): (abstracto) En caso de que reconocedor de actividades tenga algún problema y se detenga, se intentará liberar los recursos utilizados y validar nuevamente que los necesarios están disponibles.

Este método retornara un valor de “Verdadero” o “Falso” inidcando si el componente puede ser reiniciado.

Listado 3.36: Firma del método “*check*” de la clase ActivityRecognizer.

```
def check(self):
```

Además de los métodos mencionados, la clase “ActivityRecognizer” espera la implementación de algunos métodos adicionales.

- ***showData*** (alg. 3.37): (Abstracto) Este método es útil para presentar los resultados del procedimiento e inferencia de eventos, cuando el componente inicia de forma independiente. Este método será invocado en lugar del método ***send***.

Listado 3.37: Firma del método “*showData*” de la clase ActivityRecognizer.

```
@abc.abstractmethod
def showData(self, dataPredicted, dataSource):
```

- ***simulateData*** (alg. 3.38): (Abstracto) Este método es útil para simular la consulta en la pila de datos, cuando el componente inicia de forma independiente. Este método será invocado en lugar del método ***bring***. Esto permitirá no depender de los controladores de dispositivos, además que facilita las pruebas del componente en estado de desarrollo.

Listado 3.38: Firma del método “*simulateData*”.

```
@abc.abstractmethod
def simulateData(self, data:Data, limit:int=-1, ←
    ↪ lastTime:float=-1):
```

- ***stop*** (alg. 3.39): (Abstracto) Detiene la ejecición el componente.

Listado 3.39: Firma del método “*stop*”.

```
def stop(self):
```

3.6. SECCIÓN C - ANALIZADOR DE EVENTOS Y NOTIFICACIÓN DE ALERTAS

En esta sección del sistema se encuentran los componentes encargados de analizar los eventos simples detectados por los diferentes componentes Analizadores de eventos y de notificación. Siendo éstas las piezas finales para emitir notificaciones sobre la anormalidad detectada (ver fig. 3.1).

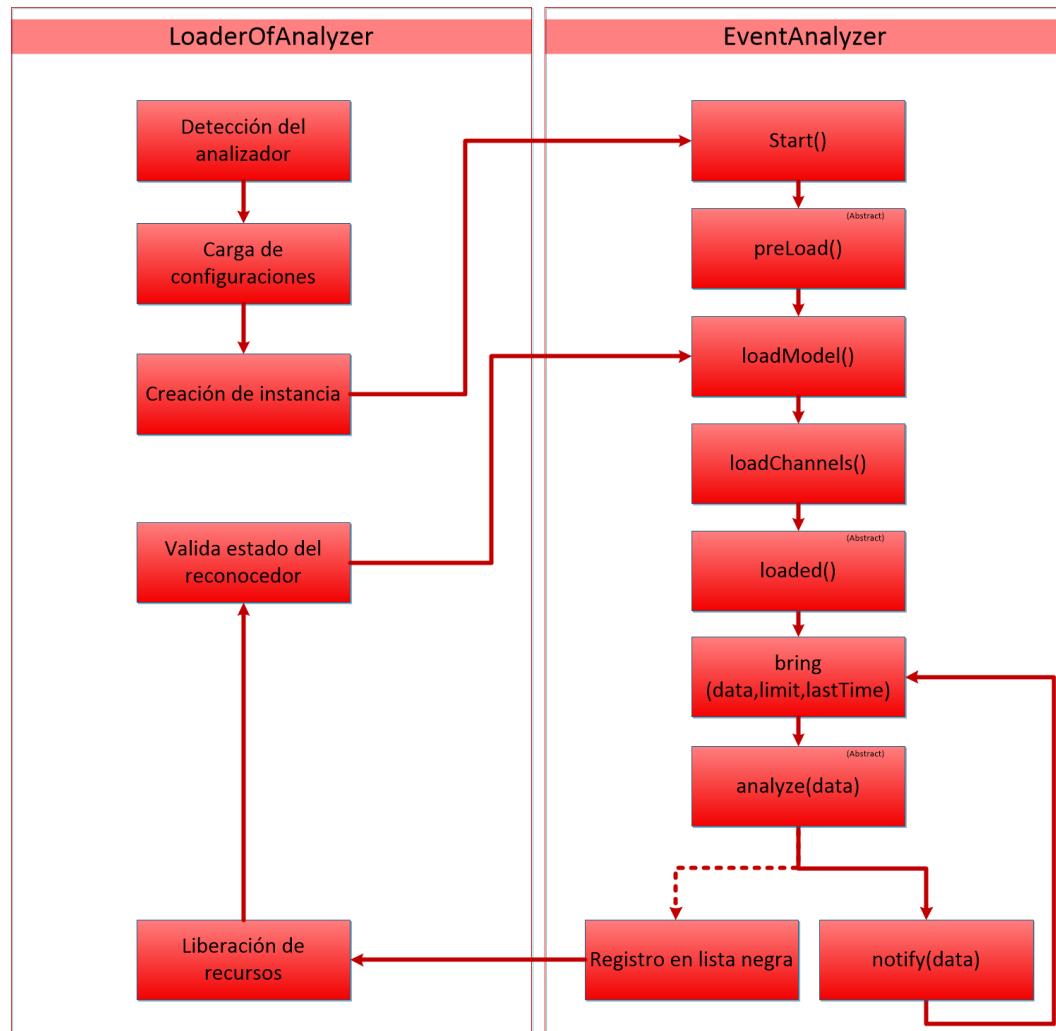


Figura 3.7: Ciclo de vida del análisis de eventos. A la izquierda las tareas realizadas por el cargador de analizadores (LoaderOfAnalyzer). A la derecha las tareas realizadas por el analizador de eventos (EventAnalyzer).

Al igual que los componentes, Controlador de dispositivos y Reconocedor de actividades, El Analizador de Eventos (“Event Analyzer”) puede ejecutarse de forma independiente, es decir que permite su ejecución simultanea en múltiples ordenadores.

Por otro lado, el Canal de notificación (“Channel”) solo está pensado como un medio de comunicación de los eventos que el Analizador requiera informar. De esta forma, un Canal de notificación solo se ejecutara a petición de otro componente. Cuando un analizador de eventos identifique una anomalía, éste iniciará los notificadores disponibles, por tanto, los componentes de notificación deben estar en el mismo ordenador que el analizador (ver figura 3.7).

A continuación serán presentados los artefactos que componen tanto el Analizador de Eventos como el Canal de notificaciones. De igual manera será presentada una plantilla, provista en este mismo trabajo, para la creación de este tipo de componentes (ver secciones 3.10 y 3.11).

3.6.1. Cargador de Analizadores - LoaderAnalyzer

De forma similar a los componentes de las secciones anteriores del sistema, el componente Analizador de Eventos cuenta con dos artefactos que permiten, por un lado, su continuo funcionamiento y por otro la identificación de los eventos anómalos. El primero de ellos es el Cargador de Analizadores.

Este componente cumple tres funciones principales, primero identifica que analizadores están disponibles para la identificación de eventos anómalos, luego carga e inicia cada uno y, por último, se asegura que los componentes iniciados se mantengan en ejecución o de ser necesario, volverlos a iniciar.

Detección de analizadores

Esta acción ocurre cuando el cargador de analizadores es inicializado y recorre la carpeta 'Analyzers' en búsqueda de archivos con nombre 'config.yaml'. De esta forma, toda sub carpeta dentro de "Analyzers" que contenga un archivo de configuración, será considerado un posible componente a cargar,

Como se mostró en la sección 3.3 cada componente debe contar con un archivo 'config.yaml', el cual permitirá al cargador identificar las variables de configuración necesarias para el funcionamiento de los componentes.

Para tomarlo como un componente válido, el archivo de configuración debe cumplir con los criterios expuestos en la sección 3.3, asegurando además que el estado del mismo esté asignado como "Activo" (Enabled).

Adicional a las variables generales de configuración, los componentes Analizadores, en el archivo "config.yaml", cuentan con algunas variables adicionales:

- **CLASSES**: Lista de eventos u objetos que puede reconocer el componente.
- **MODEL**: Ruta al archivo modelo del entrenamiento o conocimiento de la Inteligencia artificial que permitirá la inferencia.
- **FILTER_NAME**: Nombre del Reconocedor del cual se desea traer la información almacenada en la pila de datos
- **FILTER_ITEM**: Sub Nombre del reconocedor que se desea consultar.
- **FILTER_LIMIT**: Número de datos a consultar. -1 en caso de no requerir un límite de datos.

Inicio de analizadores

Gracias al uso del patrón de 'método plantilla' (ver 2.3.7), el cargador de analizadores es capaz de cargar de forma dinámica componentes, hayan sido creados en el momento que se desarrolla este sistema o posteriormente, incluso componentes que se pongan en la carpeta de "Analyzers" después de haber iniciado el sistema, siempre que cumpla con las características que se presentan en la sección de detección de analizadores.

Una vez cargado el Analizador, creado una instancia del mismo, se invoca el método *start* implementado en la clase padre del componente (ver 3.6.2). Posteriormente el método *start* irá cargando los métodos sucesivos del ciclo de vida del análisis de eventos como se muestra en la figura 3.7. En adelante, el componente será gestionado por la clase "ActivityRecognizer", salvo que se presente algún error no controlado y éste se detenga, en cuyo caso el cargador de reconocedores intentará iniciarla nuevamente.

Manteniendo Analizadores funcionando

La naturaleza del sistema impone que todos sus componentes mantengan un funcionamiento constante incluso con la posibilidad de fallos. De esta forma, si un componente de este tipo presenta

algún error no controlado que obligue su detención, el Cargador de Analizadores intentará volver a cargarlo e iniciararlo, sin afectar el funcionamiento de los otros componentes que se ejecutan en el sistema ya que cada componente se mantiene aislado de los demás.

Para ello, cuando un analizador falla y se detiene su ejecución, es registrado en una lista negra. El cargador de analizadores, cada 30 segundos recorre el listado de analizadores en lista negra (ver patrón iterador 2.3.4) e intenta volver a iniciarlos. Para ello se liberan los recursos cargados en memoria, como es el caso de la base de conocimiento, y se verifica que los recursos necesarios están disponibles, luego de esto el analizador es retirado de la lista negra y estará listo nuevamente para consultar analizar los eventos registrados en la pila de datos.

Adicionalmente, este componente es capaz, en cualquier momento y de forma automática, de detectar cambios en el archivo de configuración, recargando el componente con sus nuevas configuraciones.

3.6.2. Analizador de eventos - EventAnalyzer

Los componentes analizadores, son los encargados de realizar la interpretación de los eventos detectados por los Reconocedores de actividades. Estos componentes pueden consultar las etiquetas generadas por los Reconocedores y entregadas por la pila de datos en forma de frases cortas tipo:

- At < **Hora** > the < **Componente** > detect < **Eventos** > in < **Nombre del dispositivo** > of < **Controlador** >.".

Por ejemplo:

- At **14:00** the **InfarctSkeleton** detect "**Infarct**" in **livingRoom1** of **CamController**."

De esta forma, apoyándose en técnicas de procesamiento de lenguaje natura (PLN), es capaz de interpretar las frases consultadas, como si de un narrador se tratara, y generando alertas cuando una anormalidad sea detectada en las frases. Por ejemplo, el sistema reportara una persona tirada en el suelo todos los días a la misma hora, el sistema lo entenderá como un comportamiento habitual, sin embargo, si el sistema nunca reportó este comportamiento podría generar una alerta e invocar a los canales de notificaciones.

El componente analizador de eventos se basa en la implementación de una clase abstracta (clase padre) que define las reglas para la construcción de nuevos analizadores. Cualquier componente analizador debe heredar de la clase abstracta "**EventAnalyzer**", obligándole a implementar algunos métodos específicos, aunque también se cuenta con algunos previamente implementados. Los siguientes métodos serán presentados siguiendo el ciclo de vida del proceso de análisis de actividades mostrado en la figura 3.7.

- **start** (alg. 3.40): Inicia el módulo para empezar a analizar los eventos reportados. Así mismo, este método ejecuta los métodos subsecuentes del ciclo de análisis de eventos.

Listado 3.40: Firma del método "start"de la clase EventAnalyzer.

```
def start(self):
```

- **preLoad** (alg. 3.41): (Abstracto) Método invocado antes de iniciar el procesamiento. Tiene el objetivo de cargar los recursos necesarios para el componente o reasignar los parámetros de configuración previo a la ejecución de análisis o carga de canales de notificación.

Listado 3.41: Firma del método "preLoad"de la clase EventAnalyzer.

```
@abc.abstractmethod
def preLoad(self):
```

- ***loadModel*** (alg. 3.42): Una de las cualidades de los sistemas de inteligencia artificial es que los entrenamientos o base de conocimiento puede ser almacenada para utilizarla posteriormente. Esta función define el punto de carga de la base de conocimiento. La ruta de este conocimiento será obtenida del archivo de configuración del propio componente.

Listado 3.42: Firma del método "*loadModel*" de la clase EventAnalyzer.

```
@abc.abstractmethod
def loadModel(self):
```

- ***loadChannels*** (alg. 3.43): Identifica los componentes notificadores disponibles y los carga para su uso posterior, de esta forma la invocación se hará más rápidamente.

Listado 3.43: Firma del método "*loadChannels*" de la clase EventAnalyzer.

```
def loadChannels(self):
```

- ***loaded*** (alg. 3.44): (Abstracto) Método invocado después de la carga de la base de conocimiento y los canales de notificación. Tiene como fin la preparación o ajuste de los recursos necesarios antes del análisis de los eventos.

Listado 3.44: Firma del método "*loaded*" de la clase EventAnalyzer.

```
@abc.abstractmethod
def loaded(self):
```

- ***bring*** (alg. 3.45): Consulta los eventos reportados a la pila. Recibe como parámetro los diferentes valores de filtrado que se adecuen a componente, esto es un objeto "Data", cuántos datos traer ('*limit*') y el tiempo base de consulta ('*lastTime*') para traer los datos más recientes.

Listado 3.45: Firma del método "*bring*" de la clase EventAnalyzer.

```
def bring(self, data, limit=-1, lastTime=0):
```

- ***analyze*** (alg. 3.46): (Abstracto) Realiza el análisis de los eventos detectados por los reconocedores de actividad. En caso de identificar uno o un conjunto de eventos que deban ser notificados, invocará el método de notificación y emitirá el mensaje por todos los canales dispuestos.

Listado 3.46: Firma del método "*analyze*" de la clase EventAnalyzer.

```
@abc.abstractmethod
def analyze(self, data):
```

- ***notify*** (alg. 3.47): Envía un mensaje a los notificadores para que transmitan el mensaje por los canales disponibles. Los notificadores pueden ser de diferentes tipos, como se presentará más adelante, incluyendo envío de correo, sms, presentación de datos, el informe a la pila de datos de la anormalidad o el resguardo de los datos donde la anormalidad se detectó.

Listado 3.47: Firma del método "*notify*" de la clase EventAnalyzer.

```
def notify(self, data):
```

- ***check*** (alg. 3.48): (abstracto) En caso de que el analizador tenga algún problema y se detenga, se intentará comprobar con esta función, liberar los recursos utilizados y validar que los necesarios, según el archivo de configuración, están disponibles para su uso.
Este método retornara un valor de "Verdadero" o "Falso" indicando el componente puede ser cargado nuevamente.

Listado 3.48: Firma del método "*check*" de la clase ActivityRecognizer.

```
def check(self):
```

Adicional a los métodos presentados, la clase “**EventAnalyzer**”, espera la implementación de algunos métodos adicionales.

- **showData** (alg. 3.49): (Abstracto) Este método es útil para presentar los resultados del procesamiento e inferencia de eventos, cuando el componente inicia de forma independiente. Este método será invocado en lugar del método **notify**.

Listado 3.49: Firma del método “*showData*” de la clase EventAnalyzer.

```
@abc.abstractmethod
def showData(self, dataPredicted, dataSource):
```

- **simulateData** (alg. 3.50): (Abstracto) Este método es útil para simular la consulta en la pila de datos, cuando el componente inicia de forma independiente. Este método será invocado en lugar del método **bring**. Esto permitirá no depender de los reconocedores, además que facilita las pruebas del componente en estado de desarrollo.

Listado 3.50: Firma del método “*simulateData*”.

```
@abc.abstractmethod
def simulateData(self):
```

- **stop** (alg. 3.51): (Abstracto) Detiene la ejecución del componente.

Listado 3.51: Firma del método “*stop*” de la clase EventAnalyzer.

```
def stop(self):
```

3.6.3. Canales de notificación - CommChannel

El segundo tipo componente de la sección C del sistema son los canales de notificación o simplemente notificadores. Estos se encargan de transmitir mensajes con las alertas detectadas por los analizadores, por ejemplo, una persona en el suelo o una persona con infarto.

Cuando un componente Analizador de Eventos es iniciado, éste llamará a su vez al Cargador de Canales de Notificación (“*LoaderOfChannel*”). El cargador de Canales identifica los componentes disponibles para el envío de notificaciones y los mantendrá en memoria para agilizar la ejecución posterior.

La extensión de nuevos canales de notificación, consisten en la implementación de una clase abstracta (clase padre: “*NotificationChannel*”), que permitirá enviar mensajes por diferentes canales, por ejemplo, correo electrónico, SMS, Redes sociales, Apps móvil o cualquier otro canal de comunicación que se deseé.

La arquitectura del sistema permite que, usando una plantilla provista en este trabajo (ver sección 3.11), se puedan crear nuevos módulos de este tipo para, así, dar una mayor escalabilidad al sistema, permitiendo a futuro adicionar nuevos medios de comunicación. Aunque se aclara que para la demostración de este tipo de componentes solo se han creados dos componentes notificadores uno para correo electrónico y otro para notificación directa a una APP móvil.

De forma similar a los demás componentes ya presentados, la notificación también tiene un ciclo de vida propio, como se muestra en la figura 3.8. Dado que cualquier componente notificador debe heredar de la clase abstracta “**NotificationChannel**”, se obliga a implementar algunos métodos específicos, aunque también se cuenta con algunos previamente implementados. Los siguientes métodos serán presentados siguiendo el ciclo de vida de una notificación mostrado en la figura 3.8.

- **start** (alg. 3.52): Inicializa y mantiene en memoria el canal de notificación.

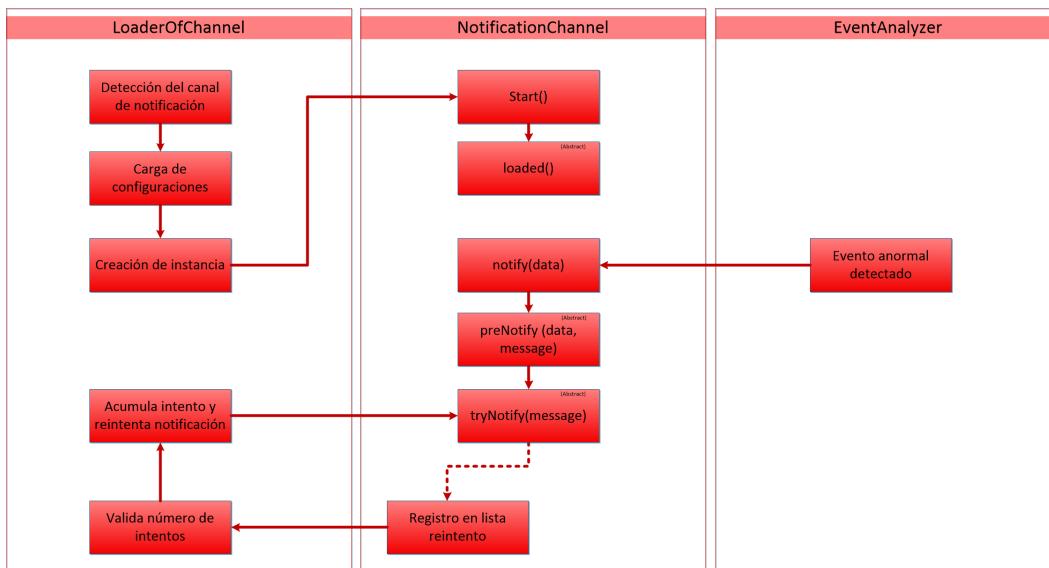


Figura 3.8: Ciclo de vida de una notificación. A la izquierda las tareas realizadas por el cargador de canales de notificación (LoaderOfChannel). Al centro las tareas realizadas por el canal de notificación (NotificationChannel). Al derecho las tareas realizadas por el analizador de eventos (EventAnalyzer).

Listado 3.52: Firma del método "start"de la clase NotificationChannel.

```
def start(self):
```

- **preLoad** (alg. 3.53): (Abstracto) Método invocado una vez cargado en memoria el canal de notificación. Tiene el objetivo de cargar los recursos necesarios para el componente o reasignar los parámetros de configuración previo a la ejecución de las notificaciones.

Listado 3.53: Firma del método "loaded"de la clase NotificationChannel.

```
@abc.abstractmethod
def loaded(self):
```

- **notify** (alg. 3.54): Este método recibe, por parte del componente Analizador de Eventos, la orden de envío de una notificación. Esta función recibirá los datos de la notificación y con esto se construirá un objeto tipo “**Message**” para posteriormente realizar el intento de notificación.

Listado 3.54: Firma del método "notify"de la clase NotificationChannel.

```
def notify(self, data):
```

- **preNotify** (alg. 3.55): (Abstracto) Este método permite la información necesaria para la construcción del mensaje, consultando, de ser necesario, información de la pila de datos. Esta función recibirá tanto los datos de la notificación como el el mensaje construido.

Listado 3.55: Firma del método "preNotify"de la clase NotificationChannel.

```
@abc.abstractmethod
def preNotify(self, data, message):
```

- **tryNotify** (alg. 3.56): (Abstracto) Este método realiza el envío del mensaje. En caso de fallar se agregará a una lista de reintentos que gestiona la clase “**LoaderOfChannel**”

Listado 3.56: Firma del método "tryNotify"de la clase NotificationChannel.

```
@abc.abstractmethod
def tryNotify(self, message):
```

3.6.4. Estructura de mensajes de notificación - Message

La clase **Message** define la estructura del objeto que será usado por los notificadores para construir el mensaje a enviar. La tabla 3.4 muestra los atributos que componen la clase **Data**.

Tabla 3.4: Atributos del mensaje a transmitir por los canales de notificación.

Atributo	Descripción
born	Fecha y hora de generación del mensaje.
from*	Remitente del mensaje.
to*	Lista de destinatarios.
cc*	Lista de destinatarios en copia.
bcc*	Lista de destinatarios en copia oculta o independiente.
subject	Título del mensaje.
message	Cuerpo del mensaje.
aux	Permite almacenar información adicional, como la forma en que se organizan los datos o cualquier otro dato a conveniencia.

ir vacío lo tomara del archivo de configuración del componente.

* En caso de

Variables de reemplazo en mensajes de notificación

Además de los atributos mencionados, es importante resaltar que todos los atributos, excepto '**born**', serán pasados por un reemplazador de tokens, lo cual permitirá el uso de palabras claves para su posterior reemplazo por los valores de contexto. Las palabras clave (tokens) disponibles son:

- **Controllers**: Controladores donde se originaron los datos que dieron lugar a la notificación.
- **Devices**: Dispositivos donde se originaron los datos que dieron lugar a la notificación.
- **Recognizers**: Reconocedores donde se identificaron los eventos que dieron lugar a la notificación.
- **Analyzer**: Analizadores donde se identificaron los eventos que dieron lugar a la notificación.
- **Events**: Eventos identificados por el Reconocedor.
- **DataControllersID**: Identificador de los datos recibidos desde los Controladores.
- **DataRecognizersID**: Identificador de los datos recibidos desde los Reconocedores.
- **DataAnalyzersID**: Identificador de los datos recibidos desde los Analizadores.
- **Phrases**: Frases utilizadas por el Analizador que dieron lugar a la notificación.
- **time**: Fecha y hora de generación del mensaje.

3.7. MODELO INTEGRADO DEL SISTEMA

Una vez presentadas las diferentes piezas del sistema es posible visualizar de forma integrada todos los componentes. Para ello serán presentados tres especificaciones que detallan la disposición e integración de las partes: Estructura de directorios, Modelo de clases y modelo de despliegue.

3.7.1. Estructura de directorios

Así como se definen los diferentes componentes y sus funcionalidades, este trabajo define de forma rígida la ubicación de los componentes. La figura 3.9 presenta el árbol de directorios agrupando los componentes en dos tipos: del núcleo y código base, por un lado, y componentes extensibles por el otro.

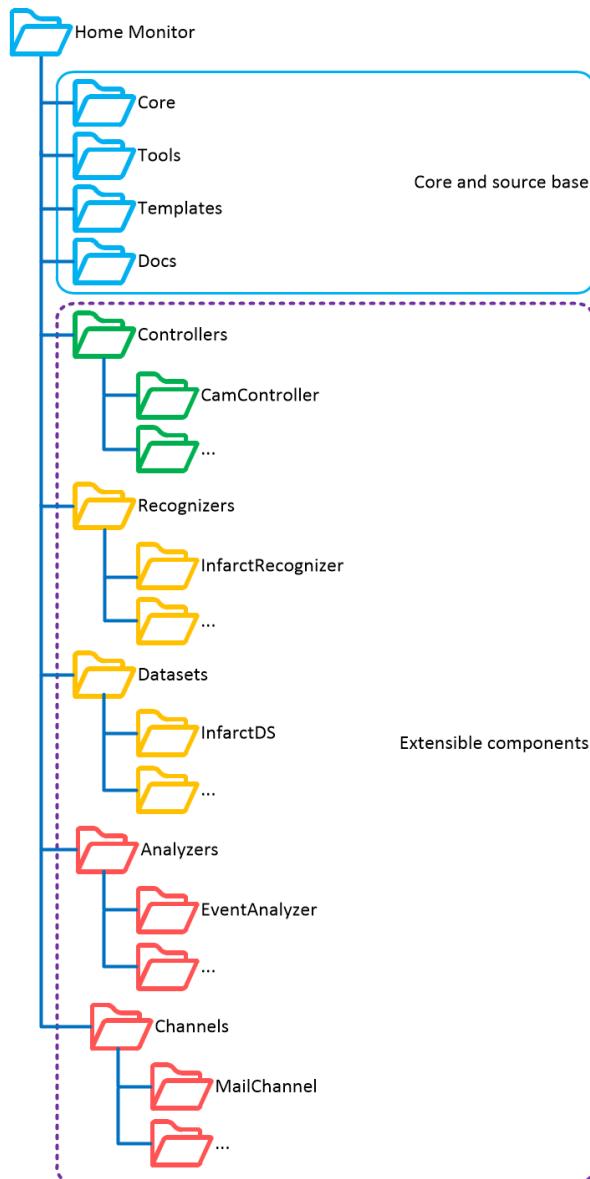


Figura 3.9: Árbol de directorios. Presenta la ubicación en que debe ir cada uno de los componentes del sistema. En azul (arriba) las piezas del núcleo del sistema, en el recuadro morado (abajo) los componentes extensibles con funcionalidades específicas.

Núcleo y código base

Todas las piezas de software que componen el segmento de núcleo y código base se caracterizan por mantenerse rígidos en su definición y programación, asegurando la estabilidad del mismo, manteniendo principalmente un modelo de comunicación inmutable. Aunque el sistema es altamente escalable, esa escalabilidad se sustenta en la estabilidad y definición estática de este segmento del sistema. Los artefacto de este segmento se dividen en cuatro categorías: Directorio de componentes del núcleo (core), Directorio de herramientas (Tools), Directorio de plantillas (Templates), Directorio de documentación (Docs).

Directorio de componentes del núcleo

En este directorio se encuentran las clases más importantes del sistema. Es aquí donde se define la estructura del sistema y los mecanismos de aislamiento y comunicación de todos los demás componentes integrados al sistema.

Listar las clases de la carpeta core.

Para empezar, aquí se encuentran los cargadores de componentes, estos se encargarán de iniciar y mantener funcionando todos los componentes del sistemas. Existe un cargador por cada tipo de componente y trabajan en conjunto con las clases padre de cada tipo de componente, de la cual heredaran cada nuevo componente que se agregue al sistema. Las clases padre, son la base fundamental de la homogeneidad del los componentes, pues aseguran una estructura igual, tanto en la programación como en el intercambio de información ya que proveen los métodos necesarios para el envío y solicitud de datos a la pila de datos.

Además de lo mencionado, en este directorio se encuentra la implementación de la pila de datos, columna vertebral para el envío y transmisión de datos entre componentes.

Finalmente, la clase 'main' que permite la carga del sistema en general.

Directorio de herramientas

En este directorio se encuentran piezas de software con funcionalidades generales, pero también clases enteras que facilitan el proceso de construcción de nuevos componentes o incluso la preparación de conjuntos de datos para entrenamientos. Las funcionalidades serán detalladas más adelante (ver secciones 3.12 y 3.13).

Directorio de plantillas

La flexibilidad para que un sistema sea escalable depende, en gran medida, de la definición de estructuras rígidas sobre las cuales todos los componentes se apoyan. Esto se logra estableciendo, por un lado, los métodos de uso común (start, stop, getData, entre otros. Ver patrón métodos plantilla en 2.3.7), y por otro lado, estableciendo la forma en que viajan los datos. Tanto lo primero como lo segundo se logra, en este proyecto, mediante la disposición de plantillas que permiten a los nuevos componentes seguir los lineamientos definidos, sin aumentar el impacto en el desarrollo o incluso reduciendo considerablemente el tiempo del mismo.

En este directorio se encuentran los artefactos que permiten la construcción de nuevos componentes, acoplables al sistema, desde cero.

Directorio de documentación

En este directorio se encuentran documentos y manuales que permiten un conocimiento del sistema.

Componentes extensibles

Todas las piezas de software que componen este segmento son componentes que extienden las funcionalidades del sistemas. Aquí se encuentran las sub carpetas donde se alojan los Controladores, Identificadores de actividad humana, analizadores de eventos y notificadores (ver fig. 3.1).

3.7.2. Modelo de clases

El sistema general ha seguido patrones de ingeniería de software y se basa en el paradigma de programación orientada a objetos (POO), la figura 3.10 presenta dependencias y relaciones entre las clases que componen el sistema.

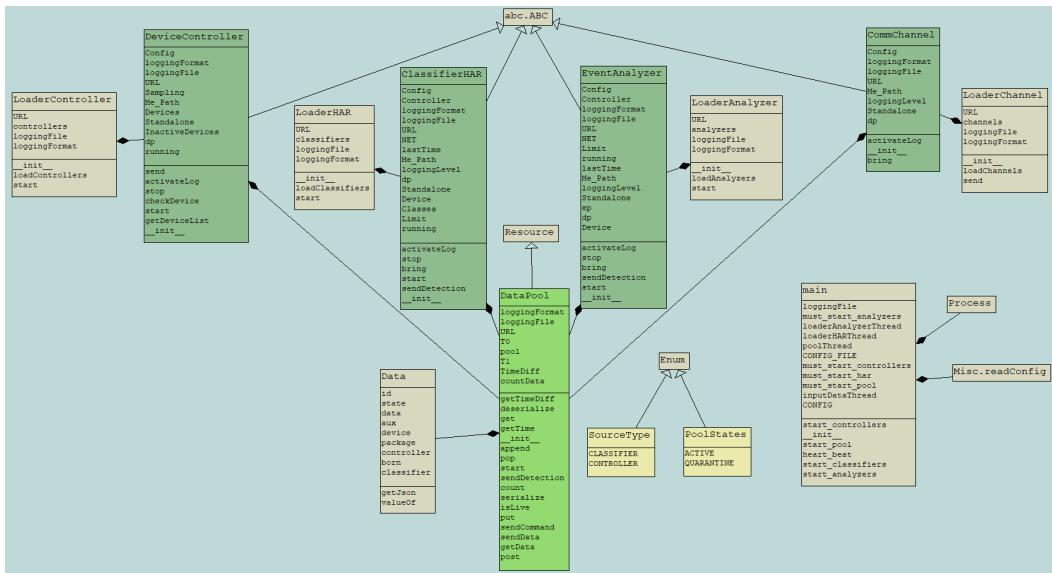


Figura 3.10: Modelo de clases del sistema. Presenta las la relación y dependencias de la s clases del núcleo y del código base.

En el modelo de clases es se evidencia en verde claro la clase 'DataPool' utilizada por todas las clases base (en verde oscuro) para la transmisión y gestión de los datos del sistema. De igual forma en color más claro las clases cargador (Loader) encargadas de mantener los componentes en funcionamiento y la sanidad del sistema.

La clase 'main', al lado derecho funciona como cargador general del sistema y aunque no hereda de otras clases se encarga de instancias mediante el modelo de hilos (procesos independientes) los servicios de carga (loader) en cada máquina que se ejecute.

3.7.3. Modelo de despliegue

Visto la ubicación de los artefactos del sistema y mencionado la alta capacidad del sistema para funcionar de forma desacoplada y distribuida la figura 3.11 presenta la forma en que cada componente puede ser dispuesto en diferentes ordenadores sin que se pierda la integridad del sistema.

Como se muestra en la figura 3.11 cada componente puede ser ejecuto en una máquina independiente, esto significa que es posible usar uno o varios ordenadores para la captura de imágenes, otros para la el procesamiento de los datos capturados. además es posible ejecutar uno o varios analizadores en diferentes máquinas y cada uno con sus canales de notificación independientes.

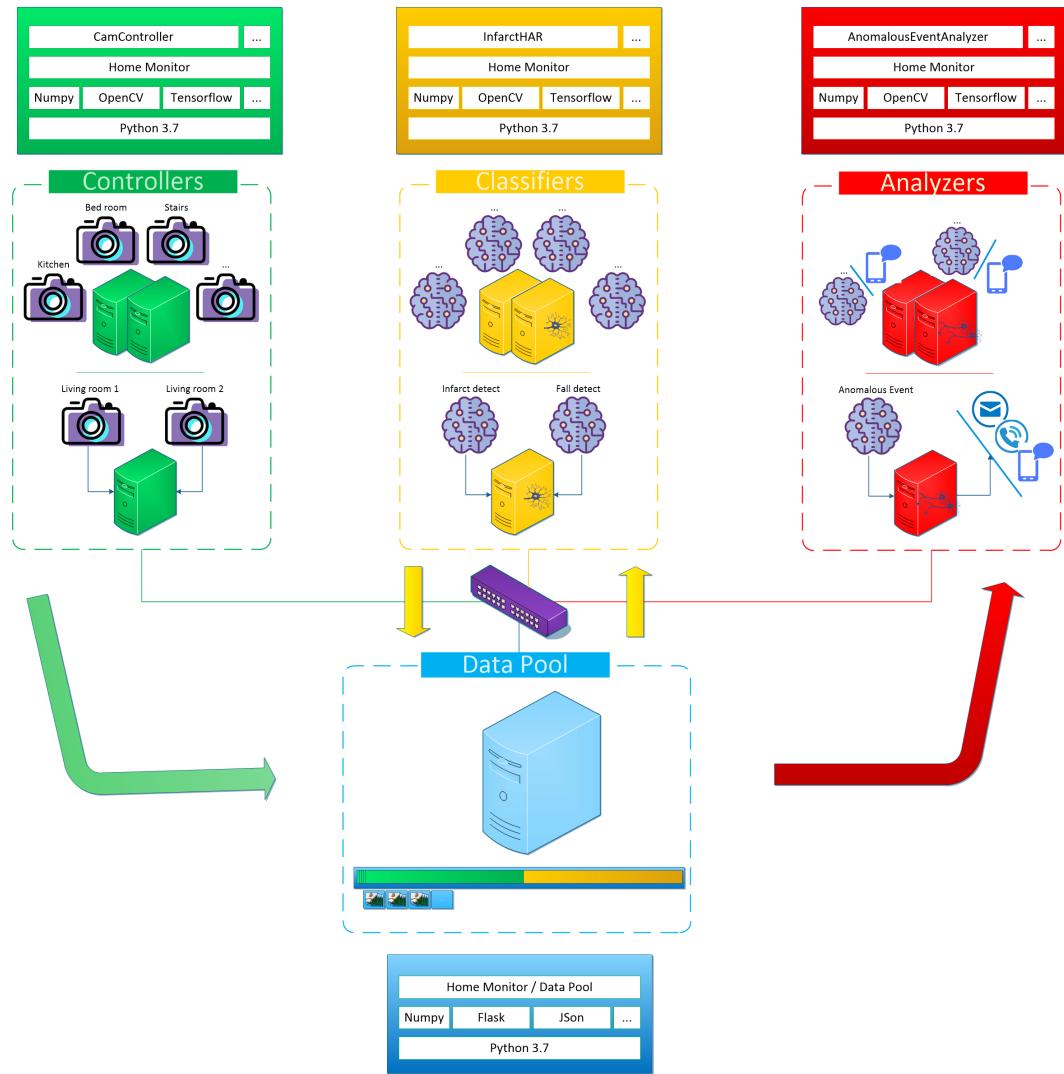


Figura 3.11: Modelo de despliegue del sistema. Presenta las secciones en que se divide el sistema y los componentes que conforman cada sección. También presenta, de arriba hacia abajo, relación que hay entre cada componente y el sistema general para el intercambio de datos. [Cambiar texto recognizers](#)

3.8. MODELO PARA LA CONSTRUCCIÓN DE UN COMPONENTE CONTROLADOR DE DISPOSITIVOS

Para asegurar una forma de construcción estándar de nuevos componentes controladores de dispositivos, en este trabajo se definió y desarrollo una plantilla que permite la implementación rápida de un nuevo componente controlador de dispositivos.

3.8.1. Plantilla Controlador de dispositivos

La plantilla aquí expuesta contiene todos los artefactos necesarios para la creación de nuevos componentes. Cada componente controlador de dispositivos está formado por un directorio que contiene tres archivos como se muestra en la figura 3.12.

Archivo de configuración controlador - config.yaml

Este archivo contiene las variables de configuración y descripción del componente. El sistema usa este archivo para identificar los controladores disponibles y las particularidades de cada uno. La tabla

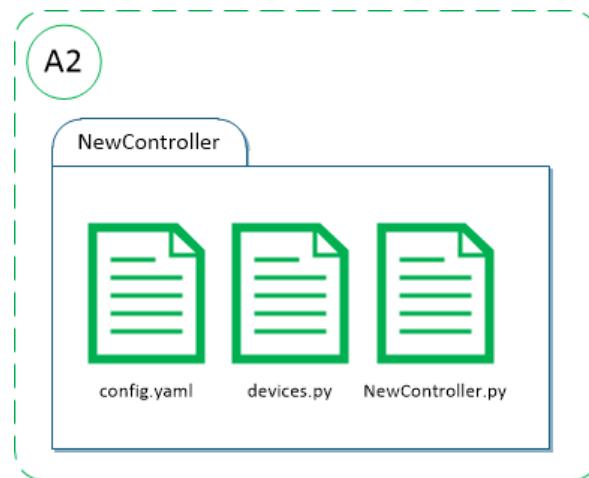


Figura 3.12: Archivos de un componente controlador.

3.5 presenta el listado de atributos disponibles. Es importante mencionar que además de estas cada componente podría adicionar las que le sean necesarias y, de forma autónoma, todas las variables serán cargadas y accesibles una vez el sistema instancie el componente.

Tabla 3.5: Atributos del archivo de configuración controlador.

Atributo	Valor	Descripción
NAME	NewController	Nombre del componente.
TYPE	CONTROLLER	Tipo de componente.
FILE_CLASS	NewController	Nombre del archivo “py” donde se encuentra la clase inicial.
CLASS_NAME	NewController	Clase para cargar el componente (debe heredar de la clase ‘DeviceController’).
VERSION	1.0.0	Versión del componente.
ENABLED	Y	Indica si el componente debe o no ser cargado.
DESCRIPTION	...	Descripción del componente.
SAMPLING	0.0	Tasa de muestreo (en segundos) para la captura de datos del dispositivo.
CHECK_TIME	30	Tasa de verificación (en segundos) de disponibilidad de dispositivos.

Archivo de dispositivos - devices.yaml

Este archivo contiene la descripción de los dispositivos que serán cargados y de los cuales se obtendrán los datos que procesará el sistema.

Listado 3.57: Estructura del archivo de configuración de dispositivos.

```

1  DEVICES: [
2    {
3      'id':0,
4      'name':'Device_0',
5      'enabled':y
6    },
7    {
8      'id':1,
9      'name':'Device_1',

```

```

10     'enabled': n
11 }
12 ]

```

El fragmento de código alg. 3.57 presenta la estructura del archivo. Como se ve muestra cada dispositivo cuenta con tres atributos: **id** que corresponde al identificador interno del dispositivo, **name** donde de almacena el nombre con el cual se reconocerá el dispositivo, este nombre podrá ser usado para filtrar las consultas que se hagan a la pila de datos. Finalmente, el atributo **enabled** indica si el dispositivo debe ser cargado o no. De manera similar, a cada dispositivo se puede asociar las variables que sean necesarias por el componente encargado de su gestión.

Clase inicial del controlador - Controller.py

Esta clase hereda de la clase padre 'DeviceController.py'. Su nombre está compuesto por el nombre del componente seguido de la palabra 'Controller' (sin espacios).

Como se mencionó anteriormente, es en esta clase donde se realiza la captura de datos de los distintos dispositivos y posterior envío a la pila de datos. la plantilla aquí expuesta contiene una copia funcional de este archivo, con todos los métodos que deben ser implementados y los comentarios que sirven de guía al desarrollador de un nuevo componente de este tipo (ver alg. 3.58). Adicionalmente, contiene un fragmento de código que permite que el componente sea ejecutado de forma independiente al sistema.

Listado 3.58: Plantilla para un nuevo controlador de dispositivos.

```

1 """
2 Home-Monitor:
3     AI system for the detection of anomalous and possibly ↪
4         ↪ harmful events for people.
5
6     Written by Gabriel Rojas - 2019 (Change)
7     Copyright (c) 2019 Go S.A.S. (Change)
8     Licensed under the MIT License (see LICENSE for details)
9
10    Class information:
11        Template to get data from a device type NewController.
12 """
13
14 import sys
15 from os.path import dirname, normpath
16
17 import math
18 import logging
19 import numpy as np
20
21 # Including Home Monitor Paths to do visible the modules
22 sys.path.insert(0, './Tools/')
23 sys.path.insert(0, './Core/')
24
25 import Misc
26 from DeviceController import DeviceController
27 from DataPool import Data
28
29 class NewController(DeviceController):
30     """ Template to get data from a device type New. """
31
32     def preLoad(self):
33         """ Load knowledge for pre processing """
34         # TODO: Put here, everything you need to load before you start capturing data
35         pass

```

```

36     def initializeDevice(self, device):
37         """ Initialize device """
38         # TODO: Load a physical device and put in 'capture' var
39         capture = None
40         return capture
41
42     def getData(self, device):
43         """ Returns a list of tuples like {controller, device, ↵
44             → data} with data elements """
45
46         dev = self.Devices[device["id"]][‘capture’]
47         # TODO: Read and return a list of tuples like controller, device, data with data
48             → elements
49         dataReturn = []
50
51         """ Example:
52             _, frame = dev.read()
53
54         if frame is None:
55             return []
56
57         height = np.size(frame, 0)
58         width = np.size(frame, 1)
59         deviceName = Misc.hasKey(device, ‘name’, device[“id”])
60
61         auxData = '{' + ’W’:{}, ’H’:{}’.format(width, height) ↵
62             → + ’}'
63
64         dataReturn.append({
65             ’controller’: ’CamController’,
66             ’device’: deviceName,
67             ’data’: frame,
68             ’aux’: auxData,
69         })
70
71         dataReturn.append({
72             ’controller’: ’CamController/Gray’,
73             ’device’: deviceName,
74             ’data’: self.preProc_Gray(frame),
75             ’aux’: auxData,
76         })
77
78         """
79
80     return dataReturn
81
82
83     def showData(self, data:Data):
84         """ To show data if this module start standalone.
85             set self.Standalone = True before start. """
86         # TODO: Put code if you want test this module in standalone form.
87         pass
88
89     def simulateData(self, device):
90         """ Allows simulate input data """
91         # TODO: Put code if you want test this module in standalone form.
92         dataReturn = []
93         return dataReturn
94
95         """
96         ===== Start standalone =====
97
98     if __name__ == ”_main_”:
99         comp = NewController()
100        comp.init_standalone(Me_Path=dirname(__file__))

```

3.8.2. Implementación de un nuevo controlador de dispositivos

A continuación, se presentan los pasos el desarrollo de un nuevo componente controlador de dispositivos, partiendo de la plantilla provista.

- **Paso 1:** Creación del componente. Para realizar este paso, basta con ejecutar la siguiente instrucción en la carpeta raíz:
 - hm -g controller < Name >
 - Ejemplo: hm -g controller cam

Esto creará una carpeta con el mismo nombre puesto al nuevo componte más la palabra 'Controller', esta nueva carpeta estará ubicada en la carpeta 'Controllers' como se muestra en la figura 3.13.

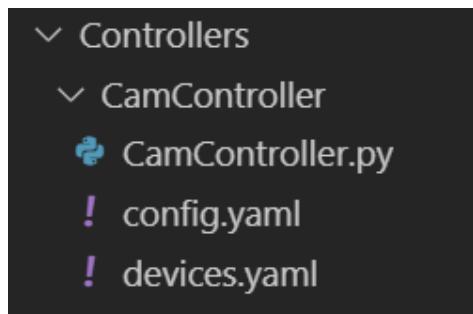


Figura 3.13: Archivos de un nuevo controlador.

- **Paso 2:** Edición de configuraciones. Aunque los archivos de la plantilla están diseñados para que puedan ser funcionales desde el primer momento, será necesario cambiar los valores que se considere necesario para el nuevo componente (ver sección 3.8.1).
- **Paso 3:** El tercer paso consiste en la modificación del archivo **devices.yaml**, incluyendo los dispositivos que se dispongan en el ordenador donde será ejecutado el componente.
- **Paso 4:** Para el último paso, se debe modificar el método '**getData**', incluyendo la lógica que permita obtener los datos de los dispositivos. El fragmento de código alg. 3.59, presenta un ejemplo de la implementación de este método.

Listado 3.59: Implementación del método '**getData**' en un nuevo controlador.

```

1 def getData(self, device):
2     """ Returns a list of tuples like {controller, device, ←
3         ↪ data} with data elements """
4
5     cam = self.Devices[device["id"]][‘objOfCapture’]
6     _, frame = cam.read()
7
8     if frame is None:
9         return []
10
11     height = np.size(frame, 0)
12     width = np.size(frame, 1)
13     deviceName = Misc.hasKey(device, ‘name’, device["id"])
14
15     dataReturn = []
16     auxData = ‘t’: “{}”, “ext”: “{}”, “W”: “{}”, “H”: “{}”,
17
18     if self.getRGB:
19         dataRgb = Data()
20         dataRgb.source_type = self.ME_TYPE
21         dataRgb.source_name = ‘CamController’
22         dataRgb.source_item = deviceName
23         dataRgb.data = frame

```

```
23     dataRgb.aux = '{' + auxData.format('image', 'png', ↪
24         ↪ width, height) + '}'
25     dataReturn.append(dataRgb)
26
27     return dataReturn
```

La variable 'self.Devices' contiene el listado de dispositivos cargados. Es importante mencionar que este trabajo facilita la implementación de nuevos componentes realizando tareas de índole general a todos los componentes, como son la carga de dispositivos, la serialización de los datos capturados y el envío de los datos. Por tal razón dichas funcionalidades no se encuentran dentro de los artefactos generados en el nuevo componente.

- **Paso opcional:** Aunque es opcional, también es recomendable implementar los métodos '**showData**' y '**simulateData**' para poder ver los datos capturados cuando el componente se ejecuta en forma independiente al sistema. Para la ejecución de independiente basta con lanzar el archivo de ejecución del componente (<*Name*>_controller.py) directamente, pues éste ya contiene las instrucciones para ello.

3.9. MODELO PARA LA CONSTRUCCIÓN DE UN COMPONENTE DE RECONOCEDOR DE ACTIVIDADES

Para asegurar una forma de construcción estándar de nuevos componentes reconocedores de actividades, en este trabajo se definió y desarrollo una plantilla que permite la implementación rápida de este tipo de componentes.

3.9.1. Plantilla Reconocedor de actividades

La plantilla aquí expuesta contiene todos los artefactos necesarios para la creación de nuevos componentes. Cada componente Reconocedor de actividades está formado por un directorio que contiene cinco archivos como se muestra en la figura 3.14.

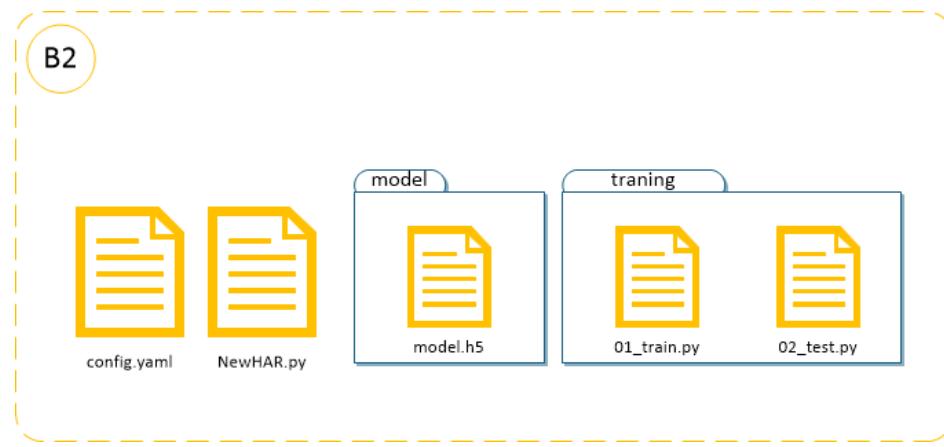


Figura 3.14: Archivos de un reconocedor de actividades.

Archivo de configuración clasificador - config.yaml

Este archivo contiene las variables de configuración y descripción del componente. El sistema usa este archivo para identificar los reconocedores disponibles y sus particularidades. La tabla 3.6 presenta el listado de atributos disponibles.

Clase inicial del clasificador - Recognizer.py

Esta clase hereda de la clase padre 'ActivityRecognizer' y debe implementar los métodos que permitirán la identificación de eventos simples. La plantilla aquí expuesta contiene una copia funcional de este archivo, con todos los métodos que deben ser implementados y los comentarios que sirven de guía al desarrollador de un nuevo componente de este tipo (ver alg. 3.60). Adicionalmente, contiene un fragmento de código que permite que el componente sea ejecutado de forma independiente al sistema.

Listado 3.60: Plantilla para un nuevo Reconocedor de actividades.

```

1  """
2  Home-Monitor:
3      AI system for the detection of anomalous and possibly ↵
4          ↵ harmful events for people.
5
6      Written by Gabriel Rojas - 2019
7      Copyright (c) 2019 G0 S.A.S.
8      Licensed under the MIT License (see LICENSE for details)
9
10 Class information:
11     Template to predict classes of New Rercognizer.

```

Tabla 3.6: Atributos del archivo de configuración clasificador.

Atributo	Valor	Descripción
NAME	NewRecognizer	Nombre del componente.
TYPE	RECOGNIZER	Tipo componente.
FILE_CLASS	NewRecognizer	Nombre del archivo 'py', donde está la clase inicial.
CLASS_NAME	NewRecognizer	Clase para cargar el componente (debe heredar de la clase 'ActivityRecognizer').
VERSION	1.0.0	Versión del componente.
ENABLED	Y	Indica si el componente debe o no ser cargado.
DESCRIPTION	...	Descripción del componente.
CLASSES*	...	Nombre de los eventos detectados.
MODEL	model/model.h5	Ruta al archivo del entrenamiento.
FILTER_NAME**	...	Nombre de un controlador.
FILTER_ITEM**	...	Nombre de un dispositivo.
FILTER_LIMIT**	-1	Cantidad de elementos a consultar cada vez en la pila de datos.

* Permite personalizar el nombre de los eventos que serán detectados por este componente.

** Estas variables son opcionales, pero permiten asignar un filtro predeterminado para las consultas que haga este componente a la pila de datos.

```

11 """
12
13 import sys
14 import numpy as np
15 from os.path import dirname, normpath
16 import logging
17 import json
18 import math
19
20 from cv2 import cv2
21 import tensorflow as tf
22 from tensorflow.keras import backend as K
23 from tensorflow.keras.models import load_model
24
25 # Including Home Monitor Paths to do visible the modules
26 sys.path.insert(0, './Tools/')
27 sys.path.insert(0, './Core/')
28
29 import Misc
30 from ActivityRecognizer import ActivityRecognizer
31 from DataPool import Data
32
33 class NewRecognizer(ActivityRecognizer):
34     """ Template to predict classes of New Recognizer. """
35
36     def preLoad(self):
37         """ Implement me! :: Do anything necessary for ↵
38             ↵ processing """
39         # TODO: Put here, everything you need to load before you start processing
40         pass
41
42     def loadModel(self):
43         """ Loads model """
44         # TODO: Use if you need an special way of loading knowledge
45         """ Example:
46             ModelPath = normpath(dirname(__file__) + "/" + ↵

```

```

    ↪ self.Config[ 'MODEL' ])
46 logging.debug('Loadding model for {} from {} ↪
    ↪ ...'.format(self.__class__.__name__, ModelPath))
47 self.NET = load_model(ModelPath)
48 self.NET._make_predict_function()
49 if logging.getLogger().level < logging.INFO: # Only ↪
    ↪ shows in Debug
50     self.NET.summary()
51 """
52 pass
53
54 def loaded(self):
55     """ Implement me! :: Just after load the model """
56     # TODO: This method is called just after load model
57     pass
58
59 def predict(self, data):
60     """ Implement me! :: Exec prediction to recognize an ↪
        ↪ activity """
61     # TODO: This method must return a list oh classes detected in Input Data and the
       auxiliar information.
62     # Avoid returning class None or background.
63     """ Example:
64     return np.array([ self.Classes[1], ]), 'Aux'
65     """
66     pass
67
68 def showData(self, data:Data):
69     """ Implement me! :: To show data if this module start ↪
        ↪ standalone """
70     # TODO: Put code if you want test this module in standalone form.
71     """ Exemple:
72     print('Classes detected: {}. Aux: {}'.format(classes,
        ↪ aux))
73     """
74     pass
75
76 def simulateData(self, device):
77     """ Implement me! :: Allows to simulate data if this ↪
        ↪ module start standalone """
78     # TODO: Put code if you want test this module in standalone form.
79     """ Example:
80     auxData = '{' + 'W':{}, 'H':{} }.format(1, 1) + '}'
81     return [{ 'timeQuery ':0}, { 'id ':0, 'data ':[], ↪
        ↪ 'aux ':auxData}]
82     """
83     pass
84
85     ===== Start standalone =====
86 if __name__ == "__main__":
87     comp = NewRecognizer()
88     comp.init_standalone(Me_Path=dirname(__file__))

```

Entrenamientos de inteligencia artificial - models

Aunque en la plantilla no se provee un archivo con un entrenamiento, si cuenta con una sub carpeta donde podrán alojarse los archivos relacionados con entrenamientos previos para su futura distribución.

Plantillas para entrenamiento y prueba

Aunque los tres primeros archivos ('config.yaml', '_Recognizer y model), son suficientes para el correcto funcionamiento de un componente de reconocimiento de actividad humana, en este trabajo se incluyó, dentro de los archivos de la plantilla dos más: '01_train.py' (archivo de entrenamiento) y '02_test.py' (archivo de prueba). Estos archivos contemplan el proceso de entrenamiento y prueba de una nueva red neuronal para lo cual bastaría con contar el conjunto de datos de entrenamiento (ver 3.12).

En primera instancia el archivo '01_train.py' contiene una red neuronal, previamente diseñada para la detección de eventos u objetos en imágenes. En este archivo se definen tres variables:

- **INPUT_PATH_TRAIN:** En esta variable se indicará la ruta donde se encuentran las imágenes de entrenamiento. Las imágenes deberán estar agrupadas en sub carpetas, donde se asumirá cada sub carpeta como una clase o categoría.
- **INPUT_PATH_VAL:** Variable similar a la anterior, pero debe contener un conjunto diferente de imágenes, con las cuales se irá validando la precisión del aprendizaje durante el entrenamiento mismo.
- **OUTPUT_DIR:** Ruta a la carpeta donde el modelo o archivo de aprendizaje será almacenado durante proceso de entrenamiento.

Además de las variables mencionadas, el archivo de entrenamiento, contiene algunas más que permiten, entre otras cosas: definir la cantidad de épocas de entrenamiento, el numero de clases a detectar o la tasa de aprendizaje.

El segundo archivo, '02_test.py' contiene la lógica para definir que tan bueno es el aprendizaje obtenido en el entrenamiento, con un conjunto de imágenes no usado anteriormente, durante el entrenamiento o validación.

Para la generalización del uso de este archivo se tienen tres variables principales:

- **INPUT_PATH_TEST:** Ruta a la carpeta con las imágenes de prueba, agrupadas bajo el mismo criterio que las imágenes de entrenamiento.
- **MODEL_PATH:** Ruta al archivo modelo, generado durante el entrenamiento.
- **CLASSES:** Listado de clases a identificar.

3.9.2. Implementación de un nuevo Reconocedor de actividades

A continuación, se presentan los pasos el desarrollo de un nuevo componente Reconocedor de actividades, partiendo de la plantilla provista en este trabajo.

- **Paso 1:** Creación del componente. Para realizar este paso, basta con ejecutar la siguiente instrucción en la carpeta raíz:

- hm -g recognizer <Name>
- Ejemplo: hm -g recognizer *rgbInfarct*

Esto creará una carpeta con el mismo nombre puesto al nuevo componte más la palabra 'Recognizer', esta nueva carpeta estará ubicada en la carpeta 'Recognizers' como se muestra en la figura 3.15.

- **Paso 2:** Edición de configuraciones. Aunque los archivos de la plantilla están diseñados para que puedan ser funcionales desde el primer momento, será necesario cambiar los valores que se considere necesario para el nuevo componente (ver sección 3.9.1).
- **Paso 3:** Entrenamiento. En este paso será necesario seguir los lineamientos para la creación de un conjunto de datos de entrenamiento (ver sección 3.12).

Una vez se disponga de los datos de entrenamiento, dependiendo el tipo de datos, podrá hacerse uso del archivo '01_train.py' para la generación del aprendizaje.

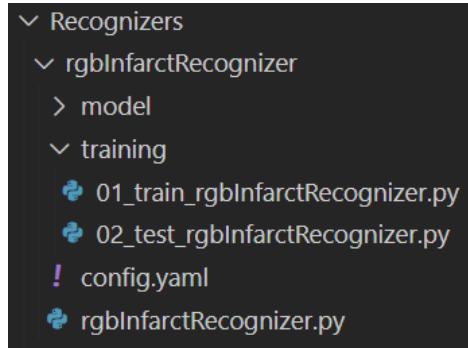


Figura 3.15: Archivos de un nuevo clasificador HAR.

En caso de disponer de un entrenamiento previo, bastará con poner el archivo de entrenamiento en la carpeta 'model'.

- **Paso 4:** Una vez generado el archivo de entrenamiento, se debe modificar el método '**predict**', incluyendo la lógica que permita identificar eventos y retornar las clases correspondientes. El fragmento de código alg. 3.61, presenta un ejemplo de la implementación de este método.

Listado 3.61: Implementación del método 'predict' en un nuevo clasificador.

```

1 def predict(self, data):
2     """ Returns values predicted """
3     array = self.MODEL.predict(data)
4     result = array[0]
5     answer = np.argmax(result)
6     return self.CLASSES[answer]
  
```

La variable 'self.MODEL' contiene la red neuronal con el entrenamiento previo. La variable 'self.CLASSES' contiene la lista de clases que es posible detectar por el modelo, esta lista deberá estar definida en el archivo 'config.yaml' del propio componente. Es importante mencionar que este trabajo facilita la implementación de nuevos componentes realizando tareas de índole general a todos los componentes, como son la carga del entrenamiento de la red y el envío de los eventos detectados a la pila de datos. Por tal razón dichas funcionalidades no se encuentran dentro de los artefactos generados en el nuevo componente.

- **Paso opcional:** Aunque es opcional, también es recomendable implementar los métodos '**showData**' y '**simulateData**' para poder verificar la correcta inferencia de eventos, cuando el componente se ejecuta en forma independiente al sistema. Para la ejecución de independiente basta con lanzar el archivo de ejecución del componente (_Recognizer.py), pues éste ya contiene las instrucciones para ello.

3.10. MODELO PARA LA CONSTRUCCIÓN DE UN COMPONENTE ANALIZADOR DE EVENTOS

En esta sección del documento, se presentará la plantilla que permite la implementación rápida de un nuevo componente para el análisis de eventos.

3.10.1. Plantilla analizador

La plantilla aquí expuesta contiene todos los artefactos necesarios para la creación de nuevos componentes. Cada componente analizador de eventos está formado por un directorio que contiene tres archivos como se muestra en la figura 3.16.

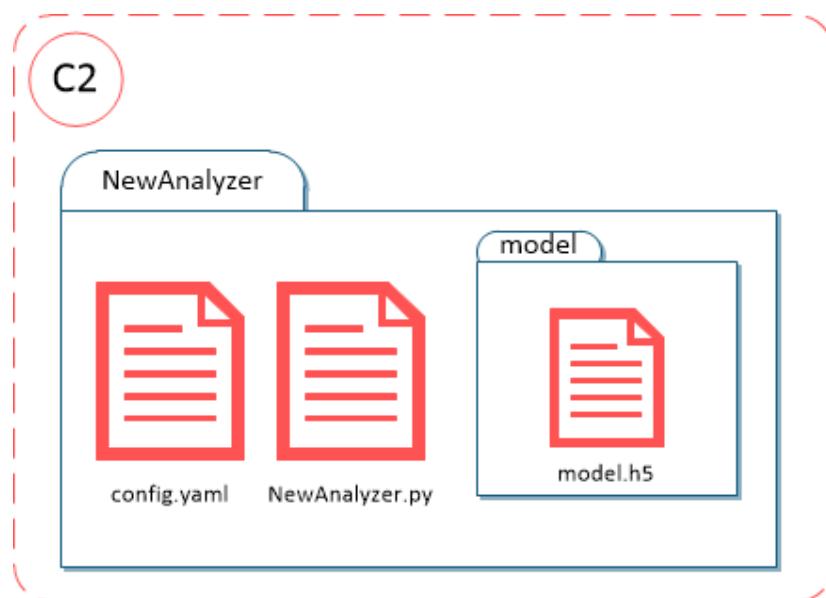


Figura 3.16: Archivos de un analizador de eventos.

Archivo de configuración analizador - config.yaml

Este archivo contiene las variables de configuración y descripción del componente. El sistema usa este archivo para identificar los analizadores disponibles y sus particularidades. La tabla 3.7 presenta el listado de atributos disponibles.

Clase inicial del Analizador - _Analyzer.py

Esta clase hereda de la clase padre 'EventAnalyzer.py'. Su nombre está compuesto por el nombre del componente seguido de la palabra 'Analyzer' (sin espacios).

En esta clase se deben implementar los métodos para analizar los eventos simples detectados por los clasificadores e identificar situaciones anómalas. La plantilla aquí expuesta contiene una copia funcional de este archivo, con todos los métodos que deben ser implementados y los comentarios que sirven de guía al desarrollador de un nuevo componente de este tipo (ver alg. 3.62). Adicionalmente, contiene un fragmento de código que permite que el componente sea ejecutado de forma independiente al sistema.

Listado 3.62: Plantilla para un nuevo Analizador de Eventos.

```
1 """  
2 Home-Monitor:  
3     AI system for the detection of anomalous and possibly ↵  
      ↵ harmful events for people.
```

Tabla 3.7: Atributos del archivo de configuración analizador.

Atributo	Valor	Descripción
NAME	NewAnalyzer	Nombre del componente.
TYPE	ANALYZER	Tipo componente.
FILE_CLASS	NewAnalyzer	Nombre del archivo 'py', donde está la clase inicial.
CLASS_NAME	NewAnalyzer	Clase para cargar el componente (debe heredar de la clase 'EventAnalyzer').
VERSION	1.0.0	Versión del componente.
ENABLED	Y	Indica si el componente debe o no ser cargado.
DESCRIPTION	...	Descripción del componente.
MODEL	model/model.h5	Ruta al archivo del entrenamiento.
FILTER_NAME**	...	Nombre de un reconocedor.
FILTER_ITEM**	...	Sub Nombre de un reconocedor.
FILTER_LIMIT**	-1	Cantidad de elementos a consultar cada vez en la pila de datos.

** Estas variables son opcionales, pero permiten asignar un filtro predeterminado para las consultas que haga este componente a la pila de datos.

```

4      Written by Gabriel Rojas - 2019
5      Copyright (c) 2019 G0 S.A.S.
6      Licensed under the MIT License (see LICENSE for details)
7
8      Class information:
9          Template to analyze classes of recognizers.
10         """
11
12
13     import sys
14     import numpy as np
15     from os.path import dirname, normpath
16     import json
17     import math
18     from time import time, sleep
19
20     from cv2 import cv2
21     import tensorflow as tf
22     from tensorflow.keras import backend as K
23     from tensorflow.keras.models import load_model
24
25     # Including Home Monitor Paths to do visible the modules
26     sys.path.insert(0, './Tools/')
27     sys.path.insert(0, './Core/')
28
29     import Misc
30     from EventAnalyzer import EventAnalyzer
31     from DataPool import LogTypes, SourceTypes, Messages, Data
32
33     class NewAnalyzer(EventAnalyzer):
34         """ Template to analyze classes of recognizers. """
35
36         def preLoad(self):
37             """ Implement me! :: Do anything necessary for ←
38                 → processing """
39             # TODO: Put here, everything you need to load before you start processing
40             pass

```

```

41 def loadModel(self):
42     """ Loads model """
43     # TODO: Use if you need an special way of loading knowledge
44     """ Example:
45         ModelPath = normpath(self.ME_PATH + "/" + ↪
46             ↪ self.CONFIG[ 'MODEL' ])
47         dataC = Data()
48         dataC.source_type = SourceTypes.ANALYZER
49         dataC.source_name = self.ME_NAME
50         dataC.source_item = ''
51         dataC.data = ↪
52             ↪ Messages.controller_loading_model.format(ModelPath)
53         dataC.aux = ''
54         self.COMMPOOL.logFromComponent(dataC, LogTypes.INFO)
55         self.MODEL = load_model(ModelPath)
56         self.MODEL._make_analyze_function()
57         if self.STANDALONE:
58             self.MODEL.summary()
59         """
60
61         pass
62
63 def loaded(self):
64     """ Implement me! :: Just after load the model and ↪
65         ↪ channels """
66     # TODO: This method is called just after load model
67     pass
68
69 def analyze(self, data):
70     """ Implement me! :: Exec analysis of activity """
71     # TODO: This method must return a list oh classes detected in simple events and
72     #       the auxiliar information.
73     """ Example:
74         x = cv2.cvtColor(data.data, cv2.COLOR_BGR2RGB)
75         x = cv2.resize(x, (256, 256), interpolation = ↪
76             ↪ cv2.INTER_AREA)
77         x = np.expand_dims(x, axis=0)
78
79         array = self.MODEL.analyze(x)
80         result = array[0]
81         answer = np.argmax(result)
82
83         dataReturn = []
84         auxData = "t :" "json", "idSource ":"{"
85
86         dataInf = Data()
87         dataInf.source_type = self.ME_TYPE
88         dataInf.source_name = self.ME_NAME
89         dataInf.source_item = ''
90         dataInf.data = self.CLASSES[answer]
91         dataInf.aux = '{' + auxData.format(data.id) + '}',
92         dataReturn.append(dataInf)
93
94
95         return dataReturn
96         """
97
98         pass
99
100 def showData(self, dataanalyzed:Data, dataSource:Data):
101     """ Implement me! :: To show data if this module start ↪
102         ↪ standalone """
103     # TODO: Put code if you want test this module in standalone form.
104     """ Exple:
105         print('Classes detected: {}. Aux: ↪
106             ↪ {}'.format(dataanalyzed.data, dataanalyzed.aux))
107         """

```

```

98     pass
99
100    def simulateData(self, dataFilter:Data, limit:int=-1, ←
101        lastTime:float=-1):
102        """ Implement me! :: Allows to simulate data if this ←
103            → module start standalone """
104        # TODO: Put code if you want test this module in standalone form.
105        """ Example:
106        dataReturn = []
107        dataReturn.insert(0, { 'timeQuery ':time()})
108        return dataReturn
109
110    """
111    """ ===== Start standalone ===== """
112    if __name__ == "__main__":
113        comp = NewAnalyzer()
114        comp.init_standalone(Me_Path=dirname(__file__))

```

Entrenamientos de inteligencia artificial - models

Aunque en la plantilla no se provee un archivo con un entrenamiento, la plantilla si cuenta con una sub carpeta donde podrán alojarse dichos archivos luego del entrenamiento de forma que puedan ser parte integra del modulo para su futura distribución.

3.10.2. Implementación de un nuevo analizador de eventos

A continuación, se presentan los pasos el desarrollo de un nuevo componente analizador de eventos, partiendo de la plantilla provista en este trabajo.

- **Paso 1:** Creación del componente. Para realizar este paso, basta con ejecutar la siguiente instrucción en la carpeta raíz:

- hm -g analyzer Name

Esto creará una carpeta con el mismo nombre puesto al nuevo componente más la palabra 'Analyzer', esta nueva carpeta estará ubicada en la carpeta 'Analyzers' como se muestra en la figura 3.17.

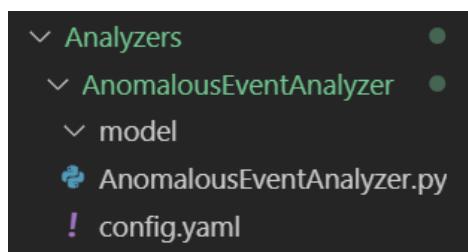


Figura 3.17: Archivos de un nuevo analizador de eventos.

- **Paso 2:** Edición de configuraciones. Aunque los archivos de la plantilla están diseñados para que puedan ser funcionales desde el primer momento, será necesario cambiar los valores que se considere necesario para el nuevo componente (ver sección 3.10.1).
- **Paso 3:** Entrenamiento. Este paso depende enteramente del alcance deseado para el componente, sin embargo, se asume que el resultado es un archivo de entrenamiento. Este entrenamiento debe ser almacenado en la sub carpeta 'model'.
- **Paso 4:** Una vez generado el archivo de entrenamiento, se debe modificar el método 'analyze', incluyendo la lógica que permita identificar situaciones complejas a partir de los eventos

3.10. MODELO PARA LA CONSTRUCCIÓN DE UN COMPONENTE ANALIZADOR DE EVENTOS

simples. El fragmento de código alg. 3.63, presenta un ejemplo de la implementación de este método.

Listado 3.63: Implementación del método 'analyze' en un nuevo analizador.

```
1 def analyze(self, data):
2     """ Detect anomalous events """
3     # TODO: Put logic
```

La variable 'self.MODEL' contiene la red neuronal con el entrenamiento previo. Es importante mencionar que este trabajo facilita la implementación de nuevos componentes realizando tareas de índole general a todos los componentes, como son la carga del entrenamiento de la red y el envío de mensajes a los notificadores. Por tal razón dichas funcionalidades no se encuentran dentro de los artefactos generados en el nuevo componente.

- **Paso opcional:** Aunque es opcional, también es recomendable implementar los métodos '**showData**' y '**simulateData**' para poder verificar el correcto análisis de los eventos, cuando el componente se ejecuta en forma independiente al sistema. Para la ejecución de independiente basta con lanzar el archivo de ejecución del componente (**_Analyzer.py**), pues éste ya contiene las instrucciones para ello.

3.11. MODELO PARA LA CONSTRUCCIÓN DE UN COMPONENTE DE NOTIFICACIÓN

En esta sección del documento, se presentará tanto la plantilla que permite la implementación rápida de un nuevo componente para el envío de los mensajes relacionados con las inferencias hechas por el analizador de eventos, como su implementación.

3.11.1. Plantilla notificador

La plantilla aquí expuesta contiene todos los artefactos necesarios para la creación de nuevos componentes. Cada componente notificador está formado por un directorio que contiene dos archivos como se muestra en la figura 3.18.

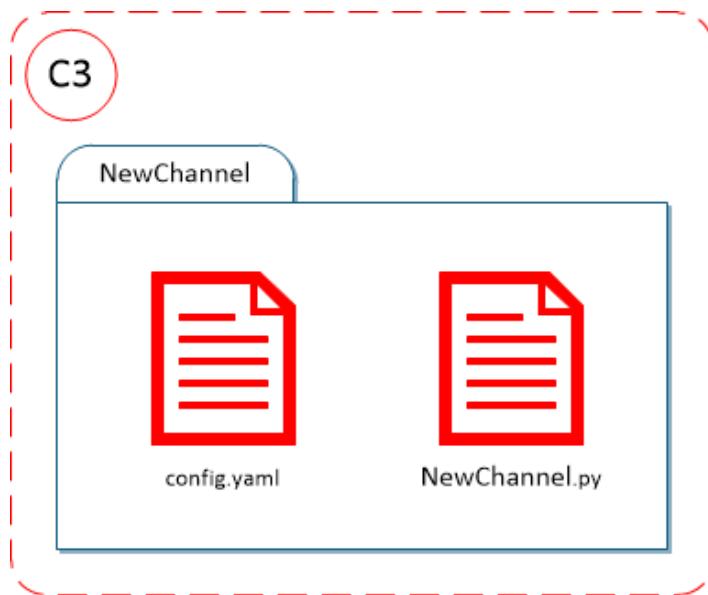


Figura 3.18: Archivos de un notificador.

Archivo de configuración notificador - config.yaml

Este archivo contiene las variables de configuración y descripción del componente. El sistema usa este archivo para identificar los notificadores disponibles y sus particularidades. La tabla 3.8 presenta el listado de atributos disponibles.

Clase inicial del clasificador - _Channel.py

Esta clase hereda de 'CommChannel.py'. Su nombre está compuesto por el nombre del componente seguido de la palabra 'Channel' (sin espacios).

En esta clase se deben implementar los métodos necesarios para el envío de un mensaje por un canal concreto, por ejemplo correo electrónico o SMS. La plantilla aquí expuesta contiene una copia funcional de este archivo, con todos los métodos que deben ser implementados y los comentarios que sirven de guía al desarrollador de un nuevo componente de este tipo (ver alg. 3.62). Adicionalmente, contiene un fragmento de código que permite que el componente sea ejecutado de forma independiente al sistema.

Listado 3.64: Plantilla para un nuevo Canal de notificaciones.

```

1 """
2 Home-Monitor:

```

Tabla 3.8: Atributos del archivo de configuración del componente notificador.

Atributo	Valor	Descripción
NAME	NewChannel	Nombre del componente.
TYPE	ANALYZER	Tipo componente.
FILE_CLASS	NewChannel	Nombre del archivo 'py', donde está la clase inicial.
CLASS_NAME	NewChannel	Clase para cargar el componente (debe heredar de la clase 'CommChannel').
VERSION	1.0.0	Versión del componente.
ENABLED	Y	Indica si el componente debe o no ser cargado.
DESCRIPTION	...	Descripción del componente.
FROM*	...	Usuario o cuenta de origen .
PASSWORD*	...	Clave del usuario que envía el mensaje.
TO*	...	Destinatarios principales.
CC*	...	Destinatarios copiados.
BCC*	...	Destinatarios ocultos.
SUBJECT*	...	Plantilla de asunto, puede usar tokens o palabras clave para remplazar variables con valores entregados por el analizador.
MESSAGE*	...	Plantilla de mensaje, puede usar tokens o palabras clave para remplazar variables con valores entregados por el analizador.

* Estas variables son opcionales y podrían variar según las necesidades del componente.

```

3   AI system for the detection of anomalous and possibly ↵
    ↵ harmful events for people.
4
5   Written by Gabriel Rojas - 2019
6   Copyright (c) 2019 G0 S.A.S.
7   Licensed under the MIT License (see LICENSE for details)
8
9   Class information:
10  Class to send notifications.
11  """
12
13 import sys
14 from os.path import dirname, normpath
15
16 # Including Home Monitor Paths to do visible the modules
17 sys.path.insert(0, './Tools/')
18 sys.path.insert(0, './Core/')
19
20 import Misc
21 from CommChannel import CommChannel, Dispatch
22
23 class NewChannel(CommChannel):
24     """ Class to send notifications. """
25
26     def preLoad(self):
27         """ Load configurations for to send message """
28         pass
29
30     def tryNotify(self, msg:Dispatch):
31         """ Implement me! :: To send the message """
32         # setup the parameters of the message
33         Of = Misc.hasKey(self.CONFIG, "OF", '') if msg.of == '' ↵

```

```

34     ↵ else msg.of
35     To = Misc.hasKey(self.CONFIG, "TO", '') if msg.to == '' ↵
36         ↵ else msg.to
37     Subject = Misc.hasKey(self.CONFIG, "SUBJECT", '') if ↵
38         ↵ msg.subject == '' else msg.subject
39     Message = Misc.hasKey(self.CONFIG, "MESSAGE", '') if ↵
40         ↵ msg.message == '' else msg.message
41
42     # TODO: Put here any method to send messages
43
44     """ ===== Start standalone ===== """
45 if __name__ == "__main__":
46     comp = NewChannel()
47     comp.init_standalone(Me_Path=dirname(__file__), autoload=False)
48     dispatch = Dispatch(of='', to='', subject='', message='')
49     comp.tryNotify(dispatch)

```

3.11.2. Implementación de un nuevo notificador

A continuación, se presentan los pasos el desarrollo de un nuevo componente notificador, partiendo de la plantilla provista en este trabajo.

- **Paso 1:** Creación del componente. Para realizar este paso, basta con ejecutar la siguiente instrucción en la carpeta raíz:

- hm -g channel Name

Esto creará una carpeta con el mismo nombre puesto al nuevo componente más la palabra 'Channel', esta nueva carpeta estará ubicada en la carpeta 'Channels' como se muestra en la figura 3.19.

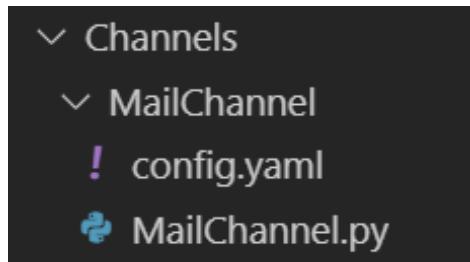


Figura 3.19: Archivos de un nuevo notificador.

- **Paso 2:** Edición de configuraciones. Aunque los archivos de la plantilla están diseñados para que puedan ser funcionales desde el primer momento, será necesario cambiar los valores que se considere necesario para el nuevo componente (ver sección 3.11.1).
- **Paso 3:** El último paso necesario en el desarrollo del componente es la implementación del método 'trySend', incluyendo la lógica que permita el envío de mensajes por un canal determinado. El fragmento de código alg. 3.65, presenta un ejemplo de la implementación de este método.

Listado 3.65: Implementación del método 'send' en un nuevo componente notificador.

```

1 def tryNotify(self, msg:Message)
2     """ Send mail """
3
4     try:
5         # setup the parameters of the message
6         Of = Misc.hasKey(self.CONFIG, "OF", '') if msg.of == '' else msg.of
7         To = Misc.hasKey(self.CONFIG, "TO", '') if msg.to == '' else msg.to

```

```
8     Subject = Misc.hasKey(self.CONFIG, "SUBJECT", '') ↪
9         ↪ if msg.subject == '' else msg.subject
10    Message = Misc.hasKey(self.CONFIG, "MESSAGE", '') ↪
11        ↪ if msg.message == '' else msg.message
12
13    msgMail = MIME_Multipart()
14    msgMail['From'] = From
15    msgMail['To'] = To
16    msgMail['Subject'] = Subject
17    #create server
18    server = smtplib.SMTP('smtp.gmail.com', 587)
19    server.starttls()
20    # Login Credentials for sending the mail
21    server.login(msgMail['From'], self.Config["PASSWORD"])
22    # add in the message body
23    msgMail.attach(MIMEText(Message, 'html'))
24
25    # send the message via the server.
26    server.sendmail(msgMail['From'], msgMail['To'], ↪
27        ↪ msgMail.as_string())
28
29    server.quit()
30 except:
31     logging.exception('Fail to send message. Err:{} ↪
32         ↪ Subject:{} , Message:{} , To:{} ↪
33         ↪ {}.'.format(sys.exc_info()[0], Subject, ↪
34             Message, To))
```

3.12. CREACIÓN DE CONJUNTOS DE DATOS PARA ENTRENAMIENTOS Y PRUEBAS

Dentro del proceso de creación de sistemas basados en estrategias de inteligencia artificial, uno de los puntos más importantes consiste en contar con un adecuado conjunto de datos tanto para el entrenamiento como para la validación del entrenamiento.

Debido a lo anterior y, teniendo en cuenta que este trabajo esta pensado para ser escalado, ampliando sus funcionalidades, algunas de ellas involucrando algoritmos de inteligencia artificial, este trabajo contempla la inclusión de conjuntos de datos variados así como la orientación para la creación de nuevos conjuntos, asegurando la aplicación de buenas prácticas.

3.12.1. Selección de datos

3.12.2. División del conjunto de datos: Entrenamiento, Validación y Pruebas

3.12.3. Aumento de datos

3.13. FUNCIONALIDADES AUXILIARES

Desde su concepción, este proyecto fue pensado como un marco de trabajo que no solo brindará una funcionalidad concreta, sino que también presentará una arquitectura altamente escalable y desacoplable para interconectar los componentes que se integren al sistema, además de un conjunto de herramientas que permitan su extensión. En este capítulo serán presentadas algunas funcionalidades que pueden ser utilizadas tanto en la construcción de nuevos módulos como en la elaboración de conjuntos de datos de entrenamiento, siguiendo los lineamientos expuestos en secciones anteriores.

Siguiendo con las directrices de arquitectura, en el directorio 'Tools' (ver fig. 3.9), de las carpetas del núcleo del sistema, se encuentran las diferentes clases con funcionalidades utilitarias. En este directorio se encuentran: la clase '**Misc**' que contiene las funciones de índole más general. La clases '**DataSplit**' y '**DataAugmented**' que facilitan el proceso de elaboración de conjuntos de datos. Por mencionar algunas, todas serán explicadas a continuación.

3.13.1. Utilidades varias - **Misc.py**

La clase '**Misc**' contiene las funcionalidades más diversas del sistema, de forma que al ser importada en los futuros desarrollos se cuente con la mayor cantidad de funcionalidades. Las funcionalidades disponibles son:

Tabla 3.9: Métodos de la clase **Misc**.

Gestión de archivos	
lsFolders(path=getcwd())	Retorna una lista con las carpetas que están en una ruta.
lsFiles(path=getcwd())	Retorna una lista con los archivos que están en una ruta.
existsFile(fileName, path=getcwd())	Retorna verdadero si el archivo existe en la ruta enviada en caso contrario retorna falso. Si la ruta no se entrega se revisa la carpeta actual.
createFolders(path)	Crea una carpeta en la ruta enviada, si las carpetas padre no existen las crea hasta completar toda la ruta.
saveByType(data, t:str, path:str)	Toma los datos almacenados en el parámetro 'data' y genera un archivo del tipo (t) en la ruta (path) designada. Los tipos posibles son: image, csv. Es útil para el almacenamiento de los datos enviados a la pila de datos.
Gestión de configuraciones	
readConfig(fileName)	Lee un archivo tipo 'yaml' y retorna un objeto con los atributos cargados.
showConfig(config)	Muestra los valores de un objeto de configuración.
importModule(path:str, moduleName:str, className:str=None):	Carga durante la ejecución un módulo, de forma que éste no tenga que existir en tiempo de desarrollo o que tenga que estar en memoria en todo momento.
loggingConf(loggingLevel=logging.INFO, loggingFile=None, loggingFormat=None)	Asigna las configuraciones para el manejo de log del sistema.

(Continúa en la página siguiente)

(Viene de la página anterior)

Aplicación de patrones	
singleton(cls)	Asigna el patrón 'singleton' a una clase, de forma que solo pueda existir una instancia de la misma durante la ejecución del programa.
Varios	
hasKey(dict, key, default)	Si una variable existe en un diccionario retorna su valor, en caso contrario retorna el valor 'default'.
toBool(value:str)	Convierte diferentes texto a tipo 'bool', esto es especialmente útil en la lectura de variables de configuración puede poner cosas como: 'true', '1', 't', 'y', 'yes', 'yeah', 'yup', 'certainly' o 'uh-huh' .
randomString(stringLength=10)	Retorna una cadena de texto aleatorio del tamaño indicado, Útil para la generación de ids.

3.13.2. Segmentación de los datos - DataSplit.py

Esta clase permite la separación de los datos que se van a utilizar para el entrenamiento en tres grupos (Entrenamiento, Validación y Prueba), siguiendo los lineamientos descritos en la sección 3.12.2. Como se mencionó, es recomendable realizar esta tarea antes de un aumento de datos para evitar la mezcla de datos en los subconjuntos.

Para el funcionamiento de esta basta con modificar el valor las siguientes variables:

- **TYPE:** Indica el tipo de dato a segmentar, Puede tomar el valor de 'images' o 'csv'.
- **INPUT_PATH:** Ruta donde están los datos originales.
- **OUTPUT_PATH:** Ruta de destino, aquí se crearan las tres subcarpetas de la segmentación en caso de imágenes o los tres archivos en caso de ser un archivo 'csv'.
- **TRAIN:** Porcentaje del conjunto original asignado al entrenamiento, ejemplo 0.7.
- **VAL:** Porcentaje del conjunto original asignado a la validación, ejemplo 0.15.
- **TEST:** Porcentaje del conjunto original asignado a las pruebas, ejemplo 0.15.

3.13.3. Segmentación de los datos - DataAugmented.py

Esta clase permite el aumentar la cantidad de ejemplo de una muestra a partir de pequeñas modificaciones de cada uno de los datos. Actualmente esta clase solo aplica para imágenes,

Para el uso de esta clase es necesario modificar el valor de las siguientes variables:

- **INPUT_PATH:** Ruta donde están los datos originales.
- **OUTPUT_PATH:** Ruta de destino, aquí se almacenarán los archivos modificados.
- **QUANTITY:** Cantidad de datos nuevos a generar.

CAPÍTULO 4

CONSTRUCCIÓN DEL SISTEMA

En este capítulo serán presentados los subsistemas que fueron desarrollados, siguiendo los lineamientos presentados en el capítulo de arquitectura (publicada como artículo independiente en [Autocita](#)), de forma que se vea la construcción a partir de las diferentes plantillas (3.8, 3.9, 3.10, 3.11). De igual forma que la arquitectura, algunos de los submódulos, por su propia naturaleza e impacto en el campo de reconocimiento de actividad humana, dieron origen a artículos específicos.

Scalable architecture for the integration of multiple AI systems applied to smart environments (Abstract)

Currently, one of the most relevant topics in research is the use of artificial intelligence (AI) algorithms in different areas. Within the wide range of algorithms, the use of artificial neural networks (ANN) stands out. In the ANN there are different designs, with different inputs and outputs, each adjusted to the needs of the problem they solve. However, it is not always easy to integrate the results from one to another, for example, if we have a system that must act according to three questions: are there people in an image?, what command did a person say?, is there enough milk for tomorrow's breakfast?, so the system must be constructed to process the three questions simultaneously, in this way, each question can be answered by a different ANN or even by another AI technique, generating different forms of answers. The system designer could solve this problem in two ways, first, by designing a very large ANN with a very high number of inputs, outputs and hidden layers to receive all possible forms of input and output. This could be easy in this case but, if the number of questions is very large or even unknown from the beginning, this task could be more complicated or non-viable, limiting size of the system. The second option is to keep ANN separate for each question and design a specific subsystem to join all the results, this option is more flexible but could affect performance if the distributed processing is not consider. In this paper we present an architecture that allows many artificial intelligence systems to be integrated, regardless of technique, algorithm, data source or even system output, without affecting performance because it allows highly distributed execution and robust enough to shut down or add subsystems without affecting the others.

Arquitectura escalable para la integración de múltiples sistemas de IA aplicados a entornos inteligentes. (Resumen)

Actualmente, uno de los temas más relevantes en la investigación es el uso de algoritmos de inteligencia artificial (IA) en diferentes áreas. Dentro de la amplia gama de algoritmos, destaca el uso de redes neuronales artificiales (ANN). En el ANN hay diferentes diseños, con diferentes entradas y salidas, cada uno ajustado a las necesidades del problema que resuelven. Sin embargo, no siempre es fácil integrar los resultados de uno a otro, por ejemplo, si tenemos un sistema que debe actuar de acuerdo con tres preguntas: ¿hay personas en una imagen?, ¿qué comando dijo una persona? ¿suficiente leche para

el desayuno de mañana?, entonces el sistema debe ser construido para procesar las tres preguntas simultáneamente, de esta manera, cada pregunta puede ser respondida por un ANN diferente o incluso por otra técnica de IA, generando diferentes formas de respuestas. El diseñador del sistema podría resolver este problema de dos maneras, primero, diseñando un ANN muy grande con un número muy alto de entradas, salidas y capas ocultas para recibir todas las formas posibles de entrada y salida. Esto podría ser fácil en este caso pero, si el número de preguntas es muy grande o incluso desconocido desde el principio, esta tarea podría ser más complicada o no viable, limitando el tamaño del sistema. La segunda opción es mantener a ANN separada para cada pregunta y diseñar un subsistema específico para unir todos los resultados, esta opción es más flexible pero podría afectar el rendimiento si no se considera el procesamiento distribuido. En este artículo presentamos una arquitectura que permite la integración de muchos sistemas de inteligencia artificial, independientemente de la técnica, el algoritmo, la fuente de datos o incluso la salida del sistema, sin afectar el rendimiento porque permite una ejecución altamente distribuida y lo suficientemente robusta para apagar o agregar subsistemas sin afectar los demás.

4.1. MÓDULO PARA INCLUIR LAS CÁMARAS WEB COMO UN TIPO SENSOR DEL SISTEMA

Parte fundamental de cualquier sistema es la entrada de datos, para este sistema se ha creado un componente extensible llamado “DeviceController” (Ver sección 3.4.2). Este componente contiene los métodos y atributos necesarios para transformar los datos capturados de un dispositivo específico y enviarlos a la pila de datos. Sin embargo, para la utilización de los distintos dispositivos físicos de que se dispongan, será necesario la creación de una clase específica que extienda a “DeviceController”, como se ve en la figura 4.1, y contiene la lógica que permite leer el dispositivo.

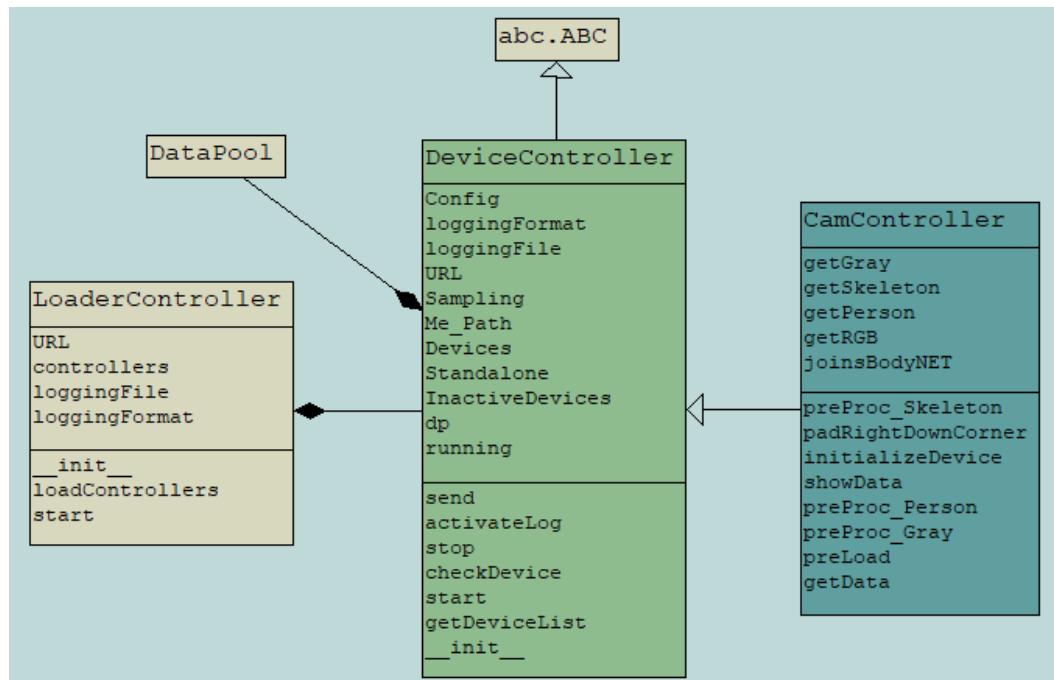


Figura 4.1: Modelo de clases de CamController. Presenta las la relación y dependencias de las clases del controlador de cámaras.

Siguiendo las instrucciones descritas en la sección 3.8, para este proyecto se estableció la captura de imágenes por medio de cámaras RGB tradicionales, por ser éstas un dispositivo de fácil adquisición y bajo coste, lo que permite la utilización de varias cámaras en un hogar, objeto fundamental de este trabajo. Sin embargo, se reitera que el diseño de este sistema no se limita a un tipo de dispositivo.

El controlador de cámaras consta principalmente de cuatro archivos:

- config.yaml
- devices.yaml
- CamController.py
- model/poseModel.h5

El archivo **config.yaml** además de las variables mencionadas en la sección 3.8.1 contiene algunas de uso específico en este controlador (ver tabla 4.1).

El archivo **devices.yaml** contiene el listado de cámaras que serán utilizadas, pudiendo identificar cada una con una etiqueta. El fragmento de código alg. 4.1 presenta la estructura del archivo. En este caso las configuraciones se aplicaran a cada cámara por separado, siendo posible activar o desactivar una cámara o cambiar el ancho y alto de la imagen capturada.

Listado 4.1: Cámaras disponibles.

12.1. MÓDULO PARA INCLUIR LAS CÁMARAS WEB COMO UN TIPO SENSOR DEL SISTEMA

Tabla 4.1: Atributos del archivo de configuración de CamController.

Atributo	Valor	Descripción
FRAME_WIDTH	320	Ancho de la imagen capturada.
FRAME_HEIGHT	240	Alto de la imagen capturada.
FORMATS	[RGB, Gray, Objects]	transformaciones que se aplicaran a los datos capturados. Pueden ser combinados para retornar uno o varios. los valores disponibles son: RGB, Gray, Person, Skeleton.
OBJECTS	[Person,]	Listado de 80 posibles objetos detectables. Esto se apoya en el trabajo Mask-RCNN (He et al. en [He2017]).

```

1 DEVICES: [
2   {
3     'id':0,
4     'name':'Stairs',
5     'enabled':y,
6     'width':320,
7     'height':240
8   },
9   {
10    'id':1,
11    'name':'LivingRoom1',
12    'enabled':n,
13    'width':320,
14    'height':240
15  }
16 ]

```

El archivo **CamController.py** contiene toda la lógica que permite funcionar al componente. Dentro de este archivo se encuentra una clase que lleva el mismo nombre y hereda de “Device-Controller”. Esta clase implementa, en primera instancia, los métodos abstractos de la clase padre. Además, contiene los métodos que permiten hacer las transformaciones a los datos originales como muestra la tabla 4.2.

Tabla 4.2: Atributos del archivo de configuración de CamController.

Método	Descripción
preProc_Gray	Convierte la imagen a escala de grises cuando la formula $Y = 0.299 * R + 0.587 * G + 0.114 * B$.
preProc_Object	Procesa la imagen capturada y retorna una lista de imágenes que contiene de forma separado cada objeto detectado, extraído del fondo.
preProc_Skeleton	Procesa la imagen capturada y retorna una lista de puntos con las coordenadas de las articulaciones de las personas en la imagen.
"__main__"	Permite la ejecución del componente de forma independiente al sistema, de esta forma poder visualizar directamente los datos que se están capturando y las transformaciones realizadas.

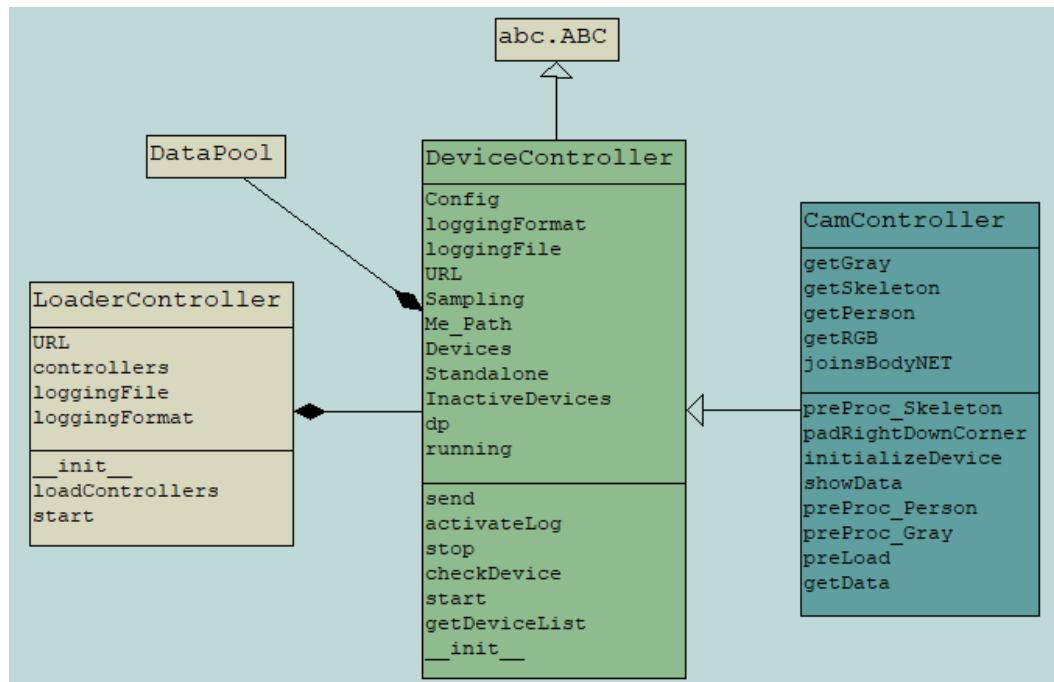


Figura 4.2: Captura y transformaciones del CamController. a) RGB, b) Grises, c) Persona extraída, d) Esqueleto. Cambio de imagen, RGB Gray, PErson y Skeleton

4.1.1. Extracción de la persona

Para localizar y extraer las personas registradas en las imágenes se utilizó el trabajo presentado por He et al. en [He2017], ya que éste permite la detección de las personas y retorna la máscara o conjunto de píxeles que la componen, simplificando el proceso de extraer y recortar la imagen, para así retornar una nube de puntos por cada persona detectada. Adicionalmente, el trabajo de He permite la extracción de 80 objetos adicionales, lo cual podría permitir a futuro la ampliación de este componente.

4.1.2. Extracción del esqueleto

La esqueletización consiste en la extracción de los puntos del cuerpo de una persona que permiten reconstruir una versión simplificada de la postura de la persona. Como se mencionó en la sección 2.6.3 existen diferentes estrategias para lograrlo. En este proceso se basa en el trabajo de Cao et al. [Cao2017], que usa el concepto de mapas de calor para estimar zonas candidatas donde es posible encontrar una articulación de una persona, logrando generar hasta un total de 18 puntos por cada persona localizada.

4.2. CONSTRUCCIÓN DE MÓDULO PARA EL RECONOCIMIENTO DE INFARTOS

Esta sección describe la utilización de la plantilla diseñada para la creación de módulos para el reconocimiento de actividades humanas (HAR). Para ello se definió un evento específico que puede ocurrir a cualquier persona. Las enfermedades cardiovasculares, según estudios de la Organización Mundial de la Salud (OMS), una de las principales causas de muerte en el mundo [OMS2018]. Adicionalmente, cuando una persona sufre un ataque al corazón, experimenta un fuerte dolor en el pecho [Patel2018, Goff1998], que puede conducir a la pérdida de la conciencia y de encontrarse solo o sola no recibir ayuda oportuna.

4.2.1. Detección de ataque cardíaco en imágenes en color utilizando redes neuronales convolucionales

El desarrollo de este módulo dio como resultado la publicación 'Heart Attack Detection in Colour Images Using Convolutional Neural Networks' [rojas2019heart]. El código asociado a este artículo puede ser descargado de forma independiente de <https://github.com/Turing-IA-IHC/Heart-Attack-Detection-In-Images>.

Heart Attack Detection in Colour Images Using Convolutional Neural Networks (Abstract)

Cardiovascular diseases are the leading cause of death worldwide. Therefore, getting help in time makes the difference between life and death. In many cases, help is not obtained in time when a person is alone and suffers a heart attack. This is mainly due to the fact that pain prevents him/her from asking for help. This article presents a novel proposal to identify people with an apparent heart attack in colour images by detecting characteristic postures of heart attack. The method identifying infarcts makes use of convolutional neural networks. These have been trained with a specially prepared set of images that contain people simulating a heart attack. The promising results in the classification of infarcts show 91.75% accuracy and 92.85% sensitivity.

Detección de ataque cardíaco en imágenes en color utilizando redes neuronales convolucionales. (Resumen)

Las enfermedades cardiovasculares son la principal causa de muerte en todo el mundo. Por lo tanto, obtener ayuda a tiempo marca la diferencia entre la vida y la muerte. En muchos casos, no se obtiene ayuda a tiempo cuando una persona está sola y sufre un ataque cardíaco. Esto se debe principalmente al hecho de que el dolor le impide pedir ayuda. Este artículo presenta una nueva propuesta para identificar a las personas con un aparente ataque al corazón en imágenes en color mediante la detección de posturas características del ataque al corazón. El método para identificar infartos utiliza redes neuronales convolucionales. Estos han sido entrenados con un conjunto de imágenes especialmente preparadas que contienen personas que simulan un ataque cardíaco. Los resultados prometedores en la clasificación de infartos muestran 91.75 % de precisión y 92.85 % de sensibilidad.

Siguiendo los lineamientos de arquitectura se generó el módulo de nombre "ColorInfarctRecognizer", para la identificación de posibles eventos de infarto.

El módulo para la identificación de posibles infartos consta de los siguientes archivos:

- config.yaml
- ColorInfarctRecognizer.py
- model/rgbInfarctModel.h5

El archivo **config.yaml** contiene las variables mencionadas en la sección 3.9.1 donde se resalta especialmente las variables de 'FILTER_NAME' y 'CLASSES' (ver tabla 4.3), siendo el tipo de información a analizar y los eventos a detectar respectivamente.

El archivo **ColorInfarctRecognizer.py** contiene toda la lógica que permite funcionar al componente. Dentro de este archivo se encuentra una clase que lleva el mismo nombre y hereda de "ColorInfarctRecognizer". Esta clase implementa los métodos abstractos de la clase padre, siendo particularmente relevante y exclusivo, respecto a otros componentes, la implementación del método '*predict*'.

El método '*predict*', ejecuta la predicción de si una persona puede estar teniendo un infarto, usando una imagen de la que se ha extraído una persona del fondo, esto es cambiar cualquier píxel del fondo por un color único (morado). La imagen es pasada a la red neuronal previamente entrenada, la

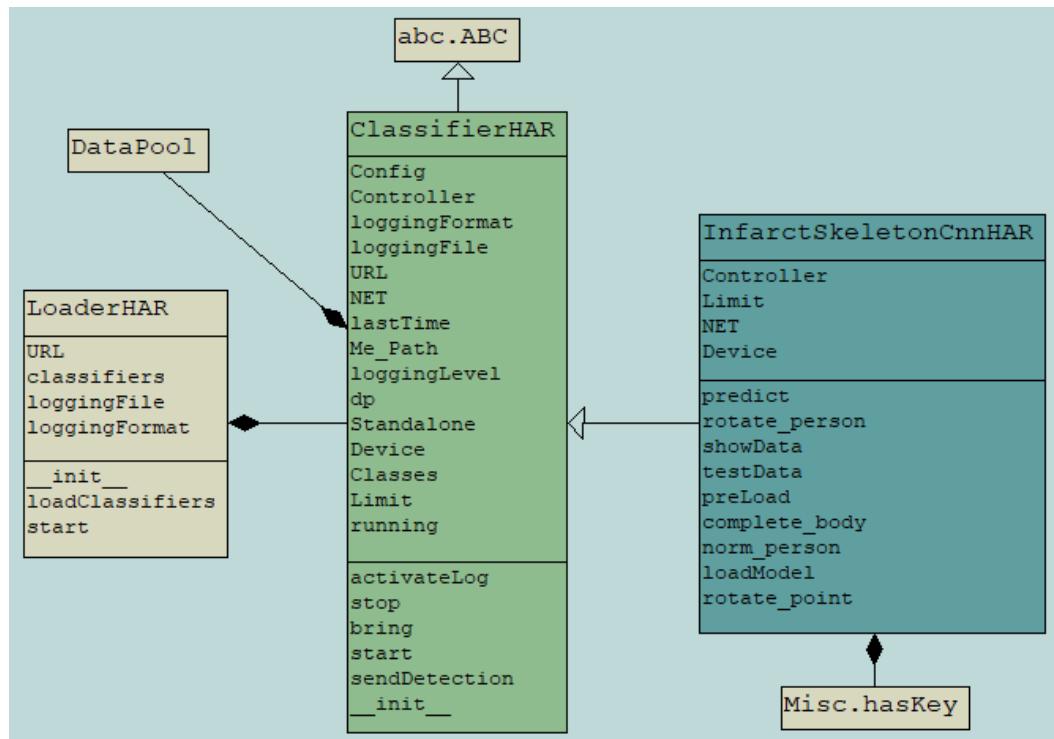


Figura 4.3: Modelo de clases de ColorInfarctRecognizer. Presenta las la relación y dependencias de las clases del módulo para identificación de eventos de posibles infartos de personas.

Tabla 4.3: Atributos del archivo de configuración de ColorInfarctRecognizer.

Atributo	Valor	Descripción
FILTER_NAME	CamController/person	Imagen de una persona extraída del fondo.
CLASSES	[None, Infarct]	Siendo 'None' una clase genérica para indicar la ausencia de evento detectado e 'Infarct' la identificación de una persona con posible infarto.
MODEL	model/rgbInfarctModel.h5	Entrenamiento previo que permite la identificación de las personas con un posible infarto.

cual tiene dos salidas, si el porcentaje de la segunda es mayor que la primera, entonces se entenderá como 'infarto' (ver figura 4.2). Además, para cumplir con los requisitos de ingreso de los datos a la red neuronal, este método ajusta el tamaño de la imagen para escalarla a 256x256 píxeles (ver alg. 4.2).

Listado 4.2: Predicción de infarto del módulo ColorInfarctRecognizer.

```

1  def predict(self, data):
2      """ Exec prediction to recognize an activity """
3      x = cv2.cvtColor(data.data, cv2.COLOR_BGR2RGB)
4      x = cv2.resize(x, (256, 256), interpolation = cv2.INTER_AREA)
5      x = np.expand_dims(x, axis=0)
6
7      array = self.MODEL.predict(x)
8      result = array[0]
9      answer = np.argmax(result)
10
11     dataReturn = []
12     auxData = '"t":"json", "idSource":{}'
  
```

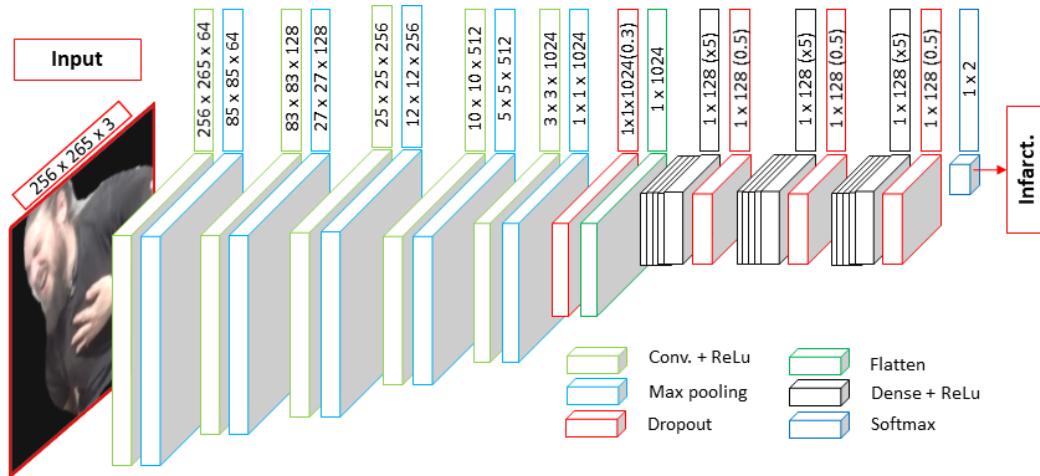


Figura 4.4: Red neuronal usada por ColorInfarctRecognizer. Presenta las capas que definen la red para la identificación de infartos en imágenes RGB tradicionales.

```

13
14     dataInf = Data()
15     dataInf.source_type = self.ME_TYPE
16     dataInf.source_name = self.ME_NAME
17     dataInf.source_item = ','
18     dataInf.data = self.CLASSES[answer]
19     dataInf.aux = '{' + auxData.format(data.id) + '}',
20     dataReturn.append(dataInf)
21
22     return dataReturn

```

4.2.2. Use of skeletons and facial expressions to identify heart attacks

Tomando un enfoque diferente para la identificación de posibles infartos en humanos, se propuso un nuevo componente, con una estrategia diferente con el objetivo adicional de reducir los tiempos de procesamiento requeridos manteniendo la precisión. En este caso en lugar de analizar toda la persona solo se consideró los puntos que forman el esqueleto y la expresión del rostro (ver figura 4.5), dando como resultado el artículo [Cita a Use of skeletons and facial expressions](#).

Use of pose estimation and pain face identification to identify heart attacks in images (Abstract)

Many cases in which deaths occur due to heart attacks, the person was alone. Getting help quickly makes the difference between life and death, however, felt pain can prevent the person from asking for help. For this reason, it is appropriate to have a mechanism that identifies these events automatically. In this article we present a work developed in this line, in the first instance we show two ways to identify the posture using the upper joints of the body. In addition to this, it is understandable that a person can have similar positions without suffering a heart attack, which leads to a high number of false positives, to reduce this we perform additional processing to verify that, in addition to the posture, the person shows a expression of pain on his or her face.

Uso de la estimación de pose y la identificación de la cara del dolor para identificar ataques cardíacos en imágenes. (Resumen)

Muchos casos en los que ocurren muertes debido a ataques cardíacos, la persona estaba sola. Obtener ayuda rápidamente hace la diferencia entre la vida y la muerte, sin embargo, sentir dolor puede evitar que la persona pida ayuda. Por esta razón, es apropiado tener un

mecanismo que identifique estos eventos automáticamente. En este artículo presentamos un trabajo desarrollado en esta línea, en primera instancia mostramos dos formas de identificar la postura utilizando las articulaciones superiores del cuerpo. Además de esto, es comprensible que una persona pueda tener posiciones similares sin sufrir un ataque cardíaco, lo que conduce a una gran cantidad de falsos positivos, para reducir esto realizamos un procesamiento adicional para verificar que, además de la postura, la persona muestra una expresión de dolor en su rostro.

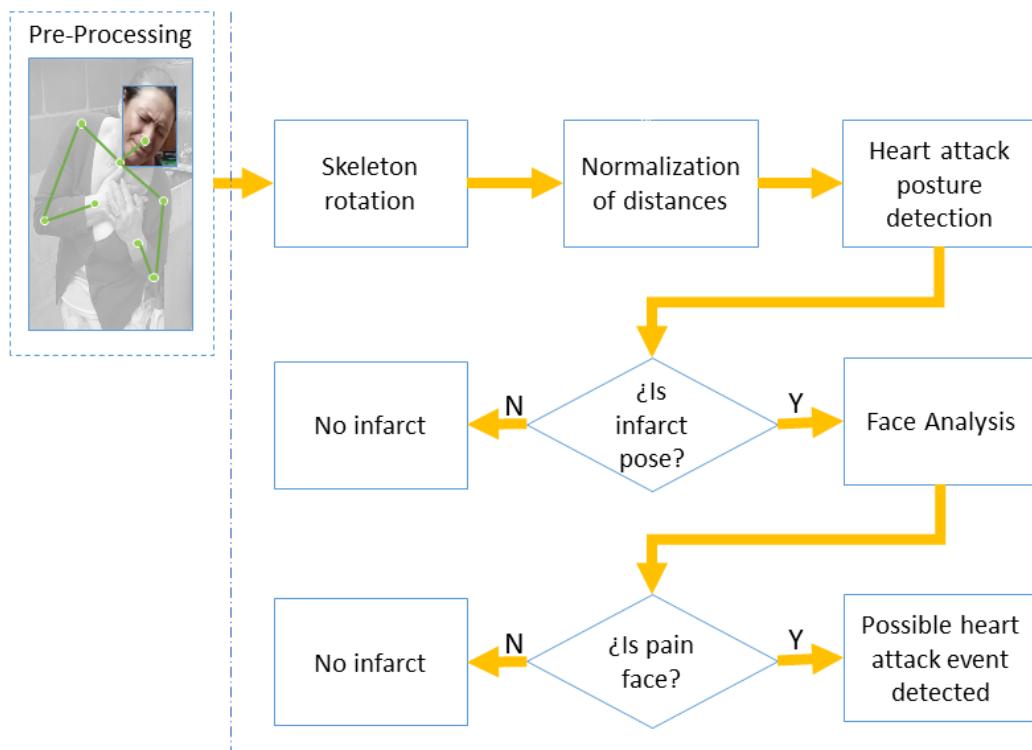


Figura 4.5: Flujo de reconocimiento de infartos del módulo SkeletonInfarctRecognizer. Presenta los pasos para la inferencia de un posible infarto usando las articulaciones superiores y la expresión del rostro.

Siguiendo los lineamientos de arquitectura se generó el módulo de nombre “SkeletonInfarctRecognizer”, para la identificación de posibles eventos de infarto.

El módulo para la identificación de posibles infartos consta de los siguientes archivos:

- config.yaml
- ColorInfarctRecognizer.py
- model/skeletonInfarctModel.h5

El archivo **config.yaml** contiene las variables mencionadas en la sección 3.9.1 donde se resalta especialmente las variables de 'FILTER_NAME' y 'CLASSES' (ver tabla 4.4), siendo el tipo de controlador (fuente de información) a filtrar para la inferencia y los eventos a detectar respectivamente.

El método '*predict*', ejecuta la predicción de si una persona puede estar teniendo un infarto, usando las articulaciones superiores, además extrae el cuadro que contiene el rostro de la persona, para luego pasarlo por las redes que identificaran de forma independiente si la persona tiene una postura equivalente a un infarto, y, además, tiene expresión de dolor (ver alg. 4.3).

Listado 4.3: Predicción de infarto del módulo SkeletonInfarctRecognizer.

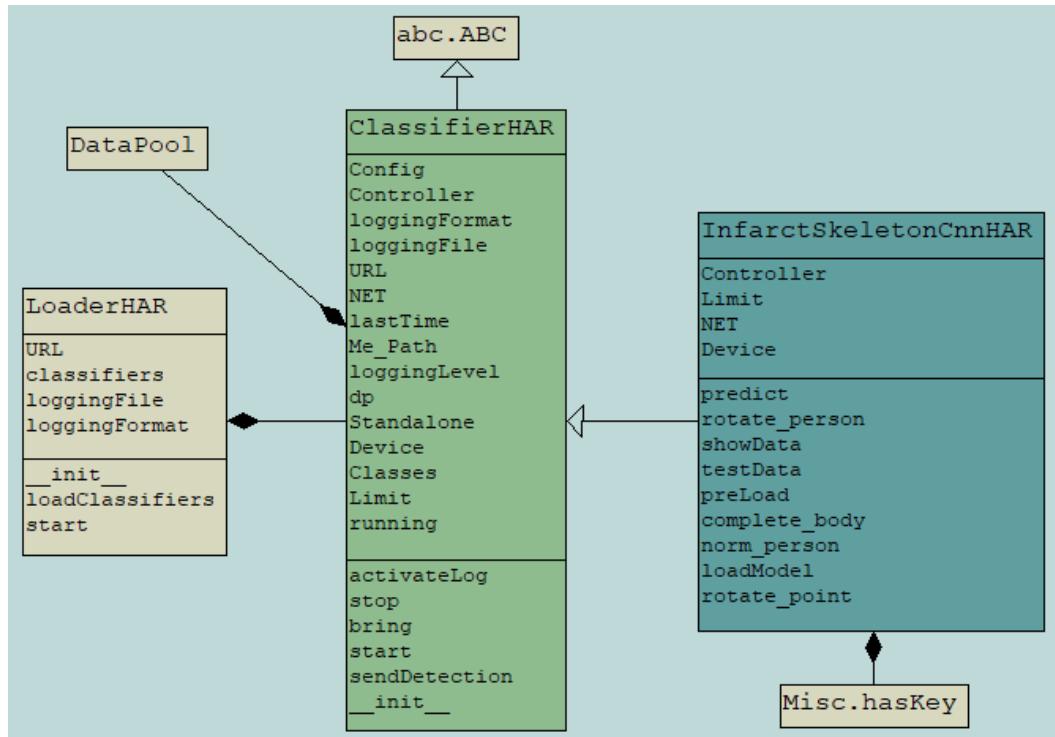


Figura 4.6: Modelo de clases de SkeletonInfarctRecognizer. Presenta las la relación y dependencias de las clases del módulo para identificación de eventos de posibles infartos de personas.

Tabla 4.4: Atributos del archivo de configuración de ColorInfarctRecognizer.

Atributo	Valor	Descripción
FILTER_NAME	['CamController/rgb', 'CamController/skeleton']	de esta forma se consulta las imágenes completas en formato 'RGB' y además los puntos de las articulaciones de las personas detectadas.
CLASSES	[None, Infarct]	Siendo 'None' una clase genérica para indicar la ausencia de evento detectado e 'Infarct' la identificación de una persona con posible infarto.
MODEL	model/SkeletonInfarctModel.h5	Entrenamiento previo que permite la identificación de las personas con un posible infarto.

4.3. CONSTRUCCIÓN DEL MÓDULO PARA EL RECONOCIMIENTO DE CAÍDAS

4.4. CONSTRUCCIÓN DEL MÓDULO PARA ENVÍO DE ALERTAS

Esta sección describe la utilización de la plantilla diseñada para la creación de módulos para la notificación de eventos. Para ello se seleccionaron dos canales: Correo tradicional y App específica con comunicación por Web Sockets (ver 2.5.9).

4.4.1. Módulo para la notificación usando correo electrónico - MailChannel

Siguiendo los lineamientos de arquitectura se generó el módulo de nombre “MailChannel”, para la identificación de posibles eventos de infarto.

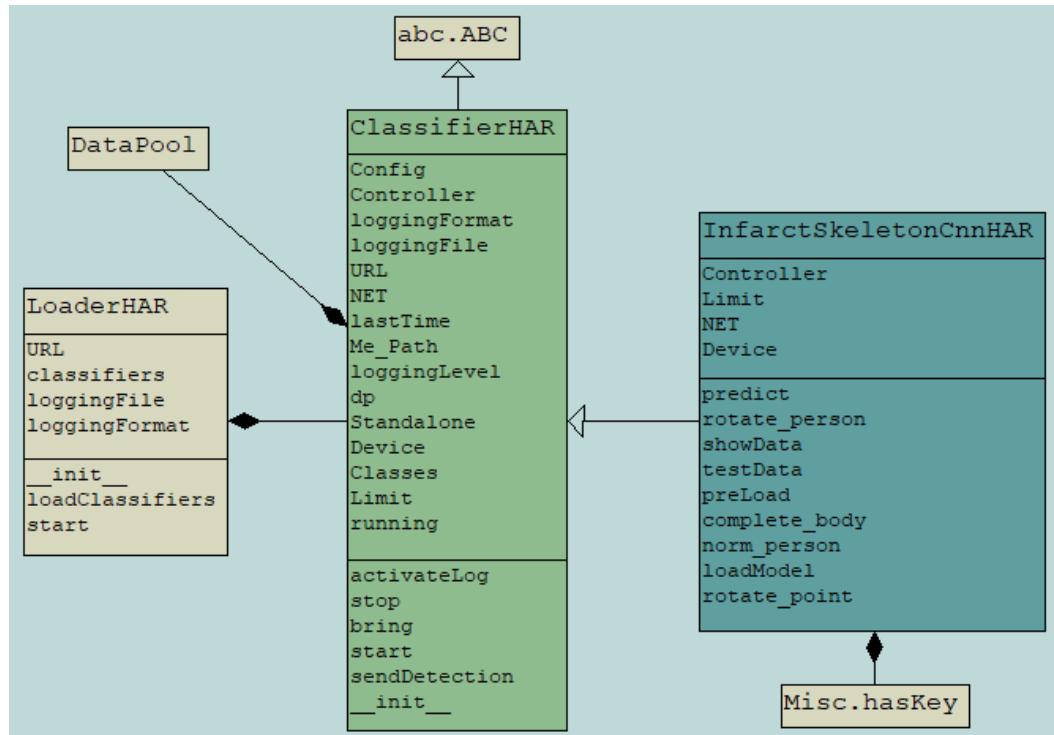


Figura 4.7: Modelo de clases de MailChannel. Presenta las relaciones y dependencias de las clases del componente que envía alertas a través de correo electrónico.

El componente MailChannel consta de:

- config.yaml
- MailChannel.py

El archivo **config.yaml** contiene, además de las variables comunes para todo componente, las mencionadas en la sección 3.11.1.

Por su parte **MailChannel.py** contiene la lógica específica de envío de correo y dentro de ella, resalta el método **'trySend'** encargado del envío del mensaje.

4.5. CONSTRUCCIÓN DEL MÓDULO PARA ANÁLISIS DE EVENTOS COMPLEJOS