UNIVERSIDAD TECNOLÓGICA DE PANAMÁ

FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

TRADUCCIÓN DE SEÑAS (LSP) A TEXTO (ESPAÑOL) EN TIEMPO REAL CON REDES NEURONALES PROFUNDAS

INTEGRANTE

ÁLVARO A. TERÁN QUEZADA 20-70-4082

ASESOR

PROF. JAVIER SÁNCHEZ GALÁN

TRABAJO DE GRADUACIÓN PARA OPTAR AL TÍTULO DE LICENCIADO EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

2022

RESUMEN

El reconocimiento de señas ha encontrado en las redes neuronales convolucionales ("*CNN*") una alternativa que ha permitido avances para afrontar este problema de visión artificial. Sin embargo, son cada vez más las redes profundas de las cuales se dispone, y las redes neuronales recurrentes ("*RNN*"), en particular, se han convertido en un medio para dar soluciones a problemas que involucran datos secuenciales. Se propone con esta investigación el desarrollo de un sistema de traducción de señas de LSP a español apoyado en *RNN* que posibilitan trabajar con señas no estáticas (como *data* secuencial), el gran reto por cumplir por parte de las *CNN*. El modelo de aprendizaje profundo presentado se enfoca en la detección de acciones, en este caso, la ejecución de las señas, procesando el contexto entre los cuadros que componen videos de las señas. La propuesta es una solución holística que considera, adicional a las manos, los componentes de postura corporal y cara, entendiendo que, al comunicarnos por lenguas de señas, importan características visuales más allá de los gestos con las manos. Se espera el impulso del estado del arte para traducción LSP-español en vista de que se amplían considerablemente las posibilidades de señas traducibles con aprendizaje profundo; para un conjunto de datos de entrenamiento de 300 videos (de 30 cuadros cada uno) para 3 clases (señas distintas) posibles, se alcanzó una precisión de 98.8%, haciendo de este un valioso sistema base para conseguir una comunicación efectiva entre usuarios de LSP e hispanohablantes.

ABSTRACT

Sign Language Recognition has found in Convolutional Neural Networks (*CNN*) an alternative that has allowed advances to face this Computer Vision problem. However, more Deep Networks architectures are available each day, and Recurrent Neural Networks (*RNN*) have become a means for providing solutions to problems involving sequential data. This research proposes the development of a sign language translation system that converts Panamanian Sign Language ("LSP") signs into text in Spanish backed up by *RNN* that makes it possible to work with non-static signs (as sequential

data), the daunting challenge to be met by ConvNets. The Deep Learning model presented focuses on Action Detection, in this case, the execution of the signs; processing the context between the frames of a determine video of some sign. The proposal is a holistic solution that considers, in addition to the Hands component, those of Face and Pose, on the understanding that, when communicating through sign languages, visual characteristics matter beyond Hand Gestures. The improvement of the state of the art for LSP-Spanish translation is expected in view of the expansion of the possibilities of translatable signs with Deep Learning; for a training dataset of 300 videos (of 30 frames each) for 3 possible classes (different signs considered), an accuracy of 98.8% was achieved, making this a valuable base system for effective communication between LSP users and Spanish speakers.

DEDICATORIA

A mi mamá, mi papá y mi hermano.

A mis abuelos, mis tíos, mis primos y mi ahijado.

A mi familia en forma de amigos y seres especiales para mí.

A los estudiantes que quieren aportar con investigación y proyectos valiosos, a quienes invito a creer en ustedes mismos, y lograr grandes cosas.

A mi universidad, en especial a mi facultad.

Al yo del pasado quien fue motivado por las razones correctas. Y al yo del futuro, para que sea fiel a sus convicciones, y siempre esté dispuesto a aprender, y a crecer como profesional y como persona.

A las luchas por inclusión y por equidad desde la justicia, la empatía y la sensatez.

A la Comunidad de Sordos de Panamá; a la espera de que esto sume en sus vidas directamente, y que represente un impacto positivo para la sociedad.

Porque hay maneras de que, al ganar algunos, ganemos todos.

AGRADECIMIENTOS

Agradezco a Dios, de forma superlativa, y desde la humildad; por esta oportunidad, y por todo cuanto tenga que agradecer en la vida. Agradezco a mi mamá, Orlenia, a mi papá, Gustavo Alberto, y a mi hermano, Gustavo José. Ha sido fundamental el apoyo y ejemplo de mi familia en esta etapa y en todo lo que me ha hecho estar aquí y ser quien soy; por esto, les agradezco en esta ocasión, y espero poder agradecerles el resto de mi vida como lo tienen merecido.

Mis abuelas, mi tía y las personas (y seres) especiales que me ha regalado la vida y que me han ayudado, apoyado y enseñado tanto. A Venezuela, por lo que significa para mí, más allá de haber nacido allí, y a Panamá por lo que ha significado estos años.

Gracias a los que han aportado para la realización de este proyecto; desde haber contribuido a que naciese mi amor por los idiomas y pudiese aprender sobre ellos, hasta haber culminado esta etapa. En el medio, hubo mucho: formación académica, orientación, recomendaciones, levantamiento de información, reuniones, y escucha.

Siento la necesidad de agradecer a las comunidades de Python, YouTube, y Kaggle. Hay muchas personas en el mundo queriendo aportar y consiguiendo que avancemos, estoy decidido a reconocerlos y ser parte de algo más grande que nosotros mismos. A aquellos cuyo interés es hacer las cosas bien: gracias. Son muchos quienes indirectamente hicieron parte de este proceso, y espero yo retribuir a esta y otras comunidades, y ser parte, de diversas formas, de muchos proyectos.

A mi universidad, mis profesores y compañeros. Aquellos que me han inspirado de alguna u otra manera. Y por sus inestimables enseñanzas: gracias. También compañeros investigadores de cuyos trabajos aprendí. Y a la Escuela Nacional de Sordos, de la cual recibí palabras de compromiso, felicitaciones y agradecimiento.

Finalmente, gracias a mi profesor y asesor, el doctor Javier Sánchez Galán. Su compromiso y pasión por la investigación pudo haber acrecentado mi interés no solo en tener una experiencia de este tipo, sino también considerar esta vía para impactar positivamente en la sociedad. Es esa la finalidad de este proyecto, y agradezco nuevamente a todos quienes lo han hecho posible.

ÍNDICE GENERAL

RESUMEN i
ABSTRACT i
DEDICATORIA iii
AGRADECIMIENTOS iv
ÍNDICE DE FIGURAS x
ÍNDICE DE TABLAS xvii
INTRODUCCIÓN 19
Problemática 20
Situación Actual 20
Descripción del Problema 22
Propuesta de la solución 23
Justificación 25
Delimitación 26
Objetivo General 27
Objetivos Específicos 27
Antecedentes 29
Estudios Relacionados 29
Bases Teóricas 31
Inteligencia Artificial 31
Aprendizaje Automático 33
Visión Artificial y Aprendizaje Automático 37
Clasificación de imágenes 38
Aprendizaje Profundo 40
Redes Neuronales Profundas 41

Redes Neuronales Convolucionales 67
Redes Neuronales Recurrentes 77
Funcionamiento de las RNN 82
Vanilla RNN 83
LSTM 87
GRU 92
Visión Artificial (dentro de ML) 94
Detección de Objetos 99
Fully Convolutional Networks 103
YOLO 104
SSD 104
Detección de Acciones 106
Topologías de la Detección de Acciones 107
Action Detection 111
Holística 112
Sign Language Recognition 113
Transfer Learning y modelos preentrenados 114
Tecnologías de desarrollo ML 117
Dependencias y librerías 117
TensorFlow Object Detection 120
MediaPipe 121
Mediapipe Holistic 127
Definición de la metodología 129
Exploración inicial 129
Definición de algoritmos y modelos 129

Recolección de datos 129
Construcción de modelo base 130
Desarrollo de modelo de inferencia 130
Análisis de resultados 130
Diseño del proyecto 131
Planificación de ideas 131
Recolección de imágenes 132
Preprocesamiento de datos 132
Mapa de etiquetado 133
Desarrollo de modelos de prueba 133
Pruebas de prototipo 134
Desarrollo de modelo final 134
Validación 134
Diseño de la interfaz 134
Desarrollo del proyecto 135
Desarrollo del módulo para señas estáticas 135
Configuración de rutas 135
Creación de mapa de etiquetado 136
Recolección de imágenes 136
Etiquetado de imágenes 136
Creación de TF records 137
Descarga de modelo TensorFlow preentrenado 137
Copiado de model config al directorio de entrenamiento 137
Actualizar Config para Transfer Learning 138
Entrenamiento del modelo 138

Carga del modelo entrenado desde el Checkpoint 138
Construcción de sistema de detección de objetos 138
Desarrollo del módulo para señas dinámicas 139
Instalación de librerías y dependencias 139
Detección de los keypoints/landmarks usando Mediapipe Holistic 140
Capturando imágenes por cámara 141
Extraer valores keypoints 141
Configuración de directorios para la colección de arreglos 141
Recolección de valores keypoint para training y testing 141
Preprocesamiento de los datos, y creación de etiquetas y features 142
Construcción y entrenamiento de la RN tipo LSTM 143
Guardar pesos (weights) 144
Construcción de sistema de detección de acciones 144
Modelo para señas estáticas 148
Configuración de las métricas a utilizar 148
Evaluación del modelo 148
Monitoreo del progreso de entrenamiento 148
Detección en tiempo real 149
Exportando el modelo 149
Modelo para señas dinámicas 149
Predicciones 149
Evaluación usando matriz de confusión 149
Ejecución en tiempo real 149
Experimentos y validación 150
Conclusiones 162

Recomendaciones 164
Referencias 166
Anexos 171
Anexos A: Desarrollo del módulo para señas estáticas 171
Configuración de rutas 171
Creación de mapa de etiquetado 171
Descarga de modelo TensorFlow preentrenado y reubicación de model config 172
Actualizar Config para Transfer Learning 172
Entrenamiento del modelo 173
Detección (y traducción) de señas 174
Anexos B: Desarrollo del módulo para señas dinámicas 176
Instalación de librerías y dependencias 176
Detección de los keypoints/landmarks usando Mediapipe Holistic 176
Capturando imágenes por cámara 178
Extraer valores keypoints 179
Configuración de directorios para la colección de arreglos 180
Recolección de valores keypoint para training y testing 180
Preprocesamiento de los datos, y creación de etiquetas y features 182
Construcción y entrenamiento de la RN tipo LSTM 183
Detección (y traducción) de señas a texto, por detección de acciones 185
Anexos C: Pruebas y validación - Modelo para señas dinámicas 189
Evaluación usando matriz de confusión (y visualización de resultados) 189

ÍNDICE DE FIGURAS

Ilustración 1 Alfabeto manual de la Lengua de Señas Panameñas (parte 1/2). Fuente: obtenido de [6]. 21
Ilustración 2 Alfabeto manual de la Lengua de Señas Panameñas (parte 2/2). Fuente: obtenido de [6]. 22
Ilustración 3 Modelo genérico de una red neuronal artificial profunda. Fuente: adaptado de [7]. 24
Ilustración 4 A la izquierda, el proceso de entrenamiento de modelos tradicionales; a la derecha, el proceso de entrenamiento para modelos de Inteligencia Artificial [23]. 31
Ilustración 5 Comparación de entradas y salidas según programación tradicional (arriba) y aprendizaje automático (abajo). Fuente: adaptado de [24]. 34
Ilustración 6 Proceso de desarrollo lógico bajo programación tradicional. Fuente: adaptado de [25]. 35
Ilustración 7 Proceso de desarrollo lógico con enfoque de aprendizaje automático. Fuente: adaptado de [25]. 35
Ilustración 8 Adaptándose a cambios automáticamente. Fuente: adaptado de [25]. 36
Ilustración 9 El aprendizaje automático puede impactar en cómo aprendemos y cómo entendemos los problemas. Fuente: adaptado de [25]. 37
Ilustración 10 Cómo interpretan las computadoras imágenes de datos de pixeles sin procesar. Fuente: obtenido de [27]. 38
Ilustración 11 Un modelo de aprendizaje automático clasificador de imágenes “imita el sistema cognitivo humano”. Fuente: adaptado de [29]. 39
Ilustración 12 Clasificador de dígitos. Fuente: obtenido de [24]. 39
Ilustración 13 Diagrama representando cómo el aprendizaje profundo está contenido en el aprendizaje automático, y este, a su vez, está incluido dentro de la inteligencia artificial. Fuente: obtenido de [24]. 40
Ilustración 14 Representaciones de datos aprendidas por un modelo clasificador de dígitos. Fuente: obtenido de [24]. 41
Ilustración 15 Neurona biológica. Fuente: adaptada de [25]. 42
Ilustración 16 Representación simplificada de una red neuronal biológica con algunas de sus partes. Fuente: adaptada de [31]. 42

Ilustración 17 Unidad lineal de la forma y=wx+b que recuerda a la ecuación de la línea. Fuente: presentada en [33]. 44
Ilustración 18 Redes neuronales artificiales haciendo cálculos simples. Fuente: presentada en [25]. 44
Ilustración 19 Meta de una red neuronal para poder predecir. Fuente: adaptado de [30]. 45
Ilustración 20 Red neuronal con varias entradas procesadas por una única neurona. Fuente: adaptada de [31]. 46
Ilustración 21 Unidad lógica umbral: una neurona artificial que computa una suma ponderada de sus entradas luego aplica una función de paso. Fuente: adaptada de [25]. 46
Ilustración 22 Arquitectura de un perceptrón multicapa con dos (2) entradas, una (1) capa oculta de cuatro (4) neuronas y tres (3) neuronas de salida (representando neuronas bias explícitamente). Fuente: adaptado de [25]. 48
Ilustración 23 MLP moderna de clasificación; incluyendo ReLU y softmax. Fuente: adaptado de [25]. 50
Ilustración 24 Función con múltiples entradas [30]. 51
Ilustración 25 Cálculo de puntaje de pérdida. Fuente: adaptado de [24]. 54
Ilustración 26 Abstracción sobre una red neuronal con pesos. Fuente: adaptado de [30]. 54
Ilustración 27 Puntaje de pérdida como valor para ajustar los pesos. Fuente: obtenido de [24]. 58
Ilustración 28 Se toman pequeños pasos en la dirección en la cual la pérdida decrece más [29]. 58
Ilustración 29 Curvas de aprendizaje. Fuente: obtenido de [25]. 59
Ilustración 30 Algunos datos de muestra. Fuente: El autor. 60
Ilustración 31 Representación de un modelo siendo entrenado por batch (el proceso se repite por iteración). Fuente: adaptado de [25]. 60
Ilustración 32 Curvas de pérdida y precisión en los conjuntos de entrenamiento (línea sólida) y validación (línea punteada). Fuente: adaptado de [29]. 61

Ilustración 33 Cómo funciona un optimizador tipo descenso de gradiente. Fuente: obtenido de [29]. 61
Ilustración 34 Regularización de detención temprana. Fuente: obtenido de [25]. 64
Ilustración 35 Red neuronal superficial genérica (izquierda) versus una red neuronal profunda genérica (derecha). Fuente: El autor. 65
Ilustración 36 Red neuronal superficial con una capa de entrada con dos (2) neuronas, una intermedia (oculta) con cinco (5) y una capa de salida con solo una (1) neurona. Fuente: El autor. 65
Ilustración 37 Red neuronal profunda evidenciando lo complejo de algunas redes. Fuente: obtenido de [34]. 66
Ilustración 38 Cambios en los pesos capa a capa. Fuente: Obtenido en [28]. 66
Ilustración 39 ConvNets con campos receptivos locales rectangulares [25]. 68
Ilustración 40 Extracción de características en una ConvNet. Fuente: adaptado de [27]. 69
Ilustración 41 Aplicando dos filtros diferentes para obtener dos mapas de características. Fuente: obtenido de [25]. 69
Ilustración 42 Invariancia a pequeñas traslaciones [25]. 70
Ilustración 43 Capas convolucionales con múltiples mapas de características, e imágenes con canales de tres colores. Fuente: obtenido de [25]. 71
Ilustración 44 Arquitectura CNN típica. Fuente: adaptado de [25]. 72
Ilustración 45 Pares convolución-pooling procesando una imagen. Fuente: adaptado de [35]. 72
Ilustración 46 Ejemplo de funcionamiento computacional de la estructura. Fuente: obtenido de [36]. 73
Ilustración 47 Cómo funcionan las convoluciones. Fuente: presentado en [24]. 73
Ilustración 48 Patches de convolución tamaño 3 x 3 (izquierda) con strides tamaño 2 x 2 (derecha). Fuente: obtenido de [24]. 74
Ilustración 49 La entrada de tipo imagen (izquierda) se convierte a algo que pueda ser entendido por la computadora: una matriz de números (derecha). Fuente: obtenido de [31]. 74
Ilustración 50 Matriz de entrada (izquierda), y filtro. Fuente: obtenido de [31]. 75

Ilustración 51 Matriz de entrada de la cual se considera un grupo de números (izquierda) y primer número generado de aplicar este filtro (derecha). Fuente: obtenido de [31]. .. 75
Ilustración 52 Estructura con bloques convolucionales. Fuente: adaptado de [37]. 76
Ilustración 53 Generando nuevas instancias de entrenamiento a partir de las existentes. Fuente: el autor. 77
Ilustración 54 Una neurona recurrente (izquierda) desenrollada a través del tiempo (derecha). Fuente: obtenida de [25]. 78
Ilustración 55 Redes feedforward (izq.) y recurrente (dcha.). Fuente: adaptada de [31]. 78
Ilustración 56 Pronosticando 10 pasos adelante, uno a la vez. Fuente obtenida de [25]. 79
Ilustración 57 Retropropagación en términos de capas (en lugar que de operaciones). Fuente: adaptada de [30]. 84
Ilustración 58 Retropropagación a través del tiempo, en cinco (5) momentos. Fuente: obtenida de [25]. 85
Ilustración 59 Regla de actualización del gradiente. Fuente: adaptada de [38]. 85
Ilustración 60 Gráficas de las funciones sigmoide (izquierda) y tanh (derecha). Fuente: adaptada de [38]. 86
Ilustración 61 Leyenda de símbolos. Fuente: obtenida de [38]. 87
Ilustración 62 Calculando el estado de célula. Fuente: obtenida de [38]. 88
Ilustración 63 Operaciones de la puerta de olvido. Fuente: obtenida de [38]. 89
Ilustración 64 Operaciones de la puerta de entrada. Fuente: obtenida de [38]. 90
Ilustración 65 Calculando el estado de la célula. Fuente: obtenida de [38]. 90
Ilustración 66 Operaciones de la puerta de salida. Fuente: obtenida de [38]. 91
Ilustración 67 Célula tipo LSTM. Fuente: obtenida de [25]. 92
Ilustración 68 Célula tipo GRU. Fuente: obtenida de [25]. 92
Ilustración 69 Célula GRU y sus puertas. Fuente: obtenida de [38]. 93
Ilustración 70 Visión artificial respecto a Aprendizaje Automático. Fuente: obtenido de [29]. 94
Ilustración 71 Machine Learning versus Deep Learning ante el mismo problema de Computer Vision. Fuente: adaptado de [27]. 95

Ilustración 72 Conjuntos de datos agrupados según la etiqueta correspondiente: “perro” (izquierda), “gato” (centro) y “lobo” (derecha). Fuente: el autor. 96
Ilustración 73 Datos para etiquetar individualmente de acuerdo con el caso. Fuente: el autor. 96
Ilustración 74 Imágenes (fotos) con variedad de poses, pero con un único ejemplar. Fuente: el autor. 97
Ilustración 75 Imágenes (fotos) con variedad de perritos, pero de una misma raza. Fuente: el autor. 98
Ilustración 76 Conjunto de datos con variedad de características, pero que puede ameritar ser mucho más nutrido dependiendo del caso de uso y el alcance del sistema. Fuente: el autor. 99
Ilustración 77 Resultado de un sistema de reconocimiento de objetos en ejecución. Fuente: el autor. 100
Ilustración 78 Salidas independientes de un localizador de objetos que reconoce perros y gatos, y los ubica en una imagen. Fuente: el autor. 100
Ilustración 79 Salidas de un detector de objetos que identifica distintos elementos. Fuente: el autor. 101
Ilustración 80 Se ubican objetos con mayor precisión para segmentación de imágenes. Fuente: el autor. 102
Ilustración 81 Misma red procesando una imagen pequeña (izquierda) y grande (derecha). Fuente: obtenido de [25]. 103
Ilustración 82 Métrica de intersección sobre unión para cajas bounding. Fuente: obtenido de [25]. 105
Ilustración 83 Una CNN detectando múltiples objetos en simultáneo. Fuente: obtenido de [25]. 106
Ilustración 84 Aplicaciones de la estimación de poses en el área de salud. Fuente: obtenida de [41]. 107
Ilustración 85 Representación del Pose tracking en ejecución. Fuente: obtenido de [42]. 111
Ilustración 86 Señas de letras “L”, “S” y “P” (de izquierda a derecha) en LSP. Fuente: el autor. 114

Ilustración 87 Transfer learning en el cual se utiliza parte de una red neuronal existente para desarrollo de otro. Fuente: obtenido de [25]. 115
Ilustración 88 Transfer Learning vs entrenamiento tradicional (desde cero). Fuente: obtenido de [44]. 116
Ilustración 89 Transfer Learning vs Machine Learning tradicional. Fuente: obtenido de [45]. 117
Ilustración 90 Keras y TensorFlow en conjunto. Fuente: obtenido de [24]. 118
Ilustración 91 Estructura detallada de un bloque de convolución de profundidad separable y tipo cuello de botella. Fuente: obtenido de [46]. 121
Ilustración 92 A la izquierda, una imagen de entrada; a la derecha, los efectos de aplicar sobre ella segmentación semántica. Fuente: obtenida de [24]. 122
Ilustración 93 Landmarks del componente pose. Fuente: adaptado de [48]. 123
Ilustración 94 Landmarks del componente “cara”. Fuente: adaptado de [51]..... 125
Ilustración 95 Landmarks del componente mano. Fuente: adaptado de [50]. 127
Ilustración 96 Diagrama sobre la metodología. Fuente: El autor..... 131
Ilustración 97 Marcos de trabajo y recursos principales en un proyecto DL. Fuente: adaptado de [24]. 135
Ilustración 98 Etiquetado de una seña con LabelImg. Fuente: el autor..... 137
Ilustración 99 Modelo en ejecución con salidas con un 95% de confiabilidad (izquierda; seña con una rotación y separación del pulgar) y un 100% de confiabilidad (postura típica) para la letra "A" en LSP. Fuente: el autor..... 139
Ilustración 100 Procesamiento de la solución según las topologías involucradas. Fuente: el autor. 140
Ilustración 101 Captura de pantalla con los landmarks correspondientes a las tres (3) topologías detrás de MediaPipe Holistic. Fuente: el autor. 142
Ilustración 102 Estados del modelo – proceso de entrenamiento. Fuente: el autor. ... 143
Ilustración 103 Resumen del modelo (estructura). Fuente: el autor. 144
Ilustración 104 Sistema en ejecución. Posición neutral (izquierda), haciendo la seña para “hola” en LSP (centro), terminando la seña y con el resultado ya en pantalla (derecha). Fuente: el autor. 145

Ilustración 105 Comenzando a hacer una segunda seña: “estoy bien” (izquierda), parte final de la seña para expresar “estoy bien” (centro), traducción de la seña mientras se retoma una posición neutral (derecha). Fuente: el autor. 145
Ilustración 106 Caso de uso típico del sistema. Fuente: el autor. 146
Ilustración 107 Diagrama del sistema. Fuente: el autor. 146
Ilustración 108 Secuencia de la seña "hola" en 6 frames. Fuente: el autor. 150
Ilustración 109 Secuencia de la seña "buenos días" en 6 frames. Fuente: el autor. ... 151
Ilustración 110 Secuencia de la seña "estoy bien" en 6 frames. Fuente: el autor. 151
Ilustración 111 Secuencia de la seña "gracias" en 6 frames. Fuente: el autor. 152
Ilustración 112 Secuencia de la seña "¿cómo estás?" en 6 frames. Fuente: el autor. 152
Ilustración 113 Matriz de confusión - clase "hola", experimento #1. Fuente: el autor.. 153
Ilustración 114 Matriz de confusión - clase "buenos días", experimento #1. Fuente: el autor. 153
Ilustración 115 Matriz de confusión - clase "estoy bien", experimento #1. Fuente: el autor. 154
Ilustración 116 Matriz de confusión - clase "gracias", experimento #1. Fuente: el autor. 154
Ilustración 117 Matriz de confusión - clase "¿cómo estás?", experimento #1. Fuente: el autor. 155
Ilustración 118 Matriz de confusión - clase "hola", experimento #2. Fuente: el autor.. 157
Ilustración 119 Matriz de confusión - clase "estoy bien", experimento #2. Fuente: el autor. 157
Ilustración 120 Matriz de confusión - clase "gracias", experimento #2. Fuente: el autor. 158
Ilustración 121 Gráfica de precisión vs época. "Epoch_categorical_accuracy" con TensorBoard. Fuente: el autor. 160
Ilustración 122 Gráfica de pérdida vs época. "Epoch_loss" con TensorBoard. Fuente: el autor. 160

ÍNDICE DE TABLAS

Tabla 1 Tipos de sistema según la "inteligencia" que presentan. Fuente: El Autor..... 32
Tabla 2 Partes de una red neuronal biológica y el elemento que inspiraron para redes neuronales artificiales. Fuente: El Autor..... 43
Tabla 3 Funciones principales de una neurona en el contexto de redes neuronales artificiales. Fuente: adaptado de [32]. 43
Tabla 4 Resumen de las métricas (y sus valores) - Experimento #1. Fuente: el autor. 155
Tabla 5 Resumen de las métricas (y sus valores) - Experimento #2. Fuente: el autor. 158

Capítulo I – Aspectos Generales

INTRODUCCIÓN

Dentro de la diversidad que caracteriza a la humanidad, las sociedades se han ido desarrollando, naturalmente, en base a mayorías; es así como las dinámicas sociales se han orientado a “cumplir” con el grueso poblacional. En cuanto a la comunicación, en particular, se han afrontado un sinnúmero de retos; la tecnología ha sido protagonista en soluciones para este tipo de problemáticas [1], [2], [3], [4]. Sin embargo, pese a haber conseguido aminorar y resolver problemas como la dificultad para comunicarse entre dos personas que no manejan el mismo idioma o haber posibilitado, por ejemplo, la comunicación en tiempo real entre personas que se encuentran a largas distancias; los avances tecnológicos no han ido al mismo ritmo en materia de *inclusión*.

Los grupos de personas con algún tipo de discapacidad se han visto menos favorecidos que sus pares sin discapacidad en cuanto a ciertos avances tecnológicos, dado que no es factible el uso de muchos de los sistemas desarrollados en distintas áreas para individuos de esta comunidad; en concreto, para algunas personas con determinado impedimento inherente a su condición. Es así como se han evidenciado progresos para estos (y otros grupos), en base a necesidades como las de autonomía personal; al tiempo que, si bien es cierto ha habido avances respecto a otros elementos que atañen a las personas con discapacidad, la investigación y el desarrollo han quedado de cierta forma en deuda con temas como algunos, justamente, entorno a la comunicación.

Una propuesta que podría marcar la diferencia para la comunidad con discapacidad auditiva en Panamá sería un sistema capaz de reconocer señas, de la Lengua de Señas Panameñas, a partir de una entrada visual y presentar un texto con la traducción correspondiente en español. Y es que se entiende la realidad actual de este grupo social como uno para el cual se ha buscado integración, pudiendo ser este sistema un paso para la inclusión integral de este grupo partiendo del supuesto que se tendría un medio de comunicación eficiente entre los usuarios de un par de sistemas de comunicación verbal y gestual, en este caso, *LSP-español*.

La *visión artificial* surge como el área de investigación dentro de la que caería este estudio y la disciplina para realizar el desarrollo sería *aprendizaje profundo*. Ya

habiéndose llevado a cabo desarrollos de *reconocimiento de señas* con *redes neuronales profundas (“Deep Neural Networks”; “DNN”)*, se está bajo la posibilidad de hacer implementaciones que no han sido típicamente estudiadas, como el uso de redes neuronales *recurrentes (“Recurrent Neural Networks”; “RNN”)*, no comunes en el área; en adición a las redes neuronales *convolucionales (“Convolutional Neural Networks”; “CNN”)*, frecuentemente utilizadas.

La hipótesis que motiva el hacer un desarrollo bajo este enfoque es conceptualmente sencilla. Se resume en que, si bien es cierto que las *CNN* son notoriamente buenas procesando *data* visual (como la imagen de una seña), tienen limitaciones para procesar agrupaciones de datos como un mismo elemento; las *RNN* por su lado, se especializan en reconocer las características secuenciales de los datos. Con base en lo anterior se pudiese especular en la oportunidad real de que el uso de esta estructura, como alternativa complementaria o alrededor de la que se haga un desarrollo, posibilite procesar agrupaciones de datos visuales como un único elemento; en particular un conjunto de poses y señas con las manos que respondan a un significado dado.

Problemática

Situación Actual

Se estima que alrededor de un *5%* de la población mundial lo conforman personas con algún grado de pérdida de audición [5]; buena parte de ellas (sobre todo aquellas cuyo tipo de discapacidad auditiva es de mayor gravedad), bien sea por necesidad en sí o conveniencia, optan por comunicarse mediante alguna *lengua de señas*. En Panamá, la situación es similar: un sector minoritario de la población se comunica principalmente por lengua de señas con otras personas de la comunidad, o bien, personas que también utilizan este tipo de sistema de comunicación verbal (como profesores o familiares, sin discapacidad auditiva).

La *Lengua de Señas Panameñas* (LSP) [6] es la lengua de señas utilizada en todo el territorio nacional; si bien es la única en el país, cabe resaltar que existen diferencias (en ocasiones marcadas) de cómo representan las señas o los gestos los individuos que se comunican en esta lengua, así como también el hecho de que, para determinado

concepto, se pueda tener una seña diferente de una región a otra. Se presenta el alfabeto manual de la LSP en Ilustración 1 e Ilustración 2.

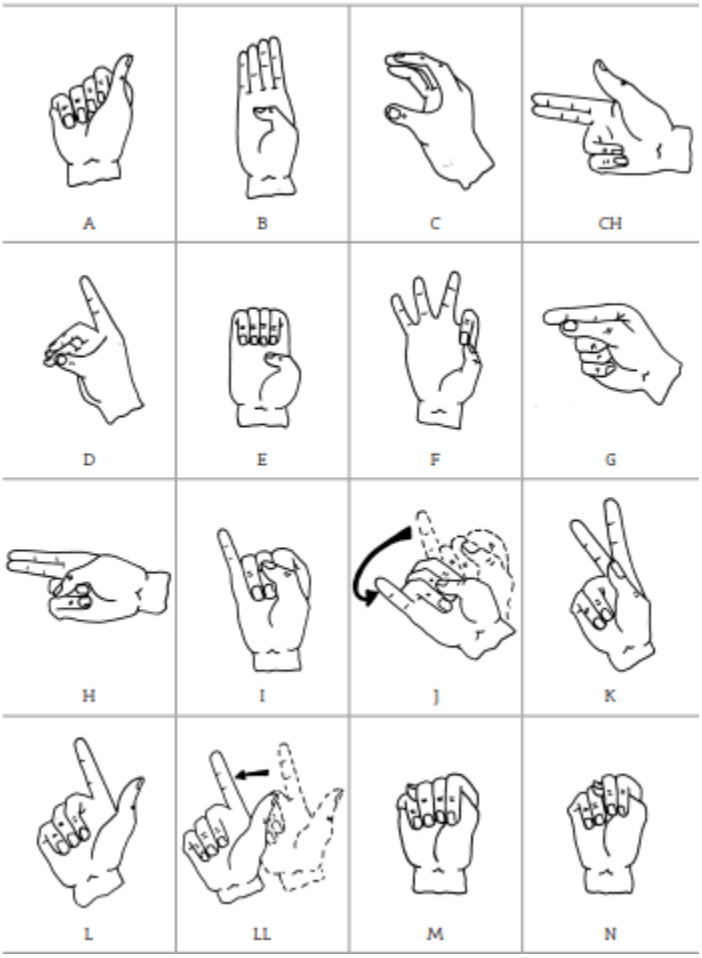


Ilustración 1 Alfabeto manual de la Lengua de Señas Panameñas (parte 1/2). Fuente: obtenido de [6].

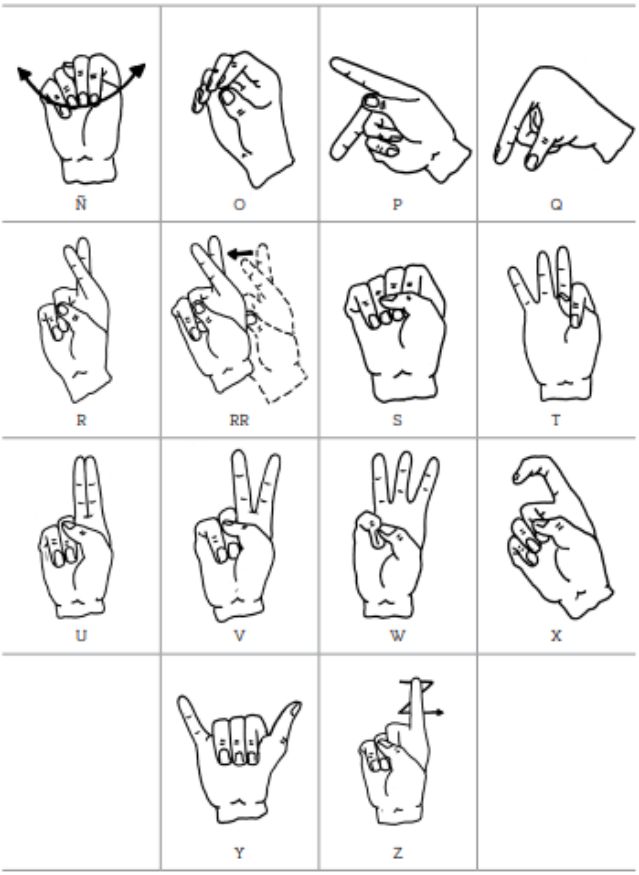


Ilustración 2 Alfabeto manual de la Lengua de Señas Panameñas (parte 2/2). Fuente: obtenido de [6].

Los esfuerzos, en el país, alrededor de la comunidad de personas con discapacidad auditiva se han dado desde una variedad de perspectivas, la mayoría de estos con componentes que buscan ser integradores. La realidad indica que existen diversas aristas a ser consideradas y muchas metas por cumplir, y que, como sociedad que busca ser más inclusiva, Panamá tendrá que mejorar su empeño y desempeño en esta constante lucha.

Descripción del Problema

Del listado de retos a los que se enfrenta Panamá en materia de inclusión, se tiene un sinnúmero de desafíos en lo relativo a los grupos de personas con algún tipo de discapacidad; entre los cuales se encuentra, evidentemente, el de personas con discapacidad auditiva. Compete, entonces, resaltar algunos de estos problemas a manera de contexto.

Necesidades no cubiertas y limitación de oportunidades, suelen ser los puntos principales. Así pues, se pueden mencionar una segmentación de la comunidad, una desatención a las nuevas necesidades dado el cambiante escenario social (nuevas tecnologías, variación de las dinámicas sociales, etc.), un sistema educativo que se desentiende deliberadamente de acciones que se podrían tomar (bien imitando modelos o por iniciativas novedosas), pobre disposición de contenido académico y de otras índoles, y pocos espacios, actividades y proyectos que busquen incluir a este grupo social a un entorno mayor, o bien, que permitan a otras personas involucrarse con quienes pertenecen a este grupo.

Se puede englobar este grupo de problemas en (o como) una única problemática: desigualdad y falta de inclusión de las personas con discapacidad auditiva en el contexto social nacional. Pudiéndose, y teniéndose que, combatir desde distintos ángulos. Dentro de estos problemas, el de contar con poca comunicación entre personas con discapacidad auditiva y aquellas que no conocen lenguas de señas ha supuesto un gran reto; siendo un problema en sí mismo, porque el relacionamiento de estos grupos (impedido sin necesidad) tendría consecuencias positivas (en términos de diversidad, colaboración y otros aspectos, y dado por la naturaleza característica de sinergia de los humanos), a la vez que tiene relación causal con elementos (generadores o aportadores) de otros problemas.

Propuesta de la solución

Una de las maneras de ir atendiendo la compleja problemática presentada es creando un canal mediante el cual pueda existir comunicación en tiempo real entre quienes no conocen LSP y quienes la emplean como principal medio de comunicación en su día a día. Esto abarcaría y cubriría varios de los inconvenientes, retos y puntos clave del problema identificado. Se aumentaría la visualización de este grupo social, y se estaría promoviendo este tipo de comunicación (entre LSP y español), e incentivando el aprendizaje de la lengua de señas panameñas, así como se dispondría de una nueva herramienta para lograr la inclusión de un grupo históricamente marginado y más recientemente integrado hasta cierto punto. Entre las ventajas también se halla el

capitalizar el talento humano con el que se dispone cuyo potencial no ha sido aprovechado al máximo.

Entonces, en vista de las ventajas que representaría una ampliación del entorno social de cada una de las personas con discapacidad auditiva (para sí mismas y para la sociedad), se propone un proyecto de investigación teórico-práctico a partir del cual se lleve a cabo el desarrollo de un sistema capaz de captar una serie de señas de la LSP y traducirlas al español, como base sustancial de lo que podría significar el establecimiento de un canal de comunicación novedoso entre personas con discapacidad auditiva que sepan LSP y personas sin discapacidad de este tipo que no sepan LSP.

Para este desarrollo se estarían utilizando herramientas y modelos de la subárea de Aprendizaje Automático ("*Machine Learning*") contenida en el área de Inteligencia Artificial, en particular, de la disciplina de Aprendizaje Profundo ("*Deep Learning*"); con enfoque en Redes Neuronales Convolucionales y Redes Neuronales Recurrentes. En Ilustración 3 se muestra un modelo de red neuronal artificial.

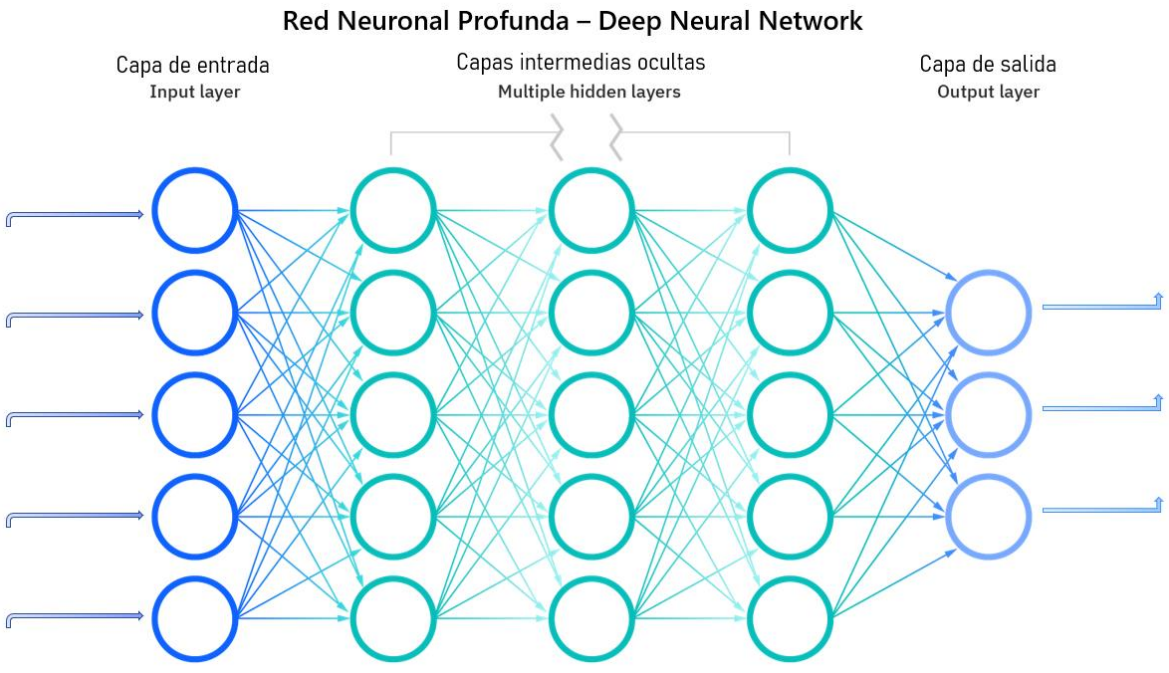


Ilustración 3 Modelo genérico de una red neuronal artificial profunda. Fuente: adaptado de [7].

Se dispondrá de desarrollos de modelos de Aprendizaje Profundo, para tener como producto resultante, un sistema capaz de traducir tanto señas estáticas (como lo son la mayoría de las letras del abecedario), como no estáticas (aquellas que ameritan una serie de movimientos o transiciones para expresar la idea o concepto). Esto se conseguirá tras unificar los conceptos detrás de las estructuras que permiten identificar señas y disponer de un componente de "*memoria*" con el cual se recordarán (principalmente) las posiciones de las manos a lo largo del tiempo, obteniendo por salida el texto en español de la traducción tras captar una seña dada.

Justificación

La razón de ser de un sistema de este tipo sería la gran oportunidad que podría llegar a ser para Panamá el que exista una comunicación más efectiva (y eficiente) entre los grupos previamente mencionados. El eventual impacto tendría consecuencias positivas en varios sentidos, destacando la importancia en términos de inclusión de un sector de la sociedad.

Por otra parte, la escalabilidad es un elemento interesante que remarcar en este proyecto. Debido a que el desarrollo pudiese crecer o transformarse en lo que respecta a alcance por señas traducibles, despliegue del sistema, mejora de la interfaz, aumento de la disponibilidad del sistema, creación de nuevos módulos (como un conversor de texto a voz), entre otras posibilidades; siendo esto innegablemente una bondad relevante.

Claramente, el que no se haya realizado un desarrollo de este tipo en Panamá (más allá de un sistema que traduce las señas estáticas con las que se representan las vocales a LSP [8]), es factor para que sea tomado en cuenta como un objeto de estudio de valor. Este fenómeno de interés se hace incluso más llamativo cuando el enfoque que se quiere dar es el de traducción, especialmente, de señas no estáticas; lo cual cuenta con poca investigación, de acuerdo con la limitada documentación que existe al respecto en el marco del estudio de reconocimiento de lenguas de señas (desde la perspectiva del Aprendizaje Profundo).

La naturaleza del proyecto, dada la disciplina con la cual se lleva a cabo, facilita la puesta a disposición del producto final; opuesto a, por ejemplo, los guantes con sensores para

reconocimiento de señas [9], cuya “masificación” representaría una inversión importante, que muchas empresas o estados quizás no estarían dispuestos a hacer, y se daría con un producto que muchas personas podrían simplemente no conseguir costearse. La factibilidad de este proyecto, o de potenciales proyectos subsecuentes, abarca puntos como la puesta a disposición del producto o servicio (en forma de aplicación móvil, por ejemplo), que sería altamente competitiva; asimismo, en términos de recursos computacionales, se puede explotar el almacenamiento de datos, el proceso de entrenamiento y el uso (ejecución) del sistema sin necesidad de capacidades de procesamiento desmedidas, a través de alternativas no locales como servidores de nube. Esto último se determina dado el estudio de proyectos similares (en especial de lo que respecta a uso de estas tecnologías), y en respuesta a modificaciones que se pueden ir haciendo a nivel de código y de estructura, para aprovechar de manera más eficiente los recursos.

Delimitación

En principio, el sistema constase de un conjunto de señas (a ser identificadas y traducidas) que permita que el proceso de diseño y desarrollo (en especial entrenamiento de los modelos de aprendizaje automático y las pruebas) sea suficientemente dinámico. Por tanto, serán consideradas señas principalmente de palabras y expresiones comunes (como saludos); la adición de algunos números, letras o conceptos adicionales responderá, mayormente, a las necesidades que vayan surgiendo a lo largo del desarrollo del proyecto (y podrían limitarse para los efectos de pruebas de modelos base).

La intención es producir un sistema que pueda reconocer tanto señas estáticas (aquellas para las cuales solo se necesita una seña que no involucra movimiento), como no estáticas (aquellas que requieren algún tipo de movimiento o gesto auxiliar, o las constituidas por varias señas). Adicionalmente, se pretende que el sistema sea capaz de trabajar con captación de datos (imágenes y videos con/de las señas) y presentación de resultados (traducción correspondiente para las señas en forma de texto) en tiempo real.

Las señas referidas corresponden a la Lengua de Señas Panameñas, y el texto escrito, al cual serán traducidas dichas señas, será en español. A su vez, los modelos a

desarrollar entrarán dentro de la disciplina de Aprendizaje Profundo; y el campo de estudio/investigación recaerá en Visión Artificial.

Objetivo General

Desarrollar un sistema capaz de reconocer, en tiempo real, señas de la Lengua de Señas Panameñas, con Redes Neuronales Profundas, presentando su traducción correspondiente en forma de texto escrito en idioma español; con el fin de establecer un canal de comunicación efectiva entre personas con discapacidad auditiva y aquellas que no manejen LSP.

Objetivos Específicos

- Determinar y analizar los algoritmos y modelos a utilizar para el desarrollo del sistema basado en aprendizaje automático con redes neuronales convolucionales (*CNN*) y redes neuronales recurrentes (*RNN*).
- Construir un detector de objetos, a ser orientado a reconocimiento de lenguas de señas, como base de un módulo del sistema propuesto.
- Diseñar y construir el módulo de aprendizaje automático con Redes Neuronales Convolucionales para reconocimiento de señas estáticas de la LSP.
- Construir un detector de acciones, a ser orientado a reconocimiento de lenguas de señas, como base de un módulo del sistema propuesto.
- Diseñar y construir el módulo de aprendizaje automático con Redes Neuronales Recurrentes para reconocimiento de señas no estáticas de la LSP.
- Implementar código para que la detección de las señas sea en tiempo real, haciendo que la traducción correspondiente permita la comunicación efectiva.

Capítulo II – Marco Teórico

Antecedentes

Estudios Relacionados

A la fecha, los principales estudios centrados en la Lengua de Señas Panameñas han tenido por producto un libro recopilatorio de apoyo en aprendizaje y enseñanza de la LSP [6], una aplicación móvil con documentación y vocabulario sobre LSP [10], y una plataforma web con una serie de traducciones de conceptos en español a su semejante en señas del LSP [11] [12], además de un sistema capaz de traducir las señas (en LSP) correspondientes a las vocales, con un alto nivel de precisión [8].

Por su parte, el campo de Visión Artificial ha tenido avances significativos como resultado de un cúmulo de proyectos e investigación. Las publicaciones que más se relacionan con este proyecto tienen que ver con Redes Neuronales Convolucionales usadas, precisamente, en el marco de traducción de señas (reconocimiento de lenguas de señas); desde luego, estos trabajos suelen estar asociados a traducciones entre *American Sign Language* (Lengua de Señas Americanas; utilizada en Estados Unidos y Canadá) e inglés, por el volumen de investigación proveniente de los países en los que conviven estas lenguas.

En el último año, entonces, se tiene, un modelo de reconocimiento de gestos (de *ASL*) con las manos entrenado con un conjunto de datos creado para ese propósito [13] y un clasificador de números (representados con señas de *ASL*) en donde se implementa un preprocesamiento especial para potenciar la precisión en la traducción [14]. Ambos casos, son enfoques en los cuales son utilizadas arquitecturas de *CNN* para traducción de señas, algo que pese a haberse intentado en varias ocasiones, sigue demostrando vigencia (razón por la cual existen publicaciones tan recientes al respecto). De cualquier forma, existe documentación relevante centrada en la misma problemática (reconocimiento de señas) en otros países; más concretamente, para otras lenguas. Se tiene, por ejemplo, un *sistema de traducción semántica de lengua de señas* (de la Lengua de Señas Árabes) usando Ontología y *CNN* [15] y una arquitectura de “red concatenada novedosa” propuesta para traducir señas (de la Lengua de Seña Bangladesí) [16]. Adicionalmente, se presentaron “*datasets”* (conjuntos de datos) para “aprovechar nuevos métodos de Aprendizaje Profundo” a la hora de trabajar con reconocimiento de señas

(orientado a Lengua de Señas Francesa, en particular, belga) [17] dada la gran cantidad de información que dificulta centrarse en los elementos más importantes (como arquitecturas o algoritmos utilizados). Existe también un sistema basado en sensores captados por un dispositivo a través del cual se recolectan los datos para realizar la traducción (de señas de la Lengua de Señas Italianas) mediante algoritmos de Aprendizaje Automático [18]. Fueron utilizados tres (3) clasificadores, de los cuales se tuvo un mejor rendimiento del focalizado en Redes Neuronales Artificiales; versus los de Máquinas de Vector de Soporte ("*Support-Vector Machine*", en inglés) y "*K-nearest*" (algoritmo que suele ser traducido como "k vecinos más próximos").

Estos últimos enfoques evidencian que se sigue buscando combatir esta problemática desde una variedad importante de ópticas, y resultan ser las *CNN* las que abanderan el grueso de los avances en el campo. Aunado a este primer tipo de Redes Neuronales, se encuentran, para los efectos, las Redes Neuronales Recurrentes. Estas se han presentado como un medio para plantear nuevas soluciones de cara a nuevos y viejos problemas, a la vez que han hecho de alternativa versus problemas previamente atendidos (hasta cierto punto, al menos). Algunos trabajos actuales apoyados en las *RNN* tienen que ver con análisis de sentimiento (conocido también como minería de opinión) que recae en Procesamiento de Lenguaje Natural (cuyas siglas en inglés son *NLP*, de "*Natural Language Processing*"), de hecho, en [19] se trabaja en simultáneo con *CNN*; o la clasificación de series temporales / cronológicas (las cuales son un área del *ML* propiamente dicho) y pronósticos de máquinas, como en [20].

Sería inapropiado no mencionar enfoques que se dieron en su momento, o que han coexistido con los de Aprendizaje Profundo (en particular con *CNN*). Entre ellos, los de Aprendizaje Automático, como el evidenciado en [21] donde se trabajó con *ML* basado en modelos y características según la composición lingüística de los signos léxicos; u otros que siguen estando debajo del paraguas de Inteligencia Artificial, como el uso de sensores de radio frecuencia para generación de datos sintéticos [22].

Bases Teóricas

Inteligencia Artificial

El término “Inteligencia Artificial” fue acuñado en los 50’s, época en la cual surgió lo que sería la versión moderna de esta área. Son incontables las posibles maneras en las cuales se define esta locución; cabe recordar que el propio término “Inteligencia”, del cual se deriva directamente, sigue siendo tema de debate dentro de la comunidad científica. Sin embargo, la Inteligencia Artificial (abreviada “IA”, y con las siglas “*AI*” en inglés) puede ser delimitada como la capacidad de las máquinas de tomar decisiones de forma similar a la de un humano, pero mediante el uso de algoritmos y de lo aprendido con ayuda de datos. Ver Ilustración 4, con algoritmos de diferentes paradigmas.

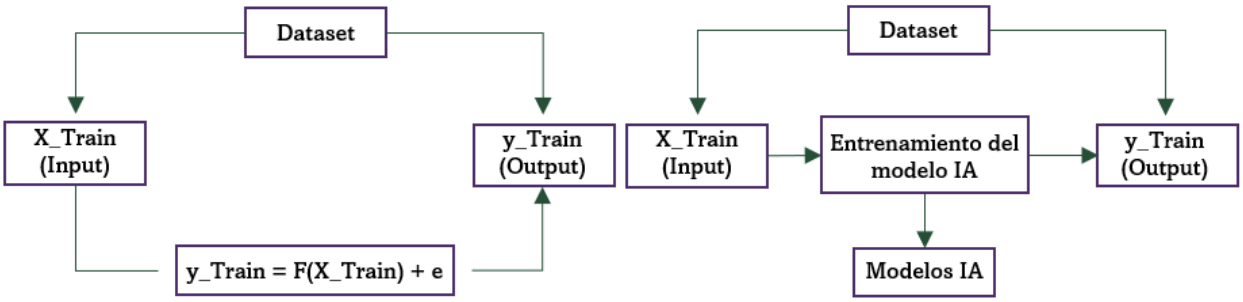


Ilustración 4 A la izquierda, el proceso de entrenamiento de modelos tradicionales; a la derecha, el proceso de entrenamiento para modelos de Inteligencia Artificial [23].

Normalmente se refiere a estas “máquinas” como “sistemas”, con el entendido que suelen constar de diversos componentes e interacciones más complejas que “simplemente” una máquina realizando una acción (como para modelos clásicos o tradicionales). Se tiene también que, una de las formas de categorizar el área es con base en dos (2) parámetros: los procesos mentales, y la conducta, haciendo distinción entre el pensar y el actuar. Una segunda categoría refiere la capacidad de emular frente al tener raciocinio. De esta manera, de acuerdo con las cuatro (4) posibles intersecciones entre estas categorías, se forma la Tabla 1 para clasificar sistemas con cierto grado de inteligencia; siendo estos: a) Sistemas que piensan como humanos. b) Sistemas que

piensan racionalmente. c) Sistemas que actúan como humanos. d) Sistemas que actúan racionalmente.

Tabla 1 Tipos de sistema según la "inteligencia" que presentan. Fuente: El Autor.

	Emulación	Racionalidad
Procesos mentales	*Sistemas que piensan como humanos*	*Sistemas que piensan racionalmente*
Conducta	*Sistemas que actúan como humanos*	*Sistemas que actúan racionalmente*

A partir de estas categorías, en las últimas décadas, se han hecho investigaciones y se han llevado a la práctica proyectos que han influenciado directa e indirectamente gran número de avances, propuestas y cambios de paradigmas en distintos países e industrias, impactando profundamente la vida de la inmensa mayoría de la población mundial. Bien sea que recaiga en una u otra, o que tenga características de más de una categoría, cada proyecto o desarrollo dentro del campo de la IA responde a intentos de parte de los investigadores o desarrolladores de emular conducta humana por medio de estos sistemas.

Hoy en día, es difícil destacar de forma superlativa alguno de los subcampos que se desprenden del área de Inteligencia Artificial, en vista, principalmente, de dos factores: 1) Lo mucho que abarca el concepto, englobando hoy día casi cualquier avance tecnológico, puesto que se busca constantemente optimizar tareas a través de la tecnología, automatizar procesos de naturaleza muy variada, e innumerables acciones que ameritan trabajar de alguna u otra forma bajo el espectro IA. 2) El constante progreso científico que, más allá del dominio de conocimiento, suele involucrar un nivel de IA; y que, en ocasiones, se percibe con un crecimiento de comportamiento al menos próximo

a exponencial. Sin embargo, subcampos como la Robótica, el Aprendizaje Automático y el Aprendizaje Profundo suelen estar presentes en las corrientes actuales de la IA.

Aprendizaje Automático

El Aprendizaje Automático se podría definir como el conjunto de estudios de la Inteligencia Artificial que busca desarrollar sistemas con capacidad de aprender (no solo la capacidad de llevar a cabo una tarea). Es un área que ha representado gran avance en la industria tecnológica y solución a diversos problemas. Los algoritmos que se utilizan allí pueden llegar a ser poderosos, sobre todo, si el volumen de datos a considerar es significativo, y los mismos se acondicionan; dado que se basan en el reconocimiento de patrones identificados en cierto conjunto de datos. Al hacer pruebas de rendimiento, disminuir errores y, en general, optimizar el desarrollo, es posible dar con sistemas altamente valiosos gracias al Aprendizaje Automático.

El término que le dio nombre a este subcampo, Aprendizaje Automático (*Machine Learning*), fue, de hecho, acuñado no mucho después del de Inteligencia Artificial. Desde entonces, ha sido más bien de tendencia variante; habiendo sido, si se quiere, dejado de lado un tiempo por la comunidad científica. Cuando su popularidad comenzó a crecer (nuevamente) un par de décadas atrás dejó de ser interrumpida. Y es que, de alguna forma, el *Machine Learning* supo capitalizar la nueva magnitud de recursos informáticos y tecnológicos con los cuales se cuenta como sociedad; como los de almacenamiento masivo en la nube o la mejora en términos de capacidad de procesamiento de las computadoras.

Esta combinación de conceptos interesantes y poderosos, junto a sistemas complejos y potentes (y al extensivo e intensivo esfuerzo de los científicos computacionales) ha conseguido hacer que el Aprendizaje Automático haya revolucionado la forma de proceder en las ciencias computacionales, llegando a convertirse incluso en un paradigma. Esto se entiende si se hace frente a la programación tradicional con el *Machine Learning*, como en la Ilustración 5.

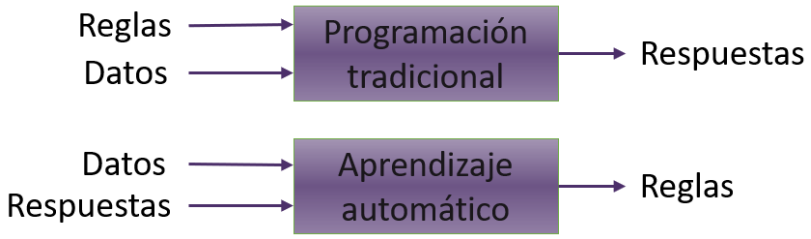


Ilustración 5 Comparación de entradas y salidas según programación tradicional (arriba) y aprendizaje automático (abajo). Fuente: adaptado de [24].

Se aprecia en la Ilustración 6, cómo en la programación tradicional el desarrollo lógico es implementado a modo de instrucciones explícitas (reglas) sobre qué debe hacer un sistema, y cómo, mediante código. Se espera, en consecuencia, que a partir de una entrada dada (en forma de datos); el sistema se valga de la lógica implementada como el algoritmo a través del cual se determinará una salida esperada (respuesta). Mientras tanto, el Aprendizaje Automático, se vale de un proceso de entrenamiento a los modelos que se desarrollan (los cuales pueden ser sistemas de por sí). Se dispone de conjuntos de datos de entrada y sus correspondientes salidas (las respuestas), "enseñando" a los modelos qué se espera a partir de ellas, en tanto que estos determinan el algoritmo (las reglas) que buscará dar la respuesta correcta con el nivel de precisión más alto posible. Por consiguiente, mucho del estudio en ciencias computacionales se ha volcado a esta versión renovada de programar; como respuesta de sus bondades. En Ilustración 6 se muestra un proceso básico de desarrollo según programación tradicional a comparar con el de Ilustración 7, para aprendizaje automático.

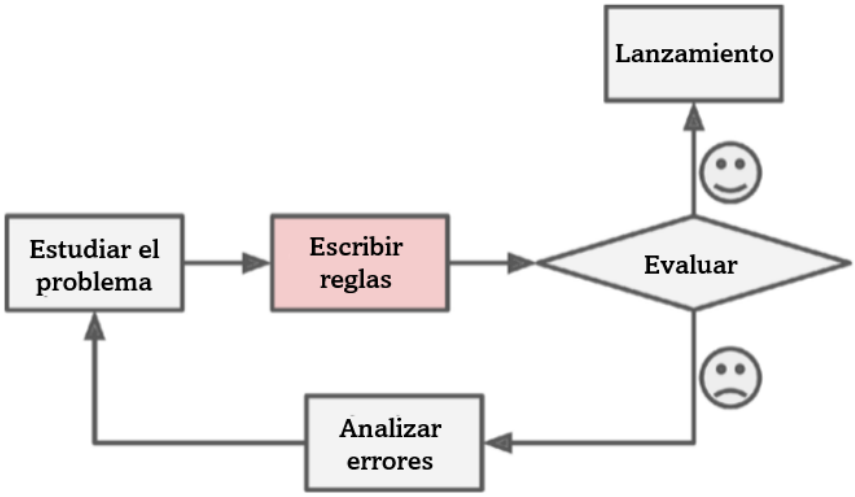


Ilustración 6 Proceso de desarrollo lógico bajo programación tradicional. Fuente: adaptado de [25].

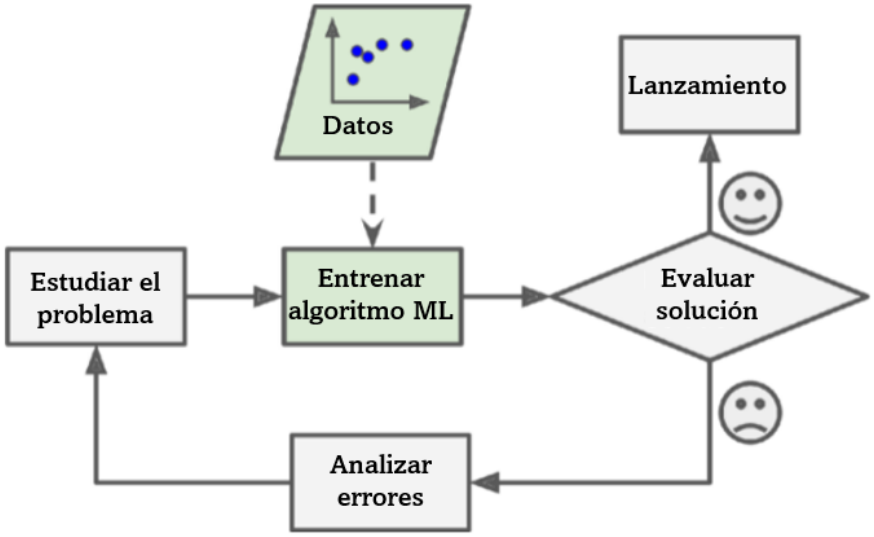


Ilustración 7 Proceso de desarrollo lógico con enfoque de aprendizaje automático. Fuente: adaptado de [25].

Entonces, en lugar de tener que escribir las reglas y evaluar el programa, un programador (desarrollador) de *Machine Learning* deberá entrenar un algoritmo (de *ML*; escogido de acuerdo con el problema y a las necesidades), apoyado en una serie de datos (usualmente de un mismo tipo de una variedad de opciones con las que se puede desarrollar hoy día), y lo que se evalúa (en el proceso iterativo de desarrollo) es la solución obtenida a partir del entrenamiento del algoritmo, por ende, del modelo.

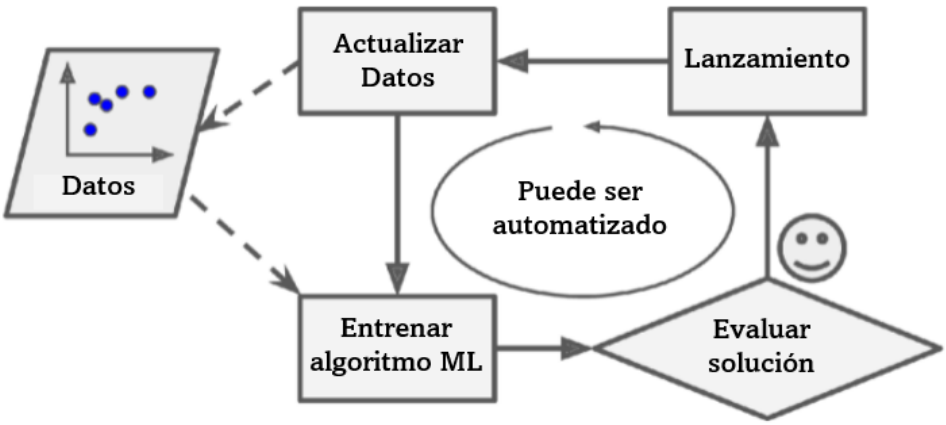


Ilustración 8 Adaptándose a cambios automáticamente. Fuente: adaptado de [25].

En Ilustración 8 se presenta un ciclo más de desarrollo (que de programación) con la importante característica de poder ser automatizado, lo cual, de hecho, dio nombre en español al campo: Aprendizaje Automático (en lugar de “Aprendizaje de Máquinas”, la que sería literalmente su traducción). Resulta que en *ML* puede llegar a ser tan expedito como retador actualizar los datos para reentrenar un modelo, lo indudable es lo provechoso de este elemento ante un amplio volumen de tareas y desafíos a los que afrontar.

Se observa, en consecuencia, cómo el Aprendizaje Automático no solo se ha hecho un lugar en sí mismo, sino que ha influenciado en aspectos y elementos de la IA, trayendo matices de complejidad a la hora de intentar subdividirla en subáreas, pero, más importante, haciéndose un ámbito abarcador de disciplinas y subconjuntos sumamente relevantes en años recientes. No sería precisamente desmesurado decir que, de las tendencias actuales de la IA [26], la mayoría se debe (total o parcialmente) al Aprendizaje Automático. Entre ellas, se encuentran la Generación del Lenguaje Natural, los Agentes Virtuales, las Biométricas, el Reconocimiento de Voz o las plataformas de aprendizaje profundo; algunas de estas, siendo disciplinas en sí mismas. Ver Ilustración 9, donde se presenta la característica de iterativo de un proceso de desarrollo con *ML*, con el beneficio de aportar al entendimiento de un problema por parte del desarrollador.

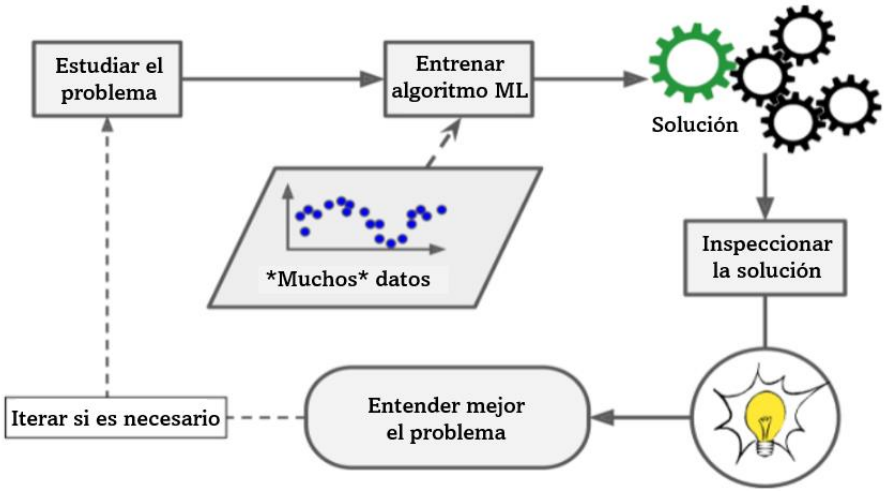


Ilustración 9 El aprendizaje automático puede impactar en cómo aprendemos y cómo entendemos los problemas. Fuente: adaptado de [25].

Visión Artificial y Aprendizaje Automático

La Visión Artificial es un campo de estudio / investigación que se centra en el diseño y desarrollo de sistemas informáticos que sean capaces de procesar información visual contenida en datos como imágenes y videos, capturando, interpretando y "comprendiendo" dicha información [27].

Similar a otros casos dentro de las ciencias computacionales, existen analogías directas respecto a la visión artificial versus la visión humana o natural. El sistema visual humano cuenta con ojos (captura), receptores en el cerebro (para acceder) y una corteza visual (procesar).

Con la humanidad en sus etapas industrial y tecnológica se ha conseguido que las máquinas imiten estas capacidades, como al capturar luz y color a través de una cámara; con el tiempo se han ampliado estas capacidades y han nacido subcampos en paralelo, adquiriendo una madurez evidenciable a través de sistemas contemporáneos. Fue, sin embargo, con el Aprendizaje Automático que se logró que las máquinas pudiesen aprender el contexto necesario para interpretar imágenes; se debe recordar que, para las computadoras, una imagen es un cúmulo de valores numéricos ordenados, y el *Machine Learning* permitió implementar algoritmos que supiesen entender estos

números en la disposición y complejidad dada. Un ejemplo de esto último se aprecia con la Ilustración 10, donde una computadora interpreta una imagen.

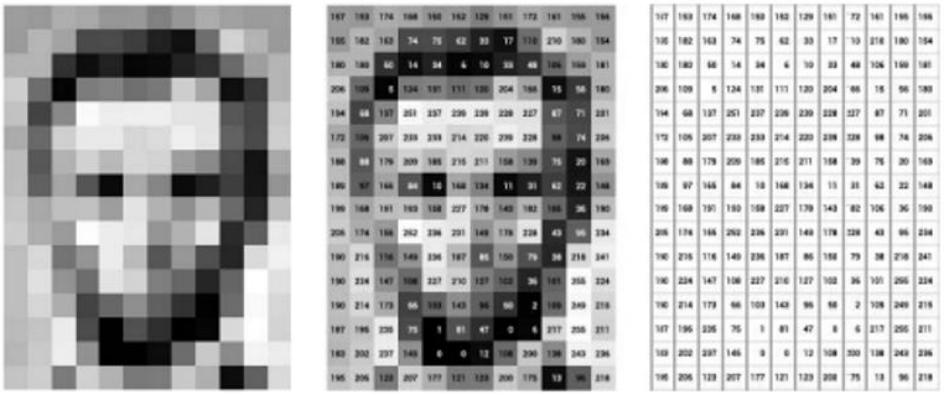


Ilustración 10 Cómo interpretan las computadoras imágenes de datos de pixeles sin procesar. Fuente: obtenido de [27].

Clasificación de imágenes

La clasificación y la regresión constituyen los problemas más importantes en el Aprendizaje Automático; en clasificación se busca predecir la etiqueta de la clase, lo cual será una entre varias (al menos dos) posibilidades en una lista predefinida. [28]. Un caso con el que con frecuencia se ejemplifica la tarea de clasificación es el de detección de "*spam*" (correo basura), en el cual se clasifica un correo (o lista de correos) como "*spam*" o "*no spam*". Sin embargo, la clasificación de imágenes responde a problemas en los cuales se deba identificar a qué categoría (clase) pertenece un elemento de entrada con forma de dato de imagen (o video); como lo podría ser la imagen de una margarita, tal como la entrada visual que se tiene en Ilustración 11, donde una persona identifica la especie (categoría) a la que pertenece esta flor.

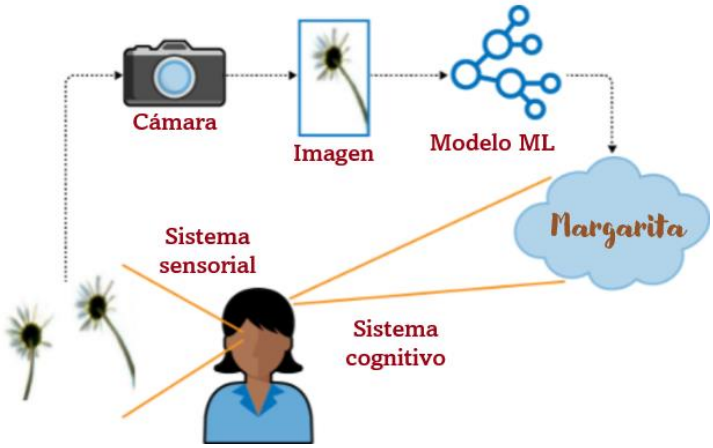


Ilustración 11 Un modelo de aprendizaje automático clasificador de imágenes "imita el sistema cognitivo humano". Fuente: adaptado de [29].

- Clasificador de imágenes: Es simplemente cualquier modelo o sistema que lleve a cabo tareas de clasificación de imágenes. Existen clasificadores binarios (o "de clasificación binaria"), como el del ejemplo del *spam*, que no es clasificador de imagen; y clasificadores multiclase, como un modelo que clasifique plantas de un grupo de tres o más clases posibles (con sus respectivas etiquetas), lo cual sí sería un clasificador de imágenes. Otro clasificador multiclase sería aquel que logre distinguir entre números del sistema decimal, como el mostrado en Ilustración 12. También existen modelos "*multilabel"* ("*label*" por etiqueta), que son los que pueden tener por salida, varias etiquetas en simultáneo, como un clasificador que "vea" tanto un carro como un semáforo y una señal de tránsito en una misma entrada.

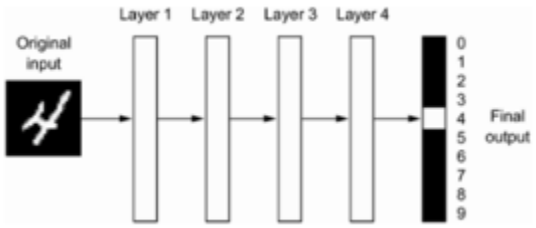


Ilustración 12 Clasificador de dígitos. Fuente: obtenido de [24].

Aprendizaje Profundo

Entre las aplicaciones o disciplinas que se desprenden del *Machine Learning* (comúnmente en unión con otras ramas de Ciencias Computacionales y afines), destaca en relevancia actual el Aprendizaje Profundo (*Deep Learning*). Ver Ilustración 13, con una representación de la relación entre estas subáreas de IA. Este se vale de técnicas y algoritmos basados en redes neuronales artificiales profundas ("*Deep Neural Networks*", o simplemente "*DNN*"), las cuales ofrecen una forma jerarquizada de aprender. Esos patrones por identificar (por seguir siendo aprendizaje automático), en el marco del *Deep Learning*, ahora vendrán dados en capas (que es como se representan estos algoritmos [30]) cada vez más "profundas", es decir, se irán aprendiendo cosas de más simples a, considerablemente, más complejas. Una versión de esto se alcanza a ver en Ilustración 14, para un clasificador de dígitos.

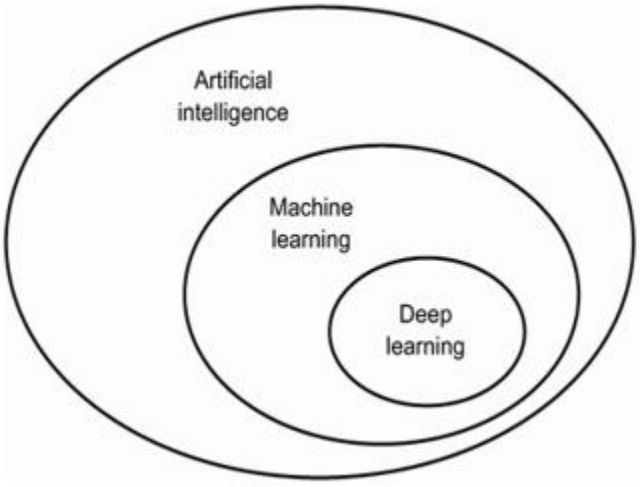


Ilustración 13 Diagrama representando cómo el aprendizaje profundo está contenido en el aprendizaje automático, y este, a su vez, está incluido dentro de la inteligencia artificial. Fuente: obtenido de [24].

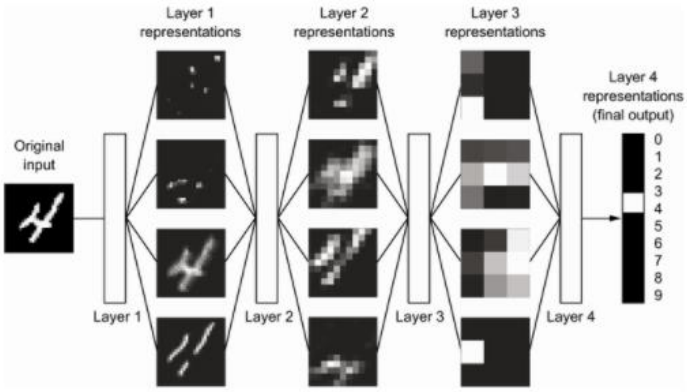


Ilustración 14 Representaciones de datos aprendidas por un modelo clasificador de dígitos. Fuente: obtenido de [24].

Redes Neuronales Profundas

Las redes neuronales profundas fungen de centro del Aprendizaje Profundo; por lo cual, hay que entenderlas partiendo de la noción que les dio origen.

Redes Neuronales Artificiales

Buena parte de la terminología (y conceptos) del Aprendizaje Profundo nace por analogía directa de redes neuronales naturales o biológicas; más importante aún, el *Deep Learning* como disciplina proviene de conceptos de estas redes neuronales (las “reales”). Ver Ilustración 15, donde se presenta una neurona biológica; y en Ilustración 16 se tiene una versión con mayor abstracción. Las neuronas biológicas, que están interconectadas dentro del cerebro, procesan y transmiten información entre sí a través de señales eléctricas y químicas. Mientras tanto, en las redes neuronales artificiales, las neuronas de una capa, comúnmente representadas por nodos, se comunican con sus capas anteriores y posteriores; según las conexiones establecidas en el diseño de la red neuronal y de acuerdo con los algoritmos que deciden cómo se van a comportar (por ende, comunicar).

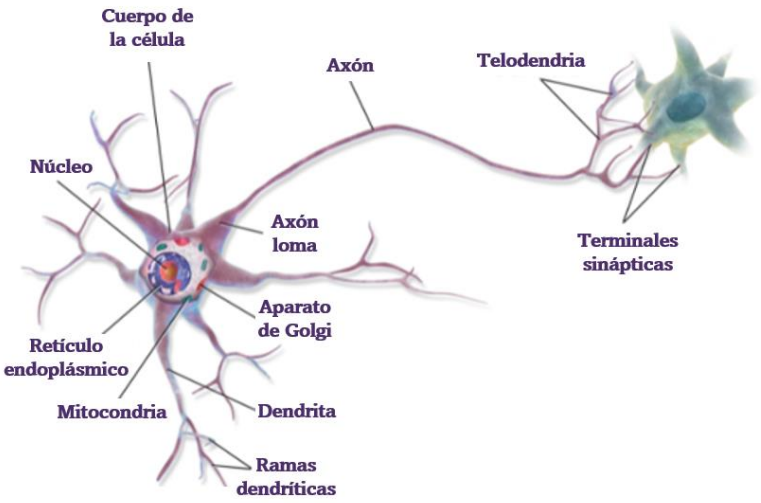


Ilustración 15 Neurona biológica. Fuente: adaptada de [25].

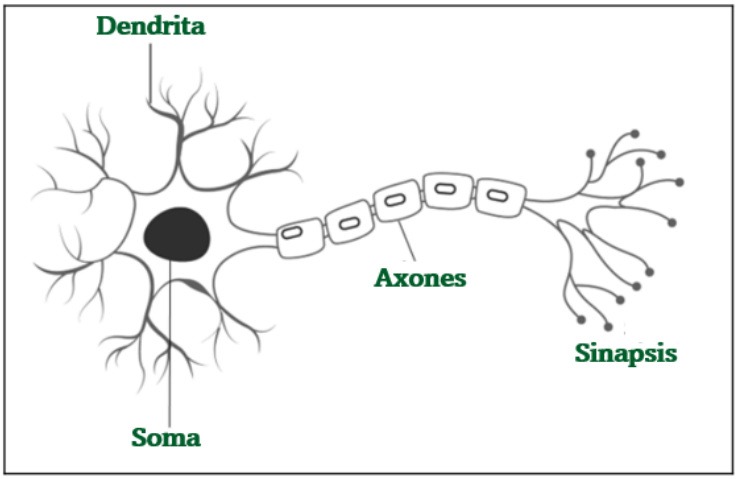


Ilustración 16 Representación simplificada de una red neuronal biológica con algunas de sus partes. Fuente: adaptada de [31].

Estos modelos computacionales inspirados en el cerebro humano comparten, por supuesto, múltiples características con sus pares naturales; ver Tabla 2. La denominación de estos elementos varía de caso a caso, y se tiene que las fundamentales coinciden con las propias partes de la unidad funcional básica del sistema nervioso humano (animal): la neurona.

Tabla 2 Partes de una red neuronal biológica y el elemento que inspiraron para redes neuronales artificiales. Fuente: El Autor.

Redes Neuronales Biológicas	Redes Neuronales Artificiales
Soma (cuerpo de la célula)	Unidad neuronal artificial (representada como nodo)
Dendritas	Entradas (ponderadas)
Sinapsis	Pesos o interconexiones
Axones	Salidas (unidades de salida)

A partir de estas partes se puede entender en buena medida cómo funciona una neurona, dado el funcionamiento de cada una de ellas, según coincidencias en Tabla 3.

Tabla 3 Funciones principales de una neurona en el contexto de redes neuronales artificiales. Fuente: adaptado de [32].

Función de la dendrita	Recibe señales provenientes de otras neuronas
Soma (cuerpo de la célula)	Suma todas las señales entrantes para generar la entrada.
Estructura del axón	Cuando la suma alcanza un valor *umbral* (“threshold”), la neurona se dispara y la señal viaja por el axón hasta las otras neuronas.
Funcionamiento de las sinapsis	Punto de interconexión de una neurona con otras. La cantidad de señal transmitida depende de la fuerza (pesos sinápticos; “*synaptic weights”*, en inglés) de las conexiones.

Aunque se ha propuesto que se les denomine “*nodes*” (nodos) o “*units*” (unidades) debido a que, por funcionamiento, son términos más precisos; sigue llamándose con relativa frecuencia al componente fundamental de una red neuronal “neurona” (artificial).

De cualquier forma, se llega a comprender en parte el funcionamiento de toda una red neuronal, partiendo del de la unidad linear (un nodo con “una sola” entrada y una sola salida), como la de Ilustración 17. Estas unidades son capaces de realizar ciertos cálculos, como se distingue en Ilustración 18.

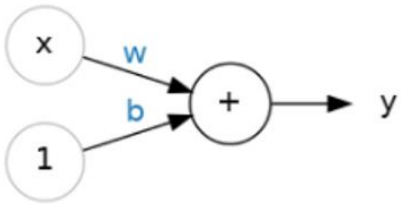


Ilustración 17 Unidad lineal de la forma y=wx+b que recuerda a la ecuación de la línea. Fuente: presentada en [33].

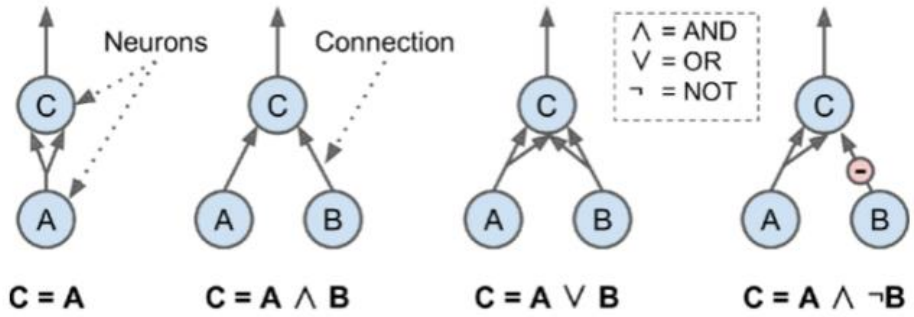


Ilustración 18 Redes neuronales artificiales haciendo cálculos simples. Fuente: presentada en [25].

Conviene describir con mayor detalle ciertos elementos, algunos de los cuales también se aprecian en lo anterior. A continuación, una serie de términos de álgebra, matemáticas, y redes neuronales, para entender los fundamentos tras las RRNN.

- Entrada (ponderada): Representada en Ilustración 17 por “x” es un valor o dato que puede variar en forma. Un número, una serie, o una imagen, podrían ser entradas de un modelo (y de los nodos de cada capa). Por cómo procesa la información una computadora, siempre se habla de datos con los cuales pueda lidiar: números, en última instancia; pero que pueden venir en estructuras como vectores, matrices y tensores.

- Peso sináptico (coeficientes): Para la unidad lineal en Ilustración 17 “w” sería el peso que tiene la conexión entre la entrada x y la neurona, todo valor (de entrada) es multiplicado por el peso de la conexión por la cual pasa. Sin importar la cantidad de entradas de una red neuronal, sus valores deberán ser multiplicados por sus pesos, que *transformarán* los valores. Un modelo de red neuronal artificial aprende al modificar estos pesos como propone Ilustración 19.

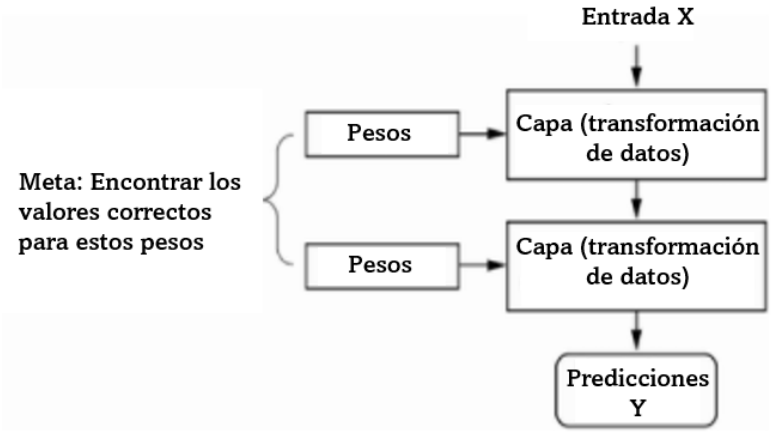


Ilustración 19 Meta de una red neuronal para poder predecir. Fuente: adaptado de [30].

- Sesgo (intercepción): El “*bias*” es un tipo de peso sináptico sin entrada asociada, que permite que la neurona modifique la salida independientemente de sus entradas. Para 17, se tiene que $1 * b = b$; por lo que el valor que llega a la neurona es b. Para la unidad lineal con un solo peso de entrada, se tiene también el sesgo, tal como si fuese cualquier otra red.
- Sumatoria: O sumatorio, Σ, representado como + en el diagrama Ilustración 20 es simplemente la suma de la multiplicación de los pesos por sus entradas asociadas, más el sesgo.
- Salida: Simbolizada y. Tras los cálculos por efectos de los pesos y el *bias* (es decir, la sumatoria), se tiene para un nodo dado una salida. Salvo la última salida, que sería el valor predicho o la salida del modelo, toda salida dentro de una red neuronal también es entrada, para otro u otro(s) nodo(s).

Los modelos de una sola neurona son llamados “modelos lineales”. La unidad lineal tiene matemáticamente la misma forma que la ecuación de la línea $z = mx + b$, con “z” siendo la línea (notación cambiada por fines prácticos), m los pesos (coeficientes), “x” la entrada (única en este caso) y “b” el sesgo (intersección). Ver Ilustración 21, para mayor entendimiento.

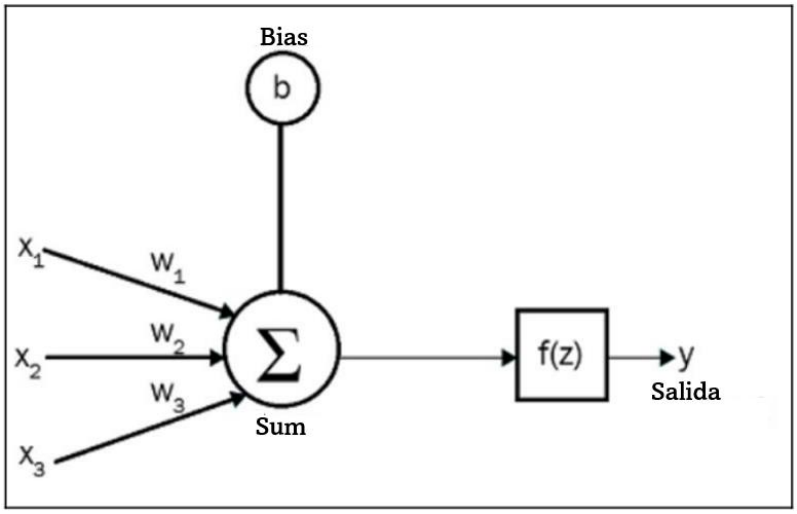


Ilustración 20 Red neuronal con varias entradas procesadas por una única neurona. Fuente: adaptada de [31].

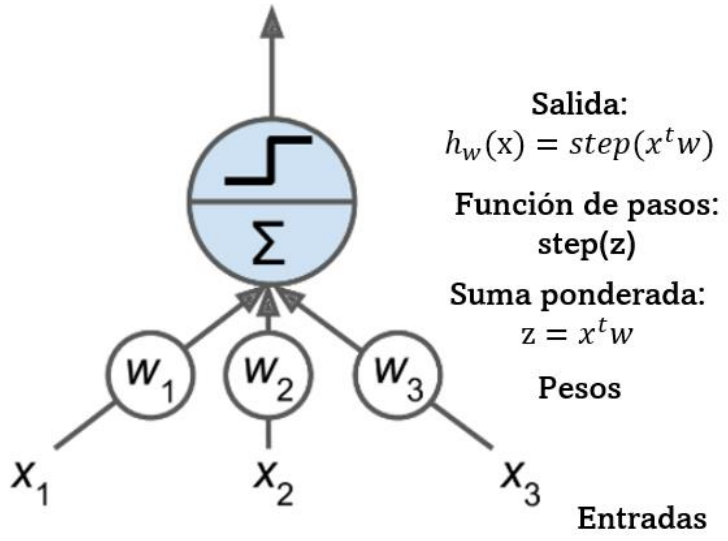


Ilustración 21 Unidad lógica umbral: una neurona artificial que computa una suma ponderada de sus entradas luego aplica una función de paso. Fuente: adaptada de [25].

- Capas (“layers”): Conviene ver las capas de una red neuronal en un momento u otro como alguna de las siguientes tres.
    - Una serie de operaciones con una operación no lineal al final [30].
    - Una topología de red que realiza algún tipo de transformación (pequeña, si se quiere) de datos dentro de la red.
    - Una estructura en la cual se organizan neuronas que no interactúan entre sí, pero sí lo hacen con neuronas de otras capas a las cuales está conectada la que las contiene.

Con base a esto, y haciendo énfasis en el último concepto planteado, se pueden categorizar las capas, o nombrar las que típicamente se encuentran en una red neuronal artificial.

a) Capa densa: Una “*Dense layer*” es cuando se tienen unidades lineales con un mismo conjunto de entradas; en otras palabras, es aquella cuyas neuronas están conectadas (mediante aristas, según la representación más común) a todas las de la capa inmediatamente anterior, de allí su nombre alternativo “*Fully-connected layer*” (capa completamente conectada).

b) Capa de entrada: Es a través de la cual se proveen los datos de entrada de la red; en esta, que es única, no se llevan a cabo cálculos. Cada entrada (representada por neuronas) tendrá influencia en determinar (predecir) la salida de la red neuronal.

c) Capa oculta: Procesa la entrada recibida por parte de la capa de entrada; toda capa de este tipo es responsable por el aprendizaje de la representación de los datos, e identifica los patrones para dar con la salida. Estas capas también realizan la extracción de las características (“*features*”) de las entradas; de haber varias, cada una se encarga de un tipo o grupo de características. Usualmente, más de una en la práctica; conceptualmente se puede decir que una red neuronal es “profunda” cuando tiene al menos una capa de este tipo.

d) Capa de salida: Es la última capa en una red, y es en la que se alojan los resultados de los cálculos (el procesamiento) realizados a lo largo de la(s) capa(s)

oculta(s). Puede tener una única neurona (como en modelos de regresión) o varias (ejemplo, 2, para un clasificador binario).

En Ilustración 22 se alcanzan a apreciar capas que estarán en toda red neuronal profunda (típicamente con varias o muchas ocultas).

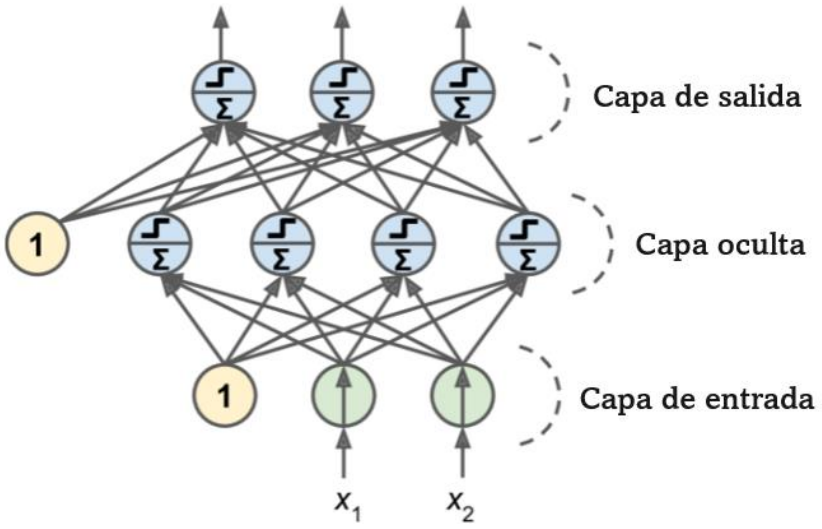


Ilustración 22 Arquitectura de un perceptrón multicapa con dos (2) entradas, una (1) capa oculta de cuatro (4) neuronas y tres (3) neuronas de salida (representando neuronas bias explícitamente). Fuente: adaptado de [25].

Cuando nos encontramos con la unidad lineal, se está frente a un modelo de regresión lineal. Se rompe con la linealidad (digamos, se imprime “no linealidad”) al resultado, “z”, al considerar la función de activación. Otro elemento importante es el apilado de capas, y por la manera más común en la que se hace esto, se suele trabajar con modelos “secuenciales”, que son pilas lineales de capas. La no linealidad sumada a varias capas apiladas, son una combinación que trae consigo incontables posibilidades. En ese sentido, otra serie de componentes de peso, siguen a continuación.

- Función de activación / transferencia: En general, cualquier función monótona (que conservan el orden dado) y no lineal, puede ser una función de activación [30].

- Función sigmoide: La “*Sigmoid activation*” es la función de activación que imita más fielmente a las neuronas de activación (de las redes neuronales biológicas), ya que, mapea entradas para valores dentro del rango [0, 1] (pensar en esto en términos porcentuales); lo cual se asemeja a la necesidad de alcanzar cierta “energía de activación”, gracias a las entradas que recibe, para enviar en ese momento señales a otras neuronas. Ver esta función en Ecuación 1.

Ecuación 1 Función sigmoide

$$S(x) = \frac{1}{1 + e^{-x}}$$

- Rectificador: Es una función de activación comúnmente usada en redes neuronales profundas, en particular, en visión artificial y reconocimiento de voz.
- ReLU: La unidad lineal rectificada es aquella (unidad) que emplea la función “rectificador”, suele darle nombre a la función en sí, tanto así que es mayormente llamada “*Rectified Linear Unit function*”. Básicamente, para las entradas cuyos valores son menores a cero (0), les asigna cero; en tanto que para aquellos mayores o iguales a cero, resulta el mismo valor; en otras palabras, “descarta” los valores negativos, o, solo considera los valores positivos.
- Función SoftMax: La función exponencial normalizada es esencialmente una generalización de la función sigmoide o función logística. Para un número posible de opciones, esta función da las probabilidades de que estas sean las salidas, por lo tanto, la suma de estos valores (que van de cero a uno) es igual a uno. Es empleada en métodos de clasificación multiclase y habitualmente se usa como función de activación de la capa de salida.

La Ilustración 23 presenta una versión de una red neuronal con capas intermedias y de salida específicas.

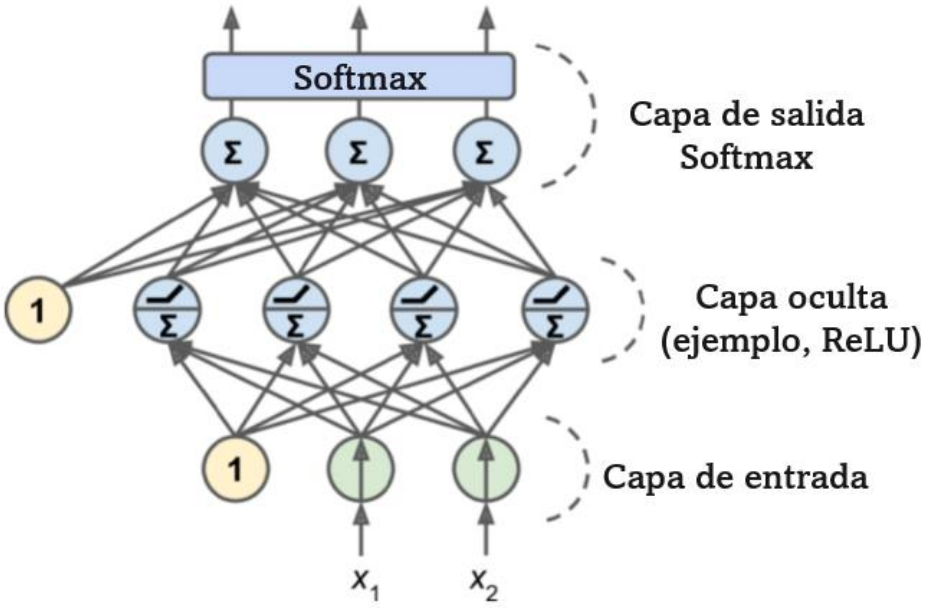


Ilustración 23 MLP moderna de clasificación; incluyendo ReLU y softmax. Fuente: adaptado de [25].

Ya habiendo definido elementos de la estructura y construcción de una red neuronal (cómo se organizan y comunican las neuronas, etc.), compete ahondar en la característica de "capacidad de aprender". Respecto a esto, se revisan los siguientes conceptos que dan pie al proceso de entrenamiento por medio del que una red neuronal "aprende".

- Objetivo: El "target" es aquel valor con el que se espera dé un modelo de *DL*. En el proceso de entrenamiento, el "*target*" es conocido, pues se consta con un *dataset* que tiene lo que serían las entradas y sus salidas asociadas; en caso de no tener las salidas, se conocen, y se definirán antes de entrenar al modelo (como cuando se etiqueta la imagen de un perrito, estableciendo el target "perro").
- Características: Los "*features*" son aquellas propiedades que definen una entrada. Puede ser simplemente un valor numérico, como también una serie de valores complejos que se espera que el modelo desarrollado extraiga de una imagen de un carro, o tal vez, datos que corresponden a un cliente del que se quiere predecir cuánto gastará en una sucursal de la empresa. La Ilustración 24 ejemplifica un modelo que recibe más de una entrada.

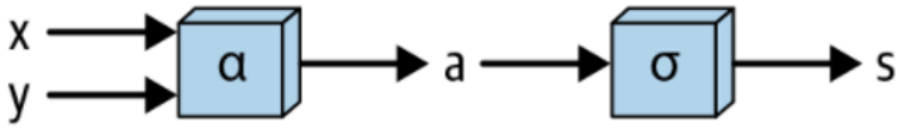


Ilustración 24 Función con múltiples entradas [30].

- Ajuste: El "*fit*", que puede ser traducido como el proceso de encajar, acoplar, engranar o ajustar, entre otros, es la técnica de Aprendizaje Automático (también utilizada en Aprendizaje Profundo) que permite "ajustar" una entrada a una salida. La entrada está expresada en términos de sus características, y la salida es el objetivo, que suele tener forma de etiqueta. Este es el proceso con el que se le "enseña" a un modelo de *DL* qué es un perro (una imagen de uno por entrada, y la salida "perro"), o que cuatro (4) es el doble de dos (2). Así, se va ejemplificando (idealmente con un alto números de pares entrada-salida), y se espera, si fue bien entrenado, que el modelo generalice, y prediga o determine una salida precisa. Para los ejemplos anteriores se podría estar esperando que el primer modelo determine si, efectivamente, la imagen dada (no vista, una vez en ejecución) es la de un perro o no (o si es un perro en lugar de un gato, o entre una lista de animales); mientras que en el segundo modelo lo que se desea es que sea determinado el algoritmo para doblar un número (es decir, la función $f(x) = 2x$).
- Datos de entrenamiento: El "*Training set*" es el conjunto de datos a utilizar para entrenar a una red neuronal (profunda, es decir, un modelo de *Deep Learning*). Se debe dividir en dos partes iguales, la que corresponde a la entrada y la de la salida. Se tiene por convención llamar a estos conjuntos de datos (a nivel de código se les asigna estos nombres a los espacios de memoria) "*X_train*" y "*y_train*".

La porción del nombre "*train*" viene del inglés para "entrenar", mientras que "X" y "y" provienen del uso típico en matemáticas para nombrar variables. La "X" suele ser la variable independiente, y en este caso, se refiere a valores no alterados, o que no deben ser procesados; por tanto, "*X_train*" es el conjunto de todas las entradas y el uso de la mayúscula simboliza que es (para casi cualquier caso de uso) un conjunto de conjuntos,

ya que son varias las características a considerar. Por su parte, “y”, conocida matemáticamente como la variable dependiente (más usada; recordar $y = f(x)$), hace referencia a que la salida, justamente, “depende” de la entrada; por esta razón, “*y_train*” es el nombre que mayormente se le asigna al conjunto de datos que son las salidas enlazadas a sus respectivas entradas, y el uso de la minúscula viene dado por ser un único elemento por salida (como lo es una cantidad, un porcentaje o una etiqueta).

- Testing set: El proceso de entrenamiento no se limita (o no debería) a “enseñar” al modelo, sino también a probarlo. De hecho, en el ciclo de desarrollo de cualquier modelo de *Machine Learning* supervisado (o de “aprendizaje supervisado”) es de esperar que este par de pasos (y otros) sea llevado a cabo en varias ocasiones. Al modelo se le enseña con ejemplos, es decir, se le dan pares entrada-salida; también existen estos pares para ponerlo a prueba, de esta forma, se sabrá qué tan bien le va (en términos de desempeño) cuando es sometido a “*data*” (datos) que no conoce (pues nunca había “visto”).

Este conjunto se *puede* dividir en subconjuntos iguales con (casi siempre) los nombres “*X_test*” y “*y_test*”. La concepción de estas denominaciones es análoga a la de los del *training set*: “*test*” por ser usados para prueba(s), “*X*” por ser conjunto de valores de entrada establecidos (por el desarrollador o por razones inherentes a los objetos de entrada), y “*y*” por ser valores únicos que responden a las entradas, y que deberían coincidir con los determinados por el modelo tras todo el procesamiento ocurrido dentro de la red neuronal. Los valores de “*X_test*” se le “pasan” a la red neuronal para que haga los cálculos, luego, se contraponen los resultados predichos con los resultados reales (en *y_test*), teniendo que darle lectura a la precisión con la que cuente en ese momento el modelo.

Ahora bien, para proyectos más completos (como aquellos a partir de los cuales se quiere crear y lanzar una aplicación móvil o web), este conjunto se excluye de la parte del proceso que corresponde a entrenamiento en sí, y se usa al final como prueba (o serie de pruebas) del rendimiento “final” del modelo; para esto, el conjunto de prueba *no presenta* los resultados reales, es decir, no se etiqueta una entrada. El “*testing*” *puede* ser llevado a cabo con datos recolectados (almacenados o no) en ejecución.

- Validation set: Medir la calidad (el rendimiento) del modelo para poder ir mejorándolo, al hacer pruebas y comparar alternativas, es indispensable para obtener un modelo útil que pueda ser lanzado como aplicación o sistema. El conjunto de datos para validación es uno excluido del proceso de construcción del modelo, pero presente durante todo el proceso de entrenamiento.

El entrenamiento de una red neuronal es iterativo. Por varias (puede llegar a ser un gran número) iteraciones, el modelo va refinando su "conocimiento"; esto lo hace gracias a la *data* de entrenamiento, y para esta se va calculando qué tan bien lo está haciendo el modelo. En este entendido, se tienen varios puntos clave para el entrenamiento, como ir viendo cómo mejoran las predicciones del modelo; sin embargo, se estarían dejando sin considerar otros elementos de valor si no se trabajase con el conjunto de validación. El *validation set* lo conforman datos que no ha visto el modelo, pero por cada iteración, mientras (idealmente) va mejorando su precisión versus el conjunto de entrenamiento, también se evalúa la precisión versus el conjunto de validación; esto *valida* que en efecto el modelo esté encontrando los patrones adecuados (y no identificando patrones incorrectos).

Entonces, el *validation set* se divide en "*X_valid*" y "*y_valid*" al igual (como era de esperar) que los dos conjuntos anteriores. "*X_valid*" es la *data* nueva de entrada con la que se validará junto al "*y_valid*" como etiqueta, que es la salida real. Conforme el modelo vaya entrenando, se irá evaluando (simultáneamente) con este set (distinto) si ciertamente la RN está tomando en cuenta características valiosas, enriqueciendo el entrenamiento.

- Métrica de monitoreo: Es cualquier métrica que ayude a evaluar la calidad de un modelo. La precisión predictiva (qué tanto "atina" un modelo) es la métrica más utilizada.
- Función de pérdida: La "*loss function*" es una función que mapea un evento o valores de una o más variables a un número real. En Aprendizaje Automático (por ende, también en *DL*) es la técnica para evaluar el rendimiento de nuestro algoritmo o modelo. Sabiendo esto, y en el contexto de entrenamiento del modelo, es fácil intuir que esos "valores de variables" que mapea son los de las pérdidas

que se tienen al calcular el error; es decir, la función ayuda a saber cuán lejos está la predicción del valor real.

Esta función es a través de la cual se mide la calidad del modelo en el proceso de entrenamiento (tanto para la *data* de entrenamiento como para la de validación) con base en la métrica de monitoreo (de nuevo, la precisión predictiva). En un problema de optimización se busca minimizar la función de pérdida, cuyo proceso de cálculo se ve en Ilustración 25. Un buen modelo es aquel capaz de calcular unos pesos suficientemente buenos, de allí que en Ilustración 26 se muestre una representación que pondera a este elemento como fundamental.

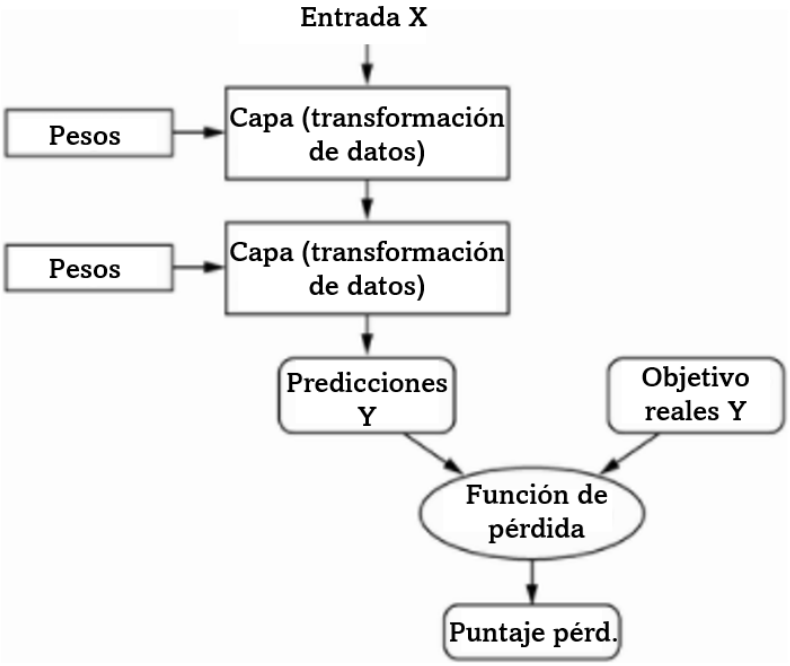


Ilustración 25 Cálculo de puntaje de pérdida. Fuente: adaptado de [24].

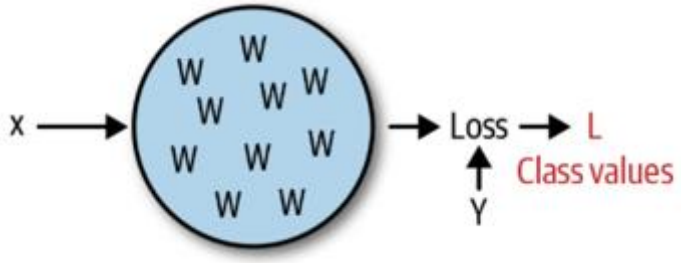


Ilustración 26 Abstracción sobre una red neuronal con pesos. Fuente: adaptado de [30].

MAE: El “*Mean Absolute Error*” (error absoluto medio) es una de las funciones de pérdida más comúnmente utilizadas en problemas de regresión [33]. Esta medida enfrenta los valores de dos variables continuas, calculando la disparidad (diferencia) mediante una sustracción entre el valor calculado y el observado. En problemas de Aprendizaje Automático es utilizada en series de datos relativos a un mismo fenómeno; siendo el total de una pérdida MAE la media de todas estas diferencias absolutas.

Cross-entropy: La “entropía cruzada” (ver en Ecuación 2) es una de las funciones de pérdida comúnmente utilizadas en problemas de clasificación. Busca calcular la entropía total entre dos distribuciones de probabilidad. Se presenta en Ecuación 3 el tipo de probabilidad que utiliza esta función.

Ecuación 2 Función de pérdida entropía cruzada.

$$L = \frac{-1}{n}\sum_{i=1}^{n} y_i \log(p_i) + (1 - y_i)\log\,(1 - p_i)$$

Ecuación 3 Probabilidad de una clase, modelada con un modelo logístico.

$$P = \frac{1}{1 + e^{-(ß_0 + ß_1 x + \cdots + ß_n X_n)}}$$

Para Ecuación 2 se puede considerar un modelo con salida de probabilidad entre 0 y 1 (que corresponden, respectivamente, a 0% y 100%), necesitando, entonces, la probabilidad (Ecuación 3) del dato “i” y la etiqueta y del mismo dato. La ecuación se emplea para los elementos “i” que van desde 1 hasta “n”. El valor “yi” corresponde a la etiqueta, mientras pi es la probabilidad del modelo; cuando se aspira a una probabilidad del 100 por ciento, y considerando un solo elemento, la ecuación pasa a tener la forma siguiente.

Ecuación 4 Función de pérdida tras haber sustituido los valores.

$$L = -1\log(p_i)$$

Y cuando, también para un solo elemento, se tiene una probabilidad del 0 por ciento, la forma de la ecuación cambia como sigue.

Ecuación 5 Función de pérdida cuando se tiene una probabilidad del 0% (para un elemento).

$$L = -\log(1 - p_i)$$

Si, por ejemplo, un clasificador múltiple con tres salidas posibles [‘Lobo’, ‘Perro’, ‘Gato’] les establece valores de probabilidad de [0.3, 0.6, 0.1], la estimación (inferencia) tendrá que enfrentarse a la ecuación $Pérdida\_clase\_X = -p(X).logq(X)$ donde *p(X)* es la probabilidad de la clase “X” en objetivo y *q(X)* es la probabilidad de la clase “X” en predicción. Se tendría, entonces, para un 60% de probabilidad (y esperando, claramente, una precisión del 100%), una pérdida como sigue:

$$Pérdida\_clase\_Perro = -(Perro).logq(Perro)$$

$$Pérdida\_clase\_Perro = -1.log0.6$$

$$Pérdida\_clase\_Perro = 0.222$$

Si en lugar de 60% se determinase una probabilidad para la clase Perro del 96% se tendría el siguiente resultado.

$$Pérdida\_clase\_Perro = -1.log0.96$$

$$Pérdida\_clase\_Perro = 0.018$$

Para los efectos, si fuese un resultado, en contraste, tan pequeño como un 3%, la clase estudiada tendría la siguiente pérdida.

$$Pérdida\_clase\_Perro = -1.log0.03$$

$$Pérdida\_clase\_Perro = 1.523$$

Evidenciando el comportamiento de la ecuación, y, por ende, del algoritmo cuando se utiliza en un modelo: el error será muy pequeño, si la probabilidad es muy alta cuando se espera que sea un 100 por ciento, y será considerablemente alto (el error) cuando la probabilidad de la predicción sea baja, habiéndose esperado el 100 por ciento.

De vuelta al primer ejemplo, en el cual la salida arrojaba que la clase perro representaba el 60% de la estimación versus las tres etiquetas consideradas; se tiene para la clase gato un 10% de probabilidad. A efectos de un clasificador para el que se espera dé por salida la clase perro (porque se le presentó una imagen de un animal de esta especie), se tiene, a su vez, que no es posible que sea un gato (ni un lobo, pues es solo una de tres especies posibles). Por lo que la pérdida de la clase gato será.

$$Pérdida\_clase\_Gato = -0.log0.1$$

$$Pérdida\_clase\_Gato = 0$$

Igual para el caso de la clase Lobo. Por lo que se entiende que, si el objetivo es [0, 1, 0] para [‘Lobo’, ‘Perro’, ‘Gato’], pues el valor cierto es el de etiqueta perro por el tipo de imagen, la predicción tendrá una forma $[y1, y2, y3]$, y la pérdida una forma $[0, p, 0]$, cuya suma sería la pérdida total para esa imagen en ese momento (por lo que sepa el modelo entonces).

La razón detrás del uso de la entropía cruzada es no darle mayor importancia (en absoluto, si se quiere) a las probabilidades, sean altas o no, de que en una imagen (en este caso) haya algo que no corresponde a la realidad (como lobo en lugar de perro). Es decir, solo importa qué tan correctamente identifique la clase presente en la imagen; y el peso del error (pérdida) no es de comportamiento lineal, puesto que la ecuación empleada se basa en una función logarítmica (inversa de una exponencial).

- Optimizador: Función que le dice al modelo cómo cambiar el valor de sus pesos. Junto a la función de pérdida, son los dos algoritmos fundamentales para determinar en la construcción del modelo; y con la *data* de entrenamiento, estos tres elementos hacen que la red aprenda. Los pesos se ajustan según los valores que vaya arrojando la función de pérdida, como es de notar en Ilustración 27. En Ilustración 28 se aprecia cómo este cambio no es brusco sino más bien tanteando.

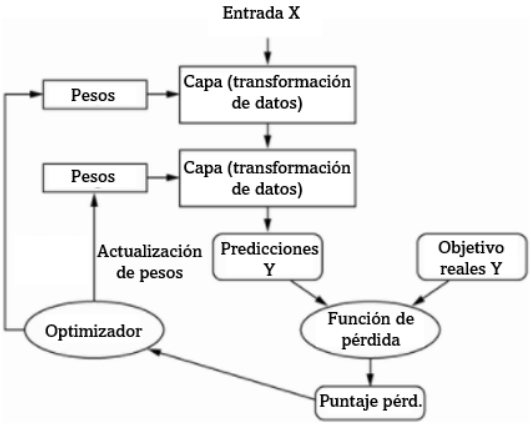


Ilustración 27 Puntaje de pérdida como valor para ajustar los pesos. Fuente: obtenido de [24].

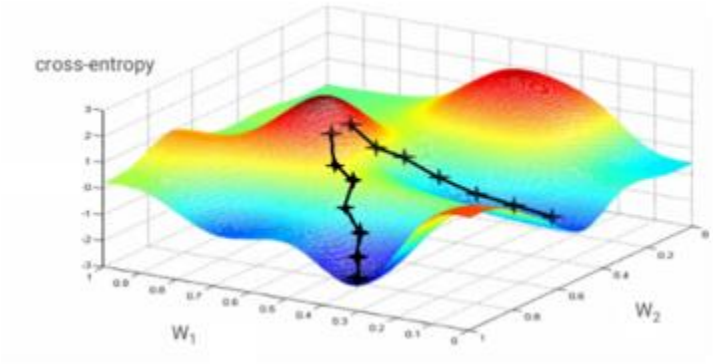


Ilustración 28 Se toman pequeños pasos en la dirección en la cual la pérdida decrece más [29].

Stochastic Gradient Descent: El “gradiente de descenso estocástico” es un conjunto de métodos iterativos de optimización, cuyos algoritmos pueden ser la función optimizador usada para el entrenamiento de un modelo, y en el caso de aprendizaje profundo, virtualmente todos los algoritmos utilizados para optimización pertenecen a esta familia

[33]. Estos algoritmos tienen propiedades que reducen el uso de recursos computacionales, lo que da paso a una mayor rapidez de iteración, al tiempo que se tiene una tasa de convergencia menor.

El gradiente es un vector que dice en qué dirección deben cambiar los pesos para que la pérdida se reduzca más rápidamente; este es usado para que descienda la curva de pérdida a un mínimo. Es un proceso estocástico puesto que es determinado por casualidad, ya que las muestras provenientes del conjunto de datos son aleatorias.

- Épocas: Son los ciclos de entrenamiento, dicho de otra manera, las iteraciones de entrenamiento del modelo. Cada época, el modelo busca aprender de la *data* de entrenamiento, la función de pérdida determina la distancia entre la salida actual (la predicha en la iteración) del algoritmo y la salida esperada, y el optimizador recibe el "puntaje" determinado por la función de pérdida, haciendo ajustes en los pesos (lo cual incluye al *bias*) de acuerdo con este valor, para mejorar el desempeño del modelo (reduciendo la pérdida y significando esto que la red neuronal está aprendiendo más).

Es de destacar o acotar varios puntos en lo relativo a las épocas. Ejemplo, los pesos no cambian basados en los puntajes de desempeño (por precisión) que arroja la función de pérdida para la *data* de validación; esto es exclusivo del conjunto de entrenamiento. Asimismo, es de notar que los primeros valores de los parámetros (pesos) son establecidos de manera aleatoria. Se ve una curva de aprendizaje en Ilustración 29.

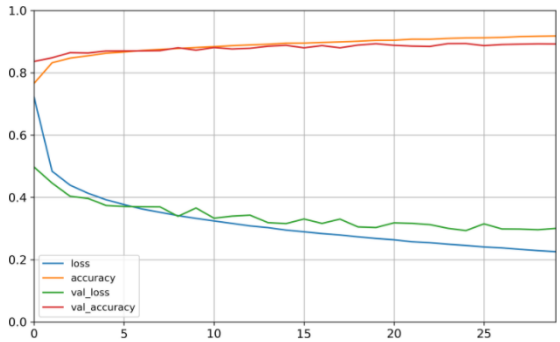


Ilustración 29 Curvas de aprendizaje. Fuente: obtenido de [25].

- Minibatch: Usualmente llamado solo “*batch*”; es una iteración de entrenamiento que, a diferencia de la época, en la cual es utilizada una ronda completa del conjunto de *training data*, usa una muestra de todos esos datos. En Ilustración 30 se toman en cuenta solo algunos de los datos, mientras que en Ilustración 31 se presenta una forma de ver el proceso.

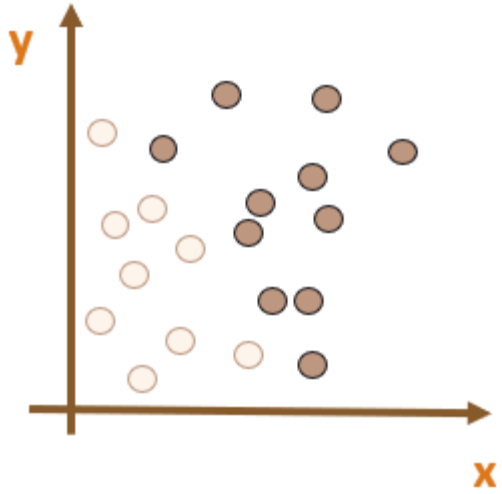


Ilustración 30 Algunos datos de muestra. Fuente: El autor.

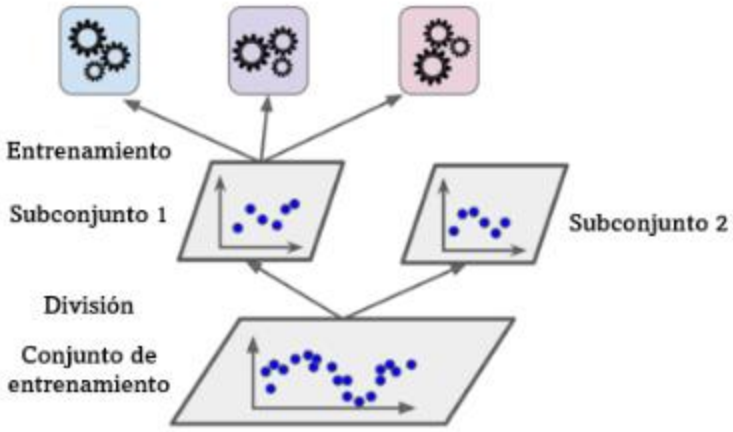


Ilustración 31 Representación de un modelo siendo entrenado por batch (el proceso se repite por iteración). Fuente: adaptado de [25].

Se pueden visualizar curvas de aprendizaje independientes para los distintos conjuntos de datos, como se muestra en Ilustración 32. Para ambos casos se utilizó un optimizador para ir mejorando la curva, y en el caso de uno de tipo descenso de gradiente, sería como se aprecia en Ilustración 33.

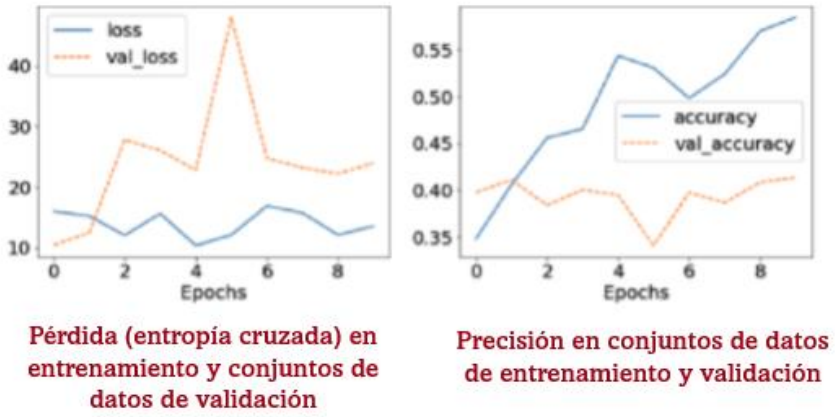


Ilustración 32 Curvas de pérdida y precisión en los conjuntos de entrenamiento (línea sólida) y validación (línea punteada). Fuente: adaptado de [29].

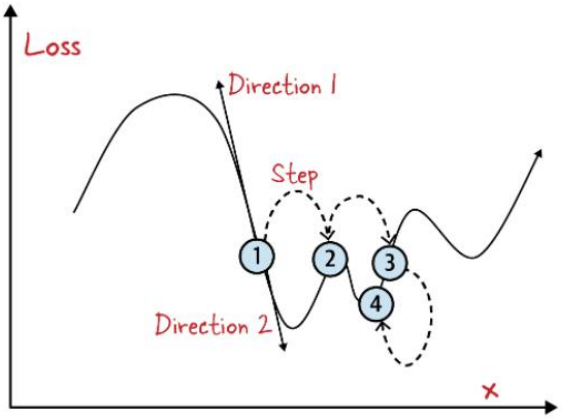


Ilustración 33 Cómo funciona un optimizador tipo descenso de gradiente. Fuente: obtenido de [29].

A sabiendas de que se quiere identificar patrones, se debe distinguir dos (2) tipos: señales y ruido. Las señales son los patrones positivos, los que de verdad se necesita; son los reales, si se quiere. Los ruidos, por su parte, son patrones que no conviene que la red neuronal encuentre, porque no son referencias reales de las relaciones entre

entradas y salidas esperadas, sino simplemente patrones ciertos para la *data* con la cual se entrena.

Vendría bien pensar en un ejemplo para entender mejor. Cuando se habla de patrones se podría estar hablando de los encontrados al estudiar canes, en particular, imágenes para identificar si en ellas hay o no algún can. Un modelo podría encontrar elementos como torsos, patas y colas, considerando distribuciones y proporciones, también pelaje u otras características (pues en ocasiones no pasamos "explícitamente" las características). Estas son positivas. No lo sería si en su lugar (de forma paralela, realmente) el modelo encontrase patrones como que el pelaje de esta especie es abundante y esponjoso. Esto sería ruido.

Para profundizar en cómo el modelo va encontrando uno u otro tipo de patrón, el porqué, y qué hacer al respecto, se debe atender a algunos conceptos adicionales.

- Curva de aprendizaje: Es una representación gráfica entre cuan precisa es una red neuronal prediciendo y cuánto entrenamiento tiene. La relación en este caso es comúnmente "mientras más se entrena, más se aprende". El historial de esta relación se lleva para tener la posibilidad de mejorar el modelo (a través de su entrenamiento) al interpretar lo que evidencia la curva de aprendizaje (y con ello al modelo).

Esta curva se grafica (automáticamente) con los puntos en un plano, que corresponden al par época-pérdida, es decir, en cada iteración se marca el puntaje calculado por la función de pérdida. A menudo (prácticamente siempre en el caso de desarrollos importantes) se tendrán dos curvas de aprendizaje para un modelo de *Deep Learning*: la de entrenamiento y la de validación. El tamaño de la brecha de estas curvas de aprendizaje refiere qué tanto ruido ha aprendido el modelo [33].

- Overfitting: El sobreajuste se da cuando se ha entrenado tanto al modelo con el conjunto de entrenamiento, que este ha comenzado a identificar patrones ruidos (solo aplicables a esta *data*); también puede darse cuando este conjunto de datos no es muy variado (por cantidad o calidad), aunque es menos frecuente cuando se utilizan un buen número (esto puede variar de decenas a millones según el

caso de uso y el modelo). En sí, el *overfitting* se identifica porque se tiene una pérdida no tan baja como debería o podría ser dado a que el modelo aprendió mucho ruido.

- Underfitting: Es cuando existe “ajuste insuficiente”, es decir, hay menos pérdida de la que podría haber porque el modelo no ha aprendido suficientes patrones positivos. Generalmente este problema no se resuelve simplemente aumentando el número de ciclos de entrenamiento (épocas) pero instintivamente se piensa en esto como posible solución a este problema, y se debería descartar.
- Capacidad: Se refiere al tamaño y complejidad de los patrones que un modelo puede (es capaz de) aprender. La cantidad de neuronas y cómo estas están conectadas determina en buena medida esta propiedad, encontrada en cualquier modelo de *ML*, para las redes neuronales. En lugar de aumentar el número de épocas, la solución a un hipotético *underfitting* presentado en un modelo, podría ser el aumento de su capacidad.
- Callback: Es la forma más empleada de incluir detención anticipada “*early stopping*” en un modelo, esto es, interrumpir el proceso de entrenamiento, mayormente cuando la pérdida de validación deja de decrecer, o simplemente cuando se ha conseguido un nivel de precisión considerado suficientemente bueno. Es una de las mejores formas de evitar el *overfitting*, puesto que se detendrá el proceso antes (ya que se retorna al mejor puntaje de pérdida para la *data* de validación) de que el modelo aprenda ruido (o más del aceptable, mejor dicho). También previene el *underfitting* en vista de que puede seguirse entrenando hasta que se hayan aprendido “suficientes” patrones positivos (en lugar de establecer ciclos de entrenamiento insuficientes).

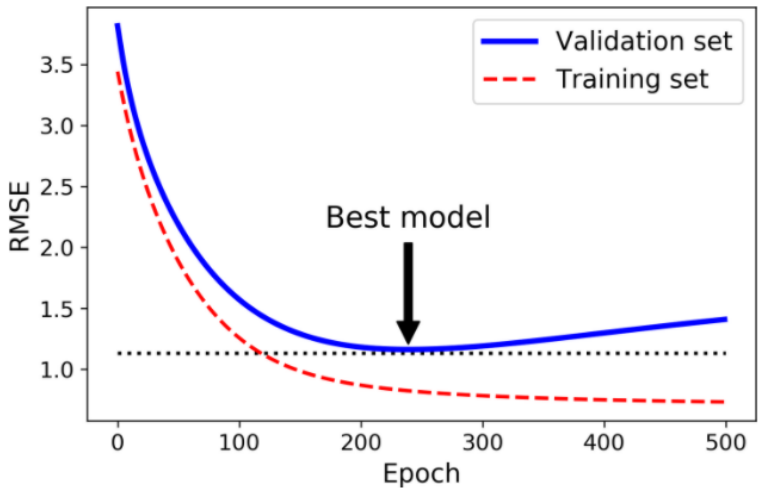


Ilustración 34 Regularización de detención temprana. Fuente: obtenido de [25].

Con *Early Stopping*, como se observa en la Ilustración 34, el entrenamiento se detiene cuando detecta que el modelo ya no está aprendiendo, pero se devuelven los valores (de los pesos) al mejor momento de precisión (lo que sería el mejor modelo) [23].

Tras concretar qué estructura posee la red neuronal, y luego de entrenar el modelo, se espera saber cuán bueno es en términos de precisión.

Validación del modelo: Es una tarea para confirmar, en principio, si un modelo es aceptable respecto al proceso de inferencia tras el entrenamiento; pero con miras a posibilitar el determinar cuán preciso y, por ende, cuán válido y valioso una vez puesto en funcionamiento.

En este proceso se resume la calidad del modelo de forma comprensible en una sola métrica, y el error absoluto medio (*MAE*) suele ser usado para tomar el valor absoluto de cada error convirtiendo cada error a un valor positivo y se calcula la media de esos errores absolutos, determinando con ello cuán alejadas de la realidad están las inferencias del modelo entrenado.

A sabiendas de que una red neuronal puede pertenecer a varias categorías de acuerdo con la clasificación; se puede subdividir una red neuronal dada en dos (2): profundas o superficiales. Esto, como es de esperarse, según su “profundidad”. A lo que se refiere

esta clasificación es a “con qué tantas capas" cuenta la red. Ver Ilustración 35, donde hay redes de ambos tipos.

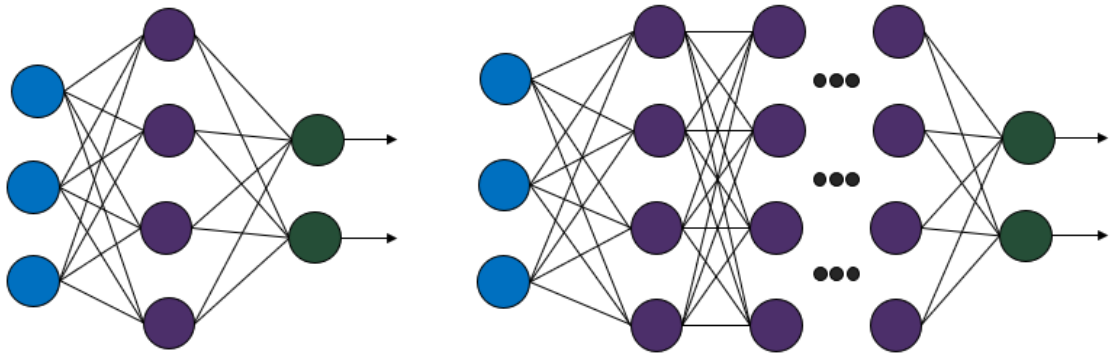


Ilustración 35 Red neuronal superficial genérica (izquierda) versus una red neuronal profunda genérica (derecha). Fuente: El autor.

La red neuronal más pequeña posible consiste en una única neurona. Asimismo, una red neuronal podría tener una sola capa oculta (capa intermedia entre la de entrada y la de salida), más la capa de entrada y la capa de salida (como en Ilustración 35). Estos casos se entienden como “redes neuronales superficiales”, o “no profundas”. Contrastan con estas aquellas redes con gran número de capas, a las cuales se les denomina “redes neuronales profundas” (también en Ilustración 35). Otra red superficial se ve en Ilustración 36. Por su parte, la Ilustración 37 presenta otra red profunda.

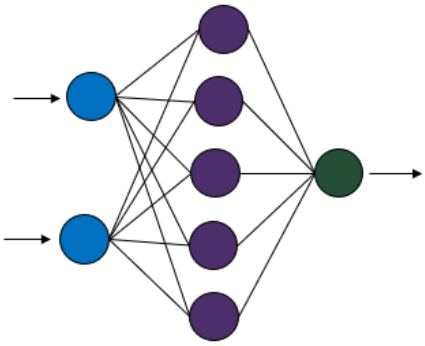


Ilustración 36 Red neuronal superficial con una capa de entrada con dos (2) neuronas, una intermedia (oculta) con cinco (5) y una capa de salida con solo una (1) neurona. Fuente: El autor.

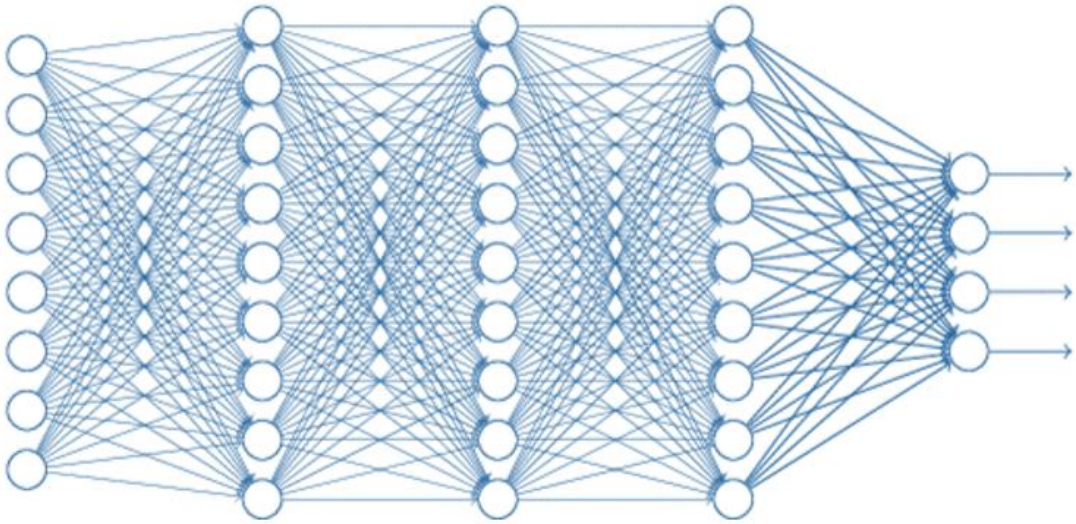


Ilustración 37 Red neuronal profunda evidenciando lo complejo de algunas redes. Fuente: obtenido de [34].

Pese a que no ha sido claramente definido lo que ya se considera una red neuronal profunda; en la práctica, se suele catalogar como tal a aquellas con al menos dos (2) capas ocultas, en conjunto con las de entrada y salida. Los pesos se van alterando por cada capa que pasan (ver Ilustración 38), siendo los últimos los resultantes de todos los cálculos previos y, a su vez, aquellos valores que se buscaban desde un principio.

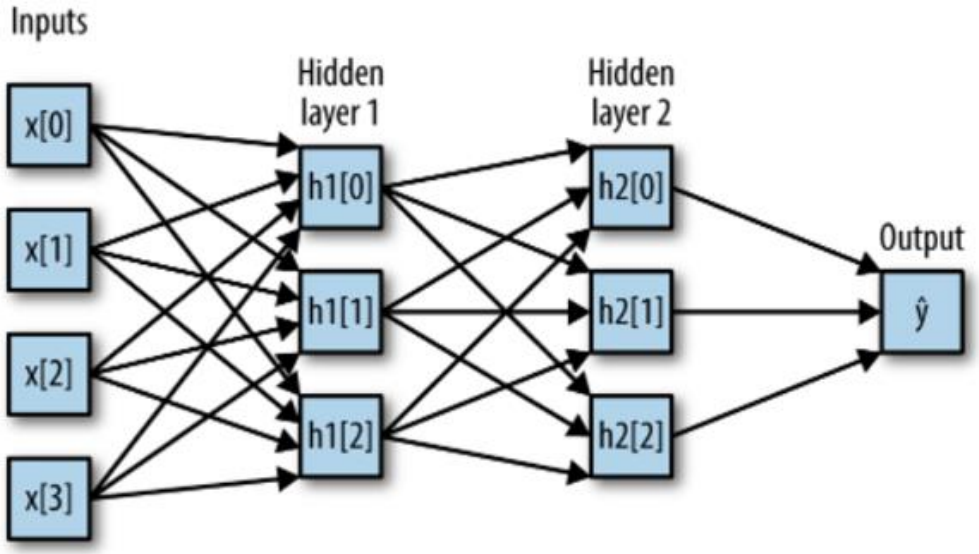


Ilustración 38 Cambios en los pesos capa a capa. Fuente: Obtenido en [28].

Importa, a la hora de trabajar con redes neuronales, cómo se comunican las neuronas, cuántas son y en qué cantidad de capas estén distribuidas. Esta última característica suele ir ligada con la complejidad del modelo desarrollado (o a desarrollar, si solo se ha diseñado la red), así como también a la propia complejidad del problema que busca ser resuelto.

En vista de que las redes neuronales profundas ofrecen más posibilidades y mayor capacidad, la estructura detrás del subcampo Aprendizaje Profundo son las redes de este tipo, como ya advertía su nombre. Dicho con otras palabras, un modelo de Aprendizaje Profundo es aquel cuya estructura es una red neuronal profunda, o al menos una.

Redes Neuronales Convolucionales

Con el paso de los años y el crecimiento de Aprendizaje Profundo como disciplina, se han concebido distintos tipos de redes neuronales. En esencia, los cambios tienen que ver con estructura y con cómo se comunican las capas, y las neuronas dentro de estas capas. Se han planteado estrategias para lidiar con problemas que previamente no se podían solventar. Si un tipo de red neuronal profunda destaca entre todos, y hasta hace no mucho tiempo de forma contundente, son las Redes Neuronales Convolucionales ("*Convolutional Neural Networ*ks", abreviadas "*ConvNet*" y de siglas "*CNN*").

Las Redes Neuronales Convolucionales dieron pie a avances significativos dentro del *Deep Learning*, y se les puede atribuir una parte importante de la popularidad alrededor de *DL* y del propio *ML*. Lo más preponderante es lo que ha significado para el campo de Visión Artificial (cuyas siglas en inglés son "*CV*", de "*Computer Vision*"); siendo conceptos casi imposibles de desligar, o al menos de no nombrar cuando se habla del otro (pese a haber existido desde mucho antes *CV* y tener muchas más aplicaciones las *CNN*).

Las *CNN* son las estructuras predominantes dentro de Visión Artificial, para entender su alcance, la razón de ser de este hecho, y sus implicaciones, atañe entender algunas de las nociones detrás de esta idea de tanto peso para las Ciencias Computacionales en la actualidad.

- Capa convolucional: Es una capa cuyas neuronas no están conectadas a cada pixel de la imagen de entrada (cuando es la primera capa) o de la parte de la imagen de la capa anterior (a partir de la segunda capa convolucional), sino únicamente a los de sus campos receptivos (la porción o “filtro” de la imagen de la cual recibe información) [25].

Estas capas se manejan bajo la operación de convolución, llamadas así por la operación matemática homónima, aunque realmente lo que utilizan (estas capas) es una correlación cruzada. Ilustración 39 presenta una perspectiva para entender el comportamiento de estas capas.

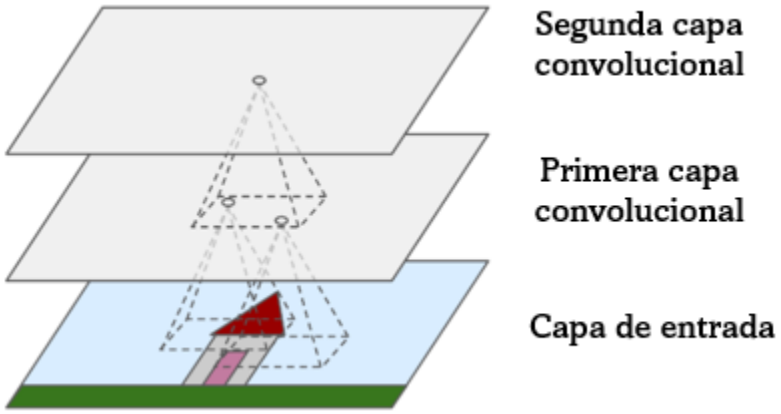


Ilustración 39 ConvNets con campos receptivos locales rectangulares [25].

- Extracción de características: En *ML* es todo aquel proceso de transformación de *data* cruda (no procesada) a características numéricas con la cualidad de poder ser procesada sin alterar la *data* original y la bondad de conllevar a mejores resultados que si se aplicase *ML* directamente a esa *data*. Retomando la definición de una capa como una especie de transformador de datos, se puede entender el papel de esta idea para una *DNN*. En *ConvNets* la estructura de capas convolucionales es la principal que permite a la red neuronal extraer características de las imágenes, llamadas “características visuales”; en realidad se suele hablar de una característica particular como consecuencia de la

operación de filtrado, una de las tres operaciones básicas de la extracción de características. Una segunda operación sería la detección de esa característica dentro de la imagen filtrada; esto se logra con la función de activación (típicamente *ReLU*). La tercera operación es la de condensación, en la cual se realza la característica estudiada. En Ilustración 40 se muestra una representación de este proceso. Este proceso tiene por consecuencia obtener un mapa de características, que varía respecto al procesamiento de las imágenes, como se evidencia en Ilustración 41.

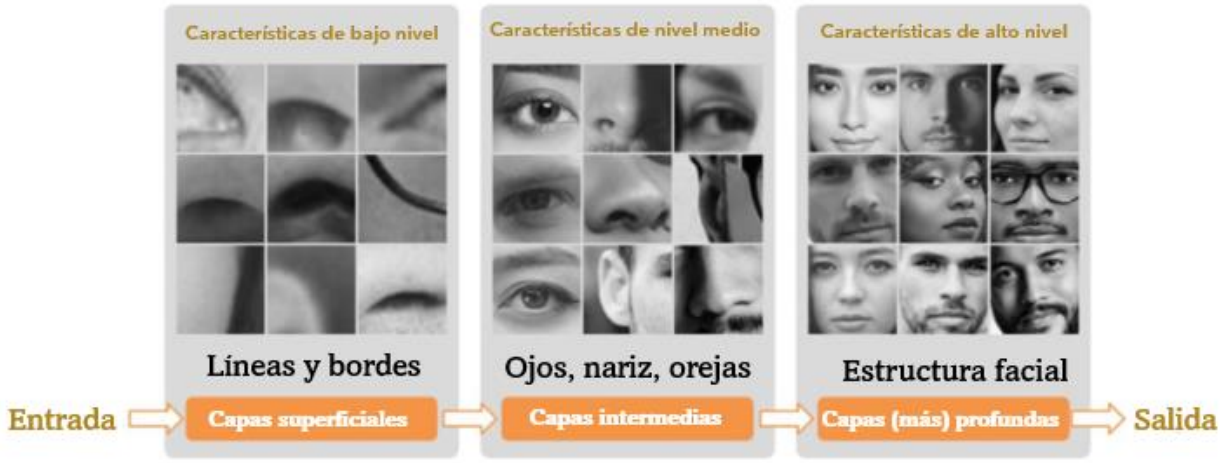


Ilustración 40 Extracción de características en una ConvNet. Fuente: adaptado de [27].

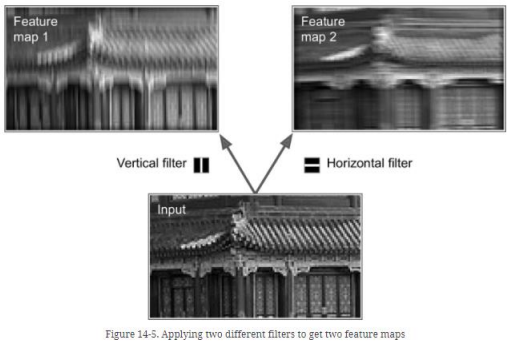


Ilustración 41 Aplicando dos filtros diferentes para obtener dos mapas de características. Fuente: obtenido de [25].

- Capa de pooling: La forma en la que este transformador de *data* aporta a la extracción de características es acentuando las características resaltadas por la capa convolucional que suele estar inmediatamente antes de esta. Le corresponde, entonces, la tarea de condensar la imagen; lo cual suele hacerse con la operación *maximum pooling*, que calcula el valor máximo (de allí su nombre) o más grande en cada parte, del mapa de características, que se haya definido (intensificando la característica). Por cómo se comporta este transformador, puede darse una invarianza pese a partir de condiciones (posicionales) distintas, como se ve en Ilustración 42.

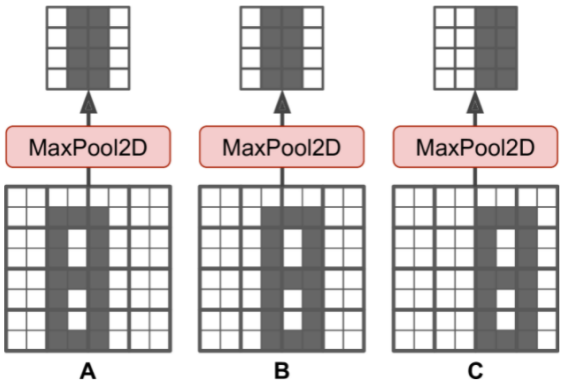


Ilustración 42 Invariancia a pequeñas traslaciones [25].

Típicamente se procesa una entrada de tipo imagen por canales de colores (rojo, verde y azul, de RGB); se observa en la Ilustración 43 su interacción con las *ConvNets*.

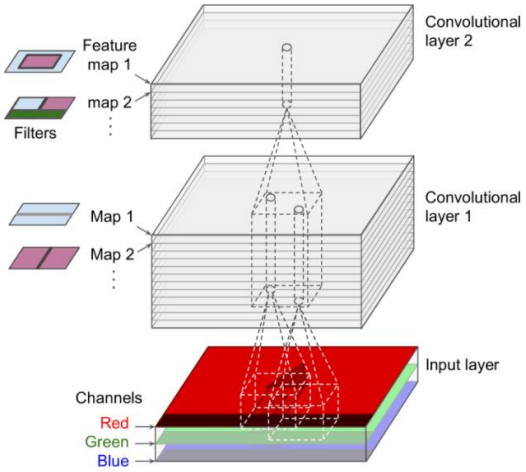


Ilustración 43 Capas convolucionales con múltiples mapas de características, e imágenes con canales de tres colores. Fuente: obtenido de [25].

- Clasificador Convolucional: Es aquel en el cual existe en su diseño al menos una capa convolucional. Una red neuronal convolucional (por ende, este tipo de clasificadores) consiste en una base para extraer los *features* de una imagen y una "cabeza" para determinar la clase. La base es formada por las capas convolucionales (al menos) y la cabeza consta principalmente de las capas densas.

  Un clasificador convolucional bien podría tener como parte de su base la red neuronal convolucional de la cual se parte (de esta no se usaría la cabeza), más las capas con las cuales se ajusta el conocimiento de la red para adecuarlo a la aplicación objetivo. La cabeza del nuevo modelo se dispondrá a definir las posibles clases y sus etiquetas.

  La arquitectura es usualmente como la observada en Ilustración 44, destacando siempre los pares de capas *convolución-pooling* como también se ve en Ilustración 45.

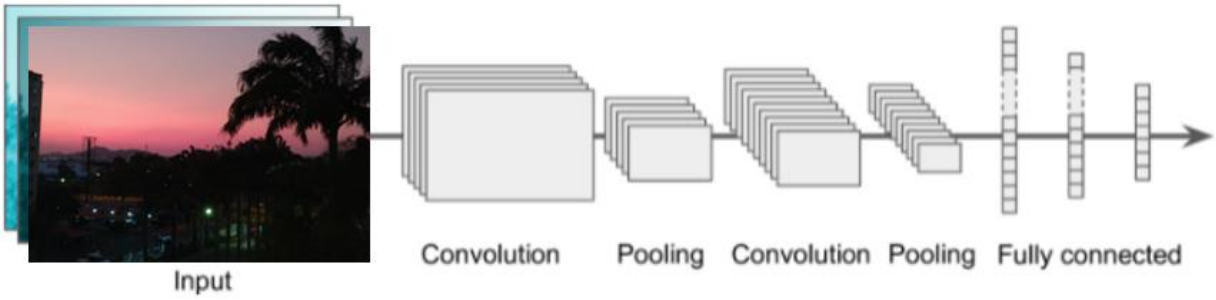


Ilustración 44 Arquitectura CNN típica. Fuente: adaptado de [25].

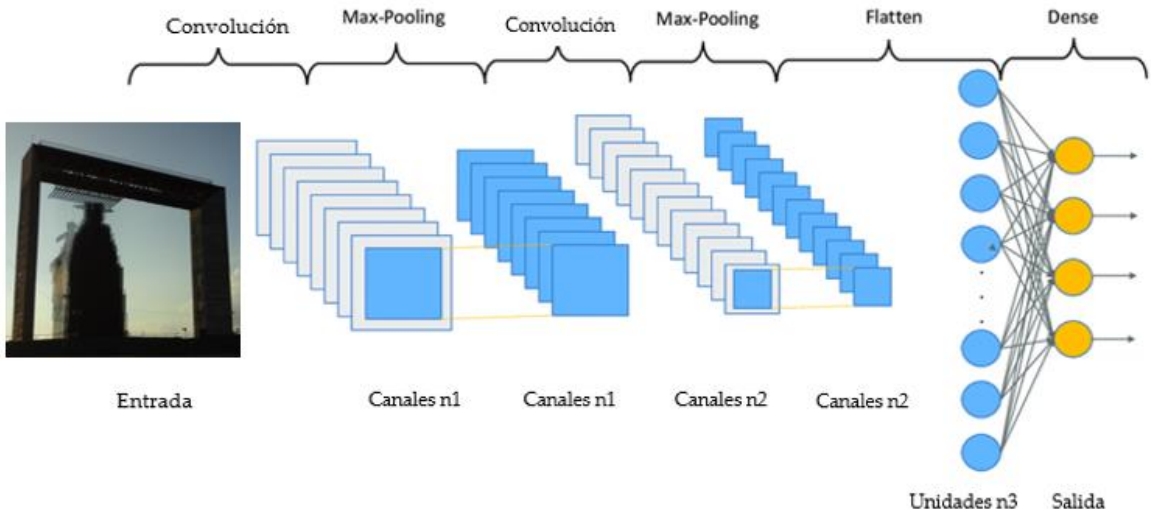


Ilustración 45 Pares convolución-pooling procesando una imagen. Fuente: adaptado de [35].

Para entender cómo es una estructura de estas respecto al procesamiento y cálculos que debe hacer, puede ser útil detallar la Ilustración 46. Las convoluciones en sí lucen como en Ilustración 47 aplicándoseles abstracción.

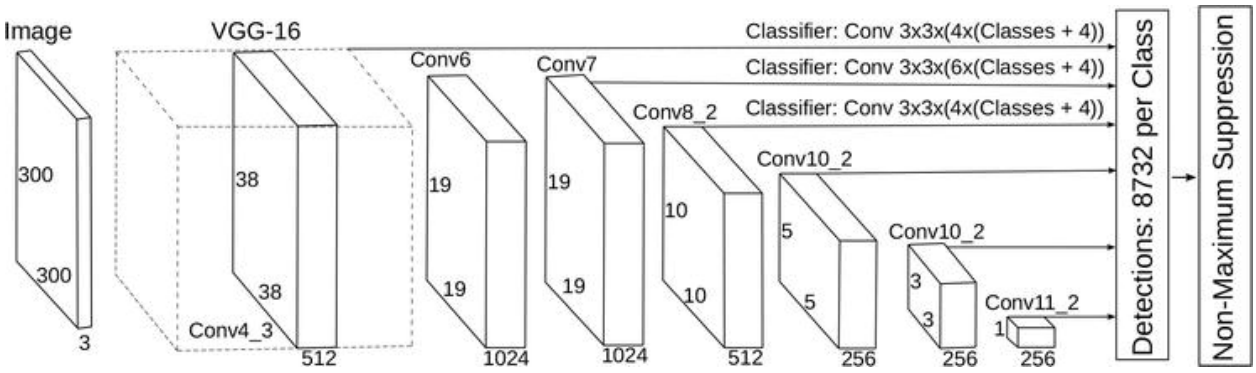


Ilustración 46 Ejemplo de funcionamiento computacional de la estructura. Fuente: obtenido de [36].

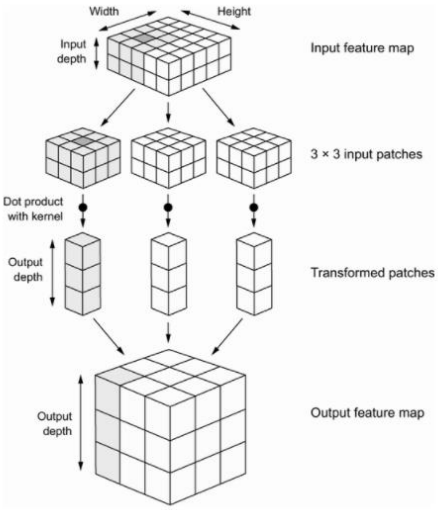


Ilustración 47 Cómo funcionan las convoluciones. Fuente: presentado en [24].

Parámetros Stride y Padding: Estos parámetros afectan a las capas de convolución y *pooling* a través de sus respectivas *sliding windows* (ventanas deslizantes) que son a través de las cuales se producen sus efectos, siendo cada una de estas un cálculo característico de las capas. El número de desplazamientos de píxeles sobre la matriz de entrada se define con el parámetro *stride*, significando que los filtros (o alteraciones) se aplican moviéndose X píxeles a la vez. Mientras tanto, el parámetro *Padding* es con el cual se define la cantidad de píxeles agregados a una imagen cuando está siendo procesada por el *kernel* (núcleo) de una *CNN*. En conjunto son gran parte de lo que propone una red neuronal convolucional desde lo conceptual. Una representación de estos parámetros se aprecia en Ilustración 48.

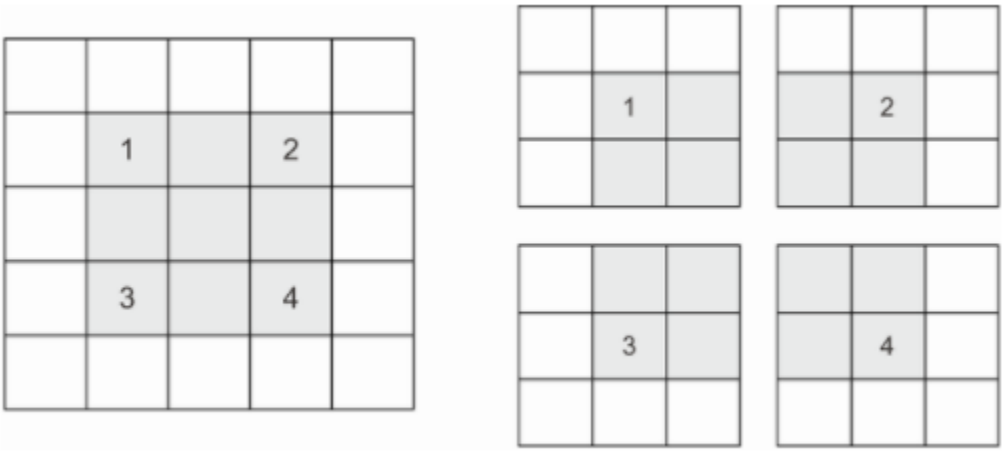


Ilustración 48 Patches de convolución tamaño 3 x 3 (izquierda) con strides tamaño 2 x 2 (derecha). Fuente: obtenido de [24].

Se decía que una computadora puede interpretar una entrada tipo imagen como un arreglo (matriz) de números como se señala en Ilustración 49. A su vez, esta matriz (que es mi entrada tras procesarse) puede ser trabajada, y se da la aplicación de filtros en estos casos (sean los convolucionales u otros); ambos elementos se muestran en Ilustración 50. En Ilustración 51 se enseña el primer resultado de aplicar este filtro que conserva y suma los valores de las esquinas superior derecha e inferior izquierda.

Ilustración 49 La entrada de tipo imagen (izquierda) se convierte a algo que pueda ser entendido por la computadora: una matriz de números (derecha). Fuente: obtenido de [31].

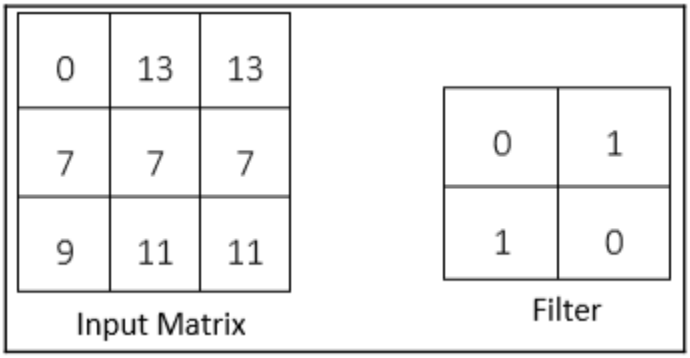


Ilustración 50 Matriz de entrada (izquierda), y filtro. Fuente: obtenido de [31].

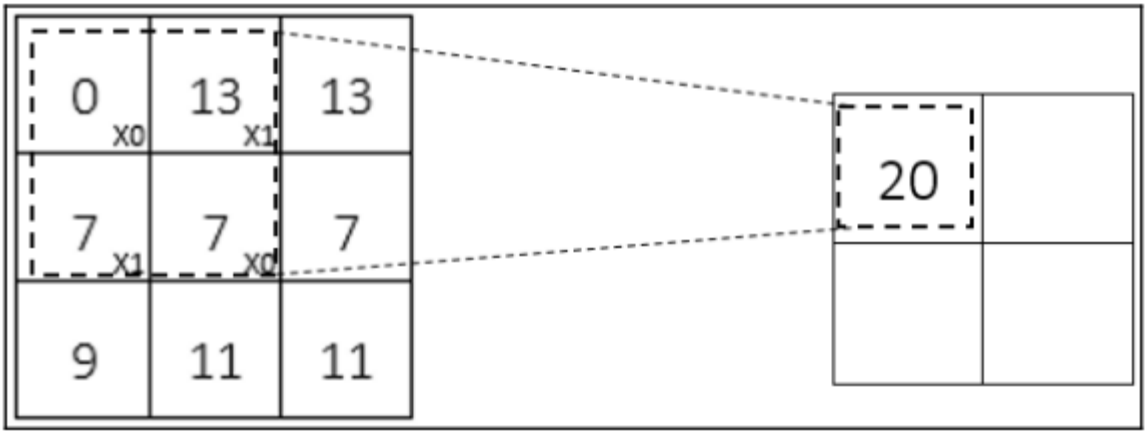


Ilustración 51 Matriz de entrada de la cual se considera un grupo de números (izquierda) y primer número generado de aplicar este filtro (derecha). Fuente: obtenido de [31].

Bloques convolucionales: Son elementos dentro de una red neuronal convolucional que ejecutan extracciones de las cuales se obtienen características producidas que colectivamente adquieren una forma que les permita dar solución al problema al cual se enfrente un modelo basado en la estructura que los contienen. Una versión “profunda” de una *CNN* es aquella compuesta con varios bloques convolucionales, que son triadas de capas de convolución, activación y *pooling*; estas propuestas son capaces de llevar a cabo ingeniería de característica compleja en visión artificial. En Ilustración 52 se ve la extracción de características y la clasificación tras aplicar estos bloques convolucionales.

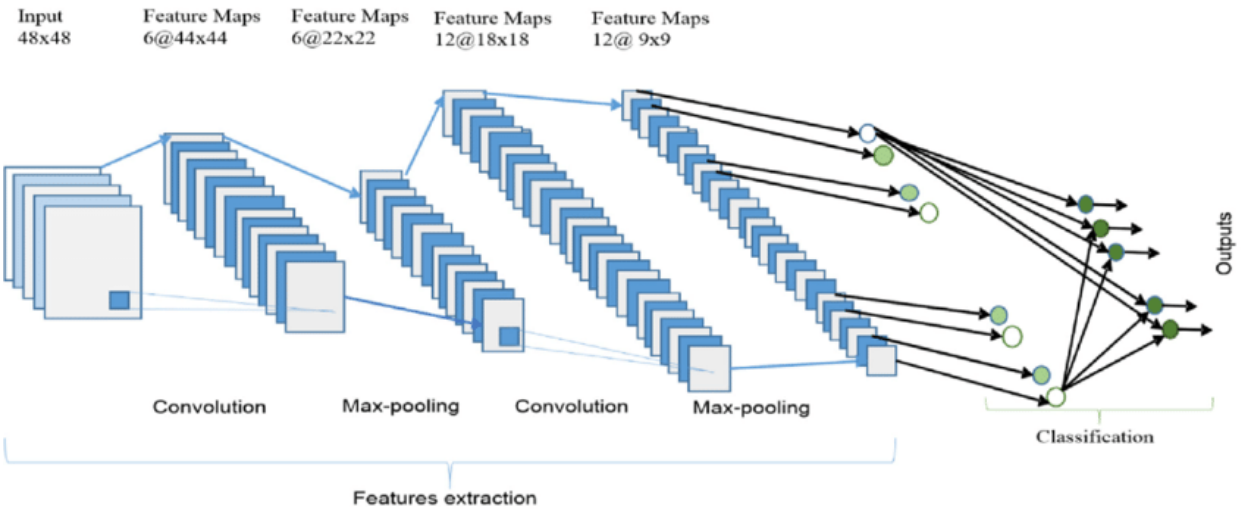


Ilustración 52 Estructura con bloques convolucionales. Fuente: adaptado de [37].

Data Augmentation: El “aumento de datos” es una técnica común en la práctica para cuando se desea tener un mayor número de datos, sobre todo de entrenamiento, a partir de aquellos con los que ya se cuenta. La lógica detrás es simple; se parte del entendido de que lo que se necesita, a efectos del desarrollo, es que el modelo identifique los patrones que sí impactan positivamente la inferencia, y de hacerse con más *data*, esta debe ser de valor. A partir de esto se tiene que hay una serie de alteraciones que se le pueden hacer a las imágenes, estas podrán tener la forma de inclinación de la imagen, cambios en su brillo y contraste, uso de solo una parte de la imagen original, entre otras.

De cualquier forma, las alteraciones a hacerle a las imágenes deberán ser no invasivas en términos de los elementos o patrones relevantes que debería presentar la imagen con la que se enseña. Un ejemplo de mala práctica con *data augmentation* podría ser el necesitar capturar una mano haciendo una seña y hacer un recorte en el cual parte de la mano deje de aparecer, puesto que sería una entrada errónea. Otro ejemplo pudiese ser un clasificador de objetos para el que se hayan considerado exclusivamente imágenes de carros completos (una cara, de frente, etc.) y que al momento de recortar la imagen solo se aprecie una pequeña parte del parachoques y una de las luces delanteras. Según cómo fue entrenado el modelo, esta entrada no aporta a la inferencia, por el contrario, entorpece el proceso. Si se estuviese hablando de un clasificador que quisiera identificar

si lo que aparece en una imagen es un carro (o le pertenece a uno) pese a solo apreciarse una pequeña de este (carrocería por debajo, techo, retrovisor u otros), entonces, la entrada anterior (imagen que pasó por proceso de *data augmentation* y resultó en una imagen de solo parte de un parachoques), no estaría mal. Esto último ejemplifica cómo varían las formas de implementar técnicas de acuerdo con las necesidades y aspiraciones particulares. Una representación gráfica se puede apreciar en la Ilustración 53.

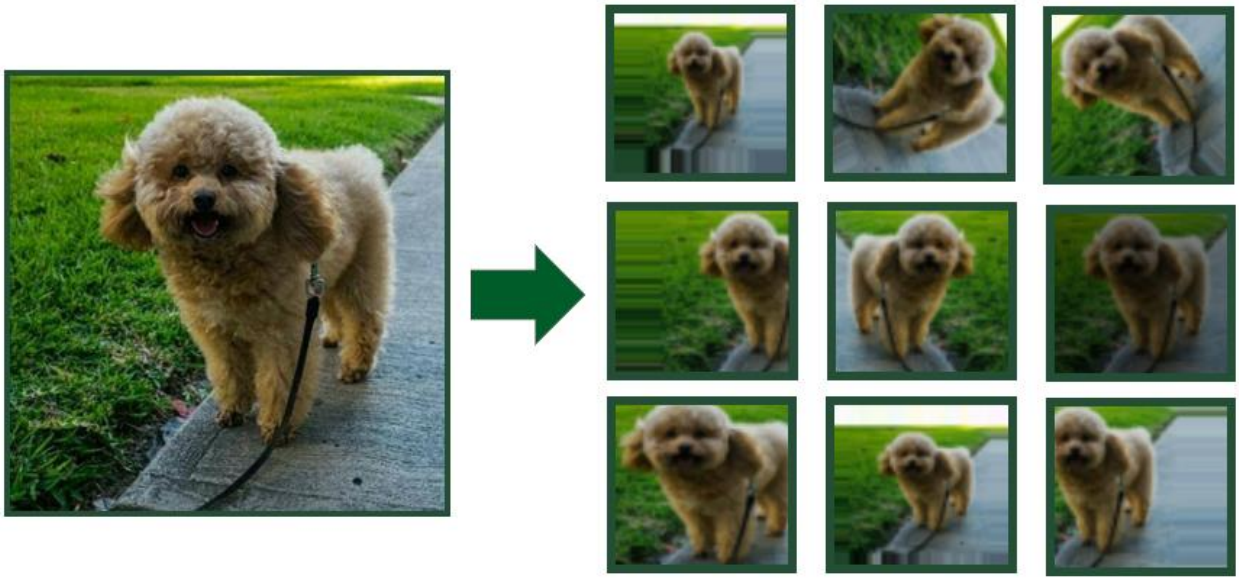


Ilustración 53 Generando nuevas instancias de entrenamiento a partir de las existentes. Fuente: el autor.

Redes Neuronales Recurrentes

Las *RNN* son redes neuronales con una arquitectura especial que les concede la capacidad de manipular y procesar secuencias de datos, extrayendo información de estas. Una forma de verlo, con un alto nivel de abstracción, es que son *DNN* con "memoria". En Ilustración 54 se ejemplifica ese *retorno a un estado* que ocurre en estas estructuras; la Ilustración 55 nos permite ver con claridad cómo este es un retorno distintivo. Esta memoria resulta ser un componente sumamente valioso, que ha permitido conseguir avances significativos en áreas como reconocimiento de voz ("*Speech Recognition*"), traducción automática ("*Machine Translation*"), análisis de secuencia de

ADN (“*DNA Sequence Analysis*”), análisis lingüístico (“*Linguistic Analysis*”), análisis de sentimiento (“*Sentiment Analysis*”, también conocido como minería de opinión), predicción de series temporales (o cronológicas, “*Time Series Prediction*”), detección de anomalías (“*Anomaly Detection*”) y generación de texto (“*Text Generation*”), de entre una nutrida lista.

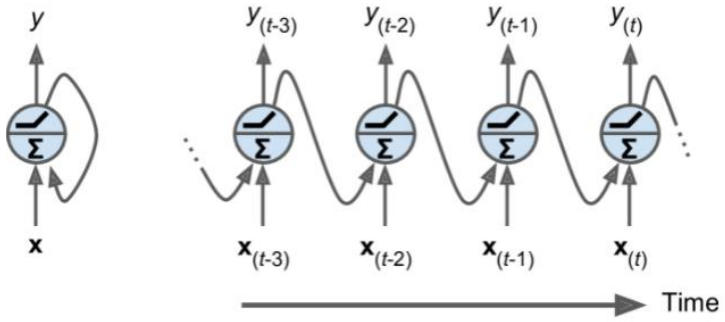


Ilustración 54 Una neurona recurrente (izquierda) desenrollada a través del tiempo (derecha). Fuente: obtenida de [25].

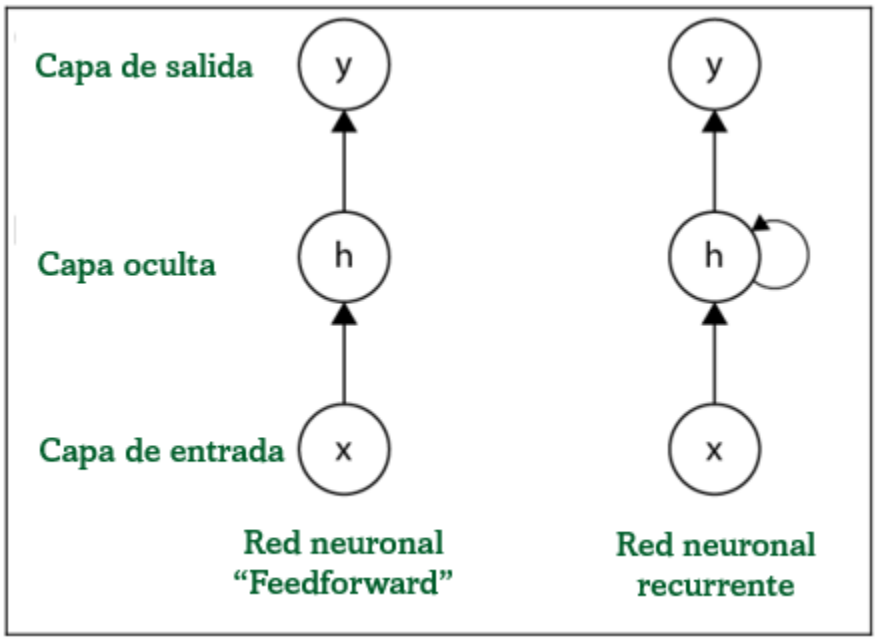


Ilustración 55 Redes feedforward (izq.) y recurrente (dcha.). Fuente: adaptada de [31].

Para afinar nuestra intuición respecto a cómo trabajan estas redes neuronales, cabe ver su cualidad principal como la de poder “predecir el futuro”; desde luego, respondiendo a un nivel de precisión y logrando esto hasta cierto punto. Algunos de los ejemplos anteriores, ayudan a visualizar la puesta en práctica de esta noción al desarrollar un modelo; de hecho, ya se hablaba de “predicción”, y también de “generación” que era a partir de lo anteriormente recibido (como señal o dato). Un caso de uso adicional podría ser un modelo que prediga los precios de las acciones en la bolsa de valores de los EE. UU., con el cual un inversor se guiase para decidir si comprar o vender determinadas acciones. Estos modelos trabajan pronosticando los siguientes pasos, como se aprecia en Ilustración 56.

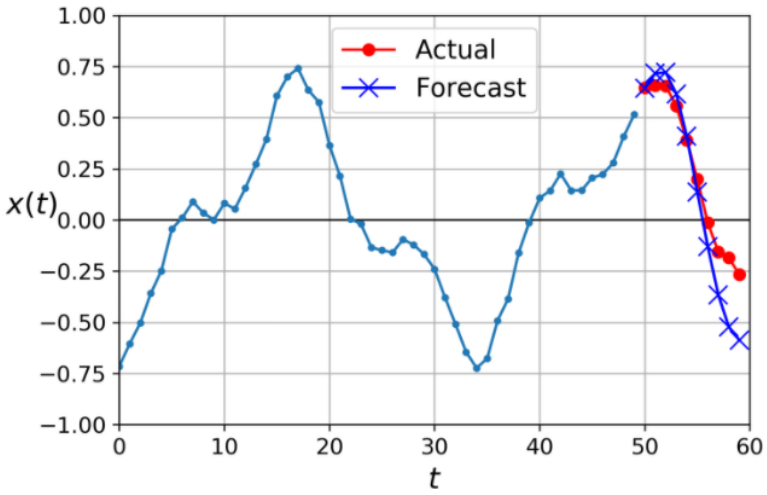


Ilustración 56 Pronosticando 10 pasos adelante, uno a la vez. Fuente obtenida de [25].

Las “*Recurrent Neural Networks*” pueden trabajar con datos de tipos tan diversos como los meteorológicos o las señales de audio, pero el foco siempre será el mismo “secuencias de datos”. Estas secuencias bien podrían ser de datos de texto (“*text data*”), como secuencias de palabras o párrafos, en el dominio de procesamiento del lenguaje natural (abreviado PLN, o *NLP* del inglés “*Natural Language Processing*”); o tal vez se pudiese estar hablando en su lugar de canciones (“*audio data*”) donde se consideran las

secuencias de señales; o del historial de temperatura y humedad de acuerdo con el tiempo meteorológico de una ciudad.

Este tipo de RRNN se entiende, entonces, con datos secuenciales; y a los modelos desarrollados con arquitecturas de esta categoría, también se les señala como “secuenciales”. No solo se abrió la posibilidad de un catálogo amplio de problemas que ahora podían ser resueltos con las *RNN*, sino que ha sabido superarse esta clase de estructura sumando, cada vez más, tareas que ensanchan el listado de opciones para las cuales valerse (o intentar valerse) de este enfoque.

Si se quiere analizar un video, en lugar de una imagen, es útil abarcar el problema teniendo presente que un video es, en esencia, una secuencia de imágenes; por lo cual es de suponer que se pueden usar redes recurrentes en este caso. Es conveniente revisar aspectos de las redes convolucionales, al tiempo que se conocen algunos de las redes recurrentes y se repasa el funcionamiento conjunto para esta tarea.

Para llevar a cabo el análisis de un video se puede construir una red neuronal convolucional para el análisis de imágenes y apoyarse en un modelo temporalmente sensitivo, como lo puede ser un modelo secuencial desarrollado a partir de una red neuronal recurrente. Mientras que, para las imágenes estáticas, que en este caso serían los cuadros del video (“*frames*”), se usan *ConvNets* para identificar características visuales; las redes recurrentes se encargarán de la agrupación de estos elementos en uno único, con todo lo que esto conlleva.

Es natural pensar en la posibilidad de simplemente replicar los efectos de una *CNN* para cada uno de los *frames* de un video. El inconveniente es el cómo trabaja esta estructura y, en principio, presenta dificultades para lidiar con esos grupos de cuadros como un todo. Las *CNN* son excelentes con observaciones independientes, y podrían ser muy buenas también con grupos de observaciones si para estos no se tuviese que tomar en cuenta el *contexto*. En otros términos, un video es una secuencia de datos (imágenes) intrínsecamente ordenados, por lo que importa cómo se comportan los elementos que allí aparecen con respecto al tiempo; se pueden nombrar ejemplos como los eventos, la aparición de los objetos y la posición, pero todo se limita al orden cronológico (exclusivamente en el sentido del pasado al presente, pues también esto es inalterable).

La forma de la entrada (imagen) para una red convolucional (al igual que para otras clases) es de dos dimensiones: cantidad de observaciones y cantidad de características. Para una red recurrente, la forma tiene un tamaño que depende de tres elementos (es decir, es tridimensional, en vez de bidimensional), estos son: el número de observaciones, el número de características y el tamaño (la “longitud”) de cada secuencia, también representada por un número. Otra manera de ver la forma de estas entradas es como un vector unidimensional para la red convolucional (aunque también existen versiones de *ConvNets* que reciben entradas tridimensionales) y el mismo vector de una dimensión, al cual se le suma la cantidad de elementos en la secuencia (que pueden ser los *frames* de un video).

Hasta ahora, se puede entender que el hecho de que exista contexto entre los *frames*, y que haya cambios a través del tiempo imposibilita que sea usada una estructura que conste con solo bloques convolucionales (y los tradicionales para procesado de entrada a salida de RRNN), dado que estas podrán hacerse cargo de datos visuales en una imagen, mas no pueden hacerlo con características temporales como el contexto entre cuadros de un mismo video.

Las redes convolucionales tratan grupos de pixeles independientes, y existen estructuras con las que se puede retener información sobre lo procesado y usarlo para la toma de decisión, o en etapas intermedias del proceso. Las redes recurrentes pueden encargarse de muchos tipos de datos de entrada o salida, y aunque no se consideraba un buen enfoque para visión artificial como en clasificación de imágenes, de videos, siendo más exactos; puede apoyarse en estructuras destinadas para ello, resultando de esa cohesión potenciales nuevas posibilidades.

Una red neuronal recurrente que recibe videos en principio se entrena con descripciones de los *frames*, y asignando una etiqueta al grupo de *frames* (la secuencia) que representan al video; es decir, se etiqueta un video en lugar de una imagen, pero habiendo impreso contexto al describir las imágenes, hasta alcanzar la precisión deseada. Para emular la visión natural (humana, si se quiere) se debe usar una cantidad de datos excesivamente grande, descartando para muchos casos de uso y por las

limitantes de recursos, este enfoque conceptualmente sencillo (relativamente, cuanto menos).

Son millones de datos los que tendrían que ser usados en el entrenamiento; además, deberían ser variados: diversos ángulos, posiciones relativas a la cámara diferentes, y más. De ser videos de animales, por ejemplo, se deben considerar las diferentes especies de una familia o las razas dentro de una especie, incluso qué actividad estén haciendo: si están volando o nadando, durmiendo o comiendo, etc.

Por lo demás, se usa una función de pérdida para comparar con la etiqueta correcta, al igual que en RRNN previamente conceptualizadas. Del mismo modo, se ajustan los pesos según el error y por acción del optimizador y se vuelven a procesar las secuencias.

Funcionamiento de las RNN

Más allá de su acción conjunta con las *CNN*, esta estructura es capaz de resolver problemas varios, y las nociones tras de sí pueden ayudar a comprender su alcance. Se revisa así la forma de la *data*, algunas clases propias de las *RNN* y otros elementos.

- Hidden states: Los estados ocultos son entradas a cualquier cosa que se haga en determinado paso o intervalo, se pueden calcular únicamente al revisar datos de pasos previos (*data* que contiene de las entradas anteriores). Se usa para predicción.
- RNNLayer: Es una clase que introduce una secuencia de datos, pasando uno a uno los elementos (ordenadamente).
- RNNNode: Es una clase en el ecosistema *RNN* que se encarga de recibir los datos y actualizar los estados de las capas ocultas por cada intervalo de tiempo. Su método (de clase) principal, denominado "*fordward*" tiene dos arreglos como entradas: un vector de la forma [*tamaño_observación, número_de_características*] para las entradas de datos a la red, y otro cuya forma es [*tamaño_observación, núm_elementos_estado_oculto*] para las representaciones de las observaciones en ese intervalo. Además, cuenta con dos arreglos (también vectores) como salidas: uno de la forma [*tamaño_observación, núm_salidas*] para salidas de la red en ese intervalo, y otro para las

representaciones actualizadas para ese intervalo con la forma [*tamaño_observación, núm_elementos_estado_oculto*].

Cada observación con las que trabaja una *RNN* es bidimensional (*tamaño_de_secuencia*, *núm_de_características*). Computacionalmente es más eficiente pasar arreglos tridimensionales, por lo que la clase RNNLayer lo toma de esta forma y lo pasa un elemento por vez. Lo consigue enviando parte de los datos de forma que se tendrá la información acumulada de la capa (de la *data* pasada en los primeros pasos). Se pueden diseñar las *RNN* para que sus arreglos de salida sean de diferente dimensionalidad que los de entrada. Esto último, sería similar a las capas densas; se dispone de un caso de un clasificador de tres (3) clases, cuya salida (predicción) arroja un resultado de probabilidad del 70%, 10% y 20% para la primera, segunda y tercera clase, respectivamente. En concreto, esto se hace con un arreglo de la forma [0.7, 0.1, 0.2].

Existen distintas formas de organizar los elementos de las clases RNNNode y RNNLayer, lo importante es que cada capa necesita procesar su *data* en un intervalo de tiempo dado antes de la capa que le sigue, asimismo, todos los intervalos deben ser procesados en orden, y la última capa será la encargada de dar por salida la dimensión (*tamaño_característica*) para cada observación.

Vanilla RNN

Así como existen redes neuronales completamente conectadas ("*Fully-connected*") que son consideradas "regulares" por no tener, por ejemplo, capas convolucionales o recurrentes, también se tiene para las redes neuronales recurrentes una versión base que sería una "*Vanilla RNN*" (de "vainilla"; por lo simple de este sabor).

Ya habiendo expuesto la esencia del funcionamiento de una *RNN*, así como sus virtudes y las posibilidades al usarlas, corresponde mencionar sus dos puntos bajos. Primeramente, los gradientes para una *RNN* se muestran inestables, lo cual impacta negativamente en el proceso de entrenamiento. En segundo lugar, la memoria que se presenta como la característica más sobresaliente de esta estructura, es por naturaleza muy limitada; es decir, la forma base de las redes recurrentes es de memoria corta.

En vista que una red neuronal recurrente *vanilla* presenta limitantes importantes, se tuvieron que atender estos problemas para capitalizar las bondades que ya presentaba. Por un lado, se aplicaron técnicas como "*recurrent dropout*" (un método de regularización), y normalización de las capas recurrentes para mitigar la inestabilidad de los gradientes. Por el otro, surgieron estructuras que "aumentaban" la capacidad de estas estructuras en términos de memoria.

Un método muy usado en *Deep Learning* es el de *Backpropagation* (de "*backward propagation of errors*"; una posible traducción es "retropropagación"). Este método calcula la gradiente de la función de pérdida con respecto a los pesos de la red neuronal y procede con este cálculo "hacia atrás". Durante este proceso de retropropagación se da un problema de reducción de la gradiente ("*vanishing*", que traduce "desvanecimiento"); la gradiente, por tanto, no contribuye con mucho aprendizaje. Las capas que tienen una actualización de gradiente tan pequeña que es (casi) insignificante, no aprenden. Por la propiedad de ser con sentido hacia atrás, usualmente son las primeras capas las que no aprenden tanto (pues "van olvidando"), por lo que se habla de tener memoria corta. Encima de esto, mientras más larga es una secuencia, más se acentúa este problema. Ver Ilustración 57, donde se presenta el gradiente de pérdida en el proceso. La retropropagación a través del tiempo se visualiza en Ilustración 58.

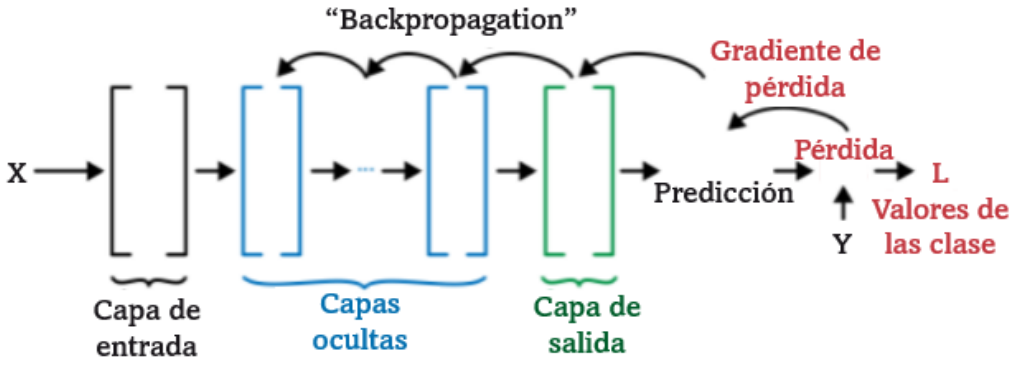


Ilustración 57 Retropropagación en términos de capas (en lugar que de operaciones). Fuente: adaptada de [30].

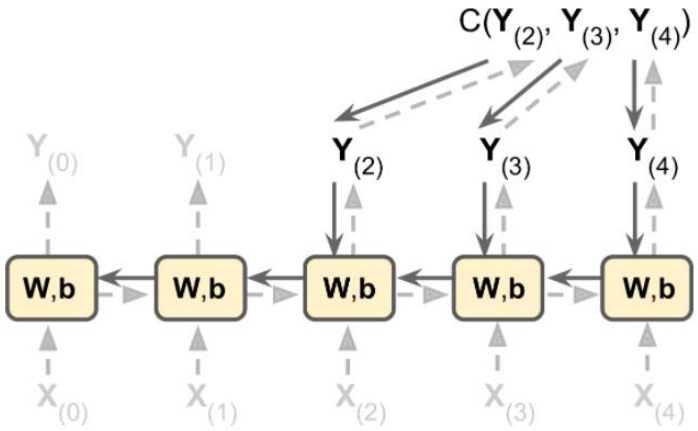


Ilustración 58 Retropropagación a través del tiempo, en cinco (5) momentos. Fuente: obtenida de [25].

La ecuación básica que describe la regla de actualización de la "*gradient descent*" es simplemente $nuevo\ peso = peso - tasa\ de\ aprendizaje * gradiente$. Ver Ilustración 59, con la forma en la cual se actualizan los pesos.

Ilustración 59 Regla de actualización del gradiente. Fuente: adaptada de [38].

Entonces, cabe preguntarse cómo se solventó este problema de reducción dramática de la gradiente. La respuesta es que se logró con la evolución de las redes recurrentes; en particular, yendo desde la *Vanilla* hasta dos nuevas propuestas: Las *LSTMs* ("*Long short-term memory*"; memoria larga a corto plazo) y *GRUs* ("*Gated Recurrent Units*"; unidades recurrentes cerradas), ambas dando solución a esta memoria de corto plazo ("*short-term memory*"). Para tener en cuenta: ambos tipos de redes neuronales comparten nociones y conceptos. Entre ellos, los presentados a continuación.

- Función tanh: Esta función de activación se comporta de forma similar a la sigmoide causando efectos también parecidos. En el caso de una "*tanh function*" las salidas se centran en cero, teniendo valores dentro del rango [-1, 1], si tiene sentido pensar en puntos porcentuales (como 0%, 33%, 75% o 100%) para la función sigmoide; la función tanh apunta más hacia lo positivo, lo negativo o lo neutral, de un algo: evento, efecto, transformación. En lugar de hacer transformaciones bruscas, esta función regula la salida. Es empleada tanto en las *LSTMs* como en las *GRUs*, al igual que la función sigmoide; ambas se pueden apreciar en Ilustración 60.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

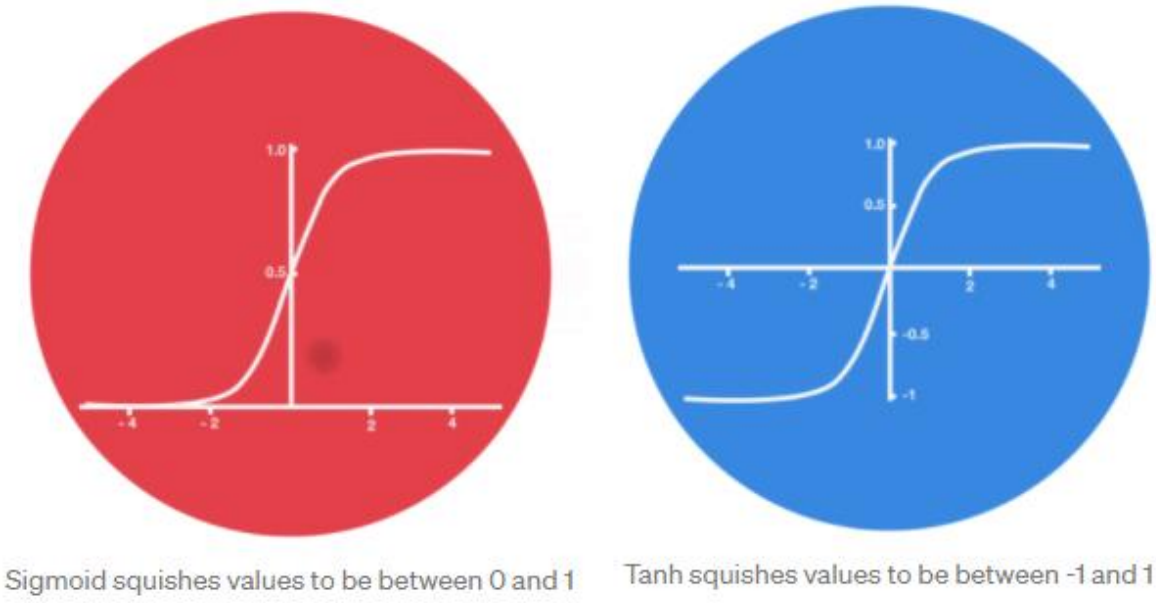


Ilustración 60 Gráficas de las funciones sigmoide (izquierda) y tanh (derecha). Fuente: adaptada de [38].

Las puertas ("*gates*") son los mecanismos implementados por estas estructuras que permitieron que tuviesen tan buen rendimiento. Estas puertas regulan el flujo de información dentro de los modelos con recurrencia. En el proceso de filtrado la

información relevante se conserva, dejando fuera lo que no se necesita; esta información son los datos en una secuencia. Lo que aprenden con esto las redes neuronales, en algunas de estas versiones evolucionadas, es a hacer predicciones tomando en cuenta información de los estados actuales y previos. Las diferencias entre estos enfoques vienen dadas por cómo escogen qué conservar. Tomar en cuenta Ilustración 61 con la leyenda de los símbolos utilizados para próximas ilustraciones.

Ilustración 61 Leyenda de símbolos. Fuente: obtenida de [38].

LSTM

- Cell state: Una célula *RNN* es cualquier elemento que tenga un estado y realice alguna operación que tome una matriz de entradas [39] lo que hace que se distingan de las neuronas comunes, por poder recordar información previa gracias al estado con el que cuentan; a este se le conoce como estado de la célula (o de la unidad). Actúa como un medio que transforma información relativa hacia la cadena de secuencia. En términos no precisos, es la memoria en estas estructuras; aunque esto sea una simplificación con abstracción alta, es una manera práctica de verlos. Una representación de este proceso se visualiza en Ilustración 62.

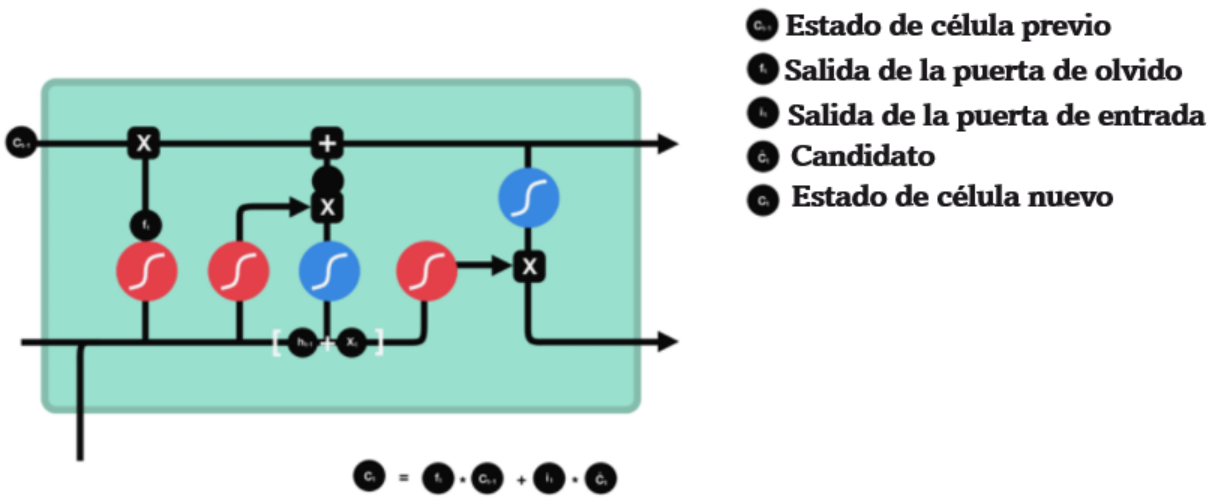


Ilustración 62 Calculando el estado de célula. Fuente: obtenida de [38].

Las células pueden enviar información a través del procesamiento de la secuencia, por tanto, técnicamente esto implica que la información de los primeros intervalos puede ser llevada hasta los últimos intervalos reduciendo los efectos de memoria de corto plazo. La información va siendo conservada o removida durante el entrenamiento de acuerdo con lo que decidan las puertas.

Las puertas dentro de *LSTM* y *GRU* son en sí mismas redes neuronales cuyo propósito es el de regular el flujo de información a través de la cadena de secuencia [38], como las redes neuronales que son, precisan de una función de activación. Al igual que las *CNN*, estas se valen de la función sigmoide para tal propósito; acá vale la pena hacer de cuenta que se olvidará la información para cuando la función de activación arroje por valor un cero (porque al multiplicar el valor se pierde la información), y será conservada cuando se tenga un uno (misma razón, al multiplicar por uno queda el mismo valor); esto es un rango, y por tanto un espectro.

- Forget gate: Decide en sí qué información debe prevalecer y cuál debe irse. Se pasa información del estado oculto previo y de la entrada actual por una función sigmoide. Ver Ilustración 63, con la participación de esta puerta en la estructura.

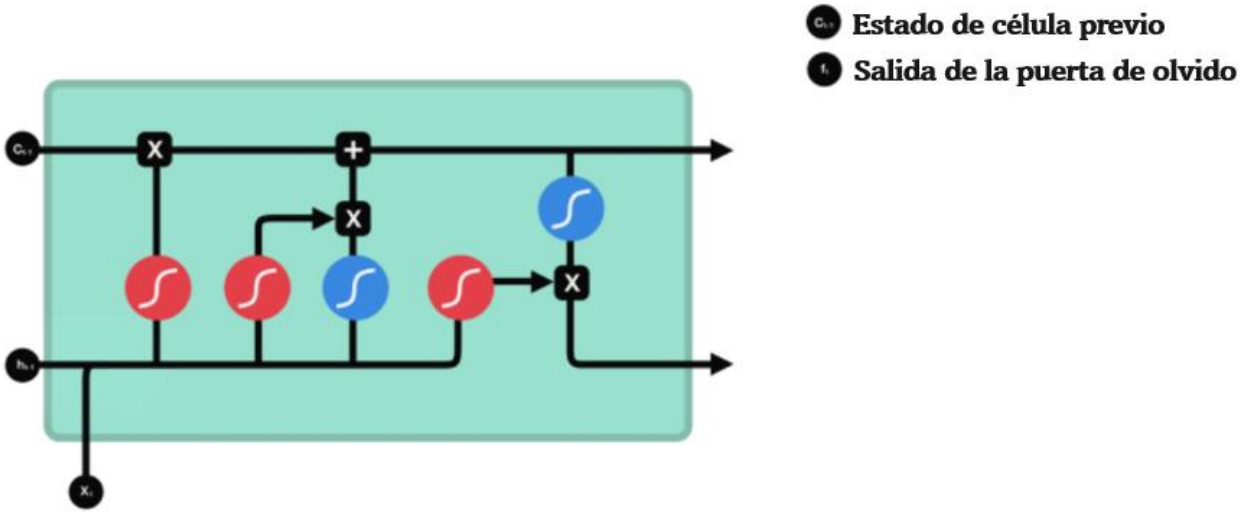


Ilustración 63 Operaciones de la puerta de olvido. Fuente: obtenida de [38].

- Input gate: Es la encargada de actualizar el estado de la célula. En la fase del algoritmo del cual se basan las *LSTMs* que involucra esta puerta, ocurren en primer lugar dos operaciones: por un lado, la red neuronal pasa el estado oculto y la entrada actual a través de la función tanh ([-1, 1]) que regula la red (con su transformación sutil); por otro, apoyada en la función sigmoide ([0, 1]) se establece la importancia de la información (hacia cero es menor, hacia uno es mayor). En un último paso en el que participa esta puerta, las salidas de ambas funciones se multiplican para que la función tanh decida qué conservar de la información procesada por la función sigmoide. Ver Ilustración 64, que muestra las salidas de las puertas de olvido y, de entrada.

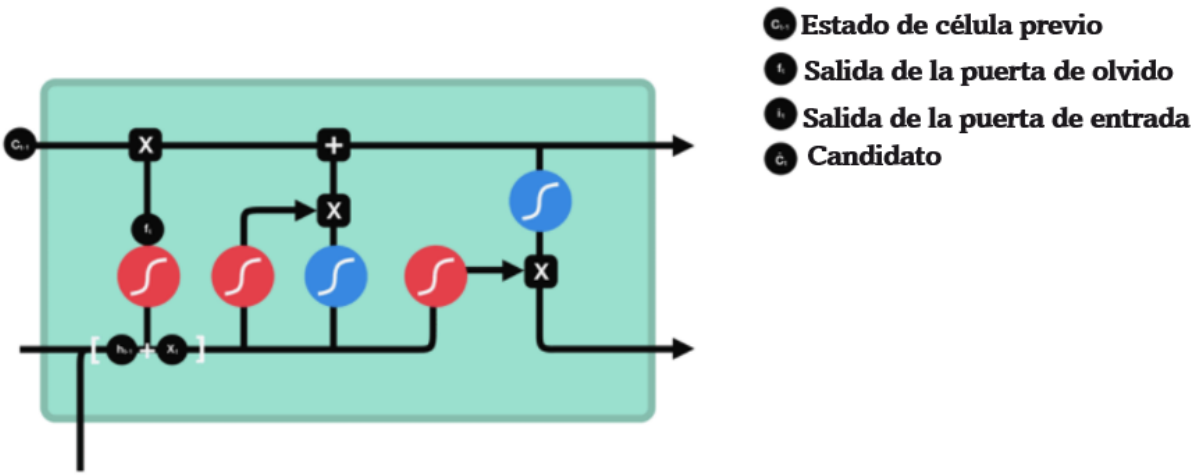


Ilustración 64 Operaciones de la puerta de entrada. Fuente: obtenida de [38].

El estado de la célula se calcula con la información que se tiene, como sigue: se multiplica a la célula por el vector de la *forget gate*, si los valores son cero o próximos a cero, se descartan; luego la salida de esta operación junto con lo del *input gate* generan la actualización del estado de la célula a nuevos valores, lo que causa que ahora se tenga un nuevo estado. Ver Ilustración 65, sobre el cálculo del estado de célula.

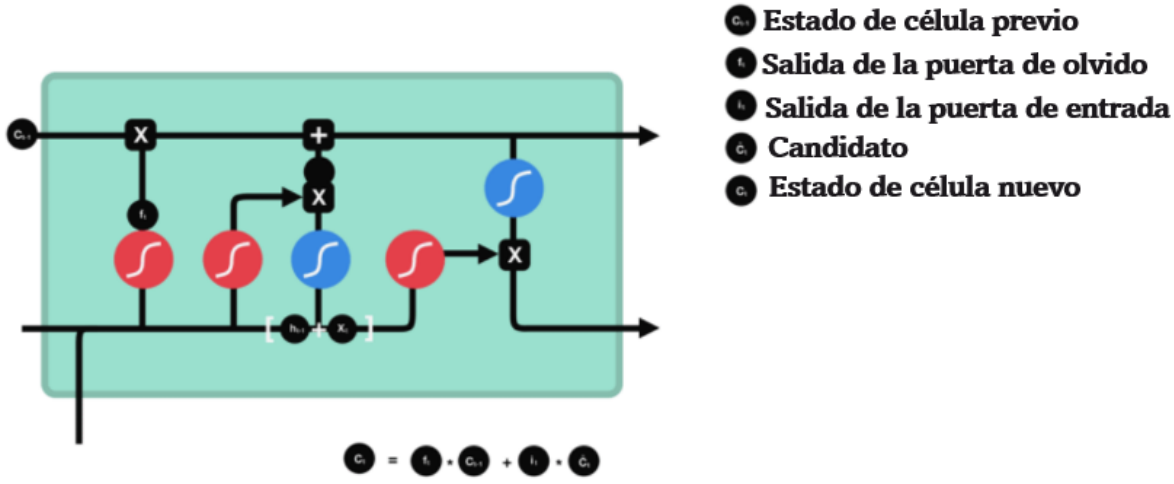


Ilustración 65 Calculando el estado de la célula. Fuente: obtenida de [38].

- Output gate: Esta decide qué debería ser el estado oculto posterior, pues va en línea con lo que será la predicción por parte del modelo; para ello pasa lo siguiente: se pasa el estado oculto anterior junto a la entrada actual a través de otra función sigmoide; luego, se pasa el estado de célula recientemente modificado (el nuevo de pasos anteriores) por otra función tanh; nuevamente, se procesan las salidas de ambas funciones (distintas a las previas) multiplicándose para que se decida qué información debe llevar el estado oculto. La salida es el estado oculto.

Los nuevos estados, el oculto y el de célula, pasan al siguiente intervalo. Repitiéndose el proceso tantas veces como número de divisiones con las que cuente la secuencia a procesar. En el caso de trabajar con videos, equivaldría el número de veces que se lleve a cabo este proceso para un video, al número de cuadros que lo compongan.

Las redes *LSTMs* consumen menos recursos computacionales que sus pares que las preceden dado que llevan a cabo operaciones más sencillas; en el proceso de entrenamiento puede ser perceptible lo mucho que se puede ahorrar computacionalmente versus las redes convolucionales, por dar un ejemplo. La puerta de salida participa de operaciones fundamentales dentro de esta estructura como se ve en Ilustración 66. Ver Ilustración 67, con otra representación de una célula *LSTM*.

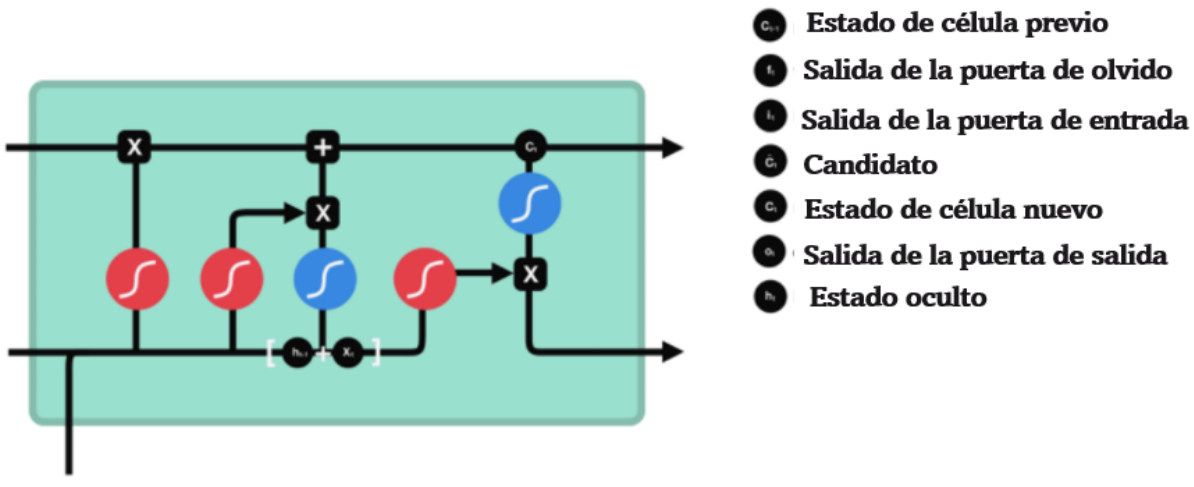


Ilustración 66 Operaciones de la puerta de salida. Fuente: obtenida de [38].

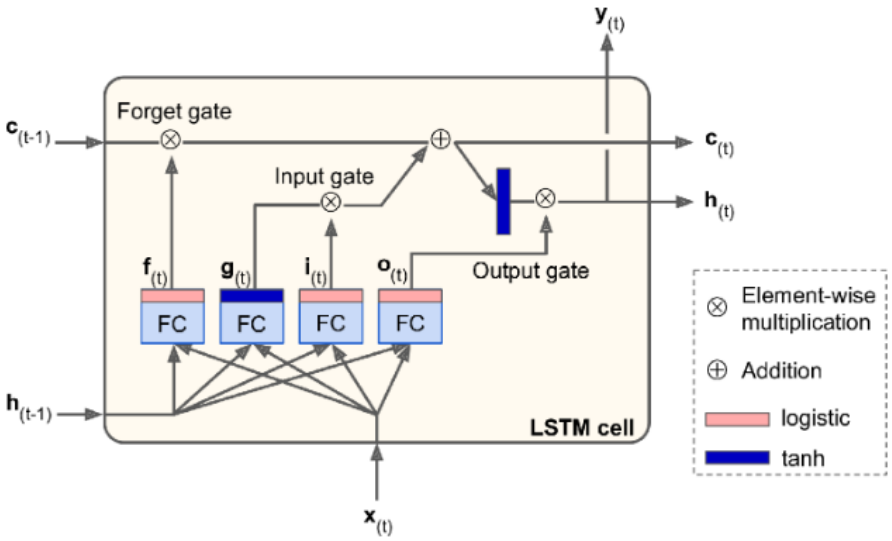


Ilustración 67 Célula tipo LSTM. Fuente: obtenida de [25].

GRU

Esta estructura es, de hecho, una simplificación de las *LSTM*. A la fecha, comparten escenario en los resultados del estado de arte para desarrollos que involucran redes neuronales recurrentes, donde la arrolladora mayoría de los proyectos de esta índole usan una u otra (en ocasiones ambas en simultáneo). Ver Ilustración 68, con representación de una célula de este tipo.

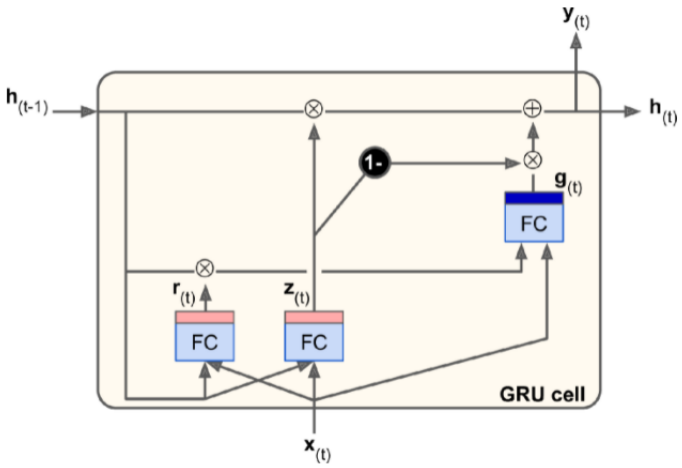


Ilustración 68 Célula tipo GRU. Fuente: obtenida de [25].

Esta estructura se diferencia de la de una *LSTM* principalmente por no hacer seguimiento del estado de una célula, sino que usa un estado oculto para transportar la información. Y consta de dos puertas: la "*reset gate*" (puerta de reinicio) y la "*update gate*" (puerta de actualización). Ver la Ilustración 69, una célula de este tipo con sus elementos.

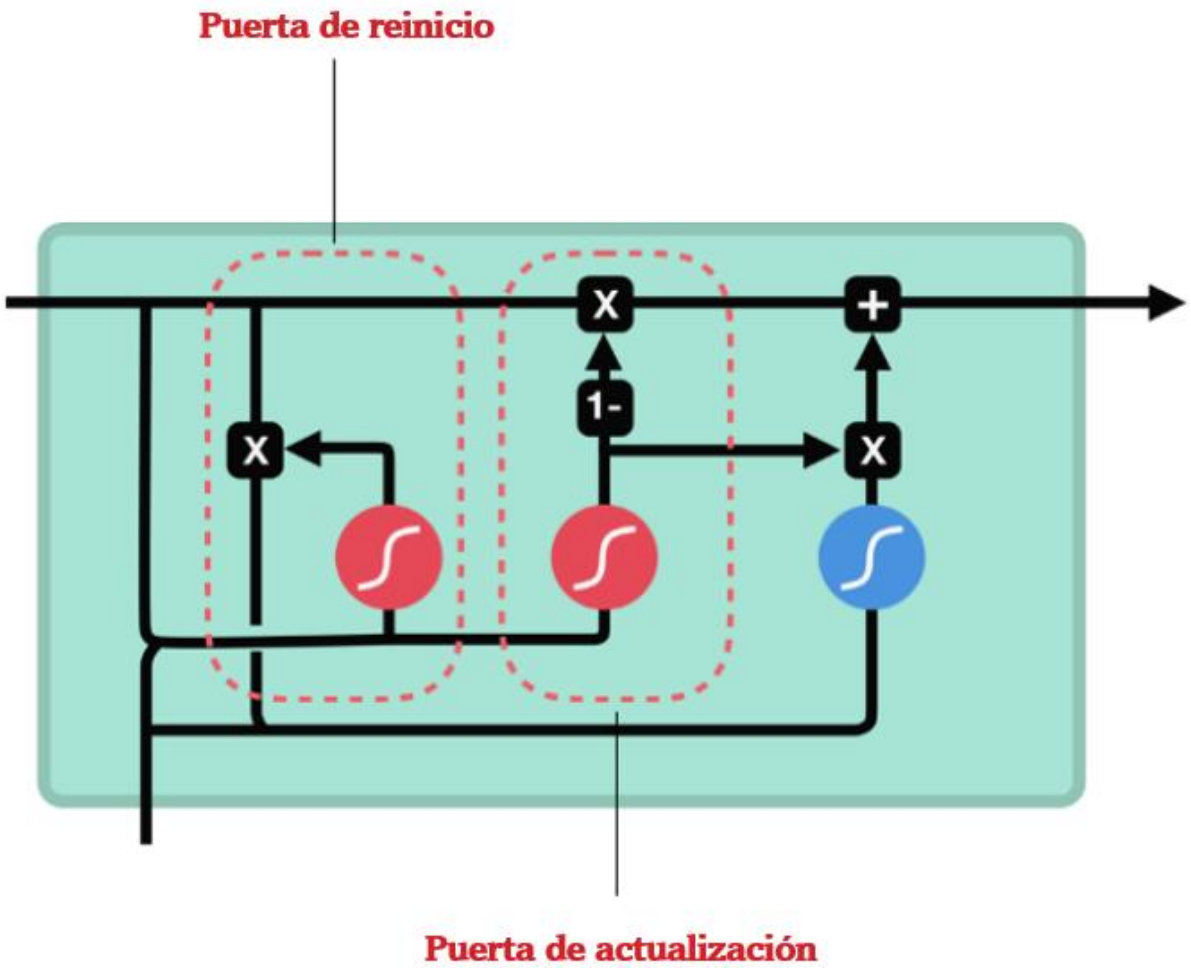


Ilustración 69 Célula GRU y sus puertas. Fuente: obtenida de [38].

- Update gate: Su acción no difiere mucho de la de una puerta de entrada para una modelo *LSTM*. Básicamente decide *qué* información se "olvida" y cuál prevalece.
- Reset gate: Por su parte, una puerta de reinicio decide *qué tanta* información pasada se olvida.

Hay incluso menos operaciones con tensores que realizar bajo este mecanismo. Pese a que esto deriva en mayor velocidad de entrenamiento, suele haber un intercambio entre aspectos positivos y negativos a la hora de usar *GRU* o *LSTM* para un desarrollo.

Visión Artificial (dentro de ML)

Pese a no haber nacido del Aprendizaje Automático, hay una relación muy estrecha entre el campo de Visión Artificial y el de *Machine Learning*; se puede tener mejor esquema mental a partir de lo presentado en Ilustración 70. Por su parte, las tareas a lo largo de los procesos del aprendizaje profundo en contraste con las del aprendizaje automático se alcanzan a ver en Ilustración 71.

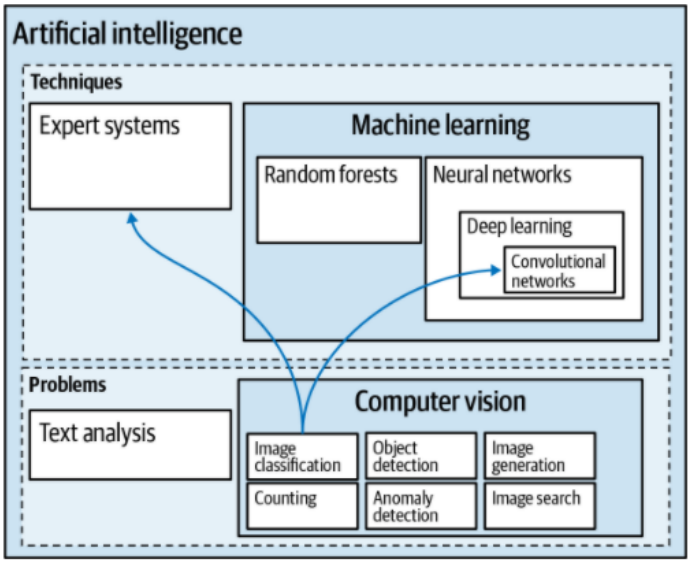


Ilustración 70 Visión artificial respecto a Aprendizaje Automático. Fuente: obtenido de [29].

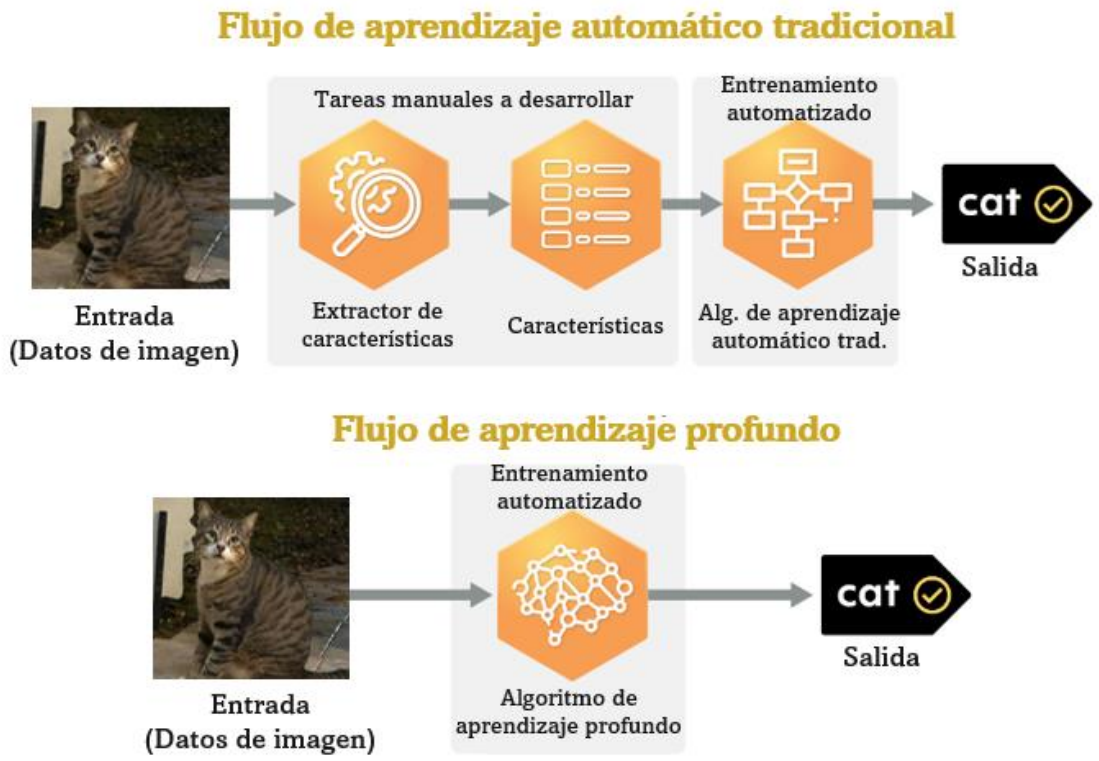


Ilustración 71 Machine Learning versus Deep Learning ante el mismo problema de Computer Vision. Fuente: adaptado de [27].

Una buena cantidad de datos es el requerimiento por excelencia para el desarrollo de sistemas de visión artificial. Ver Ilustración 72 con grupos imágenes del conjunto de datos a utilizar para entrenar un clasificador de múltiple que identifica entre perros, gatos y lobos. En la práctica puede tenerse que etiquetar una a una las imágenes, en lugar de a grupos enteros de datos; la Ilustración 73 muestra entradas para un clasificador binario que infiere si un perro es adulto o cachorro.

Ilustración 72 Conjuntos de datos agrupados según la etiqueta correspondiente: “perro” (izquierda), “gato” (centro) y “lobo” (derecha). Fuente: el autor.

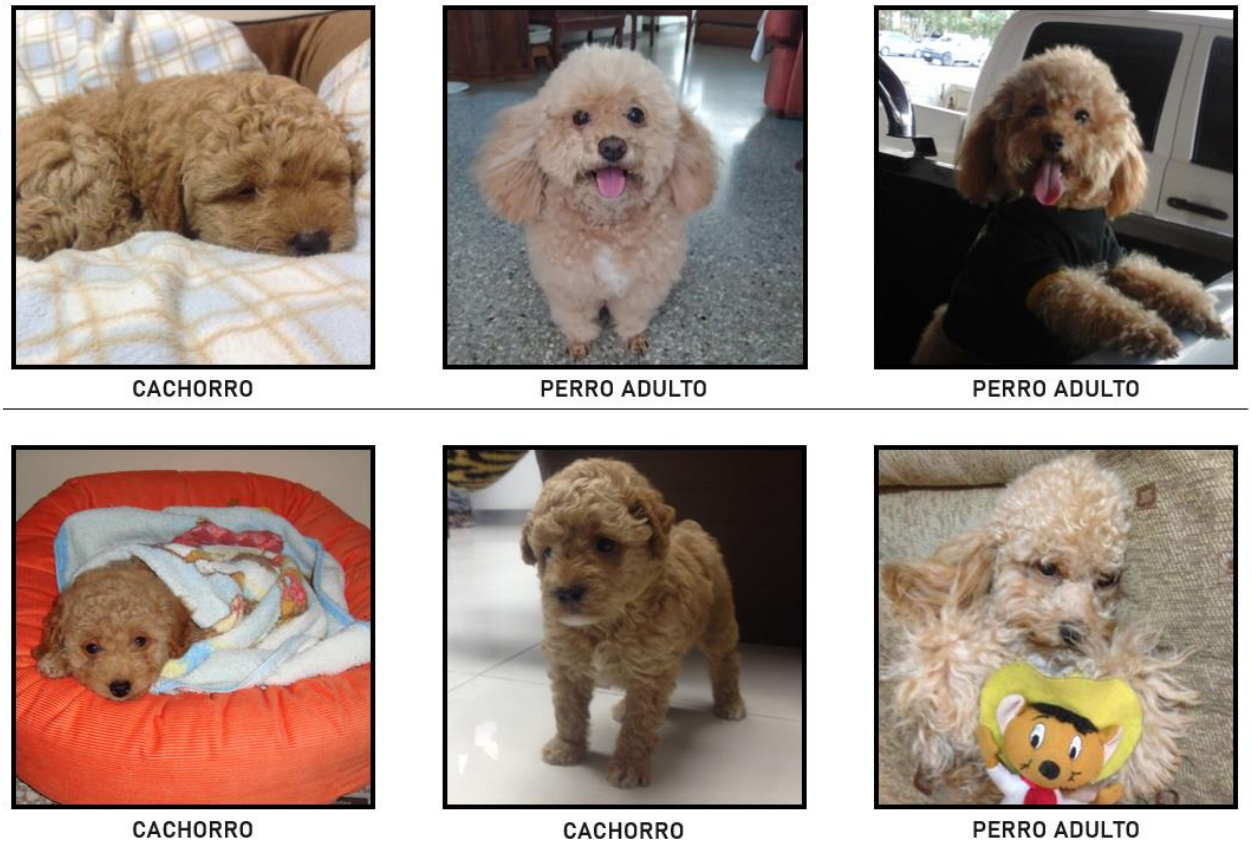


Ilustración 73 Datos para etiquetar individualmente de acuerdo con el caso. Fuente: el autor.

Más allá del número de datos, que puede variar enormemente entre casos, se deben tener consideraciones como la variedad de estos. Se alcanza a ver en Ilustración 72 un ejemplo de variedad de datos en términos de posiciones del objeto; en este caso, se trata

de una mascota de pie, sentada y acostada, también con diferentes expresiones / acciones como jadeando o descansando. Si un sistema para detección de perros se entrenase con este tipo de imágenes, fuese más interesante, para encontrar los parámetros y patrones que realmente se necesitan para identificar si el objeto dentro de una imagen es o no un canino, que un conjunto de datos que solo considerase perritos en una sola posición, como de pie mirando al mismo lado. Asimismo, es menos útil entrenar a un sistema a datos de entrenamiento que solo contemplen una misma mascota, o bien, una única raza; estos casos se aprecian en Ilustración 74 e Ilustración 75, respectivamente.

Ilustración 74 Imágenes (fotos) con variedad de poses, pero con un único ejemplar. Fuente: el autor.

Ilustración 75 Imágenes (fotos) con variedad de perritos, pero de una misma raza. Fuente: el autor.

Se busca, entonces, contar con una variedad que pasa por diversas características, para enriquecer el entrenamiento; pensar en lo diferente que pueden ser dos ejemplares de caninos, y si se espera que el modelo “aprenda” a identificar a cada ejemplar de esta especie, se debe evitar que aprenda patrones ruido (como que “todos los perros son pequeños y peludos”, lo cual evidentemente es falso), a la vez que se procura que aprenda patrones reales (las características distintivas de esta especie respecto a las demás). En Ilustración 76 se ve un grupo de datos con mayor variedad.

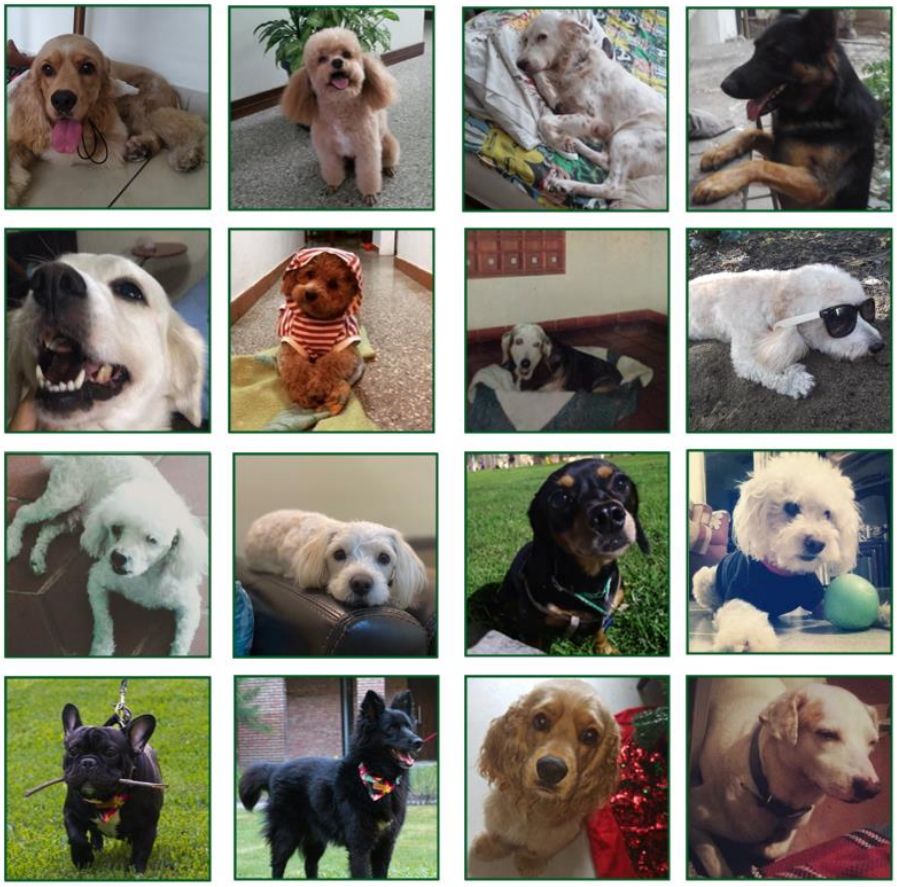


Ilustración 76 Conjunto de datos con variedad de características, pero que puede ameritar ser mucho más nutrido dependiendo del caso de uso y el alcance del sistema. Fuente: el autor.

Detección de Objetos

- Reconocimiento de Objetos: El "*Object Recognition*" es una técnica con la que se logra identificar objetos en imágenes o videos como resultado de algoritmos de Aprendizaje Automático y Aprendizaje Profundo. Ver Ilustración 77, que presenta el resultado en tiempo real para el objeto identificado.

Ilustración 77 Resultado de un sistema de reconocimiento de objetos en ejecución. Fuente: el autor.

- Localización de Objetos: La "*Object Localization*" trata de identificar la ubicación de uno o más objetos dentro de una imagen, enmarcando el objeto con un rectángulo (al que se refiere como "*box*", de caja). La Ilustración 78 presenta la tarea de localización, ubicando en la entrada visual (fotos) a los objetos (mascotas) que aparecen en escena.

Ilustración 78 Salidas independientes de un localizador de objetos que reconoce perros y gatos, y los ubica en una imagen. Fuente: el autor.

- La Detección de Objetos es el proceso de encontrar instancias de objetos en imágenes o videos. Para el Aprendizaje Profundo aplica verlo como un subconjunto del reconocimiento de objetos en el cual un objeto, además de identificado, se ubica (localiza) en una imagen estática; o cuadro (*frame*), de ser una entrada de tipo video. Otra manera de ver esta tarea es como combinación de las tareas de clasificación de imágenes y localización de objetos. Ver Ilustración 79, donde un detector de objetos identifica distintas instancias en simultáneo.

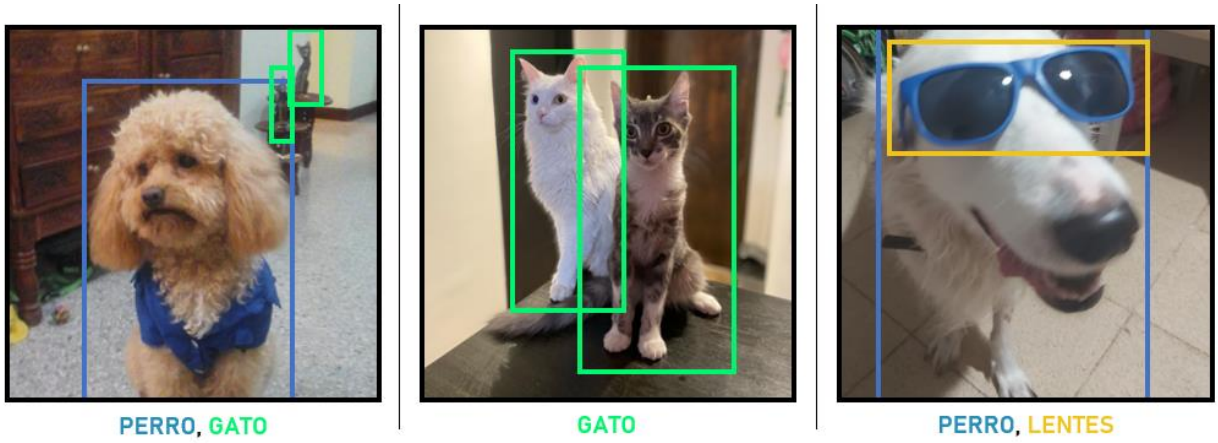


Ilustración 79 Salidas de un detector de objetos que identifica distintos elementos. Fuente: el autor.

- Segmentación de imágenes: A esta extensión directa de la detección de objetos también se le refiere como “segmentación de objetos” (por “*Object Segmentation*”, o bien, “*Instance Segmentation*”). La razón es que es una clasificación por píxeles que se le aplica a una imagen; a la instancia del objeto (u objetos) se le asigna una categoría (clase) y se le suele dibujar bordes que encierran todos los píxeles que la compongan. De este problema se subdividen otros como segmentación por texturas o segmentación por color; se trabajará con lo que se necesite o lo que convenga según el caso. Dos ejemplos de segmentación de instancias se aprecian en Ilustración 80.

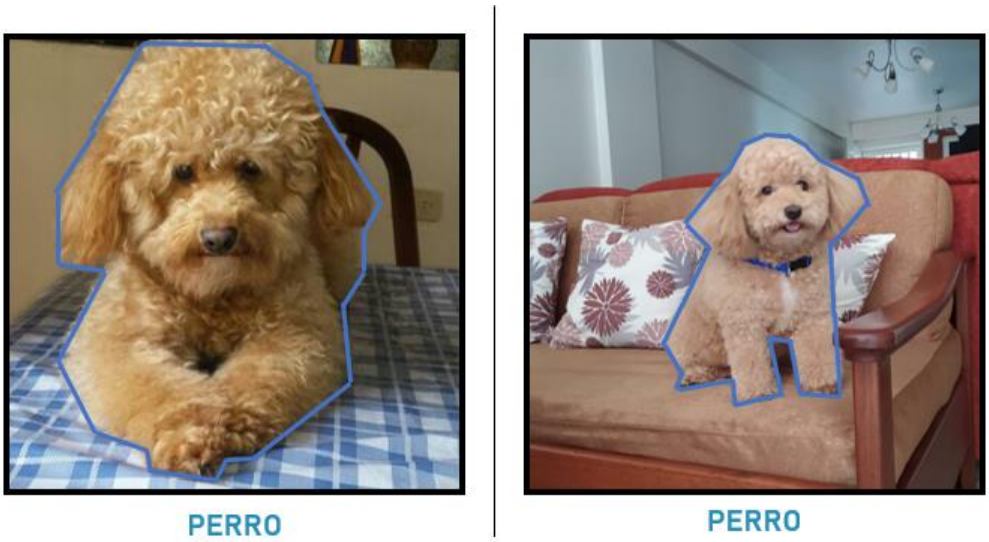


Ilustración 80 Se ubican objetos con mayor precisión para segmentación de imágenes. Fuente: el autor.

- Segmentación semántica: Es uno de los problemas especializados de segmentación de imágenes; es de los más sofisticados en la actualidad y para el cual se proponen muchos desarrollos, así como enormes conjuntos de datos para entrenar modelos de este tipo. Su propósito es el de clasificar objetos, y suele hacerlo valiéndose de algoritmos de redes convolucionales siendo más eficiente y preciso que sus predecesores. Es considerado de “alto nivel” al igual que la segmentación panóptica y la segmentación instanciada, en contraposición con los de “bajo nivel” como los de segmentación por color o los de umbralización.
- Región de interés: Típicamente presentado como “ROI” en textos (por “*Region of Interest*”), es el término usado en visión artificial (también en reconocimiento óptico de caracteres y otras áreas de aplicación) para decantar o abarcar el objeto o la porción de una imagen en consideración. En el supuesto de estar hablando de detección de objetos, sería la caja que encierra al objeto (o a cada uno de ellos), mientras que, para segmentación de imágenes, sería lo que se contenga entre los bordes por segmentación. Los puntos de interés (“*Point of Interest*”; “POI”) son puntos que referencian o representan lo que sobresale o se busca en un estudio, como puntos geográficos o puntos clave; que están dentro de una ROI.

Fully Convolutional Networks

Esta idea inicialmente pensada para la segmentación semántica posibilita que una red neuronal convolucional pueda procesar imágenes de cualquier tipo [24]. Propone, para este fin, la sustitución de la capa densa por una capa convolucional que deberá tener un número de filtros que coincida con el de unidades en la capa que sustituye. Esto hará que este tipo de red solo cuente con capas convolucionales y de *pooling*, las cuales se entienden con entradas (imágenes) de cualquier tamaño; por esta misma razón se le asignó el nombre de “redes totalmente convolucionales”.

La eficiencia de las *FCN* es mucho mayor que la conseguida con un enfoque *CNN* simple. El porqué, se resume en el hecho de solo tener que procesar la imagen de entrada en una ocasión, más allá de las dimensiones de esta. Ver Ilustración 81, sobre la diferencia de procesar entradas de distintos tamaños. Otra diferencia menor sería la forma de la salida de las capas (la original y su reemplazo); ambas serían tipo tensor pero la salida de la capa densa sería $[tamaño\ observación, número\ de\ neuronas]$ y para la capa convolucional se tendría $[tamaño\ observación, 1, 1, número\ de\ filtros]$, donde el número de neuronas y el de filtros serían los mismos bajo el mismo caso (red con la misma finalidad, pero construida bajo otro enfoque).

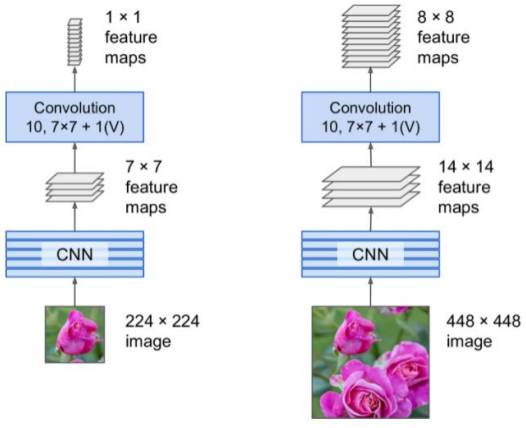


Ilustración 81 Misma red procesando una imagen pequeña (izquierda) y grande (derecha). Fuente: obtenido de [25].

YOLO

Es una arquitectura basada en redes neuronales convolucionales para detección de objetos en tiempo real, alcanzando este rendimiento incluso cuando es utilizada en videos. Es similar a la *FCN*; tanto así, que su nombre proviene de la característica de "ver solo una vez" a la imagen (de "*You Only Look Once*"). A la fecha la versión *YOLOv5* es la última propuesta (presentada con solo un mes de diferencia de *YOLOv4*). Procurando optimizar aspectos como la rapidez y la precisión, esta familia de arquitecturas se ha posicionado en un alto nivel de popularidad en desarrollos independientes y en empresas.

La *YOLOv5* se vale de un sistema "*grid*" (de cuadrícula), en el cual cada celda lleva a cabo la detección de objetos correspondiente a sí misma [40]. Esta arquitectura, además, tiene compatibilidad con los entornos más importantes, aumentando su competitividad.

SSD

El "*Single Shot MultiBox Detector*", abreviado simplemente *SSD*, tiene un nombre autoexplicativo; se traduce como detector multicaja (múltiples recuadros delimitando objetos) de toma (foto o tiro, de "*shot*") única. Tiene en común con las arquitecturas de la familia *YOLO* que consta de un sistema de cuadrícula, y "ve" solo una vez la imagen.

Este modelo predice el desplazamiento de las cajas predeterminadas de diferentes escalas (agrandado o reducción del tamaño que cambia el número de pixeles que contiene) y relación de aspecto ($ancho{:}\,altura$); esto en diferentes capas *feature* (de características). Se aplican convoluciones 3x3 en "*feature dimensions*" (las "dimensiones" son el número de vectores de características en una imagen) con lo que van calculándose salidas que se agrupan hacia el final de la red neuronal para culminar el proceso. Estas salidas son de dos tipos: las cajas que demarcan un objeto y la clase a la que este corresponde. Con esa información se lleva a cabo la técnica "*Non-maximum suppression*" (supresión no máxima, o *NMS*).

En conclusión, el *SSD* es un modelo que predice la clase (con un puntaje porcentual) a la que pertenece un objeto, así como las cajas que encierran al objeto detectado, a partir

de un conjunto fijo de cajas predeterminadas. En muchos casos no coinciden (o coincidirían) la caja de etiquetado y la de predicción, como se ve en Ilustración 82. La *NMS* se deshará de esas cajas predeterminadas que no encierran al objeto o no lo hacen de forma tan exacta, quedándose con la mejor opción para cada caso, que puede ser más de uno por imagen, como se ve en Ilustración 83. En última instancia, un filtro (convolucional de 3x3) da por salida el *feature map* (mapa de características).

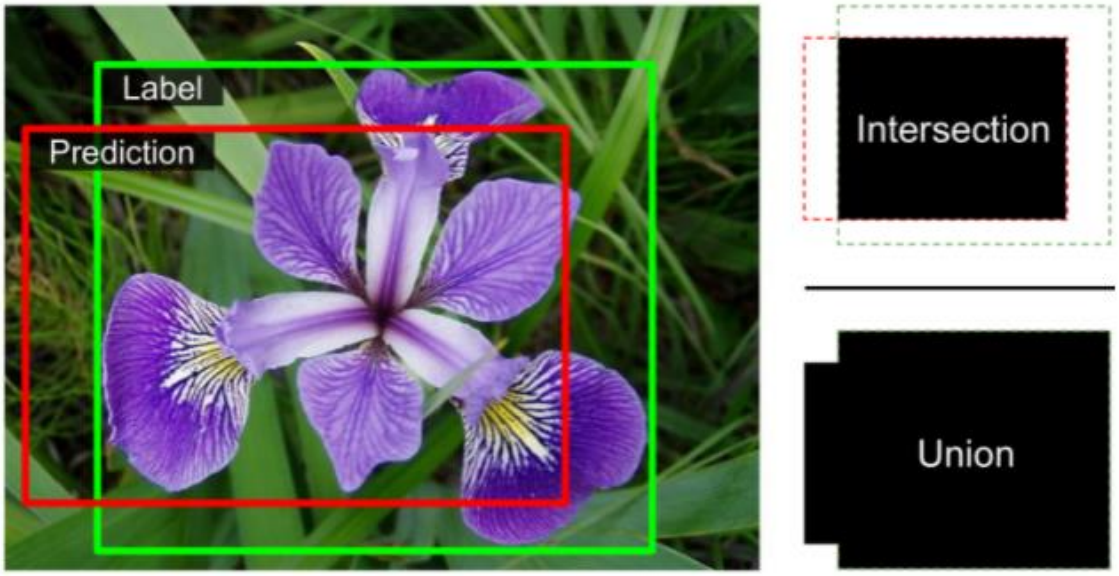


Ilustración 82 Métrica de intersección sobre unión para cajas bounding. Fuente: obtenido de [25].

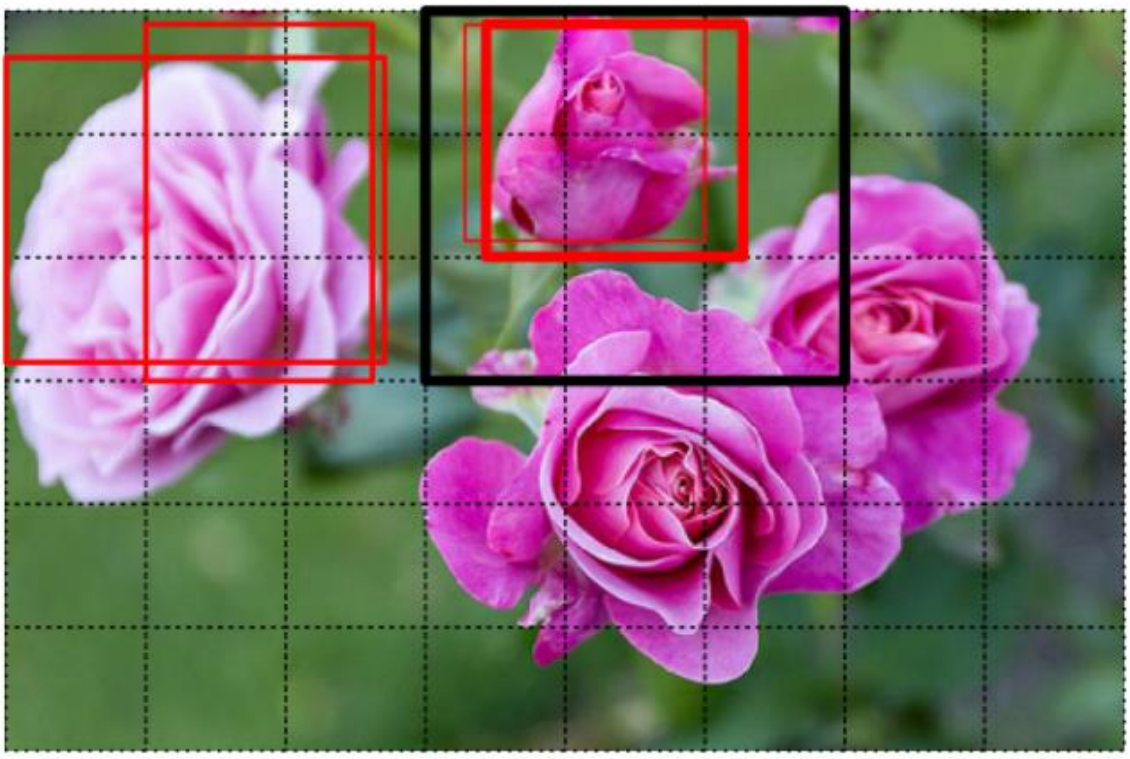


Ilustración 83 Una CNN detectando múltiples objetos en simultáneo. Fuente: obtenido de [25].

Detección de Acciones

Es sin lugar a duda una revelación que ha significado el impulso de industrias y cambios de paradigmas, propiciando avances favorecedores para áreas como la medicina (ver Ilustración 84 como ejemplo) y el entretenimiento.

Hay que discernir las topologías detrás de esta tecnología para comprender qué es en sí, qué abarca, y cómo y por qué fue concebida.

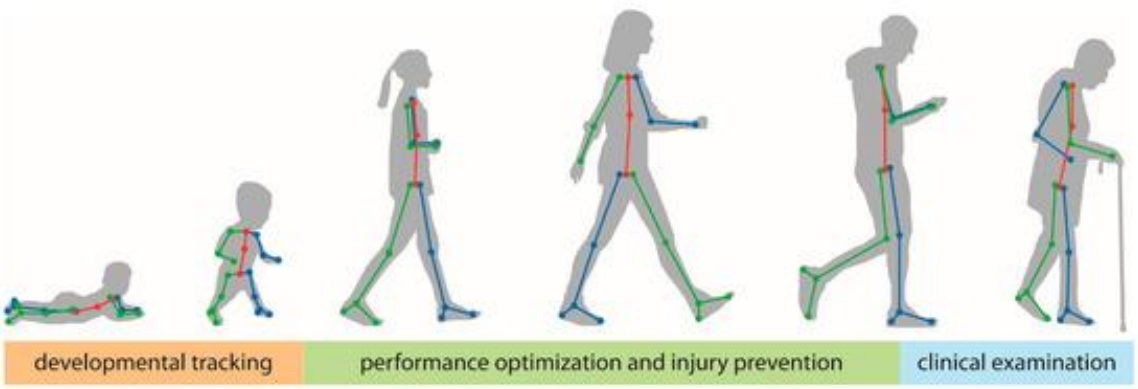


Ilustración 84 Aplicaciones de la estimación de poses en el área de salud. Fuente: obtenida de [41].

Topologías de la Detección de Acciones

Hand Tracking y Palm Detection

Las soluciones que involucran detección de manos pueden categorizarse en tipos que responden a la tarea de visión artificial para la cual fueron desarrolladas o para la que estén siendo aplicadas. Algunos de los términos que rodean al "*Hand Tracking*" (cuya traducción sería algo próximo a "seguimiento de manos") serían:

- Palm Detection: Detección de palma(s). Siendo autoexplicativo, cabe resaltar que esta solución fue pie a desarrollos que involucrasen toda la mano como componente, por ejemplo, como parte del proceso de identificar los elementos dentro de una mano.
- Hand Detection: Uno de los términos que más engloba todo este conjunto de soluciones es el de "detección de manos" es particularmente útil y preciso, usarlo en torno a proyectos cuya base es la detección de objetos.
- Hand Gesture Recognition: El reconocimiento de gestos con las manos puede estudiarse para precisar intencionalidad de un gesto, o para distinguir entre señas como "bien" o "mal" (como podría ser pulgar arriba y abajo, respectivamente), incluso para prevenir acontecimientos valiéndose de estos gestos y, por ejemplo, las expresiones faciales captadas por una cámara de seguridad (en conjunto con esa otra topología). De cualquier forma, es de esperar que sea principalmente una solución ligada al reconocimiento de señas de lenguajes de señas.

- Hand Pose Estimation: La estimación de postura de las manos es la tarea de estudiar posicionalmente una mano; el término es usado junto a *Hand Detection* indiferenciadamente, pero cabe hacer la discriminación que la *detección* implica identificar la presencia del objeto en una imagen (estática o no); y la *estimación* busca saber cómo se encuentra la mano en un momento dado, esto es, la posición de los dedos, palma y muñeca con relación a los otros elementos.

Hand Tracking es quizás el término que encuadra el conjunto de soluciones de este tipo, puesto que estos modelos incluyen de alguna u otra forma módulos de detección, estimación y/o reconocimiento. Asimismo, también los desarrollos más sofisticados con relación a *Hand Tracking* se valen de estas partes.

Destaca el uso de esta solución en las tecnologías de realidad aumentada (RA) y en la industria "*gaming*" (de videojuegos), en general, incluso en proyectos no exclusivamente dentro del dominio de aprendizaje automático. A su vez, hacen parte de proyectos más grandes, integrados con otras soluciones (desembocando esto en otras nuevas), entre las que pueden ser nombradas el reconocimiento de actividades humanas ("*Human Activity Recognition*") o análisis de actividad humana ("*Human Activity Analysis*").

Face Mesh y Face Detection

Ocurre una situación similar con las soluciones que tienen que ver con caras versus las de manos. Es entendible porque el progreso en ambas se ha dado con un paralelismo que responde a las similitudes que puede haber a la hora de estudiarlas y aplicarlas como componentes del cuerpo humano a partir de los que se pueden predecir una cantidad importante de eventos o evaluar un buen número de parámetros.

- Face Detection: Así como se puede detectar virtualmente cualquier objeto tangible, y se pueden detectar palmas o manos, también se puede llevar a cabo la "detección de caras". Hay aplicaciones que usan modelos de *Face Detection* para identificar personal de una empresa o institución, por nombrar uno entre tantos ejemplos en los cuales se quiere simplemente identificar que, para una imagen captada por cámara o no, se está visualizando un rostro humano.

- Face Recognition: El término “reconocimiento facial” suele ser la forma preferida, por ser más precisa, para referirse a los modelos, sistemas o tecnologías que son capaces de verificar la identidad de personas al captarlas por cámara, o a partir de elementos audiovisuales tipo archivo de imágenes o video. El uso más común y popular es como método de identificación biométrico (considerando datos y patrones biométricos faciales) para desbloquear un *smartphone*, y casi siempre tiene que ver con sistemas para acceso a una aplicación, servicio, o algún otro sistema.
- Face Pose Estimation: La estimación de la “*pose*” de la cara sería la versión de soluciones de la cara como componente que tiene que ver con ubicar posicionalmente en el plano (dos dimensiones) o, más comúnmente, en el espacio (tres dimensiones) los puntos más determinantes de la cara. Esta identificación de elementos incluye puntos alrededores de los ojos, punta de la nariz, comisura de los labios, o centro de las cejas, sumando cientos de puntos a los cuales se les denominan “*landmarks*” (es decir, puntos de referencia) y lo que conllevó a asignarle el nombre de “*Face Landmark Estimation*” a las soluciones de esta índole.
- Face Mesh: Es más una consecuencia directa de haber ubicado los puntos con relativa importancia para un caso, pero presentando una “malla facial” sobre el rostro del cual se identificaron (o se están continuamente rastreando) los puntos, y probablemente se extraigan los valores, usualmente posicionales, para realizar tareas como determinación de expresiones faciales con base a estos datos.

Face Tracking sería entonces la solución con la que se hace seguimiento a todos esos puntos de referencias calculados para un rostro a partir de una imagen que lo contenga. La realidad aumentada vuelve a ser un ejemplo de uso de estas aplicaciones en la cual se utiliza, entre otras cosas, para crear una superficie que reconstruya las expresiones de una persona (como un jugador de un videojuego). En la industria de cine también son empleados modelos y técnicas de esta solución, como cuando se extrajeron en tiempo real los puntos de referencia de la cara del actor de las últimas entregas de *Spider-man*, el popular personaje de cómics. En este último caso no se procedía a reconstruir las

expresiones faciales del actor, sino a leerlas para hacer una versión de lo que sería en el traje (particularmente los ojos), ejemplificando un proyecto en el cual otro u otros módulos deben participar para aprovechar lo que ya existe dentro del área.

Pose Detection, Pose Classification y Pose Estimation

"*Pose*" traduce pose o postura, y la realidad es que varía a qué se puede estar refiriendo el término en determinado momento o para cierto caso. Usualmente incluye al menos de cuello a tobillos, extendiéndose por los brazos hasta las muñecas, pero también se pudiese estar haciendo extracción de algunos *landmarks* de manos y pies, sumando decenas de puntos de referencia que se identifican, se obtienen y se estudian del cuerpo como un todo, pero que para los efectos se trata como un componente más del cuerpo entero. A las soluciones también suele denominárseles "*Human Pose*" (pose humana) en lugar de solo "*Pose*", obviamente esto excluye estudios que no vayan orientados a seres humanos, lo cual evidentemente es el grueso del área.

- Pose Detection: Esta versión de modelos de detección puede estarse refiriendo a identificar los puntos en tronco y extremidades de una persona como objeto dentro de una imagen. Algunos de los puntos corresponden a ambos hombros, ambos codos y ambas rodillas, y en versiones en las que se extraen adicionalmente puntos de las terminaciones de las extremidades se encuentran los *landmarks* de nudillos de los pulgares de las manos y los dedos índices de los pies.
- Pose Classification: Este puede ser un caso particular de otras soluciones, en el cual se categoriza el tipo de pose o cuál es dentro de un grupo de opciones. Es fácil visualizar la "clasificación de posturas" en acción si se piensa en "*hatha yoga*", la práctica del yoga en la cual una persona se prepara para la meditación mediante varias posiciones corporales. La versión ampliada en las que se toman en cuenta manos y pies responde a aplicaciones como modelos de *Yoga Pose Recognition*, en donde importan los puntos de referencia de las manos y pies para determinar si se está logrando la posición de forma adecuada.
- Pose Estimation: Prácticamente se basa en los mismos conceptos que los modelos de sus pares cara y manos, solo que la estimación de pose (o postura) tiene retos mayores en cuanto al valor posicional que representa la profundidad

de un punto. Pensar que mientras que la cara es un espacio relativamente pequeño en el cual extraemos muchos puntos, en el cuerpo extraemos solo unos pocos para cubrir un área considerablemente mayor; ahora pensar en la relación de esos puntos, y lo mucho que nos pueden decir unos de otros para el caso de la cara, versus lo poco que se puede inferir para unas muestras reducidas en el cuerpo.

Todas estas soluciones están contempladas en (Body) *Pose Tracking* o “seguimiento de posturas corporales”, o fueron parte de lo que escaló a esta solución con la que se estiman los *landmarks* para las personas que se estén captando (reconociendo) por cámara (o menos comúnmente por imágenes estáticas). Ver Ilustración 85, para una referencia más gráfica de esta solución.

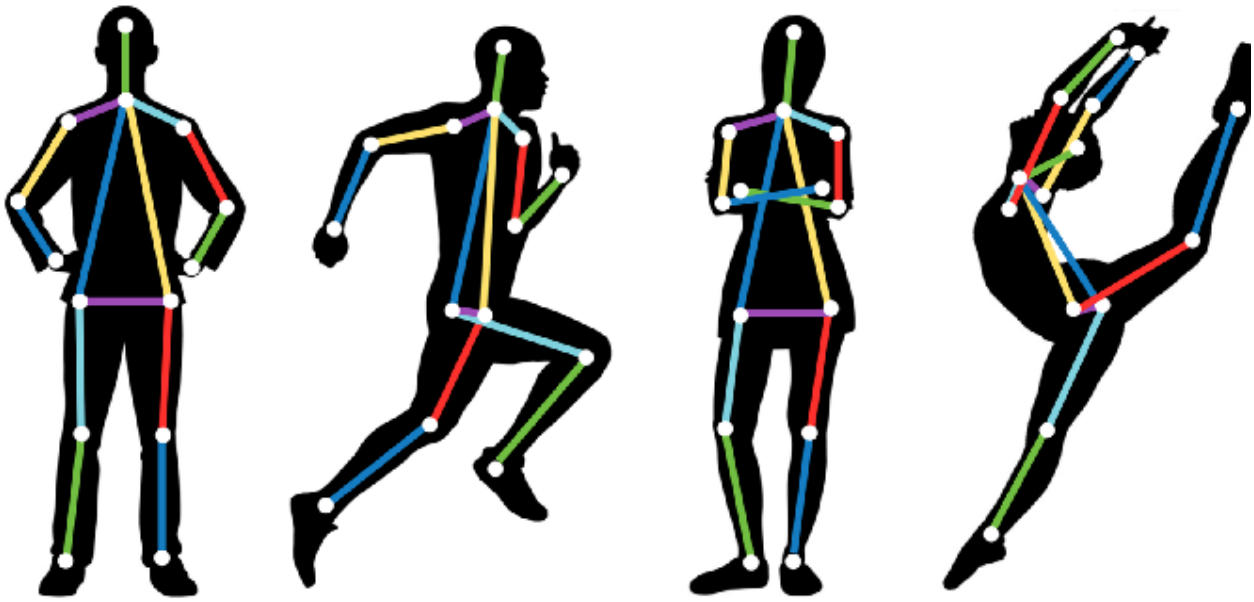


Ilustración 85 Representación del Pose tracking en ejecución. Fuente: obtenido de [42].

Action Detection

La detección de acciones es una solución muy completa que involucra topologías que dan soluciones a la mayoría de los problemas para los que se usa. En principio, son las soluciones de los componentes *Pose*, *Hands* y *Face* las que conforman a la *Action*

Detection; aunque puede considerarse como tal cualquier modelo, estructura o solución orientada a detectar acciones. No es de extrañar que no sea tan simple como haber "juntado" las tres (3) partes, por razones que van desde cómo se requieren las entradas hasta el solapamiento de puntos de referencia entre los componentes. Asimismo, el efecto que pueden llegar a tener al atender problemas es sinérgico (mayor al de la suma de sus partes), convirtiéndose con paso firme en el estado del arte para muchas tareas dentro de *Deep Learning* y *Machine Learning*.

Holística

Se focaliza la forma de interpretar y entender qué resultado se puede esperar y lograr con la *Action Detection*, mediante la "holística"; esta idea de ver un sistema desde la complejidad en términos de la relación entre sus partes. Evidentemente el "*pipeline*" o algoritmo del proceso de detección de acciones, define los procesos a los que se someterá a la *data* desde la entrada del sistema o modelo, hasta la salida (que sería presentar la acción detectada), sin embargo, los modelos aprenden a gestionar información de manera tal que llegan a apreciar las interacciones de, no solo los tres componentes implicados, sino todos los *landmarks* involucrados.

De allí, viene la idea de holística como un elemento que puede potenciar enormemente múltiples aplicaciones o dar pie a nuevos enfoques. De esta manera, se distinguen algunos usos destacables para entender su alcance.

- "*Developmental tracking*" ("seguimiento del desarrollo", de bebés e infantes)
- "*Performance Optimization and Injury Prevention*": Optimización del rendimiento y prevención de lesiones, principalmente para atletas de alto rendimiento, pero también para diferentes actividades deportivas y físicas.
- "*Fitness and Sport Analysis*": Análisis deportivo y de estado físico; centrado en resultados, evolución de las condiciones y de factores típicamente asociados a desarrollo físico, o a cambios físicos por hábitos nutricionales y de ejercicios ligados a vida saludable. Regímenes alimenticios, o rutinas de ejercicios aeróbicos (como caminatas o trotes, o ciclismo no competitivo) y anaeróbicos (como

levantamiento de pesas, abdominales o ciclismo con resistencia), son algunas de las actividades para las cuales se puede ameritar seguimiento de este tipo.

- "*Clinical Examination*": Examinación clínica, sobre todo en pacientes de tercera edad, o para aquellos con condiciones o enfermedades a los cuales se les pueda hacer seguimiento mediante aspectos visuales-posicionales.

En definitiva, son muchas las aplicaciones de la vida moderna que se engloban en esta creciente solución; algunas de las presentadas, tienen diversos desarrollos que responden al conjunto de soluciones para un componente en particular, bien sea *Pose*, *Hands* o *Face*.

Sign Language Recognition

El reconocimiento de lenguas de señas es un problema de la visión artificial abordado por algoritmos de aprendizaje automático, para los que se han destacado en desempeño y volumen de desarrollo las redes neuronales profundas. La intención final es ayudar a mejorar el estilo de vida de buena parte de las personas con discapacidad auditiva; principalmente con aplicaciones que traduzcan señas a texto o a voz, aunque, de hecho, se podría valuar cualquier propuesta tecnológica que involucre captar las señas para generar algo de valor con esto, más allá del facilitar la comunicación de las personas de esta comunidad con las personas que no sean capaces de comunicarse por lengua de señas.

Se sigue discutiendo esta tarea bajo nombres de otras parecidas, contenidas en esta o simplemente relacionadas; dando por resultado el que se refiera a SLR como detección de lenguas de señas o reconocimiento de gestos, como las más comunes. Ahora bien, resulta que bajo ciertos supuestos (como de qué tarea se habla exactamente) se podría no estar dentro del campo de visión artificial sino de, por ejemplo, robótica, otra disciplina académica de ciencias y tecnología dentro de la que se han hecho adelantos en SLR. Vale la pena hacer la distinción puesto que se plantean soluciones también dentro de este entendido, y toda área tiene sus propios retos y bondades.

Hasta el momento al reconocimiento y traducción de lenguas de señas se le ha buscado solución, o se ha procurado aportar con avances, en disciplinas como realidad virtual,

computación gráfica (modelado 3D), o procesamiento de lenguaje natural [43]. Pero el estado del arte posiciona al aprendizaje profundo como la disciplina más relevante para estos desarrollos y la proyección es que lo siga siendo por los próximos años conforme, en sí misma, crece el *machine learning* en general y el *deep learning*, a pasos agigantados. Se han presentado ya modelos de SLR cuyas predicciones tienen un alto nivel de precisión para señas estáticas, es decir, las posturas de las manos con algún significado por sí solos, sin precisar movimiento. Ejemplo de estas señas pueden ser los que aparecen en Ilustración 86: la “L” y la “S” en la Lengua de Señas Panameñas.

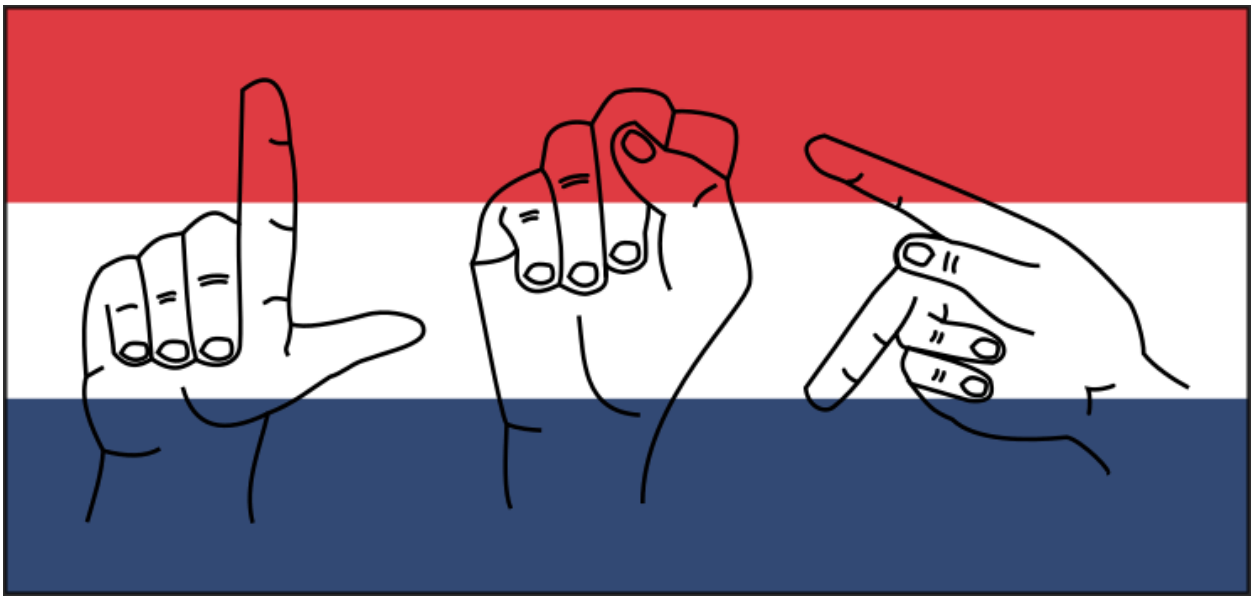


Ilustración 86 Señas de letras “L”, “S” y “P” (de izquierda a derecha) en LSP. Fuente: el autor.

Transfer Learning y modelos preentrenados

La transferencia de aprendizaje o conocimiento es una técnica con la cual se aprovecha un modelo preentrenado como base para el modelo que se está desarrollando. La razón de utilizar esta estrategia es principalmente para solventar uno de los problemas (o reto común) más grande dentro del Aprendizaje Profundo (que comparte con las demás subáreas de Aprendizaje Automático) que es la poca o insuficiente cantidad de datos.

Un modelo puede ser entrenado con un enorme número de datos, pero la capacidad computacional, o de procesamiento, puede tener que ser muy elevada para llevar a cabo este proceso. Por lo que existen estos modelos preentrenados cuyo uso es promovido por las comunidades o empresas que lo ponen a disposición de investigadores y desarrolladores particulares, o grupos de estos. Un modelo preentrenado dentro de visión artificial, preferentemente, es capaz de extraer el grueso de las características visuales de una imagen, por haber sido entrenado con millones de datos (en el contexto de visión artificial). Esto hace que usarlos de base a la hora de construir un modelo, sea beneficioso para el desarrollo, para el cual se tendrá que adecuar lo que se tiene, a las necesidades propias del proyecto. La Ilustración 87 indica el uso típico de esta técnica: aprovecharla para tareas similares.

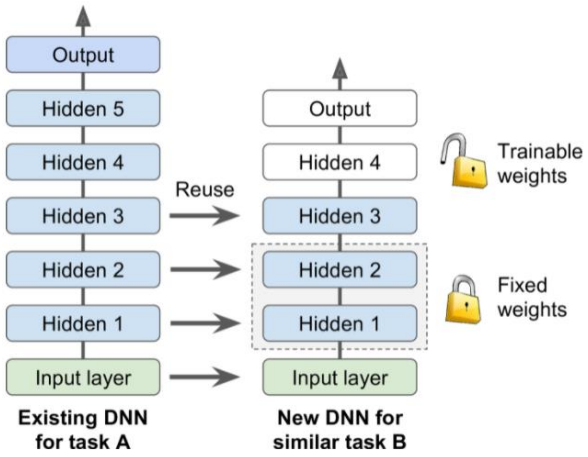


Ilustración 87 Transfer learning en el cual se utiliza parte de una red neuronal existente para desarrollo de otro. Fuente: obtenido de [25].

Hoy día es poco común que para desarrollos de modelos de *Machine Learning* no se utilice esta técnica. A diferencia del aprendizaje automático tradicional, en el cual los modelos desarrollados eran independientes, con *Transfer Learning* se pueden conectar gracias al conocimiento, aprovechando un modelo para los efectos de otro u otros. En aprendizaje profundo es una práctica sumamente común y es fácil suponer que seguirá siéndolo puesto que, pese a que cada vez las computadoras sean capaces de almacenar

y procesar más datos, la brecha entre el volumen de datos que gestiona una gran compañía con sus servidores respecto a lo que puede estar manejando una computadora personal o de instituciones como universidades, se sigue agigantando. Es de recordar, en este sentido, que los modelos preentrenados más usados suelen haberse puesto a disposición de la comunidad científica por parte de empresas; siendo mucho menos utilizados proporcionalmente, el significativo número de modelos que aportan integrantes de la comunidad como investigadores. La Ilustración 88 trata sobre cómo se utiliza *Transfer Learning* para un caso particular, basándose en un modelo con entrenamiento más general. Ver Ilustración 89, que considera al componente "knowledge" (conocimiento) como la diferencia en sí entre desarrollos en los que se usa esta técnica y aquellos tradicionales.

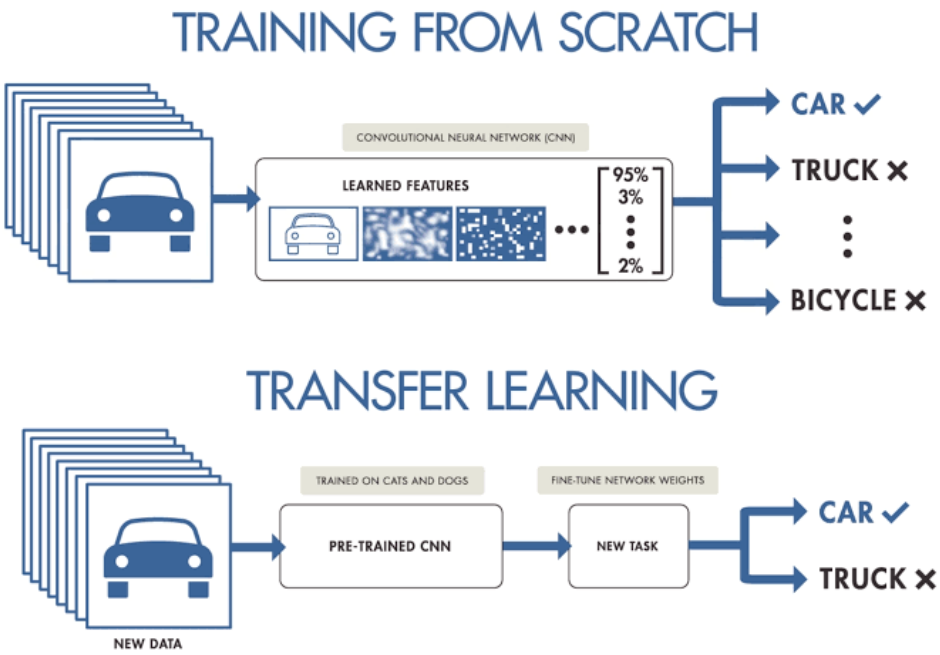


Ilustración 88 Transfer Learning vs entrenamiento tradicional (desde cero). Fuente: obtenido de [44].

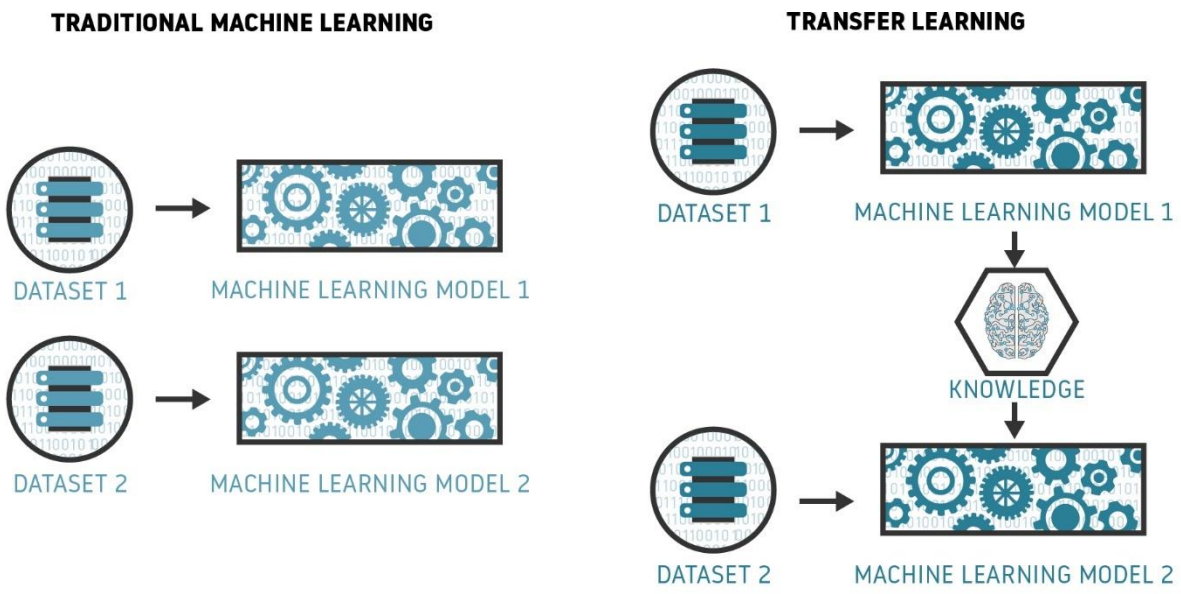


Ilustración 89 Transfer Learning vs Machine Learning tradicional. Fuente: obtenido de [45].

Tecnologías de desarrollo ML

El auge de las disciplinas de inteligencia artificial ha traído enormes avances para la ciencia y la sociedad. En particular, ha sido impresionante, aunque verosímil, en lo que se ha convertido el aprendizaje profundo; con tendencias que enuncian que seguirá incrementando su relevancia en el futuro próximo.

La investigación y aportes en el aprendizaje profundo no solo tienen forma de aplicaciones web o aplicaciones móviles, ni siquiera de sistemas comerciables. No habría sido posible que se llegase a este momento en el que está el campo de visión artificial sin las librerías (de “library”, que en realidad traduce biblioteca) destinadas al *Machine Learning* o a la *Computer Vision*, o las interfaces de programación de aplicaciones (*API*, por sus siglas en inglés) con las que se dispone para el diseño, construcción, entrenamientos y lanzamiento de modelos de aprendizaje profundo.

Dependencias y librerías

Python

Es el lenguaje de programación de alto nivel más importante de los últimos años. Es un lenguaje interpretado, de propósito general y alrededor del cual se han producido muchas

librerías. Es el lenguaje preferido por los desarrolladores e investigadores dentro de la ciencia de datos y para *machine learning*.

https://www.python.org/

TensorFlow

La librería predilecta a la hora de trabajar en proyectos de aprendizaje automático con enfoque en entrenamiento e inferencia de redes neuronales profundas; provee *APIs* estables para *Python* y se entiende con librerías que, de alguna forma, se han desarrollado pensando en la compatibilidad con esta.

https://www.tensorflow.org/

Keras

Esta librería provee a *Python* (a través de *Tensorflow*) de una interfaz para redes neuronales artificiales. Es una combinación común y potente junto a *TensorFlow*. Los aportes no se limitan a poder escribir código para desarrollos muy complejos con estas herramientas, sino a poder usar como base sus modelos preconstruidos de calidad comprobada y con alta capacidad. Ver Ilustración 90, sobre *Keras* y *TensorFlow* como capas de desarrollo.

https://keras.io/

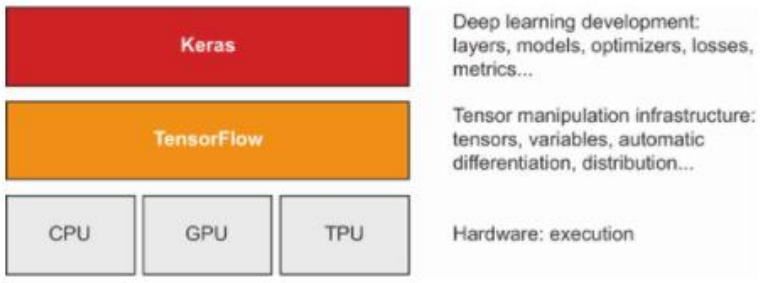


Ilustración 90 Keras y TensorFlow en conjunto. Fuente: obtenido de [24].

OpenCV

Aunque con alternativas muy completas, para algunos esta librería tiene estatus de ideal para el diseño y la construcción de modelos para tareas de visión artificial apoyados en otras librerías basadas en *Python* y altamente compatibles con *TensorFlow* y *Keras*.

https://opencv.org/

Numpy

Este ecosistema o módulo de software fue concebido para extender las capacidades de *Python* con computación numérica; teniendo implementaciones muy variadas que incluye participar en la detección de tiempo real.

https://numpy.org/

Pandas

Esta es una librería con operaciones y estructuras de datos, para análisis y manipulación de datos en formas de tablas numéricas o series temporales, con *Python*.

https://pandas.pydata.org/

Scikit-learn

Es una librería con la que se puede trabajar con algoritmos de aprendizaje automático en tareas de clasificación, regresión y "*clustering*" (agrupamiento o análisis de grupos).

https://scikit-learn.org/stable/

SciPy

Esta librería es comúnmente usada para procesamiento de imágenes; contiene además un módulo de optimización de funciones.

https://scipy.org/

os

Esta librería ayuda a trabajar mejor con las rutas de archivos y demás componentes que dependen del sistema operativo, de allí su nombre (abreviatura de "*operating system*").

https://docs.python.org/3/library/os.html

TensorBoard

Este kit de herramientas de *TensorFlow* permite la visualización de, entre otros elementos, métricas como la pérdida y la exactitud.

https://www.tensorflow.org/tensorboard

TensorFlow Object Detection

Esta interfaz de programación de aplicaciones (*API*) se utiliza en visión artificial para detectar, localizar y rastrear objetos. Se basa en topologías que permiten usar arquitecturas para detección de objetos, brindando además modelos preentrenados que pueden configurarse con facilidad. Se utilizará el modelo (de algoritmo para detección de objetos) ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8 para reconocimiento (de lenguas) de señas.

SSD

Mobilenetv2: Es una arquitectura con la idea "*Inverted Residuals and Linear Bottlenecks*" (residuos invertidos y cuellos de botella lineales) para aumentar el estado de arte de los modelos móviles (de allí "*mobile*", y "*net*" por red) en diversas tareas [46]. Es usado como *feature extractor* para *OD* con *SSD*. Conceptualmente, lo que propone es el uso de una serie de bloques de capas con varios filtros convolucionales, a sabiendas que en muchas ocasiones no es fácil o incluso posible determinar cuáles son las mejores transformaciones para aplicar a una entrada visual, o datos que estén siendo procesados a través de una entrada de este tipo.

SSDLite: Marco de trabajo que describe una forma eficiente de aplicar *MobileNetV2* [46]. Ver Ilustración 91, con un ejemplo de estructura de un bloque de convolución.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Ilustración 91 Estructura detallada de un bloque de convolución de profundidad separable y tipo cuello de botella. Fuente: obtenido de [46].

COCO 17: Es la versión 2017 del *dataset* de gran escala para detección de objetos, segmentación y *captioning* (subtítulos). Cuenta con imágenes, cuadros delimitadores y etiquetas, y es parte del catálogo de *TensorFlow* [47].

Feature Pyramid Network: Subred que da por salida *feature maps* de diferentes resoluciones. (toma una *single-scale image* de un tamaño arbitrario como entrada, y salidas de mapa de características proporcionalmente a múltiples niveles.

MediaPipe

MediaPipe es un proyecto de código abierto de Google que presenta soluciones multimodales y multiplataforma de *Machine Learning* aplicado. Estas soluciones son configurables e implementables, e incluyen propuestas tan generales como detección de objetos, o tan específicas como “Iris” (seguimiento del iris del ojo) que puede ser aplicado en fotografía computacional (reflejo de destello).

https://mediapipe.dev/

MediaPipe Pose

Esta solución de Aprendizaje Automático permite inferir la postura del cuerpo humano y hacerles seguimiento a los 33 valores posicionales (3D) que lo definen, correspondientes a los puntos de referencia empleados, con alto nivel de precisión en ejecución [48]. Actualmente tiene aplicaciones como las de conteo de repeticiones, y series, de

ejercicios físicos (*Pose Classification*) como flexiones de pecho o sentadillas, o es base en aplicaciones de baile o danza para corregir posturas y movimientos, o para calificar rutinas; también se encuentra en desarrollos que tienen relación con realidad aumentada como la superposición de objetos digitales según las acciones físicas.

La topología *COCO* [49] es el estándar para detección de personas y localización de sus *keypoints* (puntos clave), habiendo impulsado el estado del arte usado también en tareas de "*Stuff Segmentation*" o segmentación de cosas, donde "cosas" tiene más relación con materia (agua, cielo) que con objetos como en el caso de detección de objetos que se dirige a clases de "*things*" (cosas; mas no "*stuff*") como aparatos, carros o incluso personas. Se lo encuentra también en "*Panoptic Segmentation*", es decir, segmentación panóptica; la cual hace reconocimiento en simultáneo tanto de las clases "*things*" (segmentación semántica) como de la clase "*stuff*". Ver Ilustración 92, sobre segmentación semántica.

Ilustración 92 A la izquierda, una imagen de entrada; a la derecha, los efectos de aplicar sobre ella segmentación semántica. Fuente: obtenida de [24].

Por su parte, *MP Pose* es una propuesta para percepción de postura corporal que se basa en *BlazePose* [49] que considera más puntos que *COCO*; a saber, la totalidad de los puntos en Ilustración 93, de quien es superconjunto al igual que de las topologías "*BlazeFace*" y "*BlazePalm*". Se implementa el seguimiento con un módulo destinado para tal fin, y es posible predecir una máscara de segmentación, que no es más que una segmentación constituida por las clases "*human*" (humano) y "*background*" (fondo). Ver Ilustración 93, con los puntos que calcula *MediaPipe Pose*.

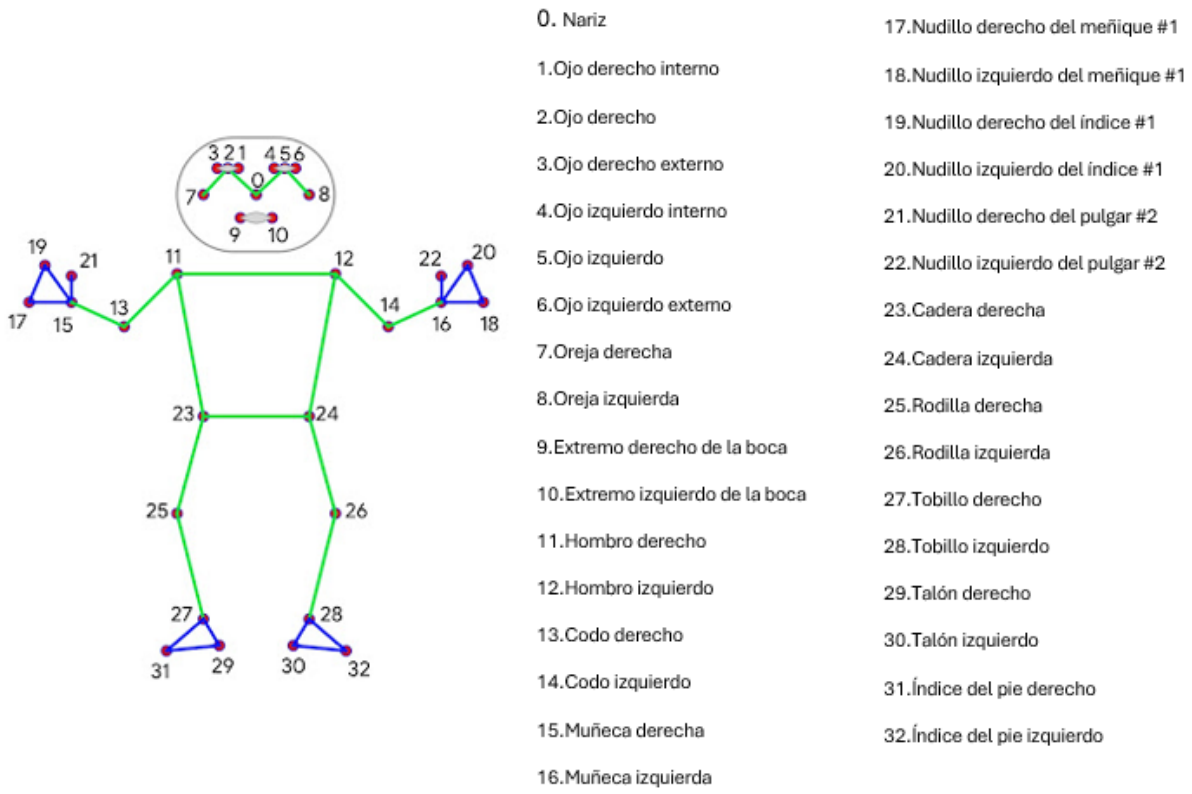


Ilustración 93 Landmarks del componente pose. Fuente: adaptado de [48].

MediaPipe Face Mesh

Un componente para el cual se amerita la lectura continua de cientos de datos para una cantidad importante de tareas es la cara y *MP Face Mesh* es una solución que estudia la geometría de la cara estimando 468 puntos de referencia (3D) de la cara por inferencia de la geometría de la superficie gracias al empleo de *Machine Learning*, sin la necesidad de ningún sensor de profundidad [50].

El modelo de redes neuronales del que se basa fue concebido para el desarrollo de aplicaciones útiles de realidad aumentada, y para conseguir esto, se dio un salto de calidad con relación a la estimación de *landmarks* faciales del momento, con el módulo "*Face Geometry*" (geometría facial) que aprovecha los puntos de referencia identificados y tras establecer un espacio tridimensional métrico (espacio coordenado desde la perspectiva de una cámara virtual, dentro del módulo) estima las geometrías del componente.

Entonces, se tiene que esta solución cuenta con un primer modelo, el de detección, que calcula la ubicación de los puntos de interés; y un segundo modelo, que determina las geometrías mediante regresión. Esta labor conjunta permite a la red redirigir esfuerzos a la precisión de las predicciones, aprovechando mejor su capacidad, en lugar de invertir en procesar transformaciones para "*data augmentation*" (aumento de la cantidad de datos al usar instancias con modificaciones leves); esto por cómo se determina el área de la cara con un tiempo significativamente menor, alcanzando una inferencia de tiempo real muy ventajoso [50].

El detector facial *BlazeFace*, del cual se vale esta solución, tiene capacidad de inferencia por *GPU* (unidad de procesamiento gráfico) [48], tiene un esquema de anclaje basado en *SSD*, y consta con una red de extracción de características inspirada en *MobileNetV1/V2*, las dos versiones de la familia de redes neuronales multipropósito enfocadas en visión artificial [51]. Ver Ilustración 94, con los puntos que calcula *MediaPipe Face Mesh*.

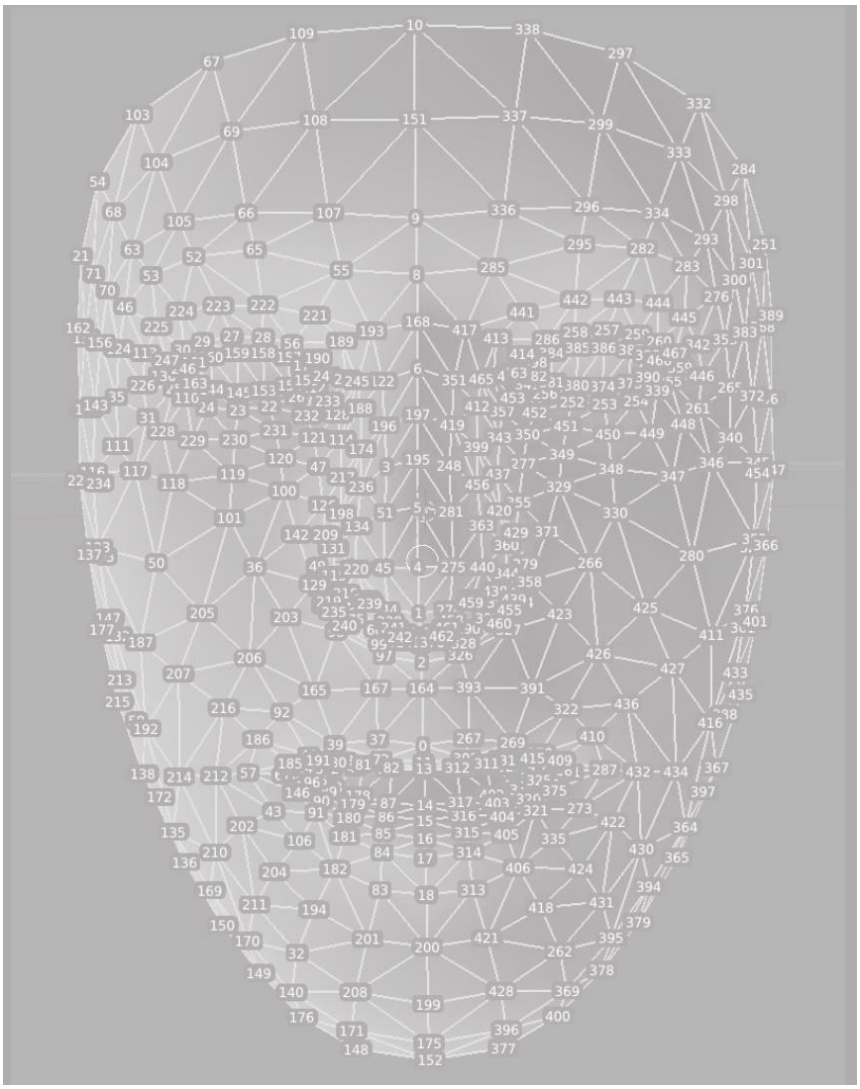


Ilustración 94 Landmarks del componente "cara". Fuente: adaptado de [51].

MediaPipe Hands

La base de aplicaciones de varios dominios puede ser la distinción de las manos como componentes, y de su movimiento a través del tiempo. *MP Hands*, dispone de dos modelos para llevar a cabo estas tareas, haciendo inferencia y seguimiento de 21 *landmarks* por mano, desde el punto que representa la muñeca hasta las puntas de los dedos, pasando por las articulaciones. Esta propuesta es particularmente relevante dentro del conjunto de soluciones de su tipo, pues recuerda a los guantes con los que se podían hacer aplicaciones de *Hand Tracking* por medio de múltiples sensores. Se puede

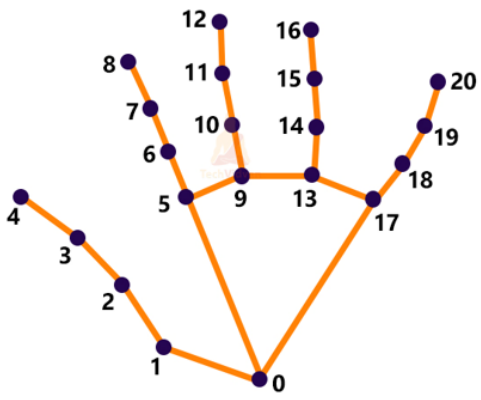
decir que la idea es similar, puesto que se basan en valores posicionales; la diferencia recae en que mientras para los guantes se usaba robótica como el centro del desarrollo, esta solución se centra en *Machine Learning* (principalmente *Deep Learning*) junto a visión artificial (para estudiar datos visuales); evidentemente, esto no los hace mutuamente excluyentes, ambos siguen siendo desarrollos de Inteligencia Artificial para los que cada vez hay más posibilidades de dar con modelos o sistemas híbridos.

El primero de los módulos involucrados es el de detección, que es un modelo *SSD* que lidia con los retos de este proceso para un componente para el cual las características visuales no son suficientemente diferenciables como, por ejemplo, para los de la cara, que aparte de la diferencia entre las partes como ojos, boca y nariz, se han implementado lógicas que consideran la distribución (posición relativa en la cara). El hecho de que se puedan superponer los puntos de la mano con tanta facilidad también complica este procedimiento, puesto que el modelo debe poder entender que una mano está cerrada o abierta (parcial o totalmente), y cómo exactamente, puesto que se puede necesitar interpretar un gesto como en el reconocimiento de señas, para el cual es vital que la detección sea acertada y, más allá de esto, que se haga en buen tiempo.

Usualmente se habla de detectores de manos cuando se amerita inferir qué gestos se están haciendo u otras tareas relacionadas; sin embargo, *MediaPipe Hands* se apoya de un modelo de detección de palmas (*Palm Detection Model*) porque, en esencia, se está haciendo estimación para un elemento más simple que sería o una palma o un puño, en lugar de los cientos de posibilidades que se tienen en términos de formaciones que involucren los dedos. Adicionalmente, estrategias como el uso de un extractor de características son usadas para tener una alta precisión.

Por su lado, el segundo módulo, es el que determina los *keypoints* de las manos, con rendimiento consistente aun en casos para los que las manos no se vean del todo, o bien estén en posiciones poco comunes; habiendo sido entrenado con datos sintéticos de heterogeneidad que incluye variedad de fondos [52] El mapeo de los valores correspondientes a las coordenadas espaciales puede darse por supervisión basados en el mapa de profundidad de imagen, o estableciendo los mismos valores para cuando se amplíe el conjunto de datos de entrenamiento con los fondos sintéticos.

Esta solución es configurable para adaptar a las necesidades del caso particular. Puede activársele un modo de imagen estática cuando se quiera que la detección de manos se ejecute para cada imagen; también se puede limitar el número máximo de manos (predefinido en dos) y establecer valores de "*confidence*" (confianza) como de detección o de seguimiento, el cual podría incrementar la confiabilidad del sistema en detrimento de su rapidez de ejecución. Ver Ilustración 95, con los puntos que calcula *MediaPipe Hands*.



0.Muñeca
1.Articulación carpometacarpiana del pulgar
2.Articulación metacarpofalángica del pulgar
3.Articulación interfalángica del pulgar
4.Punta del pulgar
5.Articulación metacarpofalángica del índice
6.Articulación interfalángica proximal del índice
7.Articulación interfalángica distal del índice
8.Punta del índice
9.Articulación metacarpofalángica del dedo medio
10.Articulación interfalángica proximal del dedo medio
11.Articulación interfalángica distal del dedo medio
12.Punta del dedo medio
13.Articulación metacarpofalángica del anular
14.Articulación interfalángica proximal del anular
15.Articulación interfalángica distal del anular
16.Punta del anular
17.Articulación metacarpofalángica del meñique
18.Articulación interfalángica proximal del meñique
19.Articulación interfalángica distal del menique
20.Punta del menique

Ilustración 95 Landmarks del componente mano. Fuente: adaptado de [50].

Mediapipe Holistic

Esta solución es, entonces, el resultado de aprovechar las ya poderosas soluciones *MediaPipe Pose*, *MediaPipe Face Mesh* y *MediaPipe Hands*, ensamblándolas de una manera astuta y apoyándose en las ideas detrás de las tres topologías, así como de la holística como concepto. Pese a ya haber sido capaces individualmente de dar solución, y ser usados en, problemas, tareas y aplicaciones varias de visión artificial, esta combinación ha provisto a este campo de una herramienta completa y altamente competitiva de *Deep Learning*: la detección de acciones.

Capítulo III – Marco Metodológico

Definición de la metodología

Exploración inicial

Primeramente, se debe hacer revisión y recopilación de fuentes como antecedentes, así como documentación relacionada al área de estudio y de la problemática, además de otras consideraciones teóricas a modo de marco teórico; que aporten en forma de cimientos para establecer ruta a seguir con claridad en múltiples elementos. Lo anterior incluye investigación sobre LSP, sobre redes neuronales convolucionales y redes neuronales recurrentes, y sobre reconocimiento de lenguas de señas, distintas a la nacional, con enfoques variados.

Se dispone de la técnica cualitativa de investigación bibliográfica para tal fin, leyendo artículos académicos que abarquen los temas expuestos con miras a sustentar la investigación actual de forma teórica.

Definición de algoritmos y modelos

Teniendo en consideración que se está trabajando con redes neuronales profundas, la investigación recae en el aprendizaje profundo. En particular se trabajará con redes neuronales convolucionales y recurrentes, habida cuenta de tener que procesar datos de tipo imagen (por lo que se centra el estudio en visión artificial) y secuencial, para los cuales son destinadas estas estructuras, respectivamente.

La investigación responde a un tipo de método deductivo ya que a partir de entradas visuales (imágenes) se hace una inferencia a través del modelo desarrollado, por cuanto se requiere la identificación de señas a ser traducidas. Como las señas se identifican según las etiquetas preestablecidas la “traducción” se consigue con el solo hecho de presentar el texto (el nombre de la clase o etiqueta, en este caso una seña) por una salida visual.

Recolección de datos

Se pretende utilizar una técnica de recolección automatizada de imágenes por cámara web; con la cual se haga captura, una a una, de las fotos o videos que serán el *dataset*, almacenándolas conforme se capta cada seña. Para este propósito se concebirá un bloque de código empleando programación con *Python*, con apoyo en la herramienta de

visión artificial *OpenCV*. La cantidad de datos a recolectar va directamente relacionada con el número de clases a considerar según el experimento, y responde a la heurística estadística, como estrategia para decisión simplificada respecto al tamaño de muestra. El método escogido es el de factor del número de clases, el cual propone una x cantidad de elementos por cada clase. Esta *x* deberá ser una potencia de 10; en este caso será $10^2 = 100$.

Construcción de modelo base

El prototipado tiene por finalidad, inicialmente, el ampliar la noción del funcionamiento de las estructuras convolucionales para la tarea de reconocimiento de señas; eventualmente, podrá ser utilizado como módulo del sistema un modelo que parta de este y que se especialice en la traducción de las señas estáticas. Por estas razones, atañe valerse de la técnica de transferencia de aprendizaje (*Transfer Learning*) partiendo la construcción del modelo de reconocimiento de señas, de un detector de objetos. Este será un modelo preentrenado del *TensorFlow Model Zoo*: *TensorFlow Object Detection*.

Desarrollo de modelo de inferencia

Para este fin se dispone del método científico del empirismo por medio de implementación del marco de trabajo Scrum, reemplazando con esto un enfoque algorítmico programado con uno heurístico. Es decir, se opta por hacer iteraciones cortas durante el ciclo de desarrollo; generando con esto entregables de valor en los que se vayan implementando avances en pro del objetivo, en lugar de proceder con un enfoque tradicional como lo es el modelo/desarrollo en cascada. Esto permite que, en detrimento de las virtudes de trabajar con el ciclo de vida de un programa, se imprima dinamismo en el proceso.

Análisis de resultados

Se cumplirá esta etapa valiéndose de un método de análisis cuantitativo a través de una herramienta común en aprendizaje automático como lo es la matriz de confusión y haciendo lectura de la precisión del modelo de inferencia. Se busca con esto validar el rendimiento del modelo desarrollado para confirmar que los resultados son confiables, y en ejecución lo serán.

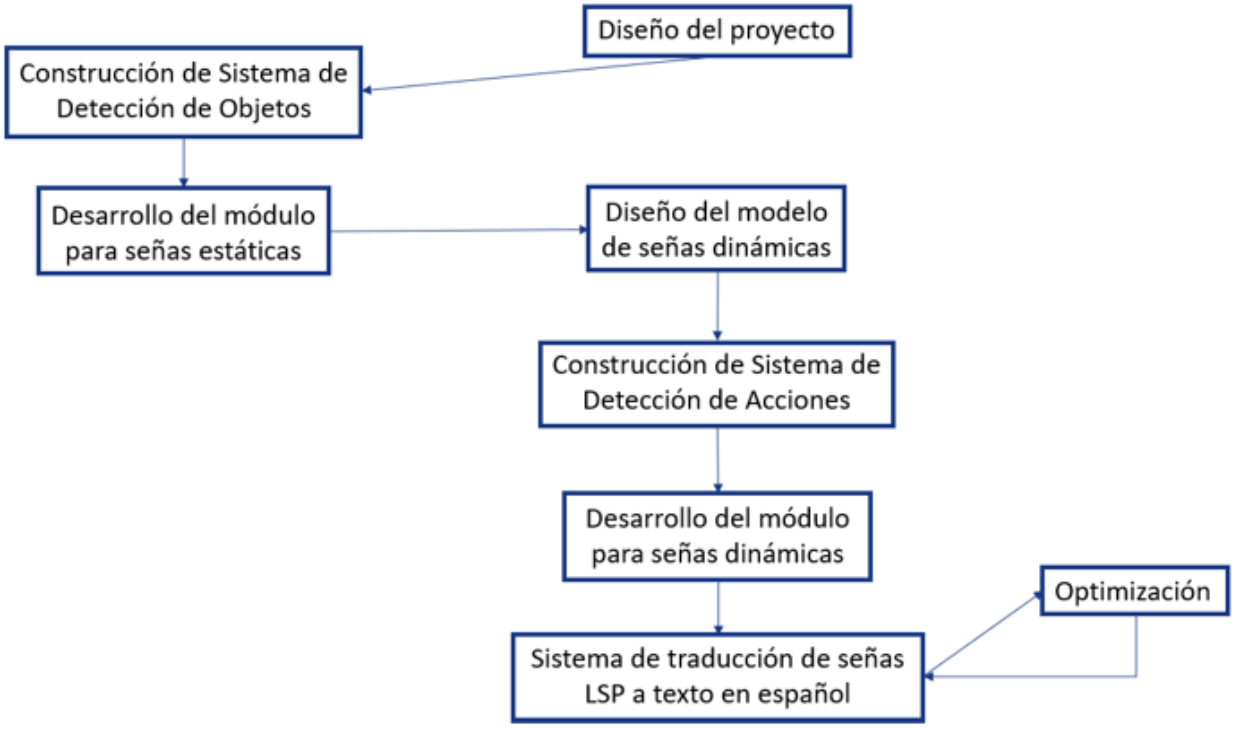


Ilustración 96 Diagrama sobre la metodología. Fuente: El autor.

Diseño del proyecto

Los elementos en los cuales se basa el sistema giran en torno a *Python* y un conjunto de librerías, aplicaciones y soluciones que hicieron parte del proceso de desarrollo, y componen al sistema como producto final.

Planificación de ideas

Sustentada en la bibliografía y en las hipótesis a partir de la documentación y el proceso de generación de ideas, se establece una ruta sobre las acciones a hacer, en forma de elementos a considerar, pruebas, y cuanto se haya determinado como útil para el desarrollo del proyecto.

Destaca en importancia el uso de redes neuronales recurrentes en un problema de visión artificial (reconocimiento de señas) como propuesta para impulsar el estado del arte.

Recolección de imágenes

Dada la naturaleza iterativa del estudio y desarrollo se plantea hacer la recolección de los datos en tres etapas:

- Primeras pruebas: se escribe el código con el cual se llevará a cabo la recolección de imágenes correspondientes a las señas estáticas, y se ajustará para recolección de los videos de las señas dinámicas. Se corre el código en ambos casos para cerciorar que la tarea será completada en el resto del proceso.
- Prototipo: se recolectan las imágenes de las señas de ambos tipos para desarrollar los modelos base. Se ajusta el código, mejorando la lógica detrás de la tarea; al tiempo que se mejora el proceso de recolección para los videos, puesto que no presentan mucha flexibilidad por propensión a ser capturas con ruido (patrones que no se quieren identificar).
- Modelos: se recolectan finalmente las imágenes y videos de las señas con las que se entrenará cada uno de los modelos. Se hace revisión de la calidad de las capturas para el modelo de redes neuronales convolucionales durante el proceso de etiquetado (exclusivo de este modelo, en forma). Se cumple con comenzar la captura de cada seña desde la posición inicial de la seña y no desde una posición neutra debido a que tener en común el comienzo (o final) de la seña en los videos entorpecerá el proceso de entrenamiento. Es importante entender que la red neuronal tras el modelo de traducción de señas dinámicas está haciendo seguimiento de todos los valores estudiados desde el momento inicial y hasta el final de cada captura, por lo que cada posición a lo largo del tiempo es determinante para el aprendizaje del modelo y su futuro buen desempeño esperado.

Preprocesamiento de datos

Este paso preliminar difiere entre los modelos desarrollados, se detalla el proceso individual de cada modelo como sigue:

- Para el modelo de traducción de señas estáticas se precisa un preprocesado de las imágenes que consiste en ubicar la región de interés (ROI) en cada una de estas. La ROI será el área que abarca la mano (o manos) que aparecen en cada

captura, y para la cual se establece una etiqueta (nombre de la clase a la que pertenece).

- En cuanto al modelo de traducción de señas dinámicas, es necesario para los videos capturados cambiar el modo de color a escala de grises para ahorrar recursos; esto se logra con código destinado a hacer la conversión para cada cuadro, es decir, toda imagen del grupo de las que se compone un video, y para la totalidad de los videos.

Mapa de etiquetado

Es una consecuencia directa del proceso de etiquetado para el modelo de traducción de señas estáticas, en el cual se da la extracción de las etiquetas generándose un archivo .xml que también contiene valores que definen el ROI para la imagen. El instrumento que da pie a este paso fundamental es *LabelImg*.

Por su lado, para el modelo de señas dinámicas se capturan las etiquetas, como parte del label map, a lo largo de la colección por cámara de las secuencias (videos).

Desarrollo de modelos de prueba

Esta etapa intermedia se subdivide en los modelos que siguen:

- De señas estáticas: incluye el diseño y entrenamiento del detector de señas adapta del detector de objetos de *TensorFlow*.
- De señas dinámicas: involucra todas las etapas con las que se dará con el modelo final, con algoritmos, valores y datos iniciales que se irán alterando, descartando o sustituyendo.

Se harán pruebas de los recursos, algoritmos y modelos con una versión simplificada de lo que sería el sistema; y conjuntos de datos similares a los que se utilizarán más adelante. Seguidamente, se dispondrá de sesiones para la recopilación de los datos en forma de imágenes y videos.

El propio desarrollo partirá con el procesamiento (aplicar filtros, etiquetar elementos, etc.) de las imágenes y videos de las señas (tanto estáticas como dinámicas) que se hayan decidido usar (saludos y respuestas cortas) para prepararlas para lo que sería el entrenamiento de los modelos (preentrenados para detección de objetos) y las

pruebas inmediatamente después. Se implementará la lógica (con visión artificial) necesaria para que el sistema tenga por característica trabajar en tiempo real.

Pruebas de prototipo

El modelo base será sometido a pruebas con las que se evaluará su rendimiento por precisión, los tiempos de entrenamientos, y puntos clave para mejorar los modelos en los distintos aspectos posibles.

Se harán los ajustes necesarios para mejorar la precisión de reconocimiento de señas, lo cual impactará proporcionalmente en la precisión de traducción (LSP a español) y en el desempeño del sistema como un todo.

Desarrollo de modelo final

Tras las correcciones y acciones pertinentes, se completa el desarrollo del sistema, con un modelo que tenga los algoritmos definidos, las clases decididas y el número de imágenes establecidas en respuesta a lo estudiado y probado en etapas previas.

Una vez desarrollado el sistema de traducción de señas de la Lengua de Señas Panameñas a texto en español, se optimizará el código para mejora integral del sistema.

Validación

Habiéndose monitoreado el progreso de entrenamiento y la precisión por cada ciclo concretado, se definen las métricas de evaluación del modelo para evaluar el rendimiento del sistema y lograr la validación de este.

Diseño de la interfaz

Finalmente se logra poner a disposición el sistema para traducir señas de la LSP a español.

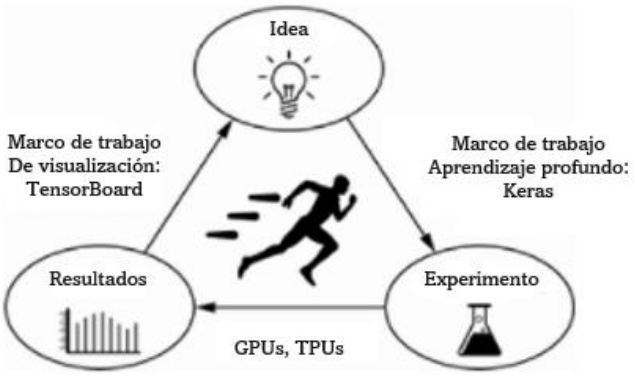


Ilustración 97 Marcos de trabajo y recursos principales en un proyecto DL. Fuente: adaptado de [24].

Desarrollo del proyecto

Desarrollo del módulo para señas estáticas

Primeramente, se instalan las dependencias necesarias, como *TensorFlow*. Para la construcción del detector de objetos con *TensorFlow* es necesario instalar el *API* de detección de objetos de *TensorFlow*. Se clona el repositorio para aprovechar el enfoque *Transfer Learning*, en vista de que se tendrá código base para la construcción de un sistema de detección de objetos para el cual habrá habido preprocesamiento y del que se tendrá los cimientos para los cambios y el entrenamiento que permitirán adaptarlo a un sistema de reconocimiento de señas.

Configuración de rutas

En primer lugar, se organiza el espacio de trabajo (archivos de entrenamiento y demás), y se configuran las rutas.

La ruta del espacio de trabajo (WORKSPACE_PATH) será aquella del directorio '*workspace'* dentro del ya creado '*Tensorflow*'. Adentro se crean los directorios para las imágenes a capturar para entrenamiento y prueba, los modelos preentrenados (donde estará en particular el de detección de objetos), los modelos a construir (tendrá dentro la de configuraciones y la de control), y el de anotaciones.

En '*Tensorflow*' adicional a la carpeta del espacio de trabajo, se tendrá una para "*scripts*" (los comandos a ejecutar como programas) y una para el modelo *API*.

Creación de mapa de etiquetado

Se amerita tener establecidas las señas a considerar, para lo cual se crea un mapa de etiquetado; se usará tanto en el proceso de entrenamiento como el de detección.

El mapa de etiquetado se almacena en un espacio de memoria llamado "*labels*". Consiste en una lista de diccionarios, en los cuales se definen la etiqueta por su nombre y se le establece un identificador.

Se guarda en un documento .pbtxt con una estructura repetitiva. Este archivo estará en el directorio de anotaciones.

Recolección de imágenes

A través de un *script* para recolección de imágenes con *OpenCV*, se hacen las capturas (fotos) de las señas con la cámara integrada. Se toman 100 capturas (fotos por cámara web) para los procesos de entrenamiento y prueba, almacenándolas en sus respectivas carpetas.

Etiquetado de imágenes

Se instala *LabelImg* (mediante PIP) para el etiquetado de las fotos capturadas; se ubica en el directorio '*Tensorflow*'. Se configura *LabelImg* con los directorios, primero de entrenamiento, y posteriormente de prueba, para etiquetar en orden la serie de señas (se puede particionar luego, en su lugar). Se tendrá para cada foto, entonces, un documento de imagen y un .xml asociado generado de este proceso, con el nombre (etiqueta) según la seña de la foto (ver la Ilustración 98); este archivo también tiene información como el tamaño de la foto (640 de ancho x 480 de altura) y los valores límites del cuadro que delimita la seña dentro de la imagen.

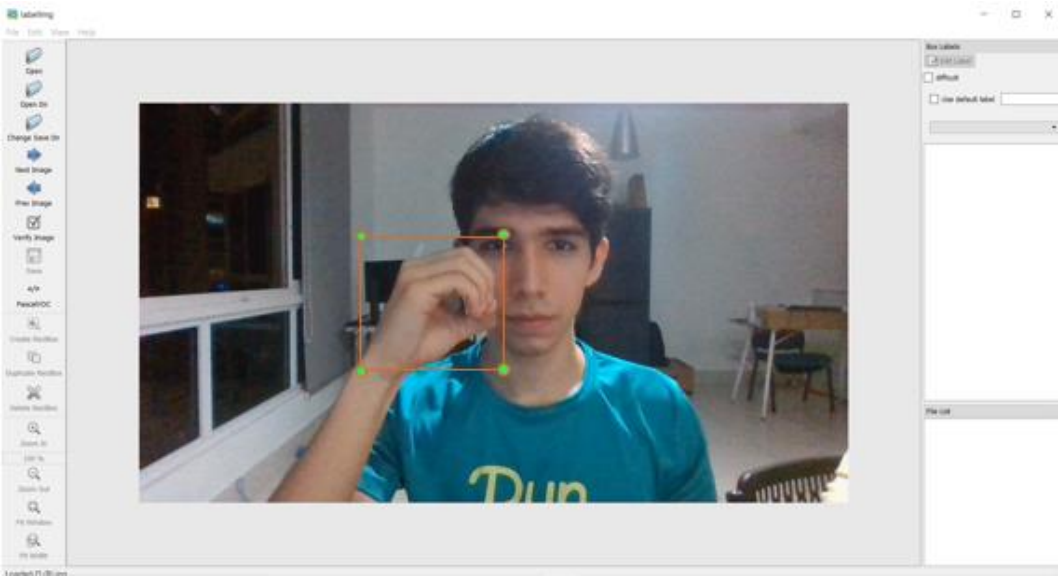


Ilustración 98 Etiquetado de una seña con LabelImg. Fuente: el autor.

Creación de TF records

En el mismo directorio de anotaciones se deben guardar los *TF records* ("*TF*" de *TensorFlow*) que son los archivos cuyo formato permitirá almacenar una secuencia de registros binarios que entienda *TensorFlow*; esencialmente se convierten las anotaciones en los registros.

Se necesita un archivo de estos para entrenamiento (*train.record*) y otro para prueba (*test.record*).

Descarga de modelo TensorFlow preentrenado

Se descargan de *GitHub* los modelos preentrenados del *TensorFlow model Zoo* (donde están las colecciones de modelos disponibles), clonando el repositorio "*models*".

Copiado de model config al directorio de entrenamiento

Se establece un nombre para gestionar como ruta. Se crea el directorio "my_ssd_mobnet"

El pipeline.config es el archivo donde están las configuraciones del pipeline (flujo del trabajo del modelo), por lo que se tendrá inicialmente la configuración del modelo preentrenado.

Actualizar Config para Transfer Learning

Se debe actualizar el archivo de configuración para poder realizar el proceso de transferencia de aprendizaje.

Se configura el número de clases 3-5 (según el experimento) y el tamaño de muestra (qué tanta *data* se procesa por época).

Entrenamiento del modelo

Se entrena al modelo para que “aprenda” a asignar a cada seña considerada su etiqueta. Con la siguiente celda se obtiene el *script* para entrenar al modelo.

Por defecto, aparece un registro de la pérdida cada 100 pasos (épocas).

Carga del modelo entrenado desde el Checkpoint

Se cargan los pesos ponderados desde el punto de control, para poder utilizar lo que ha aprendido el modelo (los modelos calculados tras el entrenamiento).

Construcción de sistema de detección de objetos

Mientras el modelo preentrenado era un detector de objetos, el entrenarlo con imágenes con las señas y sus etiquetas, de acuerdo con las clases definidas, lo convierte en un nuevo modelo que es capaz de hacer reconocimiento y traducción de señas a texto, a lo que se le denomina “sign language detector” (detector de lengua de señas); en este caso, un modelo de aprendizaje profundo basado en redes neuronales convolucionales. Ver la Ilustración 99, donde se presentan salidas del modelo de detección de objetos adaptado para reconocimiento de señas.

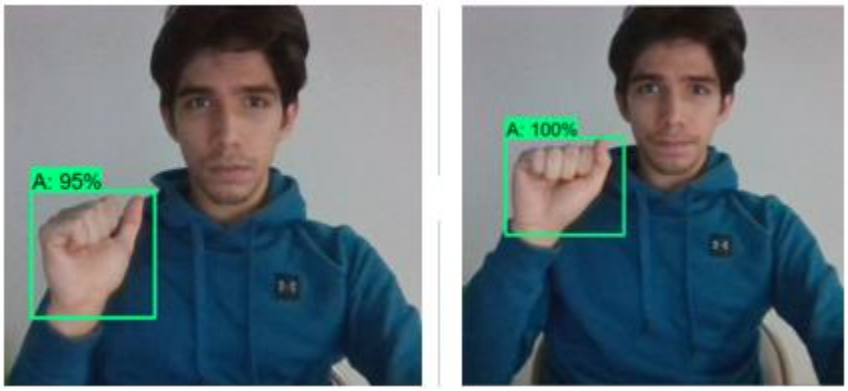


Ilustración 99 Modelo en ejecución con salidas con un 95% de confiabilidad (izquierda; seña con una rotación y separación del pulgar) y un 100% de confiabilidad (postura típica) para la letra "A" en LSP. Fuente: el autor.

Desarrollo del módulo para señas dinámicas

La propuesta de más peso en el presente estudio es aprovechar las *RNN*, no como alternativa, pues no se especializan en imágenes, sino como componente sustancial en el desarrollo de un sistema de traducción de señas.

Instalación de librerías y dependencias

Son importadas las siguientes librerías:

- OpenCV (cv2): para visión artificial en tiempo real. Usada para acceder a la cámara integrada y extraer los *keypoints*, también para procesamiento de imágenes.
- Mediapipe (mp): para extraer los puntos (*keypoints*) holísticos de cara, manos y cuerpo; primeramente, para entrenamiento, posteriormente en ejecución para las predicciones.
- Sklearn: se usa para convocar las métricas de evaluación, para la separación de los datos en los conjuntos de *training* y *testing*, y en etapas para las que se presenta el código.
- Matplotlib (mp): para la visualización de las imágenes; *pyplot* (*plt*) para visualización (elementos de interfaz).

Detección de los keypoints/landmarks usando Mediapipe Holistic

Configurando Mediapipe Holistic

Se configura la visualización de los puntos de referencia y sus conexiones, es decir, son cambios en los parámetros de formato. Los componentes “Pose”, “Face” y “Hands”, son parte de la solución propuesta y sus topologías trabajan sinérgicamente siendo procesadas como se presenta en la Ilustración 100.

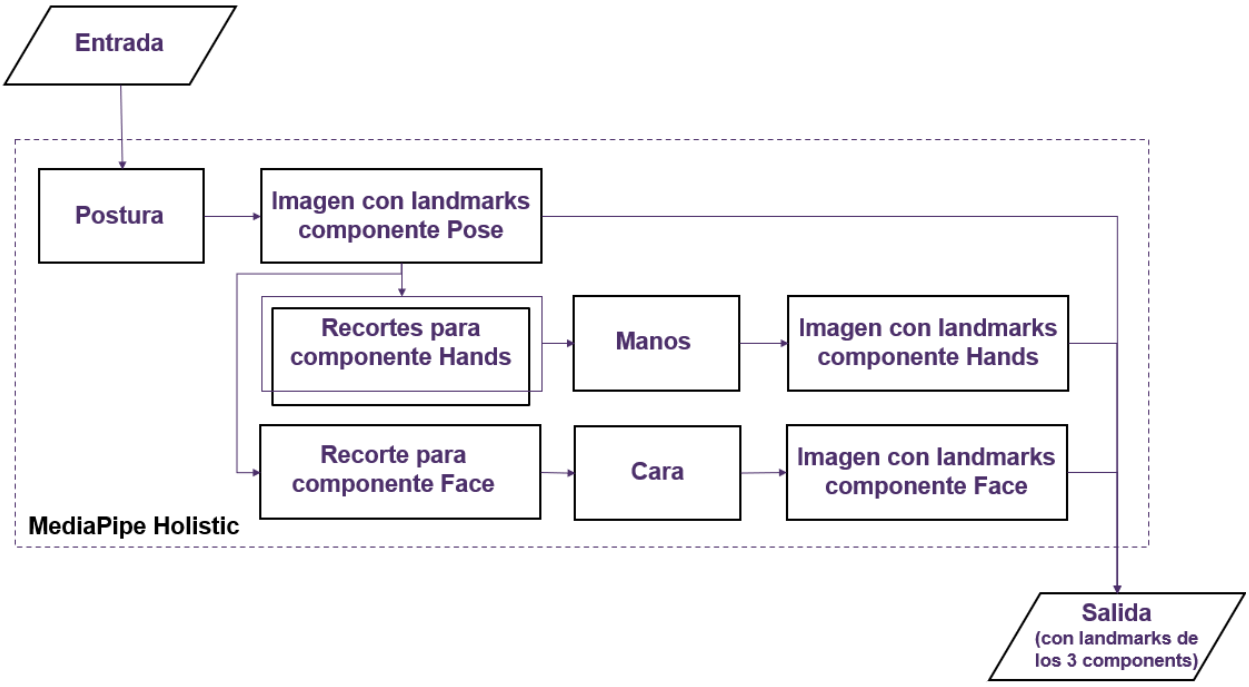


Ilustración 100 Procesamiento de la solución según las topologías involucradas. Fuente: el autor.

Se establecen colores que permitan diferenciar fácilmente los elementos, y que sean compatibles con buen número de fondos, el grosor de las líneas y el tamaño de los puntos responde a la conveniencia de presentar de esa manera cada componente. Es de acotar, que a las manos se les configuraron colores diferentes para identificarlas con mayor facilidad en los procesos de entrenamiento, pruebas y ejecución, dado que en ocasiones se trata con visualizaciones normales y en otras, en modo espejo.

Capturando imágenes por cámara

Se crea un elemento de *OpenCV* de tipo video captura, al cual se le envía el valor “1” que corresponde a la cámara integrada de la computadora, alternativamente ha de pasársele el valor “0” para hacer captura con el smartphone (puede variar dependiendo de sistema operativo y otros factores).

Los valores de confianza mínimos para detección y seguimiento son de 0.5 (50%), se consideran estos valores en este paso, puesto que serán utilizados más tarde en ejecución.

Se tiene una estructura repetitiva con la que se concatenan las iteraciones en un arreglo *Numpy* que tendrá solo valores “0” si no tenemos valores (de entrada) en algún punto. Se lleva a cabo la captura, la detección, el trazado de los puntos y líneas de los *landmarks*, y se hace la visualización por pantalla de todo esto mientras no se *quite* el programa con “q”, en su defecto, “Q”.

Extraer valores keypoints

Se logra con la siguiente función tener concatenados los resultados de las extracciones de los componentes (pose, cara, y ambas manos) en forma de arreglo, sumando 1,662 valores.

Configuración de directorios para la colección de arreglos

Se establece una ruta para los datos exportados. Se crea un arreglo con el nombre de las señas a detectar, se asigna a una variable el número de secuencias decidido (esto es, la cantidad de videos que será recolectada para cada seña): 125 y en otra variable se dispondrá de la longitud de la secuencia, es decir, del *tamaño* en número cuadros de cada uno de los videos, que será de treinta (30).

Con los datos anteriores, trabaja una doble estructura repetitiva escrita para crear todos los directorios necesarios para guardar los videos (cuadro a cuadro), lo que dará un total de 3,750 arreglos (125 videos de 30 cuadros cada uno).

Recolección de valores keypoint para training y testing

Una vez creados los directorios para almacenar los videos, se procede a la captura de las señas. El recorrido se hace en orden: primer cuadro del primer video de la primera

seña, luego los siguientes cuadros de ese video; seguidos de los cuadros del segundo video y así sucesivamente, para todas las señas. Se programó un tiempo de holgura de dos (2) segundos entre video y video, el cual también tiene efecto entre una seña y otra (último video capturado de una seña y primer video a capturar de la siguiente).

El resto de la lógica alrededor de esta etapa tiene mucha similitud con el código de la captura de prueba. En la Ilustración 101 se ve una captura con los componentes involucrados y con una serie de *keypoints* ubicados (que serán extraídos en recolección).

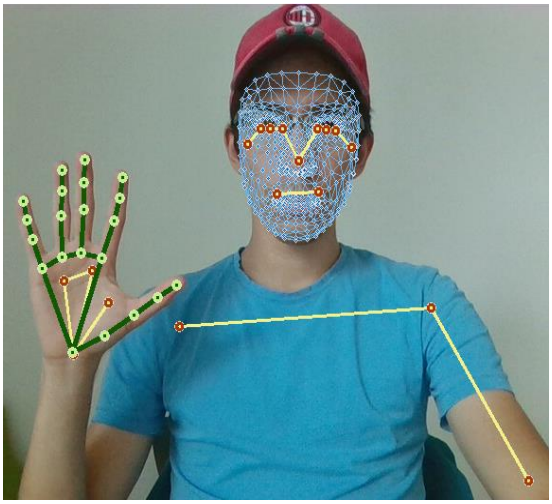


Ilustración 101 Captura de pantalla con los landmarks correspondientes a las tres (3) topologías detrás de MediaPipe Holistic. Fuente: el autor.

Preprocesamiento de los datos, y creación de etiquetas y features

Se convoca a la función *train_test_split* de *Scikit-learn* (*sklearn*) para hacer la partición de *data* acorde a lo dispuesto (9:1).

Por su parte, *to_categorical*, una de las utilidades de *TensorFlow.keras*, es usada para el etiquetado; esta función convierte un vector de clase en una matriz de clase binaria.

Se crea un mapa de etiquetado en forma de diccionario para representar cada una de las señas (“*actions*”, por ser detección de acciones). Este mapa será usado cuando se creen el conjunto de etiquetas según la seña.

Ya habiendo sido recolectados los *keypoints* (secuencias, ahora), se los debe traer y estructurarlos. Se tendrán para los 3,750 arreglos los 1662 valores que representan nuestros *keypoints*.

Se preprocesa lo anterior para tenerlo en el formato necesario para trabajar con esa *data*.

Construcción y entrenamiento de la RN tipo LSTM

Inicialmente se deben plantear las posibilidades de diseño del modelo. Se decidió esto, y se revisaban los últimos pasos para mejorar el desarrollo como un todo, entre estos cambios, haber convocado TensorBoard para ver cómo se comporta el modelo en entrenamiento, y para validación. Junto a *TensorFlow*, se usará un *callback* para detención temprana; con esto, se detendrá el entrenamiento apenas se alcance el nivel de precisión deseado. En la Ilustración 102 se presenta el proceso de entrenamiento según los estados del modelo.

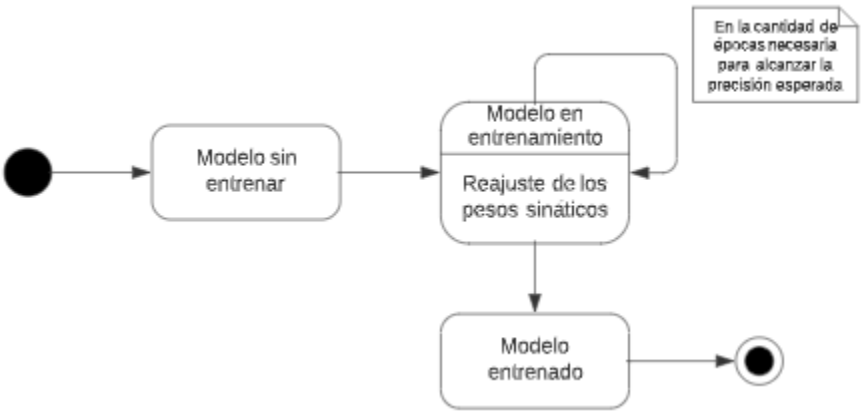


Ilustración 102 Estados del modelo – proceso de entrenamiento. Fuente: el autor.

El orden de las capas del modelo será secuencial ("*Sequential*"), adicional a esto, se trabajará con capas densas y con redes neuronales recurrentes tipo *LSTM*. La estructura del modelo se presenta en la Ilustración 103.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 64)            442112
_________________________________________________________________
lstm_1 (LSTM)                (None, 30, 128)           98816
_________________________________________________________________
lstm_2 (LSTM)                (None, 30, 64)            49408
_________________________________________________________________
lstm_3 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense (Dense)                (None, 64)                4160
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 99
=================================================================
Total params: 630,755
Trainable params: 630,755
Non-trainable params: 0
_________________________________________________________________
```

Ilustración 103 Resumen del modelo (estructura). Fuente: el autor.

Se crea un directorio log y se configuran los *callbacks* tipo TensorBoard (para visualización) y tipo "*categorical_accuracy*" para capturar la precisión de las inferencias producidas.

Guardar pesos (weights)

Fueron calculados a lo largo de los ciclos de entrenamiento los valores de los pesos de las neuronas; de estos, los últimos se conservan por ser los más precisos y para poder ser usados cargándolos cuando sea necesario. Estos valores son almacenados en un archivo de datos .h5 (formato de datos jerárquicos).

Construcción de sistema de detección de acciones

El sistema resultante es en esencia un modelo de aprendizaje profundo basado en redes neuronales recurrentes. Así como se ha dado con detectores de objetos ("estáticos"), se ha podido recientemente construir detectores de acciones, implementado esta vez para

identificar señas no estáticas. En la Ilustración 104 se puede apreciar el modelo en ejecución, mientras la Ilustración 105 presenta la continuación de la misma ejecución.

Ilustración 104 Sistema en ejecución. Posición neutral (izquierda), haciendo la seña para "hola" en LSP (centro), terminando la seña y con el resultado ya en pantalla (derecha). Fuente: el autor.

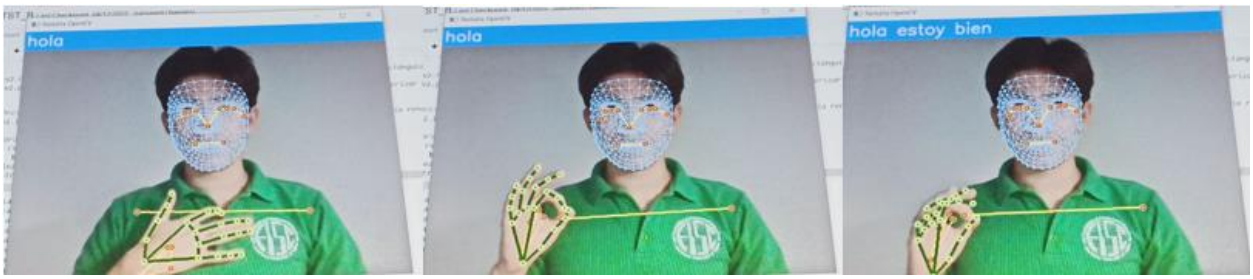


Ilustración 105 Comenzando a hacer una segunda seña: "estoy bien" (izquierda), parte final de la seña para expresar "estoy bien" (centro), traducción de la seña mientras se retoma una posición neutral (derecha). Fuente: el autor.

El sistema, una vez ejecutado, comienza a hacer la extracción de los *keypoints* a través del tiempo (de la persona captada por cámara), y con ello, la traducción de señas, de estarse haciendo alguna que el modelo sea capaz de identificar. Este caso de uso se visualiza en la Ilustración 106.

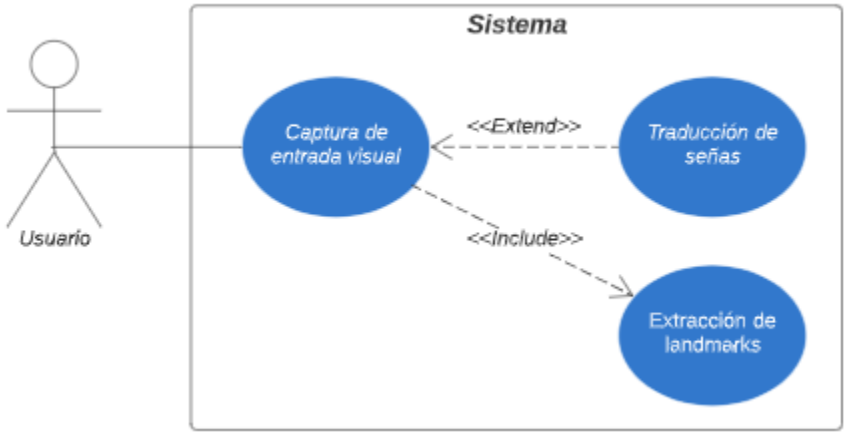


Ilustración 106 Caso de uso típico del sistema. Fuente: el autor.

El proceso para lograr la traducción involucra las etapas contempladas en la Ilustración 107, con el modelo como figura central del sistema.

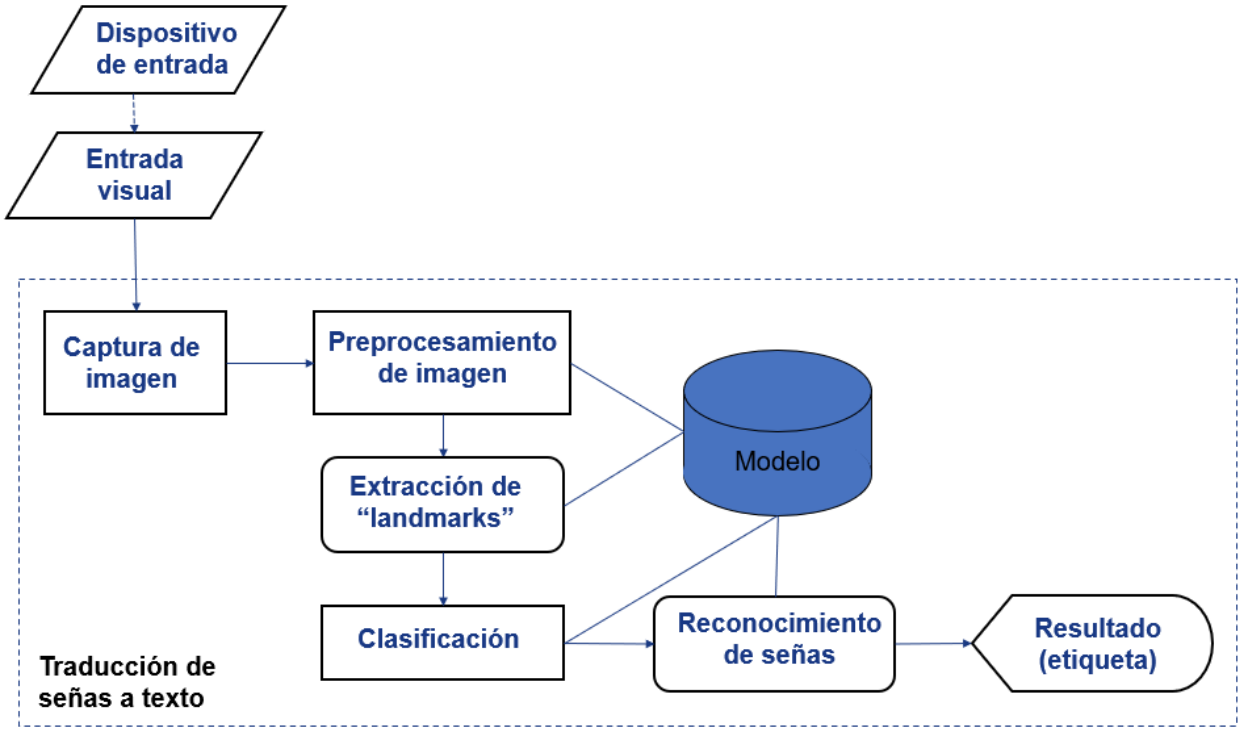


Ilustración 107 Diagrama del sistema. Fuente: el autor.

Capítulo IV – Pruebas y Validación

Modelo para señas estáticas

Más allá de las medidas de rendimiento básicas que se registran en el proceso de entrenamiento, es necesario el uso de métricas de evaluación para tener una valoración fidedigna del modelo.

Se decide utilizar la "precisión promedio media" (*mean average precision*, o mAP) para evaluar este modelo, lo cual incluye varias métricas. La precisión promedio es calculada para cada clase, y de estos valores se obtiene a su vez un promedio.

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$

Para la ecuación anterior AP_k es el promedio de la precisión de la clase k, mientras que n es el número de clases.

Configuración de las métricas a utilizar

Ha de usarse la función *plot* de la colección de funciones *pyplot*, así como la librería *numpy*, ambas previamente importadas. Además, gracias a *sklearn* podrán ser calculados los puntajes de precisión y recuperación.

Evaluación del modelo

Se crean objetos tipo arreglos *numpy* para las precisiones de las clases y para los valores "*recall*" (de la curva de recuperación de precisión) y se pasan los puntajes de predicción y las salidas esperadas. Se escribe un *script* que dé por salida una visualización de la curva "*precision-recall*", el mAP esperado es el mayor posible en una escala del cero a 1 (que sería el 100%).

Monitoreo del progreso de entrenamiento

Durante el proceso de entrenamiento se recibe retroalimentación del progreso de aprendizaje del modelo, con lo que puede irse visualizando y estudiando las salidas parciales sobre precisión alcanzada tras una serie de ciclos de entrenamiento.

Detección en tiempo real

Se consigue ejecución en tiempo real con capturas continuas con *OpenCV* para las que se hace inferencia una a una con un "*delay*" ("demora", o tiempo de espera) insignificante.

Exportando el modelo

Con los pesos calculados tras el entrenamiento requerido por definición de la propuesta, se puede exportar el modelo para traducción de señas estáticas basado en *CNN*.

Modelo para señas dinámicas

Predicciones

Se obtienen los resultados con un *predict* del *model*. Se utiliza la función *argmax()* de *numpy* para que retorne los valores máximos del eje dado, es decir, la clase que infiere es más probable que corresponda a la entrada.

Evaluación usando matriz de confusión

Las métricas por utilizar para la evaluación de este modelo serán las cuatro (4) que componen a la matriz de confusión: Verdadero positivo, falso negativo, falso positivo y verdadero negativo. Se calcula la matriz de confusión con *sklearn*, convocando la versión multietiqueta, además de la métrica de (puntaje de) predicción.

El puntaje de predicción dependerá de los valores de *ytrue* (etiquetas reales de las imágenes) y *yhat* (valores que predijo el modelo).

Ejecución en tiempo real

La traducción tiene lugar de la siguiente manera: el sistema hace captura una tras otra con un *delay* muy bajo, para cada captura se lleva a cabo el proceso de inferencia de la imagen (se espera que de una persona haciendo alguna seña o sucesión de estas) del cual resultan porcentajes para cada clase, de lo que se presenta por pantalla la etiqueta calculada como más probable, únicamente si se alcanza el valor de confiabilidad (*threshold*) del 80%. En otras palabras, se traduce en tiempo real la seña de la LSP a texto en español.

Experimentos y validación

Experimento #1:

Este experimento consta de un escenario base, y uno alternativo. Se utilizó un conjunto de datos (*Dataset_A*) de 625 videos (30 “frames” por cada uno) correspondientes a las 5 clases (señas) consideradas, a saber: “hola”, “buenos días”, “estoy bien”, “gracias” y “¿cómo estás?”. Se pueden ver versiones de secuencias de seis (6) *frames*, para ilustrar la captura de cada acción (video de una seña), en la Ilustración 108 (seña “hola”), la Ilustración 109 (seña “buenos días”), la Ilustración 110 (seña “estoy bien”), la Ilustración 111 (seña “gracias”), y la Ilustración 112 (seña “¿cómo estás?”). Cada seña se capturó, entonces, 125 veces, de las cuales un 20% (25 imágenes por clase) fue destinada a validación.

Ilustración 108 Secuencia de la seña "hola" en 6 frames. Fuente: el autor.

Ilustración 109 Secuencia de la seña "buenos días" en 6 frames. Fuente: el autor.

Ilustración 110 Secuencia de la seña "estoy bien" en 6 frames. Fuente: el autor.

Ilustración 111 Secuencia de la seña "gracias" en 6 frames. Fuente: el autor.

Ilustración 112 Secuencia de la seña "¿cómo estás?" en 6 frames. Fuente: el autor.

El tiempo de entrenamiento fue de 169.68 segundos, la pérdida llegó a 0.2217 y la *categorical_accuracy* a 0.9587, todo esto en 160 *epochs*.

A continuación, para cada clase, la matriz de confusión con las métricas de Verdadero Positivo, Falso Negativo, Falso Positivo y Verdadero Negativo.

La matriz de confusión para clase 1 (seña “hola”), se presenta en la Ilustración 113.

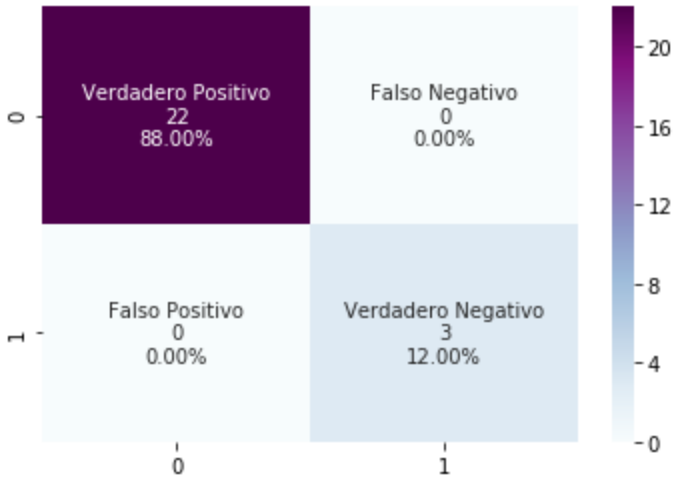


Ilustración 113 Matriz de confusión - clase "hola", experimento #1. Fuente: el autor.

La matriz de confusión para clase 2 (seña “buenos días”), se presenta en la Ilustración 114.

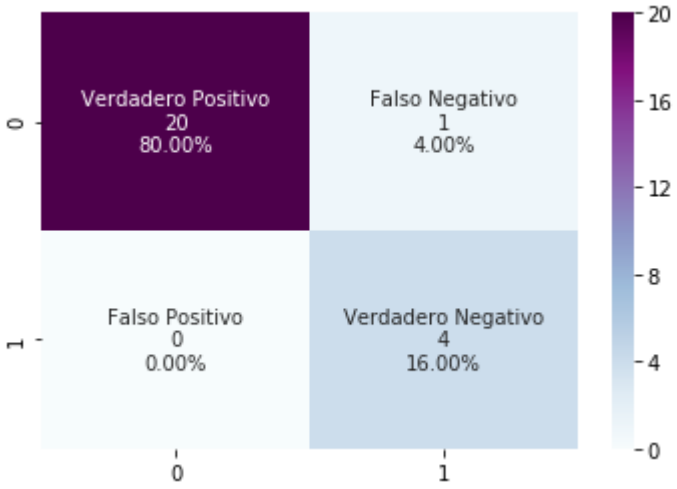


Ilustración 114 Matriz de confusión - clase "buenos días", experimento #1. Fuente: el autor.

La matriz de confusión para clase 3 (seña “estoy bien”), se presenta en la Ilustración 115.

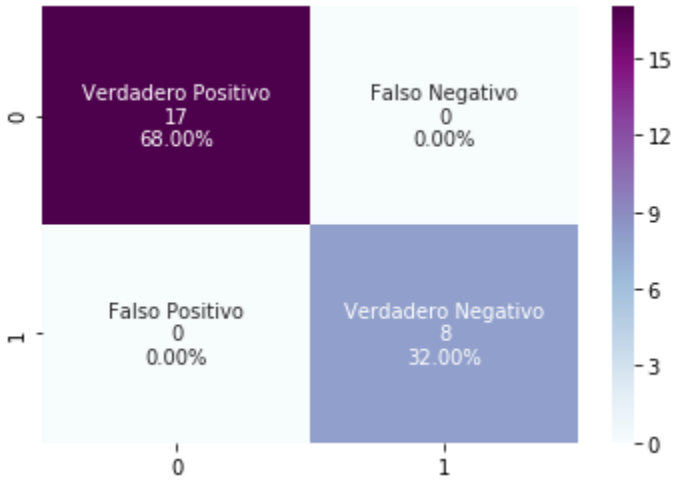


Ilustración 115 Matriz de confusión - clase "estoy bien", experimento #1. Fuente: el autor.

La matriz de confusión para clase 4 (seña “gracias”), se presenta en la Ilustración 116.

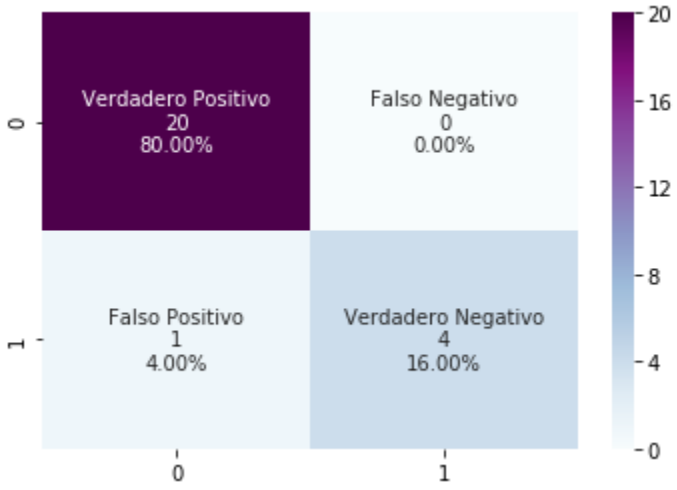


Ilustración 116 Matriz de confusión - clase "gracias", experimento #1. Fuente: el autor.

La matriz de confusión para clase 5 (seña “¿cómo estás?”), se presenta en la Ilustración 117.

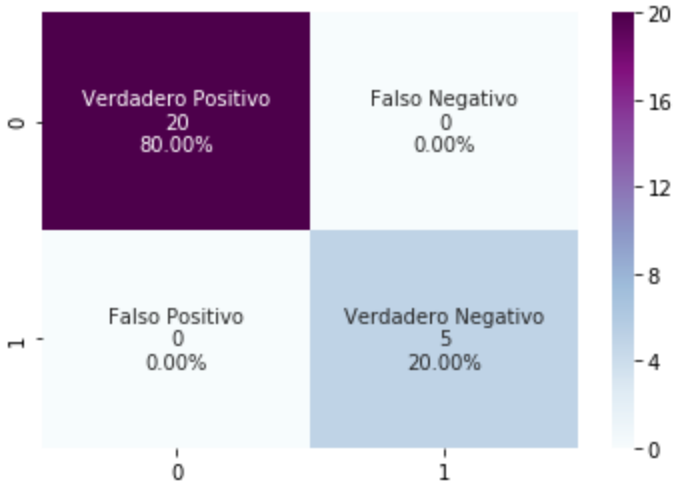


Ilustración 117 Matriz de confusión - clase "¿cómo estás?", experimento #1. Fuente: el autor.

A partir de las métricas anteriores se puede validar el modelo versus dos métricas adicionales: *accuracy* y *precision* (exactitud y precisión).

Los valores resultantes fueron un *accuracy* de 0.97 y una *precision* de 0.98. Estos y demás valores de las métricas se resumen en la Tabla 4.

Tabla 4 Resumen de las métricas (y sus valores) - Experimento #1. Fuente: el autor.

Métricas	Valores
Precisión (global)	0.958
Precisión mejor clase	1.00
Precisión peor clase	0.96
Pérdida final	0.222
Épocas	160
Tiempo de entrenamiento (s)	196.6

Escenario base: Se parte de esta versión base, que no cuenta con restricciones, y para la cual el sistema en ejecución presentó salidas descontroladas.

Escenario alternativo: Dado que en la primera prueba el modelo presentaba una seña tras otra sin aparente explicación se implementaron varios componentes. Uno de ellos controla la salida no permitiendo que se repita la misma seña de forma seguida. Otro es un valor de confiabilidad que tiene que superar el modelo para arrojar la salida, es decir, no presentará por pantalla una respuesta hasta no estar al menos 40% seguro de que sea la seña.

La diferencia entre este experimento y el anterior es apenas perceptible.

Experimento #2

Este experimento cuenta con varios escenarios con objetivos particulares. Se utilizó un conjunto de datos (*Dataset_A*) de 345 videos (30 “frames” por cada uno) correspondientes a las 3 clases (señas) consideradas, a saber: “hola”, “estoy bien”, y “gracias”. Cada seña se capturo, entonces, 115 veces, de las cuales 15 imágenes por clase (aproximadamente un 13%) fueron destinadas a validación.

La hipótesis tras este experimento es que el modelo podría tener un rendimiento mucho mejor en ejecución si se acorta el número de opciones a considerar, esto dado que existe una probabilidad compartida entre todos los elementos. Dado esta hipótesis, se tiene como objetivo mejorar el rendimiento al considerar menos clases.

El tiempo de entrenamiento fue de 52.03 segundos, la pérdida llegó a 0.9948 y la *categorical_accuracy* a 0.9879, todo esto en 46 *epochs*.

A continuación, para cada clase, la matriz de confusión con las métricas de Verdadero Positivo, Falso Negativo, Falso Positivo y Verdadero Negativo.

La matriz de confusión para clase 1 (seña “hola”), se presenta en la Ilustración 118.

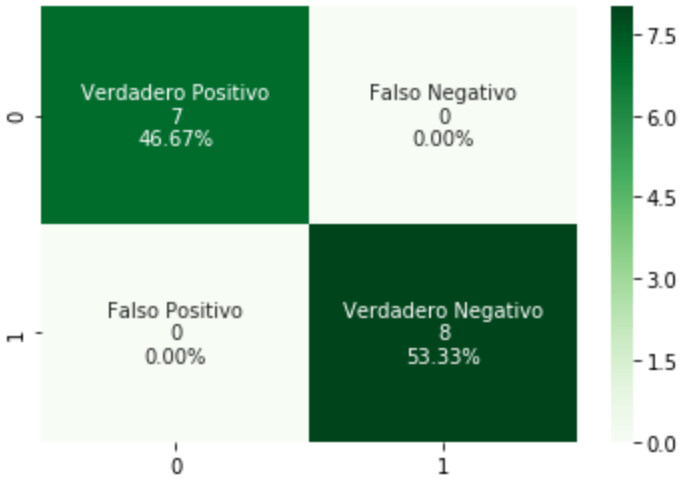


Ilustración 118 Matriz de confusión - clase "hola", experimento #2. Fuente: el autor.

La matriz de confusión para clase 2 (seña “estoy bien”), se presenta en la Ilustración 119.

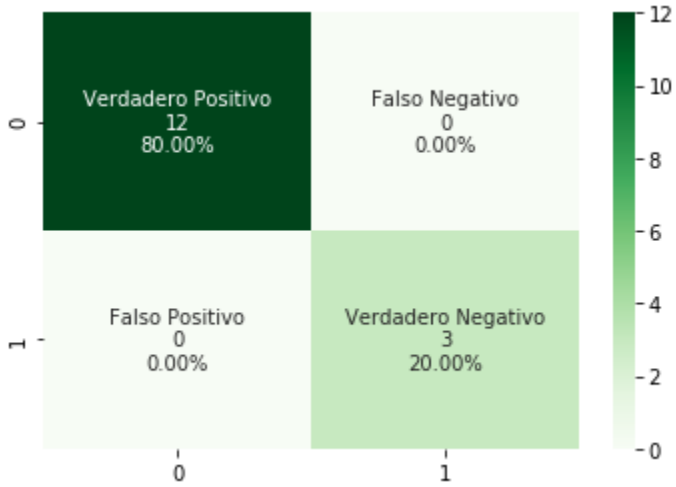


Ilustración 119 Matriz de confusión - clase "estoy bien", experimento #2. Fuente: el autor.

La matriz de confusión para clase 3 (seña “gracias”), se presenta en la Ilustración 120.

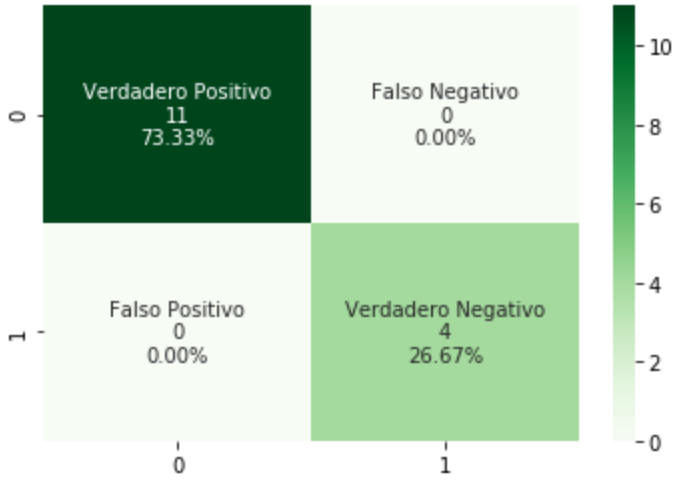


Ilustración 120 Matriz de confusión - clase "gracias", experimento #2. Fuente: el autor.

A partir de las métricas anteriores se puede validar el modelo versus dos métricas adicionales: *accuracy* y *precision* (exactitud y precisión).

Los valores resultantes fueron un *accuracy* de 0.99 y una *precision* de 0.99. Estos y demás valores de las métricas se resumen en la Tabla 5.

Tabla 5 Resumen de las métricas (y sus valores) - Experimento #2. Fuente: el autor.

Métricas	Valores
Precisión (global)	0.988
Precisión mejor clase	1.00
Precisión peor clase	1.00
Pérdida final	0.995
Épocas	46
Tiempo de entrenamiento (s)	38.88

Escenario base del Experimento #2: Reduciendo a 3 señas, la mejora del sistema en funcionamiento es perceptible y prometedora. Cabe plantearse la posibilidad de mejorar

al modelo con las técnicas usada en el experimento anterior, dado que este es un escenario sin restricciones.

Escenario alternativo #1: Se implementan los componentes anteriores (Escenario alternativo del Experimento #1) con la diferencia que el valor de confiabilidad se subió al 60% (pues ahora debe compartirse el 100% entre 3 clases en lugar de 5).

La mejora del rendimiento del modelo en ejecución es considerable. La velocidad de detección puede considerarse de tiempo real, y la precisión es alta.

Escenario alternativo #2: La hipótesis tras este escenario es que el modelo tendrá dificultad para identificar señas por entrenamiento a una distancia distinta. Queriendo ver las limitaciones del sistema se estudia cuán bueno es el rendimiento en ejecución en términos de reconocimiento de señas doblando la distancia de 0.5 metros a un (1) metro.

Se ve claramente limitado, aunque sí arroja resultados cada varios intentos. Habiendo entrenado a medio metro de distancia, la razón principal de que el modelo no funcione suficientemente bien en este caso podría deberse principalmente a la notoria diferencia entre los valores con los que se entrenó y aquellos con los que se está probando. Hay que recordar que nos referimos a valores posicionales. Una segunda razón apunta a que existe la posibilidad que las topologías de las cuales se basa la solución no sean capaces de identificar correctamente (o “tan acertadamente”) los puntos de interés.

Escenario alternativo #3: En la misma línea de querer ver las limitaciones del modelo, se estudian los resultados en ejecución para cuando hay baja iluminación. La hipótesis para este escenario es que el modelo tendrá dificultad para la captura de *keypoints* siendo ejecutado en entornos con poca iluminación. Los demás valores se respetaron respecto al Escenario alternativo #1.

En esta ocasión el modelo fue capaz de identificar las señas (tras ello traducirlas inmediatamente) sin mayores problemas. Se intuye también que el tema combinado de la distancia aumentada y la iluminación reducida sí podría ser un problema, aun habiendo entrenado al modelo con imágenes a la misma distancia que a las que se le sometiese en pruebas.

Finalmente, se muestran en la Ilustración 121 y la Ilustración 122 las curvas del aprendizaje para este segundo experimento que resultó ser la versión definitiva del modelo propuesto en el marco del proyecto.

Ilustración 121 Gráfica de precisión vs época. "Epoch_categorical_accuracy" con TensorBoard. Fuente: el autor.

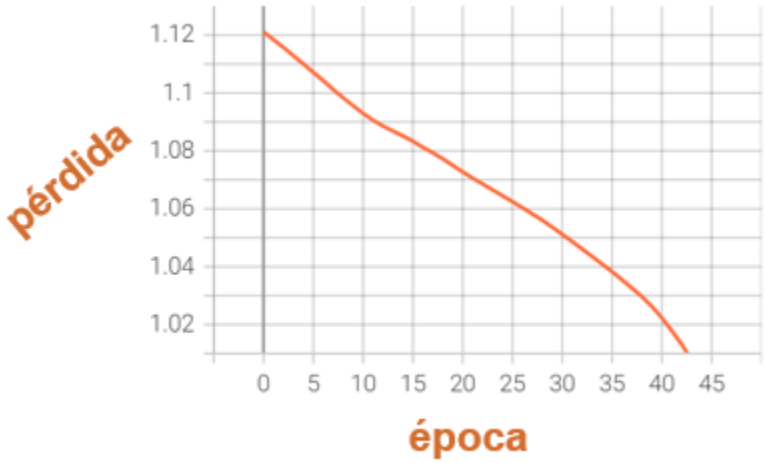


Ilustración 122 Gráfica de pérdida vs época. "Epoch_loss" con TensorBoard. Fuente: el autor.

Capítulo V – Conclusiones y Recomendaciones

Conclusiones

Indudablemente el aprendizaje automático ha demostrado ser un medio para dar solución a un sinnúmero de problemas de una variedad considerable. En particular, el aprendizaje profundo ha conseguido hacerse un lugar como una de las disciplinas más importantes de la inteligencia artificial, y es que hoy existe un gran número de estudios innovadores apoyados en *DL* que han representado cambios en el estado del arte de distintos procesos, técnicas y áreas. Este subcampo de *ML* ha aumentado su relevancia al haberse concebido estructuras de redes neuronales profundas con capacidades interesantes, sin embargo, también resaltan elementos como el disponer de herramientas y recursos facilitados por la comunidad científica (instituciones, empresas, investigadores) como bases de datos para entrenar modelos o soluciones que son base de otras nuevas.

Una de las áreas de estudio e investigación más influenciadas por el aprendizaje profundo ha sido la visión artificial, valiéndose de propuestas novedosas, como librerías y tecnologías, que se han sabido capitalizar. Las redes neuronales convolucionales constituyen las estructuras más comúnmente utilizadas para visión artificial, con una capacidad de procesamiento de *data* visual que les ha valido los últimos años una posición superlativa frente a problemas que incluyen la detección de objetos. Un modelo de este tipo puede ser adaptado para convertirse en un modelo de reconocimiento de señas, el cual puede ser el centro de un sistema de traducción de señas a texto, que a su vez presenta uno que ha sido un reto enorme para esta tarea: la limitación de identificar solo señas estáticas, es decir, solo poses de las manos.

Cada estructura de redes neuronales surgió para cubrir cierta(s) necesidad(es); y el caso de las redes neuronales recurrentes no fue distinto, siendo un planteamiento que conceptualmente permitía trabajar con recurrencias, que no es más que la posibilidad de repetir estados por los que ya ha pasado el proceso. Así, se aprovechó la bondad de las *RNN* de tener de alguna forma memoria y aunque típicamente no se habían involucrado esta estructura con el área de estudio, últimamente se ha visto cómo se han hecho implementaciones en conjunto. Existe, entonces, un paralelismo entre la conocida tarea

de detección de objetos, y la más reciente tarea de detección de acciones, para la que podemos estarnos refiriendo a movimientos de atletas, seguimiento de datos posicionales en pacientes o, como se ha propuesto en esta investigación, reconocimiento de señas dinámicas.

Se tiene entonces que, así como las señas estáticas pueden ser vistas como objetos, las señas dinámicas pueden ser vistas como acciones que se estudian a lo largo de un tiempo determinado. En virtud de poder trabajar con datos secuenciales, las redes neuronales recurrentes hacen su parte para que con detección de acciones se capten una serie de imágenes conectadas por un tipo de contexto puesto que en conjunto representan, en este caso, una seña dada, e identifican en particular cuál de las señas para las que se entrena el modelo de aprendizaje profundo es. Al presentar por pantalla la etiqueta correspondiente a la seña que infirió el modelo en ejecución, se cumple con completar una traducción.

Un enfoque de estudio no se define únicamente por las estructuras utilizadas, también por los conceptos que lo fundamentan; en tal sentido, el desarrollo de este proyecto involucró a la holística como idea determinante, considerando que además de hacer lectura y seguimiento de valores posicionales de las manos, es interesante considerar otros componentes, en particular, la cara y la postura. La conveniencia de estudiar estas características responde a que, al momento de comunicarnos mediante una lengua de señas, importa más que solo las señas que se estén haciendo con las manos, como la posición relativa entre ciertas partes del cuerpo o los gestos faciales. Estudiar, entonces, pensando en esta compleja relación entre los elementos que hacen parte de un sistema, permiten que se sea más fieles a la realidad en la práctica, de cómo se comunica un usuario de lengua de señas.

Si algo se debe destacar de lo aprendido a lo largo del proyecto, es cómo se puede reparar en las muchas posibilidades de hacer implementaciones de valor que tengan un impacto en la sociedad, no solo en estilo de vida, sino principalmente en la calidad de vida. Existen avances que permiten realizar proyectos útiles y merecedores de más investigación, con vocación y compromiso se pueden presentar soluciones, como un sistema de traducción de señas de la lengua de señas panameñas a texto en español.

Recomendaciones

En el marco del proyecto, y de la realidad que vive la investigación universitaria en Panamá, se presentan una serie de sugerencias que buscan contextualizar sobre el tema y puntualizar posibilidades como acciones y medidas a tomar para que sigan los aportes al área de estudio y a la comunidad.

- En el entendido de que un proyecto puede escalar o de que, a partir de este, pueden desprenderse otros, existen dos caminos divergentes principales a efecto de posibles trabajos futuros.
    - En primer lugar, la continuación del proyecto. La investigación se puede ampliar de diversas maneras, sobre todo con la creación de módulos. Entre las oportunidades de mejoras se puede considerar módulos como de conexiones entre los elementos, para optimizar procesos, o de estructuras que hagan que modelos gestionen independientemente la *data*, como decidir si una entrada es una seña estática o dinámica, y que sea enviada al modelo correspondiente para que este haga la inferencia. También se podría desarrollar un módulo que haga la conversión de texto a voz / audio, haciendo del sistema un traductor de señas a texto y audio. Se podría incluso hacer una implementación para que sea captada una seña (o serie de señas) y se presente una salida visual con las señas de otra lengua de señas (como la *ASL*). Una última propuesta inicial al respecto, podría ser un módulo de recolección de datos que registre imágenes o videos de señas para entrenamiento, enriqueciendo el proyecto principal.
    - Otro grupo de posibilidades se engloba en las líneas futuras de investigación relacionadas a este proyecto. Podría ampliarse la propuesta adaptando el sistema a otro par de lenguas signada-oral y gráfica, como *ASL* e inglés. Así como las *RNN* demostraron su valía para enfrentar este problema, otras estructuras bien podrían hacer lo propio; un ejemplo pudiese ser las *GAN* (de “*Generative Adversarial Network*”; red generativa antagónica).

- A los estudiantes de pregrado el llamado es para que pese a poder no estar interesados en hacer investigación en un primer momento, averigüen del tema, sean curiosos sobre las oportunidades a partir de tener este tipo de experiencia, pregunten a profesores, compañeros e investigadores de su centro educativo, involúcrense en actividades como eventos en los que se presenten investigaciones recientes. Pensar que es posible trabajar en proyecto relacionándose con pares de otras áreas (como entre alguien de sistemas computacionales y alguien de eléctrica), así como se puede tener apoyo de instituciones interesadas en proyectos de investigación. Pueden buscar activamente temas de investigación de su área de estudio; no dejará de haber oportunidades de mejora ni problemáticas que atender. Se puede enfocar la búsqueda de tema pensando en las necesidades del mercado y muy especialmente en la comunidad para tener un impacto en positivo.
- A las instituciones educativas, en especial a la Universidad Tecnológica de Panamá. Deben seguir los esfuerzos en promoción de investigación, pero también diversificarse y hacerlo buscando entender por qué existe un porcentaje tan bajo de futuros profesionales haciendo investigación. Idealmente se debería incentivar la investigación durante toda la etapa universitaria de forma integral, pero involucrando a los estudiantes entendiendo la diferencia entre aquellos que están comenzando la carrera universitaria, culminándola o “a mitad” de esta. Asignar mentores o hacer convocatorias con posibles temas de investigación podrían ser alternativas interesantes; esta última pudiese ser un listado presentado cada semestre con temas propuestos por profesores dado que ellos conocen las necesidades y qué puede ser útil, y se podría solventar lo que para muchos es un gran problema: escogencia del tema. Finalmente, establecer una directriz respecto a las giras académicas, en vista de que no todos los egresados tienen esta experiencia que se ha mostrado ser positiva para los efectos de sumar interesados en investigación; debería definirse esta directriz pensando en a qué altura de la carrera es más efectivo hacer giras académicas a centros de investigación o afines.

Referencias

[1] S. Sharma y K. Kumar, «SpringerLink,» Springer Nature Switzerland, 1 Mayo 2021. [En línea]. Disponible en: https://link.springer.com/article/10.1007/s11042-021-10768-5. [Último acceso: 10 Agosto 2021].

[2] M. Rahman, S. Islam, H. Rahman, R. Sassi, M. Rivolta y Aktaruzzaman, «IEEEXplore,» Institute of Electrical and Electronics Engineers, 24 Diciembre 2019. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/9067974. [Último acceso: 10 Agosto 2021].

[3] L. Jing, E. Vahdani, M. Huenerfauth y Y. Tian, «arXiv,» Cornell University, 7 Junio 2019. [En línea]. Disponible en: https://arxiv.org/abs/1906.02851. [Último acceso: 10 Agosto 2021].

[4] R. Fatmi, S. Rashad y R. Integlia, «IEEExplore,» Institute of Electrical and Electronics Engineers, 7 Enero 2019. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/8666491. [Último acceso: 10 Agosto 2021].

[5] Organización Mundial de la Salud, 2 Marzo 2021. [En línea]. Disponible en: https://www.who.int/es/news-room/fact-sheets/detail/deafness-and-hearing-loss. [Último acceso: 10 Agosto 2021].

[6] D. Pimentel, R. Walker y M. Fajrdo, Lengua de Señas Panameñas, Panamá: SENADIS.

[7] I. Education, «IBM,» 17 agosto 2020. [En línea]. Disponible en: https://www.ibm.com/cloud/learn/neural-networks.. [Último acceso: 8 septiembre 2021].

[8] R. Bodmer, L. Liu, W. Liu y J. C. Rangel, «Reconocimiento del lenguaje de señas mediante aprendizaje automático para niños de primaria,» Revistas UTP, Panamá, 2020.

[9] H. Raheem y F. Raheem, «American Academic Scientific Research Journal for Engineering, Technology, and Sciences,» 24 agosto 2018. [En línea]. Disponible en: https://asrjetsjournal.org/index.php/American_Scientific_Journal/article/view/4383. [Último acceso: 15 diciembre 2021].

[10] Gestión Empresarial 3000, «IPHE Inclusivo,» IPHE, Panamá, 2018.

[11] A. Rodríguez-Fuentes, L. Alaín y F. García, «EnSenias: herramienta tecnológica para aprender, enseñar, mejorar y usar la lengua de signos panameña,» íkala, Medellín, 2020.

[12] «Linyadoo,» 11 Abril 2019. [En línea]. Disponible en: http://www.linyadoo.com/ensV6/pages/index.php. [Último acceso: 10 Agosto 2021].

[13] A. Kannoth, C. Yang y M. Guanipa, «IEEE Xplore,» IEEE, 2021 diciembre 15. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/9631255. [Último acceso: 28 diciembre 2021].

[14] P. Perdana, K. Darma y P. Dharmaadi, «Jitter,» PKP, 04 octubre 2021. [En línea]. Disponible en: https://ojs.unud.ac.id/index.php/jitter/article/view/78264. [Último acceso: 16 diciembre 2021].

[15] E. Elsayed y D. Fathy, « International Journal of Advanced Computer Science and Applications,» 2020. [En línea]. Disponible en: https://pdfs.semanticscholar.org/efac/22e2eedd311e0ee59642d6a9df92823898ff.pdf.

[16] T. Abedin, K. Prottoy, A. Moshruba y S. Hakim, «Cornell University,» 25 Julio 2021. [En línea]. Disponible en: https://arxiv.org/abs/2107.11818. [Último acceso: 29 diciembre 2021].

[17] J. Fink, L. Meurant, B. Frénay y A. Cleve, «ResearchGate,» 5 abril 2021. [En línea]. Disponible en: https://www.researchgate.net/profile/Jerome-Fink-2/publication/351107565_LSFB-CONT_and_LSFB-ISOL_Two_New_Datasets_for_Vision-Based_Sign_Language_Recognition/links/6087b978881fa114b42dee5f/LSFB-CONT-and-LSFB-ISOL-Two-New-Datasets-for-Vision-Based-Sign-Lang. [Último acceso: 10 diciembre 2021].

[18] A. Calado, V. Errico y G. Saggio, «IEEEXplore,» IEEE, 2 septiembre 2021. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/9528301. [Último acceso: 27 diciembre 2021].

[19] M. Basiri, S. Nemati, M. Abdar, E. Cambria y R. Acharya, «ScienceDirect,» Elsevier, 10 Febrero 2021. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/abs/pii/S0167739X20309195#!. [Último acceso: 29 diciembre 2021].

[20] W. Yu, Y. Kim y C. Mechefske, «ScienceDirect,» Elsevier, 15 Febrero 2021. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/abs/pii/S0888327020307081. [Último acceso: 30 diciembre 2021].

[21] D. Metaxas, M. Dilsizian y C. Neidle, «NSF-PAR,» The National Science Foundation, 12 mayo 2018. [En línea]. Disponible en: https://par.nsf.gov/biblio/10065367. [Último acceso: 11 diciembre 2021].

[22] M. Rahman, E. Malaia, A. Gurbuz, G. Darrin y C. Crawford, «IEEEXplore,» IEEE, 4 Enero 2022. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/9669011. [Último acceso: 10 enero 2022].

[23] P. Mishra, Practical Explainable AI Using Python: Artificial Intelligence Model Explanations Using Python-based Libraries, Extensions, and Frameworks, Apress, 2021.

[24] F. Chollet, Deep Learning with Python, Second Edition, Manning Publications, 2021.

[25] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition, O'Reilly Media, Inc., 2019.

[26] «ECB,» ECB engineering firm, 16 Mayo 2019. [En línea]. Disponible en: https://ecbsite.com/2019/05/16/las-9-tendencias-de-inteligencia-artificial-ia-mas-utilizadas/. [Último acceso: 18 diciembre 2021].

[27] C. Fry, «Dynam.AI,» 2020. [En línea]. Disponible en: https://drive.google.com/file/d/1FClkGImOfrVSA67uKDUgtyyVPWl8Q7C2/view. [Último acceso: 15 diciembre 2021].

[28] A. Müller y S. Guido, Introduction to Machine Learning with Python, O'Reilly Media, Inc., 2016.

[29] V. Lakshmanan, M. Görner y R. Gillard, Practical Machine Learning for Computer Vision, O'Reilly Media, Inc., 2021.

[30] S. Weidman, Deep Learning from Scratch, O'Reilly Media, Inc., 2019.

[31] S. Ravichandiran, Hands-On Deep Learning Algorithms with Python, Packt Publishing, 2019.

[32] N. Singh, «XENONSTACK,» 7 abril 2021. [En línea]. Disponible en: https://www.xenonstack.com/blog/artificial-neural-network-applications. [Último acceso: 3 diciembre 2021].

[33] Kaggle, «Kaggle,» 12 octubre 2020. [En línea]. Disponible en: https://www.kaggle.com/learn/intro-to-deep-learning. [Último acceso: 20 noviembre 2021].

[34] e. a. Ali Hirsa, «Supervised Deep Neural Networks (DNNs) for Pricing/Calibration of Vanilla/Exotic Options Under Various Different Processes,» 2019.

[35] H. B. C. B.-A. J. G. I. G.-O. M. Alaiz, «Detecting Respiratory Pathologies Using Convolutional Neural Networks and Variational Autoencoders for Unbalancing Data,» *MDPI,* 2020.

[36] H. M. R. T. R. Maia, «Object Recognition Using Convolutional Neural Networks,» *IntechOpen,* 2019.

[37] Z. e. a. Alom, «A State-of-the-Art Survey on Deep Learning Theory and Architectures,» *MDPI,* 2019.

[38] M. Phi, «TowardsDataScience,» 24 septiembre 2018. [En línea]. Disponible en: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. [Último acceso: 10 octubre 2021].

[39] P. Ganesh, «TowardsDataScience,» 10 julio 2019. [En línea]. Disponible en: https://towardsdatascience.com/growing-your-own-rnn-cell-simplified-b68ba2c0f082. [Último acceso: 10 octubre 2021].

[40] «YOLOv5 Documentation,» MkDocs, 6 junio 2021. [En línea]. Disponible en: https://docs.ultralytics.com/. [Último acceso: 26 octubre 2021].

[41] K. e. a. Cherry-Allen, «Applications of Pose Estimation in Human Health and Performance across the Lifespan,» *MDPI,* 2021.

[42] H. P. E. U. M. L. i. Python, «Analytics Vidhya,» 2021. [En línea]. Disponible en: https://www.analyticsvidhya.com/blog/2021/10/human-pose-estimation-using-machine-learning-in-python/.

[43] B. Sue, «Sign Language Recognition and Translation: A Multidisciplined Approach From the Field of Artificial Intelligence,» *The Journal of Deaf Studies and Deaf Education.*

[44] H. Davidson, «Medium,» 2020. [En línea]. Disponible en: https://medium.datadriveninvestor.com/introducing-transfer-learning-as-your-next-engine-to-drive-future-innovations-5e81a15bb567.

[45] D. Martínez, «DataScience.aero,» 2020. [En línea]. Disponible en: https://datascience.aero/transfer-learning-aviation/.

[46] M. Sandler, A. Howard, M. Zhu, A. Zhmonginov y L.-C. Chen, 2018. [En línea]. Disponible en: https://openaccess.thecvf.com/content_cvpr_2018/papers/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.pdf. [Último acceso: 19 diciembre 2021].

[47] Coco, «TensorFlow,» 15 febrero 2019. [En línea]. Disponible en: https://www.tensorflow.org/datasets/catalog/coco. [Último acceso: 11 diciembre 2021].

[48] V. Bazarevsky y I. Grishchenko, «Google AI Blog,» Google, 13 agosto 2020. [En línea]. Disponible en: https://ai.googleblog.com/2020/08/on-device-real-time-body-pose-tracking.html. [Último acceso: 1 diciembre 2021].

[49] «COCO,» GitHUB, 16 Febrero 2018. [En línea]. Disponible en: https://cocodataset.org/#keypoints-2020. [Último acceso: 29 noviembre 2021].

[50] «MediaPipe,» GOOGLE LLC, 14 agosto 2020. [En línea]. Disponible en: https://google.github.io/mediapipe/solutions/pose.html. [Último acceso: 2 diciembre 2021].

[51] «MediaPipe,» Google, 9 junio 2020. [En línea]. Disponible en: https://google.github.io/mediapipe/solutions/face_mesh.html. [Último acceso: 1 diciembre 2021].

[52] «MediaPipe,» Google, 9 junio 2020. [En línea]. Disponible en: https://google.github.io/mediapipe/solutions/face_mesh.html. [Último acceso: 17 diciembre 2021].

[53] Y. Kartynnik, A. Ablavatski, . M. Grundmann y . I. Grishchenko, «Cornell University,» 15 julio 2019. [En línea]. Disponible en: https://arxiv.org/abs/1907.06724. [Último acceso: 14 diciembre 2021].

[54] M. Sandler y A. Howard, «Google AI Blog,» Google, 3 abril 2018. [En línea]. Disponible en: https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html. [Último acceso: 29 diciembre 2021].

Anexos

Anexos A: Desarrollo del módulo para señas estáticas

Configuración de rutas

```
WORKSPACE_PATH = 'Tensorflow/workspace'

SCRIPTS_PATH = 'Tensorflow/scripts'

APIMODEL_PATH = 'Tensorflow/models'

ANNOTATION_PATH = WORKSPACE_PATH+'/annotations'

IMAGE_PATH = WORKSPACE_PATH+'/images'

MODEL_PATH = WORKSPACE_PATH+'/models'

PRETRAINED_MODEL_PATH = WORKSPACE_PATH+'/pre-trained-models'

CONFIG_PATH = MODEL_PATH+'/my_ssd_mobnet/pipeline.config'

CHECKPOINT_PATH = MODEL_PATH+'/my_ssd_mobnet/'
```

Creación de mapa de etiquetado

```
labels = [{'name':'A', 'id':1},

          {'name':'E', 'id':2},

         {'name':'I', 'id':3},

         {'name':'O', 'id':4},

         {'name':'U', 'id':5}]

with open(ANNOTATION_PATH + '\label_map.pbtxt', 'w') as f:

    for label in labels:

        f.write('item { \n')

        f.write('\tname:\'{}\'\n'.format(label['name']))
```

```
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

Descarga de modelo TensorFlow preentrenado y reubicación de model config

```
!cd Tensorflow && git clone https://github.com/tensorflow/models
#
CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
#
!mkdir {'Tensorflow\workspace\models\\'+CUSTOM_MODEL_NAME}
!cp   {PRETRAINED_MODEL_PATH+'/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config'} {MODEL_PATH+'/'+CUSTOM_MODEL_NAME}
```

Actualizar Config para Transfer Learning

```
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
#
CONFIG_PATH = MODEL_PATH+'/'+CUSTOM_MODEL_NAME+'/pipeline.config'
#
config = config_util.get_configs_from_pipeline_file(CONFIG_PATH)
config
#
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(CONFIG_PATH, "r") as f:
```

```
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)
#
pipeline_config.model.ssd.num_classes = 5
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint =
PRETRAINED_MODEL_PATH+'/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-
8/checkpoint/ckpt-0'
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= ANNOTATION_PATH +
'/label_map.pbtxt'
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] =
[ANNOTATION_PATH + '/train.record']
pipeline_config.eval_input_reader[0].label_map_path = ANNOTATION_PATH +
'/label_map.pbtxt'
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] =
[ANNOTATION_PATH + '/test.record']
#
config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(CONFIG_PATH, "wb") as f:
    f.write(config_text)
```

Entrenamiento del modelo

```
print("""python {}/research/object_detection/model_main_tf2.py --model_dir={}/{} --
pipeline_config_path={}/{}/pipeline.config --
```

num_train_steps=300""".format(APIMODEL_PATH,
MODEL_PATH,CUSTOM_MODEL_NAME,MODEL_PATH,CUSTOM_MODEL_NAME))

Detección (y traducción) de señas

```
import cv2

import numpy as np

category_index = label_map_util.create_category_index_from_labelmap(ANNOTATION_PATH+'/label_map.pbtxt')

# Configuración de captura

cap = cv2.VideoCapture(1)

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while True:

    ret, frame = cap.read()

    image_np = np.array(frame)

    #

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)

    detections = detect_fn(input_tensor)

    #

    num_detections = int(detections.pop('num_detections'))

    detections = {key: value[0, :num_detections].numpy()

                  for key, value in detections.items()}

    detections['num_detections'] = num_detections
```

```
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
#
    label_id_offset = 1
    image_np_with_detections = image_np.copy()


    viz_utils.visualize_boxes_and_labels_on_image_array(
            image_np_with_detections,
            detections['detection_boxes'],
            detections['detection_classes']+label_id_offset,
            detections['detection_scores'],
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=5,
            min_score_thresh=.5,
            agnostic_mode=False)
#
    cv2.imshow('Reconocimiento de Señas (vocales LSP)', cv2.resize(image_np_with_detections, (800, 600)))
    #
    if cv2.waitKey(1) & 0xFF == ord('q'):
        cap.release()
        break
```

Anexos B: Desarrollo del módulo para señas dinámicas

Instalación de librerías y dependencias

```
import cv2

import numpy as np

import os

from matplotlib import pyplot as plt

import time

import mediapipe as mp
```

Detección de los keypoints/landmarks usando Mediapipe Holistic

```
## Configurando Mediapipe Holistic

# Variable para MPH:

mp_holistic = mp.solutions.holistic

# Variable para las drawing utilities de MP

mp_drawing = mp.solutions.drawing_utils

###

def mediapipe_detection(image, model):

    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    image.flags.writeable = False

    results = model.process(image)

    image.flags.writeable = True

    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    return image, results

###
```

```
def draw_changed_landmarks(image, results):
    # Para dibujar los landmarks de la cara (con sus conexiones)
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_TESSELATION,
                              mp_drawing.DrawingSpec(color=(191,95,0), thickness=1, circle_radius=1),
                              mp_drawing.DrawingSpec(color=(237,178,101), thickness=1, circle_radius=0.5)
                              )
    # Para dibujar los landmarks de la postura (con sus conexiones)
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(3,56,172), thickness=2, circle_radius=3),
                              mp_drawing.DrawingSpec(color=(141,241,244), thickness=2, circle_radius=3)
                              )
    # Para dibujar los landmarks de la mano izquierda (con sus conexiones)
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(3,89,17), thickness=2, circle_radius=3),
                              mp_drawing.DrawingSpec(color=(134,242,196), thickness=3, circle_radius=2)
                              )
```

```
    # Para dibujar los landmarks de la mano derecha (con sus conexiones)
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(134,242,196), thickness=2, circle_radius=3),
                              mp_drawing.DrawingSpec(color=(3,89,17), thickness=3, circle_radius=2)
                              )
```

Capturando imágenes por cámara

```
cap = cv2.VideoCapture(1)
#Para acceder al modelo mediapipe
# configurando el modelo mediapipe:
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():
#
        #Leyendo el feed
        ret, frame = cap.read() #Cuando leemos obtenemos estos dos valores return y frame (la img de la cámara)
#
        #Detección (entre feed y renderizado)
        image, results = mediapipe_detection(frame, holistic) #En vez de 'holistic' sería 'model' en general
        print(results)
```

```
        #Dibujar landmarks (entre la detección y el display)
        draw_changed_landmarks(image, results)
        #Presentar en pantalla
        cv2.imshow('Pantalla OpenCV', image)
        #Para la "current key", sale del loop
        if cv2.waitKey(10) & 0xFF == ((ord('q')) or ((ord('Q')))):
            break
    cap.release()
    cv2.destroyAllWindows()
```

Extraer valores keypoints

```
def extract_keypoints(resutls):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])
```

Configuración de directorios para la colección de arreglos

```
# Ruta para los datos exportados, arreglos numpy

DATA_PATH = os.path.join('Datos_Dataset_A')

# Acciones (5) a detectar

actions = np.array(['hola', 'buenos dias', 'estoy bien', 'gracias', 'como estas'])

# 125 'videos' de datos (por seña)

no_sequences = 125

# Tamaño/longitud de los videos (en frames)

sequence_length = 30

###

for action in actions:

    for sequence in range(no_sequences):

        try:

            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))

        except:

            pass
```

Recolección de valores keypoint para training y testing

```
cap = cv2.VideoCapture(1)

#

with  mp_holistic.Holistic(min_detection_confidence=0.5,  min_tracking_confidence=0.5)
as holistic:

    # Loop por cada acción

    for action in actions:
```

```
        # Loop por secuencia/video
        for sequence in range(no_sequences):
            # Loop por frame (según longitud del video/ la secuencia)
            for frame_num in range(sequence_length):
#
                #Leyendo el feed
                ret, frame = cap.read() #Cuando leemos obtenemos estos dos valores return y frame (la img de la cámara)
#
                #Detección (entre feed y renderizado)
                image, results = mediapipe_detection(frame, holistic) #En vez de 'holistic' sería 'model' en general DDD
                print(results)
#
                #Dibujar landmarks (entre la detección y el display)
                draw_changed_landmarks(image, results) # alternativamente: draw_styled_landmarks(image, results)
                #Tiempo de espera entre capturas
                if frame_num == 0:
                    cv2.putText(image, 'COMENZANDO RECOLECCIÓN', (120,200),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 3, cv2.LINE_AA)
                    cv2.putText(image, 'Recolectando frames para {} número de video {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
```

```
            cv2.waitKey(2000) #2 segundos
        else:
            cv2.putText(image, 'Recolectando frames para {} número de video {}'.format(action, sequence), (15,12),
                        cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
        # Exportar keypoints (extraerlos y guardarlos en las carpetas)
        keypoints = extract_keypoints(results) #results de la función de detección MP
        npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num)) # Dónde se guarda el frame y su nombre
        np.save(npy_path, keypoints)
        #Presentar en pantalla
        cv2.imshow('Pantalla OpenCV', image)
        #Para la "current key", sale del loop
        if cv2.waitKey(10) & 0xFF == ((ord('q'))or ((ord('Q')))):
            break
    cap.release()
    cv2.destroyAllWindows()
```

Preprocesamiento de los datos, y creación de etiquetas y features

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
#
label_map = {label:num for num, label in enumerate(actions)}
print(label_map)
```

```
#

#Se trae la data:

sequences, labels = [], []          # Arreglos vacíos. sequences: Features data o X. labels: y data

for action in actions:

    for sequence in range(no_sequences):

        window = []                 # Representa todos los diferentes frames que se tienen por una secuencia particular

        for frame_num in range(sequence_length):

            res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num))) #Para cargar cada frame

            window.append(res)      # Append a todas las secuencias

        sequences.append(window)    # Tomamos el video y lo anexamos a nuestras secuencias

        labels.append(label_map[action])# Para anexar nuestra etiqueta a lo anterior

###

X = np.array(sequences)

#

y = to_categorical(labels).astype(int)
```

Construcción y entrenamiento de la RN tipo LSTM

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from tensorflow.keras.callbacks import TensorBoard

from tensorflow.keras.callbacks import Callback
```

```
###
log_dir = os.path.join('Logs_A')
tb_callback = TensorBoard(log_dir=log_dir)
###
# Diseño/Definición del modelo
DESIRED_ACCURACY = 0.96
class myCallback(Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('categorical_accuracy') is not None and logs.get('categorical_accuracy') >= DESIRED_ACCURACY) :
      print("\n96% de precisión alcanzada!")
      self.model.stop_training = True
ac_callback = myCallback()
            #
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
#
model = Sequential()  #tf.keras.models.Sequential
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662))), #64 unidades LSTM. i_s=(no_frames,no_kps)
model.add(LSTM(128, return_sequences=True, activation='relu')),
model.add(LSTM(64, return_sequences=True, activation='relu')),
model.add(LSTM(64, return_sequences=False, activation='relu')),
model.add(Dense(64, activation='relu')),
```

```
model.add(Dense(32, activation='relu')),

model.add(Dense(32, activation='relu')),

model.add(Dense(actions.shape[0], activation='softmax'))

###

model.compile(optimizer='adadelta', loss='categorical_crossentropy', metrics=['categorical_accuracy'])

###

model.fit(X_train, y_train, epochs=800, callbacks=[ac_callback, tb_callback])

###

model.summary()

#

model.save('trad_senias_txt.h5')
```

Detección (y traducción) de señas a texto, por detección de acciones

```
# 1. Variables de detección

sequence = []    # Recolecta los 30 frames para generar la predicción (Capturamos los frames y se lo pasamos al algoritmo)

sentence = []    # Nos permitirá concatenar nuestro historial de detección

predictions = [] # Toma las (últimas 10) predicciones

threshold = 0.9 # Métrica de confiabilidad según la cual se harán renders solo si los resultados están por encima de este valor

#

# Implementación del modelo de predicción

cap = cv2.VideoCapture(1)

#Para acceder al modelo mediapipe
```

```
# configurando el modelo mediapipe:

with  mp_holistic.Holistic(min_detection_confidence=0.8,  min_tracking_confidence=0.6) as holistic:

    while cap.isOpened():

#

        #Leyendo el feed

        ret, frame = cap.read() #Cuando leemos obtenemos estos dos valores return y frame (la img de la cámara)

#

        #Detección

        image, results = mediapipe_detection(frame, holistic)

        print(results)

  #

        #Dibujar landmarks (entre la detección y el display)

        draw_changed_landmarks(image, results)

    #

        #2. Predicción

        keypoints = extract_keypoints(results)

        sequence.append(keypoints)

        sequence = sequence[-30:] # se toman los últimos 30 frames para hacer la predicción correspondiente

      #

        if len(sequence) == 30:
```

```
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])
            predictions.append(np.argmax(res))
            #
        #3. Lógica de visualización
            if np.unique(predictions[-8:])[0]==np.argmax(res):
                if res[np.argmax(res)] > threshold:
#
                    if len(sentence) > 0:             # Revisa si se tienen palabras en el arreglo 'sentence', si no: (*)
                        if actions[np.argmax(res)] != sentence[-1]:
                            sentence.append(actions[np.argmax(res)])
                    else:
                        sentence.append(actions[np.argmax(res)])    # (*) append
#
            if len(sentence) > 5:
                sentence = sentence[-5:] # se toman los últimos 5 valores
#
        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1) #-1 significa que llena el rectángulo
        cv2.putText(image, ' '.join(sentence), (3,30),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA) # para renderizar la oración
```

```
        #Presentar en pantalla
        cv2.imshow('Pantalla OpenCV', image) # antes cv2.imshow('OpenCV Feed', frame) porque hacía rendering del frame
#
        #Para la "current key", sale del loop
        if cv2.waitKey(10) & 0xFF == ((ord('q')) or ((ord('Q')))):
            break
    cap.release()
    cv2.destroyAllWindows()
```

Anexos C: Pruebas y validación - Modelo para señas dinámicas

Evaluación usando matriz de confusión (y visualización de resultados)

```
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score, precision_score, recall_score

#

y_pred = model.predict(X_test)

#

y_true = np.argmax(y_test, axis=1).tolist() #extrae las clases predecidas.

y_pred = np.argmax(y_pred, axis=1).tolist()

#

type(y_pred) #is a list

# convirtiendo list a array

ytrue = np.asarray(y_true)

ypred = np.asarray(y_pred)

actions_array = [0, 1, 2, 3, 4] #5 clases

actions_array = np.asarray(actions_array)

#

rr = multilabel_confusion_matrix(ytrue, ypred, labels= actions_array)

print(rr)

#

rr_flipped = np.flip(rr)

print(rr_flipped)
```

```
r = rr_flipped
print(r)
#
# Para ajustar el orden de las métricas en las matrices
print(np.flip(r[0])) # _ class confusion matrix
print(np.flip(r[1])) # __ class confusion matrix
print(np.flip(r[2])) # ___ class confusion matrix
print(np.flip(r[3])) # ___ class confusion matrix
print(np.flip(r[4])) # ___ class confusion matrix
#
# Dividir en arreglos independientes (uno por cada una de las (3/5) clases)
arr0 = rr[0] #Antes r (por flip)
arr1 = rr[1]
arr2 = rr[2]
arr3 = rr[3]
arr4 = rr[4]
#
# Sumar el primer y último elemento para cada arreglo, estos son los verdaderos positivos y negativos.
sum_arr0 = arr0[0,0] + arr0[1,1]
sum_arr1 = arr1[0,0] + arr1[1,1]
sum_arr2 = arr2[0,0] + arr2[1,1]
sum_arr3 = arr3[0,0] + arr3[1,1]
```

```
sum_arr4 = arr4[0,0] + arr4[1,1]

print(sum_arr0)

print(sum_arr1)

print(sum_arr2)

print(sum_arr3)

print(sum_arr4)

arr_sum = sum_arr0 + sum_arr1 +sum_arr2 + sum_arr3 +sum_arr4

#

# Para precisión

# Precisión = True positive / (TPos + FPos)

tpos = arr0[0,0] + arr1[0,0] + arr2[0,0] + arr3[0,0] + arr4[0,0]

fpos = arr0[1,0] + arr1[1,0] + arr2[1,0] + arr3[1,0] + arr4[1,0]

#

# Para recall

# Recall = True positive / (TPos + FNeg)

# tpos ya calculado tpos = arr0[0,0] + arr1[0,0] + arr2[0,0]

fneg = arr0[0,1] + arr1[0,1] + arr2[0,1] + arr3[0,1] + arr4[0,1]

#

np.sum(r)

###

# A partir de las métricas anteriores (falso positivo, etc.)

# se pueden calcular otras que nos ayuden a ver cómo se está comportando el modelo:

# Accuracy Precision Recall
```

```
rr = r #r = rr_flipped

# Accuracy / Exactitud

acc = arr_sum / np.sum(r) #acc = (r[0][0] + r[-1][-1]) / np.sum(r)

print(acc)

#

##acc2 = accuracy_score(y_true, y_pred) #acc2 = sklearn.metrics.accuracy_score(y_true, y_pred)

#print(acc2)

# Precision

pres = tpos/(tpos+fpos)

print(pres)

#

# Recall

rec = tpos/(tpos+fneg)

print(rec)

#CON recall = recall_score(y_true, y_pred, pos_label="positive")

###

r = multilabel_confusion_matrix(y_true, y_pred)

print(r)

#

# VISUALIZACIÓN DE MATRIZ DE CONFUSIÓN

# (para cada clase/seña)

import seaborn as sns
```

```
sns.heatmap(arr0, annot=True)
#
#Clase 1
# Seña “hola”
group_names = ['Verdadero Positivo','Falso Negativo','Falso Positivo','Verdadero Negativo']
group_counts = ["{0:0.0f}".format(value) for value in
                arr0.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     arr0.flatten()/np.sum(arr0)]
#
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(arr0, annot=labels, fmt='', cmap='BuPu')
#
#Clase 2
# Seña “buenos días”
group_names = ['Verdadero Positivo','Falso Negativo','Falso Positivo','Verdadero Negativo']
group_counts = ["{0:0.0f}".format(value) for value in
                arr1.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
```

```
                arr1.flatten()/np.sum(arr1)]
#
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
        zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(arr1, annot=labels, fmt='', cmap='BuPu')
#
#Clase 3
# Seña “estoy bien”
group_names = ['Verdadero Positivo','Falso Negativo','Falso Positivo','Verdadero Negativo']
group_counts = ["{0:0.0f}".format(value) for value in
            arr2.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                arr2.flatten()/np.sum(arr2)]
#
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
        zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(arr2, annot=labels, fmt='', cmap='BuPu')
#
#Clase 4
# Seña “gracias”
```

```
group_names = ['Verdadero Positivo','Falso Negativo','Falso Positivo','Verdadero Negativo']

group_counts = ["{0:0.0f}".format(value) for value in

        arr3.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in

        arr3.flatten()/np.sum(arr3)]

#

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in

    zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)

sns.heatmap(arr3, annot=labels, fmt='', cmap='BuPu')

#

#Clase 5

# Seña “¿cómo estás?”

group_names = ['Verdadero Positivo','Falso Negativo','Falso Positivo','Verdadero Negativo']

group_counts = ["{0:0.0f}".format(value) for value in

        arr4.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in

        arr4.flatten()/np.sum(arr4)]

#

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in

    zip(group_names,group_counts,group_percentages)]
```

```
labels = np.asarray(labels).reshape(2,2)

sns.heatmap(arr4, annot=labels, fmt='', cmap='BuPu')

# FINAL.
```