Escuela Politécnica Superior

Verificación de usuarios de smartphones a partir de sus características biométricas de escritura

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Sergio Alcaraz Sánchez

Tutor/es:
Antonio Javier Gallego Sánchez

Jorge Calvo Zaragoza

Universitat d'Alacant
Universidad de Alicante

Septiembre 2018

Justificación y objetivos

En el presente trabajo se ha desarrollado un sistema que permite la verificación de usuarios de smartphones a partir de su estilo de escritura. Para ello se han implementado diversos algoritmos que mediante una base de datos donde se recopilan distintos perfiles de uso del teclado, permiten identificar de forma automática al usuario del mismo.

Se ha desarrollado una aplicación de teclado de Android que, instalada en los distintos dispositivos, captura los parámetros biométricos de sus usuarios mientras está siendo utilizada. Como paso intermedio se realiza un tratamiento de la información recopilada con la que se seleccionan y agrupan los datos de interés.

Finalmente, con toda la información recopilada, una red neuronal recurrente analiza dicha información y la clasifica de modo que permite reconocer la persona que está utilizando el teclado. Para la validación del trabajo se ha realizado un proceso de experimentación analizando distintas configuraciones de redes y de los datos de entrenamiento, demostrando que el método propuesto permite verificar al usuario con una tasa de acierto cercana al 95%.

Índice

Índice de imágenes

Índice de tablas

1. Introducción

Durante los últimos años la popularidad de los smartphones ha aumentado considerablemente, llegando al punto en el que podemos almacenar más información personal en ellos que en nuestro propio PC.

Esto los ha convertido en un objetivo atractivo para cualquier atacante o código malicioso. Los atacantes que quieran hacerse con nuestros smartphones podrían quererlo no solo por el valor de estos, sino por el contenido que almacenamos en ellos.

Ante este problema, nos encontramos con que todavía se siguen utilizando técnicas de bloqueo como el PIN o el patrón de bloqueo, que en muchos casos no son suficientes para la protección de nuestros datos [4]. Además, si el móvil cayera en manos ajenas desbloqueado o si se nos obligara a la fuerza a desbloquearlo, toda nuestra información quedaría expuesta sin un sistema de post-autentificación que verifique que la persona que está utilizando el dispositivo es su propietaria. Con este fin, se puede utilizar una verificación mediante el análisis de la escritura para identificar al usuario del teléfono. Algunos trabajos relacionados han demostrado que los datos de los sensores de movimiento de un teléfono inteligente se pueden usar para inferir las pulsaciones de teclas pulsada en su pantalla táctil [1, 2, 3].

El objetivo es demostrar que la verificación mediante el teclado virtual es segura para corroborar la identidad del usuario del dispositivo. Para ello primero se implementará un teclado que recogerá datos biométricos sobre el uso del teclado del usuario. Estos datos serán utilizados después para entrenar una red neuronal que distinguirá si la persona que está utilizando el teclado es el propietario del terminal o no.

Este proyecto de investigación se establece en la intersección de varios campos de investigación. El foco central es la seguridad de los teléfonos móviles, además se basa principalmente en métodos de aprendizaje automático para el análisis de datos y la seguridad de nuestro terminal.

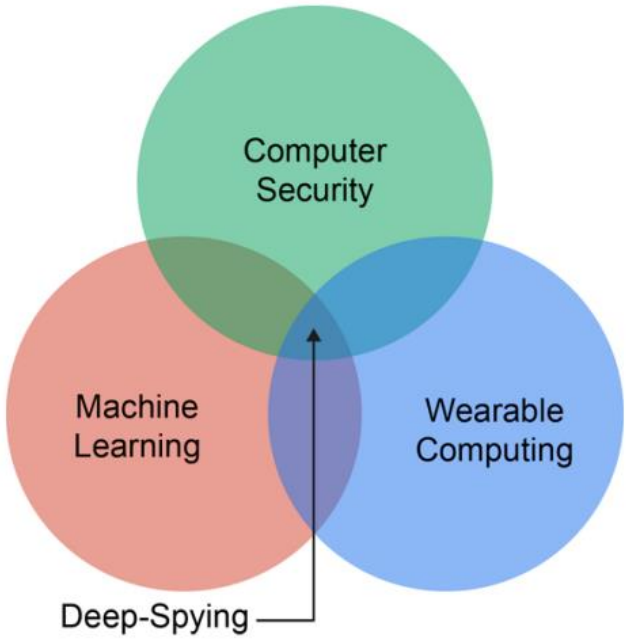

Imagen 1: Diagrama de Venn Euler

Los móviles modernos generalmente vienen con sensores de movimiento incorporados que miden los movimientos del dispositivo. El análisis de los datos de salida de tales sensores permite la estimación de los tipos específicos de movimiento que sufre el dispositivo, como la traslación, la inclinación, la sacudida, la rotación o el balanceo. Los sensores de movimiento típicos disponibles en dispositivos estándar son acelerómetro, giroscopio, sensor de gravedad, aceleración lineal y orientación. Otros sensores que también se pueden utilizar son el de presión, los datos obtenidos a través de la pantalla táctil, la cantidad de pulsaciones activas, la duración de la pulsación, o el área presionada. Los sensores basados en software generalmente derivan sus datos de sensores basados en hardware, concretamente los acelerómetros (uno para cada eje *x*, *y*, y *z*), y el giroscopio [5 - 8].

1.1 Objetivos

El objetivo general de este proyecto es desarrollar un sistema que aproveche los datos que nos proporcionan los sensores de movimiento para mejorar la seguridad de nuestro terminal. Para conseguir este objetivo general se plantean los siguientes objetivos específicos a seguir:

- Crear un teclado virtual que permita la recogida de los valores de los sensores del terminal mientras el usuario escribe con él.
- Recopilar datos de voluntarios que se han ofrecido para utilizar el teclado, estos datos serán anonimizados para la seguridad de los voluntarios.
- Diseñar una red neuronal capaz de diferenciar a cada uno de los voluntarios cuando escriben utilizando las características proporcionadas por los sensores del terminal.
- Entrenar y evaluar la red, con el fin que el propietario del terminal sea la única personal capaz de utilizar el dispositivo.

2. Marco teórico

Las redes neuronales recurrentes (RNN), en particular las basadas en Long Short-Term Memory (LSTM) [9], modelan datos secuenciales de longitud variable, logrando buenos resultados para problemas que abarcan el procesamiento del lenguaje natural, descripciones de imágenes, reconocimiento de escritura a mano y análisis genómico [10 - 17]. Las LSTM pueden capturar dependencias de corto y largo alcance, incluyendo características que definan funciones no lineales. Algunos modelos de secuencia, como los modelos de Markov, los campos aleatorios condicionales y los filtros de Kalman, se ocupan de los datos secuenciales, pero no permiten aprender las dependencias de largo alcance. Otros modelos requieren conocimiento de ingeniería de características para modelar los datos manualmente, impidiendo la inferencia de características de forma automática. Por el contrario, las redes neuronales aprenden las representación interna de los datos de forma automática, pudiendo aprender estructuras imprevistas.

La NSA ha hecho pruebas para la verificación del usuario a partir de un teclado Swipe. Según indican *"Las personas pueden reproducir tu escritura a mano en dos dimensiones, pero no podrían reproducirla en tres o cuatro dimensiones. La tercera dimensión es la presión que pones, además de las dos dimensiones en el papel. La cuarta dimensión es el tiempo. La autenticación de tipo de escritura a mano más avanzada te rastrea en cuatro dimensiones."* [18].

Utilizando estas técnicas, la NSA también ha desarrollado un sistema para el FBI que realiza la identificación biométrica de imágenes faciales, huellas dactilares, impresiones de la palma de la mano, de la retina y de imágenes de tatuajes [18].

Otra investigación que se llevó a cabo sobre la identificación de la actividad que realiza el usuario de un teléfono móvil a partir de sus sensores se realizó utilizando las redes neuronales convolucionales como método para la identificación de actividad humana [46].

Las Redes Neuronales no solo explotan la dependencia local temporal inherente de las señales de series temporales 1D y las características jerárquicas de las actividades, sino que también proporciona una manera de extraer automáticamente y de forma adaptable las características relevantes sin la necesidad de preprocesamiento [19].

Uno de los trabajos más citados en el campo del reconocimiento de actividades mediante giroscopios con varios algoritmos de minería de datos fue escrito por Bao e

Intille [20]. Este trabajo concluyó que el sensor colocado en el muslo fue el más efectivo para reconocer las diferentes actividades. Este fue el hallazgo que Kwapisz y sus compañeros utilizaron para realizar el reconocimiento de actividades humanas con solo un acelerómetro insertado en un smartphone [21]. Con sus propias características preparadas a mano a partir de datos de sensores, su trabajo mostró que los árboles de decisión J48 y perceptrones multicapa logran un mayor rendimiento en términos de precisión en comparación con otras técnicas de minería de datos; sin embargo, ambos clasificadores no pueden diferenciar de manera eficiente entre actividades muy similares, como subir y bajar escaleras.

En el trabajo realizado por Duffner, Berlemont, Lefebvre y García [22], se utilizaron las redes neuronales convolucionales junto con datos de acelerómetro y giroscopio para el reconocimiento de gestos. En este trabajo concluyeron que las redes convolucionales supera a otros métodos de última generación en gesto reconocimiento incluyendo DTW (Dynamic Time Warping) y HMM (Hidden Markov Models).

También se han llevado a cabo estudios explotando las características de los teclados físicos, como las ondas electromagnéticas [23], el sonido [24 - 26] y el tiempo [27, 28]. Sin embargo, tales características no son efectivas para su uso en un teclado virtual [29].

Usar el movimiento como canal principal implica que solo se usará la dinámica del movimiento para tratar de inferir información sobre el sistema. En nuestro caso, detectar las pulsaciones de tecla que realiza el usuario en un teclado virtual. Este método funciona en base a la observación de que el movimiento del dispositivo durante una pulsación de tecla se correlaciona y es coherente con las teclas escritas por un usuario concreto. En el caso de ataque en un dispositivo móvil, es razonable suponer que el movimiento del dispositivo, mientras el usuario escribe, se ve afectado por múltiples factores. Por ejemplo, la dimensión del dispositivo, la orientación de la pantalla, las especificaciones de los chips del sensor, el diseño del teclado, los hábitos del usuario y la posición y el movimiento relativos del usuario. Esta suposición lleva a muchos investigadores de seguridad a cuestionar la practicidad de tal ataque. Sin embargo, los estudios [30, 31] han demostrado que el ataque por inferencia mediante pulsaciones basadas en el movimiento sigue siendo efectivo y práctico a pesar de las suposiciones obvias de que los factores previamente enunciados podrían alterar la solidez y la precisión de la inferencia. Por lo tanto el

movimiento se establece como un canal de información significativo que permite la identificación del usuario.

La clasificación de la señal de los sensores de movimiento se ha explorado ampliamente en estudios que implican el reconocimiento de actividad en el campo de la Computación generalizada [32] y para el reconocimiento de gestos en el área de Interacción humano-computadora [33]. En ambos campos, el uso de sensores como el acelerómetro se estudia más profundamente, aunque algunos estudios exploraron la fusión de sensores para aumentar la robustez [34]. Si bien los enfoques utilizados en estos campos se pueden tomar prestados, existe una diferencia importante. De hecho, la duración del movimiento y la amplitud de movimiento de una pulsación son, respectivamente, mucho más cortos e inferiores al movimiento causado por un gesto (por ejemplo, deslizar el dedo, agitar la mano). Dado que la clasificación es un enfoque para reconocer patrones en la señal, se puede argumentar que los patrones pueden surgir más significativamente en señales largas con gran amplitud. Por lo tanto, esto hace que las pulsaciones de teclas sean difíciles de clasificar.

2.1 Introducción a las redes neuronales

La Red Neuronal Artificial (ANN) es una clase de modelo estadístico de inspiración biológica que consiste en un conjunto de nodos conectados (es decir, neuronas) donde cada conexión (es decir, sinapsis) tiene un peso asociado con ella [35]. Una ANN típica consiste en una capa de entrada con un número de neuronas igual a la longitud del vector de características. Su capa de salida puede construirse con un número variable de neuronas dependiendo de la tarea en cuestión (por ejemplo, igual al número de clases para la clasificación, dos neuronas de salida para la clasificación binaria, una neurona de salida para la regresión). Las ANN generalmente están compuestas por capas ocultas que contienen cada una un número variable de neuronas como se muestra en la siguiente figura.

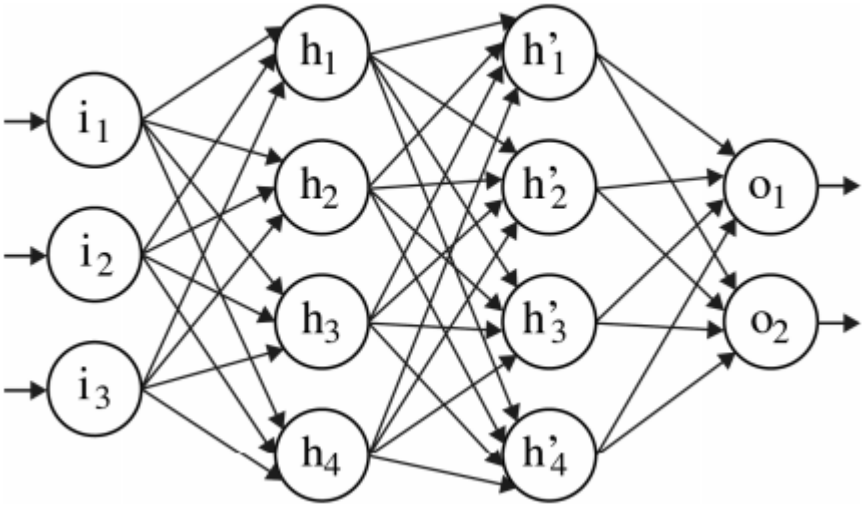

Imagen 2: Representación de las capas de una red neuronal

La red se activa al alimentar su capa de entrada con un vector de características y calcula su vector de salida gracias a la estructura interna de la red. Las neuronas obtienen sus salidas mediante el uso de una función de activación predefinida. El valor de salida de una neurona dada se puede calcular de la siguiente manera:

$$y_i = \phi(x_i) = \phi\left(\sum_{j=1}^{n} W_{ij} y_j\right)$$

Imagen 3: Cálculo del valor de salida de una neurona

Donde φ es la función de activación de la neurona, n el número de neuronas conectadas a la neurona i, W el peso asociado con la conexión entre dos neuronas, y *y* el valor de salida. La entrada *x* a una neurona es por lo tanto la suma ponderada de las salidas de las neuronas conectadas como se ilustra en la siguiente figura.

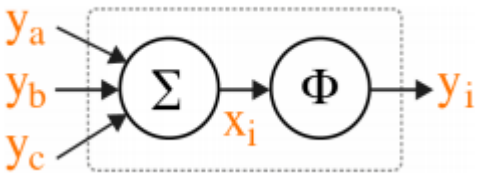

Imagen 4: Representación del cálculo del valor de salida de una neurona

Los términos Feedforward Neural Network (FNN) y Vanilla Neural Network se utilizan para referirse a la arquitectura ANN más básica donde las neuronas están conectadas hacia adelante de forma acíclica. Es decir, el flujo de activación es unidireccional desde la capa de entrada a la capa de salida.

Una ANN se entrena ajustando sus pesos hasta que se genere el vector de salida correcto a partir de una entrada dada a fin de minimizar el error global. El aprendizaje de funciones no supervisadas es un proceso en el que el modelo selecciona características automáticamente a través del entrenamiento [35]. Para esto se utiliza el método "Backpropagation", el cual se utiliza en redes neuronales artificiales para realizar esta extracción de características. Se usa comúnmente para entrenar redes neuronales profundas [36]. Backpropagation es un caso especial de una técnica más antigua y más general llamada diferenciación automática. En el contexto del aprendizaje, la propagación de errores hacia atrás es comúnmente utilizada por el algoritmo de optimización para ajustar el peso de las neuronas. En esta técnica el error se calcula en la salida y se distribuye de vuelta a través de las capas de la red.

Finalmente, una vez se ha entrenado la red, se puede realizar la fase de clasificación, que es el proceso de asignar categorías a los datos. Para esta fase simplemente se ha de realizar una pasada hacia delante de los nuevos datos que se deseen clasificar, y la red devolverá una salida con la predicción (categoría o valor) para estos nuevos datos.

3. Desarrollo del proyecto

3.1 Metodología

Para este proyecto se utilizó la metodología ágil Kanban [48]. En este método se utilizan tarjetas virtuales que representan tareas que hay que completar. Cada tarjeta tiene un estado: “Por hacer”, “En progreso” y “Hecho”. Es un método simple y visual para organizar el trabajo y dividir el proyecto en tareas más simples. Para seguir esta metodología se ha utilizado Trello [52], software gratuito de administración de proyectos. Permite crear tareas, listas, etiquetar eventos, agregar comentarios y las tareas pueden moverse de una lista a otra fácilmente.

En primer lugar, se implementa el teclado que se encargará de la recogida de datos biométricos, para comenzar a guardar datos lo antes posible y asegurarse de que se cuenta con una cantidad significativa de muestras.

En segundo lugar, se implementan los scripts que procesaran los archivos generados, que filtran y almacenan los datos en un fichero para que sea más fácil cargar los datos en la red. También se crean los algoritmos de preprocesamiento de la información, como el cálculo de características nuevas y la eliminación de datos irrelevantes.

Lo siguiente es implementar la red LSTM, así como el método de validación K-Fold Cross Validation. Una vez se tiene una cantidad significativa de datos se puede empezar el entrenamiento de la red.

Por último, se implementa un script que comprueba un fichero de texto con los datos biométricos de la escritura de una persona utilizando la red ya entrenada.

En la siguiente figura se puede ver un ejemplo de uso de la herramienta Trello para la organización de las tareas del proyecto:

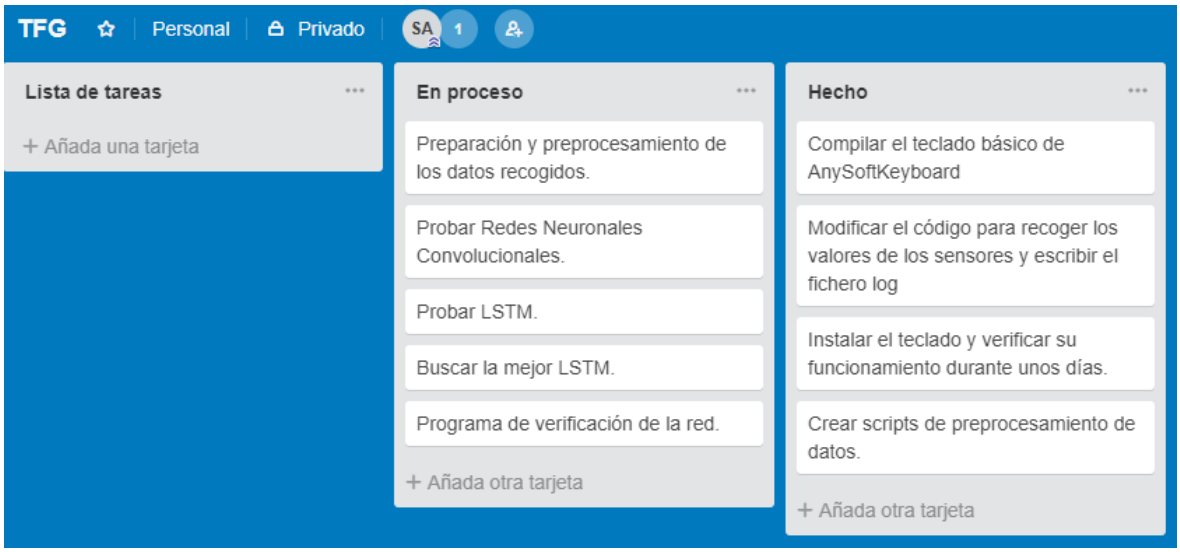

Imagen 5: Tablón de Trello

3.2 Diseño y requisitos

En la práctica, se deben superar varios obstáculos antes de que uno sea capaz de recoger los eventos táctiles que suceden en la pantalla de un dispositivo. Esto se debe a que el sistema operativo de los smartphones restringe los privilegios otorgados a las aplicaciones. En la mayoría de los casos, una aplicación no puede adquirir pulsaciones a menos que esté activa y reciba el foco en la pantalla. Esto hace que la recopilación de eventos táctiles sea muy difícil. Hoy en día, Android de Google e iOS de Apple dominan el mercado de los teléfonos inteligentes equipados con pantalla táctil, y en ambos sistemas operativos se restringe el acceso a sus funciones internas.

El objetivo principal de un keylogger es recopilar cada pulsación que tiene lugar en el teclado. Para hacerlo debe cumplir dos requisitos fundamentales: (a) obtener permisos de raíz para poder enganchar y anular los métodos internos del SO que son responsables de la detección y administración de los eventos táctiles, y (b) ejecutarlos cada vez que se escribe y recoger el comportamiento táctil del usuario.

La mayoría de los dispositivos con Android tienen sensores incorporados que miden el movimiento, la orientación y varias condiciones ambientales. Estos sensores son capaces de proporcionar datos en bruto con alta precisión, y son útiles si se desea monitorizar el movimiento o posicionamiento tridimensional del dispositivo, o si se desea monitorizar cambios en el entorno ambiental cercano al dispositivo. Por ejemplo, un juego

puede rastrear las lecturas del sensor de gravedad de un dispositivo para inferir gestos y movimientos complejos del usuario, como inclinación, movimiento, rotación o balanceo.

Pocos dispositivos con Android tienen todos los tipos de sensores. Por ejemplo, la mayoría de los dispositivos y tabletas tienen un acelerómetro y un magnetómetro, pero pocos dispositivos tienen barómetros o termómetros. Además, un dispositivo puede tener más de un sensor de un tipo determinado. Por ejemplo, un dispositivo puede tener dos sensores de gravedad, cada uno con un rango diferente.

En la Tabla 1 se pueden comprobar los distintos tipos de sensores de movimiento que puede tener un teléfono móvil, y si son sensores hardware o software, cuyos datos se obtienen aplicando ciertas operaciones sobre los datos de los sensores hardware.

Tipo	Descripción	Implementación
Acelerómetro	Fuerza de aceleración en tres direcciones: lateral (eje *x*), longitudinal (eje *y*) y vertical (eje *z*) (incluida la gravedad).	Hardware
Gravedad	Fuerza de aceleración en tres direcciones: lateral (eje *x*), longitudinal (eje *y*) y vertical (eje *z*) (incluida la gravedad).	Software
Aceleración linear	Fuerza de aceleración a lo largo de los ejes *x*, *y* y *z* (excluyendo la gravedad).	Software
Giroscopio	El cambio de orientación a lo largo de tres ángulos: inclinación (eje *x*), balanceo (eje *y*) y acimut (eje *z*).	Hardware
Orientación	Orientación alrededor de los ejes *x*, *y* y *z*	Software
Presión	Presión ejercida en la pantalla táctil.	Hardware

Tabla 1:Sensores

Con esta tecnología se diseña un teclado que recoge la información necesaria. Este teclado pedirá los permisos necesarios para acceder a los datos del hardware. Además, será la aplicación activa cuando se necesite obtener información.

Para el reconocimiento del usuario no se puede utilizar una Red Neuronal Convolucional, ya que es muy difícil reconocer a una persona por tan solo una pulsación. Es necesario que se reconozcan varias pulsaciones, es decir, una secuencia o serie temporal de pulsaciones y se analicen al mismo tiempo para dar una respuesta más acertada.

A continuación se muestra un diagrama con las distintas fases en las que se dividirá el trabajo:

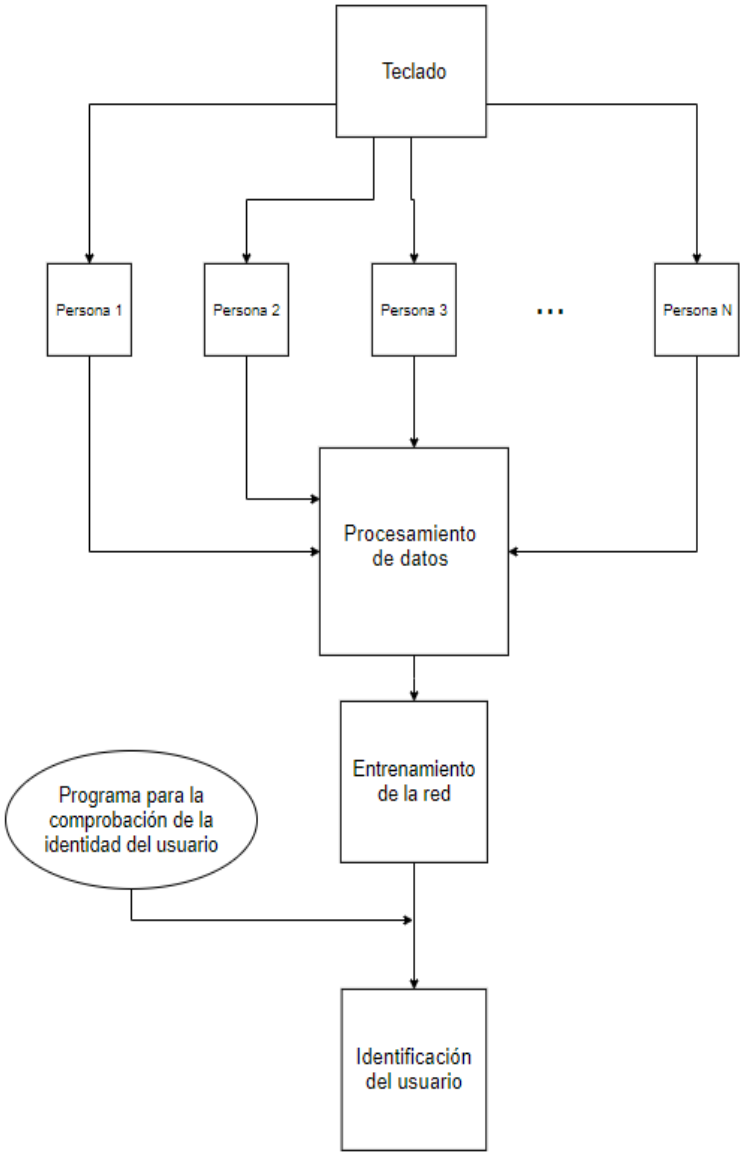

Imagen 6: Diagrama de fases del trabajo

Nuestros experimentos consisten en dos fases consecutivas. En la primera fase recolectamos datos de los voluntarios que aceptaron participar en el experimento de extraer las características de su dinámica de uso del teclado en la vida real.

En la segunda fase se ejecuta un algoritmo de selección de características y un clasificador para ordenar las características de acuerdo con su información contenida para identificar a cada persona.

3.3 Tecnologías

Como ya se ha comentado anteriormente, para la planificación del proyecto se ha utilizado Trello [52], un software gratuito de administración de proyectos, el cual permite llevar un seguimiento del estado de las tareas.

Para la implementación de la aplicación se ha utilizado el entorno de desarrollo Android Studio [37], entorno de desarrollo integrado oficial para la plataforma Android [41], utilizando el lenguaje de programación Java [40]. Se hizo uso del Android NDK [39] para poder compilar código escrito en C++ [53]. Para el control de versiones de la aplicación se utilizó Git [38]. El código base del que se ha partido es el de AnySoftKeyboard [42].

AnySoftKeyboard [42] es un teclado virtual de código libre. Tienes varios packs de idiomas disponibles y, al tener un repositorio abierto en Git [38], se puede modificar para adaptarlo al proyecto.

Por otro lado, los programas desarrollados para el preprocesamiento de datos están implementados con Python [43], junto con la librería pandas [44], una librería de código abierto con licencia de BSD que proporciona estructuras de datos y herramientas de análisis de datos de alto rendimiento y fáciles de usar. Esta librería permite cargar en memoria grandes cantidades de datos y procesarlos de una manera rápida y sencilla.

Para la implementación de la red neuronal se hizo uso de Keras [45], librería de código libre para Python. Es capaz de ejecutarse sobre TensorFlow, Microsoft Cognitive Toolkit o Theano. Diseñada para permitir la experimentación rápida con redes neuronales profundas. Keras contiene numerosas implementaciones de bloques de construcción de

redes neuronales de uso común, como capas, funciones de activación, optimizadores y un conjunto de herramientas para facilitar el trabajo con datos de imágenes y de texto.

En la siguiente figura se muestran los iconos de las tecnologías utilizadas:

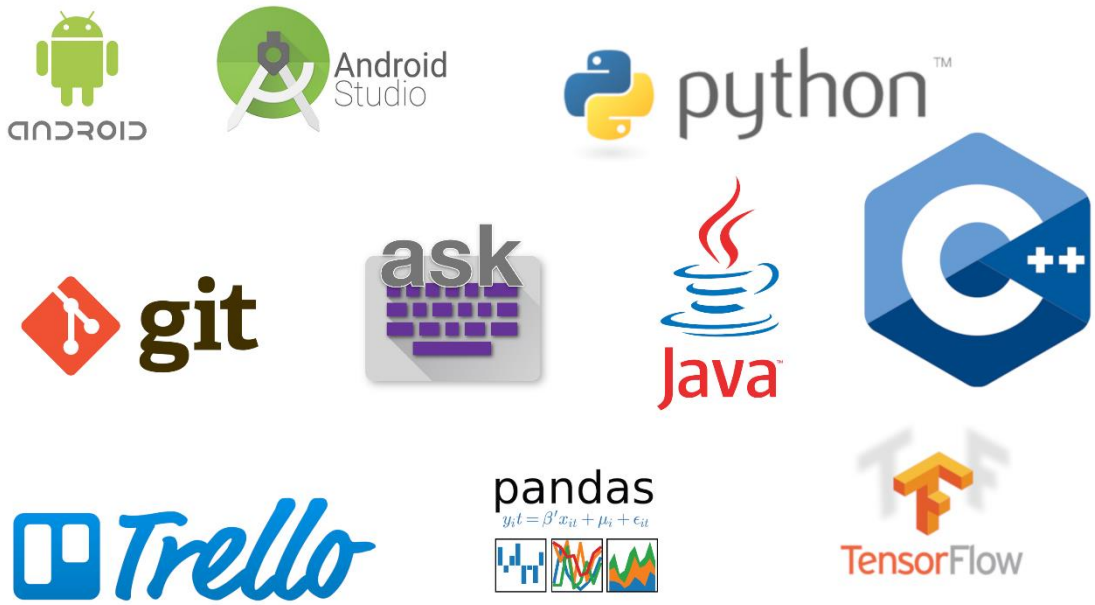

Imagen 7: Iconos de las tecnologías utilizadas

4. Implementación

4.1 Aplicación móvil

La aplicación implementada para la recogida de los datos es un teclado que almacena los datos recogidos por los distintos sensores del dispositivo móvil, como el giroscopio y el acelerómetro del dispositivo, en ficheros, dentro de la memoria externa del móvil.

Para ello se ha modificado el código fuente de AnySoftKeyboard, en concreto, se ha modificado el fichero "AnySoftKeyboard.java" para que cada vez que se pulse o se deje de pulsar sobre el teclado queden registrados todos los datos.

4.1.1 Sensores

En general, el framework del sensor utiliza un sistema de coordenadas de 3 ejes estándar para expresar valores de datos. Para la mayoría de los sensores, el sistema de coordenadas se define en relación con la pantalla del dispositivo cuando el dispositivo se mantiene en su orientación predeterminada (Imagen 8). Cuando un dispositivo se mantiene en su orientación predeterminada, el eje X es horizontal y apunta hacia la derecha, el eje Y es vertical y apunta hacia arriba, y el eje Z apunta hacia el exterior de la pantalla. En este sistema, las coordenadas detrás de la pantalla tienen valores Z negativos.

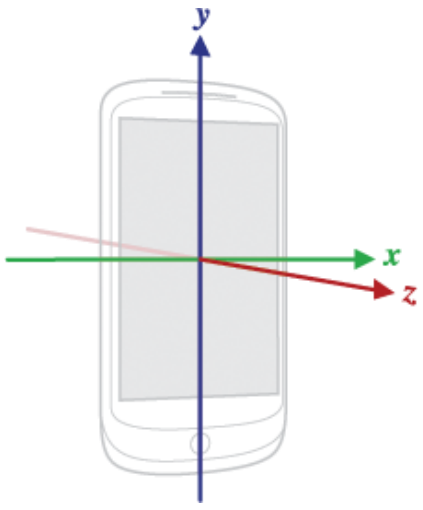

Imagen 8: Representación de ejes del móvil

El punto más importante a entender sobre este sistema de coordenadas es que los ejes no se intercambian cuando cambia la orientación de la pantalla del dispositivo; es decir, el sistema de coordenadas del sensor nunca cambia a medida que el dispositivo se mueve.

4.1.2 Funcionamiento

Al iniciar el teclado inicializamos los listeners (o funciones de escucha) de todos los sensores necesarios. En el siguiente código se puede ver como se realiza este proceso:

```
// Register sensors listeners
mSensorManager = (SensorManager)getSystemService( Context.SENSOR_SERVICE );

Sensor sensorAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
Sensor sensorGyroscope = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
Sensor sensorMagnetometer = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
Sensor sensorGravity = mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
Sensor sensorLinearAcceleration = mSensorManager.
        getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);

// SENSOR_DELAY_NORMAL < SENSOR_DELAY_UI < SENSOR_DELAY_GAME < SENSOR_DELAY_FASTEST
int sensorSpeed = SensorManager.SENSOR_DELAY_FASTEST;
mSensorManager.registerListener( listener: this, sensorAccelerometer, sensorSpeed);
mSensorManager.registerListener( listener: this, sensorGyroscope, sensorSpeed);
mSensorManager.registerListener( listener: this, sensorMagnetometer, sensorSpeed);
mSensorManager.registerListener( listener: this, sensorGravity, sensorSpeed);
mSensorManager.registerListener( listener: this, sensorLinearAcceleration, sensorSpeed);
```

Imagen 9: Listeners del teclado

Cada vez que detectamos una pulsación recogemos todos los datos necesarios, cuando detectamos que se ha dejado de pulsar volvemos a recogerlos. Es decir, por cada tecla pulsada se guardan dos registros, uno al pulsar y otro al levantar el dedo.

Los ficheros tienen la información de cada pulsación en una fila del fichero distinta, separando los datos mediante comas. A continuación se puede ver una parte del código utilizado para generar el fichero de log:

```
// Generate labelled log
return me.getPointerCount()
        +","+ me.getPressure()
        +","+ me.getSize()
        +","+ strLogGinger      // orientation, touchMajor, touchMinor

        +","+ mValuesAccelerometer[0] // x
        +","+ mValuesAccelerometer[1] // y
        +","+ mValuesAccelerometer[2] // z

        +","+ mValuesGravity[0]
        +","+ mValuesGravity[1]
        +","+ mValuesGravity[2]

        +","+ mValuesLinearAcceleration[0]
        +","+ mValuesLinearAcceleration[1]
        +","+ mValuesLinearAcceleration[2]

        +","+ mValuesGyroscope[0]
        +","+ mValuesGyroscope[1]
        +","+ mValuesGyroscope[2]

        +","+ mValuesOrientation[0]  // Azimuth
        +","+ mValuesOrientation[1]  // Pitch
        +","+ mValuesOrientation[2]  // Roll
        ;
```

Imagen 10: Escritura del fichero log

Los datos se escriben en un fichero distinto cada día para trabajar con archivos de menor volumen. Dado que estos ficheros guardan mucha información si tamaño tiende a crecer rápidamente.

Cuando se abre el teclado aparece una barra roja en la parte inferior. Esta barra es un indicador visual para avisarnos de que el teclado está guardando datos. Si se aprieta la tecla de mayúscula tres veces seguidas rápidamente se activará/desactivará el guardado de datos y además desaparecerá o reaparecerá la barra roja para que se pueda ver visualmente si está activo o no.

Por defecto al abrir el teclado se activará el log (con la barra roja), es decir, si se desactiva, la próxima vez que se vuelva a abrir el teclado se volverá a activar.

En la siguiente figura se puede ver el aspecto del teclado, incluyendo la barra roja en la primera figura y como desaparece al desactivar el log.

Imagen 11: Teclado con el registro de datos activado.

Imagen 12: Teclado con el registro de datos desactivado.

4.1.3 Descripción del fichero log

El fichero de log es un CSV que almacena los campos indicados en la Tabla 2 para cada pulsación:

Nombre del campo	**Descripción**
keyCode	Código de la tecla, de momento guarda todas las teclas.
timestampDown	Timestamp del momento de la pulsación inicial.
duration	Duración de la pulsación en milisegundos, con este campo y el

	anterior tenemos todo lo que necesitamos.
LOG_DOWN	Características de los sensores en el momento inicial de la pulsación.
LOG_UP	Características de los sensores en el momento final de la pulsación.

Tabla 2: Campos del fichero log

Ejemplo de un fichero de log:

```
key, time_down, duration, sensors_down[], sensors_up[]
key, time_down, duration, sensors_down[], sensors_up[]
key, time_down, duration, sensors_down[], sensors_up[]
```

Los sensores almacenados en LOG_DOWN y LOG_UP, al pulsar y levantar el dedo, son los mostrados en la Tabla 3 (listados en el mismo orden en el que se guardan):

Nombre del campo	**Descripción**
Número de dedos en la pantalla	El número de pulsaciones activas en la pantalla en el momento de una pulsación nueva.
Touch pressure	La presión es un float de 0 a 1. Depende del hardware. Incluso si hay un sensor de presión de hardware, el mapa entre la fuerza real y el valor de retorno de getPressure() aún depende del dispositivo. Además, algunos dispositivos sin un sensor de presión hardware simulan la presión usando el tamaño del punto de contacto, es decir, suponen que una mayor presión significa que su dedo se aplana. Otros dispositivos simplemente devuelven un valor constante.

	Se pueden generar valores superiores a 1 en función de la calibración del dispositivo de entrada.
Touch size	Devuelve un valor escalado del tamaño aproximado para el índice de puntero dado. Esto representa una aproximación del área de la pantalla que se está presionando; el valor real en píxeles correspondiente al toque se normaliza con el rango de valores específico del dispositivo y se escala a un valor entre 0 y 1. El valor del tamaño se puede utilizar para determinar el grosor los eventos táctiles.
Touch orientation	Devuelve la orientación del área táctil en radianes en sentido horario desde la vertical para el índice del puntero dado. Un ángulo de 0 radianes indica que el eje principal de contacto está orientado hacia arriba, es perfectamente circular o tiene una orientación desconocida. Un ángulo positivo indica que el eje principal de contacto está orientado a la derecha. Un ángulo negativo indica que el eje principal de contacto está orientado a la izquierda. El rango completo es de -PI / 2 radianes (el dedo apunta completamente a la izquierda) a PI / 2 radianes (el dedo apunta completamente a la derecha).
Touch major axis	Devuelve la longitud del eje principal de una elipse que describe el área táctil en el punto de contacto para el índice de puntero dado.
Touch minor axis	Devuelve la longitud del eje menor de una elipse que describe el área táctil en el punto de contacto para el índice de puntero dado.
Accelerometer *[x,y,z]*	Mide la fuerza de aceleración en m/s^2 que se aplica a un dispositivo en los tres ejes físicos (*x*, *y* y *z*), incluida la fuerza

	de la gravedad.
Gravity *[x,y,z]*	Mide la fuerza de la gravedad en m/s^2que se aplica a un dispositivo en los tres ejes físicos (*x*, *y*, *z*).
Linear acceleration *[x,y,z]*	Mide la fuerza de aceleración en m/s^2 que se aplica a un dispositivo en los tres ejes físicos (*x*, *y*, *z*), excluyendo la fuerza de la gravedad.
Gyroscope *[x,y,z]*	Mide la tasa de rotación de un dispositivo en rad / s alrededor de cada uno de los tres ejes físicos (*x*, *y*, *z*).
Orientation *[azimuth,pitch,roll]*	Mide los grados de rotación que hace un dispositivo alrededor de los tres ejes físicos (*x*, *y*, *z*).

Tabla 3: Características del fichero log

4.2 Preprocesamiento de datos

Una vez se cuenta con suficientes archivos de registro de información de los diversos usuarios, se procede al tratamiento de la información para prepararlos para el entrenamiento de la red neuronal. Los datos son anonimizados desde el primer instante, asignando un código numérico distinto y aleatorio a cada uno de los usuarios.

Cada uno de los registros obtenidos es procesado mediante un script que añadirá líneas al fichero que se utilizará para el entrenamiento de la red neuronal. Conociendo el número de identificación del usuario, se realiza una comprobación estructural línea a línea. Si dicha línea contiene todos los atributos de interés, se considera completa, se le agrega una última columna con el código de identificación y se añade al fichero de entrenamiento. Para realizar esta operación se ha utilizado el siguiente código:

```
posicion = fichero.find("/Datos/")
file = open('/home/sergio/Downloads/DatosCorregidos/DatosCompletos.txt', 'a')
with open(fichero) as fp:
    line = fp.readline()
    while line:
        if line.count(',') == 44:
            file.write(line.rstrip()+","+fichero[posicion+7]+"\n")
            lineas=lineas+1
        line = fp.readline()
file.close()
```

Imagen 13: Creación del fichero de entrenamiento

Una vez todos los logs han sido leídos y se han clasificado aquellos que son válidos con su código de usuario, se adapta la información del fichero para facilitar su tratamiento. Para manejar todos los datos se ha utilizado la herramienta Pandas.

Para tipificar el tiempo de pulsación entre teclas se adapta la información de la columna timeStampDown. Esta categoría, en base de tiempo Unix [47] con precisión de milisegundos, registra el instante en el que cada tecla es pulsada.

Para obtener la duración de la pulsación entre teclas, se recorren todas las líneas del fichero (cada una corresponde a una pulsación) de modo que el nuevo valor relativo de la primera pulsación es 0 y la posición siguiente se obtiene de restar la pulsación anterior a la pulsación deseada.

Del mismo modo, si existe un intervalo entre pulsaciones superior a 10 segundos se reinicia la contabilización, volviendo a comenzar la primera pulsación de la nueva serie con un timeStampDown de 0.

Con esta caracterización, con valores relativos, resulta sencillo reconocer el inicio de una secuencia de escritura, puesto que la primera pulsación contará con un timeStampDown de 0 y la última será la inmediatamente anterior al próximo timeStamp de valor nulo.

A continuación se puede ver el código utilizado para realizar este proceso:

```
# Calculamos lo que tarda entre teclas
df['timestampDown'] = df['timestampDown'].astype('int64') - df['timestampDown'].astype
('int64').shift()
df['timestampDown'].fillna(0.0, inplace=True)
df['timestampDown'] = df['timestampDown'].mask(df['timestampDown'] >10000, 0)
```

Imagen 14: Cálculo del intervalo entre pulsaciones.

Las etiquetas touchMajorAxisDown y touchMinorAxisDown para el inicio de la pulsación y touchMajorAxisUp y touchMinorAxisUp para el fin de pulsación respectivamente, indican las distancias respecto a sus ejes al centro de la elipse cuando comienza y cuando termina la pulsación. En la siguiente figura se puede ver un esquema de estos ejes:

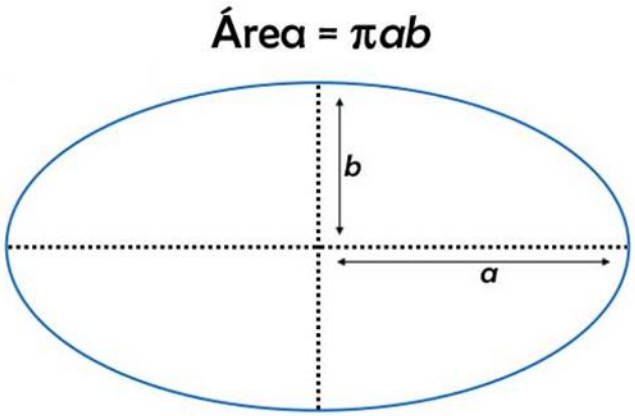

Imagen 15: Área de una elipse

Combinando ambas etiquetas de eje para el inicio de pulsación y las dos correspondientes para el final de la pulsación, se calcula el área de influencia de la pulsación. De este modo se combinan parámetros y se reducen el número de características que deben ser analizadas, optimizando el tiempo de computación.

Para realizar este proceso se utilizó el siguiente código:

```
# Calculamos el area aproximada del toque
df['areaDown'] = df.touchMajorAxisDown.astype('float64') * df.touchMinorAxisDown.astype
('float64') * math.pi
df['areaUp'] = df.touchMajorAxisUp.astype('float64') * df.touchMinorAxisUp.astype('float64')
* math.pi
```

Imagen 16: Cálculo del área de la elipse

Del mismo modo se eliminan aquellas columnas cuya información se considera irrelevante o poco discriminatoria. Por ejemplo, se elimina la característica de orientación cuyos valores representados eran booleanos.

Por último y para cumplir con los parámetros de entrenamiento de la red, se ajustan la cantidad de líneas del registro. Se comprueba que, para cada uno de los distintos identificadores de usuario, exista un número de líneas múltiplo de 30. Esto se debe a que la red neuronal evaluará las líneas del registro en lotes de 30.

4.3 Implementación de la red

Las redes neuronales recurrentes con memoria larga de corto plazo (LSTM) han surgido como un modelo eficaz y escalable para varios problemas de aprendizaje relacionados con datos secuenciales. Los métodos anteriores tienen problemas para trabajar con dependencias de larga duración. Las LSTM, por otro lado, son generales y efectivas en la captura de dependencias temporales a largo plazo. No sufren los obstáculos de optimización que plagan las redes recurrentes simples (SRN) y se han utilizado para avanzar en el estado de la técnica para muchos problemas difíciles. Esto incluye reconocimiento de escritura a mano y generación, entre otros. Una capa LSTM se compone de una célula, una puerta de entrada, una puerta de salida y una puerta de olvido. La célula registra valores relacionados con periodos de tiempo arbitrarios y las tres puertas son las encargadas de regular el flujo de información que entra y sale de la celda. En la siguiente figura se puede ver un esquema de este tipo de neuronas:

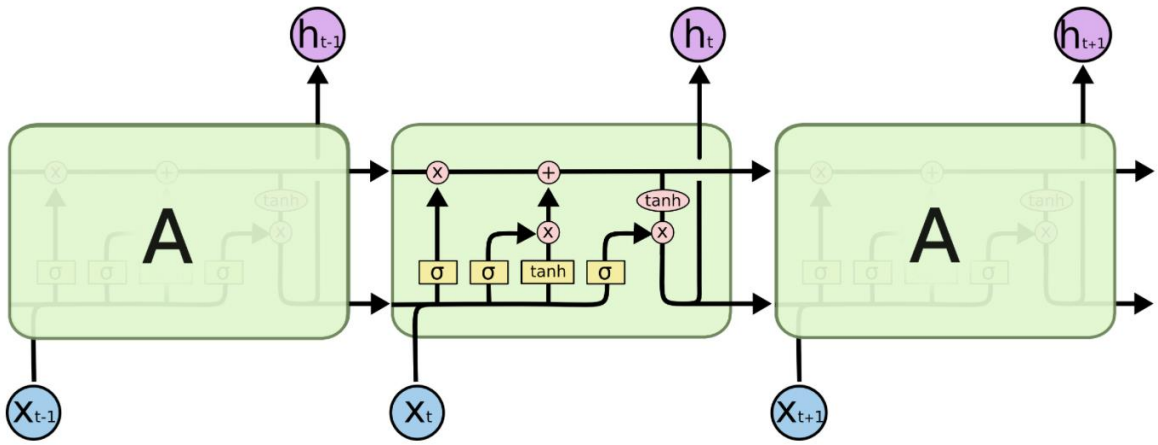

Imagen 17: Esquema de una neurona LSTM

En este proyecto, la red ha sido configurada de modo que consta de tres capas. La red neuronal recurrente empleada utiliza dos capas LSTM con 64 y 16 neuronas, respectivamente, con un dropout del 0'2 cada una; y una capa profunda con 5 neuronas de salida. Las capas LSTM son las encargadas de clasificar y analizar la información recibida y la capa profunda se encarga de evaluar los inputs y mostrar la salida final de la red neuronal. El dropout es una técnica de regularización para reducir el overfitting de una red neuronal al evitar adaptaciones complejas a datos de entrenamiento [49].

A continuación se muestran dos imágenes de la estructura de la red., en la primera se puede ver un esquema de la misma, y en la segunda imagen se muestra el resumen que muestra la librería Keras después de construir la red, en la que se incluyen el número de parámetros que contiene cada capa.

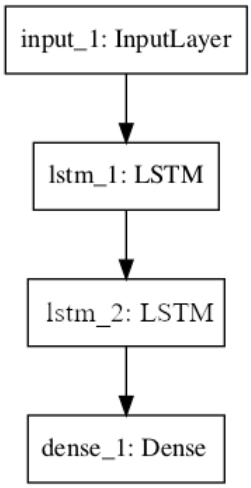

Imagen 18: Esquema de la red

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 30, 64)            27136
_________________________________________________________________
lstm_2 (LSTM)                (None, 16)                5184
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 85
=================================================================
Total params: 32,405
Trainable params: 32,405
Non-trainable params: 0
_________________________________________________________________
None
```

Imagen 19: Estructura de la red

El código utilizado para la implementación de esta red junto con el Cross Validation en Keras es el siguiente:

```
for x in range(len(Y)-1, -1, -1):
    if x % tamGrupos != 0:
        Y = numpy.delete(Y, (x), axis=0)

# 10-fold cross validation
aux=0
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
cvscores=[]
for train, test in kfold.split(X,Y):
    # create the model
    dropout1=0.2
    model = Sequential()
    model.add(LSTM(64), input_shape=(tamGrupos, len(X[train][0][0])), dropout=dropout1,
recurrent_dropout=dropout1, return_sequences=True))
    model.add(LSTM(16), dropout=dropout1, recurrent_dropout=dropout1, return_sequences=False))
    model.add(Dense(output_dim=5, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy', optimizer='rmsprop', metrics=
['accuracy'])
    print(model.summary())
    history = model.fit(X[train], Y[train], epochs=110, batch_size=30, validation_split=0.1,
shuffle=True)
    try:
        os.remove('model_json')
    except OSError:
        pass
    model.save_weights('model_weights')
    json_string = model.to_json()
    text_file = open("model_json", "w")
    text_file.write(json_string)
    text_file.close()
    # Final evaluation of the model
    scores = model.evaluate(X[test], Y[test], verbose=1)
    print("Accuracy: %.2f%%" % (scores[1]*100))
    cvscores.append(scores[1] * 100)
resultado=("%.2f%% (+/- %.2f%%)" % (numpy.mean(cvscores), numpy.std(cvscores)))
print(resultado)
```

Imagen 20: Implementación del Cross Validation

El código fuente de este proyecto está disponible en la siguiente dirección https://bitbucket.org/Sergio_Alcaraz/verificacion-biometrica-de-usuarios/src

Este repositorio es privado, para solicitar acceso al mismo contactar con el autor.

5. Evaluación

La prueba de un clasificador tradicionalmente implica la división del conjunto de datos en dos partes: uno utilizado para entrenamiento y uno utilizado para evaluación. Sin embargo, la principal desventaja de este método, denominado holdout, es que los datos utilizados para la evaluación nunca se utilizan para entrenar al clasificador y viceversa. Por lo tanto, conduce a una evaluación del desempeño menos general. Se han desarrollado diferentes técnicas para abordar este problema. En este proyecto se utiliza una solución popular denominada validación cruzada K-Fold para evaluar la calidad de los diferentes modelos de clasificación. Este método primero consiste en mezclar el conjunto de datos y dividirlo en k particiones de aproximadamente el mismo tamaño. El clasificador se entrena con k - 1 conjuntos de datos y se evalúa en la partición restante. Este proceso se repite k veces al seleccionar un conjunto de entrenamiento y un conjunto de evaluación diferentes, de manera que cada partición se utiliza a lo sumo una vez para la evaluación y k - 1 veces para el entrenamiento. Los resultados de la evaluación finalmente se promedian para representar el rendimiento global del clasificador. Por lo tanto, todos los datos se utilizan tanto para el entrenamiento como para la evaluación a fin de proporcionar una evaluación del desempeño más general y precisa [51].

Los resultados se miden usando K-Fold Cross-Validation con k = 5 entrenando cada clasificador durante 100 épocas. Los resultados se promedian en todas las combinaciones.

La cantidad de muestras obtenida para cada uno de los 5 voluntarios fue distinta. Para igualar la cantidad de muestras con las que se entrena la red se utiliza como máximo la cantidad de muestras del voluntario que menos muestras ha aportado, que fueron 51390 muestras.

Para empezar se comprueba el resultado con redes convolucionales, obteniendo resultados como máximo del 24’35% de precisión, utilizando 45000 muestras de cada uno. Estas redes, por la naturaleza del problema, no son las más idóneas para este tipo de problema, ya que se necesita analizar una secuencia de datos correlativos en el tiempo, no solo una muestra.

La siguiente opción es analizar las redes neuronales LSTM. Primero se hacen pruebas con 15000 datos de cada uno de los voluntarios y utilizando 100 épocas. Los resultados son notablemente superiores a los obtenidos con anterioridad.

La Tabla 4 muestra los resultados obtenidos en el entrenamiento de una red con una capa LSTM y con las siguientes combinaciones de neuronas y datos utilizados en cada entrenamiento.

Neuronas\Datos	**15000**	**21000**	**30000**	**51000**
20	82’16% (+/- 1’96%)	84’23% (+/- 4’35%)	85’14% (+/- 4’66%)	88’76% (+/- 3’36%)
32	86’20% (+/- 2’95%)	85’91% (+/- 5’76%)	91’56% (+/- 2’83%)	91’35% (+/- 1’58%)
64	88’92% (+/- 2’68%)	91’29% (+/- 1’37%)	90’90% (+/- 2’12%)	92’22% (+/- 2’16%)

Tabla 4: Resultados de los entrenamientos

La red alcanza un porcentaje de aciertos de un 92’22% (+/-2’16) en una de las combinaciones. Como se puede observar, funciona mejor con más neuronas.

En la siguiente prueba se aumentó el número de neuronas y se comprobó el resultado obtenido al ir aumentando el número de épocas de entrenamiento. En la siguiente figura se puede ver el resultado de este experimento:

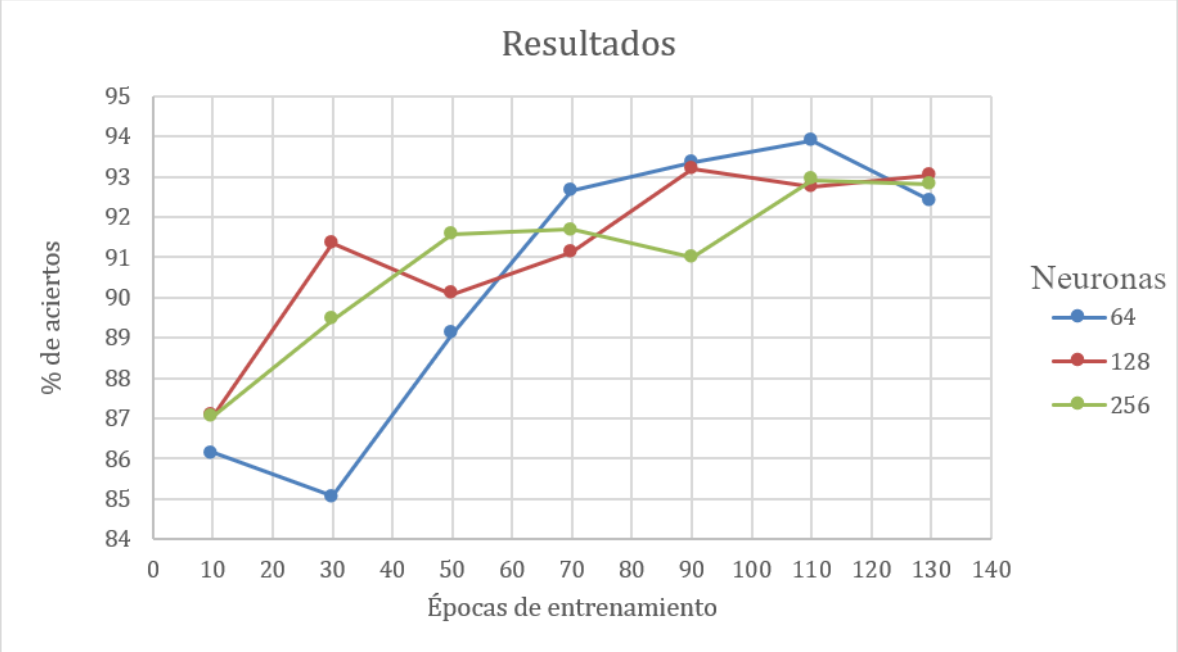

Imagen 21: Resultados del entrenamiento con más neuronas y al variar el número de épocas.

A partir de la gráfica anterior, se puede apreciar que el mejor resultado es de 93,89% con 64 neuronas y 110 épocas de entrenamiento.

A continuación se añade una capa LSTM más a la red, para la cual se prueban los valores de 16, 32 y 64 neuronas, con 90 y 110 épocas de entrenamiento, ya que son los valores que mejores resultados han obtenido en las pruebas anteriores. En la siguiente gráfica se pueden ver los resultados de este nuevo experimento:

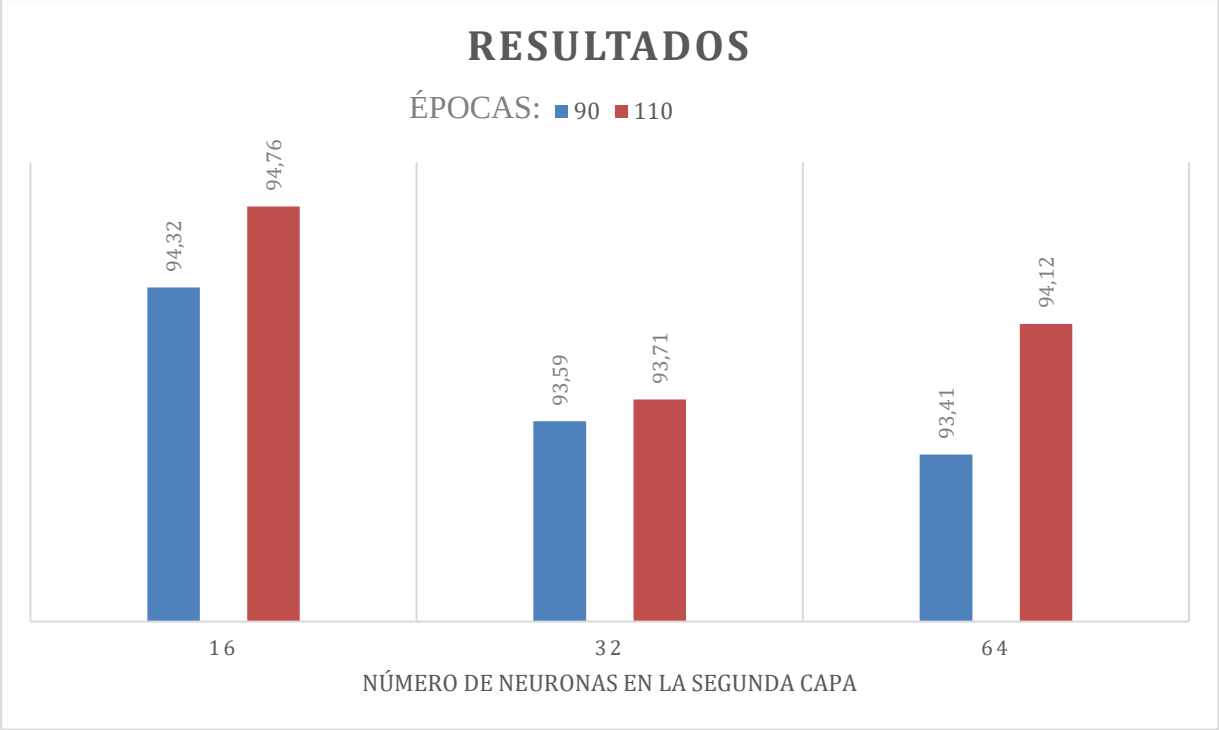

Imagen 22: Resultados con dos capas LSTM

El valor que mejores resultados obtiene es la capa con 16 neuronas, por lo que probamos con menos neuronas en esta capa para comprobar el comportamiento de la red. Los resultados para la red con una segunda capa LSTM con 8 neuronas fueron de 92’78% (+/- 2’21%) y 91’76% (+/- 1,63), para 90 y 110 épocas, respectivamente.

Finalmente, se realizaron más pruebas con la configuración de red que mejores resultados había obtenido variando la cantidad de épocas.

Imagen 23: LSTM con 16 neuronas en la segunda capa

La red seleccionada para la comprobación de los resultados es la de dos capas LSTM; con 64 neuronas en la primera capa y 16 en la segunda, 51000 datos y 110 épocas, ya que es la que mejor resultados había obtenido: 94'76% (+/- 0'77%). Como se ve en la gráfica anterior al variar el número de épocas no mejora este resultado, quedándonos finalmente con 110 épocas.

Por último, se crea un programa al cual se le pasa un archivo con datos de haber escrito una frase. Este archivo no se debe de haber utilizado durante el entrenamiento. La red neuronal ya entrenada los analiza y define qué persona es la que lo ha escrito. En los siguientes ejemplos se observa que reconoce a las personas 1 y 3.

```
sergio@sergio:~/Downloads/DatosCorregidos$ python TestRed.py 1/datos1800.txt
Using TensorFlow backend.
1/1 [==============================] - 2s 2s/step
['0.79947', '0.19805', '0.00094', '0.00080', '0.00073']
Es la persona numero 1
sergio@sergio:~/Downloads/DatosCorregidos$ python TestRed.py 3/datos3875.txt
Using TensorFlow backend
1/1 [==============================] - 1s 1s/step
['0.00065', '0.00067', '0.56093', '0.43724', '0.00050']
Es la persona numero 3
```

Imagen 24: Validación del funcionamiento de la red

6. Conclusiones

En este proyecto se propone el uso de redes neuronales de LSTM como método de seguridad adicional a los métodos más convencionales de contraseña o patrones.

El sistema desarrollado en este trabajo puede predecir el usuario que está utilizando un teclado con una precisión del 94’76% (+/- 0’77%). Nuestro clasificador puede funcionar exitosamente cuando se confronta con datos sin procesar. Demostrando así que las redes neuronales profundas son capaces de evitar con facilidad los ataques mediante el teclado, basándose en sensores de movimiento y eliminando la necesidad de procesamiento previo de los datos o de estrategias de extracción de características manuales.

Este clasificador cuenta con la ventaja de poder ser ejecutado en tiempo real y desde nuestro dispositivo móvil. Con tan solo implementar una aplicación que ejecute comprobaciones en la red, el propietario del dispositivo estará seguro de ser la única persona que escribe a través de su terminal.

6.1 Métodos alternativos de resolución

Otro método por el que se podría haber llevado a cabo la resolución de este problema es mediante el empleo de otro tipo Redes Neuronales Recurrentes (RNN) [50].

Sin embargo, puede ser difícil entrenar RNN estándar para resolver problemas que requieren el aprendizaje de dependencias temporales a largo plazo. Esto se debe a que el gradiente de la función de pérdida decae exponencialmente con el tiempo (llamado el problema del gradiente de fuga). Las redes LSTM son un tipo de RNN que usa unidades especiales además de las unidades estándar. Las unidades LSTM pueden aprender las dependencias a más largo plazo, por lo que es una mejor elección en este problema.

6.2 Trabajo futuro

Como trabajo a realizar en el futuro se propone un programa que ejecute una red LSTM preentrenada para distinguir el usuario habitual del smartphone de cualquier otra persona. Este programa analizaría los patrones de escritura y podría bloquear el móvil en caso de que detectara que la persona utilizando el teléfono no es su propietario.

Con el fin de evaluar la amenaza a la seguridad en contextos específicos, sería interesante realizar más experimentos utilizando hardware estándar como los teclados actuales de los cajeros automáticos, los teclados electrónicos de los sistemas de acceso a edificios o los teclados seguros para habitaciones de hotel, junto con un dispositivo portable capaz de recopilar información sobre el movimiento de la mano, como un reloj inteligente. Los movimientos del usuario pueden crear interferencias de movimiento importantes que potencialmente pueden ofuscar los patrones de señal tecleado. Se podrían llevar a cabo experimentos en tales condiciones para comparar los resultados y la probabilidad de tal ataque en estos contextos.

7. Referencias

[1] Liang Cai and Hao Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. *In HotSec*, 2011.

[2] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. Accessory: password inference using accelerometers on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, page 9. ACM, 2012.

[3] Zhi Xu, Kun Bai, and Sencun Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pages 113–124. ACM, 2012.

[4] Aviv AJ, Gibson K, Mossop E, Blaze M, Smith JM. Smudge attacks on smartphone touch screens. In: *Proceedings of WOOT* 2010.

[5] Ahmed Al-Haiqi, Mahamod Ismail, and Rosdiadee Nordin. On the best sensor for keystrokes inference attack on android. *Procedia Technology*, 11:989–995, 2013.

[6] Apple Inc. Motion events.

`https://developer.apple.com/documentation/coremotion`, Online, accessed 23-06-2018.

[7] Google Inc. Motion sensors.

`https://developer.android.com/guide/topics/sensors/sensors_motion`, Online, accessed 23-06-2018.

[8] Google Inc. Sensor types.

`https://source.android.com/devices/sensors/sensor-types`, Online, accessed 23-06-2018.

[9] Hochreiter, Sepp; Schmidhuber, Jürgen (1997-11-01). "Long Short-Term Memory". Neural Computation. 9 (8): 1735–1780.

[10] Auli, Michael, Galley, Michel, Quirk, Chris, and Zweig, Geoffrey. Joint language and translation modeling with recurrent neural networks. In *Empirical Methods in Natural Language Proessing (EMNPL)*, volume 3, 2013.

[11] Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc VV. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS) 27*, pp. 3104–3112, 2014.

[12] Vinyals, Oriol, Toshev, Alexander, Bengio, Samy, and Erhan, Dumitru. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3156–3164, June 2015.

[14] Karpathy, Andrej and Fei-Fei, Li. Deep visual-semantic alignments for generating image descriptions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3128–3137, June 2015.

[13] Liwicki, Marcus, Graves, Alex, Bunke, Horst, and Schmidhuber, Jurgen. A novel approach to ¨ on-line handwriting recognition based on bidirectional long short-term memory networks. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition*, volume 1, pp. 367–371, 2007.

[14] Graves, Alex, Liwicki, Marcus, Fernandez, Santiago, Bertolami, Roman, Bunke, Horst, and ´ Schmidhuber, Jurgen. A novel connectionist system for unconstrained handwriting recognition. ¨ *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009.

[15] Pollastri, Gianluca, Przybylski, Darisz, Rost, Burkhard, and Baldi, Pierre. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 47(2):228–235, 2002.

[16] Vohradsky, Jirí. Neural network model of gene expression. *The FASEB Journal*, 15(3):846–854, 2001.

[17] Xu, Rui, Wunsch II, Donald, and Frank, Ronald. Inference of genetic regulatory networks with recurrent neural network models using particle swarm optimization. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 4(4):681–692, 2007.

[18] Securityaffairs.co. Pierluigi Paganini, ‘Mandrake, NSA identifies users based on how they type on their devices’, 2015. [Onlone]. Avaliable: `https://securityaffairs.co/wordpress/37278/intelligence/mandrake-nsa-swipe-tech.html`. [Accessed: 31-Jan-2018].

[19] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444

[20] Bao L., Intille S.S, "Activity recognition from user annotated acceleration data," in *Proc. 2nd Int. Conf. on Pervasive Computing*, 2004.

[21] Kwapisz, J., Weiss, G., & Moore, S. (2010). *Activity recognition using cell phone accelerometers*. SIGKDD Explorations, 12(2), 74–82

[22] Duffner, S., Berlemont, S., Lefebvre, G., & Garcia, C. (2014). 3D gesture classification with convolutional neural networks. In *Proceedings of international conference on acoustic, speech, and signal processing* (ICASSP) (pp. 5432–5436).

[23] Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In USENIX *security symposium*, pages 1–16, 2009.

[24] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *Proceedings of the IEEE Symposium on Security and Privacy*, page 3. IEEE, 2004.

[25] Li Zhuang, Feng Zhou, and J Doug Tygar. Keyboard acoustic emanations revisited. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):3, 2009.

[26] Yigael Berger, Avishai Wool, and Arie Yeredor. Dictionary attacks using keyboard acoustic emanations. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 245–254. ACM, 2006.

[27] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security Symposium, volume 2001*, 2001.

[28] Denis Foo Kune and Yongdae Kim. Timing attacks on pin input devices. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 678–680. ACM, 2010.

[29] Roman Schlegel, Kehuan Zhang, Xiao-yong Zhou, Mehool Intwala, Apu Kapadia, and XiaoFeng Wang. Soundcomber: A stealthy and contextaware sound trojan for smartphones. In *NDSS*, volume 11, pages 17–33, 2011.

[30] Liang Cai and Hao Chen. *On the practicality of motion based keystroke inference attack*. Springer, 2012.

[31] Adam J Aviv, Benjamin Sapp, Matt Blaze, and Jonathan M Smith. Practicality of accelerometer side channels on smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 41–50. ACM, 2012.

[32] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L Littman. Activity recognition from accelerometer data. In AAAI, volume 5, pages 1541–1546, 2005.

[33] Timo Pylv¨an¨ainen. Accelerometer based gesture recognition using continuous hmms. In *Pattern Recognition and Image Analysis*, pages 639– 646. Springer, 2005.

[34] Muhammad Shoaib, Stephan Bosch, Ozlem Durmaz Incel, Hans Scholten, and Paul JM Havinga. Fusion of smartphone motion sensors for physical activity recognition. *Sensors*, 14(6):10146–10176, 2014.

[35] Christopher M Bishop. *Pattern recognition and machine learning. springer*, 2006.

[36] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[37] Android Studio, herramienta de desarrollo. https://es.wikipedia.org/wiki/Android_Studio (Consultado el 20-jun-2018) https://developer.android.com/studio/ (Consultado el 20-jun-2018).

[38] Git, Sistema de Control de Versiones. http://es.wikipedia.org/wiki/Git (Consultado el 10-may-2018) http://git-scm.com/ (Consultado el 10-may-2018).

[39] Android NDK https://developer.android.com/ndk/ (Consultado el 20-jun-2018).

[40] Java, Lenguaje de programación, https://www.java.com/es/download/ (Consultado el 20-jun-2018).

[41] Android, Sistema Operativo basado en el núcleo Linux, https://es.wikipedia.org/wiki/Android (Consultado 20-Jun-2018).

[42] AnySoftKeyboard, teclado virtual de código libre, https://github.com/AnySoftKeyboard (Consultado 20-Nov-2018).

[43] Pyhton, lenguaje de programación https://www.python.org/ (Consultado 20-Nov-2018).

[44] Pandas, librería de código abierto para procesamiento de datos, https://pandas.pydata.org/ (Consultado 20-Nov-2018).

[45] Keras, API para crear redes neuronales, https://keras.io/ (Consultado 20-Nov-2018).

[46] Charissa Ann Ronao, *Expert Systems With Applications*, 2016

[47] Tiempo Unix, ,sistema para la descripción de instantes de tiempo, https://es.wikipedia.org/wiki/Tiempo_Unix (Consultado 01-09-2018)

[48] Kanban, metodología ágil de desarrollo software, https://es.wikipedia.org/wiki/Kanban_(desarrollo) (Consultado 1-09-2018)

[49] Hinton, Geoffrey E.; Srivastava, Nitish; Krizhevsky, Alex; Sutskever, Ilya; Salakhutdinov, Ruslan R. (2012). "Improving neural networks by preventing co-adaptation of feature detectors".

[50] G. Kechriotis and E. S. Manolakos, "Training fully recurrent neural networks with complex weights," in *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 41, no. 3, pp. 235-238, March 1994.

[51] Validación cruzada (Cross Validation), https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada (Consultado 1-09-2018)

[52] Trello, software de administración de proyectos, https://es.wikipedia.org/wiki/Trello (Consultado el 20-jun-2018). https://trello.com/ (Consultado el 20-jun-2018).

[53] C++, lenguaje de programación. https://es.wikipedia.org/wiki/C%2B%2B (Consultado el 20-jun-2018).