RECONOCIMIENTO DE ACTIVIDADES HUMANAS UTILIZANDO REDES NEURONALES

Mario Garrido

Universidad de Chile
Santiago, Chile

1. INTRODUCCIÓN

Hoy en día la presencia de sensores en dispositivos utilizados masivamente (e.g: teléfonos celulares) permite la obtención de una cantidad considerable de datos que pueden ser utilizados para inferir la actividad que está realizando la persona que porta el dispositivo. En el contexto del reconocimiento de actividades humanas (o HAR, por su sigla en inglés) es de particular interés el clasificar, con el menor costo computacional posible y con la menor latencia posible, la actividad que está realizando la persona a la que corresponden las mediciones disponibles.

Para realizar una clasificación no es necesario, ciertamente, utilizar las series de tiempo de forma directa. Los métodos basados en distancias han demostrado resultados aceptables, gracias a que se ha trabajado mucho en el campo de la construcción manual ad hoc de features, con el fin de simplificar el problema para los clasificadores, comprimiendo la información clave de las series de tiempo en una representación más compacta, pero esto requiere tener un entendimiento, al menos rudimentario, del campo sobre el que se está trabajando, para tener una intuición sobre los features que producen ruido y los que son útiles, o en su defecto determinarlo a través de fuerza bruta.

En el presente trabajo se propone una arquitectura de red neuronal que aprende a escoger los features de forma automática, tomando como entrada una representación espectral de las series de tiempo originales, y produce una clasificación en base a los features generados.

Para poner a prueba la arquitectura generada se utiliza el dataset UCF-iPhone Dataset[1], del Center for Research in Computer Vision, University of Central Florida.

2. DATASET

El dataset se compone de 383 muestras de series de tiempo multivariadas, que cuentan de 9 canales correspondientes a 3 canales de acelerómetro, 3 canales de giroscopio y 3 canales de magnetómetro. Vale la pena notar que en la página del dataset[2] se indica que los datos capturados con el magnetómetro no son útiles. Pese a esto, la arquitectura desarrollada se beneficia de la presencia de los canales correspondientes al magnetómetro, pero esta discusión se deja para maś adelante (En la Fig. 1 y la Fig. 2 se aprecia que los canales correspondientes al magnetómetro presentan una diferencia substancial entre actividades). La frecuencia de adquisición de datos es de 60Hz. Las series se encuentran recortadas a 500 timesteps cada una, por lo que cada muestra tiene 4500 datos. Las clases disponibles corresponden a: Biking, Climbing stairs, Descending stairs, Gym biking, Jump roping, Running, Standing, Treadmill Walking y Walking.

3. SOLUCIÓN PROPUESTA

Para abordar la tarea de clasificar las actividades en base a su representación de serie de tiempo multivariada se propone una arquitectura de red neuronal de 2 caminos. El primer camino consiste en una red neuronal convolucional que aprende features desde la entrada y el segundo camino corresponde a una red neuronal recurrente que analiza la entrada en formato secuencial. Los resultados de ambos caminos se combinan en la penúltima capa, la que se conecta a una capa de salida con 9 neuronas (el número de las clases). La entrada de la red no es la serie de tiempo multivariada en sí misma. La arquitectura se aborda en detalle a continuación.

3.1. *Entrada*

Utilizar la serie de tiempo multivariada directamente implica que contamos con una dimensionalidad de 500 x 9, a distribuir, por cada muestra. Considerando que normalmente los kernels utilizados en convoluciones son cuadrados, esto impone limitaciones. Una opción podría ser utilizar kernels de tamaño $[500, x]$, con $x \in 1, 2, 3$, para calcular combinaciones lineales de los valores de las series, por ejemplo promedios, pero, por otro lado, se podría reconfigurar la entrada, por ejemplo a un tamaño 250 x 18, para permitir una mayor plasticidad en los tamaños de kernels, sin embargo, esta reestructuración de los datos no viene sin costo: al mover espacialmente (en la representación 2D de la serie de tiempo multivariada) los datos se fomentan ciertas relaciones y se desincentivan otras. Normalmente, los modelos de series de tiempo consideran un conjunto local de muestras para derivar

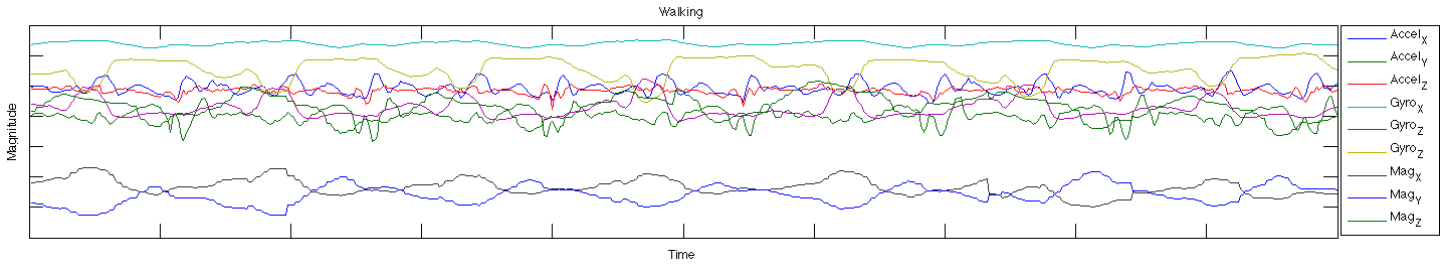


Fig. 1. *Ejemplo de la serie de tiempo multivariada para una muestra correspondiente a la acción Walking.*

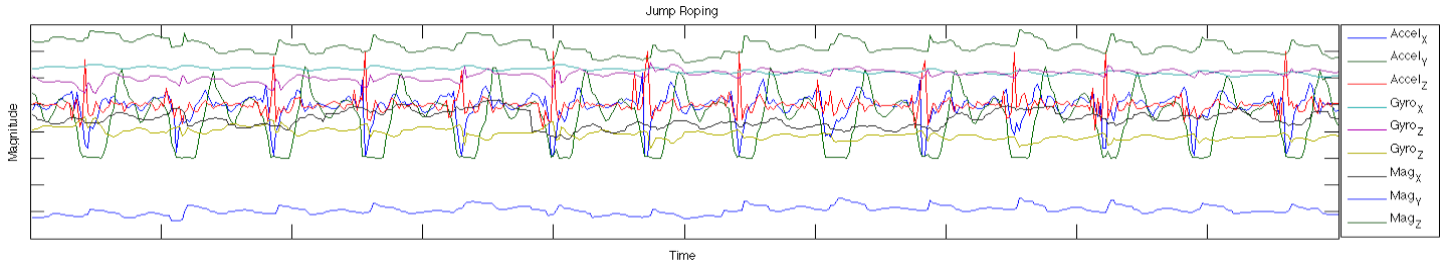


Fig. 2. *Ejemplo de la serie de tiempo multivariada para una muestra correspondiente a la acción Jumping.*

información, por lo que no sería recomendable, considerando esto, utilizar kernels que agrupen valores muy distintantes en el tiempo en la nueva distribución de los datos. Tomando lo anterior en cuenta es que se pueden utilizar capas convolucionales de 1D, donde los kernels corresponden, simplemente, a sliding windows con parámetros a determinar. Sin embargo, esta reducción puede ser interpretada como la pérdida de la oportunidad de generar una representación 2D que nos ayude a explotar la información adecuadamente. En un trabajo[3] se propone crear una representación 2D de una serie de tiempo multivariada de C canales apilando los canales en un orden específico, de forma que cada canal se encuentra, al menos una vez, contiguo a todo otro canal, lo que permite a filtros cuadrados de n x n extraer información local desde cualquier par de canales, ordenados en el tiempo (a diferencia de hacer un reordenamiento de tamaño 250 x 18). Además, cada canal se representa el mismo número de veces, por lo que no se estaría dando preferencia a una secuencia de los datos.

En la Fig. 3 se aprecia el algoritmo que genera la representación 2D de la serie de tiempo multivariada (desde aquí en adelante llamada Signal Image). En el caso del dataset UCF-iPhone Dataset, las series de tiempo multivariadas tienen 9 canales, por lo que el número de señales $N_s = 9$. La primera secuencia en ser agregada a la Signal Image es la secuencia correspondiente al primer canal. El parámetro SIS es una recopilación de todas las señales que se han ido agregando al final de la Signal Image, lo que puede ser interpretado como agregarle canales. El loop corresponde a ir incrementando el contador j y revisar si la secuencia ij o la secuencia ji existen en el Signal Image, siendo la forma fácil de verificar esto revisar si el número i es seguido por el número j en SIS, o si

Algorithm 1 Raw Signals → Signal Image

Notations:
- Signal Image (SI): a 2D array to store permutated raw signals.
- Signal Index String (SIS): a string to store signal indices, whose length is N_{SIS}.

Intput: N_s signal sequences. *// As shown in Fig.2(a), each signal is label with a sequence number. The number of signal sequences $N_s = 9$.*

Loops:
```
i = 1; j = i + 1;
SI is initialized to be the i-th signal sequence;
SIS is initialized to be 'i'; N_SIS=1;
while i ≠ j do
  if j > N_s then
    j = 1;
  else if 'ij' ⊄ SIS && 'ji' ⊄ SIS then
    Append the j-th signal sequence to the bottom of SI;
    Append 'j' to SIS; N_SIS = N_SIS + 1;
    i = j; j = i + 1;
  else
    j = j + 1;
  end if
end while
```

Output: SI.

// The final SIS is '123456789135792468147158259369483726 1' with $N_{SIS} = 37$, when $N_s = 9$.

Fig. 3. *Descripción del algoritmo para generar la representación 2 dimensional de una serie de tiempo multivariada.*

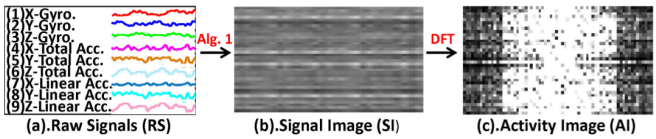


Fig. 4. *Ejemplo de flujo de transformación de una serie multivariada de 9 canales.*

el número j es seguido del número i. Como se muestra en la imagen, para el caso de 9 canales el orden final de apilamiento sería $123456789135792468147158259369483726 1$, notando que el primer canal tiene una repetición más al final. Para evitar un desbalance en la representación de la información es que se acorta la representación a la secuencia de apilamiento $12345678913579246814715825936948372 6$, es decir, se elimina el último canal.

Esta nueva representación ya es suficiente para producir resultados significativos utilizando kernels cuadrados de n x n en una arquitectura de red convolucional, pero además los autores[3] proponen calcular una FFT sobre la Signal Image para obtener lo que llaman una Activity Image. Esto se ha puesto en práctica en el presente trabajo. Tenemos así, que los datos alimentados a la red corresponderan a las Activity Image de la serie de tiempo multivariada de cada muestra. Como se mencionó anteriormente, la existencia de los 3 canales del magnetómetro mejora los resultados de la clasificación. Entendiendo la estructura de la representación escogida para la entrada vemos que esta aumenta exponencialmente a medida que aumenta el número de canales, por lo que agregar los 3 canales del magnetómetro, aunque asumiéramos que estos contienen poca información, ayuda a representar mejor los otros 6 canales. Sin embargo, estos canales del magnetómetro sí portan información y, de hecho, es posible realizar clasificación del dataset utilizándolos sin los demás, aunque estos resultados son peores.

3.2. *Red convolucional*

Una vez que tenemos la entrada, que corresponde a la Activity Image, la haremos circular por 2 caminos. Uno de ellos corresponde a una red convolucional de 2 capas. Los autores que proponen el Activity Image[3] también proponen una arquitectura de red convolucional (Fig. 5) de 2 capas convolucionales, con filtros de $5x5$. Luego de cada capa convolucional se hace mean pooling, con filtros de $4x4$ y $2x2$, respectivamente. Los strides corresponden a 1, para las capas convolucionales y 4 y 2 para las capas de pooling, respectivamente. La primera capa convolucional tiene 10 filtros, y la segunda 5. Al final de su arquitectura se agrega la información en una capa densa de 120 neuronas, que se conecta con la capa de salida con una neurona por cada clase a predecir. Para el presente trabajo se adoptó una arquitectura de red convolucional similar, descrita a continuación en detalle:

- La primera capa convolucional tiene un kernel de $5x5$, con 5 filtros, stride 1, función de activación ReLU e inicialización de He (que se sabe es bien portada para ReLU).
- La segunda capa convolucional tiene un kernel de $5x5$, con 10 filtros, función de activación ReLU e inicialización de He.
- Luego de cada capa convolucional no se hace mean pooling, sino que se utiliza batch normalization (para introducir regularización), seguido de un squeeze and excite block[7].
- En vez de una capa densa al final de la red, simplemente se acoplan (a través de una capa flatten) las salidas del segundo squeeze and excite block (que va después de la segunda capa convolucional).

Para completar la descripción de la red convolucional, describiremos el squeeze and excite block. Es una secuencia de las siguientes capas:

- Global Average Pooling en 2D (puede ser 1D o 2D).
- Capa densa de $Cmod16$ neuronas, con C el número de canales, función de activación ReLU e inicialización de He.
- Capa densa de C neuronas, función de activación sigmoide e inicialización de Xavier (que se sabe es bien portada con sigmoide).
- Capa de multiplicación entre la salida de batch normalization, de la capa convolucional inmediatamente anterior al squeeze and excite block, y un vector de tamaño C, que corresponde al resultado del squeeze and excite block: Este vector tiene un ponderador por cada uno de los canales resultantes de la capa de convolución.

Recordando que cada filtro describe un feature map, y que, por lo tanto, engendra un canal, un squeeze and excite block es equivalente a descartar o enfatizar la información de determinados feature maps al momento de clasificar determinadas muestras (Fig. 6). Es un concepto similar a la atención[11], pero con un espectro más reducido y computacionalmente mucho más barato. Otro problema que se busca evitar con esta arquitectura sencilla es la confección de una arquitectura que, por su alto poder expresivo, simplemente se sobreajuste al reducido conjunto de entrenamiento y generalice mal.

3.3. *Red recurrente*

El segundo camino de la red propuesta corresponde a una celda LSTM[12] de profundidad 1, nuevamente intentando no utilizar una arquitectura que se sobreajuste tan fácilmente (como sería, por ejemplo, una celda LSTM de profundidad

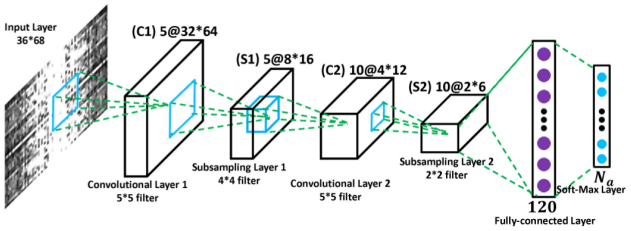


Fig. 5. *Red convolucional propuesta en[3].*

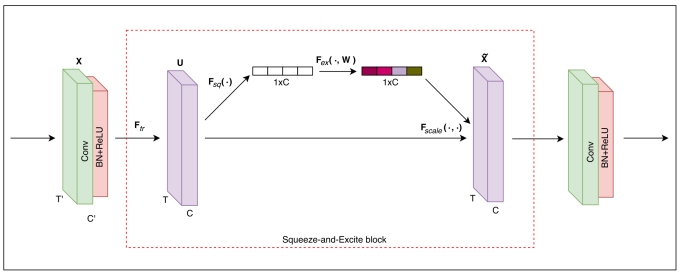
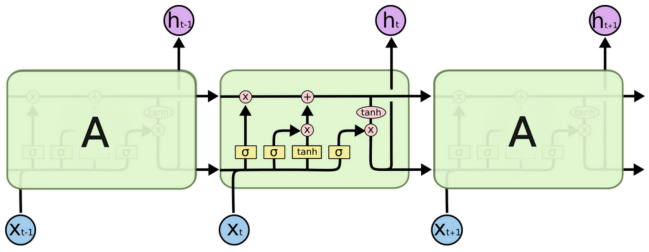


Fig. 6. *Representación de un squeeze and excite block. Corresponde al cálculo de ponderadores para cada canal de salida de una capa convolucional.*

4). Si bien, algunos autores proponen alimentar a la celda los features de una capa convolucional, regularmente la primera, optaremos por alimentarle a la celda la Activity Image, pero descompuesta en el espectro de las frecuencias. Le alimentamos, así, secuencias de 36 valores, por 251 timesteps (lo que podría interpretarse como timesteps, pero aquí es simplemente una secuencia). Las celdas LSTM[12] se prestan bien para la comprensión de entradas en formato secuencial, y, además, permiten tener entradas y salidas de largo arbitrario, aunque eso no es aprovechado en esta arquitectura, ya que la entrada y la salida cuentan con dimensionalidad fija.

Una celda LSTM es una neurona artificial con memoria

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

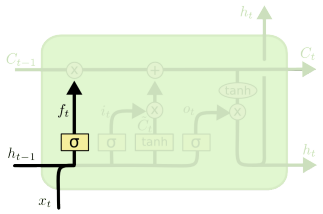


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \quad \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \quad C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Fig. 7. *Representación de una celda LSTM.*

Fig. 8. *Nomenclatura de los diagramas.*

interna, es decir, no aprende solamente pesos como una neurona artificial standard, sino que mantiene un estado interno C_t. En la Fig. 7 podemos apreciar una representación de una celda LSTM, con sus 4 compuertas básicas. Las 3 celdas corresponden a la misma neurona artificial en distintos instantes de tiempo. La que recibe de entrada la secuencia X_{t-1} corresponde a la neurona artificial en el instante $t-1$, la que recibe de entrada la secuencia X_t corresponde a la neurona artificial en el instante t, y así sucesivamente. Notamos, de lo anterior, que la neurona artificial se va alimentando información a sí misma. La salida de la red corresponde a los $\{h_i\}$, por lo que la salida de la red se compone de la salida de la neurona artificial para cada instante de tiempo. Si la muestra tiene T secuencias distintas, entonces la celda LSTM generará T salidas h_i. En el diagrama, la neurona artificial se alimenta a sí misma 2 valores, codificados como 2 flechas. La flecha superior corresponde a la banda de la memoria C_i, y la flecha inferior corresponde a h_i (la neurona da como salida h_i pero también se lo alimenta a sí misma para calcular el siguiente valor de la secuencia). Así, la celda LSTM utiliza, en el instante o paso i, la secuencia X_i de entrada, la salida h_{i-1} del instante anterior y la memoria interna C_i.

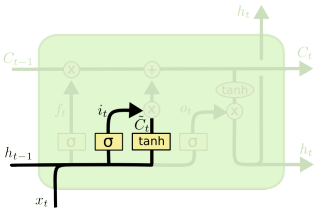


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Fig. 9. *Mecanismo de forget gate.*

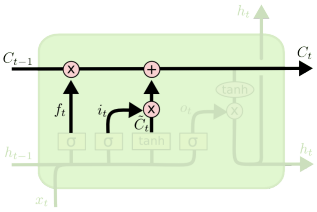
La Fig. 9 muestra el comportamiento de la forget gate. Aquí se concatenan los valores de la salida recibida del paso anterior h_{t-1} y los valores de la secuencia de entrada x_t y se operan a través de una capa densa normal con función de activación sigmoide. El fin de esta operación es determinar ponderadores por los cuales multiplicar los valores de C_{t-1}, o el estado interno de la celda recibido desde el paso anterior. Como los valores de la salida f_t están entre 0 y 1 no se puede enfatizar la información ya contenida, pero sí se puede borrar completamente (con un 0).

La Fig. 10 muestra el comportamiento de la learn gate. De forma análoga a la puerta anterior se calculan ponderadores i_t, en base a la concatenación de h_{t-1} y x_t, para determinar cuales valores se quiere aprender y en que tasa. $\tilde{C}_t$ corresponde a los valores candidatos a agregar a la memoria.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Fig. 10. *Mecanismo de learn gate.*

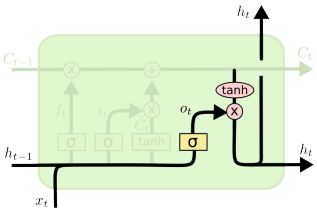


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Fig. 11. *Mecanismo de update gate.*

La multiplicación de estos 2 elementos se suma al estado de la memoria C_{t-1}, como se indica en la Fig. 11. Ahí también notamos el punto en que se olvidan elementos de la memoria, es decir, la multiplicación de f_t con C_{t-1}.

Así se ha terminado de actualizar el estado de la memoria de la celda, lo que se transmite hacia el siguiente paso. A esta actualización se le suele llamar update gate (sobre todo cuando se combinan los 2 pasos descritos anteriormente).



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Fig. 12. *Mecanismo de output gate.*

Finalmente, es necesario calcular el valor de la salida h_t, para hacer esto se utiliza el nuevo estado C_t y un conjunto de ponderadores o_t.

3.4. *Arquitectura completa*

La arquitectura completa, como ya se mencionó, se compone de estos 2 caminos que se alimentan de la FFT de la Activity Image. Esta idea de combinar una red convolucional y una celda LSTM, para la clasificación de series de tiempo, no es nueva[4], pero la arquitectura propuesta en este trabajo constituye una combinación entre [4] y [3]. Tal como se muestra en la Fig. 13, el camino de la LSTM es idéntico, por lo que la celda utilizada en la presente arquitectura también incorpora atención[11] y dropout. Por otro lado, el camino de la red convolucional es distinto, ya que las convoluciones son en 2D, con un número substancialmente menor de filtros y existiendo 2 capas convolucionales con batch normalization, ReLU y squeeze and excite block, sin global pooling, pasando directamente a la concatenación con la salida de la red LSTM, todo esto con el fin de reducir la complejidad del modelo en función de la complejidad del dataset utilizado.

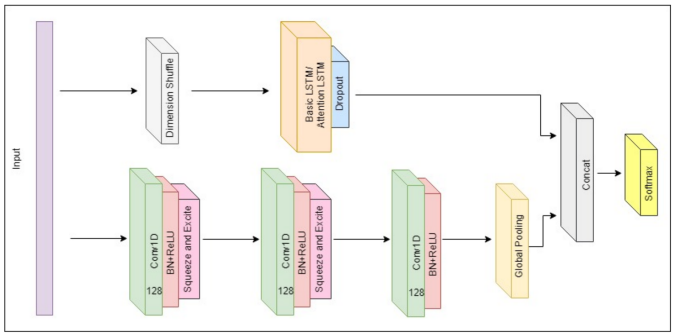


Fig. 13. *Arquitectura propuesta en [4].*

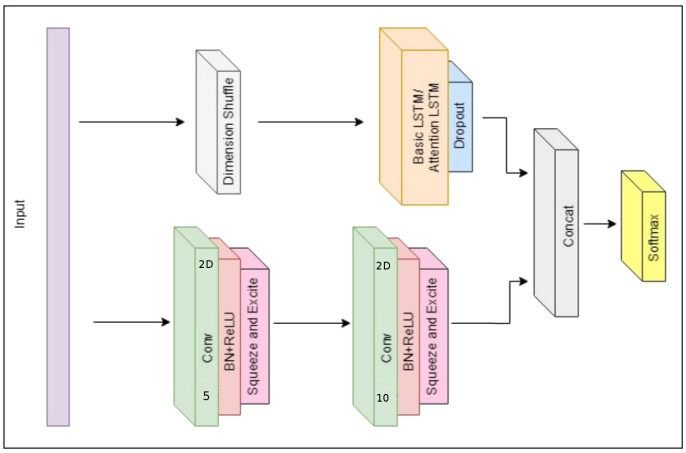


Fig. 14. *Arquitectura propuesta en el presente trabajo.*

4. EXPERIMENTOS

Esta sección describe detalles sobre los contenidos del archivo MALSTM_activity_trainandval.py, que contiene las funciones para generar el dataset, definir la arquitectura, entrenar, calcular accuracy y matriz de confusión. El código corresponde a una implementación en TensorFlow 1.5, en Python 3.6, de la arquitectura descrita, utilizando algunas herramientas de sklearn.

Para determinar los pormenores respecto a la arquitectura y sus parámetros, se dividió el dataset en conjuntos de entrenamiento, validación y test, correspondientes a 60 %, 20 % y 20 %, respectivamente. Las configuraciones fueron probadas sobre el conjunto de validación, notando los detalles que

serán comentados en la subsección siguiente.

4.1. *Hiperparámetros y detalles generales*

- Tal como proponen los autores de[3], utilizar el valor absoluto de los números complejos obtenidos en la FFT de la Signal Image es lo que da mejores resultados. Podría pensarse que la concatenación de la parte real e imaginaria podría llevar a resultados similares, pero esto producía una pérdida de accuracy de, aproximadamente, 5 %.

- Se definió que el batch size con mejores resultados era el conjunto de entrenamiento completo. El por qué se explica más abajo.

- Para compensar por el pequeño tamaño del dataset se utilizaron varias herramientas de normalización, las que hicieron que la red se demorara órdenes de magnitud más steps en converger, pero produciendo mejores resultados. Sin ningún tipo de regularización la red tiene la capacidad de llegar a un error menor a 0.0001 en algo menos de 20 steps (recordando que un step es una pasada del conjunto de entrenamiento, por el batch size). A saber, se utilizó batch normalization, dropout y squeeze and excite block.

- El tamaño de las neuronas ocultas de la celda LSTM es de 635.

- Debido a un problema de explosión de gradientes, conocido en las celdas LSTM, se incorporó gradient clipping, definiendo el valor máximo del módulo de los vectores como 5.

- El optimizar elegido fue Adam, aunque se pensó por un momento poner a prueba otra celda LSTM como optimizador[8]. El learning rate escogido corresponde a 0.0001, notando que 0.001 produce una convergencia mucho más rápida y con resultados comparables, pero es mucho más susceptible a problemas de gradiente, por lo que 1 de cada 3 experimentos terminaba en errores que se escapaban.

- Además de la regularización, para compensar por el bajo número de muestras, se utilizó ruido gaussiano de media 0 y varianza 1 para realizar data augmentation. Se probó agregar el ruido antes y después de la FFT a la Signal Image, notando que aplicarlo antes aumenta considerablemente el tiempo de convergencia, pero genera mejores resultados (generaliza mejor al conjunto de validación).

- La función de pérdida es entropía cruzada.

- Se concluyó que realizar Standard scaling a las series de tiempo originales, antes de calcular las Signal Images, disminuía el tiempo de convergencia y facilitaba la generalización.

- Como se mencionó anteriormente, el magnetómetro aporta información valiosa, por lo que sus canales no se descartaron.

- Se determinó que el punto en que la red debe dejar de entrenar es cuando alcanza una pérdida menor o igual a 0.002. Disminuir demasiado el error lleva al sobreajuste y mala generalización.

- Considerando que las redes son muy susceptibles a oscilar en la vecindad de los pesos que aprenden con las primeras muestras se determinó que los primeros 3 steps corresponden a entrenamiento sin ruido.

- Para aumentar el accuracy sobre el conjunto de validación se utilizó una estrategia de entrenamiento semi-supervisado: Se determinaban las muestras del conjunto de validación que la red estaba más de 0.999 segura sobre su etiqueta para ser incluidas al conjunto de entrenamiento. Esto efectivamente mejoraba los resultados del conjunto de validación, pero era complicado determinar las reglas de número de iteraciones a entrenar. Esto se encuentra comentado en el código.

- Debido a la naturaleza de las Activity Images, subdividir las muestras no mejora los resultados. Vale la pena recordar que esto es otra técnica de data augmentation, y que esto se utilizó a través del ruido gaussiano.

4.2. *Resultados finales*

Una vez que se determinaron los puntos anteriormente mencionados, utilizando el conjunto de validación, se experimentó con el conjunto de test. Los resultados obtenidos promedian 0.94545451 % de accuracy sobre el conjunto de test, alcanzando como máximo valor de accuracy 0.96103895 % (3 muestras mal clasificadas, Matriz de confusión en Fig. 21), aunque el valor cercano al 93 % parece ser el más común. Si bien, con este resultado no se puede decir que el problema está completamente resuelto, para ser un clasificador sin agrupación, a diferencia de lo propuesto en[10], que aprende sus propios features, pareciera ser que los resultados son buenos. En la Fig.15 notamos como la convergencia, al aplicar el ruido gaussiano después de la FFT, es más estrepitosa que en el caso de aplicar el ruido antes, como es el caso de la Fig. 16, aunque hacer esto puede provocar, de pronto, una descompensación repentina de los pesos de la red, como se aprecia en la Fig. 17, es por esto que se escogió como criterio un error, ya que esto le da a la red la oportunidad de recuperarse, incidentalmente, el caso de la Fig. 17 corresponde al mayor accuracy logrado en el conjunto de prueba. La clase 1 (Climbing) y la

clase 2 (descending) son las que más tienden a confundirse entre ellas.

5. CONCLUSIONES

La arquitectura de red propuesta se comporta bien frente al problema de clasificar las series de tiempo del dataset UCF-iPhone Dataset, detectando apropiadamente features significativos en base a la Activity Image. Considerando que la diferencia entre el mejor y el peor resultado, en términos de accuracy sobre el conjunto de prueba, es de aproximadamente 6 %, una forma directa de obtener mejores resultados serían entrenar un enjambre de clasificadores basados en la misma arquitectura y someterlos a un sistema de votación. Esto es, sin embargo, bastante prohibitivo en términos de tiempo, ya que el tiempo promedio de entrenamiento de la red, utilizando una GPU NVIDIA GTX 1070 es de, aproximadamente, 47 minutos, pudiendo fácilmente dispararse a las horas, dependiendo de la oscilación de los pesos de la red. Estos altos tiempos de entrenamiento se deben a la dificultad que agrega la regularización y el ruido gaussiano. Siendo menor el tiempo de convergencia cuando este ruido se aplica después de la FFT, pero perdiendo poder de generalización. Es lamentable que el esquema de entrenamiento semi-supervisado no diera mejores resultados, en algunos casos incluso empeorando el desempeño. Esto se debe, sin embargo, a que la definición de los pasos de entrenamiento necesarios para ajustarse a las nuevas muestras no son directos. Una forma de arreglar esto podría ser disminuir la tasa de aprendizaje al momento de entrenar con las nuevas muestras y hacer batches que contengan las muestras nuevas, sin muestras del conjunto de entrenamiento original. Otra forma de mejorar el clasificador podría ser reemplazar la red convolucional sencilla por una red convolucional como ResNet[6], pero, como ya se mencionó, se espera que esto se sobreajuste demasiado a los datos de entrenamiento. Otra opción para reemplazar la red convolucional podría ser[5], aunque vale la pena preguntarse si se podría eliminar la celda LSTM del todo, ya que esta introduce serios problemas de convergencia debido a los problemas que trae con los gradientes. Debido a la naturaleza secuencial de las muestras es tentador querer explotar las fortalezas de la celda LSTM, pero existen modelos[9] completamente convolucionales que han demostrado tener mejor desempeño en tareas de procesamiento de secuencias, aunque no necesariamente en el dominio del reconocimiento de actividades humanas.

Entrenar una arquitectura moderna de red neuronal sobre un dataset tan pequeño es todo un desafío, es por esto que se hace necesario utilizar casi cada truco disponible en el arsenal, incluso si a veces ciertas combinaciones parecieran contraintuitivas.

6. REFERENCES

[1] UCF-iPhone Dataset, Center for Research in Computer Vision (CRCV), University of Central Florida. http://crcv.ucf.edu/data/SmartPhone/Smartphone_Dataset.zip

[2] http://crcv.ucf.edu/data/UCF-iPhone.php

[3] Jiang, W. Human Activity Recognition using Wearable Sensors by Deep Convolutional Neural Networks. In Proceedings of the 23rd ACM International Conference on Multimedia, Brisbane, Australia, 26–30 October 2015; pp. 1307–1310.

[4] F. Karim, S. Majumdar, H. Darabi, S. Harford, Multivariate LSTM-FCNs for Time Series Classification. https://arxiv.org/pdf/1801.04503.pdf

[5] Chen and Y. Xue, "A Deep Learning Approach to Human Activity Recognition Based on Single Accelerometer," in SMC , 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, 2016.

[7] J. Hu, L. Shen, G. Sun, Squeeze-and-Excitation Networks. https://arxiv.org/abs/1709.01507

[8] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In International Conference on Artificial Neural Networks , pages 87–94. Springer, 2001.

[9] A. van den Oord et al, "WaveNet: A Generative Model for Raw Audio", arXiv:1609.03499

[10] C. McCall, K. Reddy, M- Shah, Macro-Class selection for hierarchical K-NN classification of inertial sensor data. In: PECCS (2012).

[11] Xu, J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, Show, attend and tell: Neural image caption generation with visual attention. https://arxiv.org/abs/1502.03044

[12] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural computation , vol. 9, no. 8, pp. 1735–1780, 199

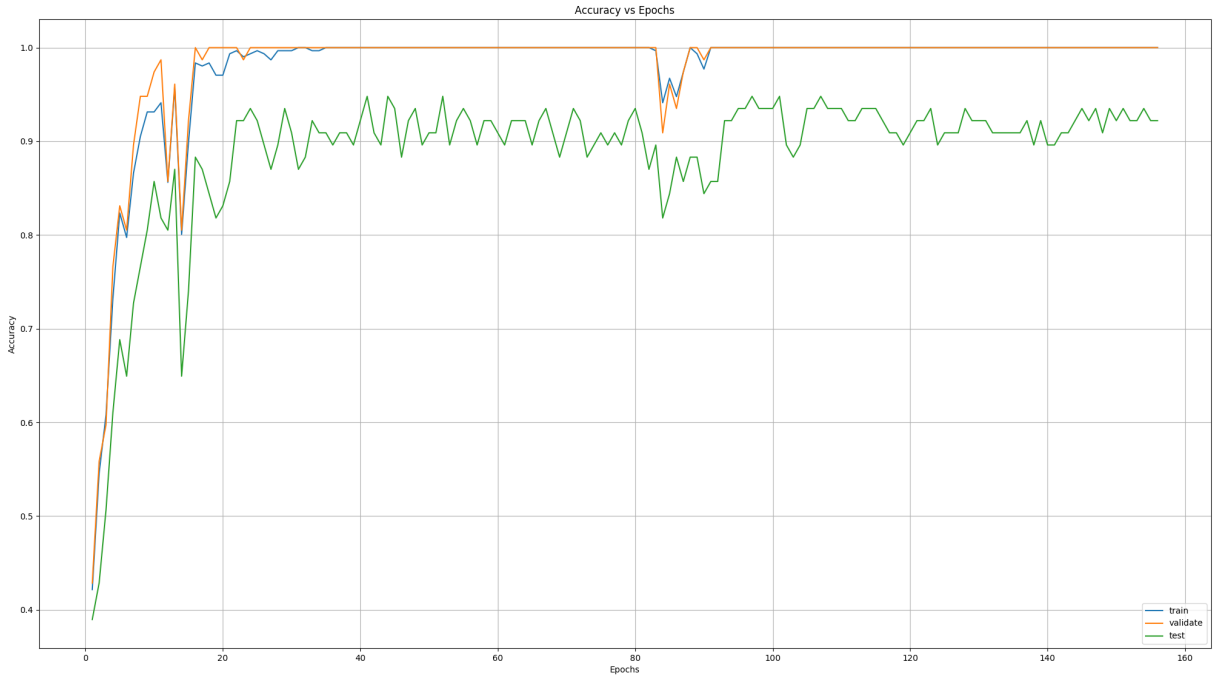


Fig. 15. *Accuracy vs Épocas, ruido gaussiano después de FFT. Convergencia errática y rápida.*

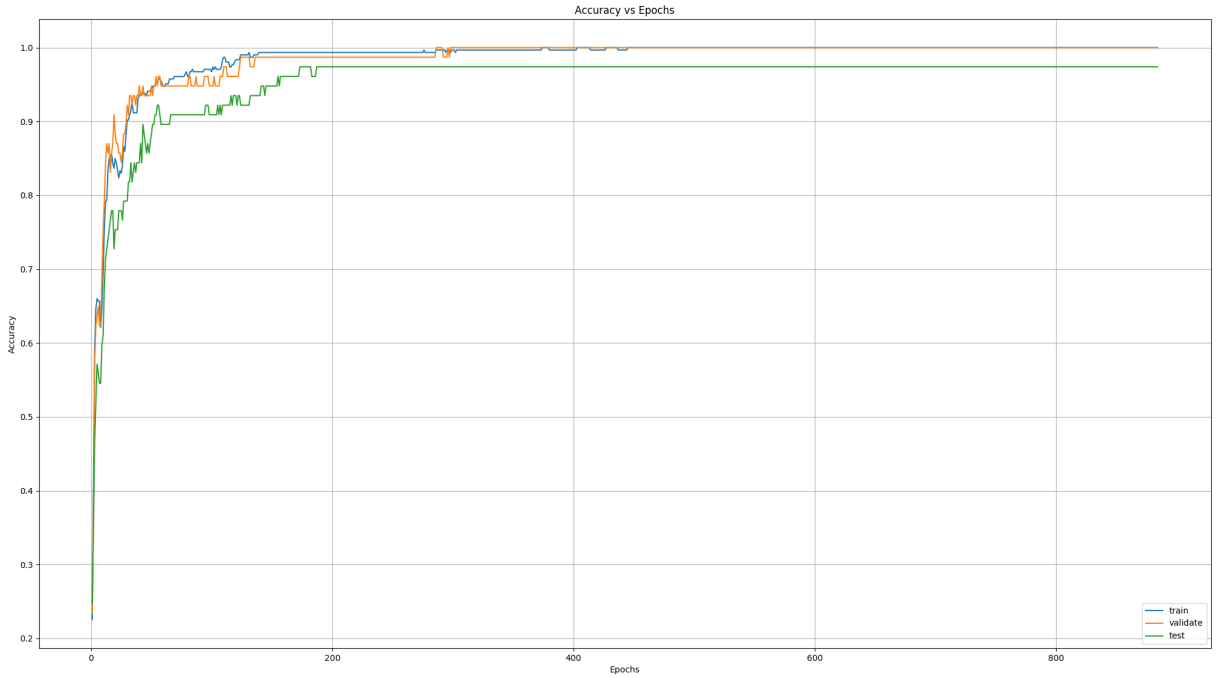


Fig. 16. *Accuracy vs Épocas, ruido gaussiano antes de FFT.*

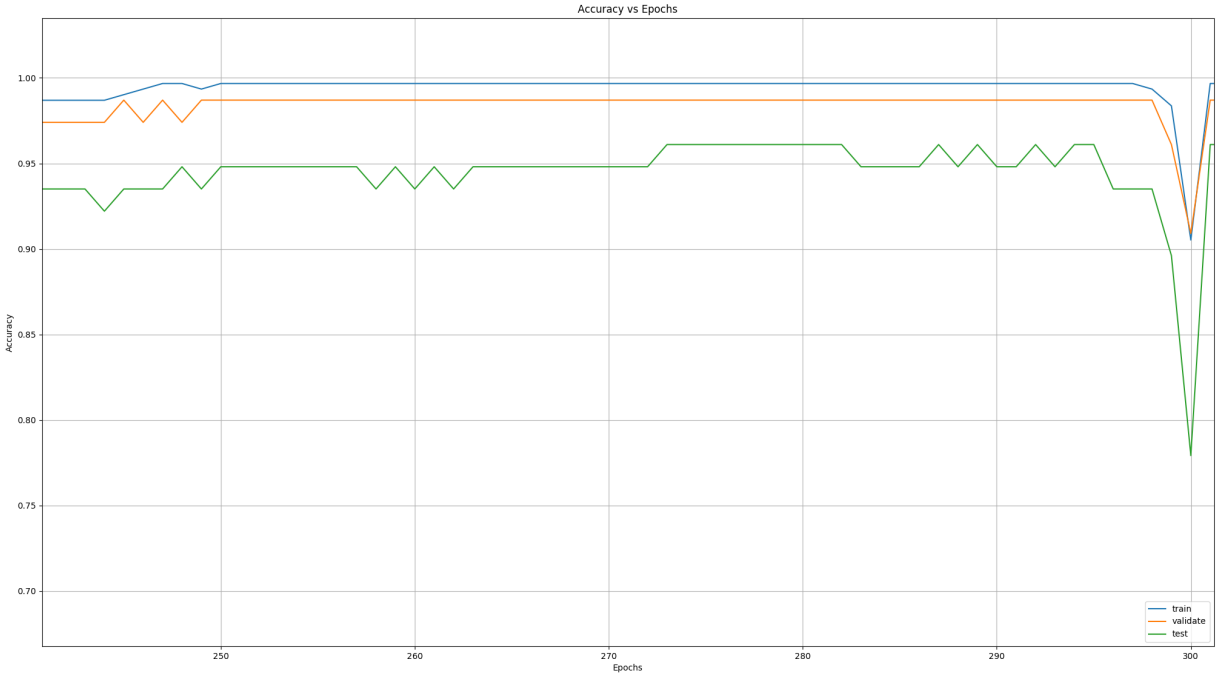


Fig. 17. *Accuracy vs Épocas, ruido gaussiano antes de FFT. Vemos como el error se dispara un poco antes de alcanzar el criterio de término.*

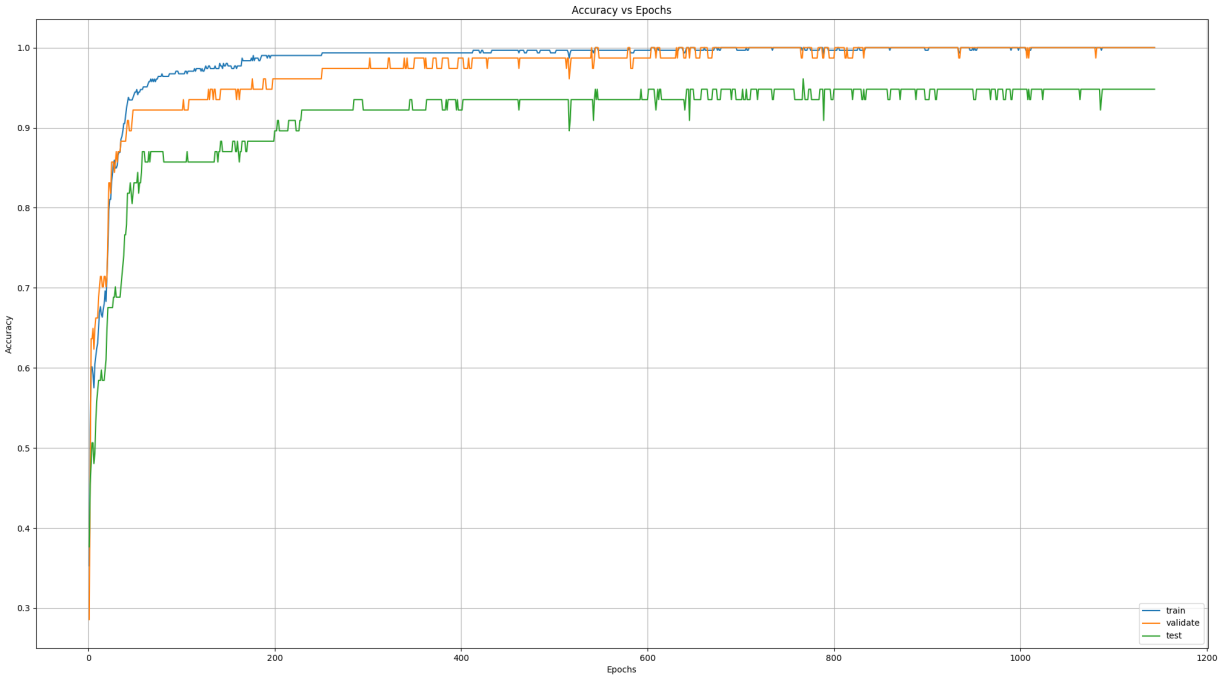


Fig. 18. *Accuracy vs Épocas, ruido gaussiano antes de FFT. Vemos como el tiempo de entrenamiento se dispara. Notamos que se podría haber parado mucho antes, pero por lo visto en la Fig. 17 no podemos saber a priori si ya estamos estables o no.*

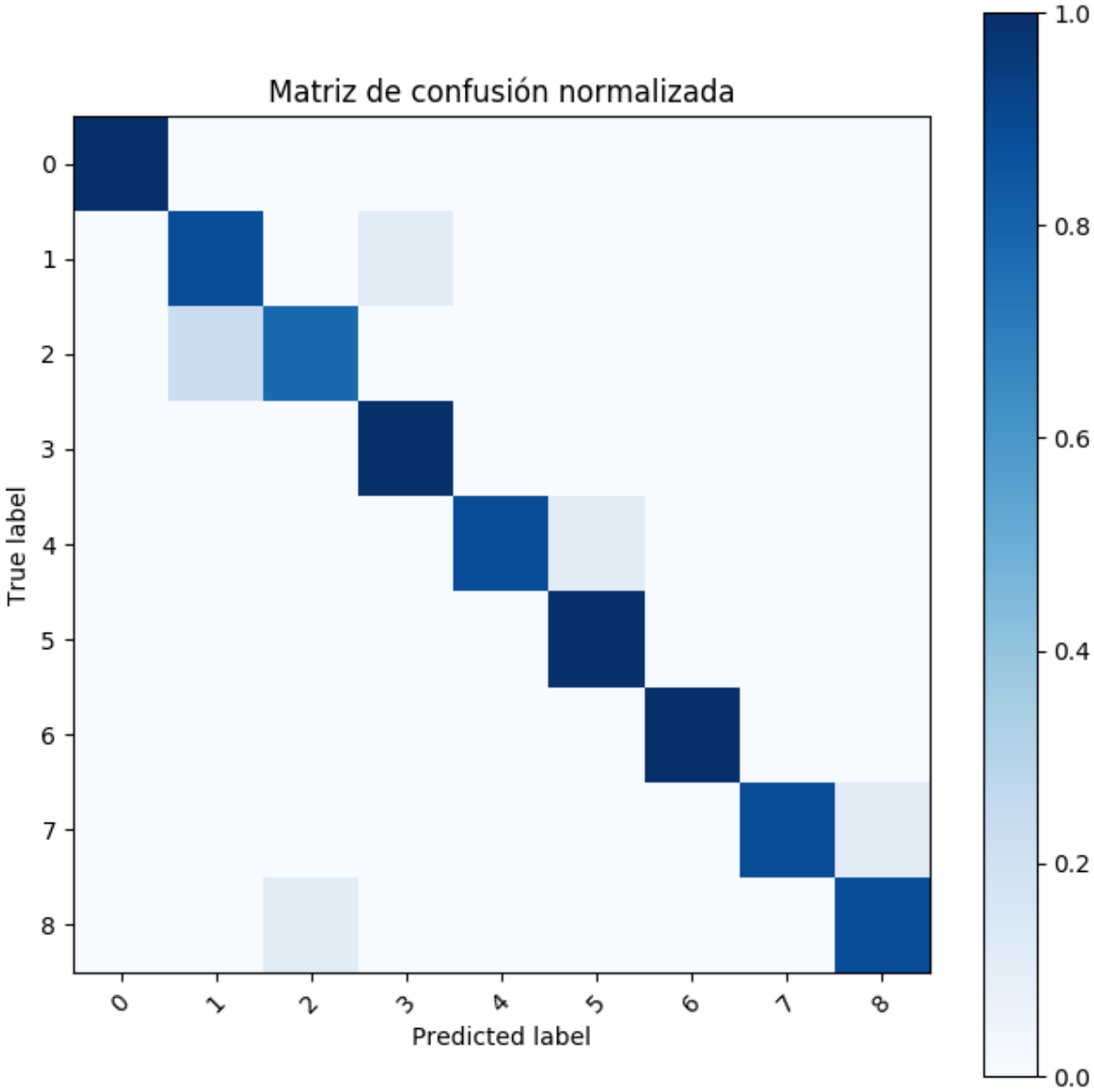


Fig. 19. *Matriz de confusión, ruido gaussiano después de FFT. Accuracy 0.9350649. Corresponde a Fig. 15.*

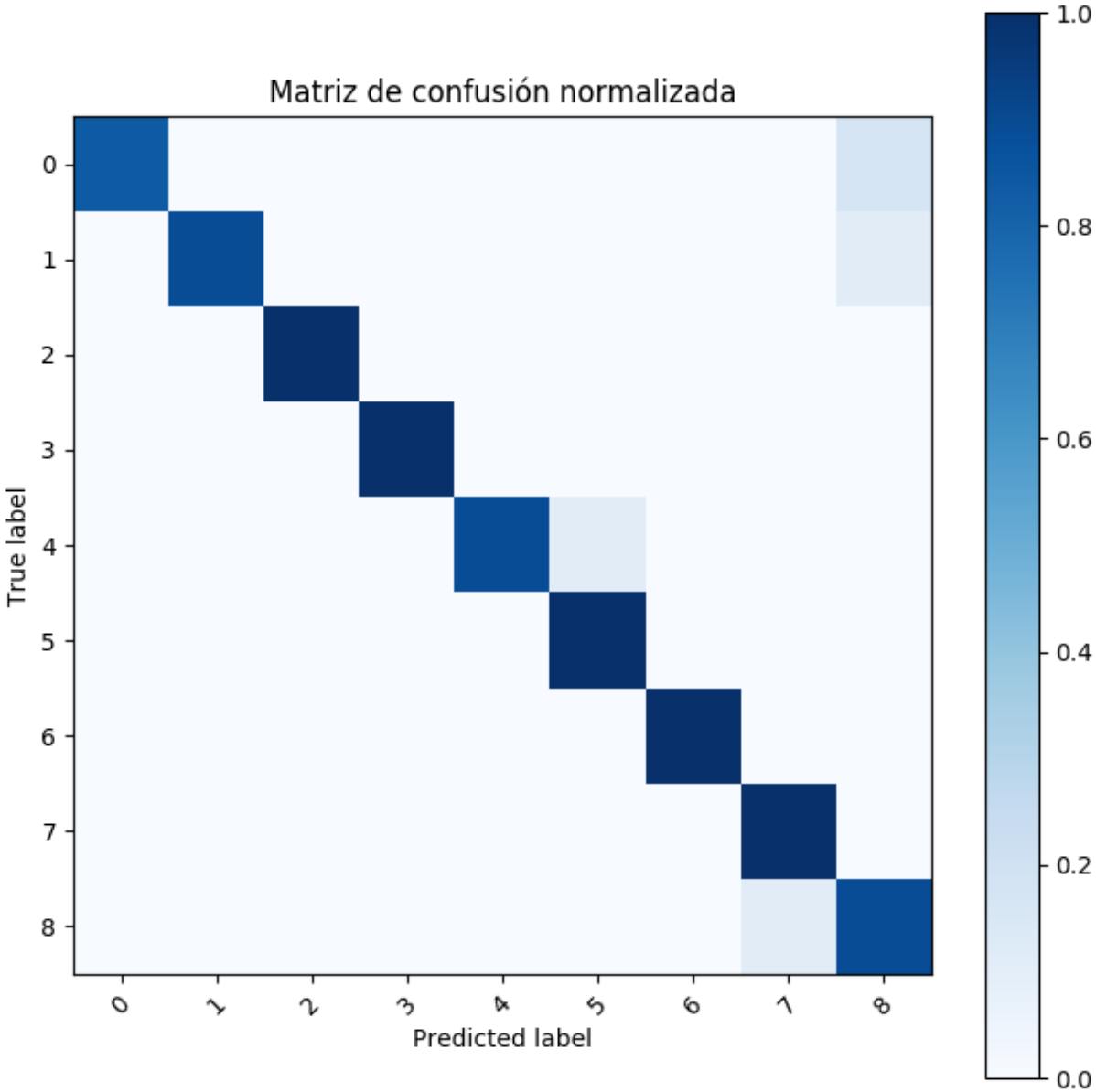


Fig. 20. *Matriz de confusión, ruido gaussiano antes de FFT. Accuracy 0.9480519. Corresponde a Fig. 16.*

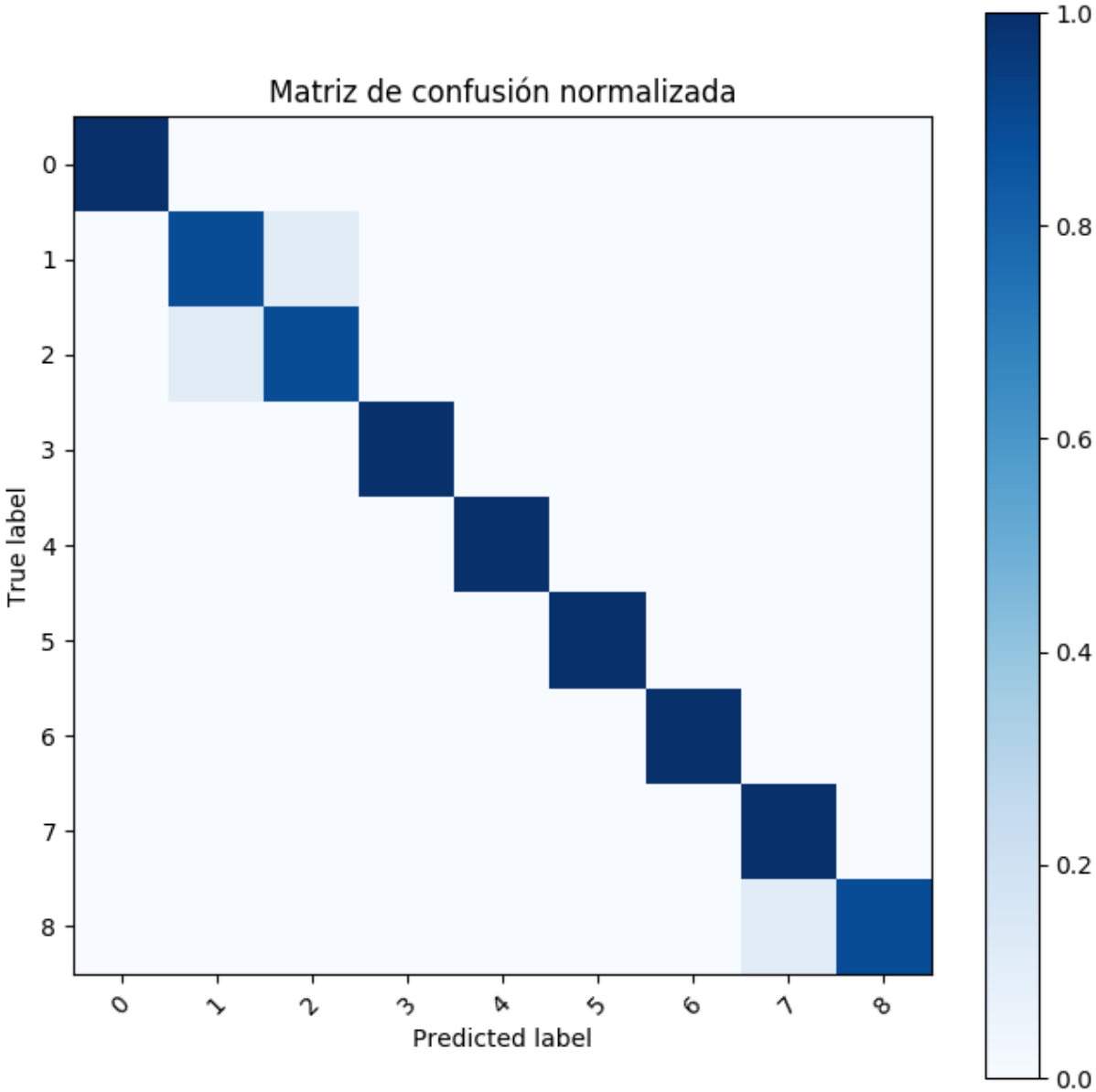


Fig. 21. *Matriz de confusión, ruido gaussiano antes de FFT. Accuracy 0.96103895. Corresponde a Fig. 17.*

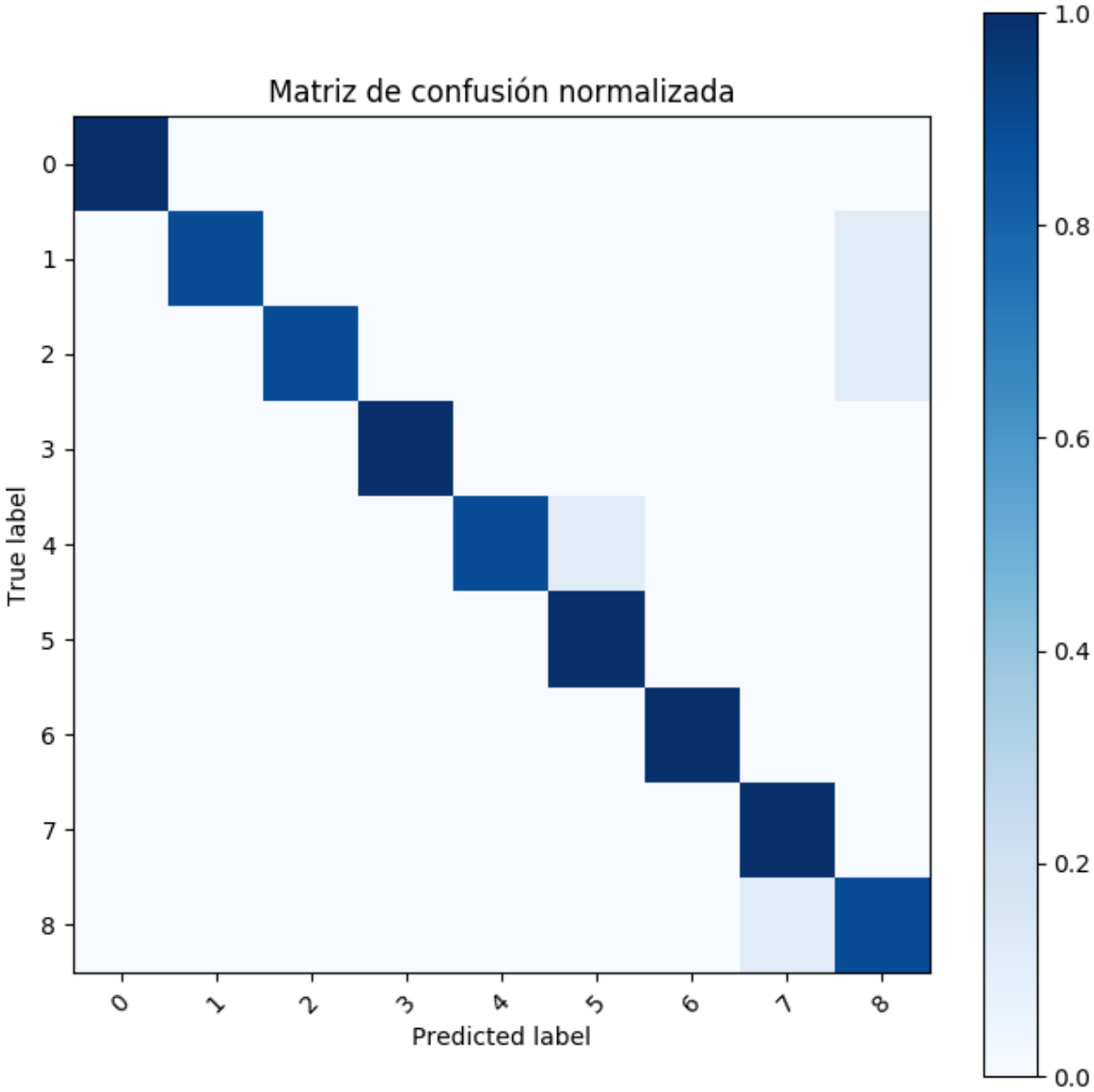


Fig. 22. *Matriz de confusión, ruido gaussiano antes de FFT. Accuracy 0.9480519. Corresponde a Fig. 18.*