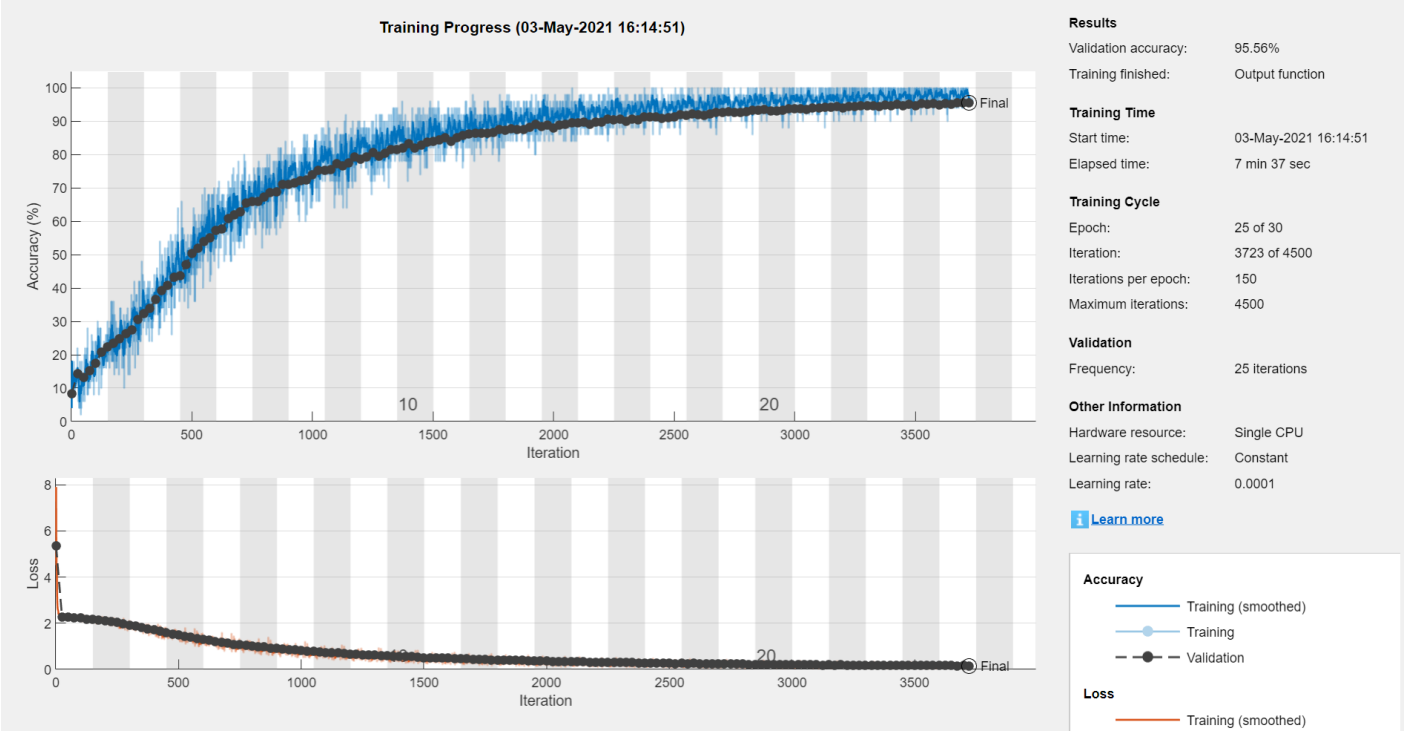
Práctica redes neuronales de convolución

Diego Hernández Jiménez

abril 2021

2. Presentar gráficamente la evolución de exactitud y pérdida durante el proceso de aprendizaje, superponiendo cada 25 ensayos de aprendizaje los valores de las mismas variables de rendimiento en el conjunto de test (ver para ello https://es.mathworks.com/help/nnet/ref/trainingoptions.html)



3. Decidir si se ha producido sobreajuste -justificando la respuesta- y en caso de respuesta afirmativa identificar el nº del ensayo a partir del que se inició.

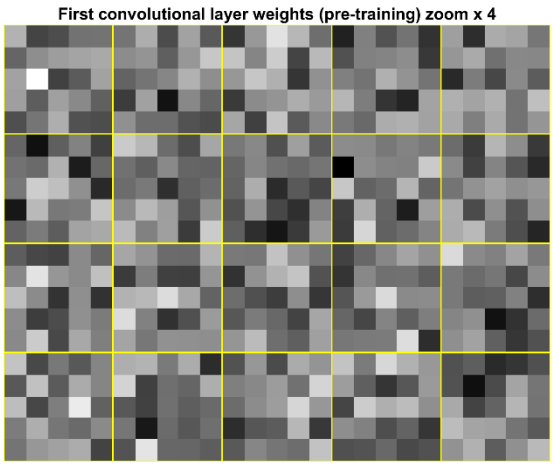
A partir de la inspección visual de ambos gráficos podría concluirse que no parece haberse producido sobreajuste. Si se examina, por ejemplo, el gráfico superior, se observa que la precisión en conjunto de validación sigue la misma tendencia que la precisión del conjunto de entrenamiento. Y no parece apreciarse una divergencia de ambas curvas a partir de la iteración 2000, que es cuando la precisión empieza aproximarse a la asíntota. De haberse producido sobreajuste, habría cabido esperar que la precisión en el conjunto de entrenamiento siguiese con la tendencia creciente mientras que la precisión del conjunto de validación se mantuviese con un crecimiento mínimo o que descendiese. No ocurre eso en este caso, al menos de manera obvia.

Sin embargo, como el juicio está basado en impresiones visuales, puede ser útil añadir una función de control que durante el entrenamiento compruebe si se está produciendo un aumento más o menos relevante en la precisión del conjunto de validación (al menos un incremento de 0,1). Si se alcanzan 10 iteraciones en las que no se produce un aumento de este tipo se finaliza el aprendizaje. Esto es lo que se pretende con la función `stopIfAccuracyNotImproving`, que no es más que una modificación de la función con el mismo nombre que se encuentra definida en la documentación de Matlab. Esta función permitirá determinar con más confianza si se estaba produciendo sobreajuste, pues controla explícitamente que no se prolongue el aprendizaje más de lo debido, y, por tanto, que se produzca sobreajuste.

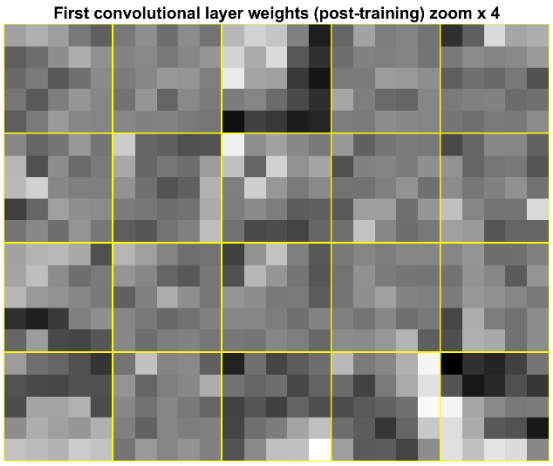
Tras incorporar esta función al proceso de aprendizaje y reajustar el modelo se vuelven a obtener los mismos resultados. El criterio de parada de alcanzar una precisión de 0,98 en el conjunto de entrenamiento se cumple antes que el criterio impuesto por `stopIfAccuracyNotImproving`. Ello implica que no se llega a tener en el conjunto de validación una secuencia de al menos diez puntuaciones de precisión con incrementos inferiores a un 1%. Por tanto, parece razonable pensar que no se produce sobreajuste.

4. Presentar las imágenes de los núcleos de la primera capa convolucional, resultantes del proceso de aprendizaje.

Los núcleos antes de comenzar el aprendizaje se establecen aleatoriamente y quedan representados así:

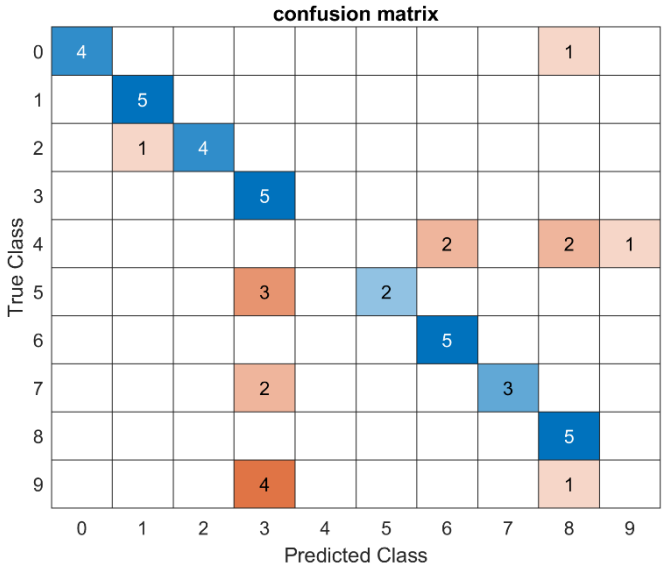


Una vez finalizado el proceso de aprendizaje, los núcleos son los siguientes:



5. Crear una base de imágenes propia con 5 ejemplos de cada uno de los diez dígitos, e indicar el rendimiento de la red cuando es aplicada a dicha base de datos. Presentar la imagen de la activación de la primera matriz (*feature*) de cada capa de la red ante el primer ejemplo creado por el alumno para para cada una de las 10 cifras.

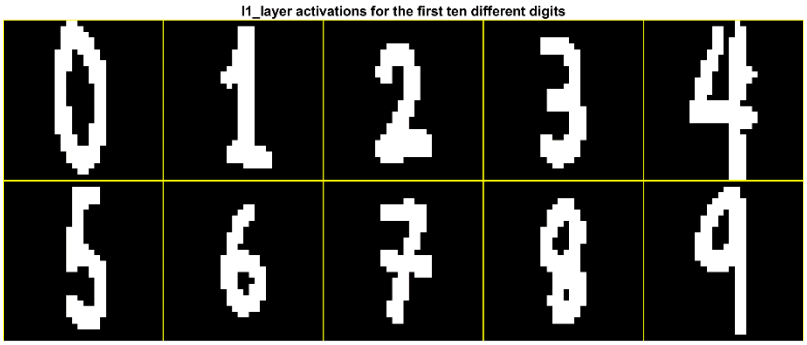
La red entrenada obtiene una precisión (*accuracy*) de aproximadamente 0,66 con la base de datos de cincuenta nuevos ejemplos y produce la siguiente matriz de confusión:



Teniendo en cuenta que en el período de entrenamiento se acabó estabilizando la precisión en 0,9556, los resultados ahora obtenidos son bastante pobres. Aunque, si se observa la matriz de confusión, se comprueba que la bajada de rendimiento no se debe a que la red no consigue clasificar muy adecuadamente todos los números, sino a que no clasifica nada bien un par de números, el cuatro y el nueve. Ello hace sospechar que existan peculiaridades en la forma en la que se han trazado esos números que no se encuentran en los ejemplares de la base de datos original. Sin embargo, resulta difícil confirmar esta sospecha, pues se han elaborado otros conjuntos de "cuatros" y "nueves" distintos, pero se ha mantenido la precisión de 0 para estos dígitos.

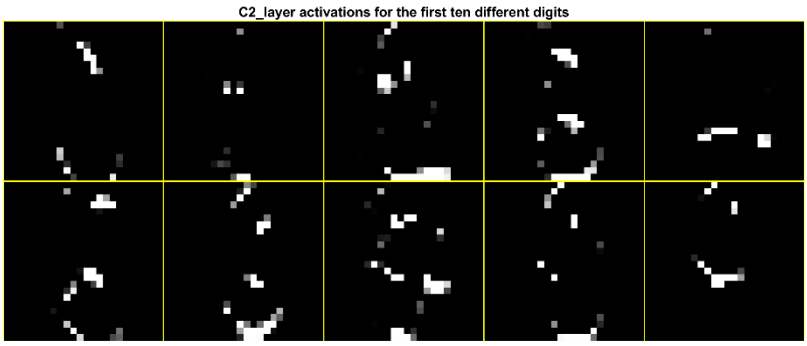
Por otra parte, el patrón de activación resultante de cada capa para el primer ejemplar de cada dígito puede verse a continuación:

Capa de entrada

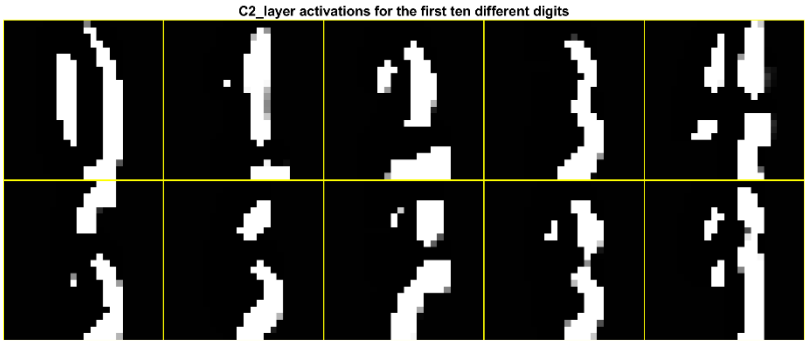


Las activaciones mostradas corresponden a la matriz única de 28 x 28 que es la imagen del dígito que se quiere clasificar.

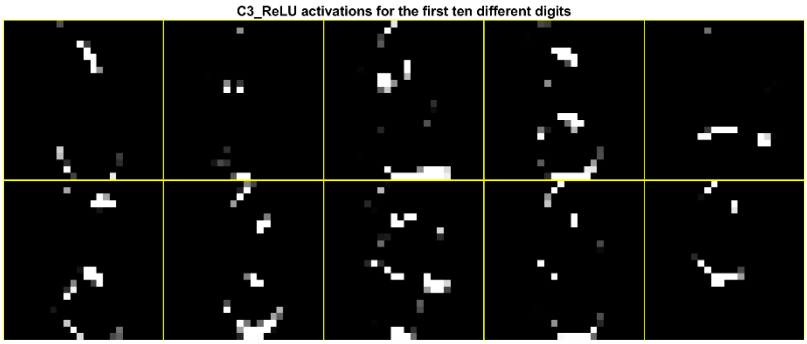
Capa de convolución



En este caso, sin embargo, debido que el *output* de la capa está compuesto por un número distinto de neuronas (24 x 24 x 20) y con una organización distinta (matrices de 24 x 24), es conveniente aclarar qué activación es la que se está mostrando. Los patrones de activación representados son los correspondientes al primero de los 20 *feature maps* de dimensiones 24 x 24 resultantes del procesamiento de convolución. Como cada filtro extrae diferentes características o *features*, de haber decidido mostrar la salida de otro de los 20 *feature maps*, habríamos obtenido otras matrices. Por ejemplo, para el tercer *kernel* se obtienen las siguientes:



Capa de rectificación lineal (ReLu)



El resultado no es muy diferente al obtenido en la capa anterior, pues la función ReLU no modifica los valores positivos, y los negativos los transforma en cero. Visualmente, las matrices resultantes son prácticamente idénticas, pues la intensidad del color negro no varía mucho cuando los valores negativos están próximos a cero. Si observamos directamente las matrices numéricas de uno de los dígitos, vemos que esto es justamente lo que ocurre.

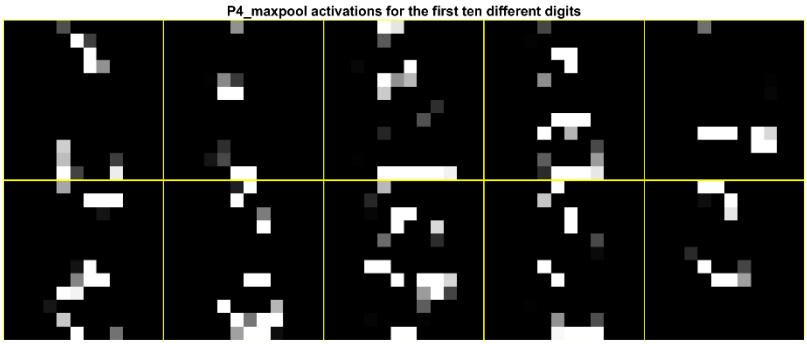
```
convol = activations(convnet,im,'C2_layer');
convol(:,:,1)
```

```
ans = 24×24 single matrix
   -0.0016   -0.0016   -0.0016   -0.0016   -0.0016 ...
   -0.0016   -0.0016   -0.0016   -0.0016   -0.0016
   -0.0016   -0.0016   -0.0016   -0.0016   -0.0016
   -0.0016   -0.0016   -0.0016   -0.0016   -0.0016
   -0.0016   -0.0016   -0.0016   -0.0016   -0.0016
   -0.0016   -0.0016   -0.0016   -0.0016   -0.0016
   -0.0016   -0.0016   -0.0016   -0.0016   -0.0016
   -0.0016   -0.0016   -0.0016   -0.0016   -0.0016
   -0.0016   -0.0016   -0.0016   -0.0016   -0.0016
   -0.0016   -0.0016   -0.0016   -0.0016   -0.0016
      ⋮
```

```
relu = activations(convnet,im,'C3_ReLU');
relu(:,:,1)
```

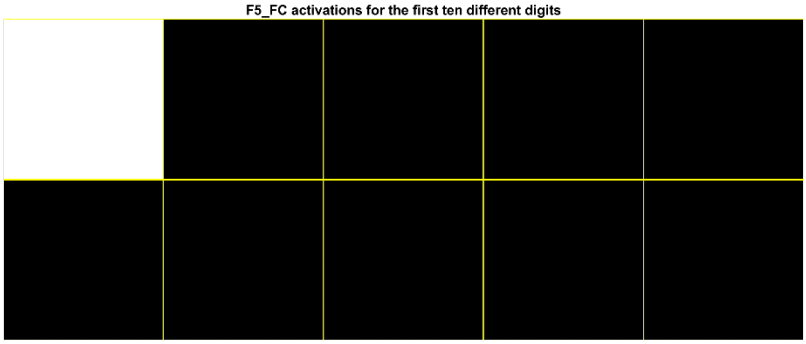
```
ans = 24×24 single matrix
     0     0     0     0     0 ...
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
   ⋮
```

Capa de Max-pooling



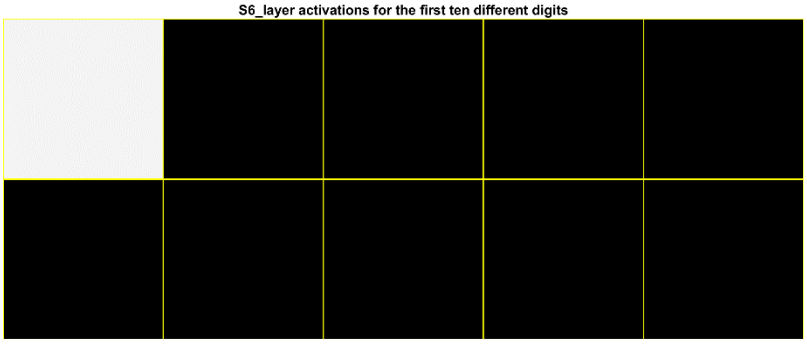
De nuevo, hay que tener en cuenta en qué capa nos encontramos para saber qué es lo que se está representando. Esta capa está muy relacionada con la de convolución, por lo que las dimensiones son parecidas. El procedimiento de submuestreo se aplica sobre cada *feature map* que se obtiene de la capa de convolución, por lo que hay 20. Aquí se representa la primera de esas 20 matrices resultado del submuestreo. Como el *kernel* en esta capa es de 2 x 2, la matriz de activación será de 12 x 12. En consecuencia, lo que se está viendo en la imagen son las matrices 12 x12 resultado de aplicar *max-pooling* sobre el primer *feature map*.

Capa de conexión total



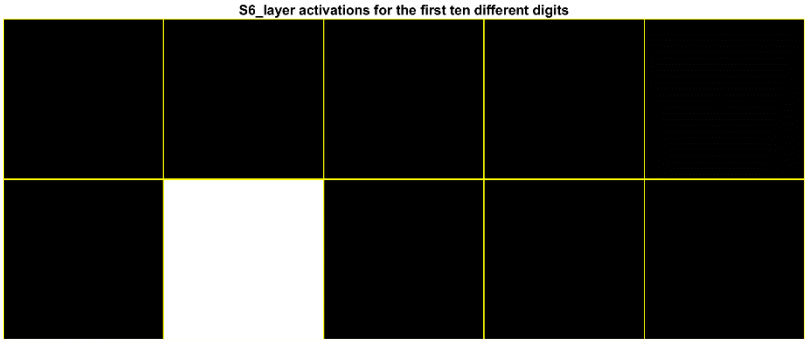
En ésta y las siguientes capas es donde se observan las mayores diferencias, y es importante explicar el motivo. Las dimensiones se han reducido drásticamente con respecto a capas anteriores. Mientras que en la capa de Max-pooling había 12 x 12 x 20 neuronas, ahora solo hay 10. Y mientras que en la capa anterior acceder al primer elemento del arreglo tridimensional equivalía a acceder a la primera de las 20 matrices de 12 x 12, ahora acceder al primer elemento del vector unidimensional equivale a acceder a la primera neurona. Como se trata de una única unidad, solo se representa con un color, de ahí que no haya variedad en la imagen. El motivo por el cual la primera de las neuronas representadas en la imagen es más clara que las demás no es fácilmente interpretable en este momento, pero tiene un significado más claro en las siguientes capas.

Capa Softmax

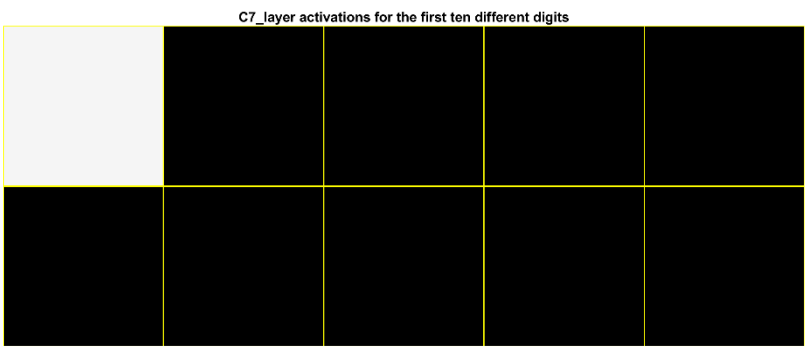


Lo dicho para la capa anterior vale ahora también, pues de nuevo se tienen 10 neuronas. No obstante, ahora resulta algo más fácil entender la distribución de colores (activaciones) en la imagen. La función softmax transforma las activaciones que recibe como entrada en nuevos valores que oscilan entre 0 y 1 que se consideran similares a probabilidades. Como tales, ya están indicando la "evidencia" a favor de que el dígito originalmente presentado sea cada uno de los diez posibles. Mayor evidencia aquí implica mayor activación de las neuronas en la salida de la capa Softmax. Eso es lo que se ve precisamente en la primera de las neuronas de la imagen. Hay que recordar que se está accediendo o mostrando la primera de las diez neuronas de la capa, es decir la activación o probabilidad que corresponde al dígito cero. Como es obvio, y dado que la red clasifica bien estos dígitos, cuando se presenta un cero, esta activación es alta, pero cuando se presenta cualquier otro número, la activación de esta primera neurona es muy baja. Es por eso por lo que el resto de las imágenes son negras.

Si en vez de solicitar la activación de la primera neurona se mostrase el de la séptima, correspondiente al número seis, se vería que la neurona con un color más claro es precisamente la que ocupa la posición de este mismo número, como se ve a continuación.



Capa de clasificación



En este caso, las activaciones representadas *no* son las de la capa de clasificación, pues realmente no hay activaciones que mostrar, la salida de esta capa es un único número. Si examinamos el *output* que se obtiene al aplicar la función `activations`, vemos que se trata del vector de activaciones de la capa Softmax. De todas formas, es posible representar la salida de esta capa de otra manera, pues sabemos que esta etapa se caracteriza por la utilización de la función de pérdida entropía cruzada sobre cada elemento de la entrada. Como resultado de esta transformación se tiene un vector de diez valores, asociados a cada uno de los diez dígitos. El dígito cuya neurona asociada tiene menor entropía cruzada es el que finalmente se elige y da lugar a la salida. Con algunas líneas de código podemos replicar este proceso (ver archivo digits_processing.mlx). Si calculamos la entropía cruzada de cada neurona para todos los dígitos obtenemos la siguiente matriz (se omiten algunas columnas):

```
all_losses = 10×10
      0.0393   15.9424   15.9424   15.9424   12.9404 ...
      6.8107    0.0983    5.9559   15.9424   11.2485
      8.8065    3.1630    0.0089   15.9424    5.6397
      4.8256    4.7518    9.4745    0.0000    5.2826
     11.7119   13.3217   13.4118   15.9424    4.5283
     15.9424    3.9942   15.9424   15.9424   11.4982
      8.3009   10.3855   15.9424   15.9424    5.6748
      5.9808    3.8129    8.6018   15.9424    9.3683
      3.6399    6.8597    5.1101   15.9424    0.6118
      8.2114    6.7884   15.9424   15.9424    0.8332
```

Cada columna representa las entropías cruzadas de uno de los dígitos, empezando desde el 0. Puede observarse como, de acuerdo con lo esperado, las entropías menores son las que determinan la predicción. De esta manera, cuando el *input* es un cero, la menor entropía se logra en la primera neurona de la capa de clasificación, la que representa al dígito cero. Y si se presta atención a uno de los dígitos que nunca se clasifican bien, como el nueve, se puede ver lo siguiente:

```
ans = 10×1
   15.9424
   12.9275
   15.9424
    4.4096
   13.6999
   15.9424
   14.8059
    4.1213
    0.0360
    4.9655
```

La menor entropía cruzada la obtiene el número ocho, una de las clasificaciones que hace la red cuando se le presentan “nueves”. Y la segunda menor entropía cruzada la tiene la neurona correspondiente al número tres. Y esto es nuevamente consistente con los datos que tenemos, pues en cuatro de los cinco ejemplares clasifica el nueve como un tres. Este sesgo también se ve en este ejemplo, pues incluso cuando no se clasifica como un tres, sino como un ocho, existe una cierta “confianza”, dicho de manera muy informal, en que el dígito podría ser también un tres.