NEURAL NETWORKS Y CONVOLUTIONAL NETWORKS EN MRI

Javier Castro Medina

1 Introducción

Se muestra y compara el desempeño de distintas **Neural Networks** (NN) y **Convolutional Neural Networks** (CNN) en la clasificación de imágenes en el contexto de la medicina. Se cuenta con un set de 3064 imágenes de resonancia magnética de tumores cerebrales, las cuales se clasificaron en 3 categorías: Glioma, Meningioma y Pituitario. El fin de este trabajo es encontrar y estudiar la dependencia que existe entre la estructura de las redes y la accuracy que estas pueden llegar a alcanzar.
Se tratará de usar el nombre original de los conceptos, la mayoría (todos) en inglés, aunque a veces se usará su traducción.

2 Marco teórico

Según la OMS (*link*) un tumor se define como un conjunto de células que crece de forma anómala en algún órgano o tejido del cuerpo presentando un comportamiento fuera de lo común, cuando estas células se forman en el cerebro se le denomina tumor cerebral. De este último, según localización, se desprenden las tres categorías mencionadas en la introducción (Glioma, Meningioma y Pituitario) y es en esta clasificación que se aplicarán y compararán las herramientas de aprendizaje.

A continuación se hará una pequeña y somera revisión de lo que es una NN y una CNN. Una NN es un método que tiene como fin aproximar funciones que van desde un espacio de dimensión finita a otro a través de composiciones de funciones lineales afín [2], bajo esta definición este método cuenta con varios campos de aplicación (finanzas, medicina, industria, computación, etc).

Supongamos que queremos modelar un sistema de la forma $f(x)$, $x \in \mathbb{R}^L$ para lo cuál contamos con datos $D = (x_i, f(x_i))_{i=1}^N$. La forma más simple de querer aproximar f es a través de una función lineal afín de la forma $l(x, w, b) = b + w^T x$, a la cual le asignamos el costo $c(w, b, D)$, que depende de los datos. Luego, la idea es minimizar c hasta encontrar w^*, b^* óptimos y representar f como $l(\cdot, w^*, b^*)$. Para que estas redes puedan lograr un mejor comportamiento frente a funciones f con fuertes no linealidades, se suele pasar las funciones $l(x, w, b)$ por funciones de activación del estilo sigmoide o relu. El grafo que se observa en 1a, justifica de alguna forma el nombre que se le da a este método.

Por otro lado, una CNN es una NN que se suele ocupar cuando los datos son imágenes. Estas redes tratan de ir aprendiendo los features a medida que también van entrenando la clasificación (o para lo que se esté usando), esto se expresa en un conjunto de layers al inicio de la red que aplican una operación, que llaman convolución, la cual tiene como fin obtener una matriz de features de la imagen original. En este documento nos referimos a estas layers como convolutional layers y asumiremos que consisten en la aplicación de la convolución, operación que se basa en aplicar una especie de producto punto entre un tensor (matriz $3D$) de dimensiones $filters \times q \times q$ ($filters$ y q pasan a ser un parámetros) y las submatrices de tamaño $q \times q$ en el input para generar el tensor output. Luego este output se pasa por un *max pooling* que consiste en dividir el tensor input en bloques de tamaño $p \times p$ (p pasa a ser un parámetro) tomar el máximo de este y retornar el tensor compuesto por estos máximos, notar que el max pooling y la convolución reducen el tamaño del input. Luego, este tensor se "aplana" y se pasa por una NN clásica como la mencionada anteriormente. Este proceso previo permite que los parámetros dedicados al feature de la imagen también se vayan mejorando en el desarrollo de la optimización.

3 Desarrollo

3.1 Datos y parámetros

Los datos (*link*) se obtuvieron en formato `.mat` por lo tanto fue necesaria una transformación sobre estos (llevarlo a `Python`) para obtener el par (M, y) dónde M es una matriz de 512×512 e $y \in \{0, 1, 2\}$ representa el *label* de cada imagen. Se asumirá que 0 representa la clase $Meningioma$, 1 a la clase *Glioma* y 2 a la clase *Pituitario*. Otro problema encontrado en la preparación de los datos, además del formato, fue una cierta cantidad de datos "dañados" que producían error al aplicar los métodos de la librería `Tensorflow` y es por esto que se determinó eliminar simplemente los datos problemáticos lo que llevó a la eliminación de 45 datos que nos deja un total de 3019 (ver tabla 1).
La cantidad total de datos se divide en tres conjuntos, uno con el 70% del total que será destinado a entrenamiento, otro con el 15% usado como datos de testeo y el 15% restante será de validación.
Fijaremos acá la herramienta de optimización y la función de activación, se usará el algoritmo *Adam* (adaptive mo-

ment estimation) [3] puesto que esto nos quita de encima la tarea de calibrar el learning rate. En cuanto a la función de activación, se declinó por usar relu por simpleza y puesto que este no es un aspecto central del estudio.

Class	Amount of data
Meningioma	708
Glioma	1426
Pituitario	885
Total	3019

Table 1: Data

3.2 Neural Network

Notemos que tenemos libertad de variar la estructura de la red en cuanto a cantidad de layers y nodos por layer. La primera NN que usaremos es del estilo que aparece en la figura 1a y nos referiremos a esta como NN1. La única hidden layer se conectará con todos los nodos de las layers adyacentes (dense layer) y no haremos extracción de features en este primer intento, por esto la input layer cuenta con un total de $L_1 = 512 \cdot 512$ nodos mientras que la hidden layer tendrá $L_2 = 100$ nodos con lo cuál la cantidad de parámetros en esta layer es de $p_2 = (L_1+1)L_2$ y finalmente la output layer tiene $L_3 = 3$ nodos con lo cuál cuenta con $p_3 = (L_2 + 1)L_3$ parámetros. Naturalmente, $p_1 = 0$.

Todas las hidden layers consideradas serán dense. La primera variación que haremos será agregar una hidden layer a NN1 generando la red NN2 dónde en ambas hidden layers se mantiene la cantidad de nodos en 100. Luego, generamos NN3 y NN4 a partir de NN1 y NN2 bajando la cantidad de nodos a 50 en las respectivas hidden layers.

En cuanto a los hiperparámetros, el batch size se fijará como 64 siguiendo lo hecho en [1] y number of epochs se calibra sobre el set de validación observando la figura 2. Con esta información se decide usar number of epochs como 45 tomando en cuenta el tiempo de ejecución.

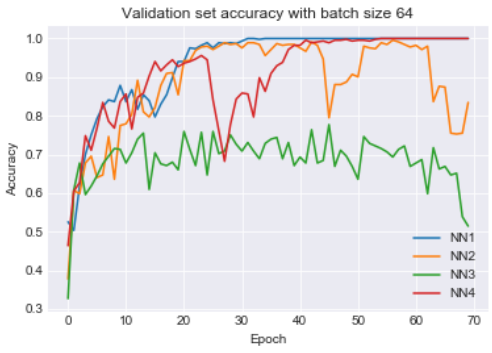


Figure 2: Accuracy of all non-features NN models

3.3 Convolutional Neural Network

Al igual que en la sección anterior, se partirá con una CNN de estructura simple sobre la cuál se irán agregando y/o cambiando parámetros de las convolutional layers. Empíricamente se observó que este tipo de redes poseen un tiempo de entrenamiento notoriamente mayor que las NN, es por esto que se decide disminuir el batch size a 32 y tomar number of epochs igual a 5.
La layer de clasificación será la misma en todas las CNN consideradas: una layer que "aplana" la matríz de features, una dense layer con 64 nodos, número que se escoge sólo por simpleza, y finalmente la output layer con 3 nodos.

La primera CNN es del estilo de la figura 1b y nos referiremos a esta como CNN1, luego creamos CNN2 agregando dos convolutional layers más sobre esta estructura. En ambas CNN, sólo por ser funcional, se usarán los parámetros $filters$ igual a 32, $q = 3$ y $p = 4$. Finalmente, CNN3 tendrá sólo una convolutional layer como las anteriores, notar que esto hará crecer la cantidad de parámetros puesto que hay solamente un proceso de max pooling.

4 Resultados

4.1 Neural Networks

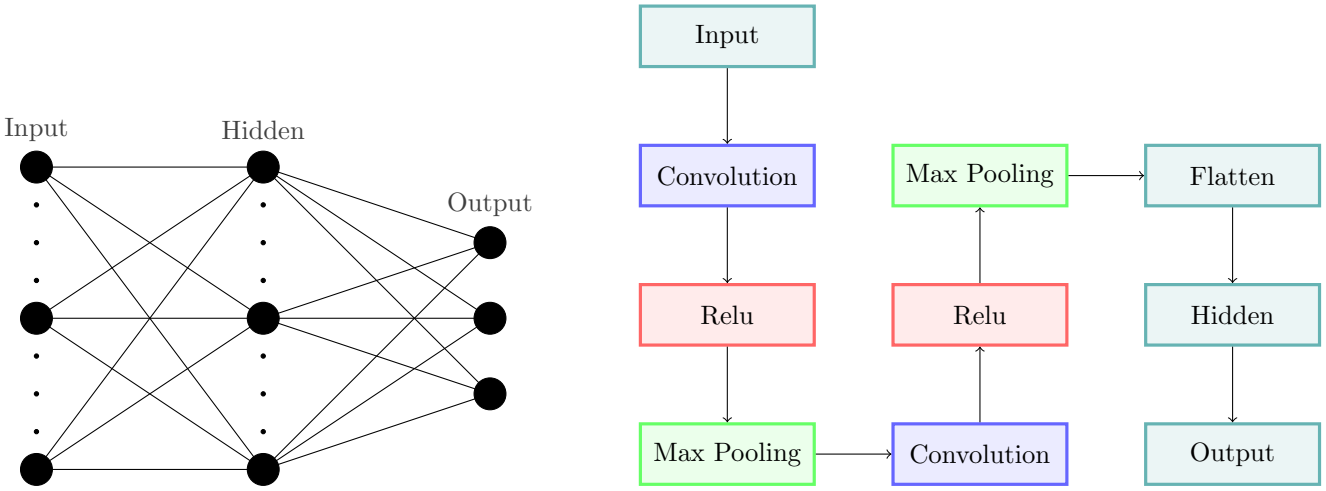
A continuación mostramos la accuracy alcanzada por cada modelo sobre los datos de entrenamiento y testeo además del tiempo de ejecución.

Model	Test Acc.%	Train Acc.%	Time
NN1	84	97	$9m\ 36s$
NN2	83	88	$9m\ 32s$
NN3	82	97	$6m\ 56s$
NN4	87	97	$6m\ 42s$

Model	Dense layers	Nodes per layer
NN1	1	100
NN2	2	100
NN3	1	50
NN4	2	50

Table 2: NN results

Vemos una clara disminución en el tiempo de ejecución cuando disminuimos la cantidad de nodos en cada layer de 100 a 50, esto se explica por el hecho de que se disminuye notoriamente la cantidad de parámetros lo que hace que el algoritmo de optimización tarde menos tiempo. Cabe señalar que la disminución de parámetro fue cercana a la mitad, se pasó de $\sim$ 26 a $\sim$ 13 millones de parámetros (se habla de parámetros de optimización). En cuanto a la columna de entrenamineto, vemos que todas las estructuras alcanzan una accuracy similar salvo la NN2 que sólo alcanza el 88% siendo esta la más grande tanto en cantidad de hidden layers como nodos en cada una de estas. Lo anterior se replica en la columna de testeo puesto que una de



(a) La input layer y la hidden layer no tienen necesariamente la misma cantidad de nodos.

(b) Los nodos Convolution-Relu-Max Pooling representan a una convolution layer.

Figure 1: Neural Networks

las estructura más simple, NN4, tiene el mejor desempeño lo cuál nos dice que para este problema una estructura mas o menos simple sería una mejor decisión. Otra observación es que con más cantidad de nodos, es preferible usar menos layers y en el otro sentido, si se ocupan pocos nodos, es mejor usar más layers.

4.2 Convolutional Neural Networks

A continuación mostramos los resultados obtenidos para los modelos CNN.

Model	Test Acc.%	Train Acc.%	Time
CNN1	85	89	$26m\ 22s$
CNN2	71	73	$25m\ 52s$
CNN3	51	47	$24m\ 17s$

Model	Parameters	Conv. layers
CNN1	1.9×10^6	2
CNN2	30×10^3	4
CNN3	33×10^6	1

Table 3: CNN results

Podemos observar que la accuracy de los métodos es similar al pasar de los datos de testeo a los datos de entrenamiento, además se mantiene el orden entre estos. Lo anterior también se pudo observar durante el proceso de optimización como se aprecia en la figura 3. Vemos también, que la accuracy no aumenta a medida que aumentamos la cantidad de convolutional layers como podría haberse esperado sino que se alcanza un óptimo cuando se toman dos convolutional layers (CNN1). El peor rendimiento lo presenta CNN3 que sólo llega al 51% sobre el set de testeo, aún así alcanza mayor accuracy que sobre los datos de entrenamiento. En cuanto al tiempo de ejecución y la cantidad de parámetros tampoco se observa la tendencia esperada, que sería mientras más parámetros más tiempo, puesto que al comparar los tiempo de CNN2 con CNN3 o CNN1 con CNN3, vemos que la estructura con más parámetros, CNN3, tarda menos tiempo. Incluso menos tiempo que CNN2 que cuenta con mucho menos parámetros por ser la red con más convolutional layers.

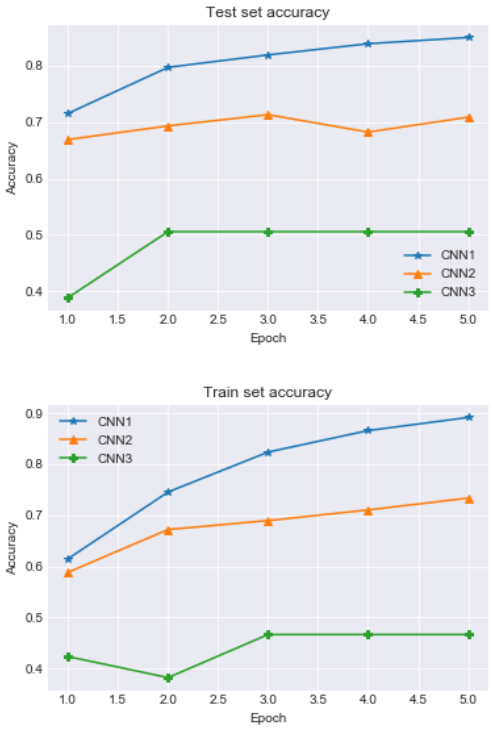


Figure 3: CNN accuracy level vs epoch during training

Una observación importante en cuanto a la figura 3 es

que la curva de CNN3 pareciera quedarse pegada sobre un valor de accuracy tanto en los datos de entrenamiento como en los datos de testeo lo cuál podría explicar su bajo desempeño. Esto se puede deber a que el algoritmo de optimización se estanca en un mínimo local lo cuál nos dice que quizás esta red no alcanza su mejor rendimiento con el algoritmo *Adam* y por lo tanto se lograría una mejor comparación con otra herramienta de optimización.

5 Conclusiones

Si bien los datos tenían un contexto bien definido, la clasificación de tumores según MRI, el proyecto tomó una dirección más dedicada a la comparación de los distintos métodos disponibles. Aún así, las redes entrenadas permiten clasificar las imágenes y determinar a la clase de tumor que pertenece con cierto grado de certeza. Una mejora para este proyecto sería partir desde un set de datos menos pesados, con el fin de optimizar tiempos, y tomar en cuenta una cantidad más grande y diversa de redes para la clasificación, también estudiar más a fondo la dependencia por ejemplo de la cantidad de layers y nodos por layers con la accuracy y tiempo de ejecución. De igual manera, se podría estudiar la extracción de features y ver como afecta esto sobre las NN.

Otro aspecto importante a destacar es la dificultad que se presenta al principio del estudio con el formato de los datos y que parte de estos presentaban problemas al ser usados por `Tensoflow`. Los datos venían en formato `.mat` como *structure array* de `Matlab` y por lo tanto se tuvo que investigar cómo transformar estos a arreglos simples de `Python`. Por otro lado, para descartar los datos problemáticos se hizo una limpieza a mano para descubrir los índices de estas imágenes en el arreglo de datos. Esto permitió usar `Tensoflow` sin problemas, salvo los problemas típicos que surgen al usar librerías de este estilo por primera vez.

En cuanto a los resultados del proyecto y la clasificación, se llega a la conclusión de que existe una fuerte dependencia entre la estructura de la neural network y su desempeño sobre estos datos. Se pasa por redes que alcanzan $\sim 50\%$ de accuracy hasta redes que bordean el 90%, como es el caso de la mejor estructura estudiada NN4 tanto en tiempo y accuracy. Importante destacar que estas redes son óptimas en cierto modo para este set de datos, por lo tanto para otro estudio habría que recalibrar parámetros y redes.

6 Video de la presentación

El video de la presentación se puede ver en este `enlace`.

References

[1] Ali Mohammad Alqudah, Hiam Alquraan1, Isam Abu Qasmieh, Amin Alqudah and Wafaa Al-Sharu, Brain Tumor Classification Using Deep Learning Technique - A Comparison between Cropped, Uncropped, and Segmented Lesion Images with Different Sizes, International Journal of Advanced Trends in Computer Science and Engineering, 2019.

[2] Kur Hornik, Maxwell, Stinchcombe and Halbert White, Multilayer Feedforward Networks Are Universal Approximators, Neural Networks 2, no. 5, 359–366, 1989.

[3] Diederik P. Kingma, Jimmy Lei Ba, Adam: A Method For Stochastic Optimization, 2017.