Proyecto 5

1st Ian Augusto Cortez Gorbalan
ian.cortez@utec.edu.pe
970895554

2nd Plinio Matías Avendaño Vargas
plinio.avendano@utec.edu.pe
927144823

I. Introducción

Este proyecto busca emplear una Red Neuronal Convolucional o CNN para clasificar imágenes de un *dataset* real y evaluar el rendimiento del modelo generado. El *dataset* que se va a utilizar contiene 21165 imágenes de radiografías pulmonares separadas en 4 categorías diferentes las cuales indican si esta pertenece a un paciente con COVID, opacidad de pulmón, neumonía viral o ninguno. Hay 3616 imágenes en la categoría de COVID, 6012 en la categoría de opacidad de pulmón, 1345 en la categoría de neumonía viral y 10192 en la categoría normal que indica que el paciente al cual le pertenece esta sano. El objetivo principal del proyecto es desarrollar un modelo eficiente que sea capaz de clasificar apropiadamente las imágenes bajo las categorías existentes para el *dataset* que se va a usar. Como objetivo secundarios se tiene, primero, realizar pruebas con el modelo desarrollado para evaluar que modificaciones a este permiten entrenar mejor el modelo. Segundo, comparar el *accuracy* obtenido después de realizar las modificaciones al modelo.

II. Explicación

A. CNN

Una Red Neuronal Convolucional o CNN es un algoritmo de *deep learning* supervisado que toma como valor de entrada una imagen. La CNN procesa esta imagen usando filtros para capturar las características de la imagen y poder reconocer patrones de las mismas.

La arquitectura fundamental de esta se puede dividir en dos partes.

1) Feature Learning: Para entender mejor esta sección de la topología de la red, es pertinente recalcar que principalmente buscamos reconocer patrones correspondientes a objetos, por lo que es lógico mirar como están ubicados los píxeles cercanos entre si, de igual manera que sera menos relevante como se relacionan los píxeles lejanos, esta intuición se puede traducir matemáticamente a computar la suma ponderada de un determinado espacio n-dimensional a esto se le conoce como *kernel*, además de las características ya mencionadas el uso de las convoluciones (aplicar el *kernel* sobre la imagen) permiten reducir las dimensiones de la imagen original, cabe destacar que el input de la convolución como mínimo es el *size* del *kernel*, y número de entrada como salida de los canales (valores por píxel) a mayor cantidad de canales (o filtros) más aumenta la complejidad del modelo.

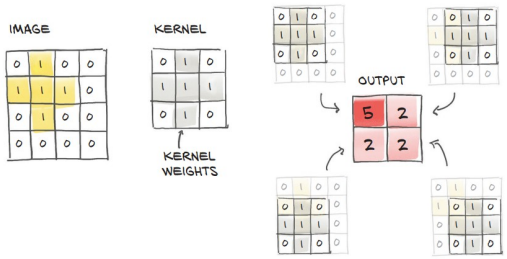

Fig. 1: *Visualización de una convolución*

Posteriormente se aplica, una función de activación (generalmente relu o tanh) con el fin de aumentar la no-linealidad. Para finalizar, generalmente se aplica una función de *pooling*, esto es para reducir aun más la imagen, de tal manera que se conserven las características más significativas.

También es relevante rescatar, que se irán modificando los *kernels* respecto a la gradiente, con el fin de reducir el error.

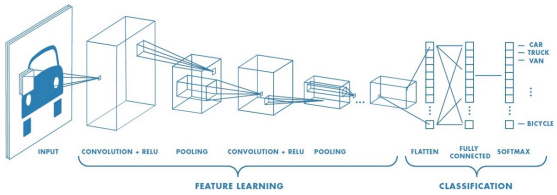

Fig. 2: *Visualización de una CNN*

2) Classification: En esta parte, realizamos un aplanamiento (conversión a un vector) del resultado entregado por la sección anterior, para pasarlo como input a un *multilayer perceptron* el cual realizara la clasificación, no entraremos en mas detalles debido a que el *mlp* fue estudio del proyecto 4.

B. Batch Normalization

Batch normalization es un método de regularización que estandariza los elementos de salida de una capa oculta al momento de ser propagados a la siguiente capa. Este permite transmitir datos entre las capas de forma estable y rápida por lo que puede mejorar la eficiencia y precisión del modelo que la utilice. El proceso de normalización con este método se basa en obtener la media y varianza de los elementos de un *batch* y luego normalizar la *data* del mismo con estos valores. Un efecto secundario de *batch normalization* es que los resultados normalizados están fuertemente relacionados a la media y la varianza de cada *batch* por lo que el modelo podría generar un *overfitting* a los datos de prueba.

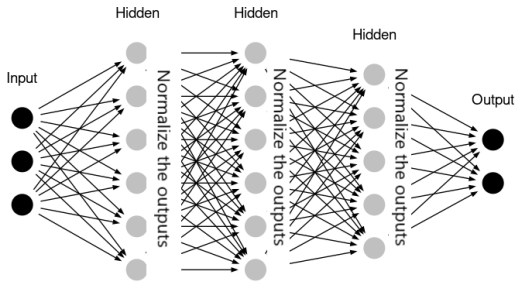

Fig. 3: *Batch Normalization*

C. Dropout

Dropout es una técnica de regularización que consiste en evitar que ciertas neuronas en una capa oculta propaguen sus salidas hacia las neuronas de la siguiente capa oculta. El proceso para elegir que neuronas van a ser ignoradas es completamente aleatorio, pero se puede determinar una probabilidad de ignorar o no una de las neuronas dentro de una capa oculta. Ignorar ciertas salidas dentro de una capa va a obligar a disminuir la complejidad del modelo, lo que es un buen método para prevenir *overfitting*.

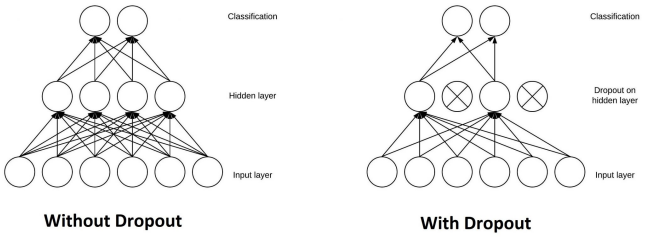

Fig. 4: *Dropout*

D. Early Stopping

Early stopping permite reducir el tiempo tomado por el modelo para realizar el entrenamiento. Consiste en comparar el último error con el penúltimo, si este es mayor, respecto a un hiperparámetro de tolerancia, se culmina el entrenamiento, en nuestro caso, calculamos el accuracy respecto al validation set y se procede de la manera anterior. Adicionalmente, esto permite que el modelo no pierda su precisión y capacidad ya que es posible que en algún momento el error producido por el modelo vuelva a aumentar y pierda el progreso realizado por el entrenamiento.

III. Experimentación

A. Consideraciones previas

El modelo fue entrenado de manera local, ya que al emplear servicios como Google Colab tardaba más de 3 horas, esto es un limitante ya que al contar con tarjetas gráficas promedio, tuvimos que limitar el *batch size* a 32 ya que nuestra vram no soporta un tamaño mayor o que no pudimos graficar el error de la validación vs las épocas del training, ya que de nuevo la ram no permitía la carga de mas fotos.

B. Hiper-parámetros

Se empleo un *learning rate* de 0.01 para entrenar el modelo. Asimismo, se va a realizar el entrenamiento en 20 épocas. Además, como se menciono anteriormente se empleara un *batch size* de 32.

C. Separación del dataset

El *dataset* original se proceso y se dividió en 3 partes. Se uso un 70% del *dataset* original para entrenar el modelo, 20% de este como validación del modelo y 10% del mismo para *testing*. Se tomaron los elementos de las particiones de forma aleatoria.

D. Experimento 1

Para el primer experimento, se utilizo la siguiente arquitectura de 4 capas. Cabe destacar, que en este modelo se utilizo una cantidad de canales de mayor a menor. La primera con 3 canales de entrada, 128 canales de salida, un *kernel* de 8, *stride* de 1 y *padding* de 2. La segunda con 128 canales de entrada, 64 canales de salida, un *kernel* de 4, *stride* de 1 y *padding* de 2. La tercera capa con 64 canales de entrada, 32 canales de salida, *kernel* de 3, *stride* de 1 y *padding* de 2. La cuarta capa con 32 canales de entrada y de salida, *kernel* de 3, *stride* de 1 y *padding* de 2. Cada una de las capas pasa por una función de activación RELU y este resultado pasa por un *Max Pool* 2D para después ser propagado a la siguiente capa.

Al realizar las pruebas, se obtuvo la siguiente gráfica para el error de este modelo.

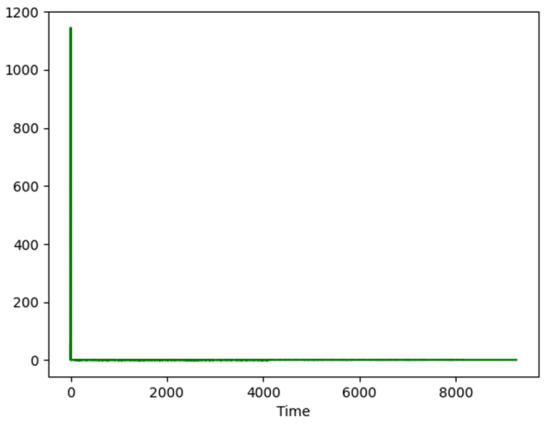

Fig. 5: Gráfica del experimento 1

Después de entrenar el modelo, se obtuvieron los siguientes valores para el *accuracy* modelo con los *datasets* de *testing*, *validation* y *training*.

Dataset de Prueba	*Accuracy*
Testing	47.856%
Validation	48.11%
Training	48.15%

De esta tabla se puede evidenciar que el modelo probado no provee información relevante y certera para la clasificación de radiografías pulmonares. Asimismo, el modelo no es capaz de aprender bien con los datos de *training* por lo que tampoco

se ajusta al *testing* y *validation*, por lo que concluimos firmemente que recae en un *underfitting*.

E. Experimento 2

Para el segundo experimento, principalmente se invirtió la estructura de los filtros del primer experimento, es decir de menor a mayor,empleando igual cantidad de capas, también se eliminó el padding en la primera capa ya que no era necesario conservar los valores de los bordes. La primera con 3 canales de entrada, 16 canales de salida, un *kernel* de 10, *stride* de 1 y *padding* de 0. La segunda con 16 canales de entrada, 32 canales de salida, un *kernel* de 6, *stride* de 1 y *padding* de 2. La tercera capa con 32 canales de entrada, 64 canales de salida, *kernel* de 3, *stride* de 1 y *padding* de 2. La cuarta capa tiene 64 canales de entrada y 128 de salida, un *kernel* de 3, *stride* de 1 y *padding* de 2. A diferencia del modelo empleado para el experimento 1, en este experimento cada capa menos las primera pasa los resultados por una función RELU y hace un *Max Pool* 2D para propagar los valores a la siguiente capa, cabe resaltar que las últimas dos capas poseen un kernel size de 3*3 inspirado en la arquitectura postulada en el paper ”Convolutional neural networks in medical image understanding: a survey”.

Al realizar las pruebas en este experimento, se graficó el error obtenido al entrenar el modelo.

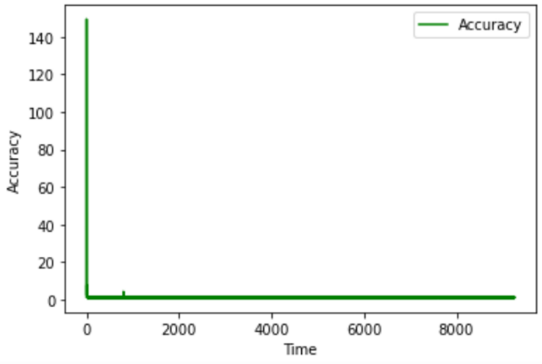

Fig. 6: Gráfica del experimento 2

Después de entrenar el modelo, se obtuvieron los siguientes valores para el *accuracy* con los *datasets* de *testing*, *validation* y *training*.

Dataset de Prueba	*Accuracy*
Testing	48.5%
Validation	48.955%
Training	47.956%

De esta tabla se puede evidenciar que el modelo probado no provee información relevante y certera para la clasificación de radiografías pulmonares. Asimismo, el modelo no es capaz de aprender bien con los datos de *training* por lo que tampoco se ajusta al *testing* y *validation*, sin embargo este modelo es ligeramente superior al anterior por lo que conservaremos esa estructura en las capas.

F. Experimento 3

Para el tercer experimento, se agregó una función de la librería *Pytorch* para utilizar el *batch normalization*. Esto se aplicó a la primera capa de la arquitectura mostrada en el experimento 2. En base al error generado por el modelo al momento de entrenar se obtuvo la siguiente gráfica.

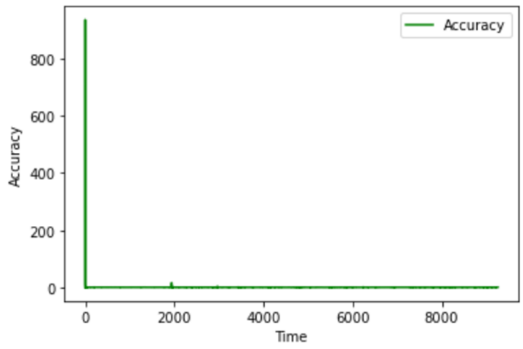

Fig. 7: Gráfica del experimento 3

Al igual que para los experimentos anteriores, se hizo una tabla con los valores para el *accuracy* de los *datasets* de *testing*, *validation* y *training*.

Dataset de Prueba	*Accuracy*
Testing	59.62%
Validation	59.319%
Training	58.015%

En este experimento se puede evidenciar que ha habido una mejora respecto a los modelos probados en los experimentos anteriores. Lo más destacable de este experimento es que el *accuracy* para los datos de *training* es ligeramente menor que para los datos de *testing* y *validation*. Sin embargo, el modelo todavía no es capaz de clasificar las radiografías de forma eficiente y apropiada ya que el *accuracy* todavía es bastante bajo.

Intuimos que esto es debido a que el modelo recae en el problema de *vanishing gradient*, que es cuando debido a las funciones de activación las gradientes se aproximan a 0, imposibilitando un correcto entrenamiento, al normalizar los outputs se prevee esto.

G. Experimento 4

Para el cuarto experimento, se utilizó la arquitectura del experimento 2 y se agregó una función de *batch normalization* a cada una de las capas del modelo. A continuación, se muestra la gráfica del error al entrenar el modelo.

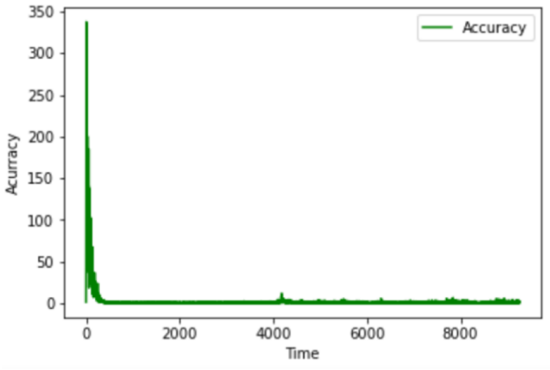

Fig. 8: Gráfica del experimento 4

Al igual que para los experimentos anteriores, se hizo una tabla con los valores para el *accuracy* de los *datasets* de *testing*, *validation* y *training*.

Dataset de Prueba	*Accuracy*
Testing	83.75%
Validation	82.305%
Training	95.619%

En este experimento ha habido una mejora considerable a los experimentos realizados anteriormente. En el modelo actual, se puede afirmar que es posible clasificar las radiografías de forma precisa. Algo que destacar es que existe un ligero *overfitting* producido ya que el *accuracy* para los datos de *training* es mayor que para los datos de *validation* y *testing*, podemos concluir que el modelo efectivamente padecía de *vanishing gradient* .

H. Experimento 5

Para el quinto experimento, se utilizó la arquitectura del experimento 4 y se le agregó una función de *early stopping* para reducir el tiempo que se toma en entrenar el modelo. Se obtuvo la siguiente gráfica con el erro producido por el entrenamiento del modelo.

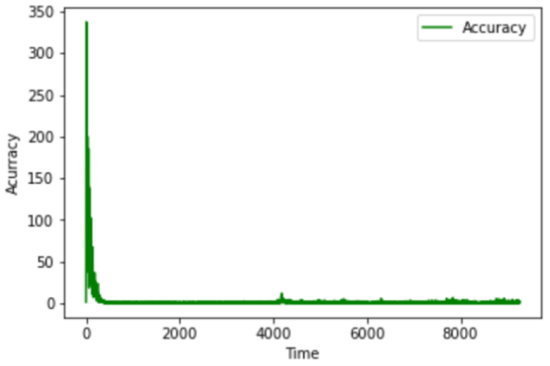

Fig. 9: Gráfica del experimento 5

Se debe mencionar que esta gráfica es igual a la del experimento 4 ya que la convergencia para estos dos modelos fue la misma. Asimismo, se hizo una tabla con los valores para el *accuracy* de los *datasets* de *testing*, *validation* y *training*.

En este experimento se obtuvo la misma precisión (la ínfima diferencia es debido a la semilla) que para el modelo probado en el experimento 4. Asimismo, el comportamiento es prácticamente idéntico por lo que podemos determinar que el *early stopping* agregado no ha sido de mucha ayuda para optimizar el modelo ni evitar el *overfitting* causado por el modelo.

Dataset de Prueba	*Accuracy*
Testing	82.775%
Validation	82.305%
Training	95.555%

I. Experimento 6

Para el sexto experimento, se utilizo la arquitectura del experimento 2 y se agregó una función de la librería *Pytorch* para utilizar el *dropout*. En cada una de las capas, hay un 25% de probabilidad de ignorar las salidas de una neurona. Esto se aplicó a cada una de las capas del modelo. Posterior al entrenamiento, se obtuvo la siguiente gráfica de error.

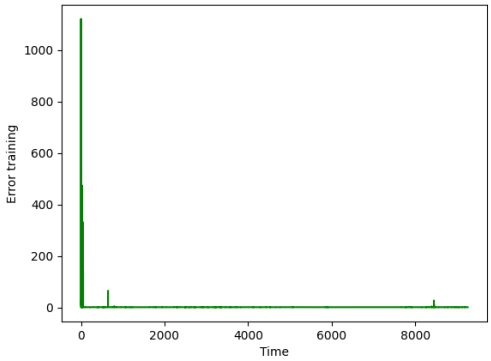

Fig. 10: Gráfica del experimento 6

Al igual que para los demás experimentos, se hizo una tabla con los valores para el *accuracy* de los *datasets* de *testing*, *validation* y *training*.

Dataset de Prueba	*Accuracy*
Testing	48.11%
Validation	48.115%
Training	47.857%

En este experimento se obtuvo una precisión mucho menor que para los dos últimos modelos. En este caso, al igual que para el experimento 3, el *accuracy* de los datos de *training* es menor que para los datos de *testing* y *validation*. Del mismo modo, podemos decir que este modelo no es apropiado para clasificar las radiografías apropiadamente, por lo que podemos concluir que recae en un *underfitting*.

J. Experimento 7

Para el séptimo experimento, se modificó el modelo empleado para el experimento 6 al implementar el *early stopping* y reducir el tiempo requerido para entrenar el modelo. Se mantiene el 25% de probabilidad de ignorar las salidas de una neurona. A continuación, se muestra la gráfica de error para el entrenamiento del modelo.

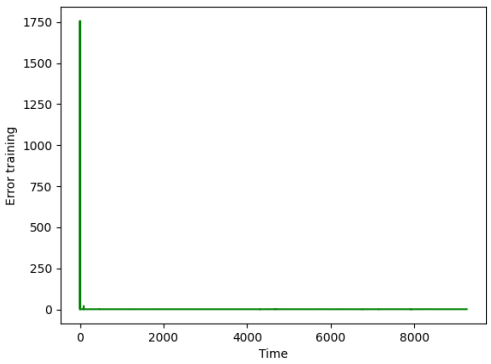

Fig. 11: Gráfica del experimento 7

A continuación, se muestra la tabla con los valores para el *accuracy* de los *datasets* de *testing*, *validation* y *training*.

Dataset de Prueba	*Accuracy*
Testing	48.11%
Validation	48.155%
Training	47.857%

En este experimento, se han obtenido un valor de *accuracy* casi igual al del experimento 6. Asimismo, se puede ver que la gráfica se ve ligeramente alterada en cuanto a comportamiento pero el valor máximo del eje X, que representa el tiempo de entrenamiento del modelo, no se ha reducido. En base a esto se puede deducir que el *early stopping* no ha afectado en gran medida la eficiencia y calidad del modelo.

K. Experimento 8

Para el octavo experimento, se utilizó el modelo del experimento 6 y se le agregó *batch normalization* a cada una de las capas. Asimismo se mantiene que para cada una de las capas, hay un 25% de probabilidad de ignorar las salidas de una neurona. Después de entrenar el modelo, se obtuvo la siguiente gráfica de error.

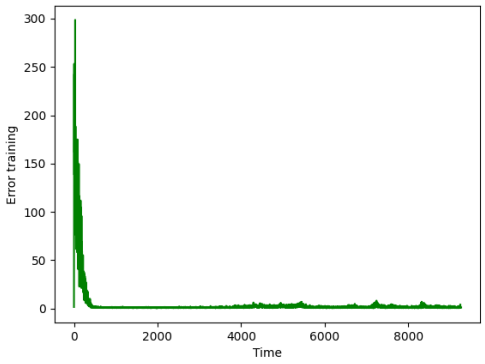

Fig. 12: Gráfica del experimento 8

Al igual que para los demás experimentos, se hizo una tabla con los valores para el *accuracy* de los *datasets* de *testing*, *validation* y *training*.

Dataset de Prueba	*Accuracy*
Testing	68.282%
Validation	67.44%
Training	67.087%

Con los resultados de este experimento se puede evidenciar que ha habido una mejora al implementar el *batch normalization* pero no se ha logrado superar o igualar a la precisión obtenida por el modelo probado en el experimento 4 y 5. Asimismo, se debe mencionar que este modelo es ligeramente mejor para clasificar las radiografías del *dataset* pero no debería ser usado por encima de los modelos del experimento 4 y 5.

L. Experimento 9

Para este último paradigma, se utilizó el modelo del experimento 8 y se le agregó la función de *batch normalization* a cada una de las capas. Además, se mantiene el 25% de probabilidad de ignorar las salidas de una neurona para cada capa. A continuación, se muestra la gráfica de error producida por el entrenamiento del modelo.

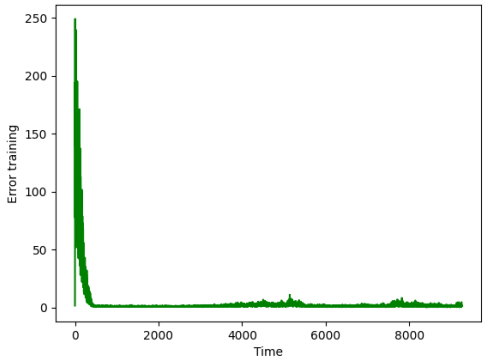

Fig. 13: Gráfica del experimento 9

A continuación, se muestra la tabla con los valores para el *accuracy* de los *datasets* de *testing*, *validation* y *training*.

Dataset de Prueba	*Accuracy*
Testing	78.187%
Validation	77.704%
Training	79.433%

Los resultados muestran una mejor respecto al modelo empleado en el experimento 8. En esta ocasión, se puede apreciar en la gráfica que el comportamiento de la gráfica se volvía errático durante toda la duración del entrenamiento del modelo. Esto probablemente llevó a que el *early stopping* no apoye mucho a mantener un modelo que funcione apropiadamente para clasificar el *dataset*.

IV. Conclusiones

En conclusión, se ha determinado que el modelo que permite clasificar los elementos del *dataset* es el entrenado en el experimento 4 compuesto con 4 capas cada una con *batch normalization*. Cabe destacar que este modelo, presenta un ligero

overfitting. En el caso de que se emplee para diagnósticos reales, este no sería capaz de determinar con certeza que enfermedad presenta una persona. Además, se puede concluir que emplear un modelo que utilice *dropout* no proporciona una mejora para estos modelos en general a menos que se utilicen métodos de normalización adicionales como *batch normalization*, además logramos detectar *vanishing gradient* como se observo en el experimento 3.

REFERENCES

[1] Stevens, E., Antiga, L., & Viehmann, T. (2020). Deep Learning with Pytorch: Build, Train, and Tune Neural Networks Using Python Tools. Manning Publications.

[2] Brownlee, J. (2019, 6 agosto). A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. Machine Learning Mastery. https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

[3] Brownlee, J. (2019, agosto 6). A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks. Machine Learning Mastery. https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/

[4] Huber, J. (2021, 28 noviembre). Batch normalization in 3 levels of understanding - Towards Data Science. Medium. https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338

[5] Sarvamangala, D. R., Kulkarni, R. V. (2021). Convolutional neural networks in medical image understanding: a survey. Evolutionary Intelligence. Published. https://doi.org/10.1007/s12065-020-00540-3