Redes Neuronales - Trabajo Final Integrador #3: Clasificador convolucional profundo sobre Fashion-Mnist.

Reinaldo Magallanes Saunders*
Facultad de Matemática, Astronomía, Física y Computación, Universidad Nacional de Córdoba
M.Allende y H. de la Torre - Ciudad Universitaria, X5016LAE - Córdoba, Argentina
(Dated: 10 de agosto de 2022)

I. INTRODUCCIÓN

Con el crecimiento del poder computacional, el fácil acceso a grandes conjuntos de datos y la posibilidad del uso de GPUs, se vieron grandes avances en el desarrollo y aplicaciones de las redes neuronales artificiales. De estos avances, el aprendizaje profundo, o *deep learning*, en redes neuronales ha de ser de los más importantes debido a la gran diversidad de aplicaciones: reconocimiento de voz, clasificación de imágenes, procesamiento de lenguaje natural, sistemas de recomendación, entre otros[1].

Las redes neuronales artificiales, inspiradas en el cerebro, son capaces de aprender y generalizar a través de la experiencia; aprenden de ejemplos y capturan relaciones funcionales entre datos, incluso si las relaciones subyacentes son difíciles de describir o desconocidas, para luego hacer predicciones sobre datos nunca antes vistos. Su éxito está en que, por el teorema de universalidad, son capaces de aproximar cualquier función, por más arbitraria que sea, y son capaces de modelar sin conocimiento previo sobre las relaciones entre las variables de entrada y salida. El aprendizaje profundo, una de las áreas del machine learning, aplica algoritmos de aprendizaje que se basan en redes neuronales artificiales con representation learning. Esto implica que la red neuronal artificial no solo hace de modelo predictivo, sino que también aprende las representaciones relevantes de los datos.

El aprendizaje profundo moderno provee un marco de gran utilidad para aprendizaje supervisado. Las redes feedforward profundas, con dos o más capas ocultas, son el son el modelo de aprendizaje profundo por excelencia. Ya sea agregando capas o neuronas a esas capas, una red profunda puede representar funciones de complejidad arbitraria, por lo que tiene la capacidad de resolver problemas complejos. Hoy en día, en aprendizaje profundo predominan las redes neuronales convolucionales o CNNs (*Convolutional Neural Networks)* y son utilizadas para atacar la mayoría de problemas debido a su rotundo éxito en aplicaciones prácticas[2].

II. REDES CONVOLUCIONALES

El objetivo de una red feedforward es aproximar alguna función f_*. En particular, para un clasificador, la relación

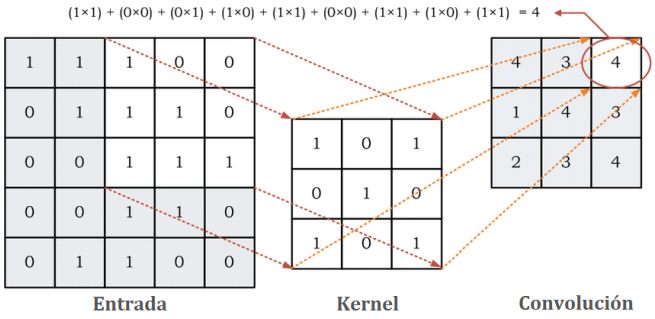

Figura 1: Representación de una convolución bidimensional donde se resalta el cálculo de uno de los elementos del resultado. Adaptación[3].

$\boldsymbol{O} = f_*(\boldsymbol{x})$ mapea una entrada $\boldsymbol{x}$ a una categoría $\boldsymbol{O}$. Una red feedforward permite definir una relación $\boldsymbol{O} = f(\boldsymbol{x}, \boldsymbol{\beta})$ y aprender los valores de los parámetros $\boldsymbol{\beta}$ que resultan en la mejor aproximación de f_*. El conjunto de entrenamiento provee ejemplos aproximados de f_* evaluada en diferentes puntos de entrenamiento y, para cada ejemplo, una etiqueta $\boldsymbol{O} \approx f_*(\boldsymbol{x})$ acompaña a cada entrada $\boldsymbol{x}$ que especifica que se espera obtener una salida similar a $\boldsymbol{O}$. El algoritmo de entrenamiento es el que determina de que manera las capas serán utilizadas para mejor implementar una aproximación de f_*

Las redes convolucionales son una clase particular de redes feedforward, cuya estructura esta diseñada especialmente para procesar datos con estructura de grilla. Los ejemplos típicos de este tipo de dato son las series temporales, las cuales pueden representarse por una grilla unidimensional, y las imágenes, que suelen representarse por una grilla bidimensional de píxeles. El procesamiento de imágenes es una de las principales motivaciones en el desarrollo de las redes convolucionales.

Como el nombre sugiere, las redes neuronales convolucionales son redes neuronales que hacen uso de una operación denominada convolución en al menos una de sus capas. La convolución es una operación $\boldsymbol{x} * \boldsymbol{k}$ que depende de dos objetos, la entrada $\boldsymbol{x}$ y el filtro, o *kernel*, $\boldsymbol{k}$, y da como resultado un objeto con estructura similar a la entrada. En machine learning, estos objetos suelen ser arrays multidimensionales o tensores de datos, donde los elementos del kernel $\boldsymbol{k}$ son parámetros que deben ser optimizados por el algoritmo de entrenamiento. A diferencia de las redes totalmente conectadas, las redes convolucionales en general tienen conectividad dispersa (*sparse connectivity*) y esto se logra exigiendo que el kernel $\boldsymbol{k}$ sea

*Correo electrónico: rei.magallanes@mi.unc.edu.ar

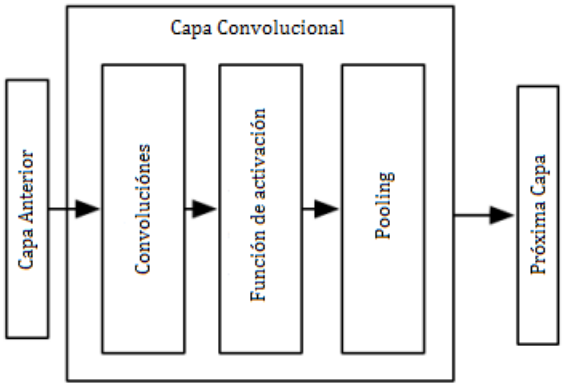

Figura 2: Estructura de una capa convolucional.

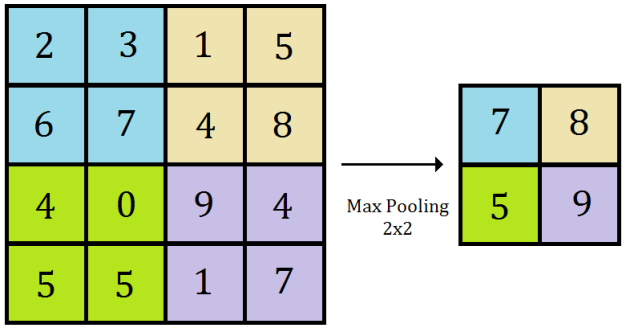

Figura 3: Max pooling 2x2 sobre un tensor bidimensional.

mas pequeño en tamaño que la entrada $\boldsymbol{x}$.

En el caso de tensores bidimensionales, se puede pensar que la convolución coloca el kernel $\boldsymbol{k}$ sobre la entrada $\boldsymbol{x}$ de forma que sus primeros elementos se encuentren superpuestos. Luego, hace una combinación lineal de los elementos del kernel, utilizando los elementos de la entrada que quedan tapados por el kernel como coeficientes. El resultado de esta combinación lineal será el primer elemento del tensor $\boldsymbol{x} * \boldsymbol{k}$. Este proceso se repite moviendo el kernel, asegurándose de recorrer por completo la entrada sin que el kernel escape. En la Figura 1 se muestra el resultado de una convolución de estas características, junto con el cálculo explícito de uno de los elementos. Notar que la convolución no preserva el tamaño de la entrada, el cual se ve reducido. El grado de reducción dependerá del tamaño del kernel $\boldsymbol{k}$.

En una red convolucional, cada elemento del kernel se utiliza en cada sección de la entrada, excepto por algunos de los elementos en los bordes. Para cada convolución, la red aprende un solo conjunto de parámetros, en lugar de aprender un nuevo conjunto para cada sección. Esto resulta en una propiedad llama equivarianza traslacional; si un objeto se mueve en la entrada de la convolución, su representación se moverá de igual forma en la salida de la convolución.

Típicamente, como se puede observar en la Figura 2, una capa convolucional de una red neuronal consiste de tres etapas. En la primer etapa, se realizan una serie de convoluciones en paralelo para producir una imagen, a partir de la entrada, por cada convolución que se realiza. A estas imágenes se les suele llamar canales. En la segunda etapa, se aplica una función de activación a todos los canales. Finalmente, en la tercer etapa, se realiza una reducción de muestreo o *pooling*. Vale aclarar que también es común referirse a la etapa de convolución y la etapa de pooling como capas separadas.

Una función de pooling reduce el tamaño del canal por secciones utilizando algún criterio que captura alguna propiedad de esa sección. Por ejemplo, max pooling reporta el valor máximo del canal en alguna sección rectangular del canal. En la Figura 3 se muestra un ejemplo de max pooling 2x2. La reducción de muestreo ayuda a la invarianza traslacional de la representación respecto de pequeñas traslaciones en la entrada. Esto es, si se modifica la entrada con una pequeña traslación, la mayoría de las salidas de la capa no cambian, luego de haber aplicado pooling. La invarianza local traslacional es una propiedad útil, ya que en muchos casos interesa si una característica esta presente, y no en que parte de la imagen se encuentra.

Un problema crucial en machine learning es como lograr que el modelo presente buen rendimiento no solo en el conjunto de entrenamiento, sino también en entradas nuevas. Existen muchos métodos desarrollados con el objetivo de reducir el error de prueba, incluso si aumentan el error de entrenamiento. Estos métodos se conocen como métodos de regularización y abarcan cualquier modificación del algoritmo de aprendizaje que se aplique con el objetivo de reducir el error a la hora de hacer generalizaciones. Uno de los métodos de regularización comúnmente utilizados es *batch normalization*, que es un método de reparametrización adaptativo que suele aplicarse luego de la etapa convolucional. Batch normalization es una forma de reparametrizar la red neuronal, de forma que reduce el problema de coordinar actualizaciones a través de muchas capas. Este problema surge debido a que todas las capas se actualizan en simultaneo, pero las actualizaciones a cada capa fueron calculadas asumiendo que las otras capas no cambian. Un análisis detallado de batch normalization puede encontrarse en la Sección 8.7.1 de *Deep Learning*[1].

III. CLASIFICADOR CONVOLUCIONAL PROFUNDO SOBRE FASHION-MNIST

Se implementó una red convolucional profunda como clasificador sobre el dataset Fashion-MNIST[4]. Este conjunto de datos está formado por 70.000 imágenes de 28x28 píxeles en escala de grises, dividas en un conjunto de entrenamiento de 60.000 imágenes y un conjunto de prueba de 10.000 imágenes. Las imágenes están etiquetadas en diez categorías, correspondientes a diferentes prendas de ropa y tipos de calzado. En la Figura 4 se pueden observar algunas imágenes del dataset con sus co-

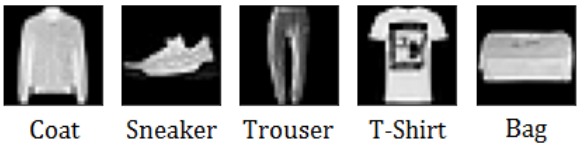

Figura 4: Imágenes seleccionadas aleatoriamente del dataset Fashion-MNIST, con sus correspondientes etiquetas.

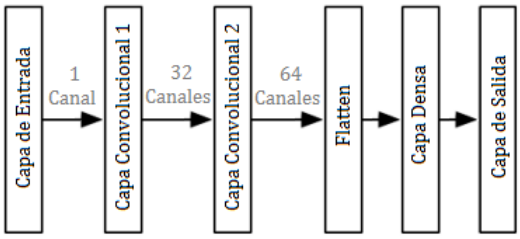

Figura 5: Representación de la arquitectura general utilizada.

rrespondientes etiquetas. Para la implementación de hizo uso de la librería PyTorch[5].

Se optó por una arquitectura usada comúnmente, mostrada en la Figura 5, que implementa la extracción de características y luego la clasificación. Consta de dos capas convolucionales que alimentan una red feedforward multicapa con una salida de diez neuronas, una para cada categoría. La primer capa convolucional recibe la entrada y produce 32 canales como salida, mientras que la segunda recibe estos 32 canales y produce 64 como salida. En ambas capas convolucionales se utilizó batch normalization como regularización, ReLU como función de activación y max pool 2x2 como reducción de muestreo. Luego de las capas convolucionales, se lleva los 64 canales de salida a un tensor unidimensional mediante la aplicación de una función *flatten* y se lo usa como entrada para una capa densa. Tanto en la capa densa como luego de la aplicación de la función flatten, se utilizó dropout como regularización y ReLU como función de activación.

Como algoritmo de entrenamiento se utilizó backpropagation en conjunto con el optimizador Adam, con los valores predeterminados por PyTorch para los parámetros, y minibatch de tamaño 500. La función de pérdida utilizada fue la función de entropía cruzada, o *Cross Entropy Loss*, debido a que es una buena medida de distancia entre distribuciones de probabilidades ya que tiene en cuenta que los elementos de la salida están restringidos a que su suma sea 1.

Se buscó optimizar sobre algunos hiperparámetros y buscar el número óptimo de épocas de entrenamiento para el modelo que dé los mejores resultados. Para ello, y con el propósito de que el tiempo de cómputo sea manejable, se eligieron valores predeterminados usuales. Los hiperparámetros elegidos fueron: el tamaño k de los kernel en las capas convolucionales, el número n de neuronas de la capa densa y la tasa p de dropout para las capas lineales. Se eligieron $k = 3, 5$, $n = 128, 256, 512$ y $p = 0{,}1, 0{,}25$, para un total de 12 modelos, los cuales se

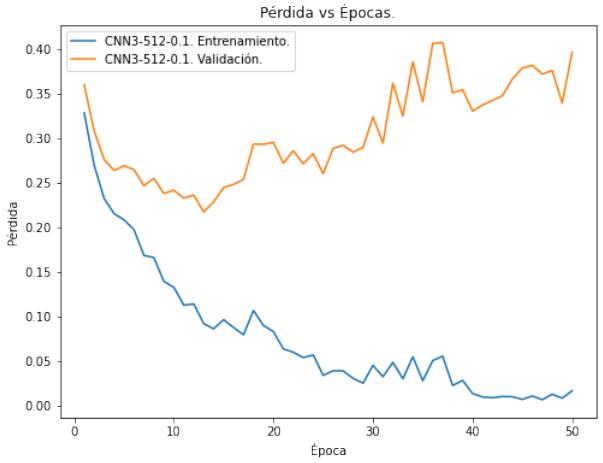

Figura 6: Curva de pérdida para uno de los modelos entrenados.

Modelo	Época óptima	Pérdida	Exactitud [%]
CNN3-512-0.1	13	0.217	92.44
CNN3-128-0.25	17	0.214	92.38
CNN3-128-0.1	16	0.228	92.15
CNN3-256-0.25	15	0.213	92.10
CNN3-512-0.25	11	0.214	91.95
CNN3-256-0.1	14	0.238	91.72
CNN5-512-0.25	11	0.231	91.62
CNN5-128-0.25	23	0.237	91.55
CNN5-512-0.1	11	0.251	91.46
CNN5-256-0.25	10	0.236	91.40
CNN5-256-0.1	14	0.258	91.27
CNN5-128-0.1	9	0.248	90.81

Tabla I: Número óptimo de épocas de entrenamiento, pérdida, y exactitud, sobre el conjunto de validación, para cada modelo.

denotarán como CNNk-n-p. Para poder comparar estos modelos entre sí, se dividió el conjunto de entrenamiento en un nuevo conjunto de entrenamiento de 50.000 imágenes y un conjunto de validación de 10.000 imágenes. Para ello se recurrió al método `random_split` de PyTorch y se utilizó un generador con un *seed* manual.

En primera instancia, se entrenó cada modelo por 50 épocas con el objetivo de buscar el número de épocas para el cual la pérdida de validación alcanza su valor mínimo. Se utiliza el mínimo en la pérdida de validación y no el máximo en la exactitud (*accuracy*) de validación debido a que la pérdida contiene información sobre cuan erróneas son las predicciones erradas. En la Tabla I se incluyen los valores de pérdida y exactitud de cada modelo para su número óptimo de épocas. Aunque no se encuentren diferencias significativas entre los primeros modelos de la Tabla I, se eligió el modelo CNN3-512-0.1, que es el que, además de presentar mayor exactitud sobre el conjunto de validación para su número óptimo de épocas,

Modelo	Época óptima	Pérdida	Exactitud [%]
CNN3-512-0.1	13	0.2667	90.89
CNN-Nahuel	19	0.2845	90.11
ANN-Juan	28	0.3131	89.26

Tabla II: Número óptimo de épocas de entrenamiento, pérdida, y exactitud, sobre el conjunto de prueba, para cada modelo.

su número óptimo de épocas y valor de pérdida sobre el conjunto de validación son de los más bajos. Este modelo fue reentrenado utilizando su número óptimo de épocas. Su curva de pérdida se puede observar en la Figura 6. Se podría pensar que existe underfitting en el conjunto de entrenamiento para el valor óptimo de épocas, ya que la pérdida continua disminuyendo, pero esto se debe a que el modelo ajusta el ruido del conjunto de entrenamiento, por lo que no distingue el ruido de la información útil.

Luego, se utilizó el conjunto de prueba para comparar este modelo con otros dos. El primero es el modelo presentado por el Mg. Nahuel Almeira[6], que es una red convolucional con una arquitectura exactamente igual a la mostrada en la Figura 5, salvo que cuenta con dos capas densas, una de 600 neuronas y una de 120, en ese orden, con una tasa de dropout de 0.25. El segundo es el presentado por el Dr. Juan I. Perotti[7], que es una red feedforward de una capa oculta de 512 neuronas. Para este modelo, se repitió el procedimiento detallado anteriormente con la tasa de dropout y se encontró un mejor rendimiento con una tasa de 0.25. Para ambos modelos, que se optó por denominar CNN-Nahuel y ANN-Juan, respectivamente, se encontró un número óptimo de épocas de entrenamiento. Los valores de rendimiento de los tres modelos sobre el conjunto de prueba se encuentran en la Tabla II. Aunque es fácil ver que los modelos convolucionales poseen mejor rendimiento, la diferencia no es grande. Sin embargo, es notable que los modelos convolucionales obtengan ese rendimiento con un número bastante menor de épocas.

IV. CONCLUSIÓN

Los clasificadores convolucionales implementan tanto la extracción de características como la clasificación de forma relativamente simple. Se puede observar que, respecto de una red de una capa oculta, ofrecen buen rendimiento con entrenamiento en un número pequeño de épocas a expensas de mayor tiempo de entrenamiento. Esto puede deberse simplemente a que la red convolucional tiene mas parámetros para entrenar.

La diferencia en exactitud del modelo CNN3-512-0.1 en los conjuntos de validación y de prueba sugiere la presencia de un *bias* en el conjunto de validación. Esto puede ser producto de las concesiones hechas con el objetivo de reducir el tiempo de entrenamiento. En particular, la forma en la que se utilizó el conjunto de entrenamiento original para construir el nuevo conjunto de entrenamiento y el conjunto de validación. La forma acertada de hacerlo sería mediante algún método de validación cruzada, que usan el conjunto de entrenamiento original en su totalidad tanto para entrenar como para validar, y promediar los resultados sobre las diferentes divisiones del conjunto original.

De no ser tan largos los tiempos de entrenamiento, se podría haber hecho una optimización mas rigurosa de los hiperparámetros. Incluso, se podría haber considerado otros hiperparámetros que fueron tomados como fijos, en particular la tasa de aprendizaje para el optimizador, otros optimizadores, como SGD, otras funciones de activación, como LeakyReLU, que no es computacionalmente demandante, etc. Para que el tiempo de entrenamiento no crezca de forma indiscriminada, habría que considerar la posibilidad de establecer un orden de jerarquía entre los hiperparámetros, o considerarlos directamente independientes entre sí.

Mas allá de esto, se pudo implementar el clasificador de forma satisfactoria. Para un dataset tan simple como Fashion-MNIST, el *trade-off* entre tiempo de cómputo y mejora de rendimiento no parece estar justificado.

[1] I. Goodfellow, Y. Bengio and A. Courville "*Deep Learning*". MIT Press, 2016.

[2] Y. Gavrilova (2020, October 8). "*A Guide to Deep Learning and Neural Networks*". Serokell Blog.

[3] Adaptación (CC BY 4.0). Obra original por N. C. Pratiwi, I. Nur, Y. Fu'adah & K. Masykuroh. "*Computer-Aided Detection (CAD) for COVID-19 based on Chest X-Ray Images using Convolutional Neural Network*". IOP Conference Series: Materials Science and Engineering. 2020; 982(1):012004. `http://doi.org/10.1088/1757-899X/982/1/012004`

[4] H. Xiao, K. Rasul & R. Vollgraf. "*Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*". arXiv. Aug 2017. `https://doi.org/10.48550/arxiv.1708.07747`. Dataset disponible en: `https://github.com/zalandoresearch/fashion-mnist`

[5] A. Paszke et al. "*PyTorch: An Imperative Style, High-Performance Deep Learning Library*". Advances in Neural Information Processing Systems 32. 2019; pp. 8024–8035. Disponible en: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[6] Disponible en: `https://drive.google.com/drive/u/1/folders/1LsN-KbYw36fw_dYhr5gheDIGFvvM9ddg`

[7] Disponible en: `https://github.com/jipphysics/curso-redes-neuronales-famaf-2021`