Trabajo de investigación

Técnicas de visualización en *Deep Learning*

Sistemas Inteligentes para la Gestión en la Empresa

9 de junio de 2019

Felipe Peiró Garrido
felipepg@correo.ugr.es

Juan Carlos Serrano Pérez
jcsp0003@correo.ugr.es

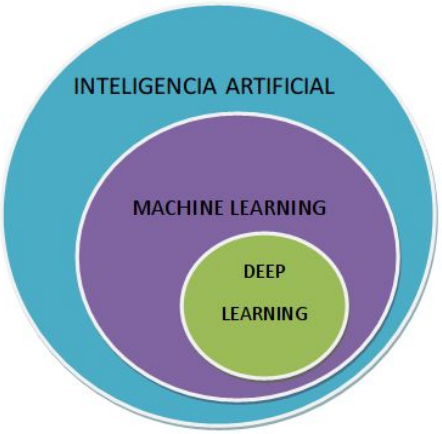
Pedro Manuel Gómez-Portillo López
gomezportillo@correo.ugr.es

Índice

1. Introducción

El *Deep Learning* es una de las áreas de investigación más populares dentro del campo de la Inteligencia Artificial. La mayoría de las nuevas investigaciones que se realizan trabajan con modelos basados en las de Deep Learning, ya que gracias a ellas se han logrado resultados sorprendentes en campos como Procesamiento del Lenguaje Natural y Visión por Computadora. [1]

En general, se tiende a hablar de Inteligencia Artificial, *Machine Learning* y *Deep Learning* de manera similar, aunque estos tres conceptos no son iguales y hacen referencia a conceptos distintos.



Relación entre IA, ML y DL, [1]

Por un lado, la Inteligencia Artificial, que nació en los años 50, hace referencia a técnicas como algoritmos de búsqueda, razonamiento simbólico, razonamiento lógico y estadística. Debido al aumento de la capacidad y al abaratamiento de las tecnologías de la información se producen, almacenan y envían más datos que nunca. De hecho, se calcula que el 90% de los datos disponibles actualmente en el planeta se ha creado en los últimos dos años, produciéndose actualmente en torno a 2,5 quintillones de bytes por día [2], por lo que surge la necesidad de técnicas que puedan manejar estos datos de manera eficiente.

El *Machine Learning* hace referencia a un amplio conjunto de técnicas informáticas que nos permiten dotar a los ordenadores de capacidad para aprender sin ser haber programado explícitamente cómo hacerlo. Existen tres tipos diferentes de algoritmos; aprendizaje supervisado, aprendizaje no supervisado y *Deep Learning*. [2]

El aprendizaje supervisado se basa en modelos entrenados con un conjunto de datos en los que los resultados de salida son conocidos. Los modelos aprenden de esos resultados conocidos y realizan ajustes en sus parámetros interiores para adaptarse a los datos de entrada.

El aprendizaje no supervisado trata con datos sin etiquetar cuya estructura es desconocida y su objetivo es extraer información significativa sin la referencia de variables de salida conocidas y mediante la exploración de la estructura de dichos datos sin etiquetar.

Por último, el nombre del *Deep Learning* hace referencia al uso de estructuras jerárquicas de redes neuronales artificiales que se construyen de una forma similar a la estructura neuronal del cerebro humano y que permite lograr el aprendizaje a través de diversas capas ejecutando un análisis de datos de manera no lineal. [2]

Los logros del *Deep Learning* son muy diversos, en los últimos años el Deep Learning ha producido toda una revolución en el campo del Machine Learning, con resultados notables en todos los problemas que requieren de ver y escuchar, problemas que implican habilidades que son naturales para los seres humanos pero que son muy difíciles para las máquinas [1]. Ejemplos de esto son los asistentes virtuales como Siri o Alexa, la clasificación de imágenes o el vencer a personas en juegos y videojuegos.

Actualmente existen muchos frameworks y librerías que trabajan con estas técnicas, como TensorFlow, Keras, Pytorch, Stickit-learn, Lasagne, DSSTINE, MXNet, DL4J, o Microsoft Cognitive Toolkit. Por tanto, puede verse que la democratización de estas tecnologías es inminente, ya que la barrera para desarrollar y utilizar redes neuronales es más baja que nunca, aunque los modelos complejos de aprendizaje profundo pueden ser difíciles de entender. [3]

Para ello, se han diseñado interfaces interactivas para ayudar a las personas a visualizar y comprender qué han aprendido los modelos y cómo hacen predicciones. En este trabajo se presentarán las más importantes y se explicará cómo se han utilizado y aplicado al trabajo de prácticas.

2. Soluciones disponibles

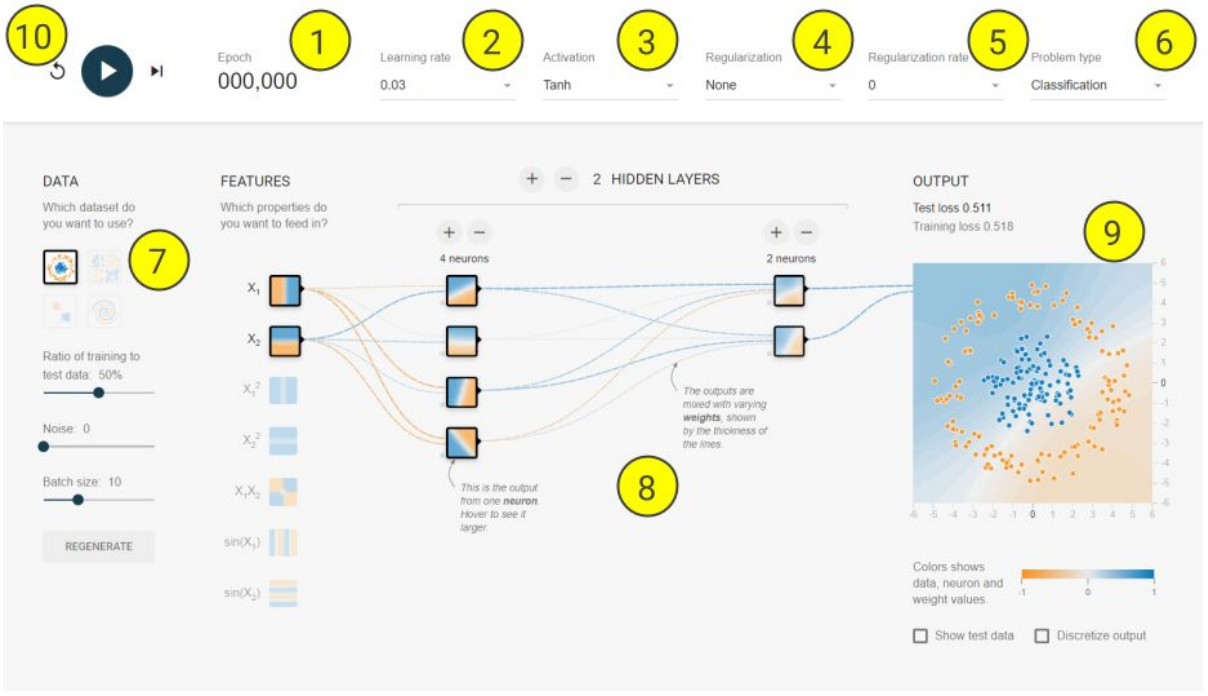
A continuación se presentan técnicas de visualización más populares. Además, se presentará un ejemplo de cada una para mostrar su potencial.

2.1. Playground

Playground[1] es una herramienta de código abierto de TensorFlow, escrita en TypeScript con la librería d3.js, que permite la generación, edición y visualización de redes neuronales en tiempo real desde un navegador web. El propio nombre hace referencia a la habilidad del usuario para "jugar" con la red neuronal como si estuviera en un patio de recreo.

Gracias a esta herramienta podemos simular redes neuronales totalmente parametrizables y ver cómo sus capas evolucionan a lo largo de las épocas.

A continuación se presenta una guía por su interfaz.



1. Época en la que la simulación se encuentra. Antes de comenzar, será 0.
2. Ratio de aprendizaje de la red neuronal.

[1] https://playground.tensorflow.org

3. Función de activación que utilizará el modelo, a elegir entre Relu, Tanh, Sigmoid y Lineal.

4. Regularización de la red neuronal, que es un método para reducir el sobreajuste y mejorar el rendimiento del modelo. Se puede elegir entre L1 o L2.

5. El radio de regularización que se utilizará.

6. El tipo de problema en el que estamos trabajando, que puede ser de clasificación o de regresión.

7. Del mismo modo, podemos elegir el dataset con el que trabajaremos y editar su aleatoriedad, la cantidad de ruido que tiene o el tamaño del lote.

8. En esta sección podemos editar las capas que tendrá nuestro modelo. Se puede modificar las capas iniciales, el número de capas ocultas y el número de neuronas de cada una.

9. Aquí puede verse la salida del modelo en tiempo real. Cuando comencemos la simulación, podemos ver cómo va evolucionando en función de los parámetros que hayamos seleccionado.

10. Por último, cuando tengamos el modelo configurado, podemos comenzar la simulación.

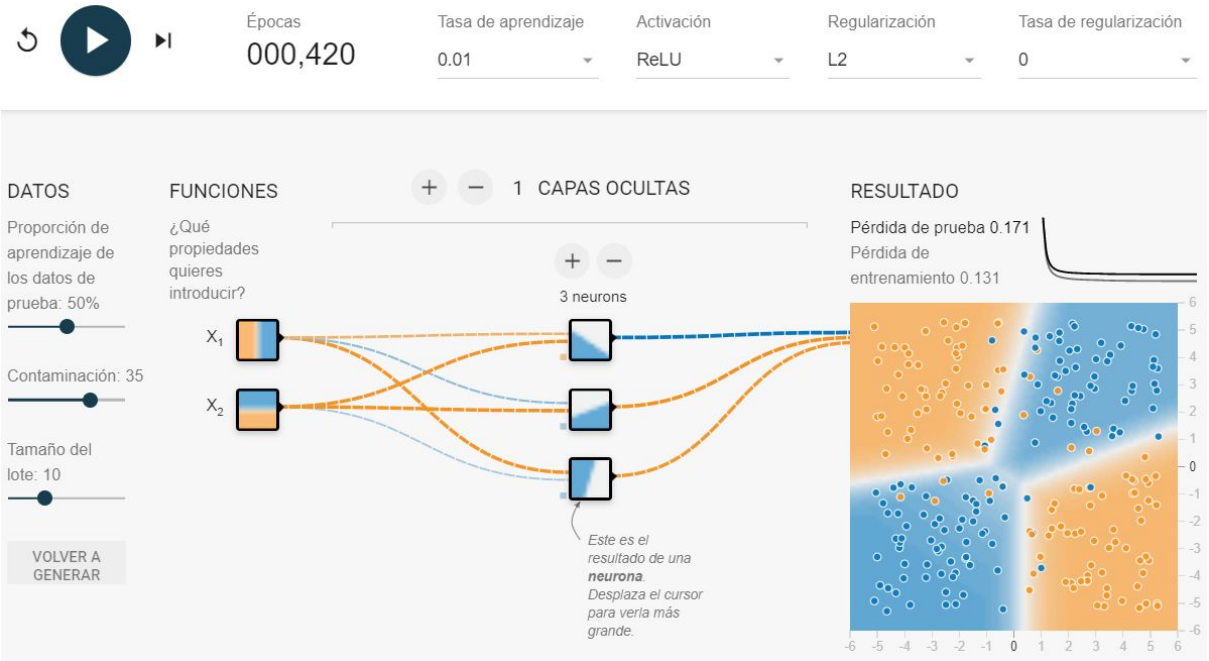
En conclusión, aunque esta herramienta está muy bien para visualizar redes neuronales teóricas desde un punto de vista educativo no permite utilizar nuestro propio dataset, lo que le resta utilidad si queremos ver cómo se estaría comportando una red en un caso concreto.

2.1.1. Ejemplo

A continuación se presenta un ejemplo de uso de Playground, obtenido de la página oficial de Google [4].

Lo primero que se hará es combina dos atributos de entrada en una sola neurona. Como estamos utilizando una función de activación lineal, y el conjunto de datos por defecto es no lineal, es imposible que el modelo funcione correctamente, da igual las épocas que le dejemos ejecutarse. Esto podemos verlo en tiempo real en el resultado del modelo, en rojo en la imagen inferior.

Para arreglar esto, podemos aumentar el número de neuronas de la capa oculta a tres y seleccionar una función de activación no lineal, como Relu (o *Rectified Lineal Unit* por sus siglas en inglés). Con estos sencillos cambios podemos ver al instante cómo la exactitud del modelo aumenta y la pérdida de prueba disminuye de 0.5 a 0.17.



Además, en cada ejecución los resultados cambian ligeramente, por lo que es fácil ver la naturaleza aleatoria de estos modelos.

2.2. TensorBoard

Actualmente existen varios frameworks en el mercado que permiten trabajar con esta tecnología. Uno de los más populares es TensorFlow[2]. Este es un framework de código abierto desarrollado por Google que, según su página web, *se utiliza para realizar cálculos numéricos mediante diagramas de flujo de datos en los que los nodos de los diagramas representan operaciones matemáticas y las aristas reflejan las matrices de datos multidimensionales (tensores) comunicadas entre ellas*.

Pero los cálculos para los que se usa TensorFlow, como el entrenamiento de una red neuronal profunda masiva, pueden llegar a ser complejos y confusos. Para facilitar la comprensión, la depuración y la optimización de estos modelos, TensorFlow ha desarrollado un conjunto de herramientas de visualización llamadas TensorBoard, que puede utilizarse para visualizar gráficos, trazar métricas cuantitativas sobre la ejecución o mostrar datos adicionales. [5]

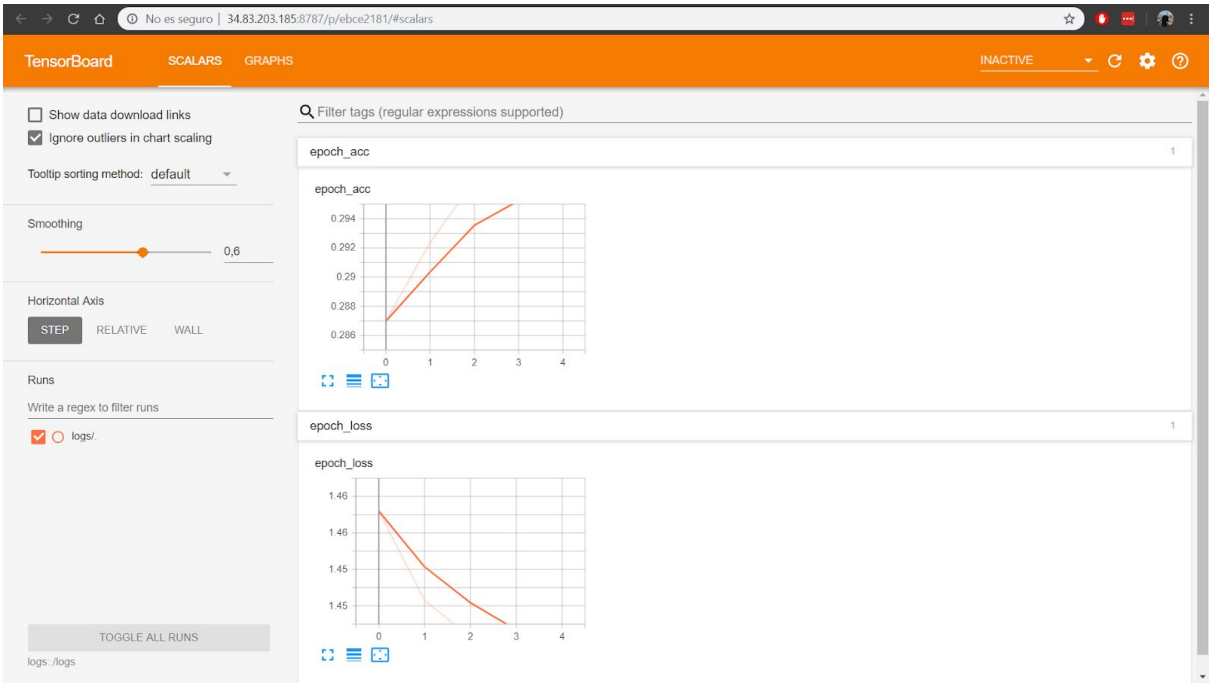
Utilizar TensorBoard en un proyecto de TensorFlow es bastante sencillo. Lo primero que debemos hacer es generar y guardar los logs en un directorio, y suponiendo que estamos trabajando en R como en la práctica, necesitaremos incluir el parámetro `callback_tensorboard(log_dir = "path/to/logs/")` cuando invoquemos a la función `fit()`.

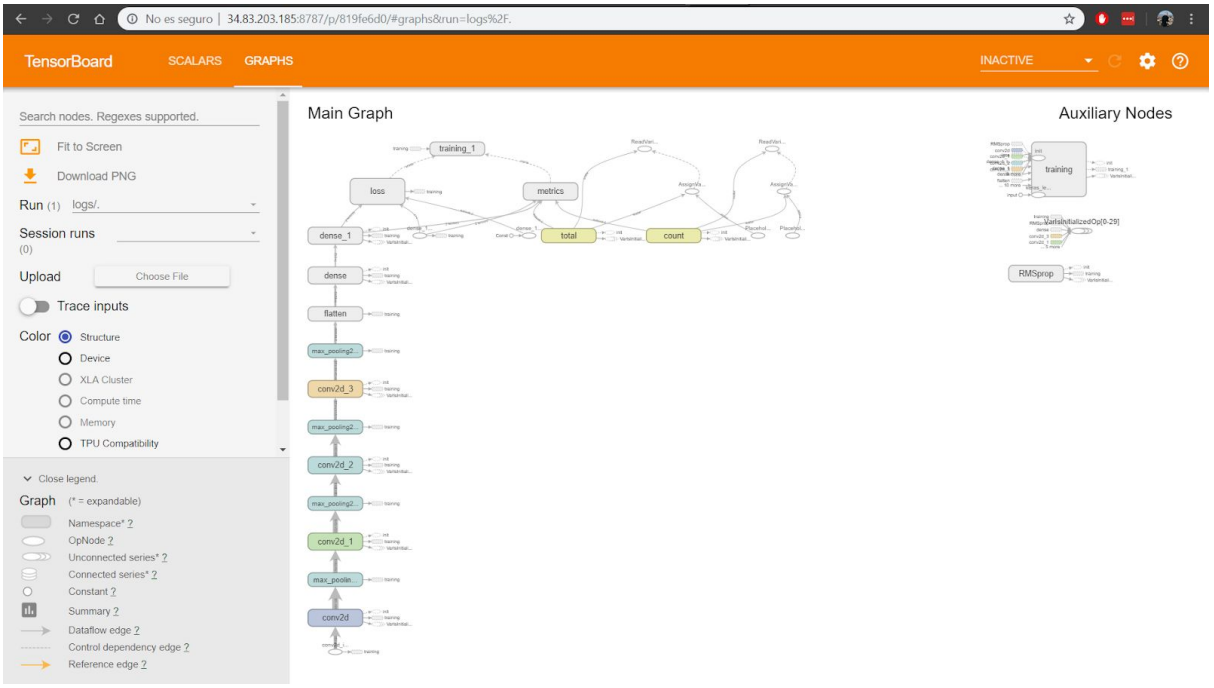
Cuando haya terminado de ejecutarse el modelo y se hayan generado los archivos auxiliares podremos ejecutar TensorBoard con `tensorboard("path/to/logs/")`, indicándole por parámetros la ruta a los archivos.

2.2.1. Ejemplo

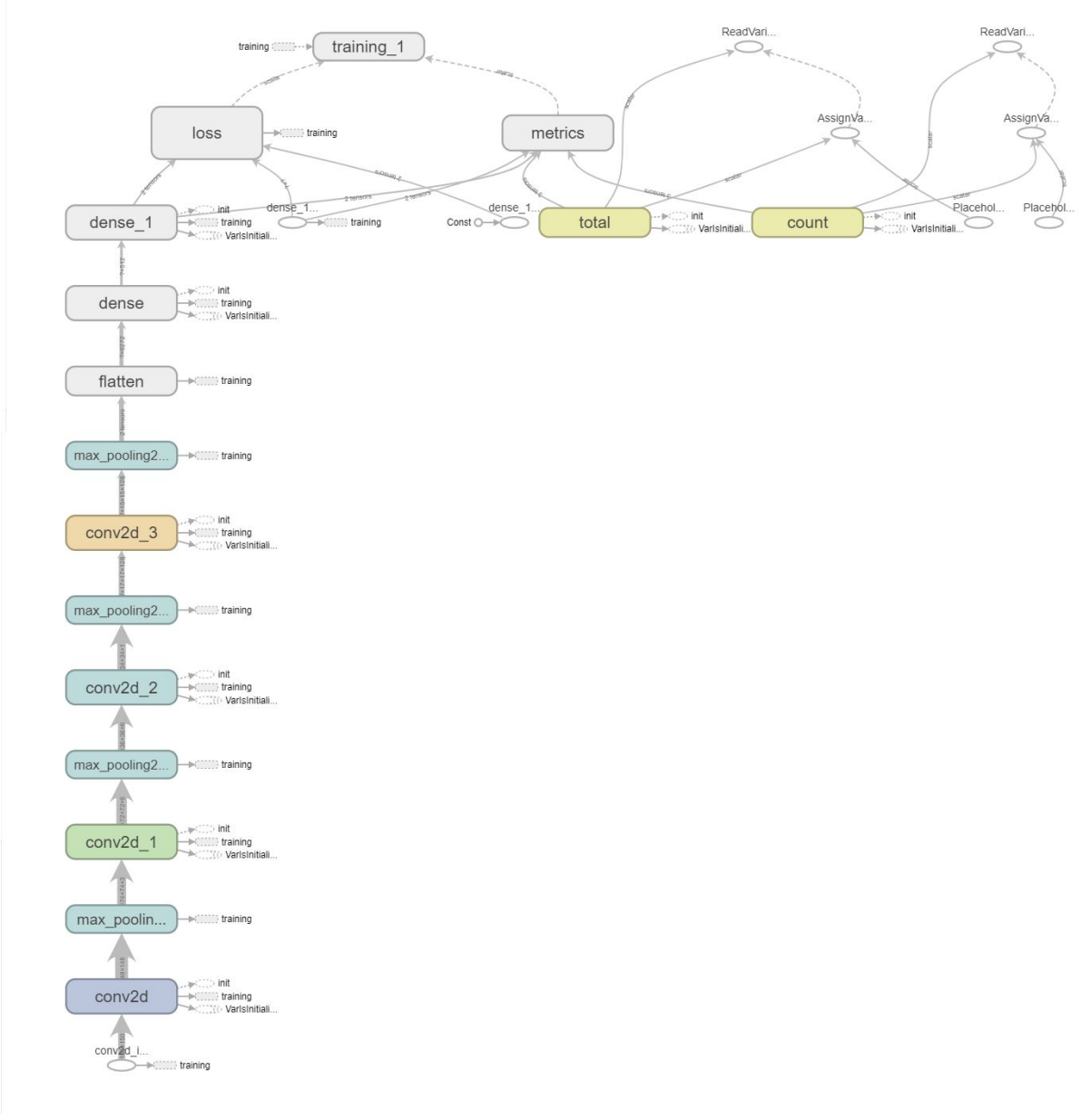
Como ejemplo de esta herramienta, y como a diferencia de Playground sí pueden utilizarse datasets propios, se ha utilizado la segunda práctica de este asignatura. A continuación se presenta las capturas de pantalla de TensorBoard sobre el modelo entrenado.

[2] https://www.tensorflow.org





También se adjunta el modelo descargado desde la ventana anterior para que el lector pueda verlo con más detalle. Como puede verse, el modelo refleja la estructura de capas y parámetros utilizada para el trabajo de prácticas.

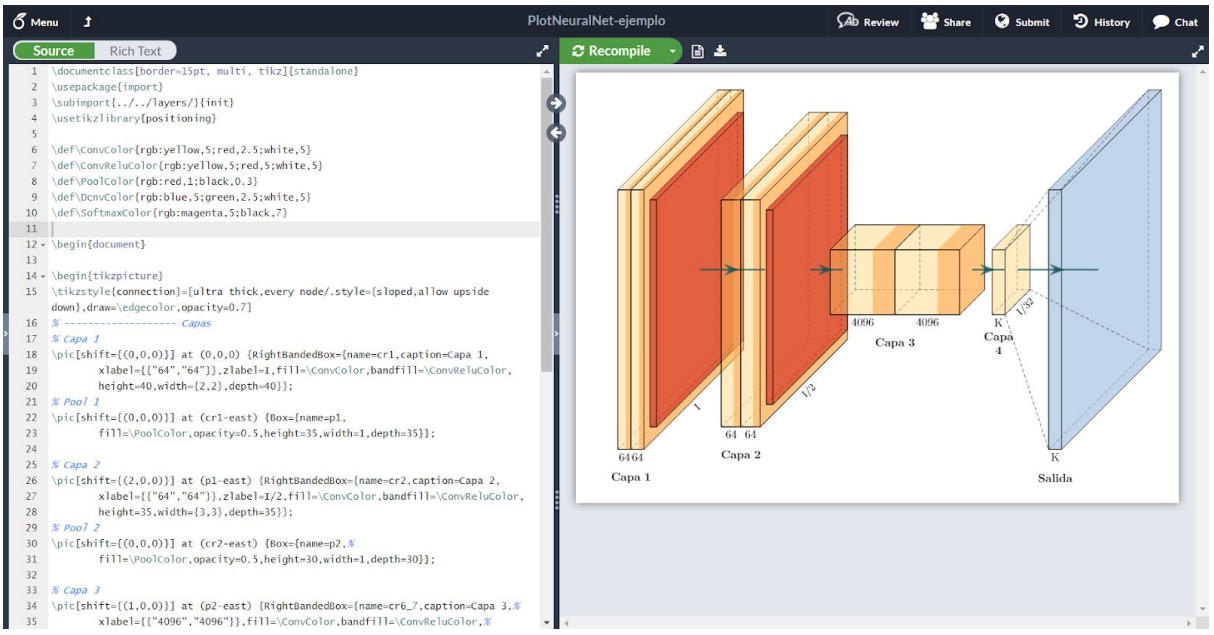

training
training_1
loss
training
metrics
ReadVari...
ReadVari...
AssignVa...
AssignVa...
dense_1
init
training
VarIsInitiali...
dense_1...
training
Const
dense_1...
total
init
VarIsInitiali...
count
init
VarIsInitiali...
Placehol...
Placehol...
dense
flatten
max_pooling2...
conv2d_3
max_pooling2...
conv2d_2
max_pooling2...
conv2d_1
max_poolin...
conv2d
conv2d_i...

2.3. PlotNeuralNet

PlotNeuralNet[3] es un proyecto desarrollado por Haris Iqbal que permite utilizar código Latex para generar gráficos de redes neuronales. Esta librería es utilizada principalmente para generar estos diagramas para presentaciones.

2.3.1. Ejemplo

Para este ejemplo se ha utilizado la página Overleaf[4]. Como pretende ser un ejemplo que muestre el potencial de esta herramienta se ha generado un gráfico simple; se han utilizado 4 capas y una de salida, y el diagrama resultante puede verse en la imagen inferior.

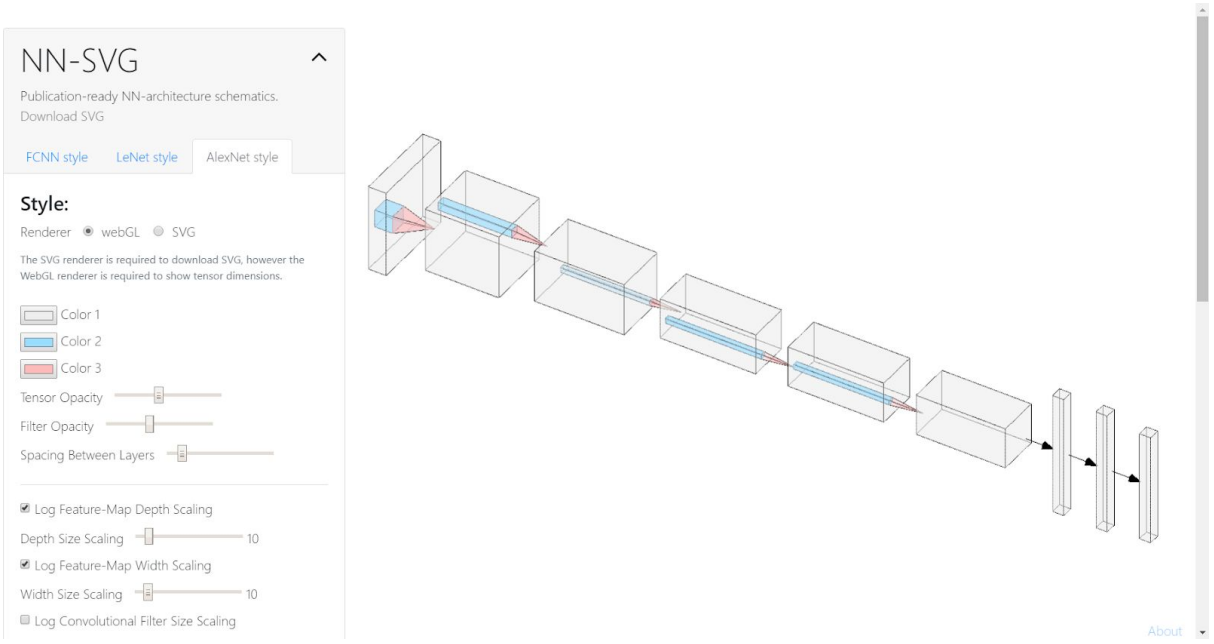


Además, el código entero Latex del ejemplo puede verse en el anexo 5.1.

[3] https://github.com/HarisIqbal88/PlotNeuralNet
[4] https://www.overleaf.com/

2.4. NN-SVG

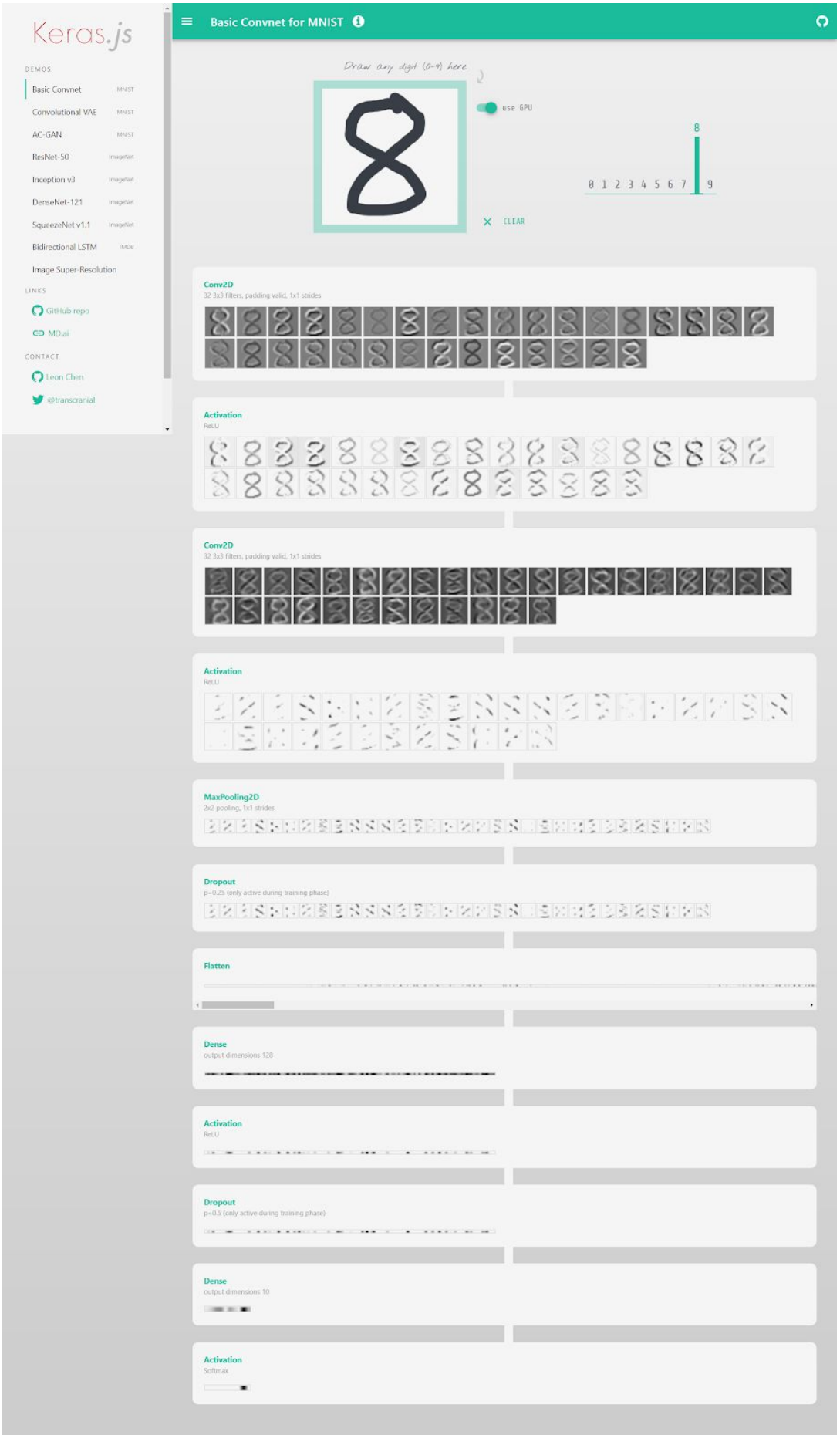
Una de las herramientas más interesantes que se ha encontrado es NN-SVG[5], una herramienta online desarrollada por Alexander LeNail que permite generar manualmente diagramas que representan redes neuronales. Dispone de tres estilos diferentes, el primero son pequeñas esferas que representan las neuronas, el segundo las representa con planos bidimensionales y por último el tercero es un visualizador tridimensional que puede moverse y girarse para obtener el plano que más nos guste. Además, todos las opciones del modelo son totalmente personalizables.



2.5. Keras.js

Otra herramienta, aunque esta no es simplemente de representación, es Keras.js[6], un proyecto que permite desplegar y utilizar una red neuronal en el navegador. Tiene multitud de ejemplos, entre los que destacan el reconocimiento de imágenes, un aumentador de resolución para fotografías o un ejemplo interactivo en tiempo real para reconocer números escritos utilizando la base de datos MNIST. Se ha elegido para este trabajo ya que cuando dibujamos un número podemos ver en tiempo real cómo las diferentes capas actualizan su valor y lo renderizan en el navegador.

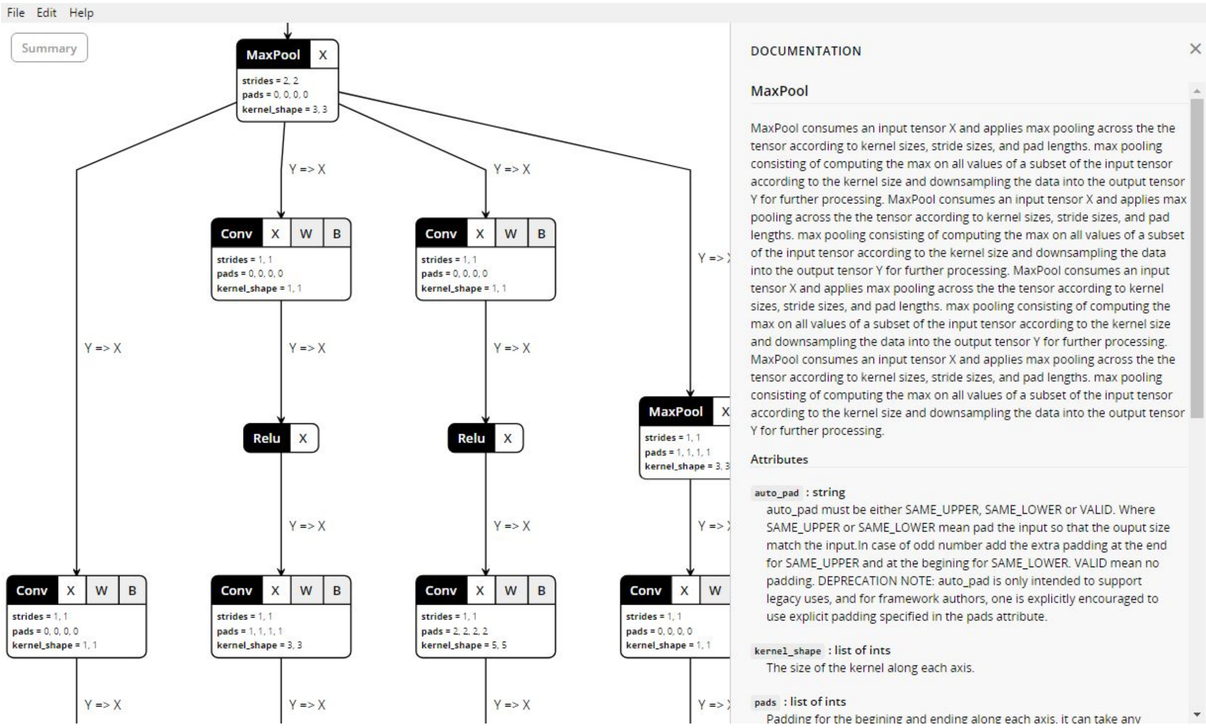
[5] http://alexlenail.me/NN-SVG/index.html
[6] https://transcranial.github.io/keras-js/#/mnist-cnn


Keras.js
DEMOS
Basic Convnet MNIST
Convolutional VAE MNIST
AC-GAN MNIST
ResNet-50 ImageNet
Inception v3 ImageNet
DenseNet-121 ImageNet
SqueezeNet v1.1 ImageNet
Bidirectional LSTM IMDB
Image Super-Resolution
LINKS
GitHub repo
MD.ai
CONTACT
Leon Chen
@transcranial
Basic Convnet for MNIST
Draw any digit (0-9) here
use GPU
CLEAR
0 1 2 3 4 5 6 7 8 9
Conv2D
32 3x3 filters, padding valid, 1x1 strides
Activation
ReLU
Conv2D
32 3x3 filters, padding valid, 1x1 strides
Activation
ReLU
MaxPooling2D
2x2 pooling, 1x1 strides
Dropout
p=0.25 (only active during training phase)
Flatten
Dense
output dimensions 128
Activation
ReLU
Dropout
p=0.5 (only active during training phase)
Dense
output dimensions 10
Activation
Softmax

2.6. Netron

Por último, otro de los proyectos más interesantes que se ha visto es Netron[7], un visor de modelos de *Machine Learning*. Este proyecto ha sido desarrollado por Lutz Roeder, desarrollador de Visual Studio, y soporta la mayoría de frameworks, entre los que destacan Keras 4, MXNet, TensorFlow, Torch y CNTK.

El propio repositorio contiene modelos de ejemplo para poder probar la herramienta en todos los formatos que anuncia soportar y, una vez abierto, podemos incluso consultar la documentación de las capas utilizadas.



[7] https://github.com/lutzroeder/Netron

3. Conclusiones

Como se ha visto, la Inteligencia Artificial es un campo extremadamente profundo y complejos que a su vez proporcionan muy buenos resultados, lo que explica su popularidad y por qué poco a poco las grandes compañías están comenzando a utilizaras.

Y se su profundidad y abstracción se sigue la necesidad de disponer de herramientas para poder visualizar y entender cómo funcionan, por lo que actualmente existen muchas herramientas para ello.

Este trabajo nos ha servido como primera toma de contacto con el número y la complejidad de herramientas disponibles actualmente para visualizar redes neuronales, como TensorBoard o Keras.js.

Como conclusión, estamos muy contentos de haber elegido este tema y haber podido aplicarlo a la parte práctica de la asignatura, permitiéndonos visualizar y entender mucho mejor los conceptos con los que hemos trabajado.

4. Webgrafía

[1] https://relopezbriega.github.io/blog/2017/06/13/introduccion-al-deep-learning/

[2] https://link.medium.com/zyzOenpBaX

[3] https://link.medium.com/13OU9MWBfX

[4]https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/playground-exercises

[5] https://www.tensorflow.org/guide/summaries_and_tensorboard

5. Anexos

5.1. Código Latex del ejemplo de PlotNeuralNet

```
\documentclass[border=15pt, multi, tikz]{standalone}
\usepackage{import}
\subimport{../../layers/}{init}
\usetikzlibrary{positioning}

\def\ConvColor{rgb:yellow,5;red,2.5;white,5}
\def\ConvReluColor{rgb:yellow,5;red,5;white,5}
\def\PoolColor{rgb:red,1;black,0.3}
\def\DcnvColor{rgb:blue,5;green,2.5;white,5}
\def\SoftmaxColor{rgb:magenta,5;black,7}

\begin{document}

\begin{tikzpicture}
\tikzstyle{connection}=[ultra thick,every node/.style={sloped,allow upside
 down},draw=\edgecolor,opacity=0.7]
% ------------------- Capas
% Capa 1
\pic[shift={(0,0,0)}] at (0,0,0) {RightBandedBox={name=cr1,caption=Capa 1,
       xlabel={{"64","64"}},zlabel=I,fill=\ConvColor,bandfill=\ConvReluColor,
       height=40,width={2,2},depth=40}};
% Pool 1
\pic[shift={(0,0,0)}] at (cr1-east) {Box={name=p1,
       fill=\PoolColor,opacity=0.5,height=35,width=1,depth=35}};

% Capa 2
\pic[shift={(2,0,0)}] at (p1-east) {RightBandedBox={name=cr2,caption=Capa 2,

xlabel={{"64","64"}},zlabel=I/2,fill=\ConvColor,bandfill=\ConvReluColor,
       height=35,width={3,3},depth=35}};
% Pool 2
\pic[shift={(0,0,0)}] at (cr2-east) {Box={name=p2,%
       fill=\PoolColor,opacity=0.5,height=30,width=1,depth=30}};

% Capa 3
\pic[shift={(1,0,0)}] at (p2-east) {RightBandedBox={name=cr6_7,caption=Capa
3,%
       xlabel={{"4096","4096"}},fill=\ConvColor,bandfill=\ConvReluColor,%
       height=10,width={10,10},depth=10}};

%% Capa 4
\pic[shift={(1,0,0)}] at (cr6_7-east) {Box={name=c8,caption=Capa 4,%
       xlabel={{"K","dummy"}},fill=\ConvColor,%
       height=10,width=2,depth=10,zlabel=I/32}};

%% Salida
\pic[shift={(2.5,0,0)}] at (c8-east) {Box={name=d32,caption=Salida,%
```

```
        xlabel={{"K","dummy"}},fill=\DcnvColor,%
        height=40,width=2,depth=40}};


%% Conexiones
\draw [connection]  (p1-east)     -- node {\midarrow} (cr2-west);
\draw [connection]  (p2-east)     -- node {\midarrow} (cr6_7-west);
\draw [connection]  (cr6_7-east) -- node {\midarrow} (c8-west);
\draw [connection]  (c8-east)     -- node {\midarrow} (d32-west);

%% Lineas de puntos
\draw[densely dashed]
(c8-nearnortheast) -- (d32-nearnorthwest)
(c8-nearsoutheast) -- (d32-nearsouthwest)
(c8-farsoutheast)  -- (d32-farsouthwest)
(c8-farnortheast)  -- (d32-farnorthwest)
;

\end{tikzpicture}
\end{document}\grid
```