Departamento de Informática y Ciencias de la Computación
Facultad de Ingeniería
Universidad de Concepción

Tarea 1: Desafío clasificación de estampillas

Iván Astorga

Resumen

El High Cadence Transient Survey (HiTS), observó el cielo durante un periodo de tiempo con el propósito de encontrar objetos que varían su brillo en corto tiempo. Para esto, se consideró una imágen de ciencia y una imágen de template, la cual fue restada de la imagen de ciencia, produciendo así la imágen de diferencia, la que se puede escalar en función del ruido estimado por pixel produciendo la imagen de diferencia razón-señal-a-ruido (SNR difference). En el caso de que un objeto aparezca repentinamente en una observación, este se debiese ver en la imágen de diferencia. Muchas veces aparecen artefactos en las imágenes de diferencia debido a malas restas, pixeles erróneos o variación estadística de las imágenes. El objetivo de este trabajo es clasificar estampillas de HiTS en artefactos y objetos reales. Para ello, utilzaremos una modelación de clasificación binaria con herramientas de aprendizaje profundo, especificamente Redes Neuronales.

Metodología

Para resolver nuestro problema, en vez de usar un modelo de clasificación, como Random Forest, usaremos una arquitectura de redes neuronales. Procederemos a realizar nuestro proyecto basándonos en la metodología de ciencia de datos. Primeramente, se definirán las librerías utilizadas: Para procesamiento de datos y gráficos importaremos las librerias *numpy, pandas, matplotlib, sklearn* y, para armar nuestra red neuronal, importaremos *Tensorflow*, y desde ella también *Keras*, la cual nos servirá para usar las capas *Dense, Conv2D, Flatten y MaxPooling2D* .

Luego de eso, cargaremos los datos para explorar un poco el dataset. Al hacer esto, apreciamos que el data set contiene 4 tipos de imágenes que usaremos para nuestro entranamiento y no hay datos faltantes ni duplicados. Luego, exploramos las dimensiones de cada conjunto y obtenemos $(4026, 441)$ en cada uno, por lo que tendremos que hacer un reajuste a la dimensión, ademas de concatenar cada uno de los set de datos, como tambien realizar un $OneHotEncoding$ con las etiquetas.

Proseguiremos con la división de nuestros datos de entrenamiento y validación, para ello, usaremos la función *train test split*, por lo que quedarán con dimensión $(2697, 21, 21, 4)$ y $(1329, 21, 21, 4)$ el conjunto de entrenamiento y el de validación respectivamente.

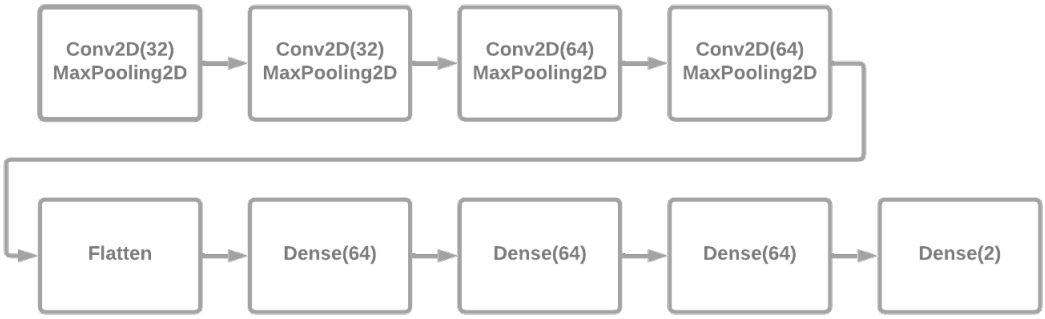
Ya con este análisis, procesamiento y división de los datos podemos entrar a modelar el problema de clasificación de imágenes. Notamos que nuestros datos serán tensores con dimensión $(21, 21, 4)$, por ende, nuestra dimensión de entrada deberá ser esa y toda nuestra red estará basada en esas dimensiones.

Modelamiento

Para la modelación, como mencioné anteriormente, usaré una arquitectura de Redes Neuronales. En este caso, escogeremos una Red Sequential Forward, la cual tendrá capas convolucionales y densas.

En un inicio armé una arquitectura de solo una capa convolucional sin maxpooling ni padding con 4 neuronas, una capa flatten y tres capas densas, con 32 neuronas las primeras dos capas y la última solo con 2 para que pueda calzar con la dimensión de salida esperada. Para la compilación usaremos *"binary crossentropy"* para la perdida, esta es sugerida por Google en Tensorflow Core para clasificación binaria. Como optimizador usaremos *Adam* y como metrica *accuracy*. Procedemos a ajustar el modelo usando 15 epocas con batch size de 50, lo que nos arroja un accuracy de 95 % en el conjunto de validación, lo cual para un primer intento es aceptable.

Luego de testear distintas arquitecturas, llegué a la conclusión de que era necesario aumentar la cantidad de capas convolucionales, decidí que fueran 4 capas convoluciones y 4 densas. A cada una de las capas convolucionales se me hizo necesario pasarla por un filtro, en este caso un MaxPooling de $(2, 2)$, además, a cada una se le agregará un padding. Testeamos hiperparámetros, en este caso la cantidad de neuronas, usando 2^n neuronas, con $n = 4, 5, 6, 7$, llegando a la conclusión de que lo mejor será usar 32 neuronas en las primeras 2 capas convolucionales y 64 neuronas en las otras 2 para luego usar una capa Flatten y así usar despues 4 capas densas de 64 neuronas cada una. Quedando de esta forma nuestra arquitectura:

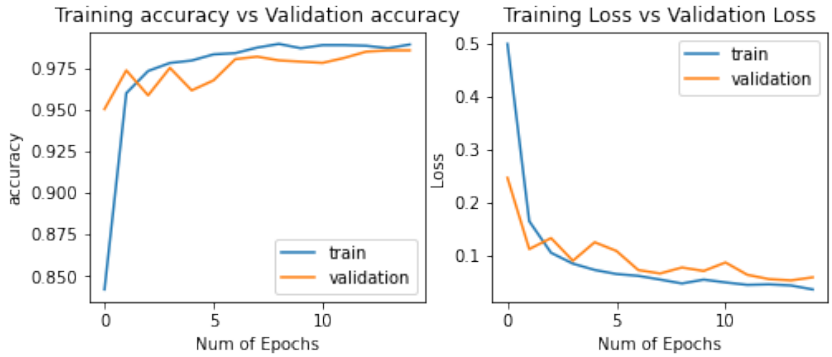


Las capas convolucionales se activarán con $ReLu$ y las densas con $Sigmoid$. Como compilador usaremos *"binary crossentropy"* para la perdida, *rmsprop* como optimizador y de métrica usaremos *accuracy*. En el entrenamiento usaremos un *batch size* de 50 y 15 epocas.

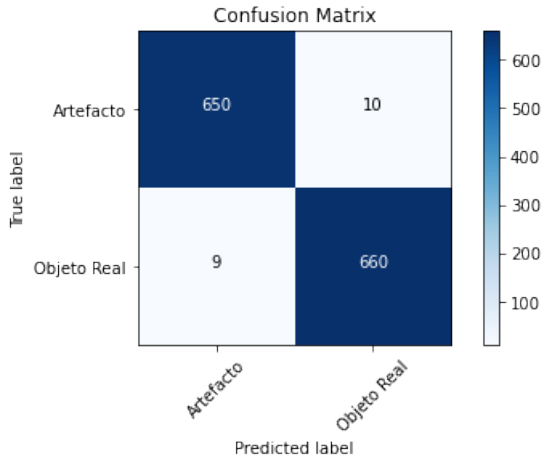
Después del tiempo de entrenamiento, se utiliza el atributo *summary* y se puede notar que fueron entrenados un total de $41{,}986$ parámetros.

Resultados

Nuestro entrenamiento demoró alrededor de 16 segundos, usando mi equipo que tiene una tarjeta gráfica Nvidia GTX 1660 de 4gb y un procesador Intel I5 de 11va generación. Para analizar el comportamiento del modelo a través de las epocas, se gráfica su *accuracy* y *loss*.



Graficamente podemos ver que el rendimiento en el conjunto de entrenamiento y validación se fue haciendo cada vez mas similar con el paso de las epocas. Usamos unas metricas para ver el rendimiento final en el conjunto de validación, donde obtenemos un *accuracy* de $0{,}9857$ y un *F1 Score* tambien de $0{,}9857$ aproximadamente. Considero que es un rendimiento muy aceptable. Se intentó realizar Data Augmentation pero se obtuvieron peores resultados. Finalizando los resultados en el conjunto de validación, graficamos las matriz de confusión:



Donde podemos notar que nuestro modelo en el conjunto de validación supo identificar 650 artefactos correctamente y solo 10 los confundió con objetos reales, y pudo identificar 650 objetos reales y solo 9 los confundió con un artefacto.

Luego de eso, probaremos el modelo en el conjunto de test, al cual también se le deberá reajustar la dimensión y concatenar las 4 tipos de imágenes. Posterior a eso, se guardó en un csv las etiquetas obtenidas en el predict para ser enviadas al ayudante, el cual nos entrego un *accuracy* de $0{,}988$ y un *F1 Score* de $0{,}9879$.

Conclusiones

Finalizando, podemos conluir que la arquitectura de Redes Neuronales propuesta obtiene un buen rendimiento, tanto en el conjunto de validación y de test, y que el modelo no se sobreajusta a los datos. También notamos que el coste computacional fue considerablemente bajo, por lo que puede ser recomendable usar Redes Neuronales en vez de un modelo de clasificación de machine learning para este tipo de problemas.

Referencias

1. Google, Tensorflow Core. “Image classification” disponible en Aquí, 2022.