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

Tarea 2: Generación de Estampillas HITS

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. Como vimos en la Tarea 1 suelen aparecer 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 como tambien crear un Autoencoder Variacional que sea capaz de generar y muestrear nuevos datos sinteticos. Para la primera tarea nos apoyaremos en nuestra red ya entrenada y para nuestro VAE usaremos una arquitectura con capas probabilísticas.

Metodología

Para realizar este trabajo usaremos un AutonEncoder Variacional. Procederemos a realizar nuestro proyecto basándonos en la metodología de ciencia de datos. Primeramente buscaremos entrenar un clasificador que nos ayudará a ver si nuestras prediciones son correctas, para este, usaremos las siguientes librerías: Para procesamiento de datos y gráficos importaremos *numpy, pandas, matplotlib, sklearn*, para crear la red neuronal usaremos como Backend *Tensorflow*, y desde ella podremos usar *Keras*, la cual nos servirá para usar las capas *Dense, Conv2D, Flatten y MaxPooling2D*. Para entrenar nuestro VAE usaremos *TensorFlow Probability* y desde ella usaremos las capas *DenseFlipout*.

Al igual que en la tarea 1 realizamos un procesamiento de los datos para que queden acorde a nuestro modelos.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 (3220, 21, 21, 4) y (806, 21, 21, 4) el conjunto de entrenamiento y el de validación respectivamente.

Realizamos el entrenamiento de nuestras redes(clasificadora y VAE) para las cuales 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 para el clasificador, escogeremos una Red Sequential Forward, la cual tendrá 4 capas convolucionales y 4 densas. A cada una de las capas convolucionales se les agrega un MaxPooling de $(2, 2)$, además, a cada una se le agregará un padding. Se usarán 64 neuronas en las 4 capas convolucionales para luego usar una capa Flatten y así usar despues 3 capas densas de 64 neuronas cada una y una de 2 neuronas para la salida. Las capas convolucionales se activarán con **ReLu**, las densas con $Sigmoid$ y la de salida con **SoftMax**. 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 63.042 parámetros.

Para la VAE utilizamos la arquitectura con capas probabilísticas, en este caso usaremos en un inicio una capa Flatten para luego usar 4 capas $DenseFlipout$, las cuales tendran 64 neuronas las 3 primeras capas y la ultimas 2 y se activaran con **ReLu** y **SoftMax** respectivamente. Cada capa Flipout tendra como función de perdida la Divergencia de Kullback–Leibler la que nos ayudará a ver cuanta información se esta perdiendo en nuestro Autoencoder. Como optimizador usaremos Adam, un learning rate de 0.0001 y tambien usaremos como perdidad, aparte de la DKL, *"binary crossentropy"*. En el entrenamiento usaremos un *batch size* de 64 y 180 epocas. Se entrenaron un total de 242.626 parámetros.

Resultados

Se uso un equipo que tiene una tarjeta gráfica Nvidia GTX 1660 de 4gb y un procesador Intel I5 de 11va generación, demorando 15 segundos el clasificador y 1 minuto 25 segundos el VAE. Para analizar el comportamiento de los modelos a través de las epocas, se gráfica su *accuracy* y *loss*.

Figura 1: Graficos de exactitud y perdida del Modelo Clasificador

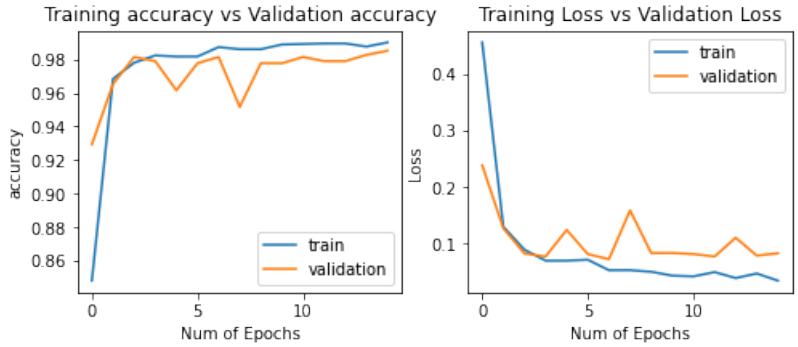
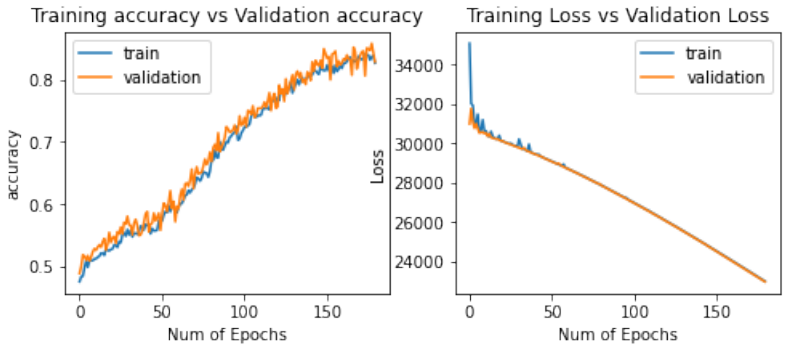


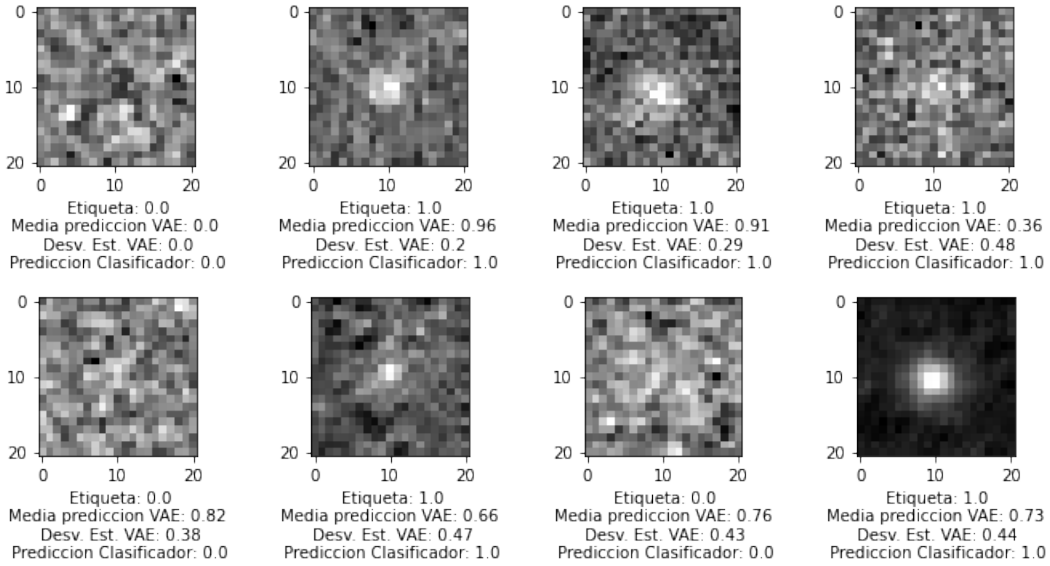
Figura 2: Graficos de exactitud y perdida del VAE



En la Figura 1 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, de manera similar en la Figura 2 vemos que el modelo mejora su rendimiento a lo largo de las epocas. Usamos unas metricas para ver el rendimiento del clasificador en el conjunto de test, donde obtenemos un *accuracy* de 0.992 y un *F1 Score* de 0.99199 mejorando el rendimiento obtenido en la Tarea 1.

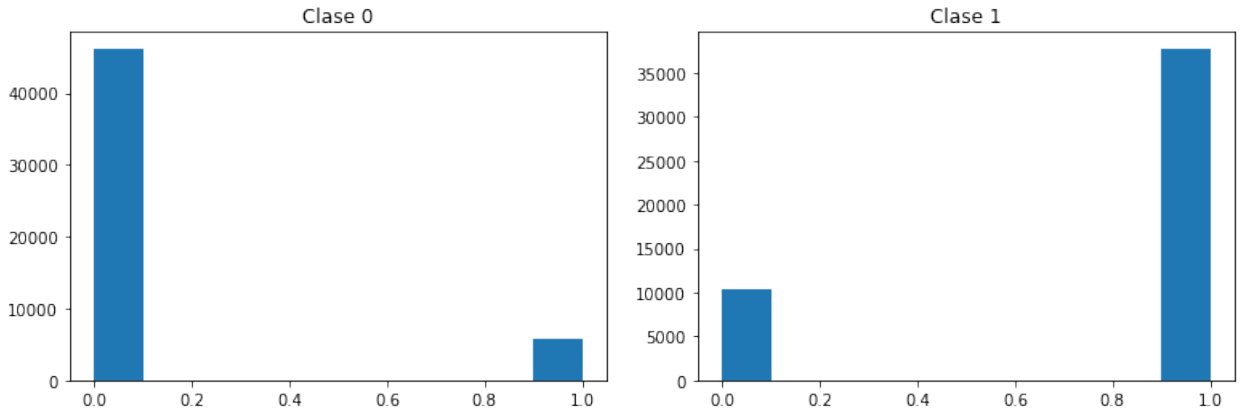
Ahora para medir el rendimiento de nuestro VAE en el conjunto de test, realizaremos 100 muestras de predicciones de 10 imagenes del conjunto de test para verificar el que tan bien lo hizo nuestro modelo.

Figura 3: Muestra de 8 imágenes del conjunto de test



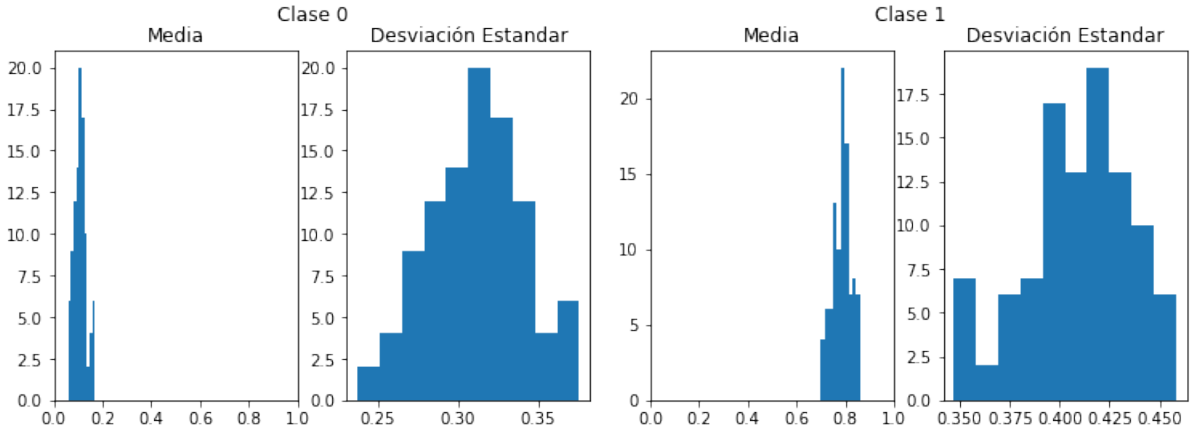
Observando la Figura 3 vemos que nuestro modelo bayesiano se equivoco en 3 de los 8 ejemplos en predecir la media de la imagen de test. Para generalizar podemos ver los histogramas por clase:

Figura 4: Histogramas por clase



En esto podemos notar que para la clase 0 de la muestra el modelo bayesiano acertó 46.102 veces y falló 5.798, es decir alrededor 89 % de precisión, y para la clase 1 acertó 37.753 veces y falló 10347, es decir alrededor de 78 %. Dado que nuestro es bayesiano, podemos estimar las medias y desviaciones estándar por clase:

Figura 5: medias y desviaciones estándar por clase



donde podemos ver que las medias se mantienen cercanas al valor de su clase, evidenciandose tambien en las desviaciones estándar las cuales en ambos casos no superan el valor 20.

Conclusiones

Finalizando, podemos conluir que la arquitectura de Redes Neuronales propuesta propuesta para el clasificador obtiene un muy buen rendimiento, tanto en el conjunto de validación y de test. De igual manera nuestro VAE obtuvo tambien muy buenos resultados por lo que sería recomendable a la hora de querer generar datos sinteticos para ayudar a futuros entrenamientos de otras redes neuronales.

Referencias

1. Google, Tensorflow Core. “Image classification” disponible en Aquí, 2022.
2. Google, Tensorflow Probability. “TFP Probabilistic Layers: Variational Auto Encoder” disponible en Aquí, 2022.
3. Keras, Probabilistic Bayesian Neural Networks, disponible en Aquí, 2022.