RT-BENE: detección de parpadeo

Esteve Soria Fabián

Febrero 2022

1 Resumen

Se generan dos modelos para la resolución del problema de detección de parpadeo.
El primero es un modelo ingenuo en el cual no tengo en cuenta la distribución de datos. Se selecciona un modelo de los disponibles en el zoo de Keras por simplicidad. Además el modelo está preentrenado en Imagenet.
El segundo modelo está también basado en un modelo preentrenado de Keras pero en este caso sí que se tiene en cuenta la distribución de las clases en el dataset y se utilizan técnicas de aumento de datos para corregir la sobre-representación de la clase "no parpadeo" con respecto a "parpadeo".
En este proyecto se consigue un F1-score de 0.9546 en el testset.

Aquí se encuentran los notebooks con el código de este trabajo: `https://github.com/sorny92/RT-BENE/blob/main/notebooks/`

2 Introducción

Se utiliza el conjunto de datos RT-BENE, este consiste en 107350 parejas de imágenes y su correspondiente etiqueta.
Cada pareja de imágenes contiene dos imágenes RGB de tamaño 36x60 pixeles de los ojos de diferentes personas. En total, el dataset, contiene 17 videos cada uno de una persona diferente donde se ha anotado en cada imagen si está parpadeando o no.
El problema a solucionar es la predicción por cada par de imágenes si estas corresponden a un parpadeo o no. Dicho problema se puede considerar como un problema de clasificación donde hay que predecir si para dicha entrada se corresponde una clase u otra. En este caso como solo hay una salida se define como clasificación binaria.

3 Métodos

Se proponen dos soluciones para dicho problema de clasificación. La primera aproximación es la más simple de aplicar y además es ingenua en el sentido de que no considera cual es la distribución de las clases en el dataset. En la segunda se utiliza un modelo más eficiente y además tiene en cuenta la distribución de las clases para poder hacer un entrenamiento más equilibrado.

Lectura de datos

El primer paso para resolver un problema de análisis de datos es su lectura misma ya que cada dataset viene en un formato diferente y por tanto no se puede seguir la

misma aproximación en cada problema. En este dataset se tiene una carpeta con las imágenes y un fichero csv donde cada fila tiene los siguientes datos:

- Identificador único
- Ruta relativa a la imagen del ojo izquierdo
- Ruta relativa a la imagen del ojo derecho
- Identificador del video
- Clase a predecir. '1' si es parpadeo, '0' si no lo es.

Una vez se ha identificado que contiene el dataset es interesante ver la distribución de este según las variables disponibles. Se muestra cuantas imágenes hay en cada uno de los videos. Por ejemplo, al tener un video de cada persona no tenemos la misma cantidad de imágenes por cada persona. Si el desbalance es muy importante se podría sobreentrenar sobre una persona determinada y no ser posible predecir correctamente sobre una población ya que pertenece a un dominio diferente.

En el caso de este dataset no existe un video con una cantidad relativamente grande de imágenes con respecto a las demás por tanto no se rebalancea con respecto al identificador del video.
Otra observación del dataset que si que se tiene en cuenta es el balance de la clase a predecir. Se puede ver como en cada video solo hay entre un 1% y 10% de imágenes que corresponde a la clase parpadeo. Esto significa que si no se utilizan medidas para corregir el desbalance producirá un efecto negativo en la capacidad del modelo de predecir correctamente.

video	Images in video	% blink frames
0	12865	7.236689
1	8671	1.476185
2	8702	9.066881
3	3205	5.210608
4	4750	2.736842
5	5355	2.054155
7	1857	8.023694
8	6108	7.514735
9	4210	1.068884
10	16559	2.131771
11	12817	5.399079
12	935	2.459893
13	9586	3.077405
14	5371	4.002979
15	1810	1.602210
16	4549	0.923280

Figure 1: Tabla con imágenes por video y porcentaje de parpadeos por video

Partición de datos

Se parte el dataset en tres particiones. Una partición de entrenamiento, otra de validación y otra de test.
Primero se parte el dataset en dos partes, entrenamiento y test, dejando el 20% del dataset para test. Un valor de 20% es relativamente generoso para el problema donde normalmente se considera un valor entre 10% y 20%.

Por ejemplo una competición como Imagenet tienen un 15% del dataset como test[**?**]. La partición se hace mediante

selección aleatoria ya que se considera que la selección estratificada no es necesaria ya que el dataset tiene una distribución de clases en cada video relativamente similar. Esto deja alrededor de 20.000 imágenes para test.
En la partición de entrenamiento se parte el dataset en dos para generar el set de validación. Se podría utilizar un sistema de k-fold para la validación pero en el entrenamiento de redes neuronales profundas se vuelve demasiado pesado para ser eficiente en el tiempo. El set de validación tiene un tamaño de 17.000 imágenes aproximadamente.

Lectura y generador de datos

Se crea una clase que permite cargar los datos del fichero .csv y permitir la carga dinámica de las imágenes antes de que sean necesitadas. Esto se hace mediante el uso de la librería tf.data.Dataset[**?**]. El generador de datos sigue el siguiente proceso:

1. Se crea una clase que cada vez que es invocada devuelve la pareja de rutas de imágenes de ojos y su etiqueta.

2. Se utiliza la función .map(), que permite invocar de manera paralela y eficiente tareas pesadas, para cargar las imágenes en memoria y poner los valores de píxel de las imágenes en el rango [0,1].

3. En la segunda versión de los modelos que se generan además existe un paso de aumento de datos que también se ejecuta mediante una función .map() de tf.Dataset. En un punto posterior se explica más en detalle el aumento de datos.

Generación de modelos

Para la resolución del problema se necesita un modelo que contenga dos entradas, una por cada ojo, y una salida, parpadeo o no. La aproximación obvia es tratar de utilizar un modelo preentrenado ya que permitiría ahorrar tiempo de entrenamiento comparado a un modelo inicializado sin preentrenamiento. Se procede a utilizar el zoo de Keras[**?**] ya que contiene una variedad interesante de modelos preentrenados en grandes datasets como Imagenet. El problema del zoo de Keras es que no tiene ningún modelo que use dos entradas. Se podría entrenar un modelo desde una inicialización random pero es sabido que el preentrenamiento suele llegar a una solución válida antes.

Para utilizar el modelo preentrenado se crea un modelo con dos entradas donde cada entrada es seguida de un extractor de características seleccionado del zoo de

modelos. Después la salida de los dos modelos es concatenada para ser conectada a una serie de capas totalmente conectadas.

Esta arquitectura permite utilizar dos entradas pero también un modelo preentrenado así podemos converger a una solución más rápidamente.

En la primera versión se usa un modelo VGG16 preentrenado en imagenet en cada rama para después añadir 2 capas totalmente conectadas: la primera con 128 nodos y la segunda con 1 ya que es la salida. El modelo en total contiene 29.560.705 parámetros.

La segunda versión utiliza DenseNet121 también preentrenado en imagenet. La arquitectura es similar a la anterior anterior solo que en esta el extractor de características es diferente y los nodes totalmente conectados tienen una configuración un poco más grande. En este caso se utilizan tres capas totalmente conectadas con 256, 128 y 1 nodos respectivamente, además las dos primeras tienen una capa dropout para mayor regularización. A cada capa del modelo se le añade un componente regulizador de los pesos de manera que se pueda tener una solución más genérica y evitar el sobreentrenamiento. En el siguiente punto se indica como se han elegido los valores. También se le añade una inicialización de pesas para la última capa basado en el algoritmo Xavier[**?**]. El modelo en total contiene 14.632.577 de los cuales 14.465.281 se pueden entrenar.

En ambos modelos se usa relu como función de activación en las capas totalmente conectadas. También se usa la función sigmoide para la salida ya que es la que permite hacer una clasificación binaria.

3.1 Entrenamiento

En el proceso de entrenamiento se utiliza una función de pérdida BinaryCrossEntropy ya que es la que se utiliza en problemas de clasificación binaria y el optimizador Adam.

En cada época se aleatoriza el dataset así no se tiene el mismo orden de imágenes todo el tiempo evitando una fuente de sesgos.

3.1.1 Modelo VGG16

En el modelo basado en VGG16 se entrena leyendo el dataset según el generador produce valores. Este entrenamiento se continua hasta un máximo de 40 épocas. Se utiliza un callback que detiene el entrenamiento si después de 5 épocas no se mejora la métrica de F1 de validación.

3.1.2 Modelo DenseNet121

En el caso del modelo basado en DenseNet121 el entrenamiento además incluye aumento de datos y equilibrado de clases.

Para equilibrar el número de apariciones de cada clase se divide el dataset de entrenamiento en 2 tablas: la tabla con los parpadeos y la tabla sin parpadeos. A continuación durante el entrenamiento se escoge con una probabilidad del 40% y 60% de cada clase, parpadeo y no parpadeo respectivamente. Se decide usar una proporción 4060 ya que se evita repetir lo máximo posible la clase que está infrarepresentada.

Para el aumento de datos se utiliza la librería Albumentations[**?**]. Se usa esta librería porque es rápida y flexible. Permite una gran variabilidad de modificaciones y además son usables en otro tipo de problemas como detección de objetos o segmentación semántica. En este caso se utilizan las siguiente técnicas de aumento de datos: rotación, modificación del brillo, compresión JPEG, modificación de saturación y hue, modificación del contraste, imagen espejo. Estas modificaciones se producen aleatoriamente en un rango que queda definido en el código de este trabajo.

En las capas de extractor de características se utiliza una regularización más suave que en los capaz totalmente conectadas ya que el extractor de características ya está preentrenado y por tanto no se quiere modificar mucho sus valores.

Como método de entrenamiento también se usa Adam pero se comienza con una learning rate más bajo, $1{\cdot}10^3$ en vez del por defecto 0.01. Por último se añade un callback que reduce el learning rate cada 5 épocas de esta manera se puede converger a una solución más ajustada al dataset.

4 Resultados

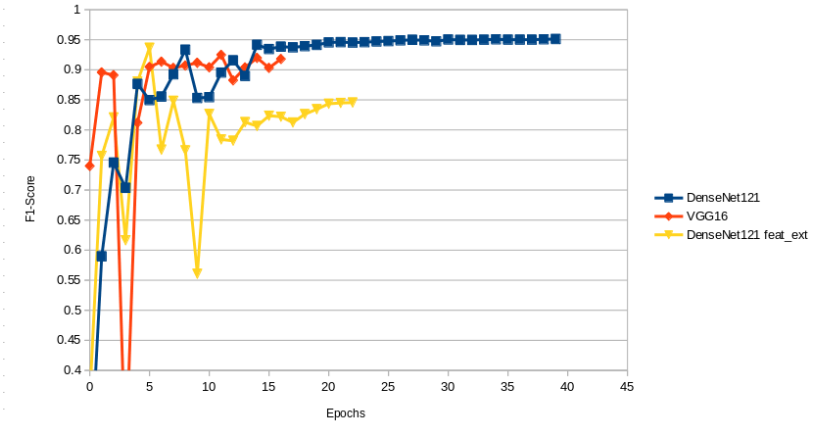

Figure 2: Gráfica de F1-score por época en en test de validación. DenseNet121 y VGG16 entrenados en todas las capas. DenseNet121 feat_ext solo en las capas completamente conectadas.

Los resultados siguientes se consiguen en una máquina con procesador AMD Ryzen 5600 y GPU Nvidia GTX1080. El tiempo de ejecución por batch de 256 imágenes es de aproximadamente **450ms**. Cada época se procesa en 60s aproximadamente.

Dado que el dataset utilizado en este proyecto tiene un dominio diferente a imagenet se presupone que se requiere entrenar todas las capas de la red para tener mayor precisión. Experimentalmente se comprueba que esto es así como se ve en la figura 2.

Los resultados sobre el testset se ven en la siguiente tabla:

Modelo	F1-score
Baseline	0.4
VGG16	0.9254
DenseNet121 feat_ext	0.8374
DenseNet121 final	**0.9543**

5 Conclusiones

Se generan 2 modelos que son capaces de superar el baseline de F1-score de 0.4. Uno de ellos alcanzando un valor máximo de 0.95 con menos de la mitad de parámetros que el modelo VGG16.

References

[1] Alexander V. Buslaev, Alex Parinov, Eugene Khvedchenya, Vladimir I. Iglovikov, and Alexandr A. Kalinin. Albumentations: fast and flexible image augmentations. *CoRR*, abs/1809.06839, 2018.

[2] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[3] Derek G. Murray, Jiří Šimša, Ana Klimovic, and Ihor Indyk. Tf.data: A machine learning data processing framework. *Proc. VLDB Endow.*, 14(12):2945–2958, jul 2021.

[4] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115, 09 2014.

[5] Keras team. Keras applications.