Deep Learning para multi-clasificación

Maria Victoria Santiago Alcalá - Dayana Aguirre Iñiguez

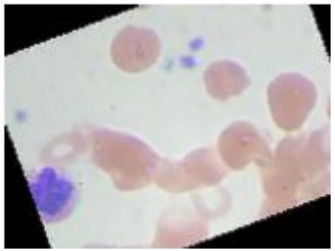
Sistemas Inteligentes para la Gestión en la Empresa

Introducción

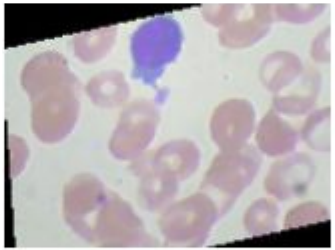
Modelo de clasificación de imágenes basado en redes neuronales profundas.

Conjunto de imágenes de células sanguíneas

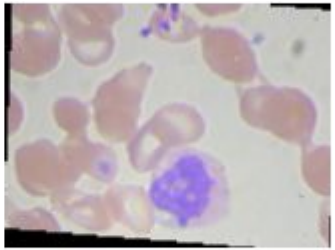
Dataset de Kaggle Blood Cell Images



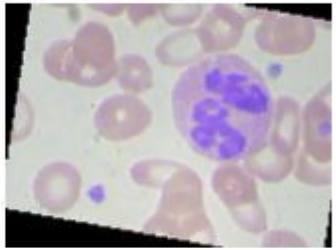
EOSINOPHIL



LYMPHOCYTE

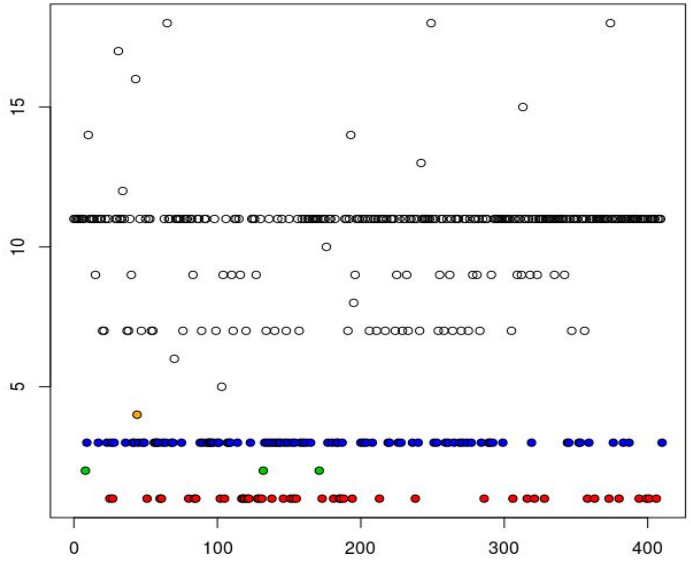


MONOCYTE



NEUTRIPHIL

Distribución de las células




```
> summary(cell_images)
    X                Image               Category
 Mode:logical   Min.   :  0.0   NEUTROPHIL:207
 NA's:411       1st Qu.:102.5   EOSINOPHIL: 88
                Median :205.0              : 44
                Mean   :205.0   LYMPHOCYTE: 33
                3rd Qu.:307.5   MONOCYTE   : 21
                Max.   :410.0   BASOPHIL   :  3
                                (Other)    : 15
```

Fundamentos teóricos

Data Preprocesing

Como primer paso hay que formatear o preprocesar los datos antes de alimentarlos a nuestra red. Actualmente, nuestros datos se encuentran en disco como archivos JPEG, por lo que los pasos para acceder a nuestra red son aproximadamente:

- Leer los archivos de imagen.
- Decodificar el contenido JPEG en cuadrículas RGB de píxeles.
- Convertir floating point tensors
- Cambiar la escala de los valores de píxel (entre 0 y 255) al intervalo [0, 1]

Fundamentos teóricos

Argumentos flow_images_from directory

1. Directory = train_dir -> ruta al directorio de destino, el que contiene un subdirectorio por clase.
2. Generator = train_datagen -> Generador de datos de imagen con una rescale = 1/255
3. Target_size = c(150, 150) -> Dimensiones de las imágenes
4. Batch_size = 50
5. Class_mode = "categorical"

Fundamentos teóricos

Creación del modelo

- Una pila de etapas alternas entre de layer_conv_2d () (con relu activation) y layer_max_pooling_2d ()
- Comenzamos con entradas de tamaño 150 × 150 y terminaremos con mapas de funciones de tamaño 7 × 7. La profundidad de los mapas de características aumenta progresivamente en la red (de 32 a 128), mientras que el tamaño de los mapas de características disminuye (de 148 × 148 a 7 × 7)
- Terminará la red con un layer_dense (4,512) y una activación softmax.
- Con el fin de evitar para evitar el sobreajuste usamos Dropout de 0.4

Fundamentos teóricos

Compilación

Utilizaremos dos diferentes optimizador RMSprop, SGD, Y ADAM al terminar nuestra red con una sola unidad softmax, usaremos la crossentropía categorical como nuestra pérdida.

```
# Compilar modelo
# https://tensorflow.rstudio.com/keras/reference/compile.html
model %>% compile(
  loss= 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(), #optimizer_adam( lr= 0.0001 , decay = 1e-6 ),
  metrics = c("accuracy")
)
```

```
# Compilar modelo
# https://tensorflow.rstudio.com/keras/reference/compile.html
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(lr=0.0001,decay = 1e-6),
  metrics = c('accuracy')

)
```

```
# Compilar modelo
# https://tensorflow.rstudio.com/keras/reference/compile.html
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_sgd(lr=0.0001,decay = 1e-6),
  metrics = c('accuracy')
```

Fundamentos teóricos

Entrenamiento

- Utilizamos la función fit_generator, como primer argumento le indicaremos lotes de entradas, debido a que los datos se están generando infinitamente, el generador necesita saber cuántas muestras extraer del generador antes de declarar una
- Época
- Steps_per_epoch=100
- Cabe indicar un validation_data y validation_steps: el cual indica al proceso cuántos lotes extraer del generador de validación para su evaluación.

Descripción de las redes empleadas

Data Augmentation

- Consiste en agregar valor a los datos básicos.
- Agrega calidad a los datos mejorandolos significativamente.

Funcionamiento

- En cada época, las transformaciones con parámetros seleccionados en un intervalo especificado al azar son aplicados a cada una de las imágenes originales del conjunto de entrenamiento.
- Los datos de entrenamiento se aumenta de nuevo mediante la aplicación de las transformaciones.

Descripción de las redes empleadas

Transfer Learning

- Consiste en usar el conocimiento que se ha aprendido de las distintas tareas para las que se tiene una inmensa cantidad de datos los cuales se encuentran etiquetados.

Funcionamiento

- Con el nuevo conjunto de datos podemos llevar a cabo el ajuste de la CNN preentrenada.
- Como este es muy parecido, podemos usar los mismos pesos con el fin de extraer las características de nuestro nuevo conjunto de datos.

Descripción de las redes empleadas

Fine Tuning

- Descongela algunas de las capas superiores para la extracción de características.
- Ajusta ligeramente las representaciones más abstractas del modelo, a fin de hacerlas más relevantes.
- Ajustar el último bloque convolucional del modelo VGG16 junto con el clasificador de nivel superior.

  Weight. Se usa para especificar qué punto de control de peso inicializará el modelo ⟶ **imagenet.**

  include_top: incluir o no el clasificador en la parte superior de la red. ⟶ **FALSE**.

  input_shape: la forma de los tensores de imagen que alimentaremos a la red. ⟶ **(150,150,3)**

Pasos para ajustar la red - Fine Tuning

- Agregar a la red personalizada sobre una red base ya entrenada; instancia de la base convolucional de VGG16.
- Frozen la red base.
- Entrena la parte que agregaste.
- Unfreeze algunas capas en la red base.
- Entrena conjuntamente estas capas y la parte que agregaste.

Pasos para ajustar la red - Fine Tuning

- Agregar a la red personalizada sobre una red base ya entrenada; instancia de la base convolucional de VGG16.
- Frozen la red base.
- Entrena la parte que agregaste.
- Unfreeze algunas capas en la red base.
- Entrena conjuntamente estas capas y la parte que agregaste.

Discusión de resultados

- Los modelos creados desde el inicio, como se ha comentado anteriormente, han sido numerosos para ello se han ido variando sus capas, las formas de activación y demás parámetros, también se han realizado numerosos entrenamientos de los cuales finalmente se ha obtenido el mejor resultado de accuracy compilando el modelo con el optimizador rmsprop.
- Se ha visto también que los mejores resultados debido a que es un problema multiclase, han sido dados por la clasificación usando categorical_crossentropy y softmax

Discusión de resultados

- Optimizador Adam:

Con este optimizador como se puede apreciar en la imagen, no se han obtenido muy buenos resultados ya que se ha obtenido un 0.82 de accuracy y un 0.78 de loss.

- Optimizador rmsprop:

Con dicho optimizador se ha obtenido los mejores resultados nos ha dado ya que se ha obtenido un 0.91 de accuracy y un 0.2 de loss.

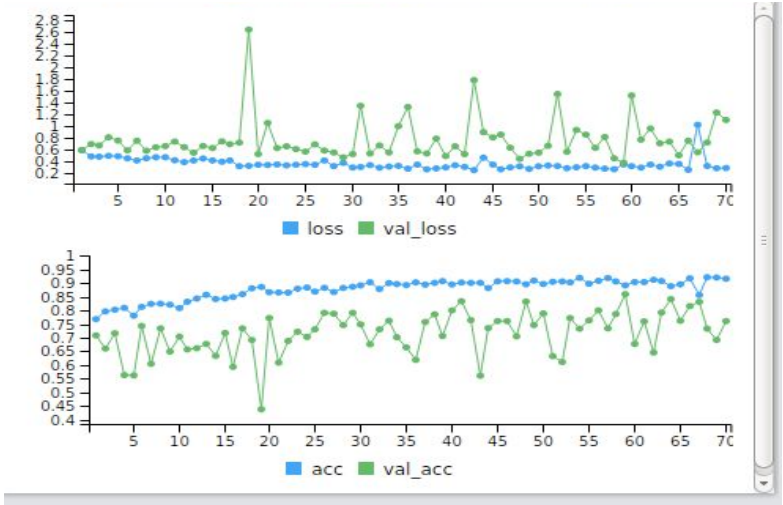
- Optimizador SGD:

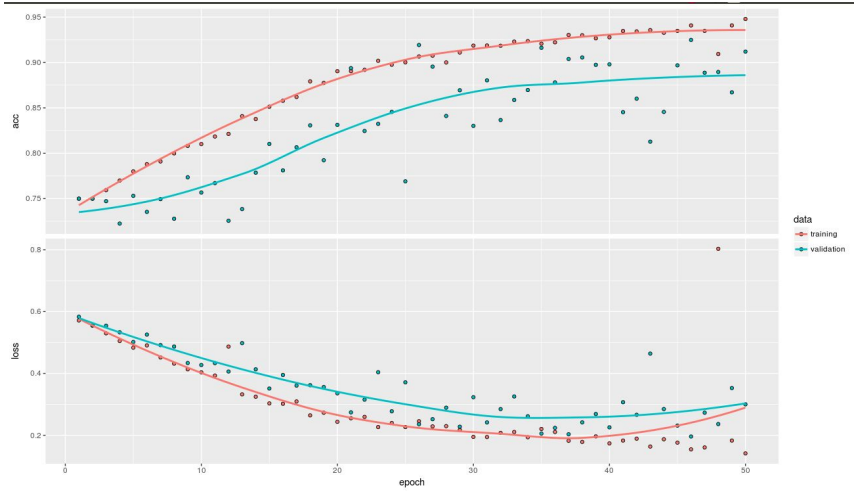
Resultados con el optimizador sgd donde se aprecia un accuracy del 0.82.

Discusión de resultados

- Data Augmentation

Se han obtenido un 0.921 de accuracy y 0.21 de loss.

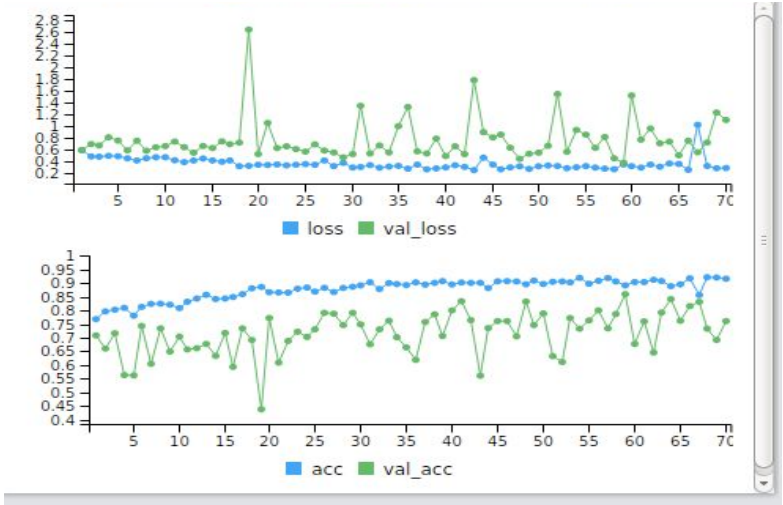


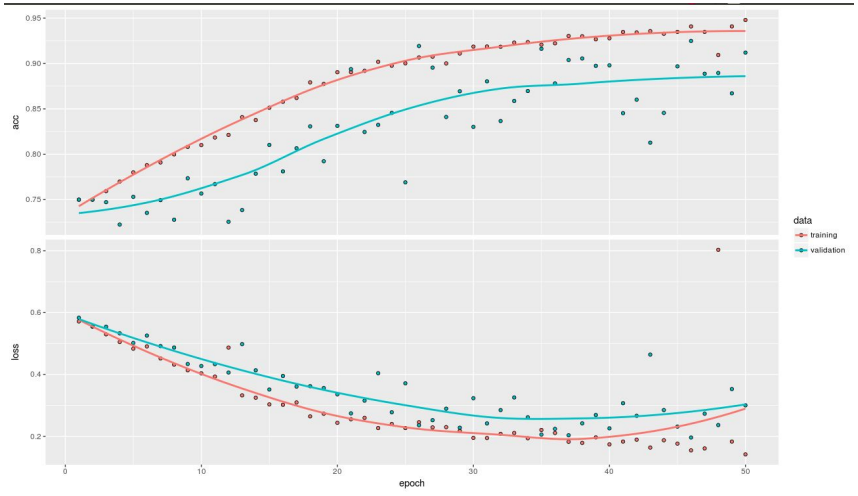


Discusión de resultados

- Transfer Learning & Fine Tuning

Se han obtenido un 0.921 de accuracy y 0.21 de loss.





Conclusiones

- Los resultados obtenidos han sido buenos, mejorables si se hubiese contado con más tiempo debido a la prolongada ejecución de cada una de las evaluaciones.
- Se ha iniciado con modelos ejecutados finalmente con el optimizador rmsprop el cual ha sido el que mejor resultado nos ha dado al principio con respecto al accuracy y el loss, seguidamente se ha aplicado data augmentation debido a que visualizando los gráficos se ha percibido una tendencia a incrementar este accuracy por lo que aumentando los casos de usos podemos ver realmente el valor de este con nuestro modelo.
- Finalmente se ha intentado aplicar Transfer Learning y FIne Tuning.