Máster Universitario en Ingeniería Informática

Sistemas Inteligentes para la Gestión en la Empresa

PRÁCTICA

Deep Learning para clasificación

UNIVERSIDAD DE GRANADA

Carlos Santiago Sánchez Muñoz

Email: carlossamu7@correo.ugr.es

DNI: 75931715K

28 de junio de 2021

Índice

1. Introducción

Esta práctica está enmarcada en el contexto del aprendizaje profundo o *Deep Learning*. Mediante el uso de diversas redes neuronales que se irán construyendo y probando en el desarrollo de la misma, se construirá un clasificador que mediante el reconocimiento de patrones permita distinguir si una noticia es falsa [3].

Los conjuntos de datos de noticias falsas no suelen proporcionar datos de imágenes y texto multimodales, metadatos, datos de comentarios y categorización detallada de noticias falsas a la escala y amplitud de nuestro conjunto de datos. El conjunto que aquí se va a usar, Fakeddit, es un nuevo conjunto de datos multimodal que consta de más de 1 millón de muestras de múltiples categorías de noticias falsas. Después de ser procesadas a través de varias etapas de revisión, las muestras se etiquetan de acuerdo con las categorías de clasificación de 2, 3 y 6 clases a través de una supervisión. Se pueden construir para dichos datos modelos híbridos de texto e imagen. Esto posibilita realizar extensos experimentos para múltiples variaciones de clasificación, lo que demuestra la importancia del aspecto novedoso de la multimodalidad y la clasificación de grano fino exclusivo de Fakeddit.

El interés de este clasificador es evidente: permitiría a empresas detectar de manera automatizada noticias potencialmente falsas o no, o a los gobiernos y a la prensa. También ayudaría a que en los medios sociales no se expendan noticias falsas.

El conjunto de datos viene dado de hasta tres formas:

- un conjunto `mini` que contiene 50 imágenes para entrenamiento, 10 para validación y 10 para test.
- un conjunto `medium` que contiene 10.000 imágenes para entrenamiento, 2.000 para validación y 2.000 para test.
- un conjunto `all` que contiene aún más imágenes y que no se va a usar en esta práctica.

Fíjese que el tamaño de validación y de test es el 20 % que el de entrenamiento. Para la práctia la clasificación que se va a realizar es la binaria en donde una noticia es verdadera o es falsa. Esto viene indicado en las carpetas de datos como `twoClasses`.

2. Fundamentos teóricos

Los algoritmos de aprendizaje para este conjunto de datos son los basados en redes neuronales. Actualmente usando *Deep Learning* con varias redes neuronales convolucionales con múltiples capas ocultas los resultados son, en general, muy satisfactorios. Esto permite tener una herramienta muy potente en marcha que se puede usar para desarrollar clasificadores que ayuden en muchos campos como por ejemplo en medicina, o en prensa distinguiendo noticias potencialmente falsas de las que no.

En este trabajo se va a comenzar construyendo una red neuronal más sencilla y posteriormente incrementarla realizando un ajuste de los diferentes parámetros de la red con el fin de reducir el error sobre el conjunto de prueba.

Dicha red neuronal va a intentar a aprender una función $f : \mathcal{X} \to \mathcal{Y}$ de modo que para cada instancia del $(x_n, y_n) \in (\mathcal{X}, \mathcal{Y})$ se tenga $f(x_n) = y_n$. Claramente los elementos de $\mathcal{X}$ son imágenes (matrices) de píxeles de tamaño 64×64 y el conjunto $\mathcal{Y} = \{Disinformation,\ Other\}$.

Tras la construcción de un modelo avanzado de aprendizaje se implementarán mejoras como el aumento de datos o *Data Augmentation* o introducir una capa de *Batch Normalization*. Asimismo se hara una implementación y puesta en práctica de un modelo de transferencia de aprendizaje o *fine tuning* y finalmente se probará con un ejemplo de *ensemble*.

3. Descripción de las técnicas empleadas

En este apartado se van a exponer la técnicas y herramientas usadas en este proyecto así como aquellos conceptos técnicos esenciales para comprender el mismo.

3.1. Librerías

De entre las muchas bibliotecas disponibles la reina indiscutible es **TensorFlow** [13], que se ha impuesto como la librería más popular en Deep Learning. Actualmente, es dificil imaginar abordar un proyecto de aprendizaje sin ella.

TensorFlow es una biblioteca desarrollada por Google Brain para sus aplicaciones de aprendizaje automático y las redes neuronales profundas, liberado como software de código abierto.

TensorFlow es una librería de computación matemática, que ejecuta de forma rápida y eficiente gráficos de flujo. Un gráfico de flujo está formado por operaciones matemáticas representadas sobre nodos, y cuya entrada y salida es un vector multidimensional (o tensor) de datos.

Por otro lado **Keras** [5] es la API de alto nivel de TensorFlow para construir y entrenar modelos de aprendizaje profundo. Se utiliza para la creación rápida de prototipos, la investigación de vanguardia y en producción, con tres ventajas: es amigable al usuario, es modular y configurable y es fácil de extender.

3.2. Construyendo una red neuronal

Para comenzar esta subsección es fundamental entender los tipos de capas disponibles en Keras, qué hacen y por qué son útiles. Se puede consultar la documentación [1]. Expongo una breve explicación de las que he usado:

- `Sequential`: Indica que la topología de la red es lineal y que todas las capas reciben múltiples entradas o *inputs* y devuelven múltiples salidas o *outputs*.
- `Conv2D`: Es una capa de convolución en $2D$, esta capa crea un núcleo (que actúa como filtro) de convolución que se pasa por todas las imágenes generando la salida. Dicho núcleo es una matriz de convolución que se puede usar para difuminar, afilar, grabar, detectar bordes. El tamaño del mismo suele ser 3×3, 5×5 o 7×7. En nuestro caso el último quizás sea muy grande teniendo en cuenta las dimensiones de las imágenes de nuestro dataset.

  El parámetro `activation` especifica el nombre de la función de activación que desea aplicar después de realizar la convolución.
- `MaxPooling2D`: hay que elegir el tamaño de esa piscina, en nuestro caso va a ser 2×2. Se van a recorrer las imágenes de características de izquierda-derecha y de arriba-abajo tomando *piscinas* de tamaño 2×2 (4 pixeles) y preservando el valor más alto de ellos.

Es un proceso de *subsampling* que permite reducir el tamaño de las muestras haciendo que el número de neuronas de la red no se dispare. Dicho *subsampling* se hace de manera inteligente ya que prevalecen las características más importantes de cada filtro.

- `Dropout`: Es una capa de regularización que desactiva neuronas en función de una probabilidad pasada como argumento. El *overfitting* ocurre cuando nuestra neurona le asigna mucha importancia a un conjunto pequeño de los valores de la salida de la capa anterior como característica predictiva. Esta capa ayuda a prevenir el textitoverfitting.

  La capa `Dropout`, lleva a 0 de manera aleatoria distintos valores del vector de entrada en cada una de las iteraciones de nuestro entrenamiento. Poner esta capa entre la última capa oculta y la de salida podría ser una mala decisión.

- `Flatten`: Convierte los elementos de la matriz de imágenes de entrada en un array plano. Esto hace que nos acerquemos a nuestro objetivo pues la salida es un array binario de tamaño 10.

- `Dense`: Es una capa completamente conectada o *fully-connected*. Devuelve una salida del tamaño indicado y usa la activación indicada, en mi caso `relu` o `softmax`.

3.3. Entrenando el modelo

El entrenamiento del modelo se va a realizar usando la función `fit_generator` sobre el modelo. Hay que indicar algunos parámetros:

- `batch_size`: tamaño del batch en el que se dividen los datos.

- `epoch`: número de épocas del entrenamiento.

Una vez añadidas las capas que se desean hay que compilar el modelo usando `compile` indicando cuál es la métrica a optimizar y el optimizador. En mi caso la métrica elegida es el `accuracy`.

En el cálculo de la pérdida se ha utilizado entropía cruzada mediante la función `categorical_crossentropy`. Asimismo, se ha utilizado RMSPROP (Root Mean Square Propagation) el cual en lugar de mantener un acumulado de los gradientes, utiliza el concepto de ventana para únicamente considerar en el aprendizaje los más recientes.

```
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)

# Inicio tiempos
start.time <- proc.time()
start.time2 <- Sys.time()

history <- model %>%
  fit_generator(
    generator = train_generator_flow,
    validation_data = validation_generator_flow,
    steps_per_epoch = 10,
    epochs = 10
```

```
  )

# Fin tiempos
end.time <- proc.time()
end.time2 <- Sys.time()
```

Por otro lado existe como ya se ha dicho un conjunto de tamaño del 20 % del de entrenamiento para usarlo como conjunto de validación.

Tal y como se observa se mide el tiempo de hasta dos formas distintas para saber cuánto tarda en entrenarse el modelo en cuestión.

4. Manual de usuario

La práctica ha sido desarrollada en lenguaje de programación R. El sistema operativo usado ha sido Windows 10 ya que la instalación de `keras` y `tensorflow` fue mejor bajo ese sistema operativo.

La versión de R usada es la última disponible hasta el momento, la 4.1.0 [2]. El IDE usado en la implementación y ejecución del software es RStudio en su versión 1.4.1717 [12].

Los paquetes necesarios para la ejecución de la práctica son: `tidyverse`, `keras`, `caret`, `mice`, `rpart`,`rpart.plot` y `reticulate`.

El último paquete indicado es un paquete que permite que Python se ejecute por abajo de R. Al hacer esa instalación obtenemos una versión de Python, la 3.7.10. A parte, se instaló la última versión de Python: 3.9.5.

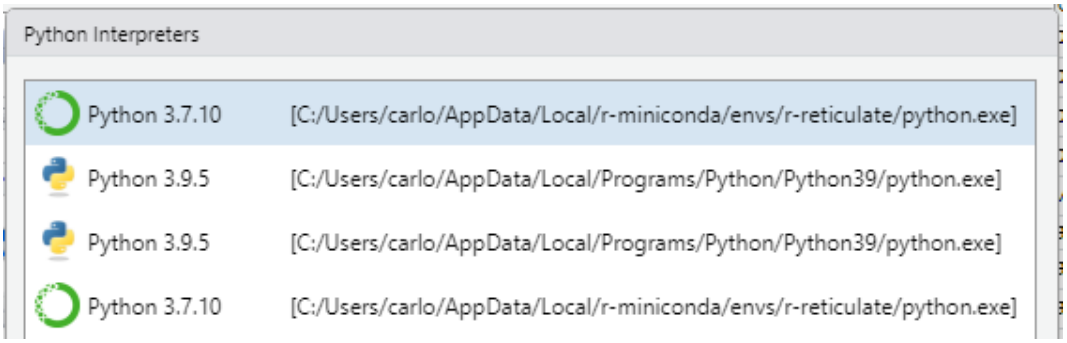


Imagen 1: Versiones de Python disponibles en RStudio

Los datos han de guardarse en una carpeta `images` dentro de otra carpeta `data` en la ruta del proyecto. Se ha usado el conjunto mini (50 imágenes) solo para el análisis de los datos y el conjunto medium (10.000 imágenes) para el análisis y el aprendizaje de los modelos. Por tanto las dos rutas que deben estar construidas en el proyecto son:

- `./data/images/mini50_twoClasses/`
- `./data/images/medium10000_twoClasses/`

Para comenzar ya sólo es necesario un paso más: borrar las imágenes que están mal o no se pueden leer. Para ello hay que ejecutar el fichero Python dado como material de la asignatura `clean_images.py`. El entorno ya está listo para ejecutar los diferentes archivos:

- `p2-eda-mini50.Rmd`: el análisis de datos sobre el conjunto mini de 50 imágenes.
- `p2-eda-medium10000.Rmd`: el análisis de datos sobre el conjunto medium de 10.000 imágenes.
- `p2-cnn.Rmd`: la red neuronal básica dada por Juan Gómez el profesor de la asignatura.

- `p2-cnn_mejorado.Rmd`: un modelo mejorado a partir del básico con el objetivo de conseguir un porcentaje de acierto superior.
- `p2-cnn_data_augmentation.Rmd`: el modelo que usa el aumento de datos de las imágenes.
- `p2-cnn_batch_normalization.Rmd`: modelo que utiliza una normalización por lotes.
- `p2-cnn_fine_tuning.Rmd`: modelo de transferencia de aprendizaje.
- `p2-cnn_ensemble.Rmd`: el código correspondiente al ejemplo de ensamblado de modelos.

Como se puede observar todos los ficheros tienen extensión RMarkdown ya que para mí ha sido lo más cómodo de cara a la implementación de los mismos. Se ejecutan, por ejemplo, desde el IDE de RStudio.

5. Análisis exploratorio

En esta sección se va a llevar a cabo un análisis exploratorio de los datos distinguiendo entre el conjunto mini y el medium. Sin embargo, lo primero es realizar la lectura de los datos.

5.1. Lectura de datos

La cantidad de imágenes que se manejan en este clasificador hacen que se necesario usar una de las herramientas propuestas en clase, los generadores de flujo. De forma dinámica va generando lotes de imágenes. Dichas imágenes sufren un escalado de 1/255.

```
train_images_generator <- image_data_generator(rescale = 1/255)

train_generator_flow <- flow_images_from_directory(
  directory = train_images_dir,
  generator = train_images_generator,
  class_mode = 'categorical',
  batch_size = 128,
  target_size = c(64, 64)          # (w x h) ->(64 x 64)
)
```

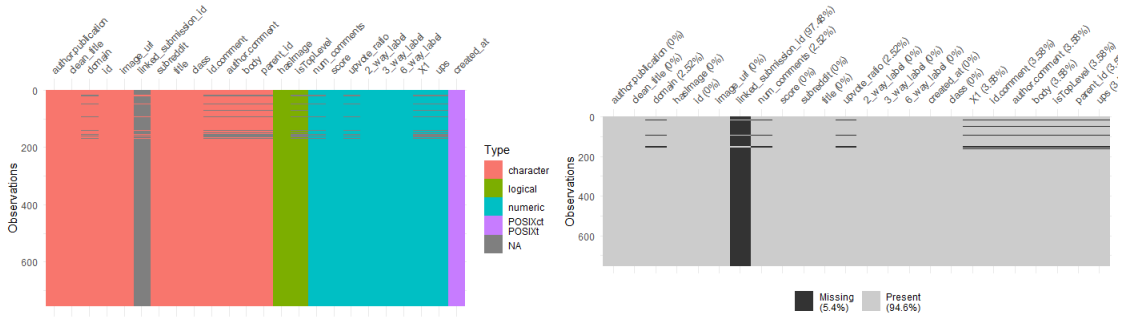
El tamaño de las imágenes (`target`) es de 64×64. El tamaño de los lotes que se pasan es de 128 y el modo de clasificación es `categorical`.

5.2. Conjunto *mini*

El análisis de los datos va a comenzar sobre el conjunto *mini* que era aquel que tenía 50 instancias. Uno de los primeros datos que se desea saber en un problema de clasificación (y aún más de clasificación binaria) es si los datos están balanceados. Observemos la distribución de clases:

Imagen 2: Balanceamiento de los datos (mini)

Otra de las primeras informaciones que me gusta poseer son los atributos del problema, los tipos de los mismos y los valores perdidos. Para ello la librería `visdat` de R es muy útil.



(a) Visualización de las variables (b) Visualización de los valores perdidos

Imagen 3: Visualización de datos (mini)

El atributo `linked_submission_id` está prácticamente repleto de valores perdidos. Usando esta librería también se puede estudiar la correlación entre las variables numéricas. Existe una correlación alta entre `3_way_label` y `6_way_label` pero esto es evidente. También existe una gran relación entre `num_comments` y `score`.

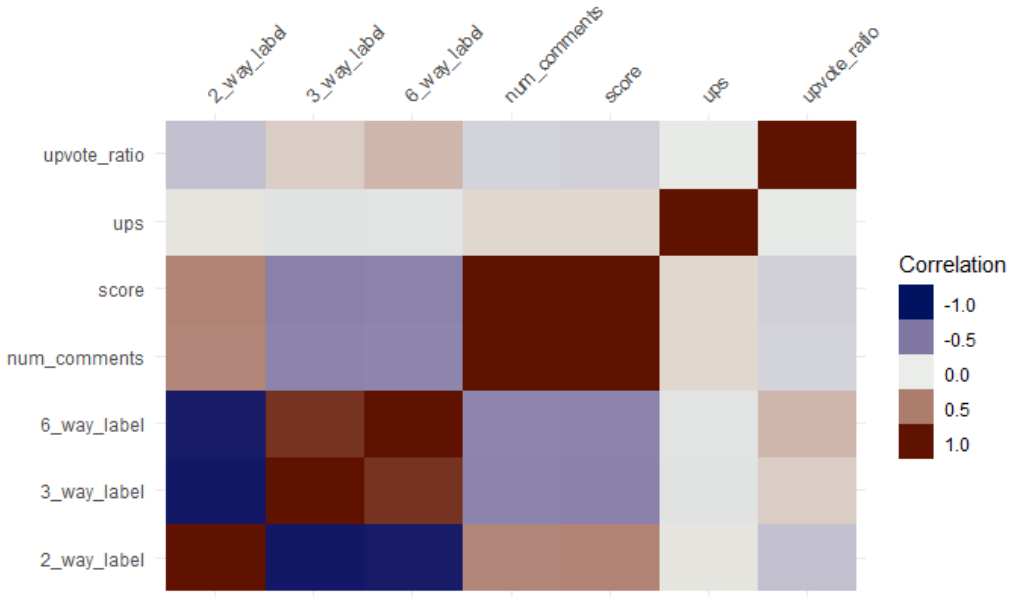


Imagen 4: Correlación de los datos (mini)

A continuación vamos a ver mediante el parámetro `created_at` la evolución temporal de la publicación de las noticias. Las conclusiones que se pueden extraer son que en los últimos años se ha publicado más que en los primeros y que antes había más desinformación que ahora.

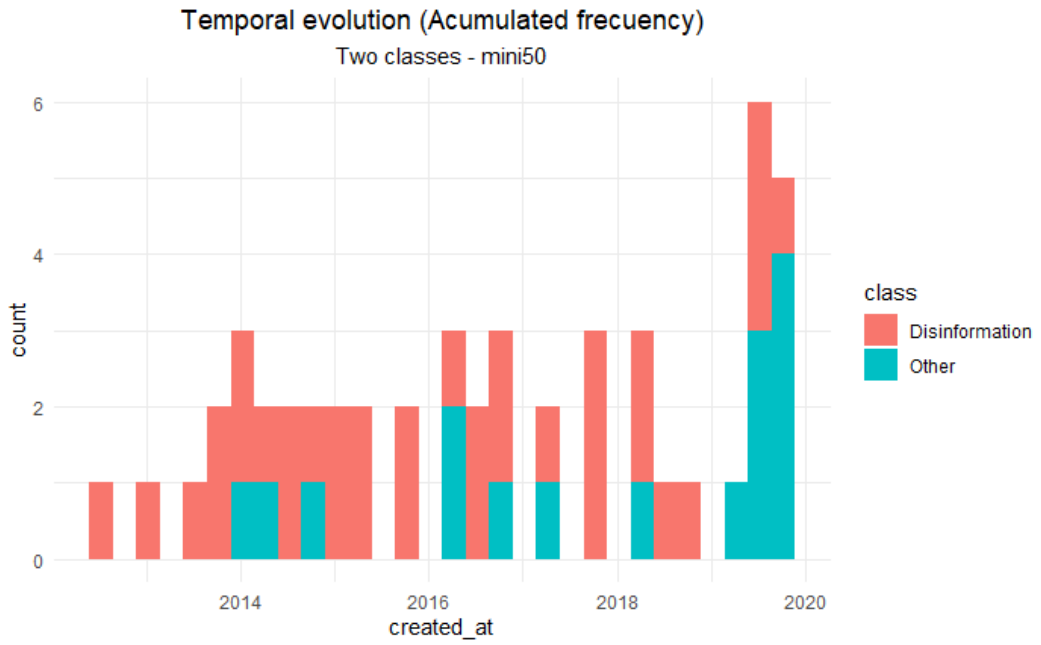
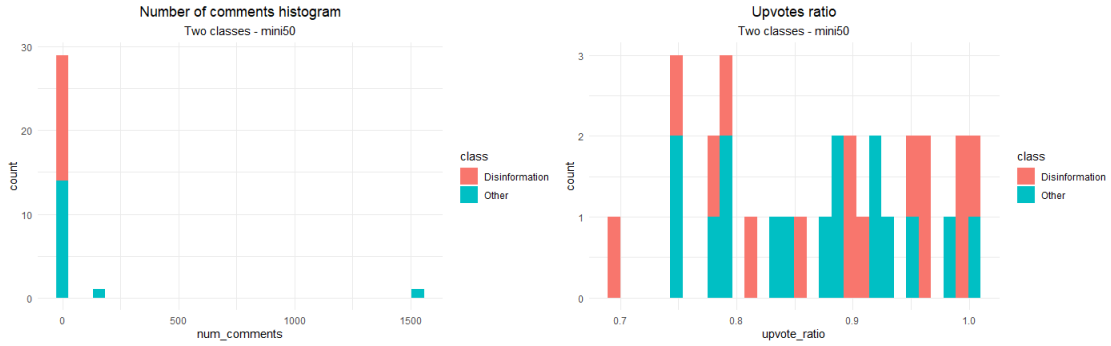


Imagen 5: Evolución temporal de los datos (mini)

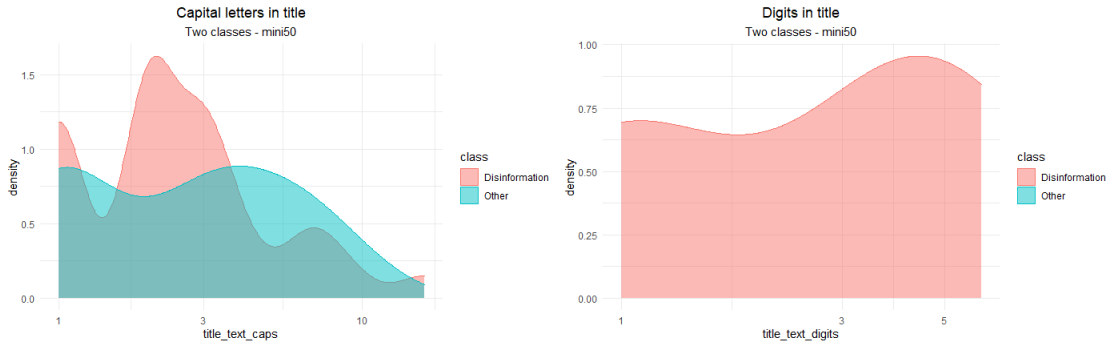
Prestando atención a las variables asociadas al número de comentarios y al ratio de votos positivos podemos extraer los siguientes histogramas en donde se distingue por clases.



(a) Número de comentarios (b) Ratio de votos positivos

Imagen 6: Número de comentarios y ratio de votos positivos (mini)

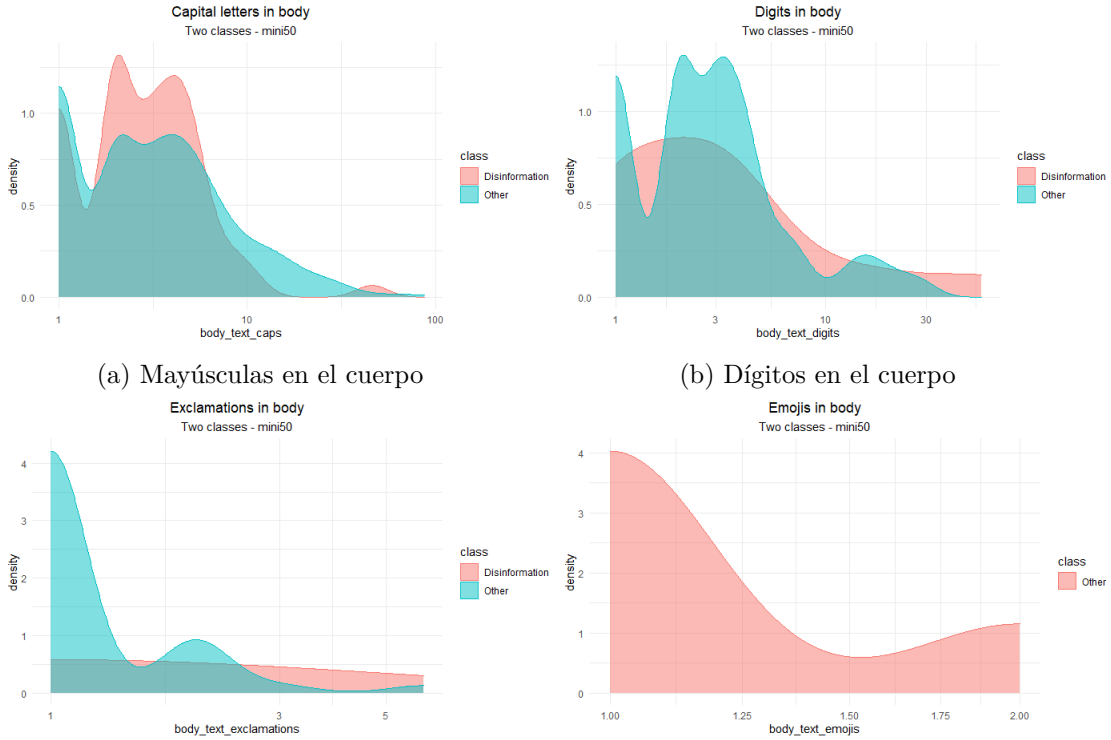
Continuamos este análisis de datos analizando información del título de la imagen e información del cuerpo (`body`). Cuando hay 2 o 3 mayúsculas en el título es más frecuente que dicha noticia provenga de desinformación. Por otro lado parece que las noticias que contienen dígitos en el título provienen de desinformación.



(a) Mayúsculas en el título (b) Dígitos en el título

Imagen 7: Información del título (mini)

Respecto al cuerpo de texto las gráficas para este conjunto de datos son las que se ven a continuación. Hay diferencias visibles pero tampoco nada demasiado extraño.

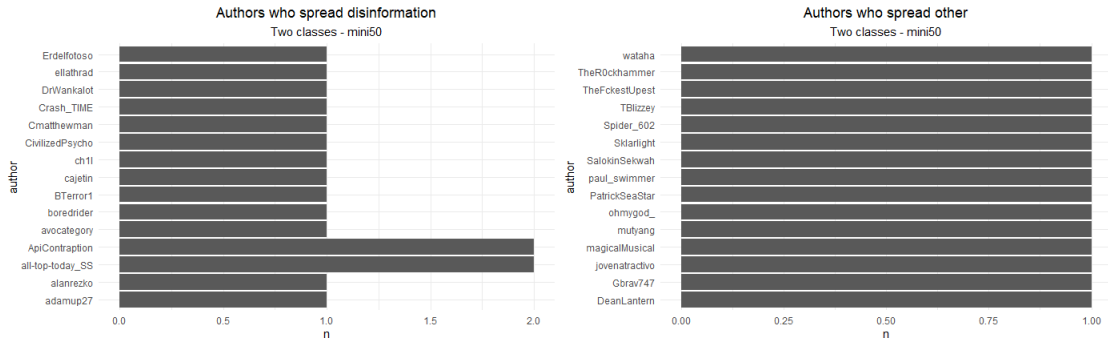


(a) Mayúsculas en el cuerpo (b) Dígitos en el cuerpo

(c) Mayúsculas en el cuerpo (d) Emojis en el cuerpo

Imagen 8: Información del cuerpo (mini)

En último lugar se van a mostrar unas gráficas de barras horizontales con aquellos autores que más publican desinformación y con los que más publican información. Dichas gráficas son las que se observan a continuación:

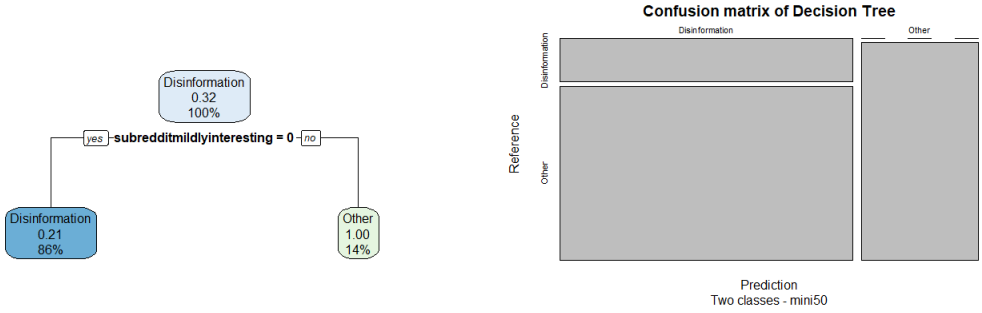


(a) Desinformación (b) Noticias verdaderas

Imagen 9: Autores que más publican (mini)

Los autores `ApiContraption` y `all-top-today_SS` son los que más noticias falsas publican.

Como curiosidad adicional se ofrecía como material por parte del profesor de la asignatura un clasificador de árbol de decisión construido sobre los datos que se poseen de las imágenes sin aprender de las imágenes en sí. El árbol hallado tras la optimización de parámetros (`cp=0.1` es el óptimo) y su matriz de confusión son los siguientes:



(a) Clasificador de árbol (b) Matriz de confusión

Imagen 10: Árbol de decisión (mini)

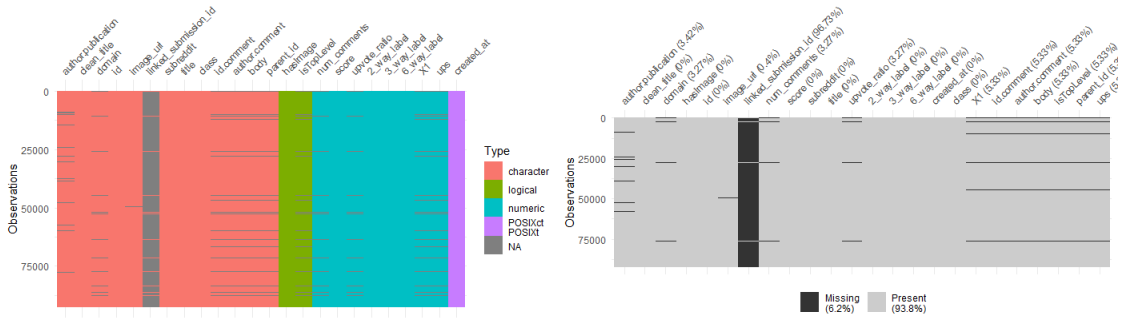
Con un modelo como este árbol tan sencillo se consigue un acierto excelente.

5.3. Conjunto *medium*

El análisis de los datos va a continuar sobre el conjunto *medium* que era el que tenía 10.000 instancias. Comenzamos viendo la distribución de los datos y si están balanceados:

Imagen 11: Balanceamiento de los datos (medium)

Al igual que antes me gusta poseer son los atributos del problema, los tipos de los mismos y los valores perdidos.



(a) Visualización de las variables

(b) Visualización de los valores perdidos

Imagen 12: Visualización de datos (medium)

El atributo `linked_submission_id` está prácticamente repleto de valores perdidos. Al igual que antes vamos a ver la correlación entre las variables numéricas.

Tal y como ya ocurría existe una correlación alta entre `3_way_label` y `6_way_label` pero esto es evidente. También existe una gran relación entre `num_comments` y `score`. En general es muy similar al del caso anterior.

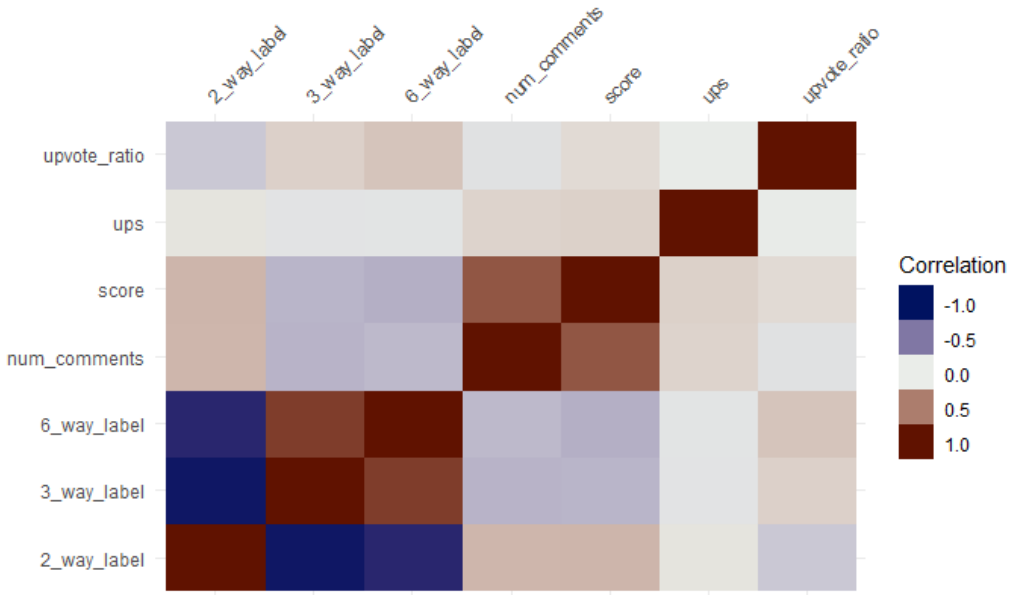


Imagen 13: Correlación de los datos (medium)

A continuación vemos mediante el parámetro `created_at` la evolución temporal de la publicación de las noticias.

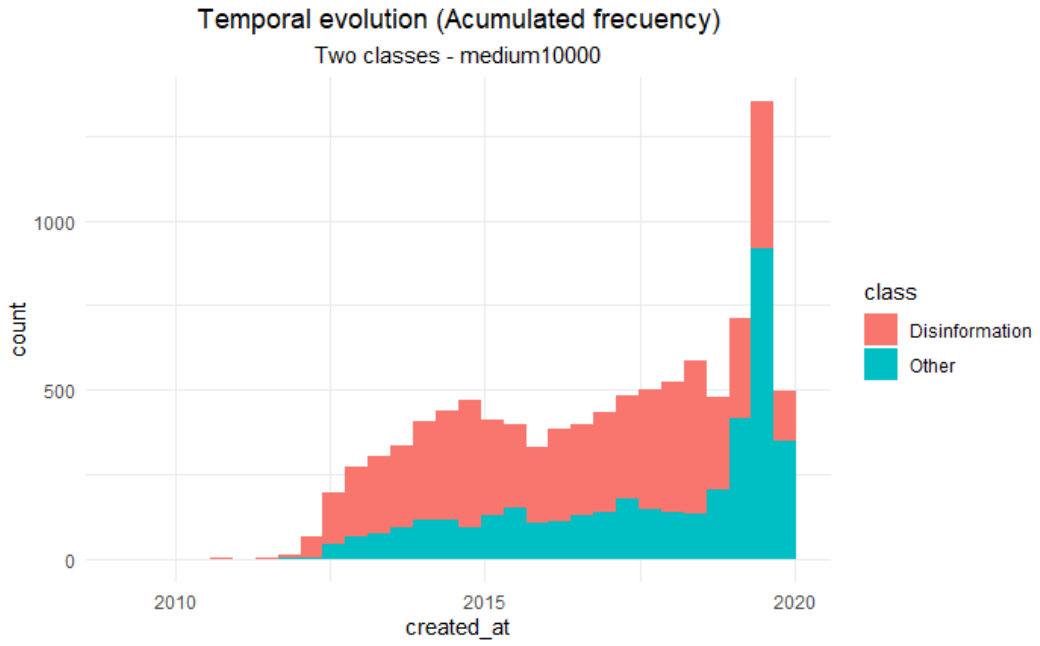
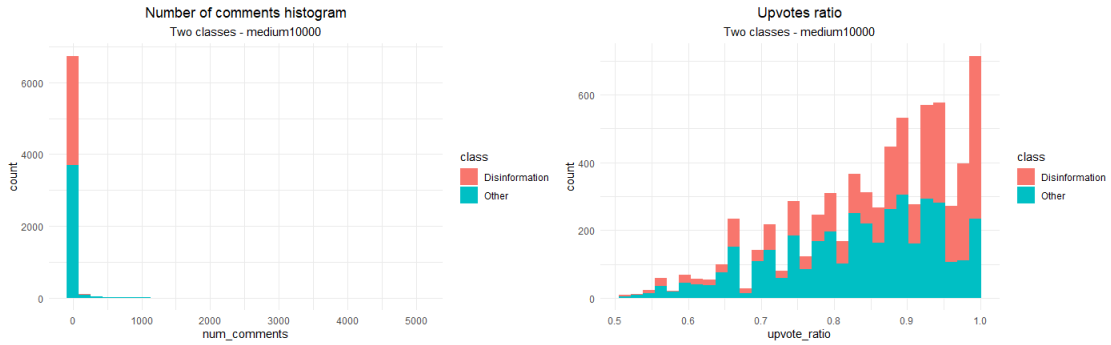


Imagen 14: Evolución temporal de los datos (medium)

Las conclusiones que se pueden extraer son que en los últimos años se ha publicado más que en los primeros y que antes había más desinformación que ahora. Esto se ve de

una manera más clara y más segura que en el conjunto *mini*.

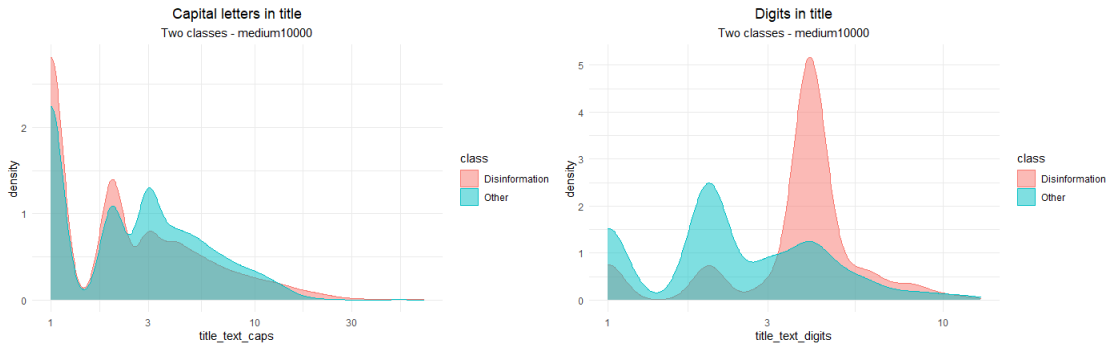
Prestando atención a las variables asociadas al número de comentarios y al ratio de votos positivos podemos extraer los siguientes histogramas en donde se distingue por clases.



(a) Número de comentarios (b) Ratio de votos positivos

Imagen 15: Número de comentarios y ratio de votos positivos (medium)

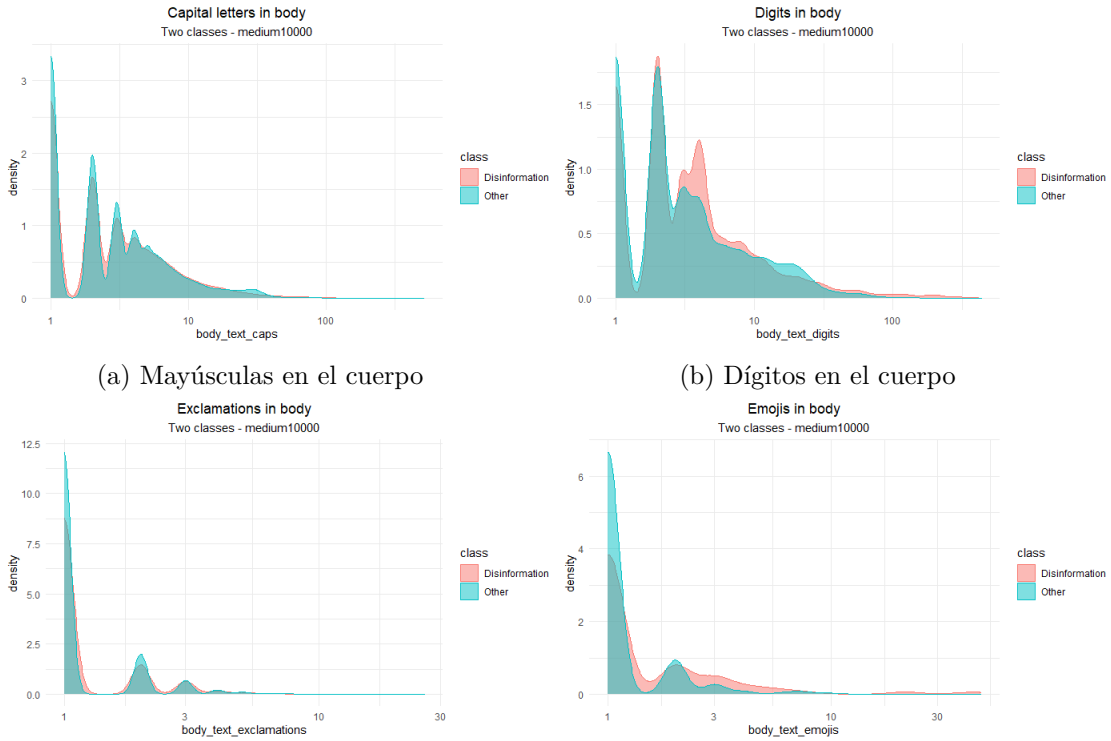
Continuamos este análisis de datos analizando información del título de la imagen e información del cuerpo (`body`). Cuando hay 2 o 3 mayúsculas en el título es más frecuente que dicha noticia provenga de desinformación. Por otro lado parece que las noticias que contienen dígitos en el título provienen de desinformación.



(a) Mayúsculas en el título (b) Dígitos en el título

Imagen 16: Información del título (medium)

Respecto al cuerpo de texto las gráficas para este conjunto de datos son las que se ven a continuación. Hay diferencias visibles pero tampoco nada demasiado extraño.



(a) Mayúsculas en el cuerpo

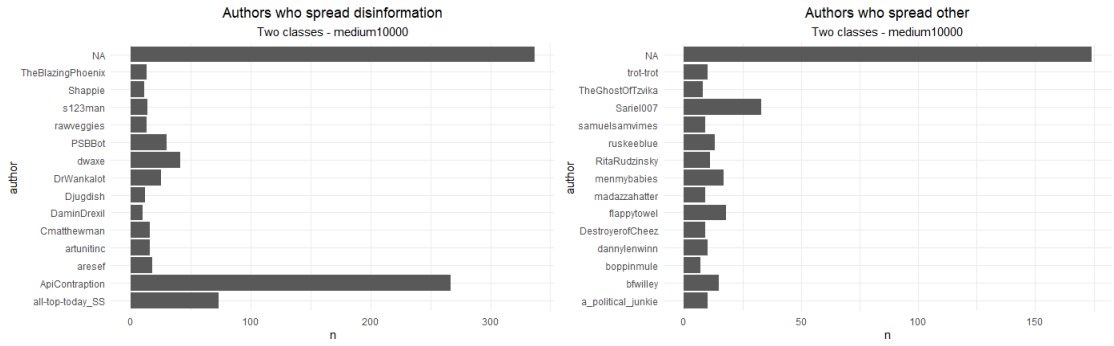
(b) Dígitos en el cuerpo

(c) Mayúsculas en el cuerpo

(d) Emojis en el cuerpo

Imagen 17: Información del cuerpo (medium)

Finalmente, se van a mostrar unas gráficas de barras horizontales con aquellos autores que más publican desinformación y con los que más publican información. Dichas gráficas son las siguientes:



(a) Desinformación

(b) Noticias verdaderas

Imagen 18: Autores que más publican (medium)

De igual manera que ya ocurrió, vuelven a ser los autores `ApiContraption` y `all-top-today_SS` los que más noticias falsas publican.

6. Modelo base

El modelo base del que partimos es el implementado por el profesor de la asignatura, Juan Gómez. Se trata de una red neuronal convolutiva. Es un modelo secuencial en el que se intercalan 4 capas de convolución con cuatro capas de pooling. Después sigue una capa fully connected para finalmente converger en las capas densas, la primera con 512 salidas y la última con 2 salidas que es la solución del problema de clasificación.

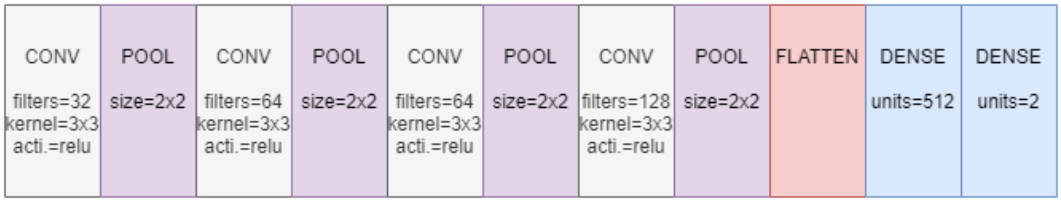


Imagen 19: Modelo de aprendizaje básico

En la imagen 19 se observa gráficamente el modelo. A nivel de código:

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32,  kernel_size = c(3, 3), activation = "relu",
        input_shape = c(64, 64, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64,  kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 2, activation = "softmax")
```

Tras compilar y entrenar el modelo los resultados son los siguientes:

Conjunto	Accuracy	Loss	Tiempo
Test	62,50 %	67,53 %	6,40 min

Mediante `predict_generator()` el modelo predice la clasificación de imágenes y podemos calcular la matriz de confusión.

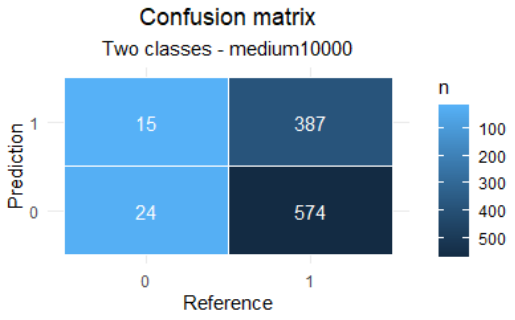


Imagen 20: Matriz de confusión del modelo básico

7. Modelo avanzado

El modelo avanzado parte del básico pero añadiendo nuevas capas ocultas. La El nuevo tipo de capa introducida es una capa Dropout que según la probabilidad que se le indique desactiva neuronas. Esta capa evita muchos problemas de sobreajuste u *overfitting*.

La nueva red neuronal convolutiva es un modelo secuencial que comienza con una capa convolutiva para después intercalar 4 capas de convolución con cuatro capas de pooling. A continuación sigue una capa fully connected para finalmente converger en las capas densas, la primera con 512 salidas, posteriormente 128 y 64 y finalmente con 2 salidas que es la solución del problema de clasificación. Estas capas se intercalan capas Dropout que desactivan neuronas de la red con una probabilidad de 0.1 la cual no es muy alta.

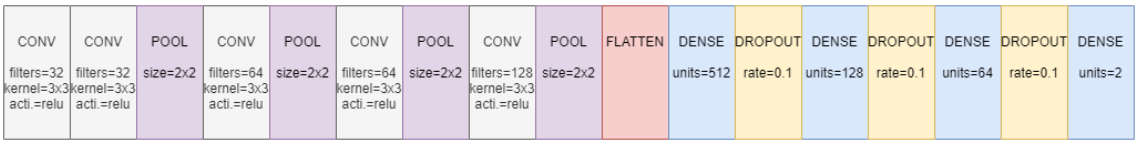


Imagen 21: Modelo de aprendizaje avanzado

En la imagen 21 se observa gráficamente el modelo. El código del nuevo modelo es:

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32,  kernel_size = c(3, 3), activation = "relu",
        input_shape = c(64, 64, 3)) %>%
  layer_conv_2d(filters = 32,  kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64,  kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 2, activation = "softmax")
```

El número de épocas usadas también ha sido 10. Esta elección ha sido meditada ya que se quiere comparar los modelos en calidad y tiempo de ejecución. Si el número de épocas es diferente no puede realizar una comparación de ese tipo. Además sabemos que un número de épocas excesivamente grande puede hacer que el modelo se sobreentrene.

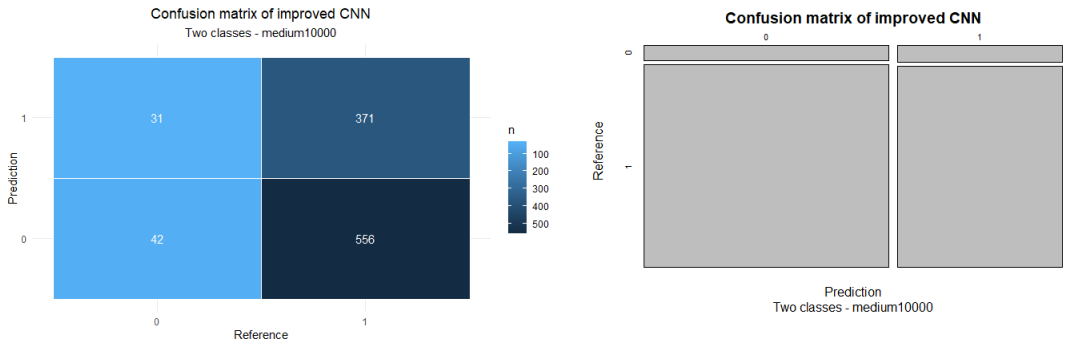
Tras compilar y entrenar el modelo los resultados son los siguientes:

```
Epoch 1/10
10/10 [==============================] - 164s 17s/step - loss: 1.0471 - accuracy: 0.6039 -
    val_loss: 0.6571 - val_accuracy: 0.6326
Epoch 2/10
Epoch 1/10
10/10 [==============================] - 39s 4s/step - loss: 0.7026 - accuracy: 0.5867 -
    val_loss: 0.6837 - val_accuracy: 0.6326
Epoch 2/10
10/10 [==============================] - 37s 4s/step - loss: 0.6734 - accuracy: 0.6219 -
    val_loss: 0.6590 - val_accuracy: 0.6326
Epoch 3/10
10/10 [==============================] - 39s 4s/step - loss: 0.6697 - accuracy: 0.6031 -
    val_loss: 0.6543 - val_accuracy: 0.6326
Epoch 4/10
10/10 [==============================] - 41s 4s/step - loss: 0.6606 - accuracy: 0.6242 -
    val_loss: 0.6499 - val_accuracy: 0.6326
Epoch 5/10
10/10 [==============================] - 38s 4s/step - loss: 0.6918 - accuracy: 0.6031 -
    val_loss: 0.6604 - val_accuracy: 0.6326
Epoch 6/10
10/10 [==============================] - 33s 3s/step - loss: 0.6637 - accuracy: 0.6086 -
    val_loss: 0.6454 - val_accuracy: 0.6326
Epoch 7/10
10/10 [==============================] - 32s 3s/step - loss: 0.6633 - accuracy: 0.6164 -
    val_loss: 0.6452 - val_accuracy: 0.6326
Epoch 8/10
10/10 [==============================] - 34s 3s/step - loss: 0.6737 - accuracy: 0.6172 -
    val_loss: 0.6633 - val_accuracy: 0.6326
Epoch 9/10
10/10 [==============================] - 32s 3s/step - loss: 0.6626 - accuracy: 0.5984 -
    val_loss: 0.6557 - val_accuracy: 0.6326
Epoch 10/10
10/10 [==============================] - 32s 3s/step - loss: 0.6570 - accuracy: 0.6125 -
val_loss: 0.6438 - val_accuracy: 0.6326
```

Sobre el conjunto de test los resultados son los siguientes:

Conjunto	**Accuracy**	**Loss**	**Tiempo**
Test	67,97 %	62,09 %	6,20 min

Para la semilla elegida desde luego son muy buenos resultados. Mediante `predict_generator()` el modelo predice la clasificación de imágenes y podemos calcular la matriz de confusión.



(a) Tipo 1 (b) Tipo 2

Imagen 22: Matriz de confusión

8. Mejoras del aprendizaje

8.1. *Data Augmentation*

La primera mejora de aprendizaje que se propone sobre los datos es el aumento de datos o `Data Augmentation` [8]. Esta técnica es especialmente interesante cuando los datos que se poseen son escasos y se puede ampliar este conjunto facilitando la labor de aprendizaje de los modelos. La gran cuestión es cómo se pueden aumentar los datos. La respuesta es realizando transformaciones de los mismos como por ejemplo rotaciones, escalados y simetrías entre otros.

Hay que tener cuidado con este aumento de datos ya que puede tener consecuencias negativas como el sobreaprendizaje. A cambio de esto es posible tal vez aumentar los datos dando variedad.

El código que es necesario incluir es el siguiente [11]:

```
data_augmentation_datagen <- image_data_generator(
  rescale = 1/255,
  rotation_range = 40,
  width_shift_range = 0.2,
  height_shift_range = 0.2,
  shear_range = 0.2,
  zoom_range = 0.2,
  horizontal_flip = TRUE,
  fill_mode = "nearest"
)
```

Tras compilar y entrenar el modelo los resultados son los siguientes:

Conjunto	**Accuracy**	**Loss**	**Tiempo**
Test	59,38 %	75,20 %	6,30 min

El resultado no es muy satisfactorio. Las pruebas realizadas con el aumento de datos sobre el modelo básico ofrecen resultados parecidos.

8.2. *Batch Normalization*

Una de las mejoras que se proponen es comprobar si realizar una normalización por lotes o `BatchNormalization` puede dar resultados satisfactorios. Sobre la red ya definida se van a añadir estas normalizaciones [15].

Esta técnica consiste en agregar un paso adicional entre la neurona y la función de disparo para normalizar la salida. En esta normalización se usan la media y la varianza de los datos o en su defecto de cada lote.

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32,  kernel_size = c(3, 3), activation = "relu",
        input_shape = c(64, 64, 3)) %>%
  layer_conv_2d(filters = 32,  kernel_size = c(3, 3), activation = "relu") %>%
  layer_batch_normalization(epsilon = 0.01) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64,  kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
  layer_batch_normalization(epsilon = 0.01) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_batch_normalization() %>%
  layer_flatten() %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 2, activation = "softmax")
```

Se han introducido hasta tres capas de normalización por lotes, la última antes de la capa que tranforma el dataset de imágenes en vecotres. Tras compilar y entrenar el modelo los resultados son los siguientes:

Conjunto	**Accuracy**	**Loss**	**Tiempo**
Test	$56,25\,\%$	$68,63\,\%$	6,11 min

Los resultados han empeorado al modelo que ya se tenía. Tras cambiar de sitio las capas no se consigue mucha mejora.

9. Transferencia de aprendizaje o *fine tuning*

La transferencia de aprendizaje o *fine tuning* [10] consiste en usar redes que han sido previamente entrenadas de modo que su aprendizaje pueda usarse sobre el conjunto de datos a tratar.

La red entrenada que voy a usar sobre el conjunto de datos de Fakeddit es `VGG-16` que ha sido estudiada en el desarrollo de la asignatura [9].

Esta red está compuesta por las siguientes capas:

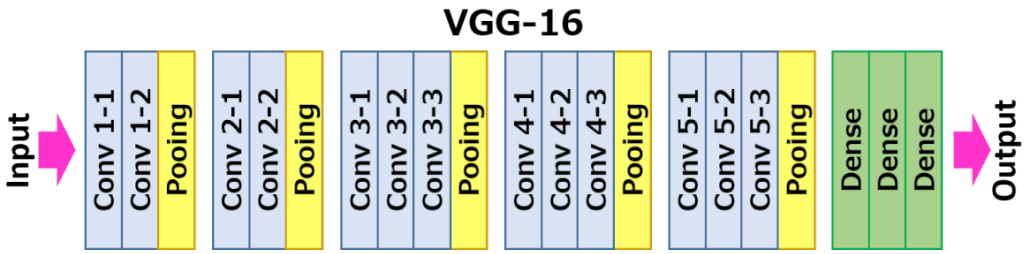


Imagen 23: Modelo `VGG-16` [9]

En esta otra imagen se puede observar mejor, ya que se descomponen cada una de las capas de la red:

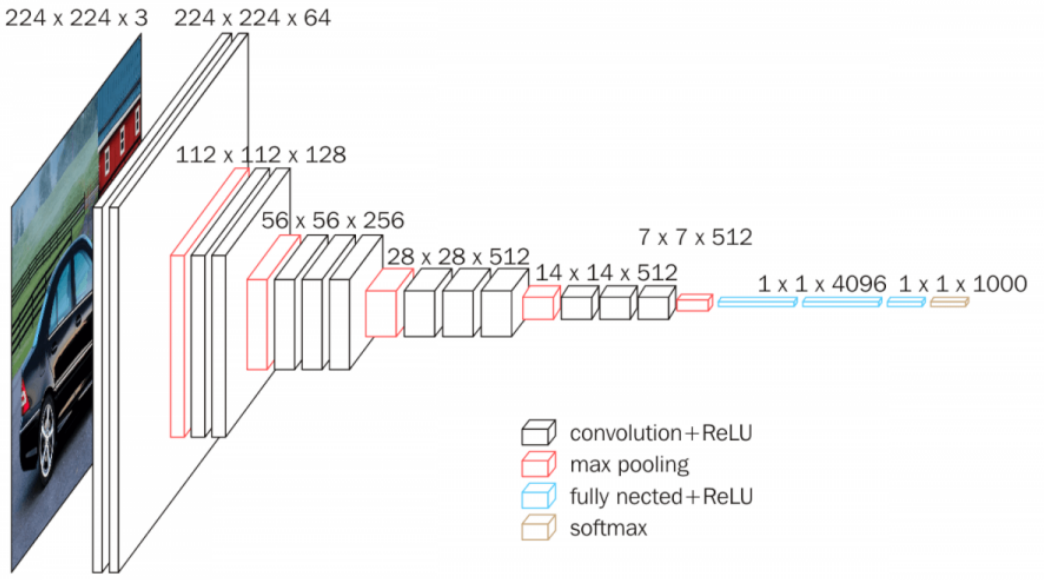


Imagen 24: Funcionamiento del modelo `VGG-16` [9]

Los pesos elegidos son los obtenidos del entrenamiento de `imagenet`. Esta elección está tomada del material impartido en clase, en concreto del ejemplo del clasificador de imágenes de perros y gatos. El código es el siguiente [7] y [6]:

```
conv_base <- application_vgg16(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(64, 64, 3)
)

freeze_weights(conv_base)

model <- keras_model_sequential() %>%
  conv_base %>%
  layer_flatten() %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 2, activation = "sigmoid")

unfreeze_weights(conv_base, from = "block3_conv1")
```

Nótese que se ha descongelado la capa `block3_conv1`. Tras compilar y entrenar el modelo los resultados son los siguientes:

Conjunto	**Accuracy**	**Loss**	**Tiempo**
Test	$52,34\,\%$	$71,22\,\%$	17,95 min

El tiempo de ejecución ha aumentado considerablemente y sin embargo los resultados no son mejores. Quizás el número de épocas sea bajo pero las limitaciones computacionales y la intención de comparar los diferentes modelos obligan a ello.

10. Métodos *ensemble* e híbridos

Otra técnica común usada en Aprendizaje Automático y también en Aprendizaje Profundo es el ensamblamiento [14] de redes neuronales. Se pueden combinar diferentes modelos con el fin de mejorar los resultados a obtener en donde la varianza pretende ser reducida.

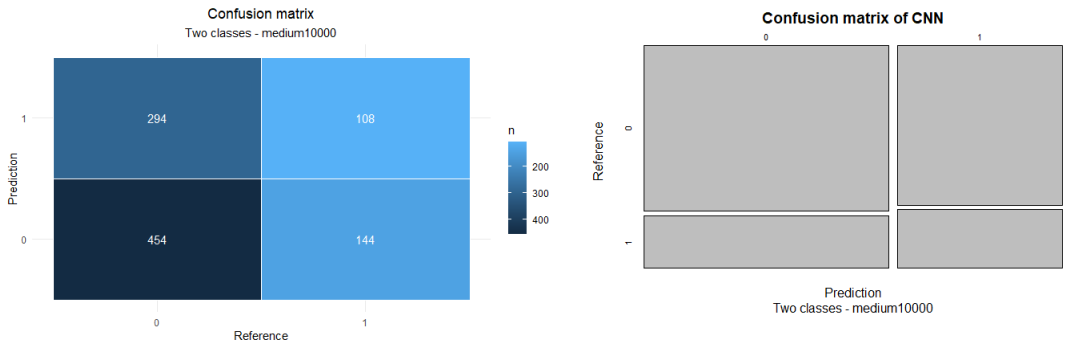
El número de combinaciones es inmenso pero lamentablemente los resultados que he obtenido no se acercaban a lo que ya sabemos que se puede conseguir. Finalmente dejándome guiar por los pasos de mi profesor he usado el *ensemble* explicado en clase: `VGG16` + `MobileNet` [4] sobre este conjunto de datos.

```
vgg16_base <- application_vgg16(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(64, 64, 3)
)

mobile_base <- application_mobilenet_v2(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(64, 64, 3)
)

freeze_weights(vgg16_base)
freeze_weights(mobile_base)

model_input  <- layer_input(shape=c(64, 64, 3))

model_list   <- c(vgg16_base(model_input) %% layer_flatten(),
                  mobile_base(model_input) %% layer_flatten())
```

Tras compilar y entrenar el modelo los resultados son los siguientes:

Conjunto	Accuracy	Loss	Tiempo
Test	59,38 %	64,13 %	19,90 min

La matriz de confusión del *ensemble*. Esta matriz de confusión es una de las que más me ha gustado a nivel de clasificación.



(a) Tipo 1 (b) Tipo 2

Imagen 25: Matriz de confusión

11. Discusión de resultados

El tamaño de la entrada se indica en todos los modelos como 64×64. En esta sección se van a discutir los resultados. Para ello se han recogido los diferentes resultados hallados en la siguiente tabla. A partir de ella se podrán discutir diferentes aspectos.

Esta discusión pretende ser lo más objetiva posible así que para ello se ha usado el mismo número de épocas y el mismo número de pasos por época en cada modelo. Cada modelo tiene una complejidad y esta determinará para estos datos el acierto y el tiempo necesitados en el ajuste.

Modelo	**Accuracy**	**Loss**	**Tiempo**
Modelo base	$62,50\,\%$	$67,53\,\%$	6,40 min
Modelo avanzado	$67,97\,\%$	$62,09\,\%$	6,20 min
Data Augmentation	$59,38\,\%$	$75,20\,\%$	6,30 min
Batch Normalization	$56,25\,\%$	$68,63\,\%$	6,11 min
Fine tuning	$52,34\,\%$	$71,22\,\%$	17,95 min
Ensemble	$59,38\,\%$	$64,13\,\%$	19,90 min

El mejor modelo (en *accuracy*) es el **Modelo avanzado**. El acierto que tiene es sorprendente. Tras realizar pruebas con otra semilla baja el accuracy un poco por lo que parece que el factor azar ayuda.

Me llama la atención que al incluir la normalización por lotes el acierto caiga de una manera notable. En un estudio real habría que aunar esfuerzos y realizar más pruebas porque colocando estas capa de forma correcta y bajo los parámetros óptimos en el dominio del problema no tengo duda de que los resultados deben ser cuanto menos mejores.

Las redes ya entrenadas no han dado un gran acierto en el conjunto de test pero las matrices de confusión eran bastante satisfactorias. En particular el del ensemble. Este modelo, como es lógico, es el que más tiempo ha tomado en su entrenamiento y ajuste a los datos.

Los modelos iniciales tienen tiempo totalmente similares aunque cabe destacar que el segundo que menos tiempo consume es el mejor en acierto. Como trabajo futuro queda probar otro tipo de mejoras en el aprendizaje, la mayoría de las cuales han sido estudiadas en esta asignatura y el mejor ajuste de los modelos para lograr aciertos aún mejores.

Tras poseer los resultados finales y sabiendo que durante el desarrollo de los modelos alguna vez me dieron que siempre clasificaba desinformación, que es la clase mayoritaria veo necesario balancear los datos antes de comenzar el entrenamiento y el aprendizaje.

12. Conclusiones

La primera conclusión que me gustaría destacar es la profundidad y múltiples opciones que existen actualmente en Deep Learning. Hay muchísimos modelos que pueden trabajar francamente bien sobre conjuntos de imágenes. Las 'posibilidades con las mejorar el aprendizaje son muy diversas y pueden ir desde un aumento de datos hasta regularizar más el modelo para evitar el sobreajuste.

Es por esto que el análisis de los datos y hacer un estudio previo resulta imprescindible. Asimismo es bien sabido que cuanto más balanceados estén más fácil será la labor de aprendizaje. En este caso particular podría ser útil realizar algún subsampling y forzar un mejor balanceo entre clases.

El correcto tratamiento de los datos así como la lectura mediante flujo es otro de los pasos indispensables en este tipo de estudios ya que los ordenadores actuales no tienen la potencia necesaria para gestionar todo.

Las herramientas existentes hoy en día como Tensorflow y su API Keras facilitan la labor a los especialistas de una manera enorme. Sin duda, la disposición de herramientas de tan alto nivel es uno de los grandes avances.

Este conjunto de datos parece que viene de un foro en donde es frecuente la desinformación por diferentes fines. La configuración final del modelo debe ser aquel que obtenga buenos resultados, en un tiempo razonable, sin sobreaprender de los mismos. En este caso y a la vista de los resultados, el modelo que yo usaría sería el modelo avanzado.

13. Bibliografía

[1] *Capas de Keras*. URL: https://keras.io/api/layers/.

[2] CRAN. *R*. URL: https://cran.r-project.org/bin/windows/base/t.

[3] entitize. *GitHub Fakeddit*. URL: https://github.com/entitize/Fakeddit.

[4] Kaggle. *Transfer Learning with VGG-16 and MobileNet*. URL: https://www.kaggle.com/timmate/transfer-learning-with-vgg-16-and-mobilenet.

[5] *Keras*. URL: https://keras.io/.

[6] Keras. *Freeze and unfreeze weights*. URL: https://www.kaggle.com/timmate/transfer-learning-with-vgg-16-and-mobilenet.

[7] Keras. *VGG16 and VGG19 models for Keras*. URL: https://keras.rstudio.com/reference/application_vgg.html.

[8] Nanonets. *Data Augmentation*. URL: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/.

[9] Neurohive. *VGG16 – Convolutional Network for Classification and Detection*. URL: https://neurohive.io/en/popular-networks/vgg16/.

[10] pyimagesearch. *Fine-tuning*. URL: https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/.

[11] Tensorflow for R. *Generate batches of image data with real-time data augmentation*. URL: https://www.kaggle.com/timmate/transfer-learning-with-vgg-16-and-mobilenet.

[12] *RStudio*. URL: https://www.rstudio.com/products/rstudio/download/.

[13] *Tensorflow*. URL: https://www.tensorflow.org/.

[14] Unipython. *Ensemble methods o métodos de conjunto*. URL: https://unipython.com/ensemble-methods-metodos-conjunto/.

[15] Wikipedia. *Batch Normalization*. URL: https://en.wikipedia.org/wiki/Batch_normalization.