

**2020/2021**

**Trabajo final**

**Descubrimiento del conocimiento Minería de datos**

**AUTORES:**

* **JULEN MERCHAN**
* **DAVID PERUJO**

**INDICE**

[**Introducción** 2](#_Toc39654670)

[**Dataset** 2](#_Toc39654671)

[**Preprocesamiento** 3](#_Toc39654672)

[Aumento de datos (Data Augmentation) 3](#_Toc39654673)

[Añadir ruido gaussiano 3](#_Toc39654674)

[Voltear verticalmente las imágenes: 3](#_Toc39654675)

[Reducción de las imágenes 3](#_Toc39654676)

[Normalizar 4](#_Toc39654677)

[Particionamiento de datos 4](#_Toc39654678)

[Extracción de características 4](#_Toc39654679)

[**Primera aproximación** 5](#_Toc39654680)

[Todas las características 6](#_Toc39654681)

[Regresión logística con parámetros por defecto 8](#_Toc39654682)

[Normalización 11](#_Toc39654683)

[Grid search con Regresión logística 14](#_Toc39654684)

[SVC 17](#_Toc39654685)

[Adaboost 19](#_Toc39654686)

[OneVsRest 23](#_Toc39654687)

[**Segunda aproximación** 25](#_Toc39654688)

[HOG 25](#_Toc39654689)

[Patrón binario local para clasificación de texturas (LBP) 34](#_Toc39654690)

[**Tercera aproximación** 39](#_Toc39654691)

[**Problemas** 44](#_Toc39654692)

[**Conclusiones** 45](#_Toc39654693)

[**Posibles mejoras** 47](#_Toc39654694)

# **Introducción**

El tema que vamos a tratar en este proyecto, es el análisis de imágenes de rayos X del tórax de diferentes personas, mediante técnicas de Machine Learning, para generar modelos de clasificación, capaces de detectar casos de COVID-19 y diferenciarlos de patologías de neumonía vírica.

Para conseguir este objetivo, analizaremos los datos mediante dos métodos y compararemos los resultados obtenidos con ambas técnicas. Previamente, realizaremos un preprocesamiento de los datos, que nos ayudará a mejorar los resultados durante la clasificación.

Al final de la memoria, reflejaremos las conclusiones obtenidas tras la realización del trabajo y sugeriremos posibles métodos para obtener mejoras en los resultados.

# **Dataset**

Necesitamos un conjunto de datos con el que poder entrenar los modelos y probar su rendimiento, para ello hemos utilizado un dataset de la página [kaggle](http://www.kaggle.com), en concreto el dataset “[COVID-19 Chest X-ray Database](https://www.kaggle.com/tawsifurrahman/covid19-radiography-database)”. Se trata de un conjunto de imágenes de rayos X del tórax de casos positivos de COVID-19, casos sin problemas respiratorios (normales) y casos de neumonía viral.

El dataset ha sido creado por un conjunto de investigadores de la Universidad de Qatar, Doha y de la universidad de Dhaka, Bangladesh, junto con sus colaboradores de Pakistán y Malasia, en colaboración con una serie de médicos.

La versión del dataset que hemos utilizado consta de:

1. 219 imágenes positivas para COVID-19.
2. 1341 imágenes normales.
3. 1345 imágenes de neumonía viral.

Todas las imágenes están en formato PNG y tienen un tamaño de 1024 \* 1024 píxeles. No obstante, como veremos más adelante para facilitar el tratamiento posterior de las imágenes para su clasificación, decidimos reducir el tamaño de éstas.

Como veremos más adelante en el documento, tuvimos que ampliar el conjunto de datos, más concretamente las imágenes de COVID-19 para intentar mejorar los resultados obtenidos. Dichas imágenes fueron sacadas de un repositorio de [GitHub](https://github.com/ieee8023/covid-chestxray-dataset/tree/master/images), en este repositorio se recogen 235 imágenes de COVID-19 así como de otras enfermedades respiratorias.

# **Preprocesamiento**

Antes de comenzar a construir los modelos para realizar la clasificación, debemos realizar un preprocesamiento de los datos, que nos ayudará a obtener mejores resultados en la futura clasificación.

## Aumento de datos (Data Augmentation)

Como en la mayoría de problemas de visión por computador, hemos creído necesario aumentar el conjunto de datos para realizar un mejor entrenamiento, que permita obtener modelos de una mayor calidad. Para ello, hemos aplicado 2 métodos que nos han permitido generar imágenes de forma artificial con las que aumentar el dataset:

### Añadir ruido gaussiano

Hemos añadido a las imágenes una pequeña cantidad de ruido gaussiano, lo que nos proporciona por cada imagen de entrenamiento una nueva. Al añadir imágenes con ruido, hacemos más robusto al clasificador frente a muestras futuras.

### Voltear verticalmente las imágenes:

Hemos considerado que como estamos tratando con imágenes de tórax, es igualmente válida una imagen obtenida al hacer un “flip”. Por ejemplo, un pulmón izquierdo afectado por COVID-19 es tan válido como la imagen del flip en la que se vería el otro pulmón igualmente afectado. De esta manera, el clasificador aprende que ambas situaciones son válidas.

Además de estas 2 técnicas como hemos comentado antes, también ampliamos el dataset utilizando imágenes de COVID-19 de otro repositorio.

## Reducción de las imágenes

Hemos decidido reducir el tamaño de las imágenes de 1024\*1024 píxeles a 256\*256 píxeles, ya que consideramos que tratar con tamaños de imágenes tan grandes supone una tarea más costosa y por tanto más lenta. Al reducir a este tamaño, el número de características a tratar se ve disminuido considerablemente sin perder, excesivamente en rendimiento.

## Normalizar

Como para entrenar un clasificador es mucho más rápido hacerlo con valores pequeños (entre 0 y 1), que usando valores muy grandes, decidimos normalizar todos los datos y se apreció una gran diferencia de tiempo.

## Particionamiento de datos

Hemos dividido los datos mediante la técnica de “hold-out”, siguiendo la típica regla de 60% de train, 20% de test y 20% de validation. Pero en nuestro caso, en cada conjunto generado, la proporción de ejemplos de cada clase no ha sido la misma. De hecho, la cantidad de imágenes para los casos de COVID-19, es bastante más baja que para los casos de neumonía vírica o para los casos sanos.

## Extracción de características

Como primera aproximación del problema, consideramos a todos los píxeles de cada imagen como posibles características a la hora de clasificar. Por lo que previamente, utilizando la técnica de “KBest selection”, se seleccionaron aquellas características que aportaban una mayor información para clasificar las imágenes, descartando al resto.

En la segunda aproximación, utilizamos técnicas de la librería “scikit-image” para extraer características de más alto nivel y comprobar si los resultados obtenidos eran o no mejores que tratando los pixeles directamente, como veremos posteriormente.

# **Primera aproximación**

Para la primera aproximación hemos considerado que los píxeles de la imagen eran directamente las características a utilizar para realizar la clasificación. Para ello, tras hacer el preprocesamiento de las imágenes, hemos pasado cada uno de los conjuntos a un formato de dos dimensiones, donde cada fila representa una muestra del problema y cada columna corresponde a una característica o pixel de la muestra.

Después, hemos generado un objeto del tipo SelectKBest, pasándole como parámetro el filtro uni-variable ANOVA (f\_classif) para determinar la calidad de las variables individuales del problema. Puesto que, generalmente con este filtro se obtienen muy buenos resultados, hemos aplicado la técnica de SelectKBest para un rango de valores desde 1 hasta el número total de píxeles en el problema. De esta manera, hemos podido obtener el valor de score (puntuación) para las distintas cantidades de variables (pixeles en nuestro problema), lo que nos ha ayudado a seleccionar solo aquellas variables que son necesarias para la clasificación y descartar aquellas que no son necesarias.

Para la selección de características hemos generado un gráfico de barras utilizando la librería de pandas como podemos ver en la Figura 1. En esta gráfica podemos apreciar que el mayor porcentaje de los píxeles está alojado en el extremo izquierdo de ésta, donde los valores de score son los más bajos. Como estos píxeles aportan muy poca información para realizar la clasificación de las imágenes, los consideraremos prescindibles para nuestro problema. Esto, a priori, debería mejorar el rendimiento de la clasificación y el coste computacional, lo que provocará descensos significativos en el tiempo de ejecución como veremos más adelante.

Consideramos que los píxeles asociados a zonas de un bajo valor de score, muy probablemente corresponden a zonas de las imágenes como, por ejemplo, píxeles del fondo negro que dan contexto a la imagen, pero no aportan demasiada información por sí mismos. Por lo que, en definitiva, no aportan mucho valor a la hora de clasificar una imagen como neumonía vírica, COVID-19 o pulmones sanos. No obstante, debemos tener en cuenta que seleccionar un valor de k excesivamente alto, puede provocar que se eliminen demasiadas características y como consecuencia el modelo generado sea demasiado simple y su rendimiento disminuya en exceso.

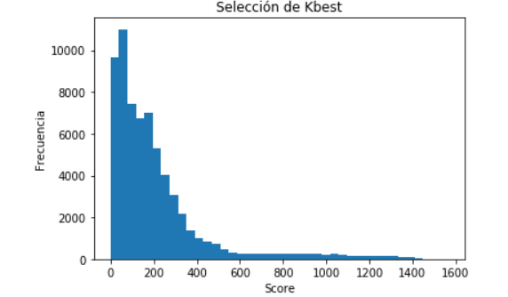
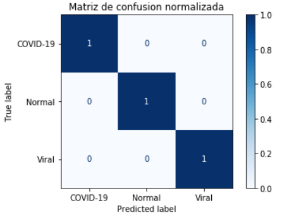
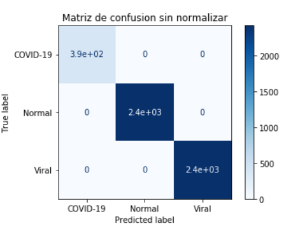


Figura 1

## Todas las características

Los resultados previos que hemos obtenido con todas las características del problema se muestran en las figuras: Figura 2, Figura 3 y Figura 4.

Para train:



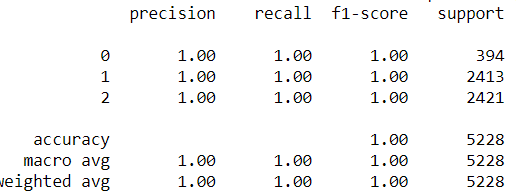
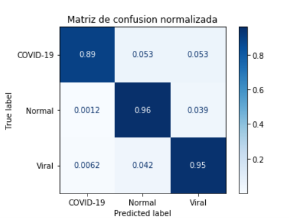
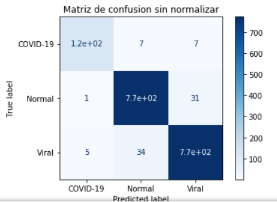


Figura 2

Para val:  


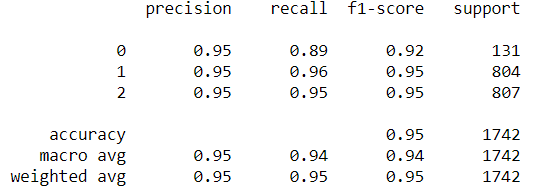
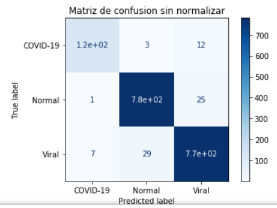
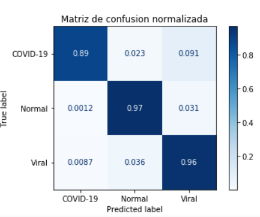


Figura 3

Para test:



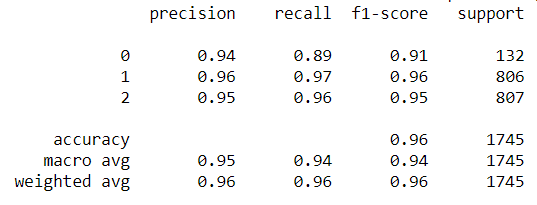


Figura 4

## Regresión logística con parámetros por defecto

Hemos seleccionado 5 valores de score (k): 0 (El valor 0 representa el uso de todas las características de la imagen), 100, 200, 300 y 450. Entrenaremos una regresión logística con los parámetros por defecto y así obtener los resultados para cada valor de k. Con el objetivo de determinar a partir de qué valor de score debemos considerar a un conjunto de variable (pixeles) de suficiente calidad para nuestro caso. Los resultados obtenidos para cada valor de K son los que podemos apreciar en la Tabla 1.

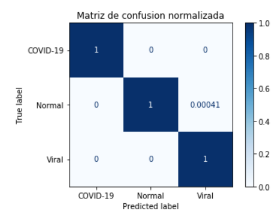
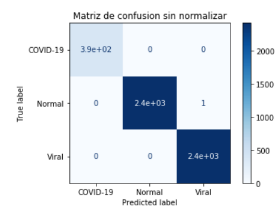
Tabla 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Score | Accuracy-Train | Accuracy-Val | Accuracy-Test | Número de características |
| 0 | 100% | 94.89092996555684% | 95.87392550143267% | 65536 |
| 100 | 100% | 94.31687715269805% | 94.89971346704871% | 40473 |
| 200 | 100% | 92.8817451205511% | 93.46704871060172% | 23031 |
| 300 | 100% | 92.42250287026407% | 91.34670487106017% | 12230 |
| 450 | 100% | 86.33754305396096% | 86.36103151862464% | 6335 |

Comprobamos que, para un valor de score de 200, los resultados tanto en validación y test son peores en accuracy y en recall que para el caso inicial (0, utilizando todas las variables). Sin embargo, la perdida de aproximadamente un 2.4% de accuracy en test y un 2% en validación, nos resulta asumible, teniendo en cuenta que la velocidad de ejecución se reduce a más de la mitad. Esta reducción, evidentemente, es debida a que pasamos de tener 65535 variables a tan solo 23031.

No obstante, para todos los casos vemos que existe una diferencia en accuracy entre validation y test, que va desde aproximadamente un 5% hasta un 14%, lo que nos sugiere que tenemos un problema de varianza. Para arrojar más luz sobre el problema, decidimos generar la matriz de confusión para comprobar cómo se estaban clasificando los ejemplos del dataset tanto en entrenamiento como en validación y en test. Como se puede observar para un score de 200, en las figuras: Figura 5, Figura 6 y Figura 7.

Para train:



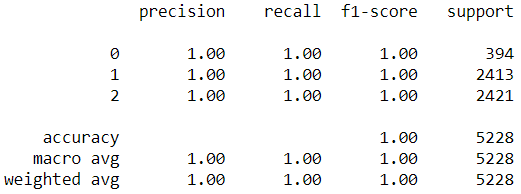
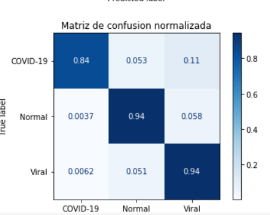
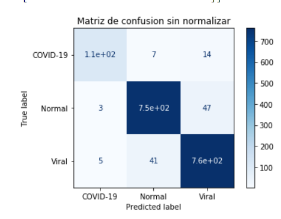


Figura 5

Para val:



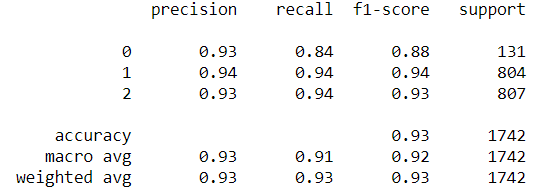
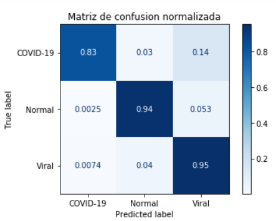
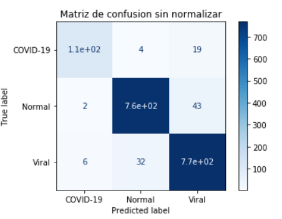


Figura 6

Para test:



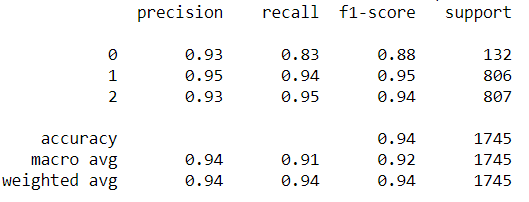


Figura 7

Como podemos apreciar en las figuras anteriores (Figura 5, Figura 6 y Figura 7), durante la fase de entrenamiento, prácticamente se clasifican correctamente todos los ejemplos, como ya intuíamos, viendo el porcentaje de accuracy de train (100%). Sin embargo, observamos que, tanto en validación como en test, donde más falla el clasificador, es a la hora de diferenciar casos de COVD-19 de casos de neumonía viral. Como vemos el porcentaje de personas con COVID-19 que han sido detectadas como tal, es de un 83% en test y de un 84% en validación, es decir que el recall es de 0.83 y 0.84 respectivamente.

No obstante, cabe destacar que a pesar de estos porcentajes vemos como un 0.14 (en test) y un 0.11 (en validación) de los casos de COVID son detectados como neumonía viral. Por tanto, nuestro clasificador no lo está haciendo tan mal, ya que del 17% (en test) y del 16% (en validación) de los casos que no son detectados como COVID cuando realmente los son, al menos, los detecta como una neumonía. Apenas dejamos un 5% y un 3% respectivamente de pacientes enfermos sin clasificarlos como COVID o neumonía viral.

Nuestro problema de varianza reside en que estamos clasificando mal varios ejemplos de la clase COVID-19 como refleja el recall para esta clase. Por tanto, lo que intentaremos a lo largo del proyecto será priorizar la detección de casos de neumonía viral y de COVID, es decir, priorizaremos tener un recall alto para evitar pasar por alto casos de neumonía, lo que debería reducir la precisión puesto que habrá ocasiones en las que clasificaremos personas sanas como casos de posible neumonía (viral o COVID).

Todo esto puede ser debido, a que los píxeles por sí mismos aportan poca información a la hora de clasificar las imágenes. De manera que en la fase de train, al cabo de varias iteraciones, el modelo acaba sobre aprendiendo los valores de los ejemplos del conjunto y cuando debe generalizar para nuevos ejemplos (validación y test), lo hace relativamente mal.

Algo a destacar, es que durante todas las pruebas apreciamos como los tiempos de ejecución eran excesivamente elevados. Es por esto, que decidimos que una manera de reducir los tiempos de ejecución, sería normalizando los datos para dejarlos en un rango de valores de [0,1]. Esto nos permitiría tratar con datos de menor tamaño durante las pruebas y por tanto los tiempos de ejecución se reducirían, sin perder demasiado en rendimiento, como muestran las tablas: Tabla 2 y Tabla 3. Además, para el caso de score 200, ganamos algo de accuracy en test, pero perdemos en validación.

## Normalización

Tabla 2

|  |  |  |
| --- | --- | --- |
|  | Selección de Score | Todas las características |
| Datos sin normalizar | 793.404999256134(s) | 646.3641974925995(s) |
| Datos normalizados | 409.6446096897125(s) | 588.0058348178864(s) |

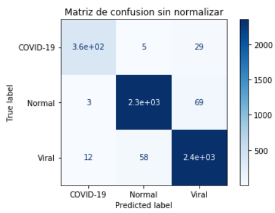
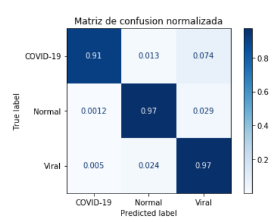
Tabla 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Score | Accuracy-Train | Accuracy-Val | Accuracy-Test | Tiempo |
| 0 | 99.86610558530987% | 95.17795637198623% | 95.5300859598854% | 588.0058348178864(s) |
| 100 | 100% | 94.48909299655568% | 94.89971346704871% | 410.7783679962158(s) |
| 200 | 100% | 93.22617680826636% | 93.98280802292264% | 224.06287145614624(s) |
| 300 | 99.73221117061975% | 92.25028702640643% | 91.74785100286533% | 148.37313866615295 |
| 450 | 97.05432287681714% | 90.01148105625718% | 89.68481375358166% | 62.62581205368042(s) |

Para solucionar el problema de varianza que mencionábamos anteriormente, hemos decidido utilizar algún tipo de regularización, aplicando la técnica de “grid search”. Con esto, buscamos la combinación de parámetros más adecuada para la regresión logística que nos permita resolver este problema. Además, nos dimos cuenta que, por defecto, la regresión logística utiliza como algoritmo de optimización “liblinear”, que funciona en problemas de one-vs-rest, pero para problemas multiclase se suelen utilizar otros algoritmos de optimización como: “newton-cg”, “sag”, “saga” y “lbfgs”. Por tanto, los parámetros que decidimos probar con “grid search” son los algoritmos de optimización que acabamos de mencionar, las funciones de regularización “l1”, “l2” y “elasticnet” y por supuesto varios valores del hiper parámetro C (que es el inverso a la regularización).

Observamos que el “grid search” nos devuelve como mejor configuración la función de optimización “saga”, como función de regularización “l2” y como valor de C el valor 100. Sin embargo, cuando entrenamos el modelo con estos valores, vemos que los porcentajes de acierto tanto en train, test y validación disminuyen, por lo que pasamos de tener un problema de varianza a tener un problema de varianza y Bias. Aun así, decidimos mostrar las matrices de confusión, pues nos planteamos la posibilidad de que, aunque el porcentaje de acierto resulte peor, quizás el recall en la clase COVID y en la clase viral haya aumentado y las precisiones hayan disminuido con lo que el accuracy se vería afectado.

Para train:



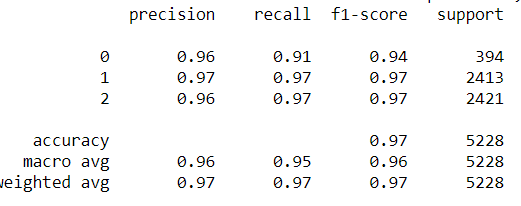
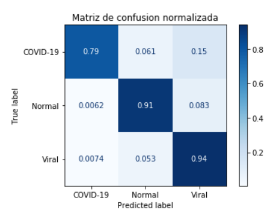
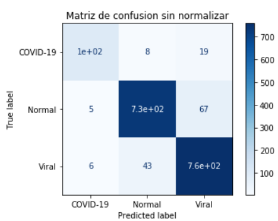


Figura 8

Para val.



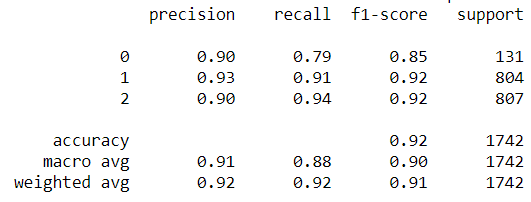
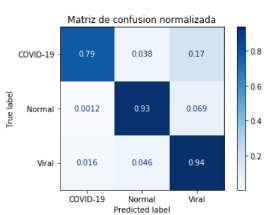
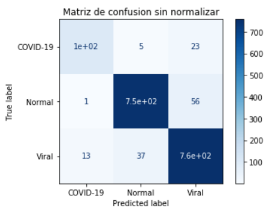


Figura 9

Para test:



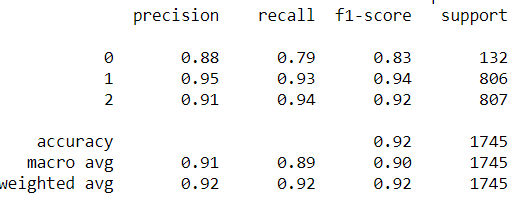


Figura 10

Como vemos, nuestra suposición era incorrecta y tanto el recall para la clase COVID-19 y la clase viral como para los valores de precisión han disminuido. Por este motivo, obtenemos un valor más bajo de accuracy tanto en test como en train.

Concluimos que, esto puede ser debido a que, según la documentación de sklearn para la regresión logística las funciones “sag” y “saga” funcionan bien para problemas con conjuntos de datos grandes, pero en nuestro caso el conjunto de datos es más bien pequeño y en este tipo de situaciones “liblinear” tiende a funcionar mejor. También esto puede ser debido a que un enfoque de one-vs-rest funciona mejor para nuestro ejemplo que tratarlo como un problema multiclase.

## Grid search con Regresión logística

Por tanto, nos centraremos en buscar una buena combinación para liblinear, ya que los mejores resultados que ha encontrado grid search están asociados a la función de optimización “saga” y son más bajos respecto a los que teníamos con “liblinear”, así que no tiene sentido probar para el resto de funciones de optimización. Los resultados obtenidos se muestran en la tabla.

Tabla 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F. Optimización | F. Regularización | C | Accuracy train | Accuracy val | Accuracy test |
| saga | l2 | 100 | 96.60% | 91.27% | 92.15% |
| liblinear | l1 | 2 | 99.60% | 93.29% | 94.04% |
| liblinear | l1 | 5 | 99.72% | 93.74% | 94.27% |
| liblinear | l1 | 10 | 99.81% | 93.51% | 93.93% |
| liblinear | l1 | 100 | 100% | 93.91% | 93.81% |

Grid search nos decía que el mejor resultado para la regresión logística era:

1. Función de optimización: “liblinear”
2. C = 100
3. Función regularización: “l1”

Sin embargo, tras hacer pruebas observamos que el mejor valor de C se obtiene con C=5, lo que puede indicar que Grid search se ha guiado por el valor de accuracy en train y por eso nos ha devuelto como mejor valor C=100, cuando su porcentaje de acierto en test es más bajo que en el resto. Comprobaremos los resultados con las matrices de confusión:

Para C=5 en validación:

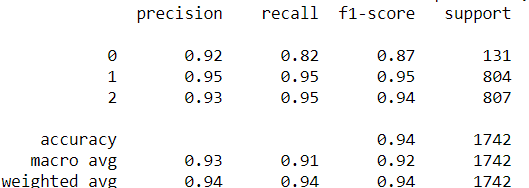


Figura 11

Para c=5 en test:

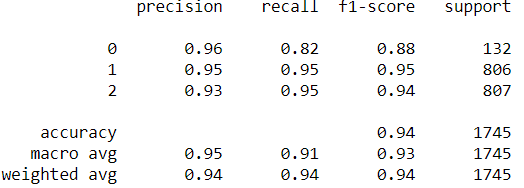


Figura 12

Para C=100 en validación:

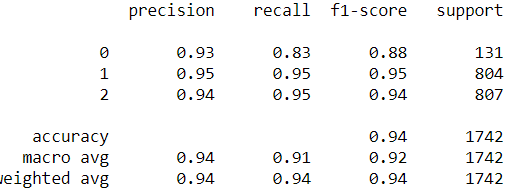


Figura 13

Para C=100 en test:

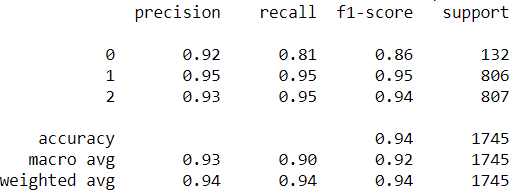


Figura 14

Como vemos, el recall y la precisión en test resultan superiores para C=5, que para C=100 para la clase de COVID-19. No obstante, nos damos cuenta de que los resultados son algo inferiores en cuanto a recall sin aplicar regularización (pasamos de 0.83 a 0.82), por lo que el aumento en accuracy es debido al aumento de precisión para la clase COVID-19 (pasamos de 0.93 a 0.96), es decir estamos clasificando como COVID un menor número de ejemplos que no son de esta clase.

Se puede observar, aplicando regularización con la función penalty l1 y un valor de C mayor a 1 y menor o igual a 5, conseguimos que los resultados obtenidos mejoren en train, validación y test (a nivel de accuracy) para un score 200.

Decidimos probar otra solución, añadiendo más ejemplos de COVID a nuestro dataset con el fin de mejorar la predicción aumentando el recall. Para ello, descargamos un conjunto de imágenes de un repositorio de [GitHub](https://github.com/ieee8023/covid-chestxray-dataset/tree/master/images) y les aplicamos el mismo tratamiento que a las imágenes iniciales.

Tabla 5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Accuracy-Train | Accuracy-Val | Accuracy-Test | Recall COVID (test) | Recall neumonía viral (test) | Precisión COVID (test) | Precisión neumonía viral  (test) |
| Dataset ampliado (sin regularización) | 100% | 94.69% | 94.11% | 0.92 | 0.93 | 0.93 | 0.94 |
| Dataset ampliado (con regularización) | 100% | 94.75% | 94.16% | 0.91 | 0.92 | 0.94 | 0.95 |

Observamos que al ampliar el número de imágenes de la clase COVID-19 solo nos lleva a un menor valor de accuracy en el caso de la regularización, mientras que en el caso sin regularización seguimos teniendo un aumento del porcentaje de acierto respecto del caso original. Además, se produce una mejora en cuanto a recall en el caso sin regularización para la clase COVID, que es justo lo que estábamos buscando, aunque perdemos en la clase neumonía viral, pasando de un 0.95 a un 0.93, de forma similar ocurre con el caso con regularización.

De todo esto, concluimos que tanto la regularización como añadir más ejemplos producen un mayor porcentaje de acierto. No obstante, donde verdaderamente vemos una mejora a nivel de recall es añadiendo más ejemplos de la clase de COVID-19, mientras que con la regularización lo que observamos es un incremento en precisión, lo que nos lleva a valores mayores de accuracy como hemos visto anteriormente. Por otro lado, hemos demostrado que la selección de características nos ha permitido reducir los tiempos de ejecución prácticamente a la mitad, perdiendo muy poco porcentaje de accuracy y de recall.

Damos por concluida la búsqueda de posibles soluciones para este clasificador y decidimos probar para esta misma aproximación del problema otro tipo de clasificadores, en este caso SVMs que deberían dar a priori un mejor resultado debido a la naturaleza de las mismas.

## SVC

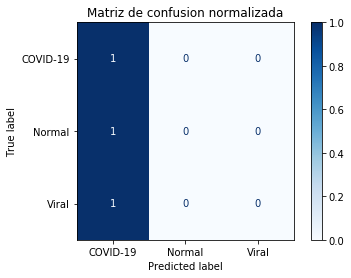
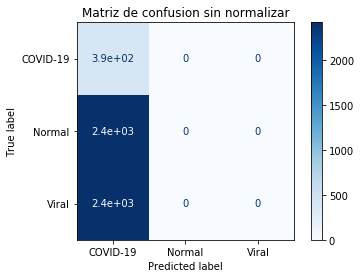
Los resultados obtenidos para distintos scores (k), usando SVC (Support Vector Classification) son los que aparecen en Tabla 6:

Tabla 6

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| K | Best\_parameters | Best\_score | accuracy Train | accuracy Test | accuracy Val |
| 0 | {'C': 10, 'kernel': 'rbf'} | 0.9665260230 | 97.1308339 | 95.70200573 | 94.37428243 |
| 100 | {'C': 10, 'kernel': 'rbf'} | 0.9634656517 | 96.7865340 | 95.24355300 | 94.48909299 |
| 200 | {'C': 10, 'kernel': 'rbf'} | 0.9537105583 | 95.0841622 | 93.92550143 | 93.16877152 |

Veremos las matrices de confusión para un score de 200 en las figuras (Figura 15, Figura 16, Figura 17).

Para train:



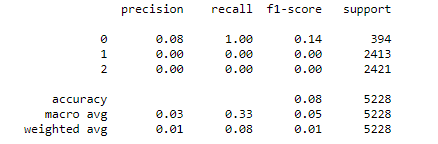
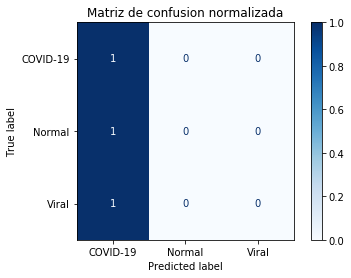
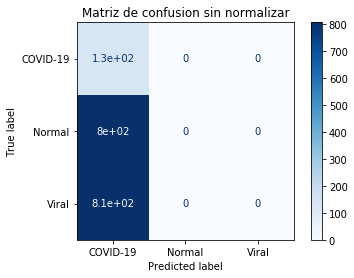


Figura 15

Para val:



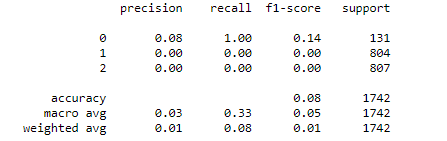
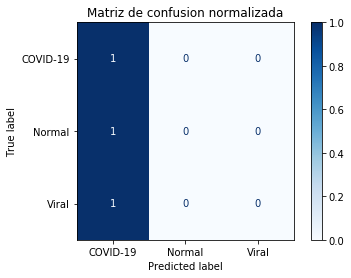
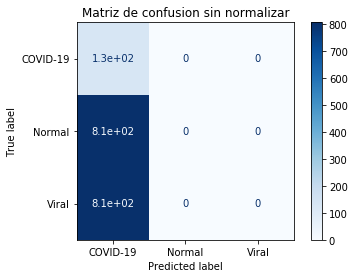


Figura 16

Para test



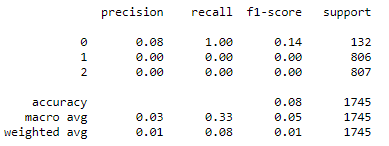


Figura 17

Observamos que el grid search nos devuelve como mejor valor de C el valor 1 y mejor kernel el rbf. Además, cuando entrenamos el modelo con estos valores vemos que los porcentajes de acierto siguen siendo malos.

Puesto que la regresión logística no ha dado muy buenos resultados, como última opción decidimos recurrir a las SVMs. Hemos elegido Adaboost ya que su finalidad es encontrar un clasificador fuerte, a partir de otros débiles y generalmente suele dar muy buenos resultados.

Adaboost crea un clasificador fuerte mediante la combinación lineal de clasificadores débiles. Lo logra entrenando de forma iterativa una serie de clasificadores débiles, de modo que el nuevo clasificador se enfoque en los datos que fueron erróneamente clasificados por su predecesor. De esta forma el algoritmo se adapta y obtiene mejores resultados, es decir, los clasificadores comenten errores diferentes (diversidad) y se corrigen entre ellos, lo que hace que a priori el porcentaje de acierto sea mucho mayor.

## Adaboost

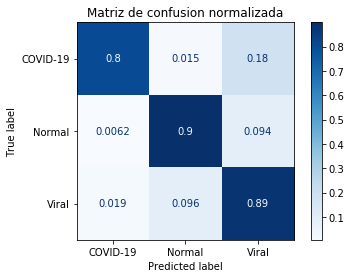
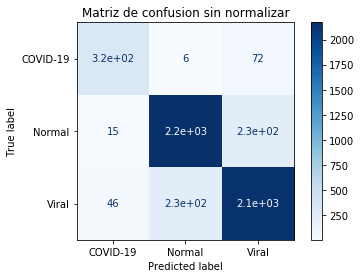
Los resultados obtenidos para distintos scores (k), usando Adaboost con dos estimadores base distintos (Logistic Regression, DecisionTreeClassifier), son los que aparecen en Tabla 7:

Tabla 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| K | base\_estimator | n\_estimators | accuracy Train | accuracy Test | accuracy Val |
| 0 | Logistic Regression | 4 | 91.9854628921 | 91.40401146 | 90.5281285 |
| DecisionTreeClassifier | 100 | 82.6893649579 | 80.0 | 80.1377726 |
| 100 | Logistic Regression | 4 | 90.6273909716 | 91.06017191 | 90.0688863 |
| DecisionTreeClassifier | 100 | 84.2195868400 | 81.08882521 | 81.0562571 |
| 200 | Logistic Regression | 4 | 89.116296863 | 89.79942693 | 88.1171067 |
| DecisionTreeClassifier | 100 | 78.997704667 | 75.98853868 | 76.1768082 |

Observaremos las matrices de confusión para un score de 200 con Logistic Regression en las figuras (Figura 18, Figura 19, Figura 20).

Para train:



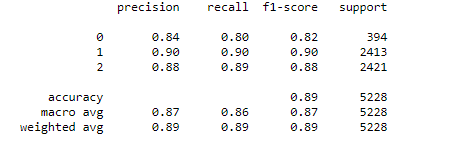
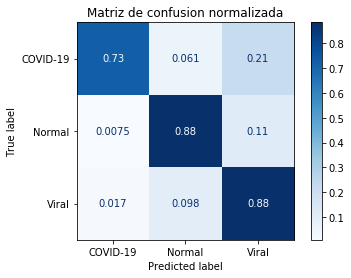
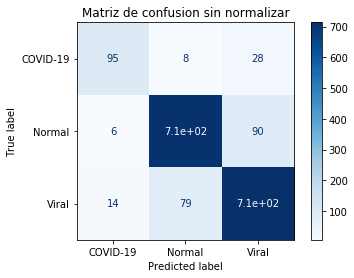


Figura 18

Para val:



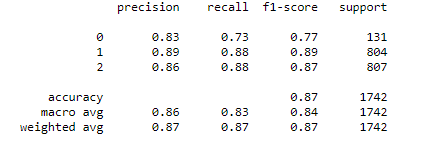
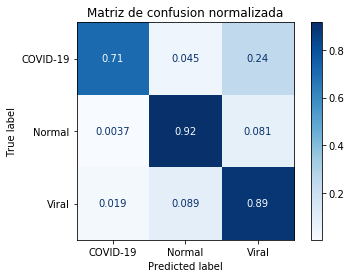
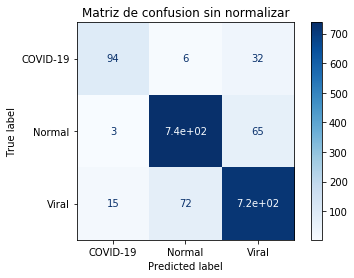


Figura 19

Para test:



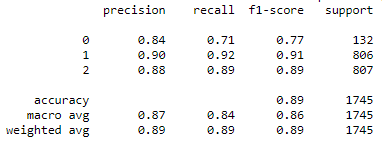
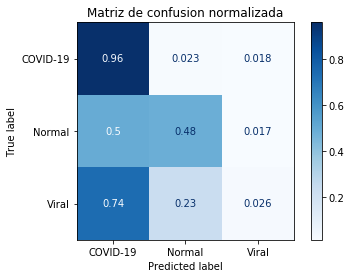
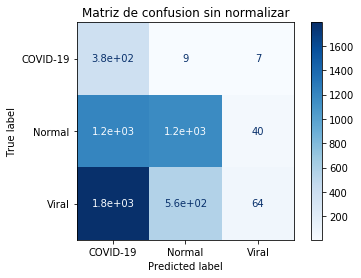


Figura 20

Como se puede ver en los resultados obtenidos con regresión logística son buenos, pero queremos seguir mejorándolos.

Observaremos las matrices de confusión para un score de 200 con DecisionTreeClassifier en las figuras (Figura 21, Figura 22, Figura 23).

Para train:



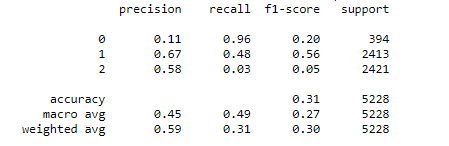
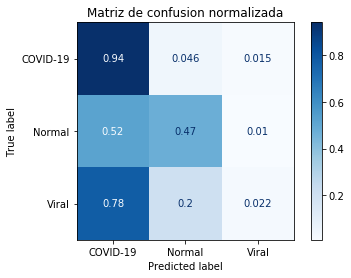
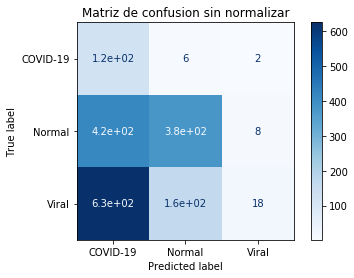


Figura 21

Para val:



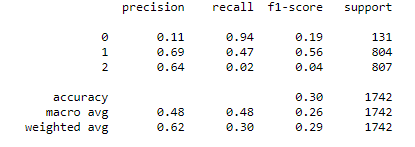
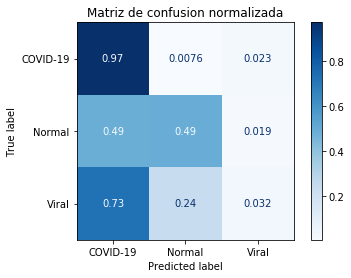
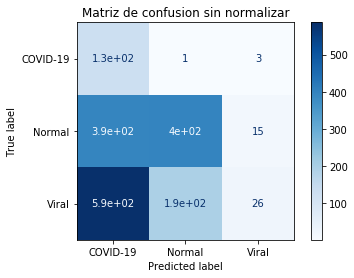


Figura 22

Para test:



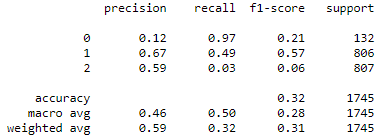


Figura 23

Como hemos observado que los resultados con Arboles de decisión no son buenos, por ello, hemos decidido utilizar un OVO ya que nuestro problema puede funcionar mejor en situaciones de one vs rest.

## OneVsRest

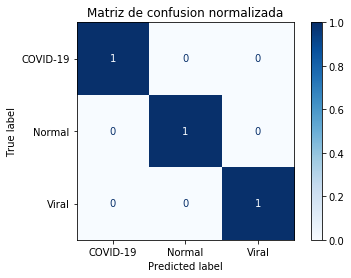
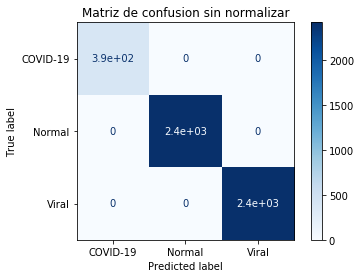
Los resultados obtenidos para distintos scores (k), usando el OneVsRest con Regresión logística y sin regularización, son los que aparecen en la Tabla 8.

Tabla 8

|  |  |  |  |
| --- | --- | --- | --- |
| K | accuracy Train | accuracy Test | accuracy Val |
| 0 | 100.0% | 96.10315186246419% | 95.46498277841562% |
| 100 | 100.0% | 95.35816618911174% | 94.77611940298507% |
| 200 | 100.0% | 94.21203438395416% | 93.85763490241102% |

Observaremos las matrices de confusión para un score de 200 con OneVsRest en las figuras (Figura 24, Figura 25, Figura 26).

Para train:



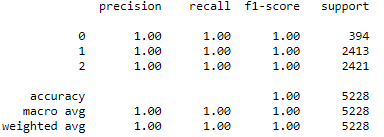
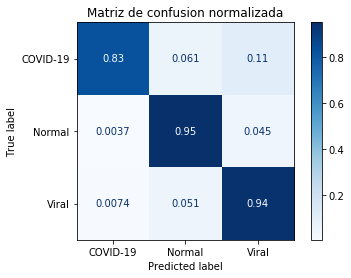
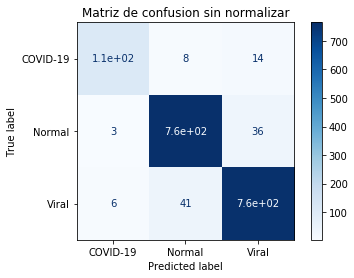


Figura 24

Para val:



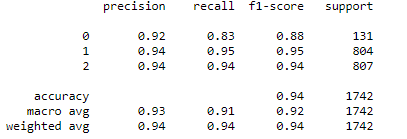
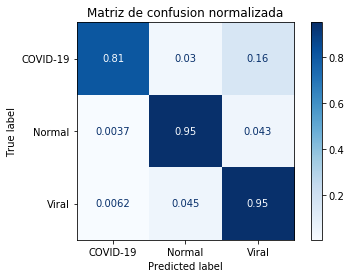
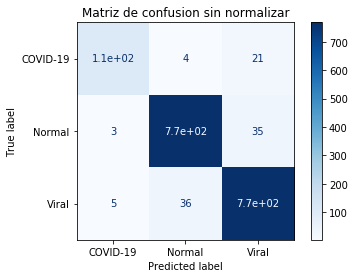


Figura 25

Para test:



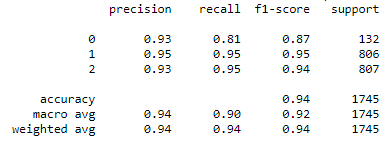


Figura 26

Viendo los resultados obtenidos, se percibe una clara mejoría usando OneVsRest que otros clasificadores.

De todos modos, seguimos queriendo obtener una mejora y estudiaremos mas casos.

# **Segunda aproximación**

Para esta segunda aproximación, en vista de los resultados obtenidos en las primeras pruebas, hemos decidimos que una mejor solución es extraer características de las imágenes mediante técnicas como HOG o LBP en vez de tratar de forma directa los píxeles de la imagen. Lo que en principio debería dar mejores resultados ya que estaríamos tratando con características de un mayor nivel que en la primera aproximación.

Para estas pruebas utilizaremos la librería Scikit-image que implementa una gran variedad de funciones para la extracción de características de las imágenes.

## HOG

Hog es una técnica que obtiene características de una imagen teniendo en cuenta la orientación del gradiente en partes localizadas de la imagen. La idea es que la forma de un objeto dentro de una imagen se puede especificar mediante la distribución de gradientes de intensidad.

La imagen se divide en pequeñas regiones conectadas llamadas celdas. Para los píxeles dentro de cada celda, se compila un histograma de direcciones de gradiente. El descriptor es la concatenación de estos histogramas.

Para sacar las características de HOG de nuestros conjuntos de train, validación y test construimos una función que nos permite extraer por cada imagen de cada conjunto su descriptor de HOG, es decir la concatenación de histogramas de gradientes de cada una de las imágenes. Cada descriptor obtenido es de dimensión (1,2048) es decir, cada ejemplo pasa a tener 2048 características, por lo tanto, cada conjunto pasa a ser de dimensión (N,2048) donde N es el número de ejemplos iniciales que tenía asociado dicho conjunto.

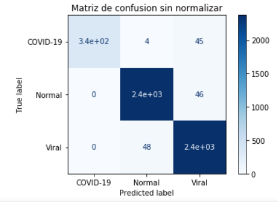
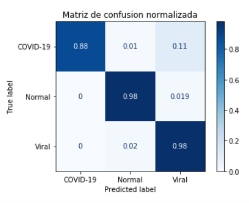
Para las pruebas seguiremos un esquema similar al de la fase anterior. Primero probaremos a realizar una clasificación con todas las características obtenidas utilizando HOG y comprobaremos los resultados obtenidos. Para ello probamos con una regresión logística (Tabla 9) utilizando los parámetros por defecto en el constructor.

Tabla 9

|  |  |  |
| --- | --- | --- |
| Accuracy-Train | Accuracy-Val | Accuracy-Test |
| 97.26472838561591% | 92.0780711825488% | 92.03438395415473% |

Como observamos los porcentajes de acierto son bastante buenos, apenas tenemos diferencia entre el acierto en train y en validación y test, no obstante, los resultados aún pueden ser mejorables. Imprimiendo las matrices de confusión (Figura 27, Figura 28, Figura 29) vemos que el recall en los casos de COVID-19 sigue siendo bastante bajo tanto en validación como en test a pesar de que el accuracy sea elevado, pese a que por ejemplo en el caso de test solo un 66% de los casos de COVID-19 son detectados como tal, un 25% se detecta como neumonía vírica, por lo que, aunque no está detectando que los pacientes tienen COVID si detecta que tienen neumonía. Esto resulta bastante positivo pues al menos dejamos un menor número de personas afectadas por algún tipo de neumonía sin clasificar como personas enfermas de neumonía, a nivel clínico los tratamientos pueden ser distintos, pero al menos la presencia de neumonía en los pulmones es detectada.

Para train:



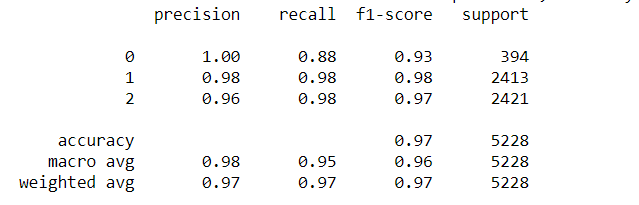
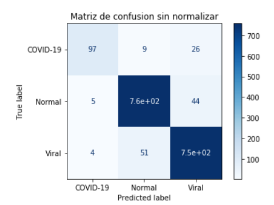
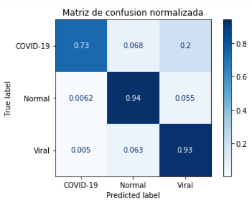


Figura 27

Para test:  


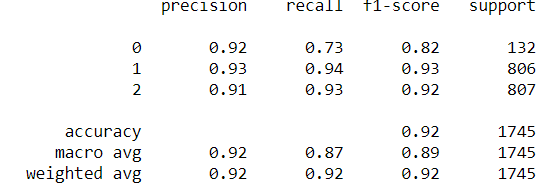
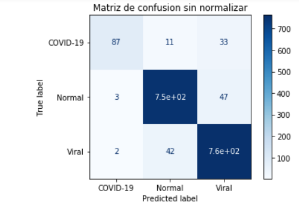
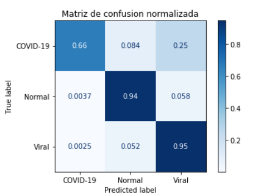


Figura 28

Para val:



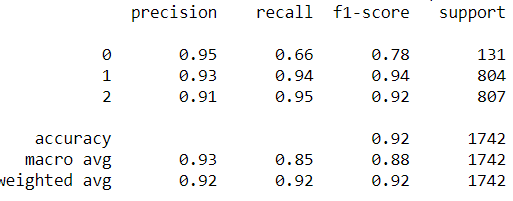
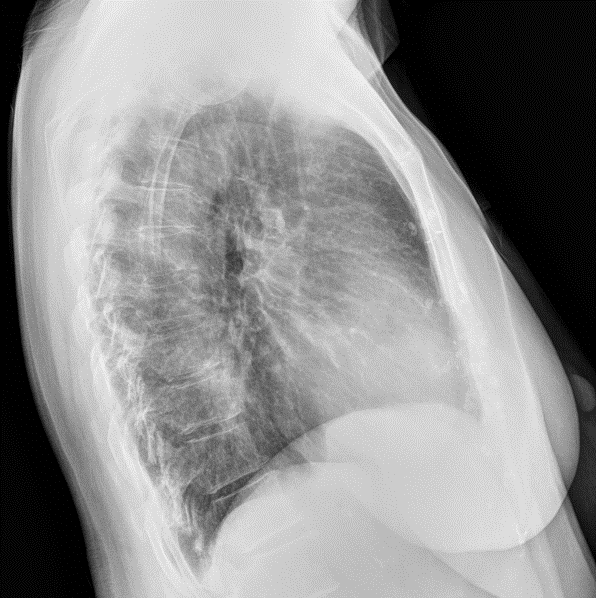
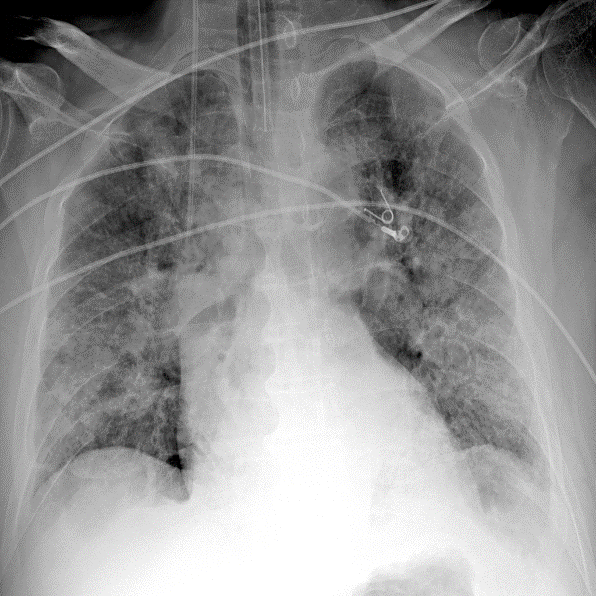


Figura 29

Algo a destacar es que los resultados que se han obtenido en esta prueba ya muestran una clara evidencia de que esta técnica funciona peor que en la primera aproximación, pues tanto los valores de accuracy como los valores de recall para la clase COVID-19 son bastante más bajos que los obtenidos en el primer caso.

Esto puede estar debido entre otras razones a que generalmente HOG se utiliza en visión por computador como una técnica para la extracción de características para la detección de objetos teniendo en cuenta la forma de los mismos, sin embargo, para nuestro caso, tenemos que la mayor parte de las radiografías de rayos X están tomadas del tórax en la parte frontal mientras que otras están tomadas de forma lateral (Figura 30 y Figura 31), esto puede hacer que las características extraídas con HOG para dos imágenes dentro de una clase no sean las mismas. Además, otro gran problema es que una mancha de neumonía en un pulmón resulta diferente en cada paciente por lo tanto resulta complicado establecer una forma común para todas las manchas de neumonía y también encontramos que manchas de neumonía viral resultan muy parecidas a las de casos de COVID-19 lo que puede confundir al clasificador como veíamos en los valores de recall para esta clase.

No obstante, intentaremos mejorar al igual que en la primera aproximación los valores de recall para las clases de COVID-19 y neumonía viral.

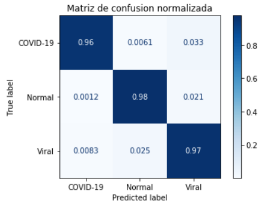
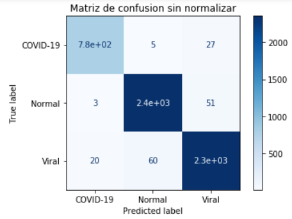


Figura

Figura

Como en el primer caso probamos a aumentar el número de ejemplos de la clase COVID-19 para ver si logramos aumentar el porcentaje de recall ya que en este caso resulta bastante bajo, tras las pruebas vemos que aumenta considerablemente, pasamos de un 0.73 a un 0.89 en test y de un 0.66 a un 0.89 en val, manteniendo además el valor de recall en la clase viral como muestran las matrices de consfusión (Figura 30, Figura 31, Figura 32).

Para train:



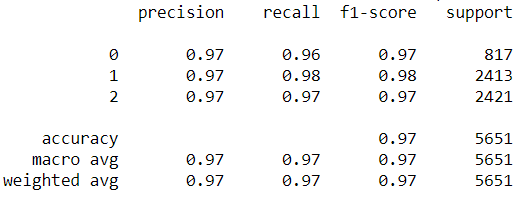
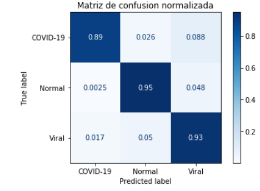
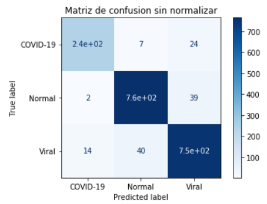


Figura 32

Para test:



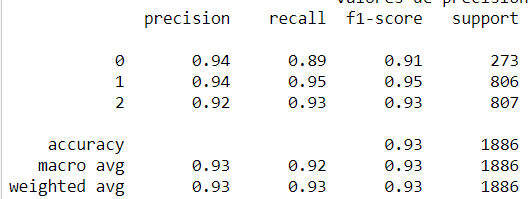
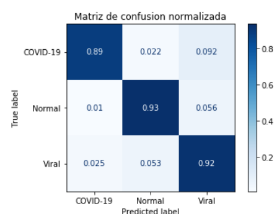
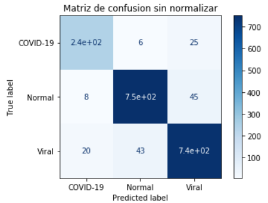


Figura 33

Para val:



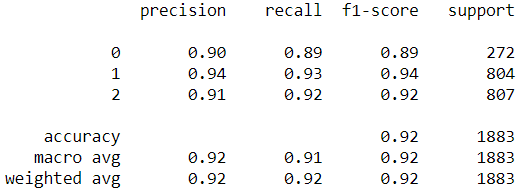


Figura 34

Probaremos a aplicar regularización al caso anterior para ver si los resultados tienden a mejorar o no, para ello aplicaremos la regularización que vimos en la primera aproximación (penalty ==” l1”, C=5, solver ==” liblinear”)

Tabla 10

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Acc Train | Acc Val | Acc Test | Recall COVID (test) | Recall Viral (test) | Precisión COVID (test) | Precisión viral (test) |
| Sin regularización | 97.06% | 92.19% | 93.32% | 0.89 | 0.93 | 0.94 | 0.92 |
| Con regularización | 99.36% | 92.35% | 92.52% | 0.87 | 0.93 | 0.94 | 0.91 |

Como vemos en la Tabla 10, al aplicar regularización, ocurre algo similar a lo que pasaba en la primera aproximación, obtenemos peores resultados en cuanto a recall para la clase COVID y además estamos perdiendo en accuracy para test, por lo que descartamos el uso de regularización para mejorar los resultados obtenidos.

Valoramos la opción de seleccionar solo algunas características como hicimos en la primera aproximación para ver si eliminando algunas características podemos mejorar los resultados que hemos obtenido para este caso.

Imprimimos el plot de score tras aplicar selecKBest y obtenemos el siguiente gráfico (Figura 33):

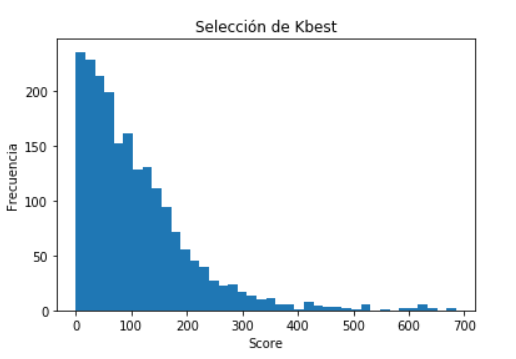


Figura 35

Por tanto, tras examinar la gráfica de barras decidimos probar aquellos conjuntos de variables para los que el score sea superior a 25, 50, 75, 100, 150, 200 y obtenemos los siguientes resultados (Tabla 11):

Tabla 11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Score | Accuracy-Train | Accuracy-Val | Accuracy-Test | Número de características |
| 0 | 97.06% | 92.19% | 93.32% | 2048 |
| 25 | 96.57% | 92.35% | 92.52% | 1704 |
| 50 | 96.02% | 92.51% | 92.26% | 1382 |
| 75 | 95.10% | 92.09% | 91.57% | 1118 |
| 100 | 94.50% | 91.02% | 91.30% | 886 |
| 150 | 92.16% | 90.28% | 89.98% | 518 |
| 200 | 89.10% | 87.63% | 87.01% | 281 |

Viendo los resultados obtenidos en cuanto a accuracy, observamos que los resultados tienden a ser peores cuando eliminamos características. Decidimos comprobar los valores de recall en test y validación para COVID y neumonía viral para cada uno de los casos anteriores para comprobar si al menos estos mejoran o no (Tabla 12).

Tabla 12

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Score | Recall COVID  (val) | Recall viral  (val) | Recall COVID  (test) | Recall viral  (test) | Tiempo de ejecución |
| 0 | 0.89 | 0.92 | 0.89 | 0.93 | 9.868009567260742(s) |
| 25 | 0.86 | 0.93 | 0.87 | 0.93 | 6.4882495403289795(s) |
| 50 | 0.88 | 0.93 | 0.87 | 0.93 | 5.273128986358643(s) |
| 75 | 0.88 | 0.92 | 0.86 | 0.92 | 4.617154121398926(s) |
| 100 | 0.87 | 0.92 | 0.85 | 0.92 | 3.200742483139038(s) |
| 150 | 0.84 | 0.91 | 0.82 | 0.92 | 1.764519453048706(s) |
| 200 | 0.81 | 0.89 | 0.79 | 0.89 | 1.0134670734405518(s) |

Como vemos en la Tabla 12, los valores de recall en ambas clases también tienden a disminuir conforme utilizamos un menor número de características, especialmente en el caso de COVID-19 en test, con lo que obtenemos peores resultados. En esta aproximación, además, a diferencia de la anterior no tiene mucho sentido reducir el número de características para ganar en tiempo de ejecución, pues como se puede ver en la tabla los tiempos de ejecución son realmente bajos, ya que con HOG partimos inicialmente teniendo 2048 características. Por tanto, concluimos que los mejores resultados se obtienen utilizando todas las características y que además no resulta necesario eliminar características pues los tiempos de ejecución para esta aproximación resultan bastante adecuados.

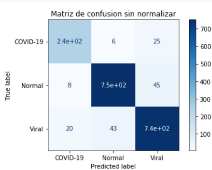
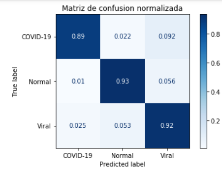
Por último, decidimos probar a utilizar OVR con HOG y con un número mayor de ejemplos de la clase COVID-19, pues en la primera aproximación mejoraba aún más los resultados obtenidos con regresión logística. Utilizando todas las características estos son los resultados que hemos obtenido en la Tabla 13.

Tabla 13

|  |  |  |  |
| --- | --- | --- | --- |
| Acc Train | Acc Val | Acc Test | Tiempo de ejecución |
| 97.06% | 92.19% | 93.32% | 11.183987140655518(s) |

En lo que respecta a las matrices de confusión (Figura 34, Figura 35, Figura 36), obtenemos:

Para train:

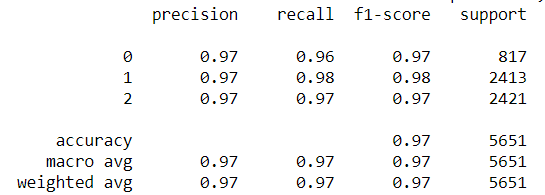
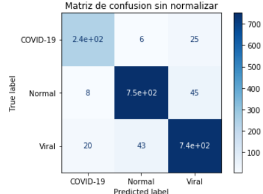
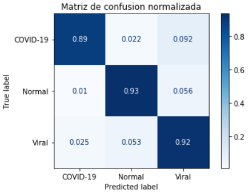


Figura 36

Para validación:



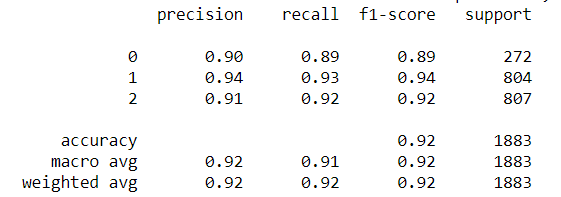
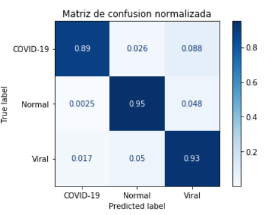
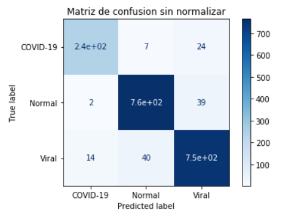


Figura 37

Para test:



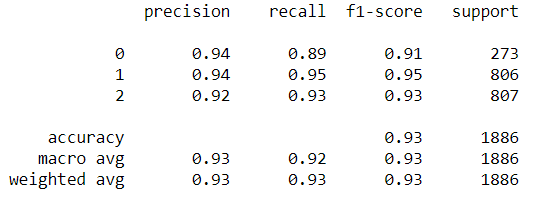


Figura 38

Comprobamos que los resultados obtenidos con OVR son muy similares a los de la regresión logística, por lo que no mejora mucho y nos aumenta mínimamente los tiempos de ejecución, por tanto, concluimos que para esta aproximación con HOG, los mejores resultados que se han obtenido son para regresión logística con los parámetros por defecto y utilizando un mayor número de ejemplos de la clase COVID-19.

Sin embargo, pese a que los resultados obtenidos distan de ser malos, comprobamos como la primera aproximación sigue dando mejores resultados, esto puede ser debido a que como comentábamos antes HOG habitualmente es utilizado para el reconocimiento de objetos en imágenes a partir de sus formas y para nuestro caso de uso encontramos radiografías de tórax de frente y perfil, además de que las manchas de neumonía en los pulmones de los pacientes (tanto viral como de COVID-19) son muy diferentes provocando que el reconocimiento sea más complicado. Por otro lado, reducimos el número de características drásticamente, pasando de 65536 a 2048 algo que también puede haber afectado, aunque también ha tenido efectos positivos como podemos ver en los tiempos de ejecución.

## Patrón binario local para clasificación de texturas (LBP)

Para la extracción de características mediante la textura, vamos a utilizar Local Binary Pattern (LBP). Proporciona información de textura altamente discriminatoria

y se usa para proporcionar información robusta relacionada con patrones.

Divide la imagen en trozos iguales de un determinado tamaño y extrae valores de LBP, que se llaman parches. Los parches se representan en un solo vector para servir como textura.

LBP examina los puntos que rodean un punto central de los parches y prueba si los puntos circundantes son mayores o menores que el punto central (es decir, da un resultado binario).

Como los mejores resultados obtenidos en la primera aproximación han sido el de Regresión logística y el de OneVsRest con Regresión logística, vamos a observar qué resultados nos dan.

Usaremos como función de optimización liblinear, como función de regularización l1, para un score 200 y normalizando.

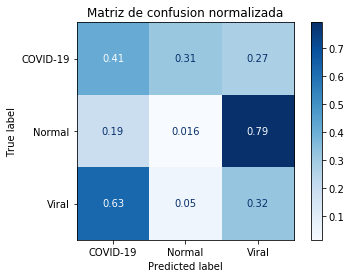
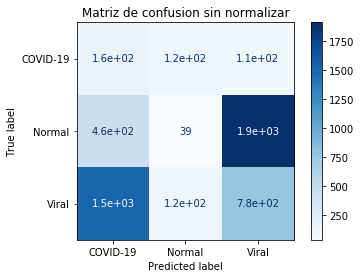
Para la Regresión logística obtenemos estos resultados (Tabla 14):

Tabla 14

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C | Accuracy train | Accuracy val | Accuracy test | Tiempo (s) |
| 2 | 96.4039785% | 64.75315729047072% | 65.90257879656161% | 85.82578253746033 |
| 5 | 99.4452945% | 64.8105625717566% | 64.18338108882521% | 115.62876462936401 |
| 10 | 99.8852333% | 63.77726750861079% | 63.32378223495701% | 135.03434300422668 |
| 100 | 100.0% | 62.51435132032147% | 61.83381088825215% | 138.33332514762878 |

Para C = 5 las matrices de confusión obtenidas son (Figura 37, Figura 38, Figura 39):

Para train:



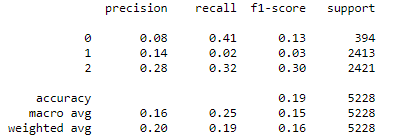
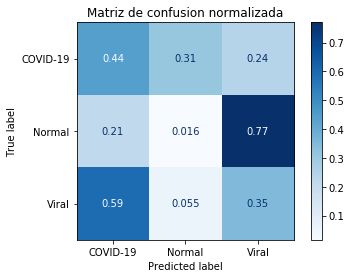
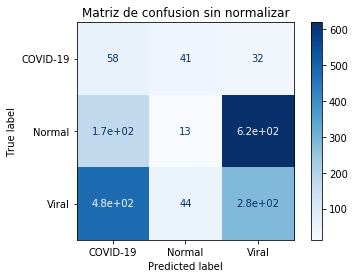


Figura 39

Para val:



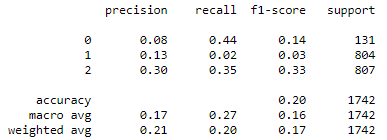
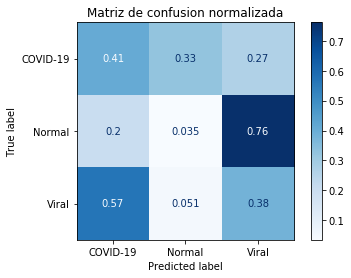
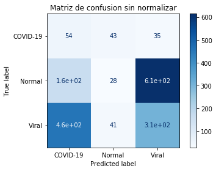


Figura 40

Para test:



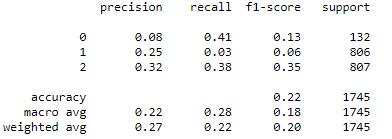


Figura 41

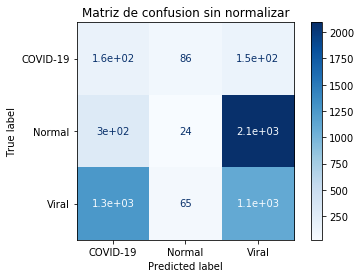
Como vemos en los resultados, son peores que los obtenidos anteriormente en la primera aproximación, por lo que ahora probaremos con OneVsRest con LR (Tabla 15):

Tabla 15

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C | Accuracy train | Accuracy val | Accuracy test | Tiempo (s) |
| 2 | 96.4422341% | 64.69575200918484% | 65.95988538681948% | 104.65287041664124 |
| 5 | 99.4070390% | 64.12169919632606% | 63.72492836676218% | 152.5573170185089 |
| 10 | 99.8852333% | 63.71986222732492% | 63.09455587392551% | 162.75443291664124 |
| 100 | 100.0% | 62.16991963260619% | 61.77650429799427% | 180.42601370811462 |

Nuevamente, para C = 5 las matrices de confusión obtenidas son (Figura 40, Figura 41, Figura 42):

Para train:



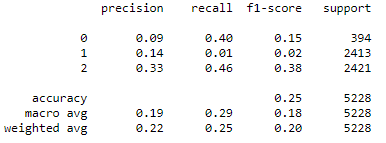
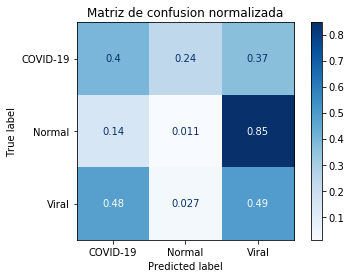
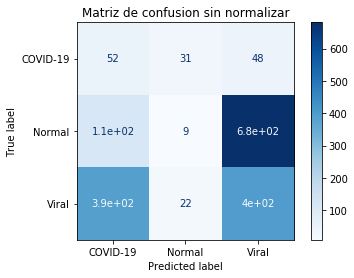


Figura 42

Para val:



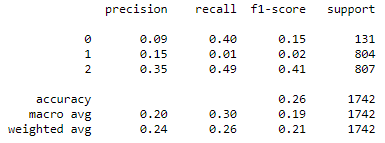
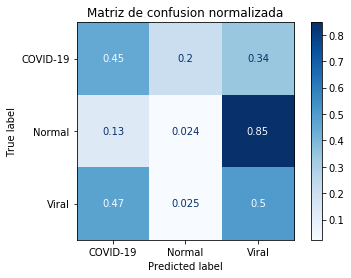
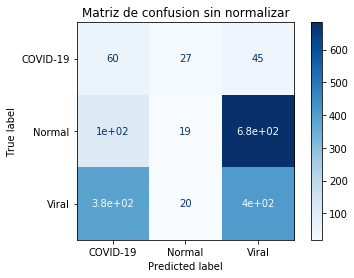


Figura 43

Para test:



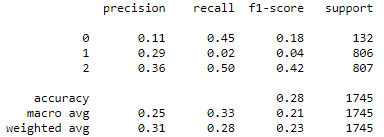


Figura 44

Como se puede observar claramente, mediante esta técnica, también obtenemos peores resultados que los anteriores debido a que la extracción de características busca texturas y en imágenes de rayos X no se pueden apreciar.

De esta forma, no ayuda a identificar de manera correcta si se trata de un caso de covid-19, normal o neumonía viral, por lo que descartamos este método.

# **Tercera aproximación**

Para esta tercera aproximación decidimos utilizar técnicas de Deep Learning, pues se ha demostrado a lo largo de los últimos años que los resultados utilizando este tipo de métodos tienden a ser mejores que el Machine Learning tradicional para la resolución de problemas complejos como el que estamos tratando.

Para visión por computador generalmente se utilizan redes neuronales convolucionales, pues se ha demostrado que para esta rama de la IA funcionan especialmente bien. Antes de empezar a explicar la tercera aproximación y los resultados obtenidos, debemos agradecer al usuario [Adrian Yijie Xu](https://towardsdatascience.com/@adrianitsaxu) el notebook que ha compartido en la página [https://towardsdatascience.com/](https://towardsdatascience.com/detecting-covid-19-induced-pneumonia-from-chest-x-rays-with-transfer-learning-an-implementation-311484e6afc1) el cual hemos modificado para adecuarlo a nuestro problema. Este notebook nos ha sido de gran ayuda puesto que apenas teníamos conocimientos sobre el manejo de la librería Keras para la creación y entrenamiento de redes neuronales convolucionales.

Para la ejecución de este notebook hemos utilizado la plataforma Google Colab, ya que las máquinas virtuales disponibles en esta tienen instaladas todas las dependencias necesarias para la utilización de Keras con soporte de GPU, además el hardware que tienen asignados resulta superior al que tenemos disponible en nuestros ordenadores para la realización de este proyecto.

Comenzamos viendo el procedimiento seguido en el notebook que hemos utilizado, en primer lugar, debemos realizar un preprocesamiento de los datos y preparar los batch para el entrenamiento del modelo de la red, para ello Keras cuenta con una función llamada ImageDataGenerator que permite entre otras cosas: pretratar las imágenes, ampliar el dataset aplicando técnicas como flip, preparar los batch para el entrenamiento.

Una vez preparados los datos el siguiente paso es definir la arquitectura de la red, en este caso comprobamos como Adrian Yijie Xu utiliza la técnica de Transfer Learning, esta técnica consiste en utilizar un modelo pre-entrenado (Keras tiene varios disponibles) y utilizar el dataset disponible para continuar el entrenamiento del mismo, generalmente se entrenan las últimas capas del mismo y el resto se dejan “congeladas”, es decir, no se actualizan los valores de los pesos de las mismas. En este caso la salida del modelo pre-entrenado se pasa a una capa flatten donde se pasa a una única dimensión, después la salida pasa por una capa fully-connected cuya función de activación es una ReLu y finalmente la capa de salida es una softmax donde se realiza la clasificación de las imágenes. Todo lo anterior podemos verlo resumido en la salida de una de las celdas del notebook:

Layer (type)                 Output Shape              Param #

=================================================================

vgg16 (Model)                (None, 8, 8, 512)         14714688

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_2 (Flatten)          (None, 32768)             0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_3 (Dense)              (None, 256)               8388864

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_4 (Dense)              (None, 3)                 771

=================================================================

Total params: 23,104,323

Trainable params: 8,389,635

Non-trainable params: 14,714,688

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

Como modelo, observamos que utiliza una red VGG16 pre-entrenada, esta red suele tener un aspecto similar a este:



No entraremos en detalle en la arquitectura pues consideramos que no es el objetivo de este proyecto, aunque el estudio y ajuste de esta arquitectura u otras similares podría aportar mejoras a el problema que estamos intentando resolver mediante Deep Learning.

Una vez definida la arquitectura, lo siguiente es entrenar el modelo, en este caso los parámetros que se han definido para utilizar durante el entrenamiento son los siguientes:

* Batch size: 10
* Learning Rate: 0.0001
* Número de epoch: 100
* Función de optimización: Adam

Para el resto de los parámetros se utilizarán los valores por defecto.

Realizamos una primera ejecución sin ampliar el dataset y obtenemos los siguientes resultados (Figura 43) para la fase de entrenamiento:

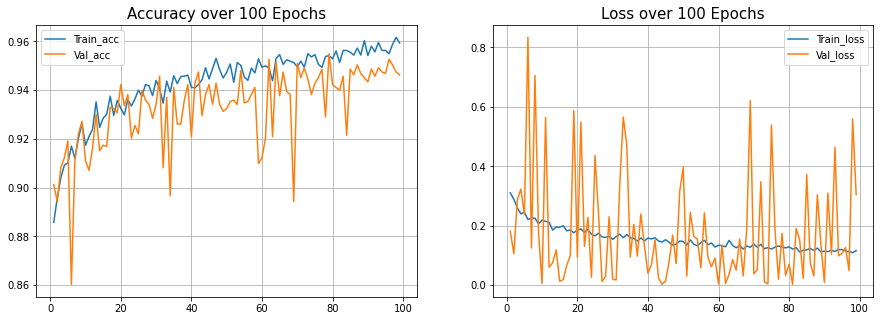


Figura 45

Debemos mencionar que durante la fase de entrenamiento también se realiza el proceso de validación con el modelo en cada epoch, como vemos en las gráficas (Figura 43), tenemos representados los valores de accuracy loss tanto para train como para validación.

Viendo la gráfica de loss (Figura 43) consideramos que algo que podría mejorar los resultados es aplicar algún tipo de regularización o utilizar dropout, ya que parece que el modelo está sobre aprendiendo. No obstante, consideramos que esto solo es una pequeña prueba que nos va a permitir comparar resultados aplicando una metodología diferente a la aplicada en el primer enfoque, por lo que no entraremos a modificar la red o los parámetros del entrenamiento que venían por defecto en el notebook.

Para la fase de test obtenemos los siguientes resultados:

175/175 [==============================] - 2s 12ms/step

Test loss: 1.8954096958623268e-05

Test accuracy: 0.8914285898208618

A priori, con el valor de accuracy los resultados no parecen ser malos. Sin embargo, cuando decidimos imprimir la matriz de confusión (Figura 44) y vemos que efectivamente son bastante buenos, pero pueden mejorarse respecto a la primera aproximación, donde los resultados eran mejores. Esto puede ser debido a que los parámetros establecidos para el entrenamiento deban ajustarse o a que debemos utilizar una arquitectura diferente (Figura 44):

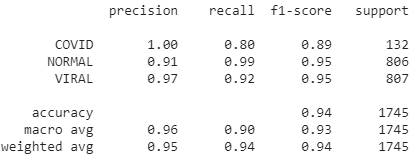


Figura 46

Probamos a realizar la misma prueba, pero esta vez ampliando el conjunto de imágenes de la clase COVID-19. Obtenemos los siguientes resultados para train y validación (Figura 45):

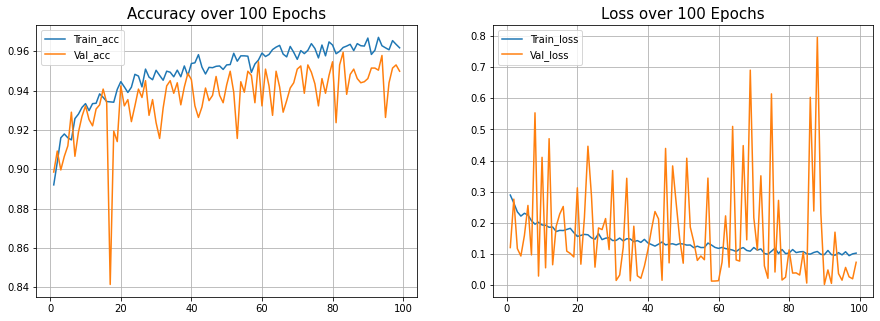


Figura 47

Viendo la gráfica de loss consideramos que algo ha mejorado el añadir más ejemplos de COVID-19, pues parece que el loss de validación en los últimos pasos tiende a estabilizarse y a estar próximo al de train.

Para la fase de test obtenemos los siguientes resultados:

189/189 [==============================] - 2s 8ms/step

Test loss: 3.224252939224243

Test accuracy: 0.8835979104042053

Efectivamente viendo los resultados de loss en test y el accuracy, parece ser que el rendimiento ha mejorado considerablemente respecto al primer caso, pues el loss ha disminuido enormemente y hemos conseguido una mejora sustancial en el accuracy de test. Nos proponemos mostrar la matriz de confusión para ver si los resultados han mejorado en cuanto a recall de la clase COVID y neumonía (Figura 46).

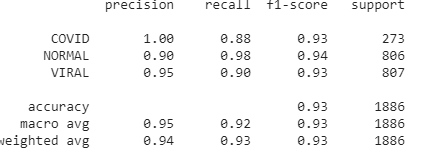


Figura 48

Comprobamos que, aunque los resultados mejoran aún están lejos de los que obtuvimos en la primera aproximación.

Por último, concluimos, que los tiempos de ejecución son mayores que en los casos de la primera y segunda aproximación, también depende del número de epoch que se establezca y el tamaño del batch. Los resultados obtenidos como vemos, son bastante buenos, pero consideramos que pueden mejorarse mucho más incluso ajustando más los parámetros del modelo y la arquitectura de la red neuronal.

# **Problemas**

En este apartado queremos comentar tres problemas que hemos tenido durante la realización del proyecto:

1. Tuvimos un problema con la función listdir() de la librería os, en concreto no sabíamos que cuando devuelve los nombres de los ficheros de un directorio lo hace en desorden, por lo que al principio estuvimos trabajando con las imágenes del dataset en desorden esto provocaba que la salida esperada de cada imagen no concordase con la misma y por tanto que el rendimiento de los clasificadores fuese pésimo en todas las ocasiones (el accuracy en test y validación apenas superaba el 35% y el recall y precisión de COVID eran prácticamente 0). Cuando nos dimos cuenta llevábamos tres cuartas partes del informe, por lo que tuvimos que volver a repetir los resultados y volver a redactar todo el informe.
2. Hemos visto que la función almacenaImagenes del notebook en algunas ocasiones no borra de forma correcta el directorio dataset antes de introducir las nuevas imágenes y a veces devuelve un error, en esta situación hemos comprobado que volver a llamar a la función soluciona el problema.
3. Para la ejecución del notebook que hace uso de la librería Keras decidimos en primer lugar utilizar la plataforma Google Colab, pues todas las dependencias necesarias para ejecutar el notebook ya se encuentran instaladas en la plataforma y además el hardware que ofrecen estas máquinas virtuales es superior al de los ordenadores con los que hemos tenido que trabajar. Por desgracia tuvimos problemas con el almacenamiento de la máquina virtual, pues era inferior al que necesitábamos y finalmente tuvimos que instalar Keras y sus dependencias para la ejecución con soporte de gráficos en nuestros PCs. Más adelante descubrimos que el problema era debido a una función en el notebook que copiaba de manera recursiva el directorio del dataset, por lo que decidimos descartar el uso de este dataset y probamos con uno diferente de nuevo en la plataforma Google Colab.

# **Conclusiones**

Como hemos visto a lo largo de este documento, hemos querido dar dos enfoques para la resolución de este problema, el primero ha sido mediante la utilización de técnicas de Machine Learning tradicional como: regresión logística, regresión logística con regularización, SVMs… mientras que el segundo enfoque está orientado a Deep Learning con la utilización de redes neuronales convolucionales para la resolución del problema.

Hemos comprobado que en el enfoque tradicional lo que mejores resultados nos ha dado ha sido tratar directamente los píxeles de la imagen como características del problema, pues esta técnica nos ha dado los mejores resultados tanto en accuracy como en recall de las clases COVID-19 y neumonía viral (que es lo que pretendíamos desde un primer momento), algo que en un principio no esperábamos ya que tratar los píxeles directamente para problemas tan complejos como este no suele dar muy buenos resultados. No obstante, tenemos serias dudas de si el comportamiento de esta aproximación sería igual de bueno en un entorno real.

También hemos observado que tratar los pixeles como tal conlleva un mayor coste computacional pues se debe trabajar con una mayor cantidad de información, lo que implica tiempos de ejecución mayores para realizar el entrenamiento, no obstante, utilizar técnicas como selectKBest para reducir el número de características nos ha permitido reducir los tiempos de ejecución a más de la mitad sin perder demasiado rendimiento en los modelos. Además, nos hemos dado cuenta que para este problema son necesarias cada una de las características de la imagen pues siempre que eliminábamos alguna de ellas perdíamos en rendimiento.

Con respecto a la segunda aproximación dentro del primer enfoque del problema, hemos comprobado como existen diversas técnicas para la extracción automática de características de las imágenes, lo que en muchos casos permite obtener un menor número de características pero de más alto nivel y de mejor calidad que en la primera aproximación, en el caso de HOG hemos visto como los resultados eran casi igual de buenos que para la primera aproximación pero reduciendo drásticamente el número de características a tratar y por tanto los tiempos de ejecución, algo que en la primera aproximación era inviable, pues si reducíamos en exceso el número de características acabamos teniendo un rendimiento demasiado malo. Sin embargo, como ya hemos comentado, los resultados siguen siendo mejores en la primera aproximación, esto posiblemente se debe a que trabajamos tanto con imágenes de tórax de frente como de perfil y a que las formas de las manchas de neumonía en los pulmones son muy variadas por lo que este método no termina de funcionar perfectamente.

En cuanto a LBP hemos podido comprobar como los resultados han sido bastante malos con respecto al resto de técnicas, esto puede ser debido a que como comentábamos más arriba esta técnica permite extraer características relacionadas con la textura de las imágenes y en nuestro caso de uso las imágenes resultan bastante planas en cuanto a textura con lo que esta técnica no aporta demasiada información útil para realizar una clasificación óptima de las imágenes del dataset.

Por último, debemos hablar del segundo enfoque que como hemos podido ver los resultados que nos ha devuelto han sido muy buenos que es justo lo que esperábamos, pues se ha demostrado en muchas ocasiones como utilizar Deep Learning en problemas de visión por computador permite una mejor resolución de problemas de clasificación utilizando para ello redes neuronales convolucionales. Por tanto, pensamos que esta es la mejor forma de resolver este problema de entre los dos enfoques que le hemos dado.

# 

# **Posibles mejoras**

Consideramos que algunas de las posibles mejoras que se podrían dar a este problema para mejorar aún más los resultados son:

1. Conseguir un dataset de mayor tamaño y más variado, pues como hemos visto lo que ha conseguido mejoras sustanciales en todos los casos ha sido añadir más ejemplos de COVID-19 al dataset del problema.
2. Utilizar otras técnicas de extracción de características, pues nosotros solo hemos podido probar HOG y LBP, pero hemos visto como librerías como scikit-image tiene una gran variedad de funciones y alguna de ellas podría funcionar mejor para nuestro caso de uso.
3. Aprender a utilizar librerías como Keras o similares y proponer una arquitectura de capas convolucionales diferente con la que mejorar los resultados, modificar los parámetros del proceso de entrenamiento, añadir dropout a la red...