Universidad de Granada

Proyecto Final

Clasificación de radiografías torácicas para detección de COVID-19

Alberto Luque Infante
David Villar Martos

Quinto curso del Doble Grado de Ingeniería Informática y Matemáticas:
Visión por Computador

28 de enero de 2021

Índice

1. Descripción del proyecto

El proyecto consiste en abordar el problema de clasificación de radiografías torácicas con la intención de poder detectar casos positivos de COVID-19.

Partiendo de un modelo de red convolucional preentrenado con la base de datos ImageNet, lo utilizaremos como base para encontrar el mejor modelo que nos permita llevar a cabo esta tarea.

En la base de datos escogida se utilizan imágenes clasificadas en tres clases distintas: radiografías de pulmones sanos (NORMAL), radiografías de pulmones en casos positivos de COVID-19 (COVID) y radiografías de pulmones afectados con neumonía vírica (Viral Pneumonia). Tener este conjunto de datos es muy interesante puesto que no sólo tendremos que distinguir pulmones sanos de pulmones enfermos, sino también distinguir entre pulmones enfermos con neumonías víricas, siendo distinto el tipo de virus.

Por tanto, nuestro problema de clasificación va a consistir en implementar un modelo que nos permita clasificar estas imágenes en las tres clases.

Primero haremos un análisis previo del problema para poder enfocarlo correctamente. A continuación comenzaremos con la lectura y preprocesamiento de los datos, conformando los conjuntos de train y test.

Hemos elegido como modelo inicial preentrenado con ImageNet el modelo DenseNet121. Partiendo de una primera versión muy básica poco a poco le iremos añadiendo mejoras hasta encontrar el modelo más óptimo.

Posteriormente, utilizaremos otros modelos conocidos preentrenados también en ImageNet para poder compararlos entre sí.

Finalmente, vamos realizar un análisis pormenorizado de los resultados obtenidos, visualizando los mapas de activación y mapas de calor con el objetivo de detectar qué zonas de las imágenes de entrada son más discriminativas de cara a realizar la clasificación. De esta forma podremos entender un poco mejor el modelo y extraer conclusiones que puedan ayudar también al campo de la medicina.

Por último, comentaremos algunas propuestas o aspectos que se podrían mejorar en un futuro en base a la experiencia obtenida.

2. Análisis previo del problema

El uso de la radiografía simple de tórax para la detección de patologías torácicas, es una técnica muy efectiva y se considera la exploración base a realizar, debido a la gran cantidad de información que es capaz de aportar. Se utilizan también otro tipo de técnicas, tales como la tomografía computerizada, resonancia magnética, radioscopia o ecografía, pero siempre como apoyo a la radiografía torácica.

Las proyecciones básicas de la radiografía simple de tórax son la postoanterior (frontal) y la lateral. Normalmente se practican en inspiración máxima y sostenida, con el paciente en bipedestación. La lectura de la radiografía de tórax por parte de un médico se hace siempre de forma sistemática, analizando secuencialmente y por orden: partes blandas, hueso, diafragma, mediastino, hilios pulmonares, pleura y parénquima pulmonar.

Clínicamente la neumonía se define como una consolidación pulmonar en una radiografía de tórax junto con signos y síntomas clínicos de infección respiratoria (fiebre, tos y expectoración). La consolidación pulmonar se ve reflejada a nivel radiológico en que aumenta la opacidad de los pulmones debido al cúmulo de otras sustancias más densas que el aire.

La consolidación pulmonar típica es debida con frecuencia a la presencia de una neumonía, pero este patrón radiológico no es específico de ella, ya que cualquier enfermedad que ocupe el espacio aéreo producirá la misma imagen. Por tanto, el examen radiológico es indispensable para el diagnóstico, pero carece de especificidad, por lo que el diagnóstico de la radiografía está ligada a la correlación clínica del paciente.

Las neumonías con respecto al nivel radiológico se dividen en 3 tipos: Neumonías lobulares y segmentarias, Bronconeumonías, y por último, Neumonías intersticiales.

La neumonía vírica COVID-19 entra dentro de este último grupo. Desde el mes de diciembre del año 2019, cuando se detectó el primer caso de COVID-19, el virus se ha propagado a nivel mundial teniendo una alta tasa de contagio, considerándose una pandemia desde marzo de 2020. Ha provocado desde entonces la mayor crisis sanitaria que se ha vivido globalmente en los tiempos modernos, y es una prioridad el intentar controlar la pandemia. Las radiografías torácicas permiten diagnosticar la enfermedad y entender mejor cómo afecta al organismo, por lo que consideramos muy relevante su estudio. Sin embargo, es posible confundir una neumonía vírica provocada por COVID-19 con otras neumonías víricas ocasionadas por otros patógenos.

Existen ya algunos estudios los cuales intentan encontrar patrones radiológicos que intenten diferenciar el COVID-19 de otros tipos de neumonías víricas, como en
https://www.researchgate.net/publication/339839348_Performance_of_radiologists_in_differentiating_COVID-19_from_viral_pneumonia_on_chest_CT o en
https://insightsimaging.springeropen.com/articles/10.1186/s13244-020-00933-z, donde se encuentran dificultades para ello.

Por tanto, nuestro estudio tratará de desarrollar mediante redes neuronales preentrenadas, un método efectivo para clasificación de radiografías torácicas postoanteriores, que sea capaz de distinguir pacientes con pulmones sanos de pacientes con pulmones enfermos, por neumonías víricas, pero además, que sea capaz de distinguir entre neumonías víricas, centrándonos en la distinción de que el agente patógeno sea el COVID-19 o no, para un mejor estudio y entendimiento de esta enfermedad.

3. Base de Datos elegida para el problema

3.1. Información de la Base de Datos

La base de datos elegida para el proyecto se encuentra en Kaggle :
https://www.kaggle.com/tawsifurrahman/covid19-radiography-database.

La base de datos, contiene un total de 3886 imágenes de radiografías torácicas postoanteriores en su actualización del 5 de enero de 2021 (versión 3). La distribución de imágenes por clases es de 1200 imágenes de la clase COVID, 1341 imagenes de la clase NORMAL y 1345 imágenes de la clase Viral Pneumonia.

El conjunto de imágenes del dataset proceden de distintas fuentes. Las imágenes de la clase COVID proceden de las siguientes fuentes:

- 400 radiografias torácicas frontales de la fuente de github : https://github.com/armiro/COVID-CXNet
- 183 radiografias torácicas frontales del colegio médico de Hannover, Alemania: https://github.com/ml-workgroup/covid-19-image-repository/tree/master/png.
- 617 radiografías torácicas frontales de entre las siguientes fuentes

  https://sirm.org/category/senza-categoria/covid-19/ , Sociedad Italiana de Radiología Médica

  https://eurorad.org, base de datos operada por Sociedad Europea de Radiología (ESR)

  https://github.com/ieee8023/covid-chestxray-dataset . Aquí las imágenes vienen de distintas fuentes públicas además de otras fuentes indirectas de hospitales y físicos.

.

El resto de imágenes (imágenes de pulmones sanos e imágenes de pulmones con otro tipo de neumonías víricas) provienen del dataset https://github.com/ieee8023/covid-chestxray-dataset. Las imágenes de este conjunto de datos provienen de un centro médico en Guangzhou.

No hemos encontrado evidencia, ni de que se repitan imágenes en las imágenes con coronavirus, cosa que podría haber sido posible por el hecho de que se han extraído de varias fuentes, y de hecho hay varias fuentes que cogen imágenes de otras que aquí aparecen (la de 400 coge de sirm, por ejemplo) ni de que haya varias imágenes correspondientes a un mismo paciente en distintos instantes de tiempo.

Este conjunto de imágenes podría considerarse un conjunto de datos pequeño en comparación con otros grandes conjuntos de datos ampliamente empleados en el campo de visión por computador como pudiera ser Imagenet, CIFAR,... Sin embargo, debido a lo relativamente reciente que es la pandemia del COVID-19(1 año y 2 meses), y debido a que son imágenes médicas, es complicado encontrar grandes conjuntos de datos para esta tarea. El conjunto de datos actual es el más extenso de entre todos los que hemos podido encontrar.

3.2. Lectura de la Base de Datos

Para que las clases estén equilibradas vamos a tomar el mismo número de imágenes por clase, 1200, luego tendremos un total de 3600 imágenes.

Para leer las imágenes, creamos un vector de etiquetas con las 1200 etiquetas para cada clase y utilizamos la función implementada *leerImagenes*.

Leemos las imágenes de cada clase haciendo interpolación bilineal para que tengan un tamaño de (224,224), que es el tamaño que tienen las imágenes de ImageNet.

```
def leerImagenes(clases, num_imgs, path):

  imgs_covid = np.array([img_to_array(load_img(path + "/" + clases[i] + "/" +
      clases[i] + " (" + str(i+1) + ").png",
                                          target_size = (224, 224),
                                              interpolation="bilinear")) for i in
                                              range(0,num_imgs)])
  imgs_normal = np.array([img_to_array(load_img(path + "/" + clases[i] + "/" +
      clases[i] + " (" + str(i-num_imgs+1) + ").png",
                                          target_size = (224, 224),
                                              interpolation="bilinear")) for i in
                                              range(num_imgs,2*num_imgs)])
  imgs_viral = np.array([img_to_array(load_img(path + "/" + clases[i] + "/" +
      clases[i] + " (" + str(i-2*num_imgs+1) + ").png",
                                          target_size = (224, 224),
                                              interpolation="bilinear")) for i in
                                              range(2*num_imgs,3*num_imgs)])

  return imgs_covid, imgs_normal, imgs_viral
```

Un ejemplo de una imagen de cada clase es el siguiente:

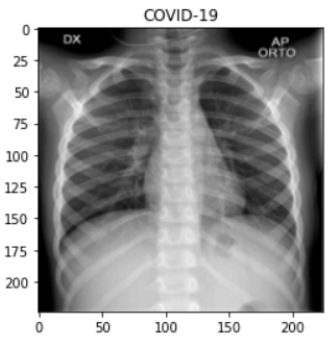

Figura 1: Imagen de la clase COVID

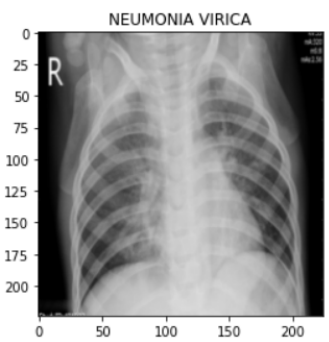

Figura 2: Imagen de la clase Viral Pneumonia

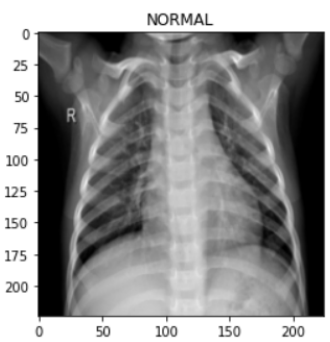

Figura 3: Imagen de la clase NORMAL

3.3. Creación de los conjuntos de train y test

Primero creamos un vector agrupando las imágenes de las tres clases y cambiamos las etiquetas por números enteros, asignando el valor 0 a la clase COVID, el 1 a la clase NORMAL y el 2 a la clase Viral Pneumonia.

A continuación, usando la función *to_categorical* obtenemos una representación binaria de las clases (la clase 0 pasa a ser [1,0,0], la 1 es [0,1,0] y la 2 es [0,0,1]).

Después de hacer una permutación de las imágenes, dividimos en proporción 80 %-20 % para obtener los conjuntos de entrenamiento y de test, obteniendo 2880 imágenes para entrenar y 720 para test.

4. Implementación de los modelos

4.1. DenseNet121

DenseNet121 es el modelo que hemos elegido para empezar. En las primeras versiones vamos a utilizar el modelo, con los pesos de ImageNet, solo como extractor de características, a partir del cuál añadiremos modificaciones. Finalmente, haremos un ajuste fino del modelo donde sí moveremos los pesos.

4.1.1. Version 0: Extractor de características inicial, sin entrenamiento

Primero vamos a utilizar el modelo DenseNet121 preentrenado con IMAGENET sin entrenar ningún peso. Partiendo de las características extraídas, simplemente vamos a añadir la capa softmax y comprobaremos que los resultados no son para nada buenos, pues todavía no hemos entrenado nuevos pesos, no estamos adaptando la red a nuestros datos en absoluto.
Extraemos las características:

```
#Definicion inicial de nuestro modelo preentrenado en imagenet.
#Para usarlo como extractor de caracteristicas simple

#Vamos a cargar el modelo antes del ultimo pooling, para usarlo como extractor de caracteristicas
feat_extractor = DenseNet121(include_top=False, weights='imagenet', pooling='avg')
feat_extractor.trainable = False

#Una vez tenemos el modelo, vamos a compilarlo usando un optimizador y una funcion de perdida
opt = SGD(lr=0.01, decay= 1e-6, momentum=0.9, nesterov=True)

feat_extractor.compile(optimizer=opt, loss="categorical_crossentropy", metrics=["acc"])

#Con esto tenemos el modelo de densenet hasta que nos deja los datos en forma de vector unidimensional
#de dimension 1024
feat_extractor = keras.Model(inputs=feat_extractor.inputs, outputs=feat_extractor.layers[-1].output)

feat_extractor.compile(optimizer=opt, loss="categorical_crossentropy", metrics=["acc"])

# Extraer las caracteristicas de las imagenes con el modelo anterior.
car_train = feat_extractor.predict(x_train, verbose=1)
car_test = feat_extractor.predict(x_test, verbose=1)
```

Una vez tenemos las características, añadimos la capa softmax y hacemos las predicciones.

```
inputs = keras.layers.Input(shape=[1024])
outputs = keras.layers.Dense(units=3, activation="softmax")(inputs)


dense_model = keras.Model(inputs=inputs, outputs=outputs)
opt = SGD(lr=0.01, decay= 1e-6, momentum=0.9, nesterov=True)
dense_model.compile(optimizer=opt, loss="categorical_crossentropy", metrics=["acc"])

#Clasificamos
y_preds = dense_model.predict(car_test, verbose=True)
```

```
11
12 print("La accuracy del modelo es: " + str(calcularAccuracy(y_test, y_preds)))
13
14 y_test_conf = np.argmax(y_test, axis=1)
15 y_preds = np.argmax(y_preds, axis=1)
16 print("La matriz de confusion de las predicciones ha sido: \n",
       confusion_matrix(y_test_conf, y_preds))
```

RESULTADOS.....

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	

4.1.2. Version 1: Extractor de características inicial, entrenando la capa de salida

Ahora vamos a reentrenar sólo la capa de salida, sin ningún tratamiento adicional. La única diferencia respecto a la versión anterior es que sí que entrenamos la última capa:

```
1 hist = dense_model.fit(car_train, y_train, batch_size=32, epochs=20,
      validation_split=0.1)
```

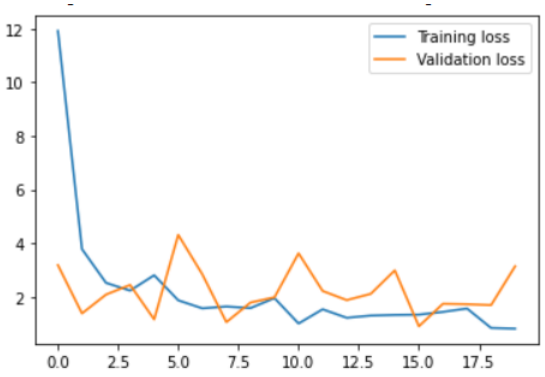

Figura 4: Training-Validation Loss

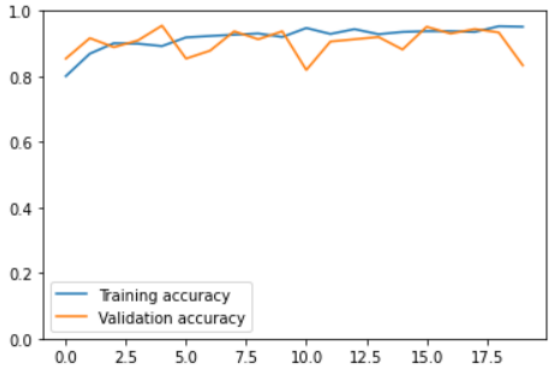

Figura 5: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20

4.1.3. Version 2: Data augmentation, más capas densas

Seguimos empleando la red preentrenada como extractor de características pero se realiza un preprocesado más exhaustivo de imágenes, con image augmentation y normalización, además de hacer un modelo denso de mayor profundidad.

Además de hacer una normalización de las imágenes, para que tengan media cero y varianza 1, hemos detectado que hay imágenes que salen muy blancas y otras muy oscuras, luego debemos conseguir invarianza frente al brillo (añadiendo el parámetro brightness).

En cuanto al data augmentation, los pulmones en las radiografías salen siempre verticales, por lo que no tiene mucho sentido añadir rotaciones pero sí flips horizontales. Los zooms de ampliación también pueden ser interesantes, puesto que las imágenes tienen siempre los pulmones en la zona central, y podemos evitar posibles fuentes de ruido de letras que tienen algunas de las imágenes en las zonas laterales, que se perderán haciendo zoom.

Para aplicar todo esto, creamos un objeto de la clase ImageDataGenerator:

```
#Image data generators

train_generator = ImageDataGenerator(featurewise_center = True,
                            featurewise_std_normalization = True,
                            validation_split=0.1,
                            horizontal_flip=True,
                            brightness_range=[0.8,1.25],
                            zoom_range=[1,1.2])
train_generator.fit(x_train)

test_generator = ImageDataGenerator(featurewise_center = True,
                            featurewise_std_normalization = True)

test_generator.fit(x_train)

it = train_generator.flow(x_train, batch_size=1)
```

Añadimos también alguna capa densa más para ganar profundidad:

```
inputs = keras.layers.Input(shape=[1024])
layer1 = keras.layers.Dense(units=500, activation="relu")(inputs)
layer2 = keras.layers.Dense(units=200, activation="relu")(layer1)
outputs = keras.layers.Dense(units=3, activation="softmax")(layer2)
```

RESULTADOS.....

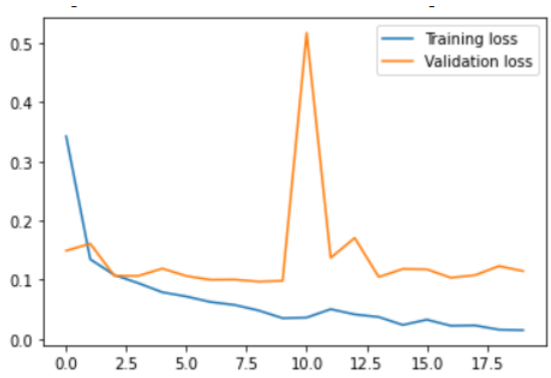

Figura 6: Training-Validation Loss

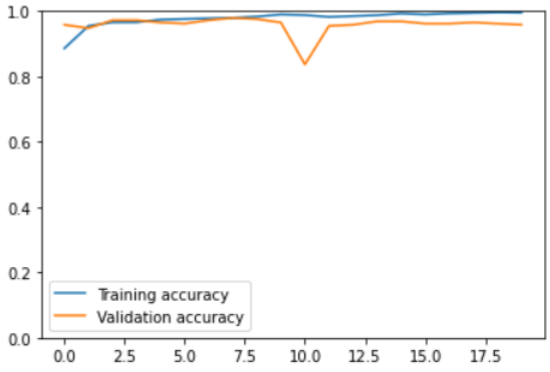

Figura 7: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20

En esta última versión puede ser que tengamos un poco de overfitting puesto que el validation

loss no decrece al mismo ritmo que el training loss. Regularizamos añadiendo una capa de Dropout.

```
inputs = keras.layers.Input(shape=[1024])
layer1 = keras.layers.Dense(units=500, activation="relu")(inputs)
layer2 = keras.layers.Dropout(0.5)(layer1)
layer3 = keras.layers.Dense(units=200, activation="relu")(layer2)
outputs = keras.layers.Dense(units=3, activation="softmax")(layer3)
```

RESULTADOS.....

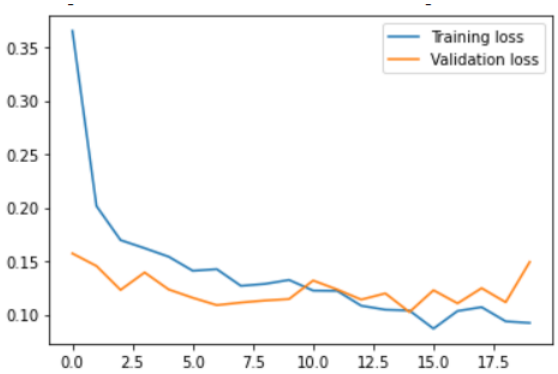

Figura 8: Training-Validation Loss

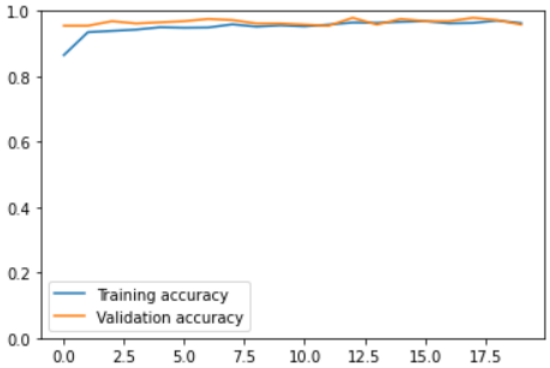

Figura 9: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20
DN-V2-reg	0.0883	0.9683	0.1494	0.9583	0.9569	20

4.1.4. Version 3: Fine tunning

Vamos a probar ahora a hacer un ajuste fino de la red completa, tomando como pesos iniciales los de imagenet. Ahora ya no solo vamos a utilizar DenseNet como extractor de características, sino que, en vez de estar la red congelada, vamos a ir moviendo sus pesos con el entrenamiento.

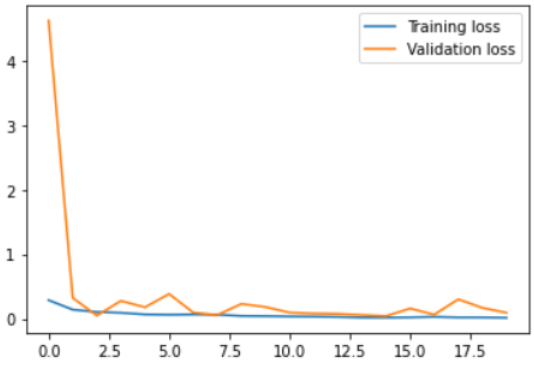

Figura 10: Training-Validation Loss

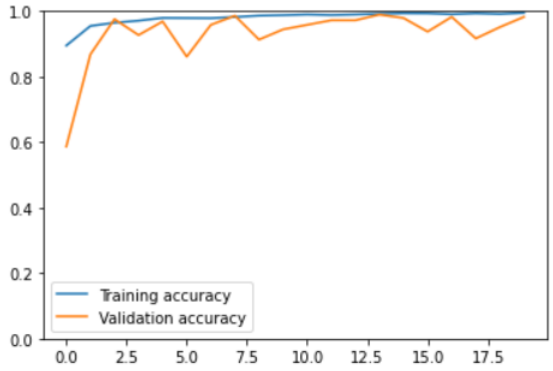

Figura 11: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20
DN-V2-reg	0.0883	0.9683	0.1494	0.9583	0.9569	20
DN-V3	0.0089	0.9964	0.0924	0.9826	0.9736	20

4.2. ResNet50

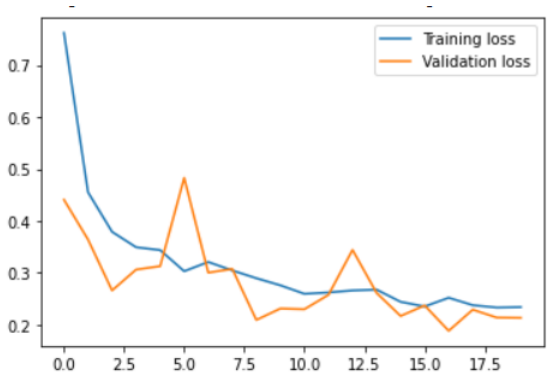

Figura 12: Training-Validation Loss

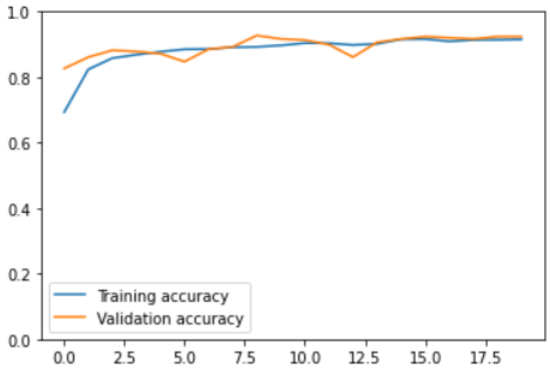

Figura 13: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20
DN-V2-reg	0.0883	0.9683	0.1494	0.9583	0.9569	20
DN-V3	0.0089	0.9964	0.0924	0.9826	0.9736	20
ResNet	0.2519	0.9063	0.2134	0.9236	0.8805	20

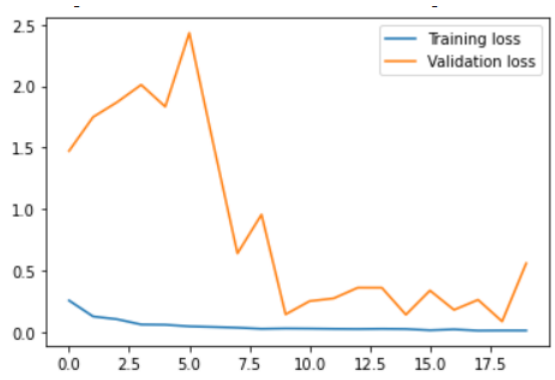

Figura 14: Training-Validation Loss

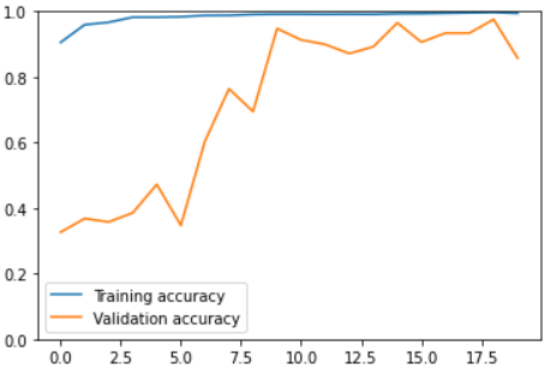

Figura 15: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20
DN-V2-reg	0.0883	0.9683	0.1494	0.9583	0.9569	20
DN-V3	0.0089	0.9964	0.0924	0.9826	0.9736	20
ResNet	0.2519	0.9063	0.2134	0.9236	0.8805	20
ResNet-FT	0.0102	0.9952	0.5594	0.8576	0.8736	20

4.3. VGG16

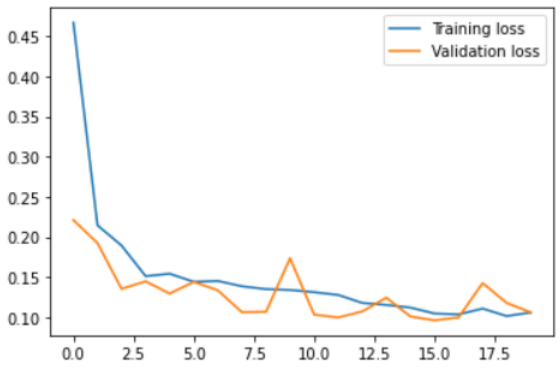

Figura 16: Training-Validation Loss

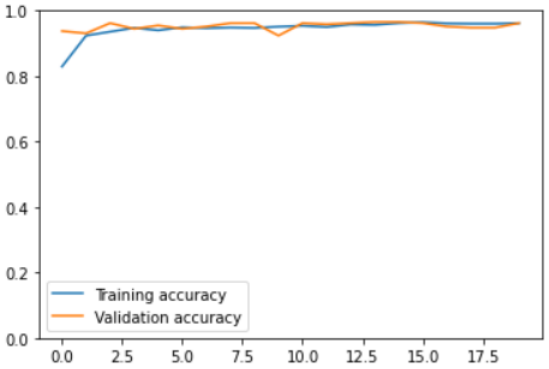

Figura 17: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20
DN-V2-reg	0.0883	0.9683	0.1494	0.9583	0.9569	20
DN-V3	0.0089	0.9964	0.0924	0.9826	0.9736	20
ResNet	0.2519	0.9063	0.2134	0.9236	0.8805	20
ResNet-FT	0.0102	0.9952	0.5594	0.8576	0.8736	20
VGG16	0.1057	0.9626	0.1063	0.9618	0.9375	20

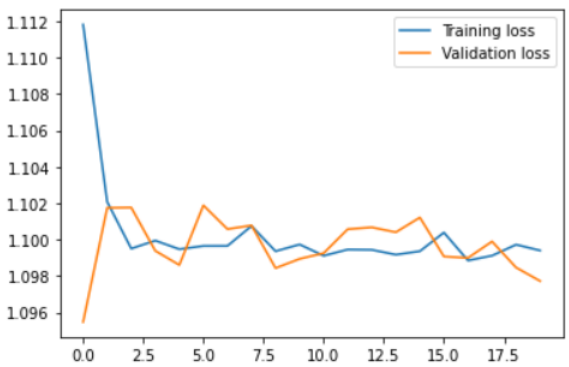

Figura 18: Training-Validation Loss

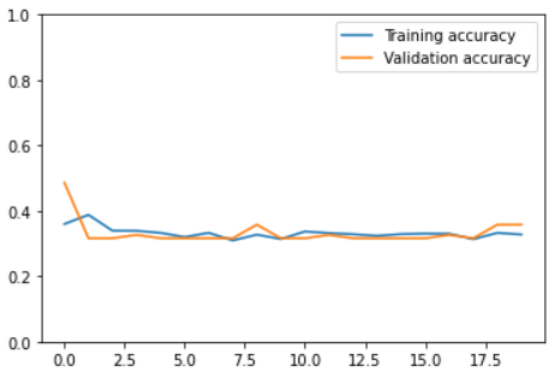

Figura 19: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20
DN-V2-reg	0.0883	0.9683	0.1494	0.9583	0.9569	20
DN-V3	0.0089	0.9964	0.0924	0.9826	0.9736	20
ResNet	0.2519	0.9063	0.2134	0.9236	0.8805	20
ResNet-FT	0.0102	0.9952	0.5594	0.8576	0.8736	20
VGG16	0.1057	0.9626	0.1063	0.9618	0.9375	20
VGG16-FT	1.099	0.3159	1.0977	0.3576	0.3388	20

4.4. InceptionV3

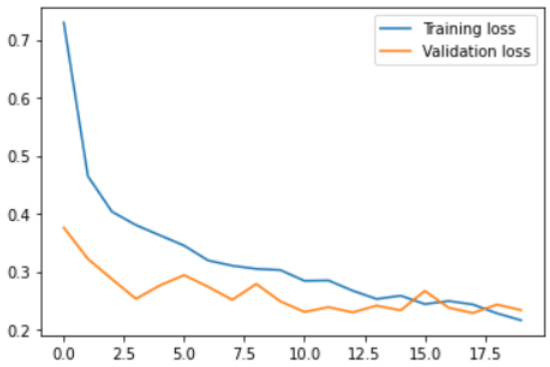

Figura 20: Training-Validation Loss

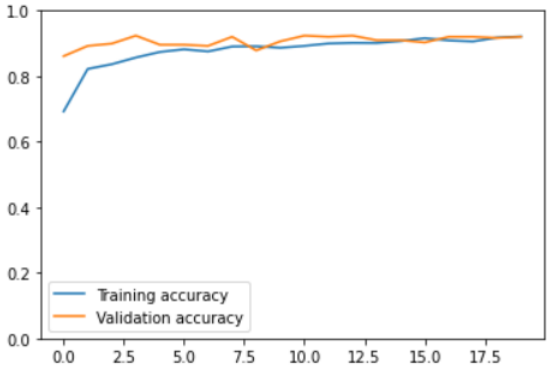

Figura 21: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20
DN-V2-reg	0.0883	0.9683	0.1494	0.9583	0.9569	20
DN-V3	0.0089	0.9964	0.0924	0.9826	0.9736	20
ResNet	0.2519	0.9063	0.2134	0.9236	0.8805	20
ResNet-FT	0.0102	0.9952	0.5594	0.8576	0.8736	20
VGG16	0.1057	0.9626	0.1063	0.9618	0.9375	20
VGG16-FT	1.099	0.3159	1.0977	0.3576	0.3388	20
InceptionV3	0.2194	0.9231	0.2337	0.9201	0.8903	20

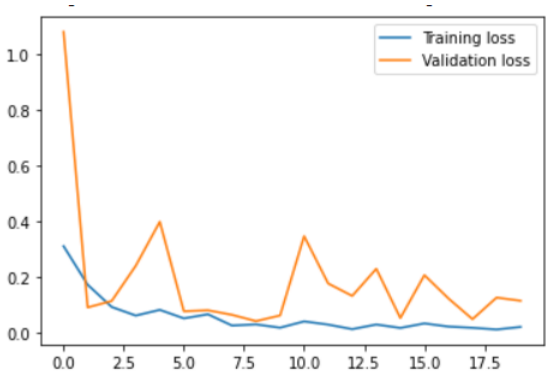

Figura 22: Training-Validation Loss

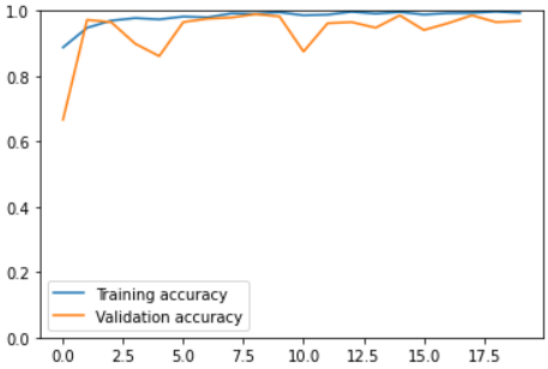

Figura 23: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20
DN-V2-reg	0.0883	0.9683	0.1494	0.9583	0.9569	20
DN-V3	0.0089	0.9964	0.0924	0.9826	0.9736	20
ResNet	0.2519	0.9063	0.2134	0.9236	0.8805	20
ResNet-FT	0.0102	0.9952	0.5594	0.8576	0.8736	20
VGG16	0.1057	0.9626	0.1063	0.9618	0.9375	20
VGG16-FT	1.099	0.3159	1.0977	0.3576	0.3388	20
InceptionV3	0.2194	0.9231	0.2337	0.9201	0.8903	20
InceptionV3-FT	0.0146	0.9953	0.1142	0.9688	0.9792	20

4.5. Xception

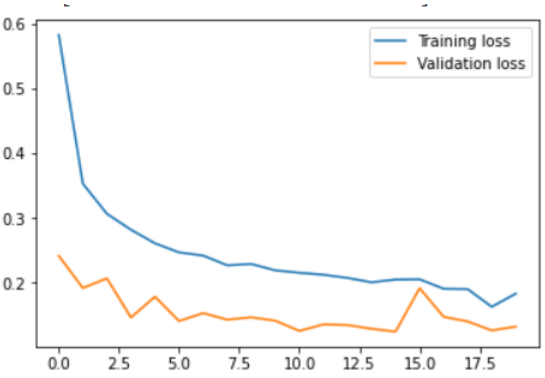

Figura 24: Training-Validation Loss

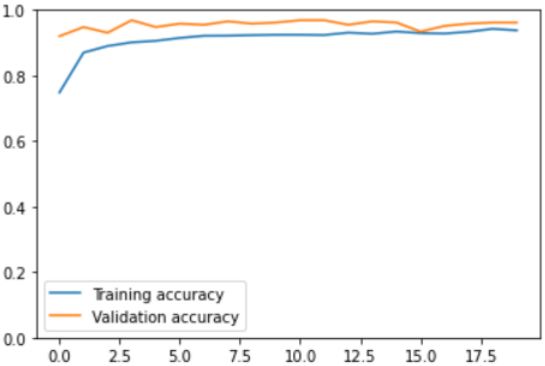

Figura 25: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20
DN-V2-reg	0.0883	0.9683	0.1494	0.9583	0.9569	20
DN-V3	0.0089	0.9964	0.0924	0.9826	0.9736	20
ResNet	0.2519	0.9063	0.2134	0.9236	0.8805	20
ResNet-FT	0.0102	0.9952	0.5594	0.8576	0.8736	20
VGG16	0.1057	0.9626	0.1063	0.9618	0.9375	20
VGG16-FT	1.099	0.3159	1.0977	0.3576	0.3388	20
InceptionV3	0.2194	0.9231	0.2337	0.9201	0.8903	20
InceptionV3-FT	0.0146	0.9953	0.1142	0.9688	0.9792	20
Xception	0.1746	0.9405	0.1319	0.9618	0.9153	20

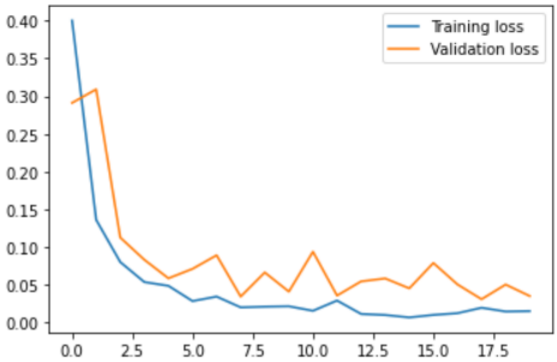

Figura 26: Training-Validation Loss

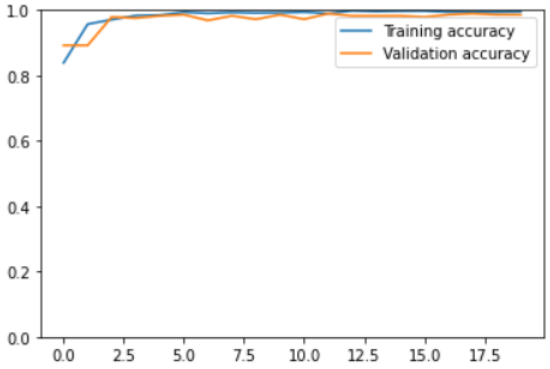

Figura 27: Training-Validation accuracy

Modelo	Loss	Accuracy	Validation Loss	Validation Accuracy	Test Accuracy	Número Épocas
DN-V0					0.3264	
DN-V1	0.6501	0.9588	3.1393	0.8333	0.8361	20
DN-V2	0.0109	0.9973	0.1146	0.9583	0.9597	20
DN-V2-reg	0.0883	0.9683	0.1494	0.9583	0.9569	20
DN-V3	0.0089	0.9964	0.0924	0.9826	0.9736	20
ResNet	0.2519	0.9063	0.2134	0.9236	0.8805	20
ResNet-FT	0.0102	0.9952	0.5594	0.8576	0.8736	20
VGG16	0.1057	0.9626	0.1063	0.9618	0.9375	20
VGG16-FT	1.099	0.3159	1.0977	0.3576	0.3388	20
InceptionV3	0.2194	0.9231	0.2337	0.9201	0.8903	20
InceptionV3-FT	0.0146	0.9953	0.1142	0.9688	0.9792	20
Xception	0.1746	0.9405	0.1319	0.9618	0.9153	20
Xception-FT	0.0157	0.9946	0.0352	0.9861	0.9875	20

5. Análisis del modelo: mapas de activación, mapas de calor y conclusiones

6. Posibles propuestas de mejora

7. Referencias

https://www.kaggle.com/tawsifurrahman/covid19-radiography-database