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

27 de enero de 2021

Índice

1. Descripción del proyecto **2**

2. Análisis previo del problema **3**

3. Base de Datos elegida para el problema **3**
3.1. Información de la Base de Datos 3
3.2. Lectura de la Base de Datos 3
3.3. Creación de los conjuntos de train y test 4

4. Implementación de los modelos **4**
4.1. DenseNet121 4
4.1.1. Version 0: Extractor de características inicial, sin entrenamiento 4
4.1.2. Version 1: Extractor de características inicial, entrenando la capa de salida 5
4.1.3. Version 2: Data augmentation, más capas densas 5
4.1.4. Version 3: Fine tunning 6
4.2. ResNet50 7
4.3. VGG19 7
4.4. InceptionV3 7
4.5. Xception 7

5. Análisis del modelo: mapas de activación, mapas de calor y conclusiones **7**

6. Posibles propuestas de mejora **7**

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, utilizarlo 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).

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

AQUI PODRIAN IR COSAS DE LAS QUE TE PASO TU AMIGO DE MEDICINA

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, en su actualización del 5 de enero de 2021 (versión 3), contiene un total 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, entre otras fuentes, de un colegio médico alemán y del SIRM (Italian Society of Medical and Interventional Radiology).

Para las clases NORMAL y Viral Pneumonia también se han utilizado imágenes de distintas bases de datos públicas, algunas de ellas pertenecientes al ESR (European Society of Radiology).

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 interpolando para que tengan un tamaño de (224,224), que es el tamaño que tienen las imágenes de ImageNet.

```
1  def leerImagenes(clases, num_imgs, path):
2
3    imgs_covid = np.array([img_to_array(load_img(path + "/" + clases[i] + "/" +
         clases[i] + " (" + str(i+1) + ").png",
4                                            target_size = (224, 224),
                                                 interpolation="bilinear")) for i in
                                                 range(0,num_imgs)])
5    imgs_normal = np.array([img_to_array(load_img(path + "/" + clases[i] + "/" +
         clases[i] + " (" + str(i-num_imgs+1) + ").png",
6                                            target_size = (224, 224),
                                                 interpolation="bilinear")) for i in
                                                 range(num_imgs,2*num_imgs)])
7    imgs_viral = np.array([img_to_array(load_img(path + "/" + clases[i] + "/" +
         clases[i] + " (" + str(i-2*num_imgs+1) + ").png",
8                                            target_size = (224, 224),
                                                 interpolation="bilinear")) for i in
                                                 range(2*num_imgs,3*num_imgs)])
9
10   return imgs_covid, imgs_normal, imgs_viral
```

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

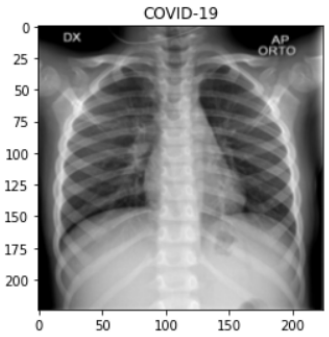

Figura 1: Imagen de la clase COVID

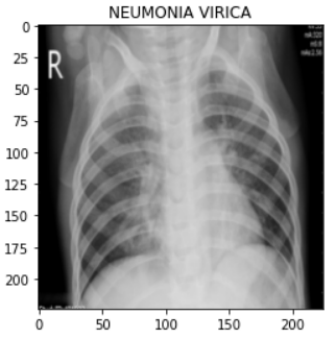

Figura 2: Imagen de la clase Viral Pneumonia

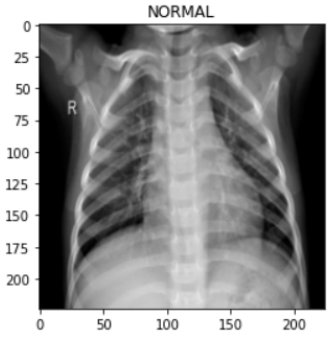

Figura 3: Imagen de la clase NORMAL

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)

print("La accuracy del modelo es: " + str(calcularAccuracy(y_test, y_preds)))

y_test_conf = np.argmax(y_test, axis=1)
y_preds = np.argmax(y_preds, axis=1)
print("La matriz de confusion de las predicciones ha sido: \n", confusion_matrix(y_test_conf, y_preds))
```

FALTAN RESULTADOS CUANDO EJECUTEMOS CON TODAS LAS IMGS

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:

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

RESULTADOS.....

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.....
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.....

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.

4.2. ResNet50

4.3. VGG19

4.4. InceptionV3

4.5. Xception

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

6. Posibles propuestas de mejora