Memoria
Chest X-Ray

José Luis Gómez Antón / jlga10@alu.ua.es
Juan Carlos López Gutiérrez / jclg7@alu.ua.es

Índice

Nivel Básico

Clasificación binaria sobre si existe o no neumonía

En un primer momento empezamos con un modelo de red básico el cual distingue dos clases de radiografía; una radiografía normal y una radiografía con neumonía.
En este momento el modelo de red que tenemos es una red creada con un modelo **Secuencial**, el cual está compuesto de capas convolucionales seguidas cada una de ellas con un filtrado de max pooling. Tras estas convoluciones aplicamos una capa flatten para disponer las diversas matrices en forma de array unidimensional el cual se va reduciendo con varias capas densas hasta quedarnos con un array unidimensional de longitud 2 ya que por el momento contamos con dos clases.

Hicimos una serie de pruebas donde obtuvimos un acc de aproximadamente el 85% con el algoritmo de optimización adam, que en un principio venía con sgd, si usábamos otros como adamax, adadelta, etc. Obteníamos los mismo resultados o peores, por lo que nos quedamos con adam.

Al principio para cambiar la arquitectura de la red metimos 2 capas Dropout antes del maxPooling pero los resultados obtenidos eran los mismos.

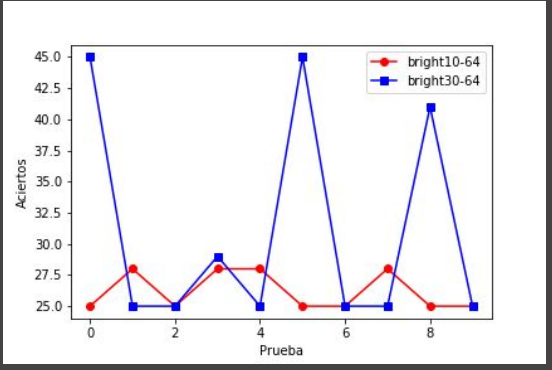
Aplicación de la validación cruzada 10-CV

Para la realización de la validación cruzada 10-CV hemos utilizado Stratified KFold de la librería sklearn.model_selection con un seed para dividir las imágenes siempre partiendo de un mismo punto. De esta forma vamos a poder comparar diferentes implementaciones de modelos entre ellos.

Hemos distribuido el DataSet de la siguiente forma: 80% training & 20% test
Esta distribución la hemos hecho con un split, y una vez hecha la partición hemos utilizado el método kfold.split para dividir nuestro training en training (90%) y validación (10%).

El método de validación cruzada lo hemos usado para poder comparar diferentes modelos.

Por ejemplo: En este caso el primer fold ha podido ser beneficioso, ya que para las imágenes usadas en ese fold han dado un cierta mejoría. Sin embargo, podemos ver que puede haber empeorado para otras particiones al aplicar dicho cambio (Ej: aumentando de brillo).



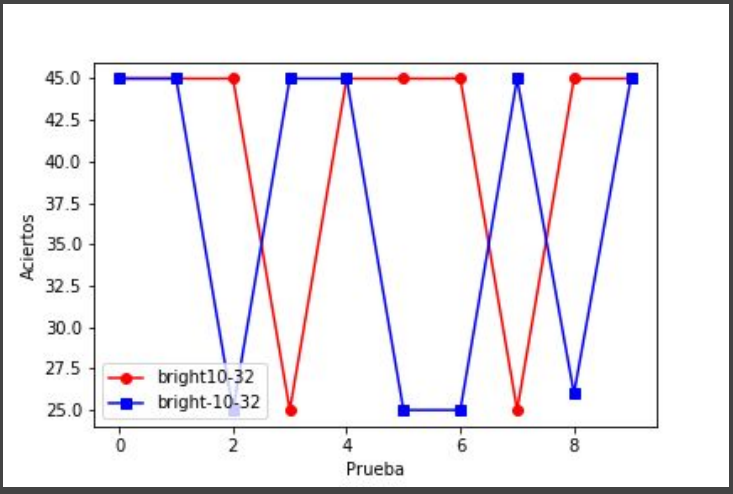
Para poder saber si mejora o no, haces diferentes folds y así poder sacar un test estadístico, como el que se comenta en el siguiente apartado (Wilcoxon).

Un punto a remarcar sobre la validación cruzada es que gracias a StratifiedKFold cuando se hacen las diferentes particiones para poder hacer después el análisis comparativo, estos particionamientos se hacen teniendo en cuenta de qué clase es cada imágen por lo que cada particionado tendrá el mismo porcentaje de ejemplos de cada clase.

Estudio comparativo con test estadísticos (Wilcoxon)

Para el estudio comparativo con test estadísticos hemos usados el test de Wilcoxon el cual pasando 2 arrays con los resultados del Cross Validation K Fold nos da un valor p-value el cual marca con qué porcentaje la red X es mejor a la red Y.
Para poder ver de forma visual este experimento también hemos generado para cada estudio una gráfica donde podemos verlos representados.

Ejemplo: Bright10-32-bright vs Bright-10-32
En este ejemplo vamos a comprar una red entrenada con un aumentado de datos en el cual se ha podido aumentar en alguna imágen el brillo hasta un 10% y son imágenes de 32x32.
La estamos comparando con otra red entrenada con imágenes en las que puede verse reducido su brillo un 10% y son de tamaño 32x32 también



En esta gráfica podemos ver cómo las redes más o menos se comportan.
Ambas funcionan bastante mal. El resultado del test Wilcoxon ha sido un p-value de 0.33 por lo que la red está un 67% segura de que la red bright10-32 es mejor a la segunda, este test no nos vale, ya que no es fiable.

A continuación veremos otro que sí lo sea.

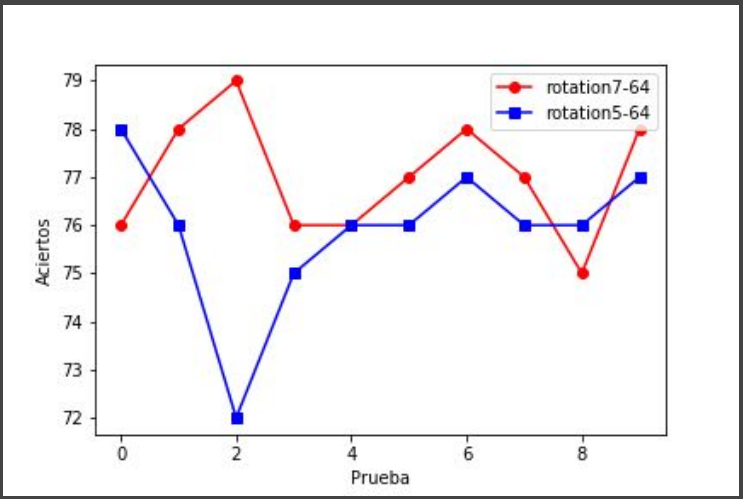
Ejemplo 2: Rotation7-64 vs Rotation5-64.png

La primera red ha sido entrenada con una posible rotación de 7 grados, mientras que la segunda ha sido de 5 grados, ambas redes con imágenes de 64x64.

Wilcoxon signed rank test with continuity correction

data: c(76, 78, 79, 76, 76, 77, 78, 77, 75, 78) and c(78, 77, 77, 76, 76, 74, 75, 75, 75, 77)
V = 24, p-value = 0.05226
alternative hypothesis: true location shift is greater than 0

p-value: 0.05



En este caso el test de Wilcoxon nos ha dado un p-value de 0.05 por lo que está un 95% seguro de que la red con una rotación de 7 grados es mejor a la de 5.

En este caso un 95% es un dato fiable, ya que la probabilidad es alta, por lo que podríamos intentar hacer más pruebas cambiando el grado de rotación a ver si nuestra red mejora. Tiene cierta lógica, ya que una imagen no siempre va a estar recta y más cuando estamos hablando de radiografías o fotos de radiografías.

Técnicas de aumentado de datos: ImagePreprocesing

Como técnica de aumentado de datos hemos usado Image Preprocessing de Keras al cual le podemos pasar diferentes parámetros para modificar imágenes. Los que hemos usado para generar nuevas imágenes son:

- Zoom_range: Al cual le hemos añadido un rango de zoom.
- Brightness_range: Al cual le pasamos un rango para oscurecer o abrillantar la imagen.
- Rotation_range: Le pasamos un Int con los grados que puede torcerse la imagen.
- Rescale: Para rescalar imágenes a un tamaño definido.
- Horizontal_flip: Giro en eje horizontal.
  ...
- Vertical_flip: Giro en eje vertical.

Para poder usar este datagen, hemos tenido que crear un “fit_generator” al que se le pasa como parámetro (el datagen, validation_data, steps_per_epoach, epoach, class_weight, callbacks)

- Datagen: Es nuestro generador de imágenes al cual le datos los datos de entrenamiento y su Output en formato categorical para poder compararlo con la salida de la red, el tamaño del batch_size y una seed para que genere las mismas transformaciones en el crossbar validation.
- Validation_data: Tupla de Datos de Validación y su salida en formato Categorical para poder comparar con el output.
- Steps_per_epoach: Número de iteraciones del batch antes de que se dé por finalizada dicha época.
- Epoach: Número de épocas.
- Class_weight: Es un parámetro que hemos añadido, ya que para equilibrar las imágenes de las clases hemos dado más peso a las imágenes del conjunto que menos tenía.
- Callbacks: Hemos añadido un early_stopping de forma que cuando pasan 10 épocas y el error de validación no ha bajado, damos por sentado que ya no estamos entrenando, por lo que finalizamos el entrenamiento.

Nivel Medio

Clasificación ternaria (No enfermo,Bacteriana, Vírica)

Para poder clasificar las imágenes según si el paciente está sano, tiene una neumonía bacteriana o una neumonía vírica, lo primero que cambiamos fue la carga de las imágenes distinguiendo entre diferentes tipos de neumonía según el título de las propias imágenes.

La siguiente modificación necesaria es hacer que la red nos de como salida un array unidimensional de tres elementos representando cada uno de los tres elementos a una clase concreta dentro de nuestro dataSet, normal, neumonía bacteriana o neumonía vírica respectivamente.
En este array lo que queremos es que se nos dé como resultado en cada una de las posiciones 0 si no pertenece a la clase correspondiente o 1 si pertenece a la misma, para lo cual aplicamos en la última capa una función sigmoide la cual nos ayuda a forzar a que tome una decisión.

Ajustes combinados: Escalado de imágenes, número de capas…

El escalado de imágenes lo hemos realizado con las mismas pruebas empezando por 32 y terminando por 128 [(32x32) , (64x64) , (128x128) , (256x256) , (512x512)] . Hemos ejecutado algunas pruebas con 256 y 512, pero hemos tenido que aumentar la RAM de Google Collab ya que llenábamos la memoria, como estas pruebas no han salido muy bien las hemos omitido (Fallos Collab, etc).

En la arquitectura de la red hemos hecho algunos cambios desde su estado inicial, y hemos utilizado diferentes algoritmos de optimización.

A continuación vamos a detallar los tipos de arquitecturas usadas para poder compararlas:

Arquitectura 1 para predicción ternaria

```
def cnn_model(input_shape):
    #
    # Neural Network Structure
    #

    model_input = Input(shape = input_shape, name='model_input')

    model = Conv2D(6, (8, 8), activation="relu")(model_input)
    model = MaxPooling2D(pool_size=(2, 2))(model)

    model = Conv2D(16, (5, 5), activation="relu")(model)
    model = MaxPooling2D(pool_size=(2, 2))(model)

    model = Conv2D(32, (3, 3), activation="relu")(model)
    model = MaxPooling2D(pool_size=(2, 2),strides=(2,2))(model)

    model = Flatten()(model)

    model = Dense(120, activation="relu")(model)

    model = Dense(80, activation="relu")(model)

      model_output = Dense(params["nb_classes"], activation='softmax',
name='model_output')(model)

    model = Model(inputs=model_input, outputs=model_output)

    return model
```

Arquitectura 2 para predicción ternaria

```
def cnn_model(input_shape):
    #
    # Neural Network Structure
    #

    model_input = Input(shape = input_shape, name='model_input')

    model = Conv2D(32, (5, 5), activation="relu")(model_input)
    model = MaxPooling2D(pool_size=(2, 2))(model)

    model = Conv2D(64, (5, 5), activation="relu")(model)
    model = MaxPooling2D(pool_size=(2, 2))(model)

    model = Conv2D(64, (5, 5), activation="relu",
padding='same')(model)
    model = MaxPooling2D(pool_size=(2, 2),strides=(2,2))(model)

    model = Flatten()(model)

    model = Dense(120, activation="relu")(model)

    model = Dense(80, activation="relu")(model)

    model_output = Dense(params["nb_classes"], activation='softmax',
name='model_output')(model)

    model = Model(inputs=model_input, outputs=model_output)

    return model
```

Arquitectura 3 para predicción ternaria

```
def cnn_model(input_shape):
    #
    # Neural Network Structure
    #
```

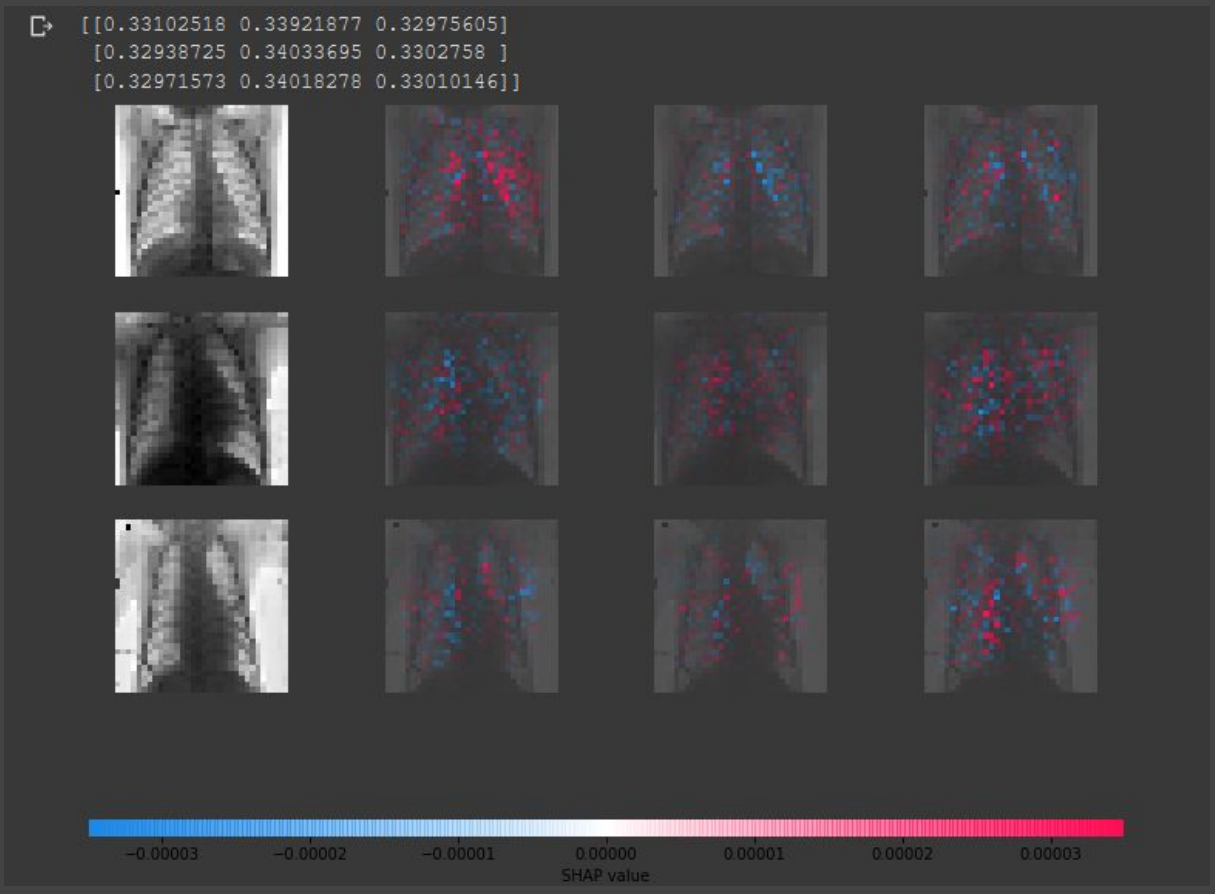
```
    model_input = Input(shape = input_shape, name='model_input')

    model = Conv2D(32, (5, 5), activation="relu")(model_input)
    model = MaxPooling2D(pool_size=(2, 2))(model)

    model = Conv2D(64, (5, 5), activation="relu")(model)
    model = MaxPooling2D(pool_size=(2, 2))(model)

    model = Flatten()(model)

    model = Dense(120, activation="relu")(model)

    model = Dense(80, activation="relu")(model)

    model_output = Dense(params["nb_classes"], activation='softmax',
name='model_output')(model)

    model = Model(inputs=model_input, outputs=model_output)

    return model
```

Arquitectura	SGD	ADAM	ADADELTA
1	loss: 1.09 acc: 0.46 Area under Roc: 0.5 acc: 0.75, 0.77, 0.78, 0.76, 0.74, 0.72, 0.77, 0.77, 0.77, 0.75	0.54, 0.77 0.8277 0.77, 0.77, 0.75, 0.75,0.76, 0.76, 0.76, 0.77, 0.76, 0.77	-
2	loss: 0.55 acc: 0.78 Area under Roc: 0.8308 acc: 0.78, 0.76, 0.77, 0.71, 0.77, 0.78, 0.75, 0.76, 0.78, 0.77	1.36 0.76 0.8077 0.77, 0.78,0.76,0.78,0.78, 0.74,0.77,0.77,0.77, 0.77	-

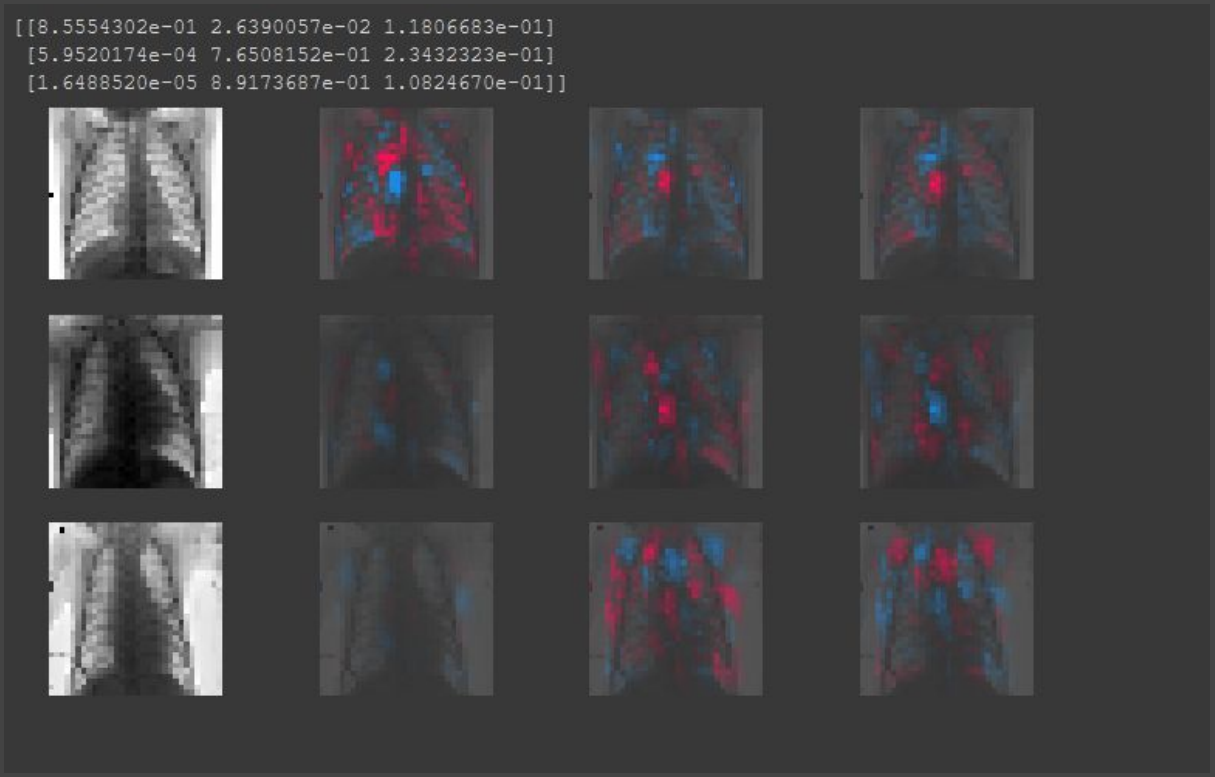
3	1.18 0.79 0.8203 0.75, 0.77, 0.78, 0.76, 0.78, 0.77, 0.78, 0.78, 0.78, 0.79	1.18 0.79 0.8203 0.78,0.77, 0.78, 0.76, 0.76, 0.78, 0.76, 0.77, 0.77, 0.78	0.97 0.77 0.8212 0.78, 0.76, 0.74, 0.76, 0.72, 0.78, 0.73, 0.77, 0.77, 0.77

A continuación vamos a ver los diagramas deep de shap de las arquitecturas:

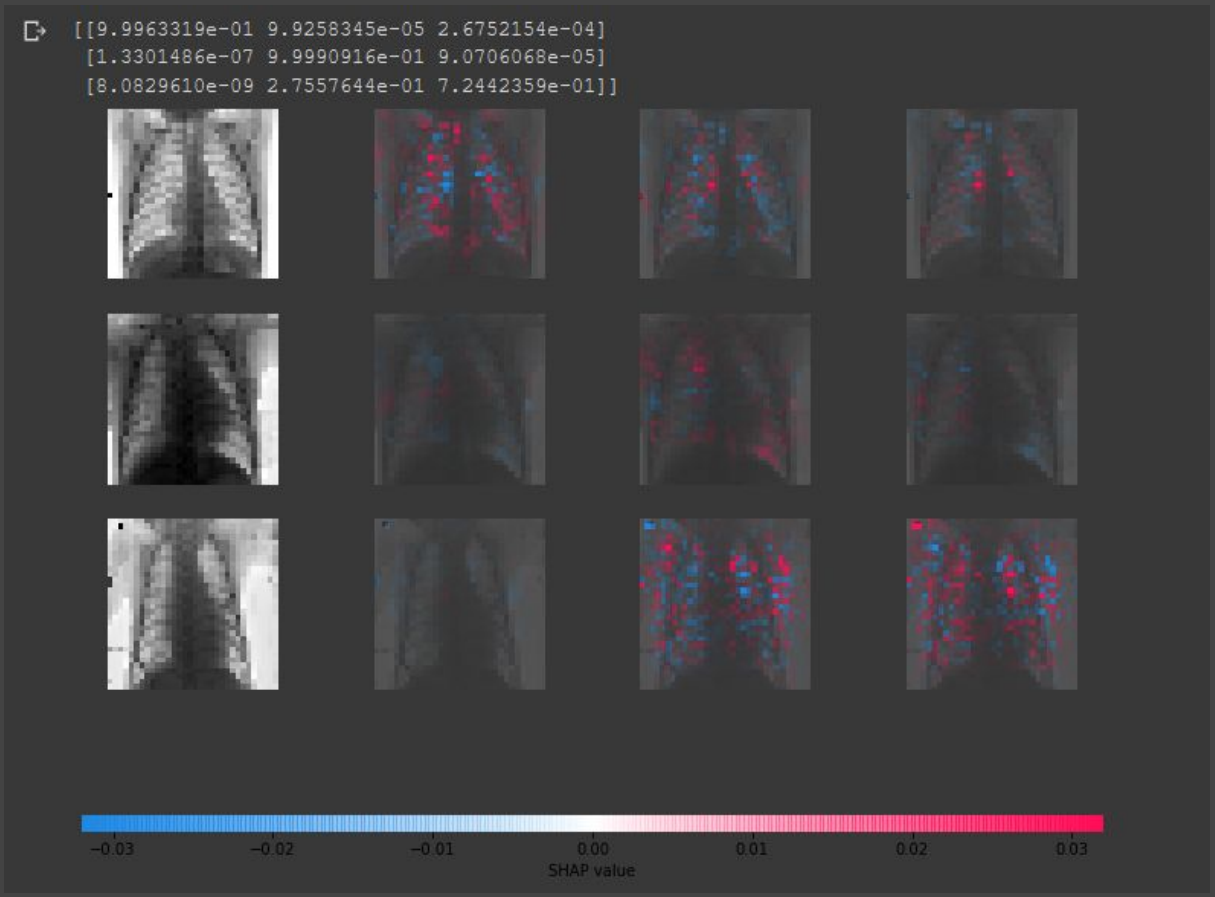
Arquitectura 1 con SGD predicción ternaria



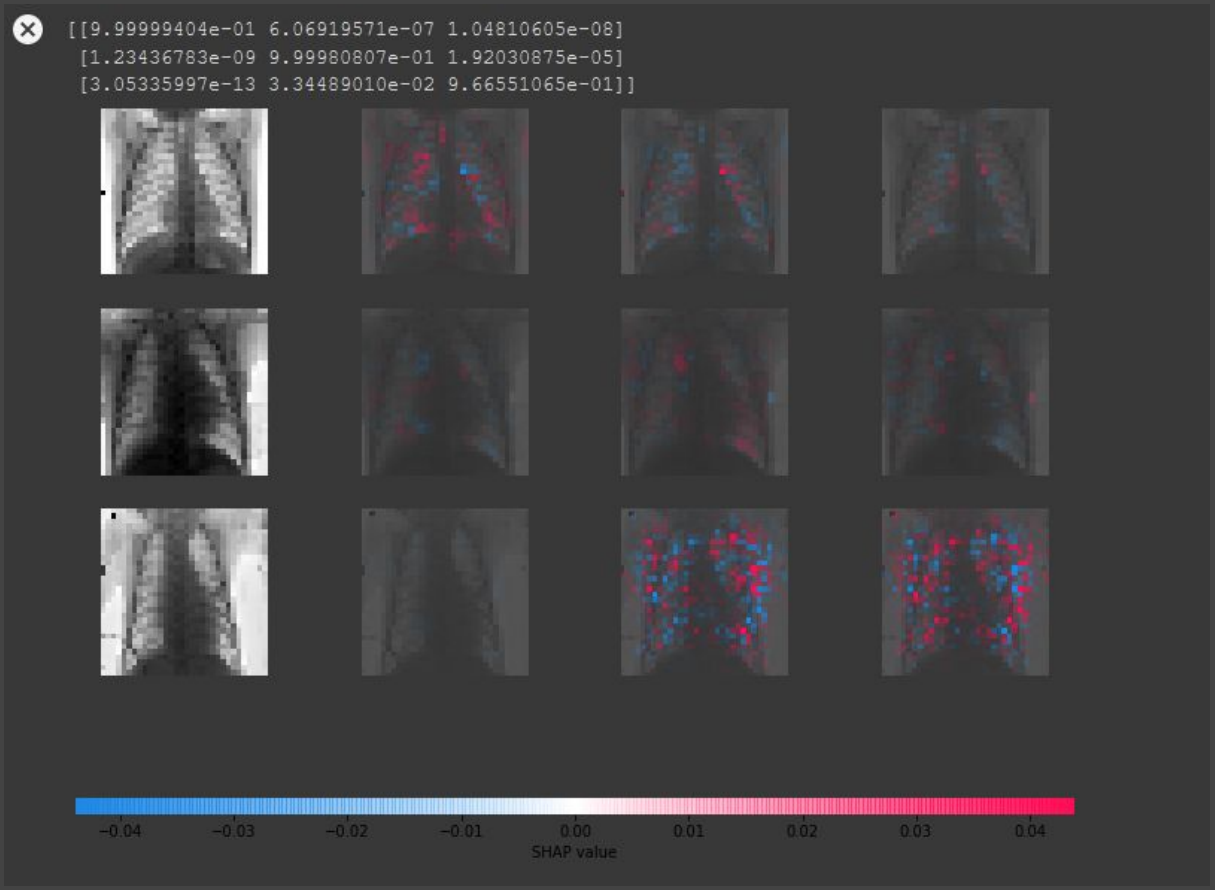
Arquitectura 2 con SGD predicción ternaria



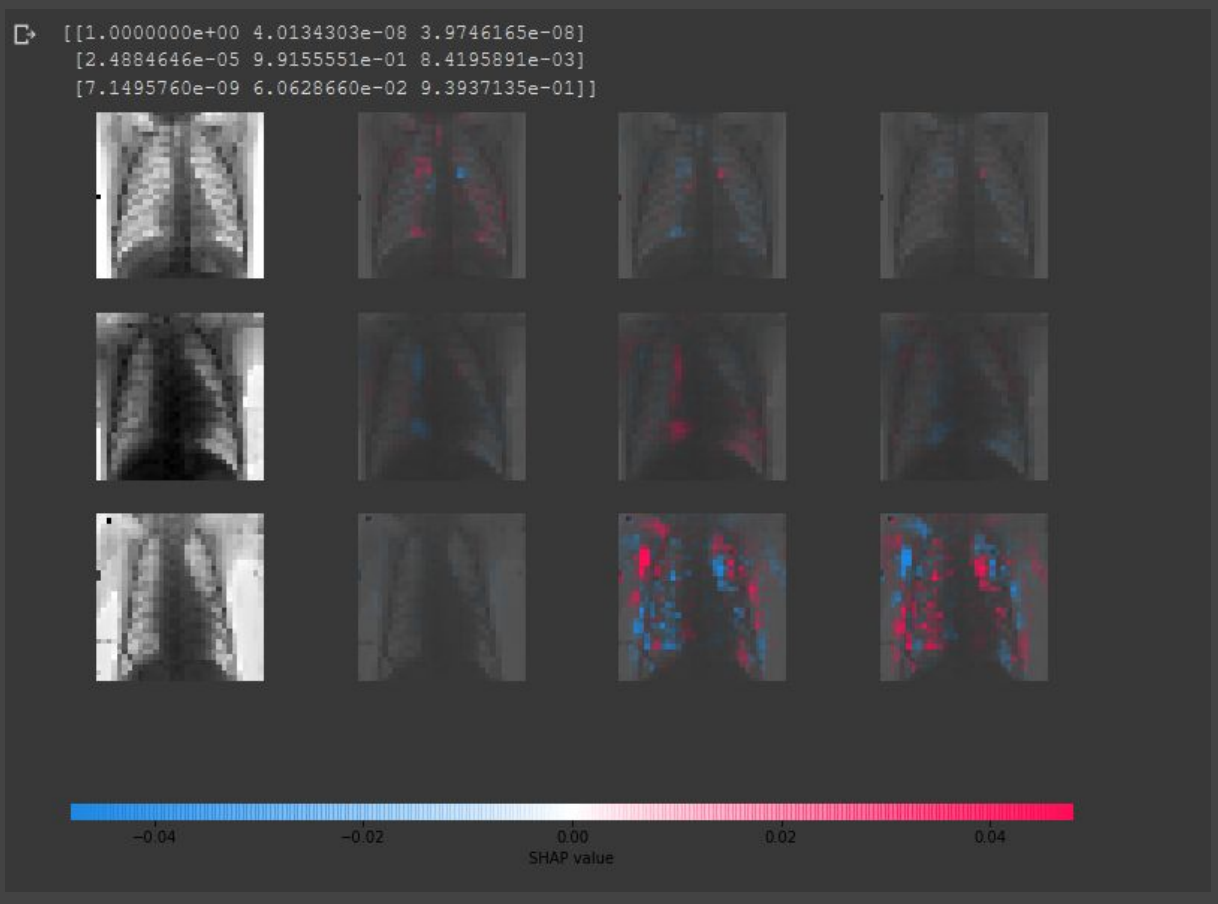
Arquitectura 2 con Adam predicción ternaria



Arquitectura SGD predicción ternaria

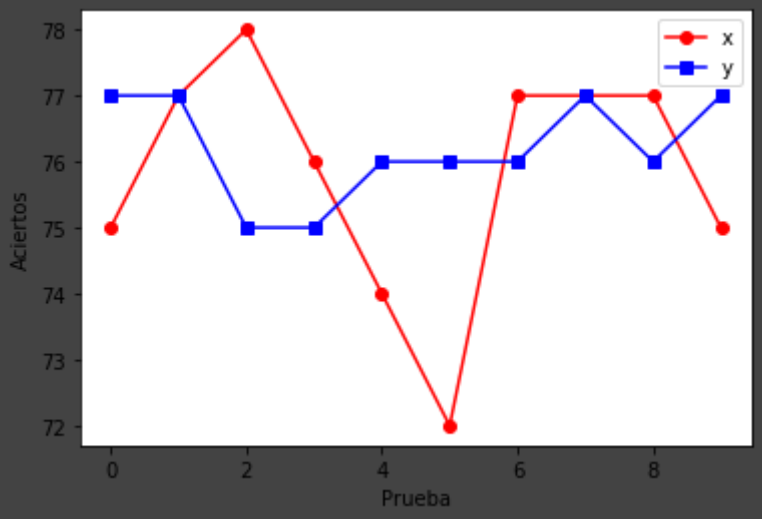


Arquitectura Nueva2 con Adadelta predicción ternaria con EarlyStopping



Para finalizar este apartado vamos a comprar las arquitectura entre ellas, para ver cual de ellas es mejor aplicando tests de Wilcoxon.

Resultado de test de Wilcoxon:



Comparación Arquitectura 1 SGD vs Adam

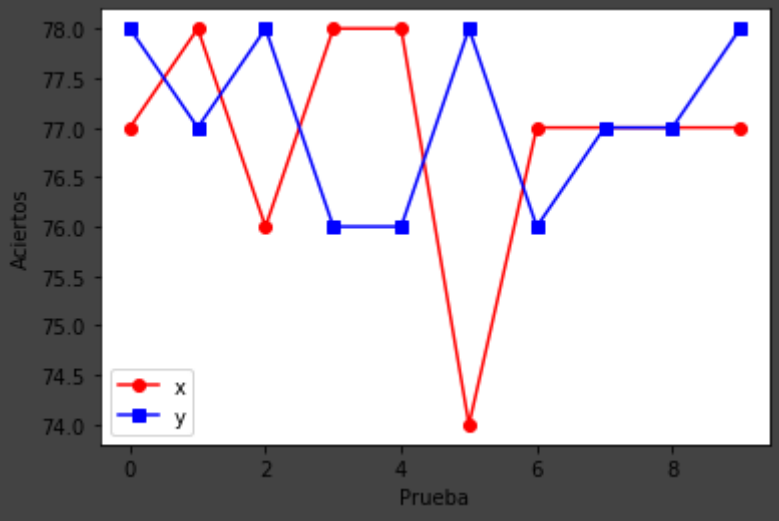
Resultado completo del test de Wilcoxon

Wilcoxon signed rank test with continuity correction

data: c(75, 77, 78, 76, 74, 72, 77, 77, 77, 75) and c(77, 77, 75, 75, 76, 76, 76, 77, 76, 77)
V = 13, p-value = 0.7817
alternative hypothesis: true location shift is greater than 0

p-value: 0.78; Puesto que el p-value es de 78 podemos decir que sgd es mejor que adam con un 22% de fiabilidad.

Arquitectura 2 Adam vs arquitectura 3 Adam



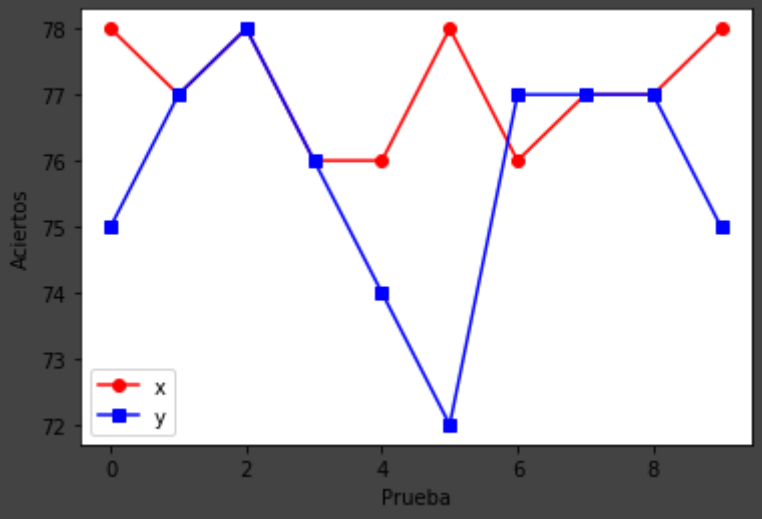
Resultado completo del test de Wilcoxon

Wilcoxon signed rank test with continuity correction

data: c(77, 78, 76, 78, 78, 74, 77, 77, 77, 77) and c(78, 77, 78, 76, 76, 78, 76, 77, 77, 78)
V = 17, p-value = 0.5846
alternative hypothesis: true location shift is greater than 0

p-value: 0.58, puesto que el p-value es de 0.58 podemos saber que adam arc 2 es mejor a adam arc 3 con un 62% de fiabilidad

Arquitectura 3 Adam vs Arquitectura 1 SGD



Resultado completo del test de Wilcoxon

Wilcoxon signed rank test with continuity correction

data: c(78, 77, 78, 76, 76, 78, 76, 77, 77, 78) and c(75, 77, 78, 76, 74, 72, 77, 77, 77, 75)
V = 14, p-value = 0.05203
alternative hypothesis: true location shift is greater than 0

p-value: 0.05. En este caso puesto que el p-value es del 0.05 podemos decir que Arquitectura 3 Adam es mejor que Arquitectura 1 SGD con un 95% de fiabilidad. Al ser una fiabilidad tan alta si podríamos decir que es mejor que la otra en los casos anteriores no tenemos una fiabilidad buena como para asegurarlo.

Escalado progresivo: Progressive Resizing

El escalado progresivo es una técnica que puede ser de mucha ayuda durante el entrenamiento y fases de optimización de una red neuronal convolucional.
Consiste en crear una red convolucional de un tamaño definido por ejemplo 32x32, una vez que ya hemos entrenado la red con los datos de entrada 32x32 vamos modificando nuestro modelo para que pueda entrenar con otro tamaño de imagen, concretamente el doble, 64x64. Entonces cargamos nuestra red previamente entrenada y seguimos entrenando con imágenes de dicho tamaño. Y repetimos el proceso duplicando el tamaño del tamaño de las imágenes de entrada.

Para terminar de realizar esta técnica, debemos complementarla con otra que se llama transfer learning esta técnica nos permite reusar capas y pesos de las capas anteriores cuando estamos construyendo nuevas a partir de ellas. De esta forma cogemos toda la parte de la red anterior menos el input_shape anterior y le ponemos el de la siguiente capa que queremos. Por supuesto como vamos a cargar los pesos que debemos guardar en un archivo (.h5) nuestra red podrá diferenciar entre diferentes tipos de imágenes de diferentes tamaños.

Esta parte del transfer learning ha sido la que nos ha fallado a nosotros, ya que conseguimos crear un modelo y guardarlo, pero no hemos sido capaces de generar a partir del anterior otro nuevo cambiando su input_shape. Los ejemplos encontrados no han sido de mucha ayuda, ya que todos trabajan sobre redes ya preentrenadas generando modelos de manera secuencial y no lo hacen desde cero lo que perjudica al entendimiento.

Keras Functional API

El API funcional de keras nos permite hacer estructuras más complejas como modelos con múltiples salidas o múltiples entradas.
Es interesante modificar el modelo actual creado de manera secuencial para poder agregar futuras mejoras del modelo que lo hagan más complejo.

SHAP (SHapley Additive exPlanations)

SHAP es un método de aproximación unificado el cual explica la salida de cualquier modelo de machine learning.
Para poder explicar la salida de nuestra red y poder entender por qué elige esta decisión hemos usado DeepExplainer (DeepSHAP), es una aproximación rápida hecha por los valores de SHAP.

Como podemos ver en la siguiente imagen (Figura 1):

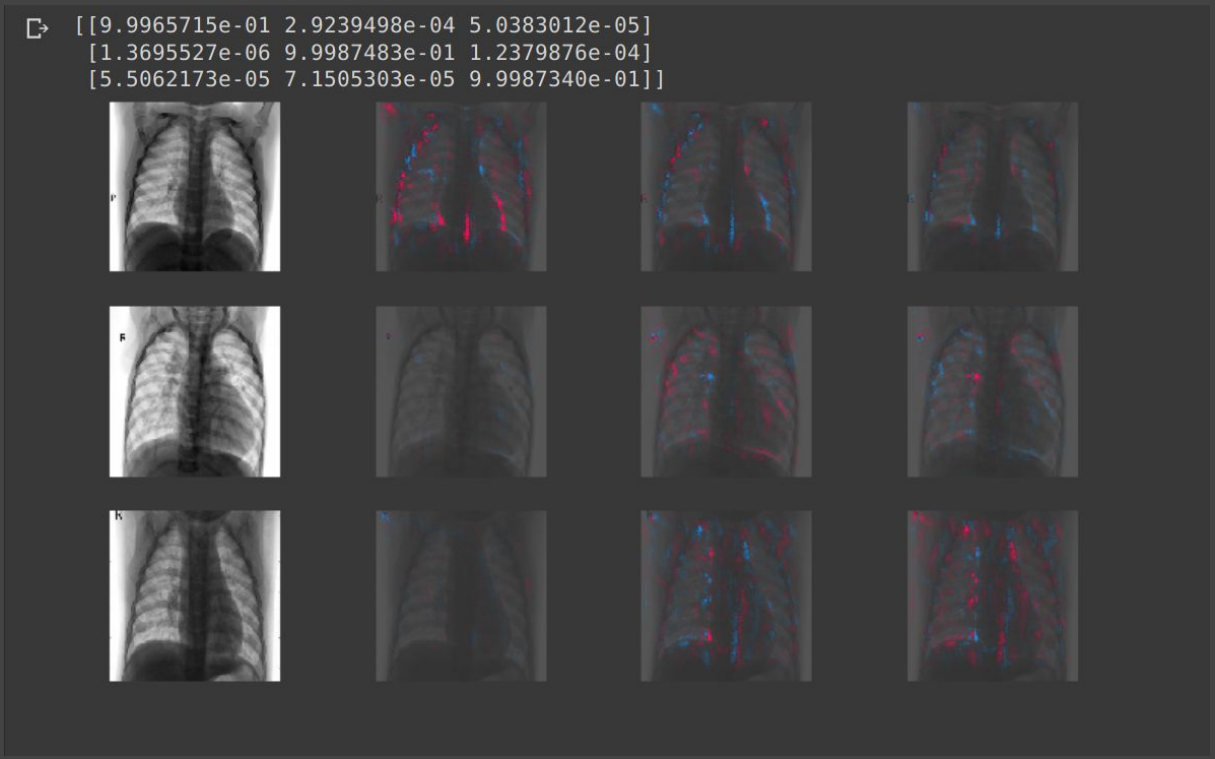


Figura 1

En la figura 1 podemos ver como las imágenes están coloreadas por 2 colores: Rojo y Azul.

La parte coloreada de Rojo nos indica la parte de la imagen en la que se está fijando la red para decir que la imagen pertenece a esa clase. Siendo la primera columna “Normal”, segunda columna “Neumonía Bacteriana” y la tercera “Neumonía Vírica”. Mientras que la parte Azul es en la que se fija para decirnos que la imagen no pertenece a esa clase.

Como podemos ver en la imagen, para la imagen número 1, elegimos la clase “Normal”, para la 2 elegimos la clase “Bacteriana” y para la 3 “Vírica”. En la parte superior de la imagen tenemos los valores que nos indican la predicción que ha tenido la red con un rango entre [0 - 1] , siendo 1 un 100% de fiabilidad en su elección.

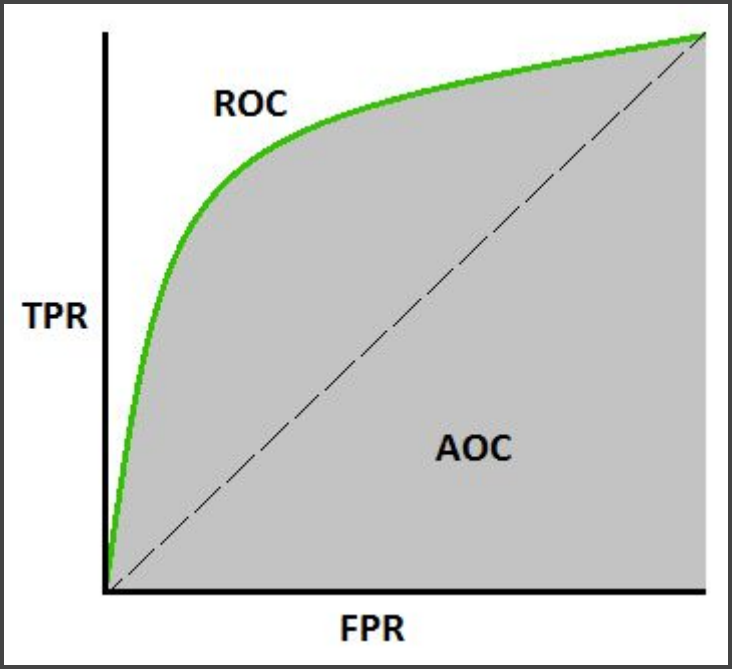
Test estadístico: Área Roc

Una curva ROC es un gráfico que muestra el rendimiento de un modelo de clasificación en todos los umbrales de clasificación.

AUC o test under roc: Mide toda el área bidimensional por debajo de la curva ROC completa. Proporciona una medición agregada del rendimiento en todos los umbrales de clasificación posibles. Una forma de interpretar el AUC es como la probabilidad de que el modelo clasifique un ejemplo positivo aleatorio más alto que un ejemplo negativo aleatorio.

El AUC representa la probabilidad de que un ejemplo aleatorio positivo se posicione a la derecha de un ejemplo aleatorio negativo.

El AUC oscila en valor del 0 al 1. Un modelo cuyas predicciones son un 100% incorrectas tiene un AUC de 0.0; otro cuyas predicciones son un 100% correctas tiene un AUC de 1.0.



El AUC es conveniente por las dos razones siguientes:

- El AUC es **invariable con respecto a la escala**. Mide qué tan bien se clasifican las predicciones, en lugar de sus valores absolutos.
- El AUC es **invariable con respecto al umbral de clasificación**. Mide la calidad de las predicciones del modelo, sin tener en cuenta qué umbral de clasificación se elige.

Sin embargo, estas dos razones tienen algunas advertencias, que pueden limitar la utilidad del AUC en determinados casos:

- **La invariabilidad de escala no siempre es conveniente.** Por ejemplo, en algunas ocasiones, realmente necesitamos resultados de probabilidad bien calibrados, y el AUC no nos indicará eso.

- **La invariabilidad del umbral de clasificación no siempre es conveniente.** En los casos en que hay amplias discrepancias en las consecuencias de los falsos negativos frente a los falsos positivos, es posible que sea fundamental minimizar un tipo de error de clasificación. Por ejemplo, al realizar la detección de spam de correo electrónico, es probable que quieras priorizar la minimización de los falsos positivos (aunque eso resulte en un aumento significativo de los falsos negativos). El AUC no es una métrica útil para este tipo de optimización.

Nivel Avanzado

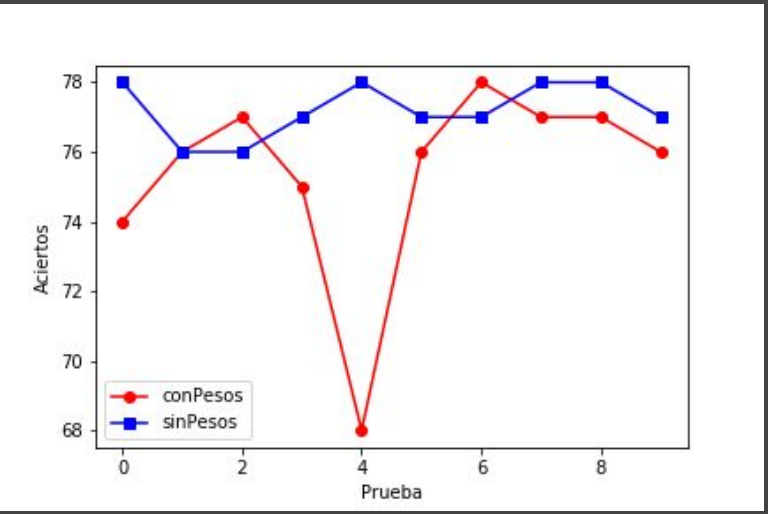
Equilibrar número de ejemplos entre clases

Una vez que se aplicó el cambio de la estructura de la red para que aprendiera a distinguir entre las tres categorías que ya hemos mencionado, uno de los problemas que nos encontramos fue que los ejemplos entre las tres clases no estaban equilibrados lo cual hacía que la red aprendiera de una manera errónea.

El problema que esto causa se puede entender con un ejemplo sencillo:
Si tenemos un dataSet el cual contiene un 20% de imágenes de una clase y un 80% de otra, es probable que la red cuando esté tratando de aprender a distinguir entre las clases siempre diga que se tratan de la segunda clase ya que es una tasa de acierto de un 80% decir siempre que la clase correcta es la segunda pero realmente está aprendiendo de una manera errónea.

El primer cambio que se introdujo fué cargar el mismo número de imágenes de cada una de las clases y así nos evitamos el desbalance de ejemplos entre clases. Pero esto no era una buena opción ya que realmente estamos perdiendo información de las clases de las que tenemos más ejemplos.

Y la solución por la que finalmente optamos fue en pasarle a la función de entrenamiento un vector de pesos de cada clase lo cual nos permite que esta diferencia de ejemplos entre imágenes no se aprecie tanto dándole más importancia a las clases de las que menos ejemplos tiene para que las aprenda a distinguir de una manera más equilibrada.



Pero una vez que hicimos la pruebas correspondientes, no sólamente no mejoraba la capacidad de clasificación del modelo una vez entrenado, si no que podemos ver haciendo un test de Wilcoxon que efectivamente, empeora el rendimiento notoriamente ya que los resultados obtenidos son los siguientes:

data: c(74, 76, 77, 75, 68, 76, 78, 77, 77, 76) and c(78, 76, 76, 77, 78, 77, 77, 78, 78, 77)
V = 7, p-value = 0.9748

alternative hypothesis: true location shift is greater than 0

p-value: 0.97

Por lo que podemos decir con una seguridad muy alta que teniendo en cuenta el volumen de los ejemplos de cada clase respecto al volumen total de los ejemplos para intentar equilibrar el aprendizaje del modelo no es una buena estrategia ya que empeora los resultados de las predicciones del modelo entrenado.

Aportaciones propias

Lectura de parámetros desde un json

Para poder hacer diversas pruebas de una manera sencilla una de las implementaciones que se llevaron a cabo fue la de poder leer de un archivo de configuración en formato json los parámetros de los cuales queríamos poder llegar a hacer pruebas entre los cuales se encuentran las dimensiones con las que vamos a cargar las imágenes para entrenar el modelo, los parámetros que le queramos aplicar en el aumentado (brillo, rotación, etc), el tamaño de batch, número de épocas y demás.
Un ejemplo de fichero JSON apto es el siguiente:

Json para añadir un 10% de brillo en Imágenes de Tamaño 32x32

```
{
  "datagenParams" : {
        "brightness_range" : [1, 1.1]
  }
}
```

Json para añadir Zoom desde [0.9, 1.2], por lo que tenemos un rango de zoom desde -10% que sería alejarse y un 20% de aumento.

```
{
  "datagenParams" : {
        "zoom_range" : [0.9, 1.2]
  }
}
```

Automatización de Test Wilcoxon

Una vez realizada la lectura de parámetros desde un json, hemos realizado una automatización de forma que si añadimos diversos jsons para realizar diferentes pruebas de aumentados de datos, etc . Después de su ejecución obtenemos los Test de Wilcoxon y unas gráficas visuales de cómo han ido las pruebas comparándolas 2 a 2 todas con todas (Excepto

con sigo misma, ya que no tendría sentido comprar la prueba 1 con la 1). De esta forma podemos ver que cambios han beneficiado la predicción de la red y podemos hacer más pruebas añadiendo estos cambios.

Callbacks

Otra de las mejoras añadidas han sido un par de operaciones a realizar en los callbacks, los que hemos utilizado han sido:

- **CSVLogger**
  Añadir CSVLogger nos ha permitido ir guardando en ficheros la información que vamos obteniendo del proceso de entrenamiento (accuracity, loss, accuracity de validación y loss de validación) para posteriormente poder hacer uso del mismo para hacer pruebas concretas que no hayamos hecho en el momento como podrían ser comparaciones entre modelos que no hayamos hecho en el momento, revisión del correcto funcionamiento del EarlyStopping, entre otros.

- **EarlyStopping**
  Tor un lado nos ha sido de gran utilidad para no tener un entrenamiento innecesario ya que en el momento en el que el modelo ya no mejora distinguiendo los ejemplos designados para validación es innecesario a la par que contraproducente ya que nop va a mejorar más y se puede estar haciendo overfitting a causa del exceso de épocas y por otro lado, que no deja de ser importante es que ha ayudado al coste temporal ya que en más de una ocasión termina antes de las 50 épocas que tenemos como margen lo cual es más notorio cuando hemos hecho pruebas con por ejemplo imágenes en grandes resoluciones que el tiempo de ejecución en cada época empieza a ser largo y hace más tedioso el proceso.

Pruebas

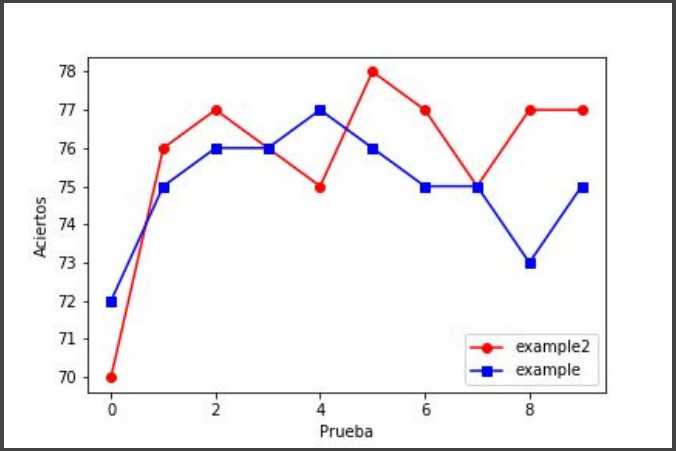
Aquí vamos a poner algunos ejemplos de las pruebas que hemos llevado a cabo en la investigación para obtener el mejor clasificador posible.
Como bien hemos estado comentando hemos estado haciendo diversas pruebas jugando con el aumentado de datos, el tamaño de las imágenes de entrada y demás.

Tamaño de las imágenes de entrada

Para ver si con aumentando la dimensión de las imágenes de entrada se puede mejorar la precisión del modelo resultante, una de las comparativas que vamos a ver es si el

clasificador mejora con cualquier dimensión de imagen mayor a 32x32 y con esta prueba obtenemos los siguientes resultados:

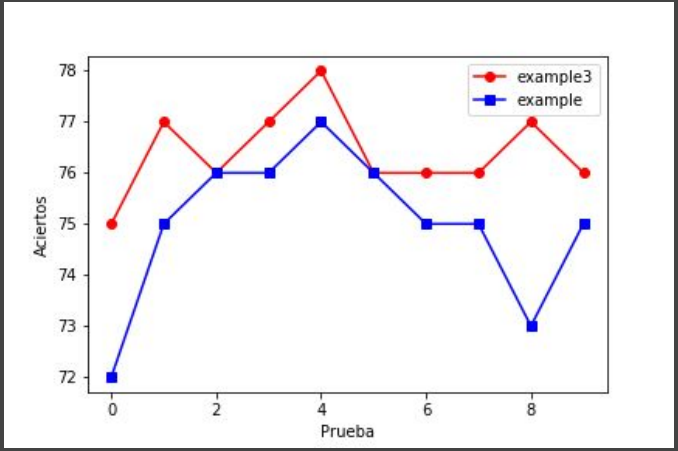
64x64 vs 32x32:



data:
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
and
c(72, 75, 76, 76, 77, 76, 75, 75, 73, 75)
V = 26, p-value = 0.1404
alternative hypothesis: true location shift is greater than 0

p-value: 0.14

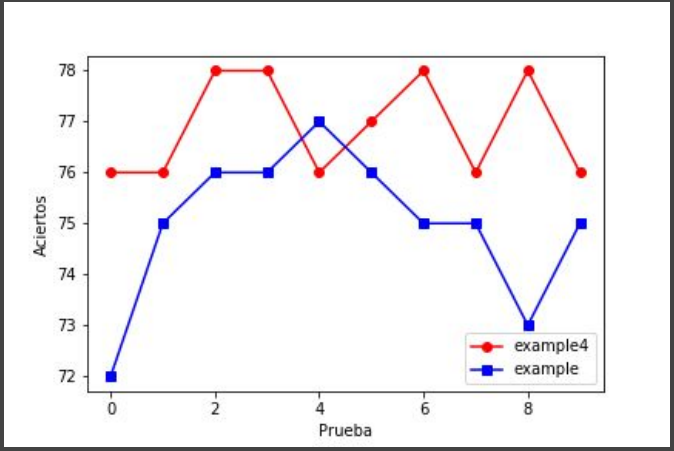
128x128 vs 32x32:



data:
c(75, 77, 76, 77, 78, 76, 76, 76, 77, 76)
and
c(72, 75, 76, 76, 77, 76, 75, 75, 73, 75)
V = 36, p-value = 0.005988
alternative hypothesis: true location shift is greater than 0

p-value: 0.01

256x256 vs 32x32:

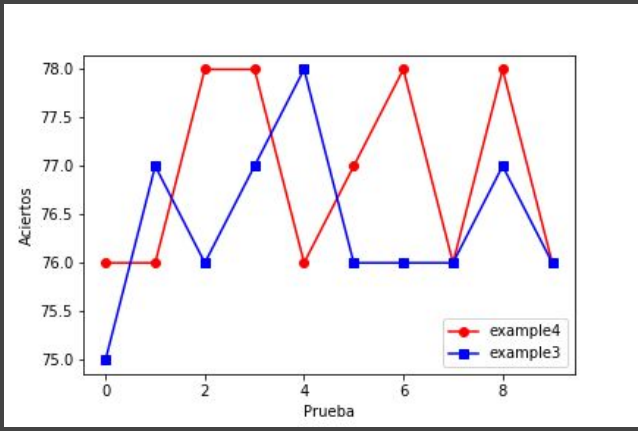


data:
c(76, 76, 78, 78, 76, 77, 78, 76, 78, 76)
and
c(72, 75, 76, 76, 77, 76, 75, 75, 73, 75)
V = 52, p-value = 0.006562
alternative hypothesis: true location shift is greater than 0

p-value: 0.01

Aquí podemos ver como en se puede asegurar que tanto el modelo entrenado con 128x128 (example3) y el modelo entrenado con 256x256 (example4) mejoran con una fiabilidad de un 99% respecto a un modelo entrenado con imágenes de 32x32 (example).

De aquí podemos sacar que estos dos casos son mejores que el caso base del que partíamos, pero como podemos ver si comparamos 128x128 vs 256x256:



data:
c(76, 76, 78, 78, 76, 77, 78, 76, 78, 76)
and
c(75, 77, 76, 77, 78, 76, 76, 76, 77, 76)
V = 26, p-value = 0.1395
alternative hypothesis: true location shift is greater than 0

p-value: 0.14

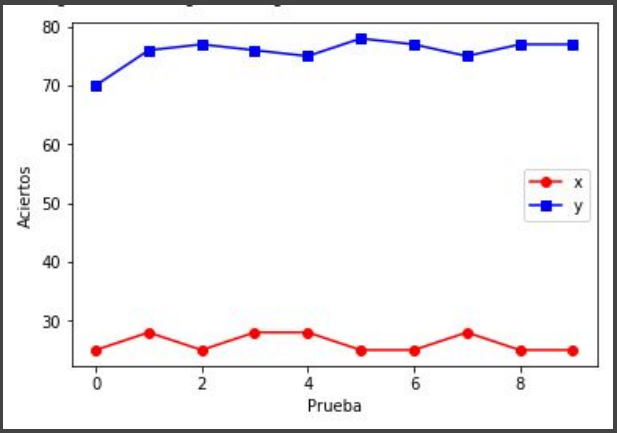
No podemos decir con certeza si el modelo de 256x256 es mejor al de 128x128 ya que la fiabilidad que nos da el test de Wilcoxon es tan solo del 86% lo cual no es un porcentaje lo suficientemente alto como para decir que sí que lo es.

Brillo

Otra de las pruebas que hemos hecho es comprobar si mejora el modelo probando a aplicar aumentado de datos a las imágenes de manera que se aplique en cada época del entrenamiento intentando que el modelo aprenda de una manera más genérica para intentar conseguir una tasa de acierto mayor frente a variaciones en las imágenes de entrada, para esto vamos a mostrar el estudio realizado para comprobar si aplicar un aumentado de datos basado en una modificación del brillo de las imágenes con las que el modelo está aprendiendo es beneficiosa o por el contrario perjudicial y no consigue otra cosa que no sea confundir al modelo.

Para comprobar esto tenemos aquí diversas pruebas:

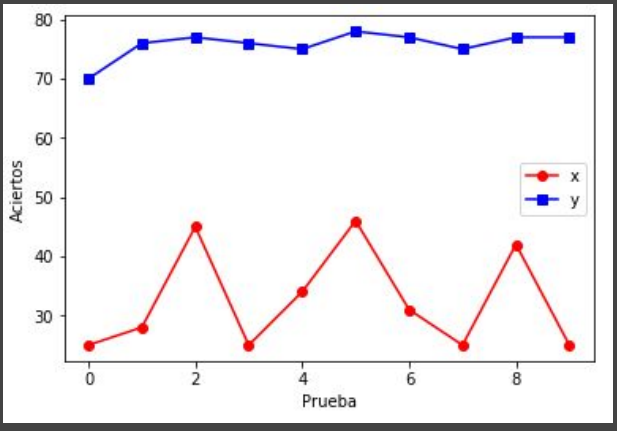
Aumentando hasta un 10% el brillo vs sin aplicar:



data:
c(25, 28, 25, 28, 28, 25, 25, 28, 25, 25)
and
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
V = 0, p-value = 0.998
alternative hypothesis: true location shift is greater than 0

p-value: 1.00

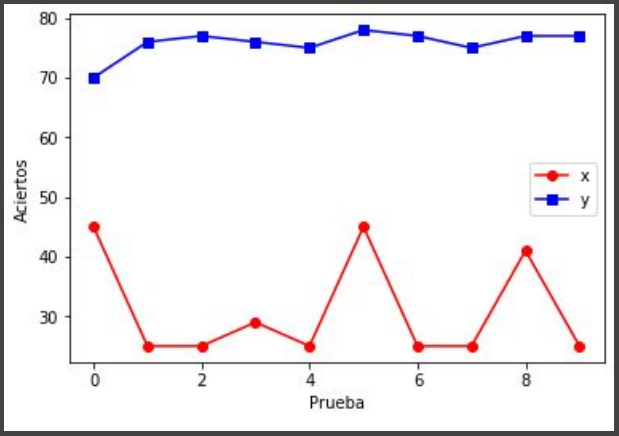
Reduciendo hasta un 10% el brillo vs sin aplicar:



data:
c(25, 28, 45, 25, 34, 46, 31, 25, 42, 25)
and
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
V = 0, p-value = 0.9979
alternative hypothesis: true location shift is greater than 0

p-value: 1.00

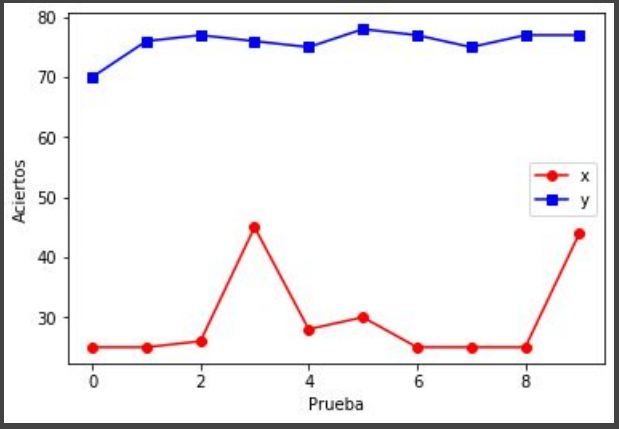
Aumentando hasta un 30% el brillo vs sin aplicar:



data:
c(25, 28, 25, 28, 28, 25, 25, 28, 25, 25)
and
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
V = 0, p-value = 0.998
alternative hypothesis: true location shift is greater than 0

p-value: 1.00

Reduciendo hasta un 30% el brillo vs sin aplicar:



data:
c(25, 28, 45, 25, 34, 46, 31, 25, 42, 25)
and
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
V = 0, p-value = 0.9979
alternative hypothesis: true location shift is greater than 0

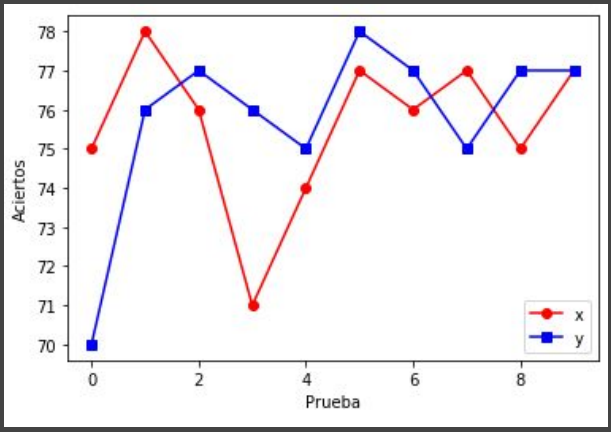
p-value: 1.00

Como podemos observar aplicar un porcentaje aleatorio de brillo dentro de los rangos con los que hemos probado, aplicar brillo al momento de aplicar un aumentado de datos en las imágenes de entrenamiento esto lo que hace es empeorar el modelo haciendo que bajen drásticamente las tasas de acierto que el modelo consigue tras hacer el entrenamiento por lo que podemos asegurar que en este caso aplicar brillo variable para aumentar los datos de entrenamiento no es beneficioso, lo cual teniendo en cuenta que son radiografías las cuales es preciso que se vean en blanco las partes importantes que el especialista tenga que reconocer, tiene sentido que aplicar este aumentado empeora los resultados ya que se puede estar metiendo ruido que empeora el aprendizaje.

Zoom

Para intentar simular que el tamaño del tórax sea más pequeño o más grande la modificación que se nos ha ocurrido probar ha sido aplicar de manera variable un zoom a las imágenes para lo cual hemos obtenido estos resultados:

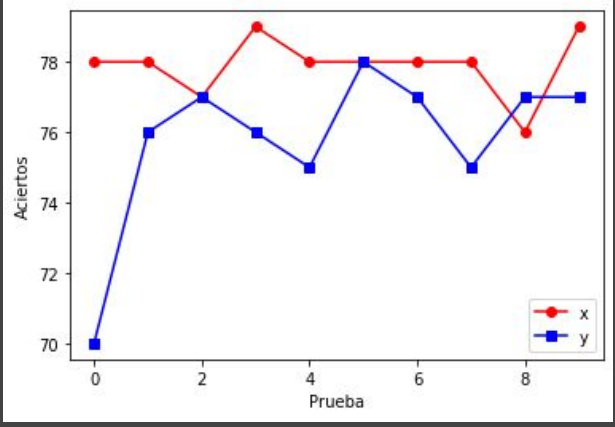
Aplicando zoom de hasta x0.9 vs sin aplicar:



data:
c(75, 78, 76, 71, 74, 77, 76, 77, 75, 77)
and
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
V = 20.5, p-value = 0.618
alternative hypothesis: true location shift is greater than 0

p-value: 0.62

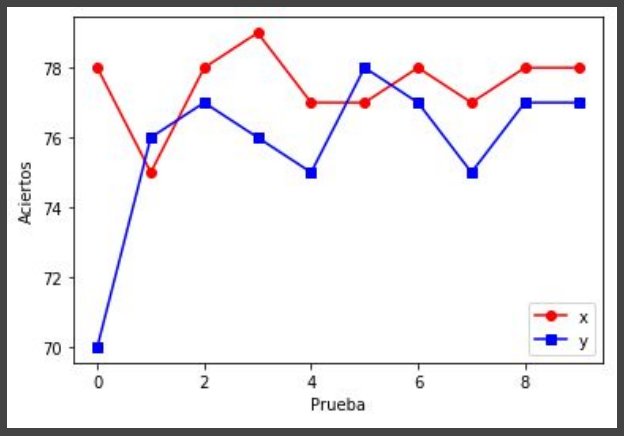
Aplicando zoom desde x0.9 hasta x1.2 vs sin aplicar:



data:
c(78, 78, 77, 79, 78, 78, 78, 78, 76, 79)
and
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
V = 34.5, p-value = 0.012
alternative hypothesis: true location shift is greater than 0

p-value: 0.01

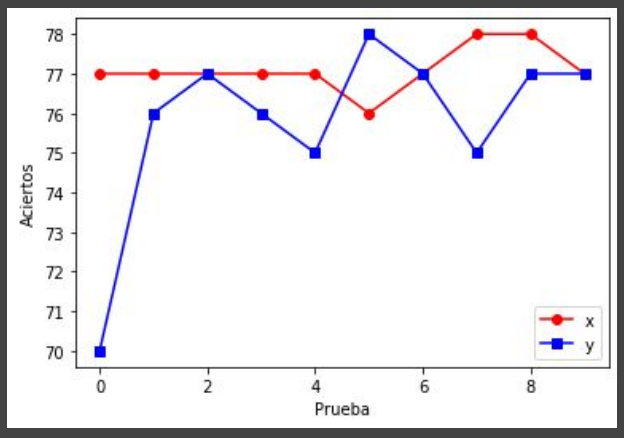
Aplicando zoom desde x0.8 hasta x1.4 vs sin aplicar:



data:
c(78, 75, 78, 79, 77, 77, 78, 77, 78, 78)
and
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
V = 48, p-value = 0.0184
alternative hypothesis: true location shift is greater than 0

p-value: 0.02

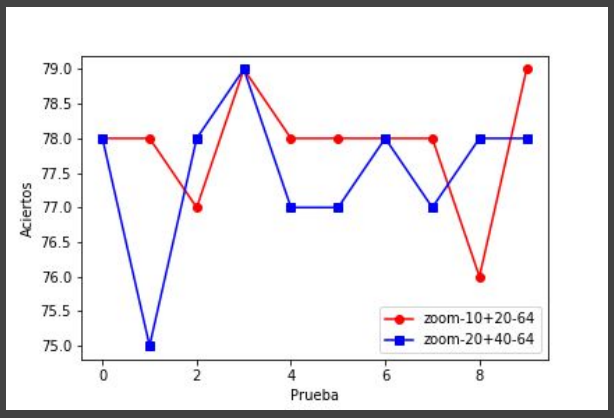
Aplicando zoom de hasta x1.2 vs sin aplicar:



data:
c(77, 77, 77, 77, 77, 76, 77, 78, 78, 77)
and
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
V = 23.5, p-value = 0.06239
alternative hypothesis: true location shift is greater than 0

p-value: 0.06

Como podemos ver las pruebas que mejor resultados son la segunda y la tercera prueba. Con lo que nos disponemos a compararlas entre ellas a ver cual de las dos es mejor:



data:
c(78, 78, 77, 79, 78, 78, 78, 78, 76, 79)
and
c(78, 75, 78, 79, 77, 77, 78, 77, 78, 78)
V = 19, p-value = 0.215
alternative hypothesis: true location shift is greater than 0

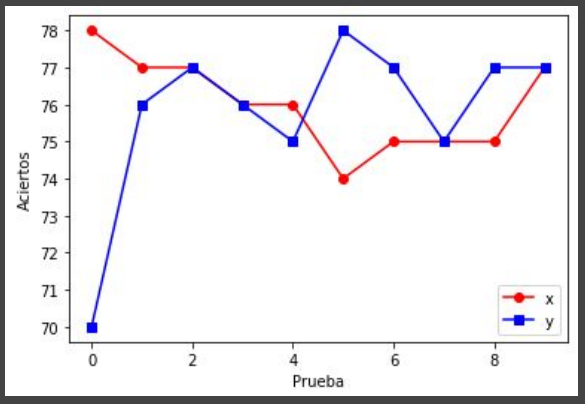
p-value: 0.21

Como podemos apreciar el test de Wilcoxon nos da con un 79% de probabilidad que aplicarle a las imágenes un zoom de entre x0.9 hasta x1.2 da mejores resultados que aplicar un zoom de entre x0.8 hasta x1.4 por lo cual no es un porcentaje lo suficientemente alto como para confirmar que es mejor, lo que sí que podemos asegurar en esta prueba es que aplicar un aumentado de datos aplicando un zoom aleatorio de entre los dos rangos mencionados nos ayuda a que el modelo aprenda mejora a clasificar mejor.

Rotación

También probamos a aplicar rotación simulando que las radiografías pueden estar algo torcidas pero como podemos ver a continuación, no nos ha dado los resultados esperados si no que los resultados obtenidos nos indican que el entrenamiento no se ve apenas afectado por esta perturbación:

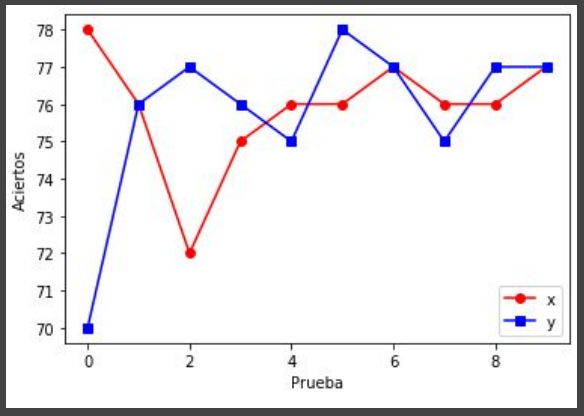
Aplicando hasta 3º de rotación:



data:
c(78, 77, 77, 76, 76, 74, 75, 75, 75, 77)
and
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
V = 9, p-value = 0.6634
alternative hypothesis: true location shift is greater than 0

p-value: 0.66

Aplicando hasta 5º de rotación:



data:
c(78, 76, 72, 75, 76, 76, 77, 76, 76, 77)
and
c(70, 76, 77, 76, 75, 78, 77, 75, 77, 77)
V = 12, p-value = 0.6665
alternative hypothesis: true location shift is greater than 0

p-value: 0.67

Lo que obtenemos con estas pruebas es que no podemos asegurar que mejore ni que empeore, ya que se acerca al 50% lo que sería más acertado sería decir que se mantiene constante frente a este aumentado.

Otras pruebas

Hemos visto unas cuantas pruebas las cuales han sido interesantes y que representan mejor los resultados de la investigación, pero para poder tener en cuenta muchas más combinaciones hicimo muchas más combinaciones las cuales almacenamos en drive, con ayuda de que teníamos la posibilidad de hacer comparativas de una manera muy sencilla gracias a los parámetros almacenados en diversos archivos de configuración, aquí te dejamos un enlace donde tenemos muchas otras combinaciones con sus respectivos test comparativos que nos ha ayudado a identificar las mejores combinaciones en diversas pruebas:

https://drive.google.com/open?id=15U3Jl98ofj8b_kX92zWof_jganzg3msX