

Fundamentos de Deep Learning

Informe final proyecto



UNIVERSIDAD DE ANTIOQUIA

Grupo de trabajo:

CAMILO VÉLEZ PALACIO

JUAN PABLO YARCE ESCOBAR

Profesor:

RAUL RAMOS POLLAN

Universidad de Antioquia

Facultad de ingeniería

Medellín

2024

1. Notebooks

Exploración: Este es el primer notebook a revisar, como primer paso se debe correr la primera celda que va a instalar 3 bibliotecas (si no las teníamos previamente) necesarias para la correcta ejecución del código, estas son Git, Pillow y matplotlib, el siguiente paso es tener la base de datos es por eso que para que cualquier persona pueda hacerlo decidimos disponer al principio de los cuadernos donde fuese necesario un apartado para que replicara estos recursos, sin embargo, al tratarse de tantas imágenes hacen que la data fuera muy pesada, se tuvieron que revisar un par de opciones, la primera que consideramos fue subir la base de datos a google drive y utilizar el comando !wget para obtenerla, sin embargo tuvimos la dificultad de que al usar este comando el archivo que descargaba era un archivo vacío que no correspondía, otra opción que probamos fue gdown, una librería que permite específicamente descargas desde google drive, con esta opción la cosa pintaba bien, hasta que obtuvimos un error que indicaba que estábamos intentando descargar muchos archivos, vimos que gdown tenía un límite de descarga de 50 archivos, mientras que nuestra BD tiene más de 8000 por lo que descartamos esta opción, finalmente llegamos a la opción de subir los datos a un repositorio de github y usar git para descargar los datos, esta opción resultó siendo la elegida pues aunque tuvimos un pequeño contratiempo también con el tamaño de la base de datos, lo logramos solucionar y poder permitirle al usuario que descargue los datos a su entorno, lo hicimos de la siguiente manera:

```
1 # Definir las variables del repositorio de GitHub, la rama y la carpeta que queremos
2 REPO="https://github.com/CamiloVelezP/DeteccionDeMascarillas"
3 BRANCH="main"
4 FOLDER="dataset"
5
6 # Inicializar un nuevo repositorio de Git en una carpeta temporal llamada 'temp_repo'
7 !git init temp_repo
8 %cd temp_repo
9 !git remote add origin $REPO
10 !git config core.sparseCheckout true
11 !echo $FOLDER/ > .git/info/sparse-checkout
12 !git pull origin $BRANCH
13 %cd ..
14
15 # Nos salimos de la carpeta del repositorio temporal y lo eliminamos
16 !mv temp_repo/$FOLDER .
17 !rm -rf temp_repo
```

Esta segunda parte de lo que se encarga es crear un repositorio temporal local y con el código “core.sparseCheckout” permite que solo copie partes específicas del repo, en este caso la carpeta que contiene la bd, luego trae de un repositorio remoto y guarda estos archivos de manera local y elimina el repositorio temporal, esto permite tener los archivos a la mano para replicar los resultados obtenidos.

Una vez tenemos los recursos de la base de datos en el entorno podemos proceder con el notebook, en este notebook vamos a darle un vistazo a algunos detalles de la base de datos, conteo de imágenes por carpeta, tamaño promedio de la imágenes, dimensiones promedio de las imágenes, entre otros, así como las respectivas gráficas que acompañan esta exploración.

Modelo: El notebook "Modelo" detalla el proceso integral de construcción y evaluación de una red neuronal convolucional (CNN) diseñada para clasificar imágenes de personas con mascarillas, sin mascarillas y con mascarillas mal puestas. El flujo de trabajo comienza con la carga de imágenes desde un repositorio de GitHub, organizándolas en las tres categorías mencionadas. Las imágenes se redimensionan a 128x128 píxeles y se convierten a arrays de NumPy, seguido de su normalización dividiendo los valores de los píxeles por 255. Posteriormente, se etiquetan de acuerdo a su categoría correspondiente.

El modelo CNN se construye utilizando la API secuencial de Keras. La arquitectura incluye dos capas convolucionales con 32 y 64 filtros respectivamente, cada una seguida de una capa de max pooling para la extracción eficiente de características. Tras aplanar los datos extraídos, se utilizan dos capas densas con técnicas de dropout para prevenir el sobreajuste, culminando en una capa de salida con la función de activación softmax para la clasificación multiclase.

La compilación del modelo se realiza con el optimizador Adam y la función de pérdida **sparse_categorical_crossentropy**, empleando la precisión (**acc**) como métrica de rendimiento como se menciona en el primer informe. Para el entrenamiento, los datos se dividen en conjuntos de entrenamiento y validación, implementando callbacks como **ReduceLROnPlateau** para ajustar la tasa de aprendizaje y **EarlyStopping** para detener el entrenamiento en ausencia de mejoras en el rendimiento. El modelo se entrena durante 20 epochs, monitoreando continuamente la pérdida y la precisión.

Una vez entrenado, el modelo se evalúa utilizando una matriz de confusión y un reporte de clasificación para medir su desempeño en cada clase. Los resultados finales de la evaluación muestran una precisión del 98.44%, indicando una alta capacidad de generalización del modelo en datos no vistos. Además, se realizan pruebas individuales con imágenes específicas para verificar visualmente el rendimiento del modelo.

Test: En este cuaderno encontraremos un espacio que usamos para probar el modelo ya entrenado, necesitaremos los datos así como en los anteriores cuadernos y lo obtendremos igual, ya que en el paso anterior obtuvimos el modelo entrenado como un archivo, lo ideal era poderlo usar en otros notebooks, para eso usamos gdown ya que en este caso no necesitábamos descargar más de 50 archivos y lo hicimos así:

```
1 file_id = '1yZ72WmBQl5FBS55hv2apaA0nbJGU151E'
2 destination = 'modelo.zip'
3
4 import gdown
5
6 # Descargar el archivo usando gdown
7 gdown.download(id=file_id, output=destination)
8
9 # Descomprimir el archivo
10 !unzip {destination} -d /content/
11 |
```

En este caso definimos una variable llamada `file_id`, este es un identificador que hace parte de los URL que provee google cuando queremos compartir un archivo, un detalle importante es que antes de tomar el `file id` del URL la carpeta debe estar en modo “cualquier persona con el enlace” y el tipo de permiso en “Lector” así:

Acceso general



Cualquier persona con el enlace ▾

Lector ▾

Cualquier usuario de Internet con el enlace puede verlo

Una vez tenemos el URL simplemente tomamos lo que va después de “/d/ hasta antes de “/view”, ese es el id del archivo, lo que descarguemos se guardará en el entorno con la dirección que definimos en el código y cómo este archivo está comprimido, lo descomprimimos.

Más adelante importamos todas las librerías que son necesarias para el correcto funcionamiento del código y cargaremos en el entorno el modelo que acabamos de descargar. Aparte del modelo, en los archivos que descargamos en este notebook, vienen unas imágenes que usaremos para testearlo, algunas hacen parte del dataset original y otras que no lo hacían, para esto tomamos algunas fotos de celebridades y de uno de los integrantes de este equipo y usando una función que acepta una ruta hacia un archivo ejecutaremos varias pruebas.

Gracias a esto último, cualquier persona podría subir una imagen de un rostro al entorno de collab y usando el método `Image_Test` y dándole como parámetro la ruta a esta imagen, revisar si la persona tiene mascarilla, si la tiene mal puesta o directamente no la tiene.

2. Descripción de la solución

Preprocesamiento: En este proyecto, se procesaron imágenes de tres categorías diferentes: `with_mask`, `without_mask` y `mask_weared_incorrect`. A continuación, se detallan las decisiones de diseño y el flujo de trabajo implementado para pre-procesar estas imágenes de manera efectiva.

El primer paso en el preprocesamiento fue cargar y preparar las imágenes para su uso en el modelo. Para lograr esto, se definieron rutas específicas para cada categoría de imágenes. A continuación, se recorrieron las carpetas correspondientes para cargar las imágenes, redimensionarlas y convertirlas al formato adecuado. Este proceso se realizó de la siguiente manera:

1. **Definición de Rutas de Carpetas:** Se establecieron las rutas para las carpetas que contenían las imágenes de cada categoría. Estas rutas se almacenaron en variables como `without_mask_path`, `with_mask_path` y `mask_weared_incorrect_path`.
2. **Carga y Procesamiento de Imágenes:** Para cada imagen en las carpetas especificadas, se llevaron a cabo los siguientes pasos:
 - **Carga de Imagen:** La imagen se cargó utilizando la biblioteca `PIL` (Python Imaging Library).
 - **Redimensionamiento:** Las imágenes se redimensionan a 128x128 píxeles para garantizar la consistencia en las dimensiones de entrada del modelo. Este tamaño se eligió porque es lo suficientemente pequeño para un procesamiento eficiente, pero lo suficientemente grande para conservar características importantes de la imagen.
 - **Conversión a RGB:** Todas las imágenes se convirtieron al formato RGB para asegurar que todas las imágenes tuvieran tres canales de color, independientemente del formato original.
 - **Conversión a Array de NumPy:** Las imágenes se convirtieron a arreglos de NumPy, lo que facilita su manipulación y procesamiento posterior en bibliotecas de aprendizaje profundo como Tensor Flow y Keras.
 - **Almacenamiento de Imágenes Procesadas:** Las imágenes procesadas se agregaron a una lista denominada `data`, que almacenó todas las imágenes preprocesadas de las tres categorías.

Arquitectura: Se utilizó el modelo secuencial de Keras para construir esta red, combinando capas convolucionales, de pooling y densas para extraer y procesar características de las imágenes.

El modelo comienza con una capa convolucional de 32 filtros, seguida de una capa de max pooling para reducir la dimensionalidad. Luego, se añade una segunda capa convolucional con 64 filtros, seguida nuevamente de una capa de max pooling. Este enfoque permite que la red capture características complejas de las imágenes.

Después de las capas convolucionales, las características extraídas se aplanan y pasan a través de dos capas densas intercaladas con capas de dropout. Esto ayuda a prevenir el sobreajuste. La primera capa densa tiene 128 unidades y la segunda 64, ambas con la función de activación ReLU. La capa final es una capa densa con 3 unidades (una por cada clase) y utiliza la función de activación softmax para la clasificación.

Callbacks

Para optimizar el proceso de entrenamiento y mejorar la generalización del modelo, se utilizaron dos callbacks importantes: `ReduceLROnPlateau` y `EarlyStopping`.

Reducción de la Tasa de Aprendizaje (`ReduceLROnPlateau`)

El callback `ReduceLROnPlateau` ajusta dinámicamente la tasa de aprendizaje del optimizador cuando el rendimiento del modelo se estabiliza. Esto ayuda a que el modelo continúe mejorando incluso después de que la mejora inicial haya disminuido. Las configuraciones utilizadas fueron:

- **Monitor:** "`val_acc`" (precisión de validación)
- **Factor:** 0.5 (la tasa de aprendizaje se reduce a la mitad)
- **Paciencia:** 3 epoch(espera 3 epochs sin mejora antes de reducir la tasa de aprendizaje)
- **Tasa de aprendizaje mínima:** 0.00001

Parada Temprana (`EarlyStopping`)

El callback `EarlyStopping` detiene el entrenamiento si el rendimiento del modelo no mejora después de un número determinado de epochs, lo que ayuda a prevenir el sobreajuste y ahorrar tiempo de entrenamiento. Las configuraciones utilizadas fueron:

- **Paciencia:** 5 epochs (espera 5 epochs sin mejora antes de detener el entrenamiento)
- **Verbose:** 1 (imprime un mensaje cuando se detiene el entrenamiento)

Compilación del Modelo

La compilación del modelo se llevó a cabo utilizando el optimizador Adam, la función de pérdida `sparse_categorical_crossentropy` y la métrica de precisión (`acc`). La elección del optimizador Adam se debe a su eficiencia y capacidad para adaptarse a diferentes problemas de optimización, mientras que `sparse_categorical_crossentropy` es adecuada para problemas de clasificación multiclase con etiquetas enteras. La métrica de precisión permite monitorear el rendimiento del modelo durante el entrenamiento.

Entrenamiento del Modelo

El modelo se entrenó utilizando el conjunto de datos de entrenamiento escalado, reservando un 30% de los datos para validación (`validation_split=0.3`). El entrenamiento se realizó durante 20 epochs con los callbacks de reducción de la tasa de aprendizaje y parada temprana. La reducción de la tasa de aprendizaje ajusta dinámicamente el aprendizaje cuando el rendimiento se estabiliza, y la parada temprana detiene el entrenamiento si el modelo deja de mejorar, ayudando a prevenir el sobreajuste y a ahorrar tiempo de entrenamiento.

3. Descripción de las iteraciones

- a. **Exploración:** En la primera iteración, nos centramos en explorar el problema y los datos. Tomamos de la página “Kaggle” un conjunto de datos separados en carpetas que contenían imágenes de primeros planos de rostros de personas, estos podían tener mascarilla, no tenerla o tenerla pero no correctamente ubicada. Luego, exploramos las características de estas imágenes, como su

tamaño, resolución y formatos. Además, realizamos un cuaderno exploratorio de estos datos para mostrar gráficamente algunos detalles de esta misma base de datos.

- b. **Diseño del modelo:** En esta segunda fase diseñamos la arquitectura del modelo CNN a usar. Investigamos arquitecturas de CNN previamente desarrolladas para tareas similares haciendo uso del apartado “Code” que permite ver proyectos de personas que han utilizado previamente los datos que escogimos y seleccionamos una arquitectura base. Ajustamos esta arquitectura según las necesidades específicas del proyecto y definimos la estructura del modelo, incluyendo capas convolucionales, capas de pooling, capas de regularización y capas completamente conectadas. También por utilidad se configuran dos callbacks importantes para el entrenamiento del modelo: reducción de la tasa de aprendizaje y "parada temprana".
- c. **Entrenamiento:** En esta etapa entrenamos el modelo en los datos de entrenamiento. El modelo se entrena por 20 epochs como máximo y para esto primero pre-procesamos los datos de entrada y Compilamos el modelo especificando el optimizador, la función de pérdida y las métricas de evaluación, y se utiliza una validación del 30% de los datos de entrenamiento luego entrenamos el modelo utilizando los datos de entrenamiento, mientras que el modelo se va entrenando vamos tomando métricas como el “accuracy” y “loss” en los cuales se puede notar una mejoría con cada nueva epoch.
- d. **Análisis de resultados:** En esta epochfinal, revisamos los resultados obtenidos en la iteración previa, como ya mencionamos se tomaron las métricas de accuracy y loss en cada epoch para poder revisarlas de manera más específica, sin embargo más adelante se revisa un informe de clasificación que de igual manera confirma que los resultados obtenidos son positivos gracias al modelo.

4. Resultados

Resultados del Entrenamiento

El modelo se entrenó durante 20 epochs, mostrando una mejora constante tanto en la precisión como en la reducción de la pérdida, tanto en el conjunto de entrenamiento como en el de validación. A continuación, se presenta un resumen de los resultados más destacados del entrenamiento:

- **Epoch 1:**
 - Pérdida de entrenamiento: 0.7446
 - Precisión de entrenamiento: 68.95%
 - Pérdida de validación: 0.3866
 - Precisión de validación: 87.71%
- **Epoch 10:**
 - Pérdida de entrenamiento: 0.0816
 - Precisión de entrenamiento: 97.55%
 - Pérdida de validación: 0.1340
 - Precisión de validación: 96.29%
- **Epoch 15:**

- Pérdida de entrenamiento: 0.0405
- Precisión de entrenamiento: 98.70%
- Pérdida de validación: 0.1007
- Precisión de validación: 97.35%
- **Epoch 20:**
 - Pérdida de entrenamiento: 0.0217
 - Precisión de entrenamiento: 99.32%
 - Pérdida de validación: 0.0807
 - Precisión de validación: 97.99%

Estos resultados indican que el modelo mejora significativamente a lo largo del entrenamiento, alcanzando una alta precisión y reduciendo la pérdida de manera efectiva. La precisión en el conjunto de validación también se incrementa consistentemente, lo que sugiere que el modelo se generaliza bien a datos no vistos. Las técnicas de reducción de la tasa de aprendizaje y parada temprana contribuyeron a optimizar el proceso de entrenamiento y prevenir el sobreajuste, resultando en un modelo robusto y bien ajustado.

Aparte, el modelo se evaluó utilizando un conjunto de pruebas y los resultados se resumen a través de la matriz de confusión y el informe de clasificación.

Desempeño General

La matriz de confusión muestra que el modelo clasificó correctamente la gran mayoría de las imágenes en sus respectivas categorías (with_mask, without_mask, y mask_weared_incorrect). Los pocos errores de clasificación indican un alto nivel de precisión en las predicciones del modelo.

El informe de clasificación confirma estos resultados, mostrando una precisión global del 98%. Además, las métricas de precisión, recall y F1-score para cada clase se mantuvieron consistentemente altas, alrededor del 98% al 99%. Estos resultados sugieren que el modelo tiene un excelente desempeño en la clasificación de imágenes en todas las categorías, con una baja tasa de falsos positivos y falsos negativos.

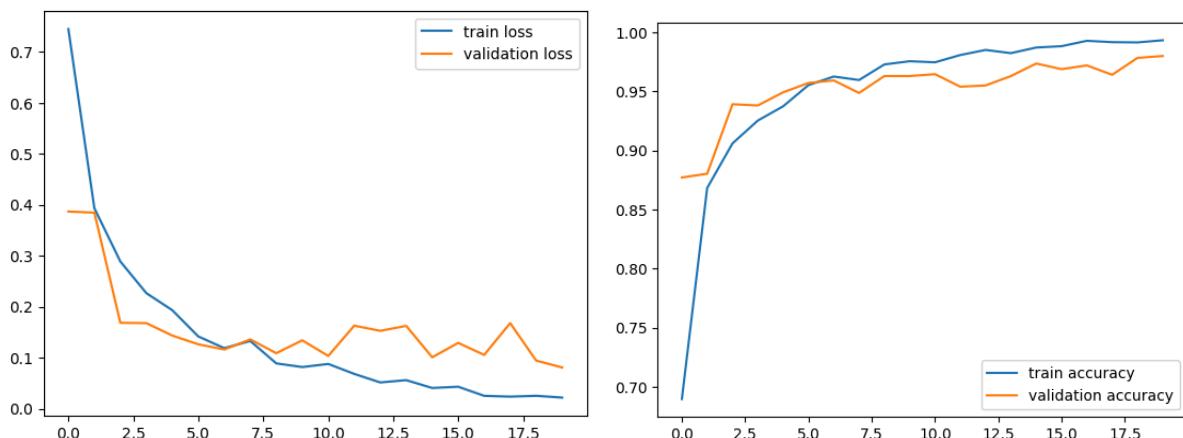
```
Test Confusion Matrix:
[[919 18 1]
 [ 6 841 10]
 [ 2  5 893]]
Test Classification Report:
             precision    recall   f1-score  support
with_mask        0.99      0.98      0.99     938
without_mask     0.97      0.98      0.98     857
mask_weared_incorrect  0.99      0.99      0.99     900
accuracy           -         -         -       2695
macro avg          0.98      0.98      0.98     2695
weighted avg       0.98      0.98      0.98     2695
```

Después del entrenamiento y la validación del modelo, se realizó una evaluación final utilizando el conjunto de pruebas. Esta evaluación final proporciona una medida del rendimiento del modelo en datos no vistos, permitiendo validar su capacidad de generalización.

La evaluación del modelo arrojó los siguientes resultados:

- **Pérdida (Loss):** 0.1076
- **Precisión (Accuracy):** 98.44%

El modelo demostró una alta precisión en el conjunto de pruebas, alcanzando un 98.44%. Este resultado es consistente con las métricas obtenidas durante el entrenamiento y la validación, confirmando que el modelo generaliza bien y mantiene su rendimiento en datos no vistos. La baja pérdida y la alta precisión indican que el modelo es robusto y fiable para la clasificación de imágenes en las categorías de personas con mascarillas, sin mascarillas y con mascarillas mal puestas.



Por último, en el notebook de “test” están las pruebas de funcionalidad con datos del dataset en cuestión al igual que datos por fuera del dataset para demostrar la robustez de las predicciones al igual que la flexibilidad del modelo

ACLARACIÓN

Nuestro modelo fue entrenado con un tipo de imágenes que tiene un par de particularidades, lo primero es que son imágenes de 128 pixeles x 128 píxeles y lo segundo y más importante es que son imágenes que muestran un primer plano a la cara de las personas, esto es de tenerse en consideración cuando se quiera probar el modelo puesto que cualquier imagen no va a ser bien interpretada, la imagen a probar idealmente debe tener las dimensiones mencionadas, pero si se esperan resultados lo más correctos posible lo más importante es que la imagen muestre un primer plano de la cara de la persona, otros elementos como ropa, cuellos, entre otras, podrían hacer que el modelo pierda precisión y mal interprete la imagen.

Esto quiere decir que por ejemplo, esta imagen que vemos a continuación



Es bastante mas valida y otorga un mejor resultado que está



Dentro del testing, el código hace un “reshape” para poder ajustar la imagen a las dimensiones correctas, todo esto para asegurarse el adecuado funcionamiento del modelo, pero la cuestión es que si se sube una imagen cuyas dimensiones no sean las adecuadas como podría ser una selfie, haría que al momento de “reajustar” las dimensiones, quede de forma aplanada, hasta perdiéndose detalles en el camino.