PR 4.2. DEEP LEARNING

Juan Pedro Linares

DOCUMENTACIÓN APARTADO 1

1. Seguir el ejemplo de clasificación de imágenes de flores tratado en los apuntes. Generar el colab y probar su funcionamiento.

Importación de librerías necesarias y preprocesamiento

En este primer apartado importamos las librerías que nos hará falta para utilizarlas en nuestros estudios de redes neuronales.

* La librería “pandas” para la manipulación del dataset
* La librería “tensorflow” para trabajar con funcionalidades para las redes neuronales

Importación de un paquete específico de una librería:

* Importamos “drive” para montar nuestro drive de nuestro correo
* Importamos “train_test_split” para separar nuestros datos de entrenamiento y test

Luego de ello, descomprimimos nuestra carpeta con las imágenes y el dataset que vamos a tratar igual que por otro lado, por otro lado, definimos el path que vamos a utilizar en nuestro código.

Finalmente leemos el dataset.

Definir el conjunto de datos y variables

Para nuestro estudio, hemos dividido utilizando la función train_test_split, los datos de test y los datos de entrenamiento. En este caso, hemos puesto que los del entrenamiento sean un 70% de los datos totales

Por otro lado, hemos utilizado la misma función para separar los datos de test y los de validación, siendo por tanto en este caso, un 15% de los datos totales y los de validación otro 15%

Finalmente actualizamos los índices de cada partición

Red neuronal

Normalizamos nuestras imágenes para que tengan la misma escala. De igual manera declaramos nuestras variables para su posterior uso

Luego de ello, generamos lotes de datos para entrenamiento, validación y prueba utilizando generadores de flujo de datos. De esta manera podemos procesar los datos de manera adecuada.

A cada generador, le añadimos el dataframe adecuado, el directorio, el nombre de la columna con nuestras imágenes, con las etiquetas, controlador de extensión de las imágenes, el modo de clasificación, en este caso categóricas, las dimensiones para las imágenes, el tamaño del lote que anteriormente hemos declarado y la lista de clases que también anteriormente hemos declarado anteriormente

Luego de ello hemos escalado nuestros datos, principalmente porque hay columnas en los que los valores numéricos están entre una gran franja (horquilla), mientras que otras columnas tienen una franja mucha más pequeña.
Luego de un par de pruebas, verificamos que escalando los datos, obteníamos mejores resultados.

Cargamos el modelo que vamos a usar, en este caso InceptionV3, prentrenado con ImageNet

Con el siguiente código, vamos a no hacer entrenables todas las capas del modelo base

Posteriormente hemos añadido nuevas capas a nuestro modelo base.
Agregamos la capa “GlobalAveragePooling2D’ para reducir la dimensionalidad de los datos. Esta capa calcula el promedio de cada canal de características en una imagen y produce una salida para cada canal.
Agregamos una capa densa de 1024 neuronas y una función de activación relu. Esta capa está hecha para aprender datos complejos
Por último, agregamos otra capa densa con un número de neuronas igual al número de clases (n_classes). Con la función ‘softmax’ que asignará una probabilidad a cada clase

Creamos un modelo con todas estas capas creadas anteriormente además del modelo base. Luego con summar() visualizamos un resumen del modelo. También compilamos el modelo con la función de pérdida utilizada para los problemas de clasificación, el optimizador para ajustar los pesos del modelo (Adam) y las métricas para evaluar el rendimiento de nuestro modelo durante el entrenamiento

Tenemos, por tanto, divididos entre parámetros entrenables (aquellos que aprenden y se ajustan al entrenamiento) y los parámetros no entrenables (ya que son fijos y no se ajustan al entrenamiento, debido al modelo base o a las capas congeladas)

Para ir terminando con nuestro estudio, entrenamos el modelo, añadiendo 10 épocas y nuestro generador de flujos. Se van mostrando mensajes de progreso y errores al finalizar cada época. Además, hemos añadido use_multiprocessing para acelerar el entrenamiento

Por último, validamos nuestro modelo con nuestro conjunto de datos, resultándonos con un 81,17% de precisión y una pérdida de 57,38%, bastante adecuada dada la alta precisión que hemos obtenido en nuestro modelo

CONCLUSIÓN

Con este estudio, analizamos las imágenes de nuestros datos asociados a nuestro dataset, entrenando un modelo de red neuronal para que acierte la clase correcta para la flor analizada.

Podemos ver, luego de realizar todo el proceso, de un resultado del 81,17%, un resultado más que óptimo y adecuado para nuestro estudio

-

DOCUMENTACIÓN APARTADO 2

2. El siguiente enlace contiene un codelabs usando redes preentrenadas. Leerlo detenidamente y seguir sus pasos. Finalmente, generar un colab con el código

Importación de librerías necesarias y preprocesamiento

En este primer apartado importamos las librerías que nos hará falta para utilizarlas en nuestros estudios de redes neuronales.

* La librería “pandas” para la manipulación del dataset
* La librería “tensorflow” para trabajar con funcionalidades para las redes neuronales
* La librería “pydot” para poder crear nuestra imagen del árbol de decisiones
* La librería “numpy” utilizada para distintas funciones

Importación de un paquete específico de una librería:

* Importamos “drive” para montar nuestro drive de nuestro correo
* Importamos “train_test_split” para separar nuestros datos de entrenamiento y test
* Importamos “os, sys, math” funcionalidades adicionales de python

Definir el conjunto de datos y variables

En el 1º código que estamos tratando, establecemos variables de configuración igual que, por otro lado, establecemos la división de los datos en conjunto de entrenamiento y validación.

Estas variables de configuración son tales que la cantidad de épocas, definir el tamaño de las imágenes, donde se encuentran los archivos, el tamaño del lote, las clases del os objetos presentes en nuestros datos, los nombres de archivo de nuestras imagenes etc

Además como hemos comentado, definimos que el 19% de los archivos se utilizarán para la validación y el 81% restante para el entrenamiento. Luego de ello añadimos los parametros para calcular el numero de pasos necesarios asi como establecer adecuadamente los datos de entrenamiento y de validación

Los resultados del mismo, son, de los 16 archivos en total, 13 serán para entrenamiento y 3 para validación. Tambien nos informan del tamaño del lote para entrenamiento y para los de validación

Red neuronal

En el siguiente codigo, creamos funciones útiles para nuestro estudio, estás son:

dataset_to_numpy_util: Toma un conjunto de datos y los agrupa en lotes de tamaño, luego va itereando los lotes para convertir las imagenes y etiquetas en arreglos de numpy. Luego los devuelve.

title_from_label_and_target: Define si la predicción es correcta o no, devolviendo si es correcta, la clase y si es incorrecta, que clase debería ser

display_one_flower: Esta función nos hace visualizar una imagen de una flor de nuestros datos con sus etiquetas

display_9_images_from_dataset: Toma nuestros datos y nos muestra 9 imágenes además de sus etiquetas correspondientes.

display_9_images_with_predictions: Lo mismo que anterior añadiendo las predicciones

display_training_curves: Mostramos las curvas de entrenamiento y validaciones, todas estas con sus etiquetas correspondientes

Luego, en el siguiente código definimos dos funciones más:

read_tfrecord: Analiza un ejemplo y devuelve la imagen y su etiqueta de clase correspondiente.

load_dataset: Carga nuestro conjunto de datos a partir de los archivos TFRecord

Posteriormente, utilizando 2 funciones tratadas anteriormente, cargamos nuestro dataset, y mostramos 9 imagenes (de nuestras flores) de los datos de entrenamiento

Continuamos definiendo una nueva funcion, get_batched_dataset, esta devolverá los datos preprocesados y preparados para su entrenamiento o validación.

Y justo después, creamos el modelo de red neuronal. Elegimos la arquitectura MobileNetV2, se congela el modelo preentrenado y posteriormente se construye, compila y se muestra el resumen del mismo.

Lo entrenamos, dándole todos los datos de configuración que seleccionados anteriormente. Obteniendo resultados ya en las últimas épocas bastante elevado de mas de un 87%

Posteriormente, visualizamos las curvas con los datos de entrenamiento con la función declarada anteriormente. Podemos ver según las épocas, como se trata la precisión del modelo, o la perdida del mismo. En ambas vemos, que a más épocas, mejores resultados

Para terminar, trataremos con las prediciones, es decir, generamos una entrada aleatoria de los datos de validación, la ejecutamos con nuestro modelo y obtenemos las predicciones del mismo
Finalmente, mostramos las etiquetas de las imágenes de flores y la coincidencia de las mismas cuando visualizamos las imágenes, siendo todas acertadas

CONCLUSIÓN

A través de este estudio hemos podido comprobar la importancia de la declaración de funciones para su posterior uso. De igual manera hemos podido ver como tratar de analizar las imágenes con sus etiquetas asociadas.

Hemos visto los muy buenos resultados, llegando a un 87% y como luego se pueden mostrar las predicciones de la mejor manera

DOCUMENTACIÓN APARTADO 3

3, Revisar este cuaderno de kaggle de detección de género en imágenes. Crear un colab y probar su funcionamiento.

Importación de librerías necesarias

En este primer apartado importamos las librerías que nos hará falta para utilizarlas en nuestros estudios de redes neuronales.

* La librería “pandas” para la manipulación del dataset
* La librería “tensorflow” para trabajar con funcionalidades para las redes neuronales
* La librería “pydot” para poder crear nuestra imagen del árbol de decisiones
* La librería “numpy” utilizada para distintas funciones
* La librería “base64” convertir imagenes a formato base64 para verlas mejor en Jupyter
* La librería “metrics” para metricas de evaluacion
* La librería“tensorflow.keras” para construir/entrenar modelos de aprendizaje profundo

Importación de un paquete específico de una librería:

* Importamos “train_test_split” para separar nuestros datos de entrenamiento y test
* Importamos “os, sys, math” funcionalidades adicionales de python
* Importamos “ImagenDataGeneratos” se utiliza para generar lotes de imágenes
* Importamos “image” para funciones de manipulación de imágenes

Definir el conjunto de datos y variables

Después de importar, definiremos las variables de nuestro proyecto. Estas serían: las rutas de la carpeta de las imágenes y la ruta de una imagen de ejemplo. Por otro lado, definiremos el conjunto de datos; la cantidad de muestras utilizadas en entrenamiento, validación y pruebas. Bajadas las de entrenamiento para no sobrepasar la ram de google colab, las dimensiones de las imágenes, el tamaño de los lotes y el número de épocas

Posteriormente, cargamos los atributos de nuestro dataset para luego visualizar uno a uno todos los atributos disponibles en nuestro conjunto de datos

Luego también visualizaremos una imagen de ejemplo, en la que mostraremos además los atributos asociados a la misma

Despues, mostraremos un gráfico de barras para ver la cantidad de imágenes etiquetadas con “Male” y con “Female”

Leemos nuestro csv y visualizamos los primeros 5 datos para ver que se ha cargado correctamente, así como también en la que vemos la repartición de los datos en las distintas particiones y finalmente unimos la información de la partición con los atributos de los conjuntos de datos

Red neuronal

Luego crearemos dos funciones útiles en nuestro estudio:

load_reshape_img: la cual carga una imagen dado un nombre de archivo, la convierte a array y la normaliza.

generate_df: genera un dataframe de las imágenes y etiquetas de la partición que elijamos y del atributo específico, así como también la cantidad de muestras. Dependiendo de que sea la partición, actuará de una manera u otra, si no es conjunto de pruebas, las etiquetas se convierten a categóricas, en cambio, si lo es, se cargan y se redimensionan las imágenes

En el posterior código, definimos un generador de imágenes, la cual muestra una imagen cargada y se aplica una aumentación de datos para generar 10 imágenes más adicionales.

Luego de ello, se prepara el conjunto de datos de entrenamiento y se configura el generar para aplicar la aumentación de datos durante el entrenamiento. El resultado del mismo proporcionará lotes de imágenes aumentadas y etiquetas correspondientes

Procedemos generando el conjunto de datos de validación, indicando la partición y el atributo “MALE” como la cantidad de muestras deseadas

Continuamos creando nuestro modelo, para ello tenemos que definir el modelo que vamos a utilizar, en este caso es el modelo InceptionV3.

Vamos a definir capas personalizadas para añadir al modelo anteriormente creadas. “GlobalAveragePooling2D” el cual calcula el promedio de todas las activaciones en cada mapa de características. Se agrega una capa dense de 1024 unidades, una capa de dropout ayuda a prevenir el sobreajuste al apagar aleatoriamente algunos datos del entrenamiento así como otra capa densa de 512 unidas con activation relu o una capa densa final de 2 unidades con función de activation “softmax”.

Añadimos estas capas personalizadas que hemos definido a nuestro modelo para dejarlo definitivo. Mismo proceso que el resto de redes neuronales, añadimos capas, decidimos las capas no entrenables, lo compilamos con las características que nos interesen.

Posteriormente vamos a configurar un callback para utilizarlo luego, en este caso un “ModelCheckpoint”. Este se utiliza durante el entrenamiento para guardar los pesos del modelo en un archivo HDF5 cada vez que se mejore la métrica especificada. Para ello, especificamos la ubicación y nombre del archivo donde guardaremos los pesos del modelo, la verbosidad que elijamos y “save_best_only” mencionado justo antes

Después de todos estos detalles, entrenamos nuestro modelo, utilizando el generador de datos de entrenamiento, y añadiendo los distintos atributos que hemos ido definiendo, además del callback anteriormente creado. Colocamos use_multiprocessing para habilitar varios múltiples procesos para la carga de datos. Guardamos el historial del entrenamiento

Hemos obtenido muy buenos resultados, con poca pérdida y una gran precisión. Para evaluar los resultados de mejor manera, visualizamos gráficos de líneas de la precisión o pérdida en función de las épocas

Continuamos cargando nuestro modelo anteriormente guardado gracias a nuestro callback. Con dicho modelo, lo evaluaremos para ver los resultados de la predicción, calculando la predicción del mismo, dando un 81,10%, un muy buen rendimiento

Le damos uso al funcionamiento de nuestro modelo, en este caso, utilizamos varias funciones:

La función “img_to_display”, el cual toma el nombre de un archivo y te devuelve en base64 la imagen.

La función “display_result” el cual muestra los resultados de la predicción en formato HTML. La imagen además se muestra junto con un icono del género que corresponda y se formatea con estilo CSS

La funcion “display” para mostrar el contenido HTML que hemos generado

La función “gender_prediction”, lee la imagen utilizando OpenCV y realiza transformaciones en la imagen, y luego se realiza la predicción con nuestro modelo creado anteriormente. Devuelve el resultado de la predicción

Finalmente, seleccionamos aleatoriamente 8 imágenes de la partición de prueba de nuestro dataframe, utilizamos las funciones anteriormente mencionadas para mostrar las imágenes con sus distintas predicciones, de manera visual

Conclusión

Lo realmente interesante que hemos podido ver en este estudio es la aplicación que se puede dar a nuestro modelo una vez terminado (como es lo último que hemos aplicado), con buenos resultados.

Eso sí. Como también lo comentan en el estudio, tiene algunas limitaciones, como que no está contemplado el modelo en el caso en el que haya distintas caras en la misma foto, como también, que la distancia y la perspectiva de la cara es igual en todas las imágenes tratadas, en otro tipo de fotos de caras, las predicciones pueden ser mucho menores

DOCUMENTACIÓN APARTADO 4

4. Resolver el problema de clasificación de imágenes de animales planteado en el siguiente dataset

Importación de librerías necesarias

Importamos las librerías:

* La librería “keras” para construir y entrenar redes CNN
* La librería “os” para interactuar con el sistema operativo
* La librería “shutil” para realizar operaciones de manipulación de archivos y directorios
* La librería “plotly” de visualización interactiva para crear gráficos y visualizaciones
* La librería “seaborn” de visualización de datos
* La librería “pydot” para poder crear nuestra imagen del árbol de decisiones
* La librería “numpy” utilizada para distintas funciones

Importación de un paquete específico de una librería:

* Importamos “glob” encuentra archivos que coinciden con un patrón de directorio
* Importamos “tqdm” muestra una barra de progreso en bucles o procesos
* Importamos “ImageDataGenerator” genera lotes de datos de imagen para los modelos
* Importamos “Sequential” crea modelos de red neuronal secuenciales
* Importamos “load_model” carga modelos guardados desde archivos
* Importamos “dense, GlobalAvgPool2D” capas para nuestro modelo
* Importamos “ModelCheckpoint, EarlyStopping” callbacks para nuestro modelo
* Importamos “InceptionV3” modelo preentrenado de reconocimiento de imágenes

Definir el conjunto de datos y variables

Una vez que hemos subido los datos que vamos a analizar en nuestro drive, lo descomprimimos con “!unzip”

Luego de ello, definimos algunas de nuestras variables para su posterior uso. Además de ello, visualizamos las clases de nuestros datos, así como la cantidad que hay

Posteriormente, calculamos el número de muestras de cada clase y luego las mostramos. Contamos los números de archivo de cada directorio (clases)

Creamos un diccionario llamada class_name_size en el que relacionamos el nombre de la clase con su tamaño correspondiente.

En el siguiente código, mostramos datos de manera aleatoria de las clases originales y creamos una muestra de datos en un nuevo directorio. Seleccionamos un número máximo de imágenes por clase y se copian desde la ubicación original de muestra correspondiente. Importante añadir que expecifamos con sample_percentm que el porcentaje de muestra será de un 0.1 (10%). Tomamos un mínimo de 2000 imágenes a tomar. Importante tambien que cambiamos los nombres de los directorios al español.

Acto seguido, mostraremos de nuevo la distribución de las clases y el nuevo nombre de las clases, el cual hemos cambiado

Continuaremos creando y configurando un generador de datos de imágenes a través de Keras (especificando el 20% que se utilizarán para datos de validación) y luego creando dos flujos de datos, el de entrenamiento (‘train_data’) y el de validación (‘valid_data’) a partir de un directorio de imágenes de muestra. Redimensionar las imágenes, aplicamos transformaciones de aumento de datos y las dividimos en subconjuntos de entrenamiento y validación

Luego creamos varias funciones para mostrar imágenes aleatorias de las de entrenamiento.

La función “show_image” para mostrar una imagen

La función “get_random_data” que toma una tupla de datos que contiene imagenes y etiquetas y selecciona aleatoriamente una imagen y su correspondiente etiqueta.

Por último, creamos la visualización que queremos enseñar con la ayuda de las funciones anteriormente mencionadas. Esta visualización será imágenes de animales aleatorios y sus correspondientes etiquetas,

Red neuronal

Entrenamos nuestro modelo, en este caso, como también hemos utilizado en otras prácticas, con arquitectura Inception V3 preentrenada con el conjunto de datos ImageNet

En dicho entrenamiento, especificamos el tamaño de entrada de imágenes, añadimos nuevas copas densas, así como más tarde luego lo compilamos utilizando el optimizador “adam” y la métrica de accuray. Como también hemos utilizado en anteriores actividades, crearemos dos callbacks, ‘EarlyStopping’ para detener el entrenamiento si no hay mejora después de 3 épocas y ‘ModelCheckpoint’ para guardar solo el mejor modelo durante el entrenamiento. Finalmente lo entrenamos con fit añadiéndole el resto de variables necesarias.

Luego calculamos la pérdida y la precisión de nuestro modelo “inception” utilizando nuestro conjunto de datos de validación. Mostramos y analizamos los resultados, los cuales podemos evaluar que son adecuados para nuestro estudio.

Para verlo de una mejor manera, vamos a visualizar 2 gráficas, la cual veremos la precesión según las épocas o la medida de pérdida.

Finalmente, creamos una visualización de los datos, de las etiquetas que eran y de la predicción. De manera visual podemos ver una muestra de nuestros datos y predicción

Conclusión

Podemos concluir que que podemos obtener con un buen modelo, resultados óptimos a la hora de analizar las imágenes. En nuestro caso no solo vemos vemos que tenemos una alta precisión (90%) si no que también podemos ver de primera mano esos resultados de manera visual viendo la predicción dada y la imagen que ha tratado

Incluso con imágenes más complicadas o que como tal, no deben ser de animales, sino dibujos de la misma, obtiene muy buenos resultados