TRABAJO CREATIVO
NEUROCOMPUTACIÓN Y REDES NEURONALES
(README)

Jaime Enríquez Ballesteros

INDICE

1. Introduccion

Para este trabajo creativo, he desarrollado una aplicación basada en redes neuronales, utilizando tensorflow, ladjfkla de goooogle, y python, cuyo objetivo es el de reconocer numeros y operandos a partir de imágenes.

Para este proyecto he utilizado dos bases de datos: la primera, unas 60000 imágenes de números entre el 0 y 9, provenientes de la famosa *MNIST database,* con la que mediante la propia aplicación, las descargo para entrenar a una red neuronal. La segunda es una base de datos que yo he realizado manualmente, mediante aprox. 40 imágenes de cada operando (+, -, x, /) que resulta en 169 imágenes, las cuales sirven para entrenar a otra red neuronal.

Las imágenes de los operandos, fueron escaneadas y recortadas, tras escribirlos a mano, para que la red neuronal pudiera reconocer operandos que habian sido escritos por una persona.

Antes de realizar este proyecto, ya había utilizado tanto python como tensorflow. Aunque tras tiempo sin utilizarlos, y para familiarizarme con las redes neuronales, leí un artículo titulado ***Neural networks and back-propagation explained in a simple way***[1] para aprender el funcionamiento interno de las redes neuronales, y el libro ***Deep learning with Python***[2] de François Chollet.

Se realizaron dos tutoriales, en ***jupyter notebooks***[3], localizados en la carpeta *"tutoriales_tf"* sacados de la pagina web oficial de tensorflow, que me ayudaron con el reconocimiento y detección de imágenes con dicho módulo.

Para el entrenamiento de las imágenes de operandos, en un primer momento, se utilizaron imágenes descargadas de internet, mediante un script en python (localizado en "*dataset_imagenes*") que mediante una API de Bing, descargaba imágenes insertando una palabra / frase de búsqueda. Estas imágenes están localizadas en "*dataset_imagenes/datasets/internet*". El script está basado en aquel hecho en **pyimagesearch.com** por Adrian Rosebrock[4].
Tras ver los pobres resultados, decidí utilizar imágenes escritas a mano (en "*dataset_imagenes/datasets/scanner*").

AVISO: Para utilizar una imagen como input para calcular una operación, se deberá tener en cuenta que los elementos de la operación deben de estar bien separados. De esta manera, cada elemento podrá ser detectado por la aplicación de manera clara y precisa. (ejemplo: "*4_+_4.jpeg*")

2. Módulos

La aplicación consta de dos módulos secundarios (“*reconocimiento_numeros.py*” , “*reconocimiento_operaciones.py*”) y un módulo principal: “*main.py*”.

- “*reconocimiento_numeros.py*” consta de tres funciones:

  1. *cargar_numeros_desde_mnist()*: que carga las imágenes y las etiquetas que las acompañan desde la base de datos MNIST.

  2. *crear_modelo_numeros(train_images, train_labels)*: que crea el modelo de red neuronal sobre el que se va a entrenar mediante las imágenes y etiquetas que se han obtenido en *cargar_numeros_desde_mnist()*. Se utilizarán tres capas; dos densas y sobre ellas, se allanarán las imágenes para así utlizar un array de 1 dimensión en vez de imágenes de 2 dimensiones. Se utiliza el optimizador de Adam, recomendado para detección de imágenes, se tratará de mejorar la precisión del modelo.

  3. *prediccion_numeros(model, image)*: que trata de predecir que número es el insertado en formato de imagen (*image*), mediante el modelo que ha sido insertado como primer argumento.

- “*reconocimiento_operaciones.py*” consta de 3 funciones:

  1. *transformacion_operaciones_a_datalabels(directorio_dataset)*: que crea dos arrays, uno con imágenes con operandos y otro con sus etiquetas correspondientes (suma, resta, multiplicacion, division), a partir de las imágenes localizadas en *directorio_dataset*.

  2. *crear_modelo_operaciones(directorio_dataset)*: donde se crea el modelo relativo a la base de datos con imágenes, localizada en *directorio_dataset*. El modelo está compuesto por 6 capas. Las tres primeras, con tres capas de convolución, ideales para entrenar modelos con pocas imágenes. Sobre ellas, como en *crear_modelo_numeros()* dos capas densas y otra que allanará las imágenes para crear arrays de 1 dimensión.

  3. *prediccion_operacion(model, image)*:que trata de predecir que operación es la insertada en formato de imagen (*image*), mediante el modelo que ha sido insertado como primer argumento.

-*"main.py"* que consta de 3 funciones:

1. *"main()"*: ejecuta el programa principal de la aplicación donde se entrenan y crean los modelos y se predice con ellos, mediante una serie de imágenes que sirven como outputs.

2. *"solve_operation(pred)"*: pred es una lista que contiene los elementos de la operación a resolver, tanto operandos como números, como strings. Resuelve las operaciones, teniendo en cuenta la prioridad de las multiplicaciones y divisiones.

3. "*split_image_into(input_path, n)*": donde se divide una imagen en n partes. La dividirá dependiendo de cuantos elementos tiene la operación (operandos + números). Devuelve un array de imágenes, cuya longitud es n.

3. Funcionamiento

Para probar la aplicación se recomienda consultar el menú que incluye la aplicación:

python main.py -h

Algunos ejemplos son:

1. *python main.py -eo dataset_imagenes -en -op prueba_imagenes/op1.jpeg 3 -r*

2. *python main.py -eo dataset_imagenes -operand prueba_imagenes/0004_plus.jpeg*

3. *python main.py -en -num prueba_imagenes/6_num.jpeg*

(1) Crea y entrena modelos. Predice operación y su resultado
(2) Crea y entrena modelo de operandos. Predice operando
(3) Crea y entrena modelo de numeros. Predice numeros

(*) Se incluyen imágenes para probar el funcionamiento de la aplicación en el directorio *“prueba_imagenes”*.

AVISO 2: Los modelos tardan aproximadamente 2 minutos en crearse, cargar y entrenarse. Para obtener un buen rendimiento se necesita 1 minuto como mínimo de entrenamiento.

4. Dependencias

La aplicación requiere algunas dependencias, que pueden ser instaladas facilmente desde la terminal, mediante la herramienta basada en python, *pip* o mediante un apt-get install. Las dependencias son las siguientes:

- **cv2**
- **imutils**
- **numpy**
- **tensorflow**
- **keras**
- **requests***

(*) No son necesarias si no se pretende descargar imágenes de internet que vayan a ser utilizadas para entrenar los modelos

Si se tiene algún problema al insalar alguna dependencia, se recomienda consultar en internet.

5. Resultados

* Los resultados obtenidos con la aplicación, no han sido perfectos al 100 %. Aunque al entrenar a los modelos, la precisión sea de más de un 90% en ambos de ellos, la predicción no suele ser acertada, sobre todo en cuanto a los números. Mediante la experimentación he logrado deducir que esto se debe, principalmente, a que las imágenes de MNIST contienen números muy bien marcados (con mucho grosor). Mientras que en las imágenes de prueba con números de poco grosor, el modelo suele equivocarse.

* El modelo de operandos, tras empezar a utilizar la base de datos con imágenes escaneadas hechas a mano, los operandos empezaron a ser reconocidos por el modelo con mayor precisión.

* Aún así, el modelo de operandos suele equivocarse entre sumas y multiplicaciones, y entre restas y divisiones; debido a la similitud en cuanto a su forma y las pocas imágenes que contiene su modelo.

* Tras mucha experimentación, se puede deducir que el acierto de los modelos suele ser de un 60 % aproximadamente (3/5) en cuanto a los números y de un 80% (4/5) en cuanto a los operandos. Por ello, calculamos que el acierto para una operación compuesta por n números es aproximadamente:

ACIERTO (%) = [0.8^(n-1) * 0.6^n] * 100

* Causas de fallos (supuestas):

Como se ha mencionado anteriormente, estos datos contradicen los datos de precisión que aparecen mientras se entrenan los modelos. Se justifica debido a que esa precisión se calcula sobre las imágenes pertenecientes al propio modelo. Al **hacer una foto, debido a la luz**, o **al cambiar de instrumento de escritura** (boli,lapiz,rotulador), distinto al que se utiliza en las imágenes del modelo, se suele tener problemas para reconocer los elementos.

Otro problema suele ser **el ajuste horizontal y/o vertical** (que la foto encuadre con la horizontal y vertical) de la foto a predecir, que resulta en confusión entre suma y multiplicación o entre resta y división

7. Bibliografía

1. *Assaad MOAWAD,* **Neural networks and back-propagation explained in a simple way**

https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

2. François CHOLLET, **Deep learning with Python**
http://www.deeplearningitalia.com/wp-content/uploads/2017/12/Dropbox_Chollet.pdf

3. Adrian ROSEBROCK, **How to (quickly) build a deep learning image dataset**
https://www.pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/

4. Adrian ROSEBROCK, **Easy guide to buid new tensorflow datasets with Keras**
https://www.dlology.com/blog/an-easy-guide-to-build-new-tensorflow-datasets-and-estimator-with-keras-model/

5. What is a NN
https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.59203&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

6 Basic classification tf:
https://www.tensorflow.org/tutorials/keras/basic_classification

7. Importign data tf:
https://www.tensorflow.org/guide/datasets

8. MNIST Database:
http://yann.lecun.com/exdb/mnist/

Otras consultas:

Split image into multiple pieces:
https://stackoverflow.com/questions/5953373/how-to-split-image-into-multiple-pieces-in-python

Pasar imagen a balnco y negro:
https://www.blog.pythonlibrary.org/2017/10/11/convert-a-photo-to-black-and-white-in-python/