

OCR with Neural network

Daniel Azemar i Carnicero
daniel.azemar@e-campus.uab.cat

Richard Segovia Barreales
richard.segovia@e-campus.uab.cat

Abstract

The main goal of this project is using a neural network to build an OCR that can recognize characters and symbols. From an image, we will be able to recognize the lines, split the lines into words and the words into characters, and then use the neural network to predict which letter it is. Another goal is to be able to analyze nonperfect text (for example text taken by a camera).

Introduction

En los últimos años el OCR se ha convertido en un punto de interés en la sociedad. No es un problema nuevo, pero con el avance de las capacidades computacionales, se ha hecho más posible para el alcance de todos y más rápido.

El hecho de poder convertir textos antiguos, manuscritos o libros que sólo existen en papel, a digital evitando la entrada por teclado y de forma masiva implica un gran ahorro de recursos humanos y aumento en productividad.

También, gracias al creciente uso de dispositivos electrónicos con entrada táctil, un buen reconocimiento de los caracteres escritos permitirá al usuario ahorrar tiempo al poder pasar sus anotaciones directamente a documentos de texto.

Todo esto beneficiaría tanto a empresas como a usuarios particulares.

En nuestro caso vamos a implementar un OCR que mediante redes neuronales [5] permita digitalizar textos escritos a máquina. Para las redes neuronales usaremos dos alternativas, las redes neuronales convencionales y las redes neuronales con capas de

Project Final Report

convolución. En cuanto a el tratamiento de la imagen para separar las frases, las palabras y las letras, utilizaremos métodos en primer lugar de morfología y de labelling para poder identificar las distintas partes. También sería interesante el uso de métodos de homografía para posicionar correctamente el documento que debe de ser analizado.

Project Materials and Data

Finalmente hemos usado el dataset [7] que contiene letras escritas a máquina que nos serán muy útiles para entrenar la red neuronal ya que tiene una gran variedad de fuentes. Hemos elegido este dataset porque procede de una de los repositorios más grandes y fiables. El dataset es de gran tamaño por lo que en el momento de ajustar parámetros para la red hemos usado la mitad o un cuarto del tamaño original.

Como preprocesamiento de los datos, intentaremos usar métodos de morfología y filtrado lineal, para eliminar ruido, manchas y otros problemas y quedarnos únicamente con los caracteres reales que hay en el texto.

En cuanto a la implementación hemos decidido usar keras [2] con backend de theano [1] porque permite una gran variedad de arquitecturas de redes neuronales, además de que permite el uso de la GPU de forma totalmente transparente para el entrenamiento de la red neuronal usando PyCuda [3] y otras librerías de Nvidia.

Methodology

Para realizar la parte de detección de caracteres estamos usando la librería *keras* sobre *theano*. Ejecutandolo en la GPU para ganar velocidad con CUDA y PyGPU. Todo realizado en Windows 10, solucionando los errores manualmente a lo largo de la instalación (las librerías también están disponibles para MacOS y Linux). Se ha usado

Project Final Report

un portátil MSI con un i5 4210H, 8GB de RAM y una Nvidia GTX 850M para entrenar la red neuronal.

Para realizar el tratamiento de imagen estamos usando las librerías *skimage*, *opencv*, *numpy* y otras adjuntadas en el código.

Hemos hecho una pauta a seguir general para la detección de caracteres en una sola imagen.

1. Entrenar la red neuronal con la base de datos.
2. Tratar de mejorar la calidad de la imagen (ruido).
3. Binarizar la imagen, para hacer morfología y resaltar las letras
4. Rotar la imagen (por si la imagen está torcida).
5. Obtener las imágenes de caracteres:
 - a. Detectar líneas
 - b. Detectar palabras
 - c. Detectar caracteres
6. Pasar las imágenes a la red neuronal y generar texto resultante.
7. Mostrar el resultado.

Pseudocódigo 1: Código de referencia para cumplir objetivos.

Todos los apartados los comentaremos a continuación con detalle.

1. Entrenar la Red Neuronal con la base de datos

La red final utilizada es una red neuronal convolucional basada en [8] pero con modificaciones pertinentes para que se adapte correctamente con nuestros datos. Finalmente hemos usado únicamente el dataset [7]. Para tratar de mejorar los resultados, seguimos los métodos de análisis de [9] y [10].

En un principio estábamos usando redes neuronales sin capas convolucionales, estas redes estaban dando resultados de entre el 60 % y el 79 % de accuracy en función de las capas utilizadas. La que obtuvo mejores resultados fue una con 1024x1024-Relu, 256x256-Relu, 512x512-Relu, 64x64-Softmax con un 79% de accuracy utilizando la mitad del dataset.

Aparentemente, en este momento los resultados podrían considerarse buenos, acertando el 79% de las letras, el problema es que las imágenes de las letras extraídas del texto, no siempre están en las mismas posiciones y no son perfectas. En ese momento decidimos utilizar redes neuronales con capas de convolución para ver si de esta manera se podía conseguir más robustez en la predicción.

Project Final Report

Al utilizar redes neuronales convolucionales, nos basamos en la red propuesta por [8], pero como nuestros datasets y estructura eran diferentes, modificamos ciertos parámetros. Finalmente acabamos con la siguiente estructura:

- Convolution, 3x3 Relu in, 32x32 out
- Convolution, 3x3 Relu in, 32x32 out
- MaxPooling, 2x2
- Convolution, 3x3 Relu in, 64x64 out
- Convolution, 3x3 Relu in, 64x64 out
- MaxPooling, 2x2
- 256x256-Relu Layer
- 64x64-Softmax Layer

Con esta red, los resultados obtenidos fueron del 82 % de accuracy, pero al introducirlos de nuevo con las imágenes extraídas del texto, los resultados fueron mejores que con la red no convolucional, pero no excesivamente buenos. Aparte de esto, las redes convolucionales son mucho más lentas de procesar, con lo que tardábamos 20 minutos con un cuarto del dataset y 40 con la mitad del dataset. Por lo que estábamos ciertamente limitados en el número de pruebas a realizar.

Finalmente nos decidimos a utilizar métodos de data augmentation, rotar, aumentar y trasladar las imágenes, con el fin de aumentar la robustez de la red neuronal ante desperfectos o imágenes no perfectas, más similares a las extraídas del texto. Para realizar estas modificaciones utilizamos la función integrada en keras, *ImageDataGenerator* que nos permite realizar estas operaciones de forma transparente y optimizada. Este data augmentation consistirá en realizar zoom, movimientos de la imagen, y rotaciones leves.

Para este caso, utilizamos dos entrenamientos, uno entrenando una red desde 0 con estos datos aumentados, y en segundo lugar entrenamos una red primero con los datos sin aumentar y después con los datos aumentados. Los resultados fueron de un 65 % de accuracy para la primera con datos aumentados y un 83 % con datos sin aumentar, para la segunda red, obtuvimos un 67% de acierto para los datos aumentados y un 84 % de accuracy para datos sin aumentar. La mejora en la

Project Final Report

predicción del texto extraído fue notable con la utilización del data augmentation y el texto extraído fue mejor para la red previamente entrenada con datos sin aumentar.

Finalmente reentrenamos la red neuronal anterior, usando data augmentation con la segunda parte del dataset, que no se había utilizado por la limitación en el cómputo, y los resultados mejoraron ligeramente.

2. Tratar de mejorar la calidad de la imagen

2.1 Denoising: Utilizando la técnica de N-means conseguimos suavizar los defectos provocados por noise en la imagen.

2.2 Neutralizacion: Con tal de evitar posibles cambios bruscos de luz i/o manchas, aplicamos una neutralización de la imagen (usando un erode de ella misma y dividiéndola con la original). Conseguimos resultados como los siguientes:

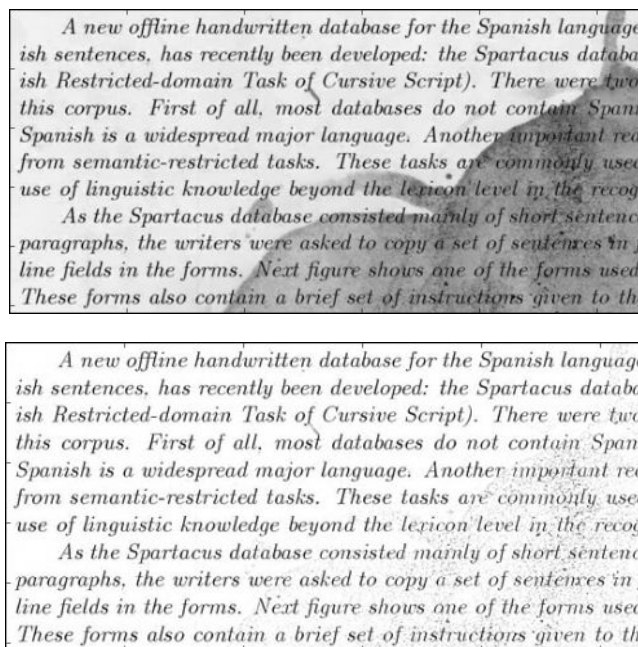


Figura 1: Neutralizacion de una imagen. Arriba: Imagen original. Abajo: Imagen neutralizada.

3. Binarización

3.1 Thresholding: Aunque el neutro nos devuelve una imagen aparentemente óptima para realizar un thresholding global, no lo es. Podemos comprobar en las imágenes de

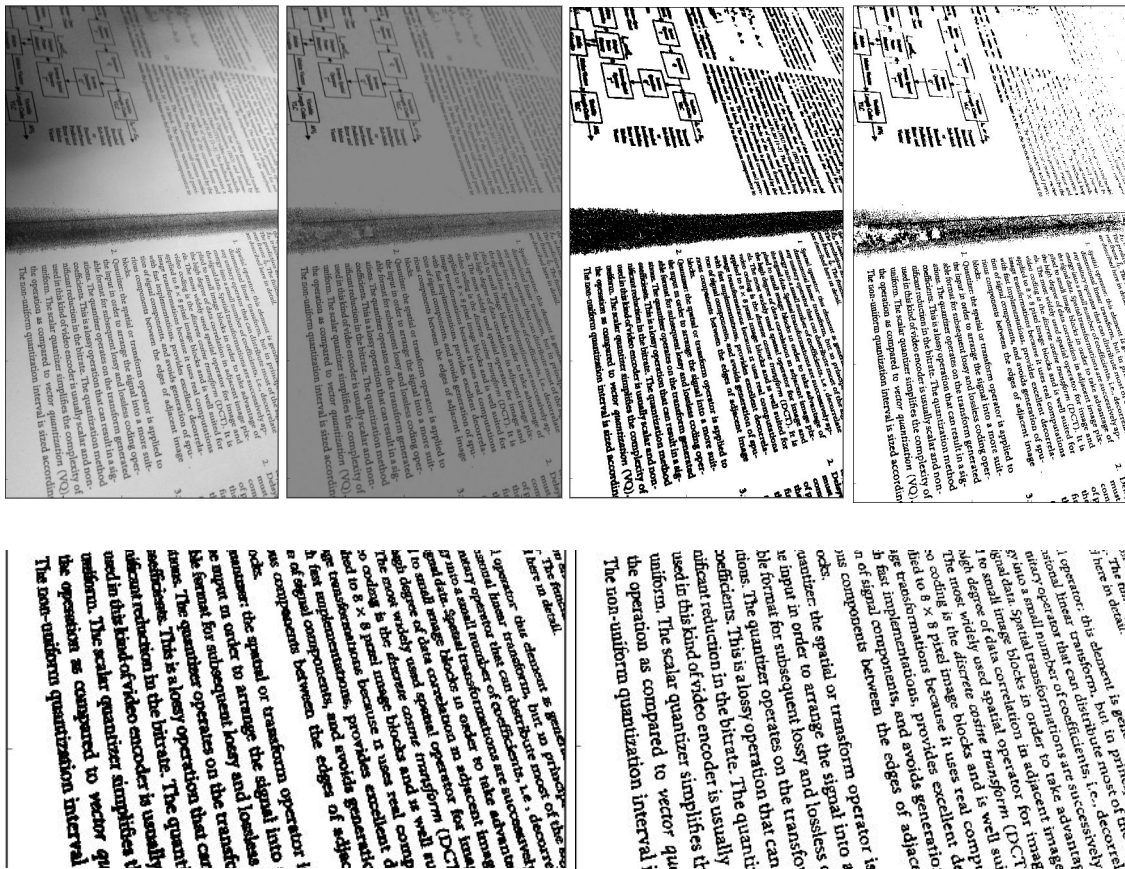


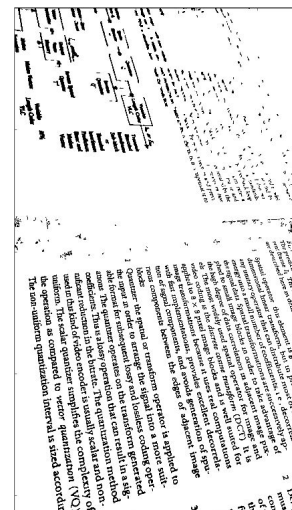
Figura 2: Thresholding de una imagen.

Arriba: De izquierda a derecha: Original, Neutralizada, Thr. Global, Thr. Local
Abajo: Diferencia al detalle entre un thresholding Global y uno Local.

la figura 2 cómo aplicando un thresholding global texto pierde detalle. Escogiendo el local conseguiremos mas calidad en las letras, que es lo que queremos conseguir para que la red neuronal funcione de forma óptima.

3.2 Eliminar manchas: Con tal de reducir el número de inputs falsos a la red neuronal, tratamos de eliminar tanto las manchas grandes como las pequeñas, escogiendo aquellas cuyo tamaño supera o se va de la desviación estándar.

Podemos contar las regiones iniciales y finales de este proceso, con lo que conseguimos inicialmente un total de 4442, y finalmente 1320.



4. Rotación de la Imagen.

Para añadir más valor a nuestro trabajo, hemos querido sumar que se puedan detectar caracteres incluso en fotografías donde el texto no está completamente alineado horizontalmente.

Para esta tarea hemos utilizado la transformación de *Radon* que implementa *scikit-image* de python. Ésta transformación consiste en lanzar rectas con un ángulo variante en la imagen y encontrar el ángulo donde coincidan más píxeles. Podemos ver los resultados en la Figura 4.

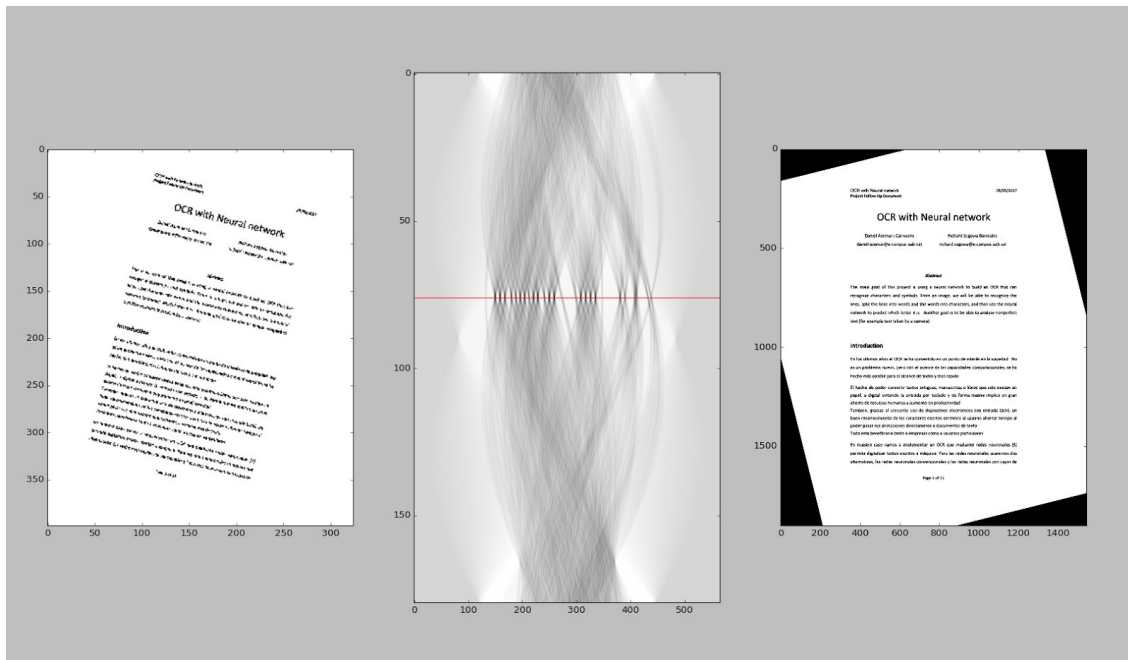


Figura 4: Utilizando la transformada de radon conseguimos detectar el ángulo de rotación.
Por orden: Imagen Rotada, Transformada de Radon de la Imagen, Imagen reorientada.

5. Detección de caracteres

Una vez tratada, binarizada y fijada la rotación de la imagen, podemos empezar a detectar los caracteres. Con tal de conseguir una estructura que después podamos reconstruir, hemos detectado líneas, palabras y símbolos por separado.

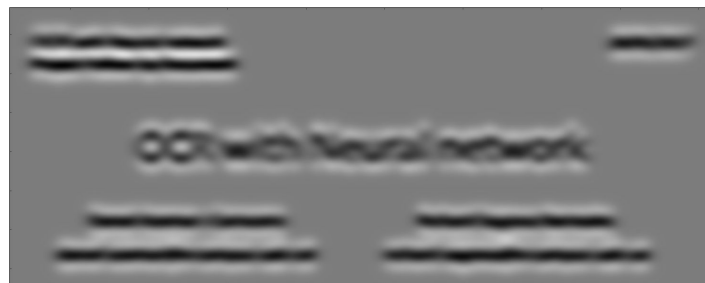
5.1 Detección de Líneas: El proceso que hemos utilizado para la detección de líneas es

el siguiente:

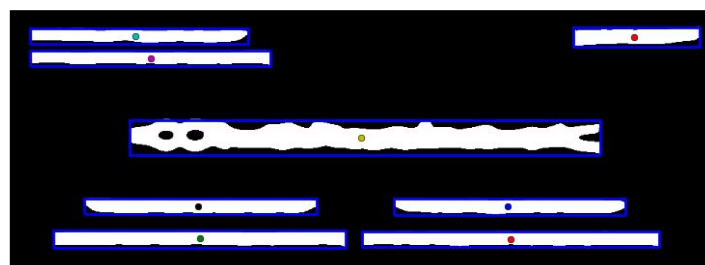
- a. Pasar filtro de frecuencias (Sobel). Con ello conseguimos resaltar los contornos de la palabra.



- b. Esborronado gaussiano en X. Con ello conseguimos una máscara interesante para hacer thresholding.



- c. Ahora, haciendo un thresholding sacamos las manchas de dónde se encuentran las líneas. Con un *regionprops* podemos sacar la posición de ésta.



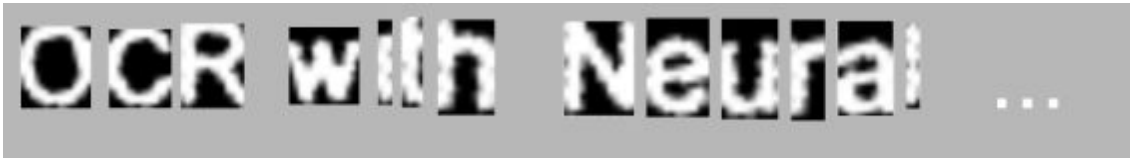
- d. Recortamos las regiones en la imagen original, y pasamos las imágenes resultantes al siguiente nivel.

5.1 Detección de Palabras: Este apartado simplemente usamos morfología sobre la imagen entrante para identificar y recortar las palabras para pasarlas al siguiente nivel.



Project Final Report

5.3 Detección de Carácteres: Ahora sólo nos falta realizar un *regionprops* para detectar las letras. Con tal de seleccionar bien los acentos y los otros símbolos que son la unión de una o más formas (ies, jotas, etcétera) hemos realizado un closing vertical. Invertimos el color de la imagen y la guardamos en la estructura.



Con esto ya tenemos una estructura de líneas, que contiene palabras, que éstas a la vez contienen imágenes de letras.

6.Pasar las imágenes a la red neuronal y generar texto resultante.

En cuanto a la predicción de las imágenes extraídas del texto, tendremos que introducirlas en el predictor de la red neuronal que nos dirá, con más o menos acierto que letra o numero es. Como la red neuronal ha sido entrenada con imagenes de letras centradas y de 64x64, utilizamos la función *rescale* implementada por nosotros la cual adapta en primer lugar el tamaño de la imagen a 64x64 teniendo en cuenta el aspect ratio original de la letra, para evitar deformaciones. Después esta función permite dejar un borde blanco entre la imagen original y el límite de la imagen nueva, para evitar que la imagen se junte con el borde de la imagen nueva permitiendo así una mejor predicción de los datos por parte de la red neuronal.

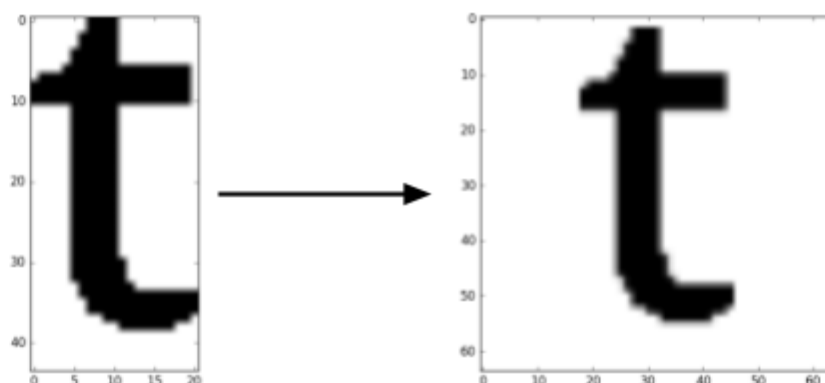


Imagen 1. Ejemplo de imagen transformada utilizando la función rescale.

Results

Non Convolutional Net	Accuracy Regular Dataset
Dense 512-512-128-128-256 Relu + 62 Softmax	62%
Dense 512-64-64-256 Relu + 62 Softmax	70%
Dense 512-512-128-256 Relu + 62 Softmax	71%
Dense 512-128-256 Relu + 62 Softmax	76%
Dense 512-128-128-256 Relu + 62 Softmax	76%
Dense 1024-256-512 Relu + 62 Softmax	79%

Tabla 1. Resultados de diferentes capas sin convolución. Nótese que dense significa que son matrices cuadradas de redes neuronales, por lo tanto si pone 512, en realidad son de 512x512.

Convolutional Net	Accuracy. Regular Dataset	Accuray. Augmented Dataset
Without Augmented data	81%	-
Retrained with Augmented data	84%	67%
Regular retrained twice with Augmented data	84%	67%
Regular trained only with Augmented data	83%	65%
Regular trained twice only with Augmented data	83%	66%

Tabla 2. Resultados de la misma red neuronal convolucional, con diferentes datos de entrenamiento.

Tanto mirando la primera como la segunda tabla se puede observar como las diferencias entre datos de accuracy no son muy elevadas entre ellos, aunque como veremos a continuación, el texto obtenido difiere bastante según la red que se use. Esto por lo tanto es debido a que las letras extraídas del texto, son, por forma, por movimiento o aliasing diferentes a las utilizadas en la red para entrenar y validar los resultados. Por lo que podemos extraer, que hubiese sido más adecuado utilizar un dataset algo más similar y no tan perfecto. Por esta razón, al introducir los datos con data augmentation, hemos obtenido mejores resultados, ya que estos tenían

Project Final Report

movimientos y deformaciones que han permitido a la red neuronal obtener mayor robustez frente a datos menos perfectos.

A continuación encontramos un fragmento del texto ejemplo que hemos ido usando a lo largo de la práctica para verificar los progresos.

Texto original:

Sistemes Multimèdia

Pràctica Sessió 5: Implementació dels Components d MPEG

Jordi Serra
jordi.serra@uab.cat

Objectius:
Els objectius d'aquesta pràctica són:

Texto de la red: Dense 512-64-64-256 Relu + 62 Softmax.

sisteTes MultiTddid
Prhrchca sesshd Imphemenmchd dehs dMPEG 5Tq c0mp0nenB
Jordi serra
jQJdiV3eJJuQuubVcal
Objectius00
Els objectius dlaquesta prdctica s6nTF

Texto de la red convolucional, entrenando con un cuarto del dataset:

SiStemeS Multimedia
PTaCt7Ca sess7o ImphemenZaC7o deLs dMPEG 5r1 G0mpOnenG
JordI Serra
JordI SerrauabCat
ObjectiuSnn
EIS obleCtlus dJaQeSta DraCtlca SOn

Texto de la red convolucional, una vez entrenada con el data augmentation.

Project Final Report*SiStemeS Multimedia**PractiCa SeSSI6 Lmplementac76 deLS dMPEG 5rI CompOneng**JordI Serra**JordI serraQuabcat**ObjeCtiusuu**Els Oblectlus d1aquesta practlca son*

Observando los resultados, podemos observar la evolución positiva que ha supuesto tanto utilizar redes neuronales convolucionales, cómo utilizar data augmentation para aumentar la robustez.

Podemos observar como la predicción es errónea sobretodo en caracteres no entrenados, como puntos y arrobas. Y como otros caracteres no entrenados como los acentos, son solucionados con más o menos acierto.

En cuanto a errores con caracteres entrenados, podemos distinguir de dos tipos. Los primeros serian letras que son iguales tanto en mayúsculas como en minúsculas y que por tanto su única diferencia es el tamaño de la misma. La otra fuente de problemas está en letras que son muy similares entre ellas, como puede ser la i, la t, la f y la l, o la o i el 0. En estas se aprecia una cierta mejora, aunque aún se producen diversos fallos.

Future development

Una vez vistos los resultados, es evidente que hay margen para la mejora, por eso hemos creído conveniente, comentar algunas de las mejoras que creemos que son posibles y que ayudarían a mejorar los datos y la funcionalidad.

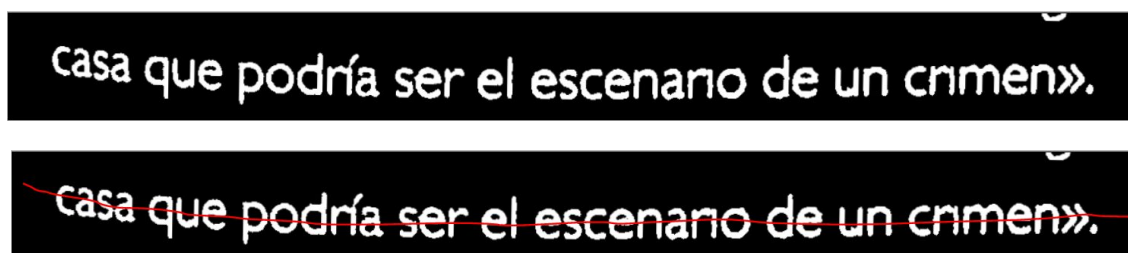
Mejorar la Red Neuronal: En primer lugar, creemos que la capacidad de cómputo nos ha limitado bastante, ya que el tiempo era limitado, y nuestros recursos no muy potentes. Por lo que creemos que aún hay margen de mejora en el apartado de la red neuronal. Tanto realizando más entrenamientos con mas datos, o probando más el data augmentation. Además, apenas hemos cambiado la arquitectura de la red neuronal convolucional, por lo que hay mucho margen de optimización en este apartado.

String Matching, PostProcesado: Una vez extraído el texto, uno de los problemas más comunes que nos encontrábamos son errores en letras que son iguales en mayúsculas y minúsculas o errores con las letras parecidas. Por esta razón, creemos que sería interesante introducir posteriormente a la extracción del texto algún algoritmo de string matching, predicción de palabras o incluso una red de markov, con el objetivo de reducir estos errores fácilmente detectables en medio de otras palabras.

Más símbolos a la hora de hacer el entrenamiento: Otro error muy común en nuestro OCR es problemas a la hora de predecir palabras con acentos o símbolos no entrenados. Por lo tanto sería interesante encontrar un dataset de estos símbolos y utilizarlos también para entrenar la red y poder así predecir mejor los resultados.

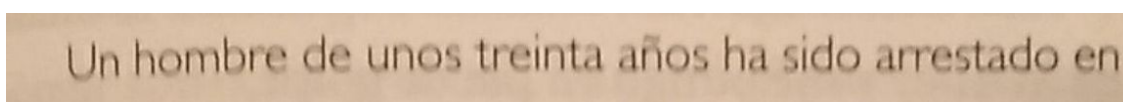
Homografía. En cuanto al pretratamiento de las imágenes, en estos momentos tenemos implementada la posibilidad de rotar el texto en caso de que se encuentre torcido. Esto es muy útil para multitud de situaciones, pero sería muy interesante implementar un algoritmo de **homografía para corregir proyecciones más complejas**, aumentando de esta forma la usabilidad del OCR en multitud de nuevas situaciones.

Nivelar frase: Otra mejora relacionado con lo anterior sería la nivelación horizontal de una línea, para casos dónde la foto no está tomada totalmente recta.



Podríamos tratar de alinear ésta línea en el cuadro de la imagen, para así obtener unas letras menos distorsionadas.

Mejorar Thresholding: El thresholding de Sauvola se puede quedar corto con letras un poco borrosas. Un ejemplo es el siguiente:



Un hombre de unos treinta años ha sido arrestado en

En este caso, podríamos mirar de utilizar morfología para arreglar estos errores, aumentando el peso de la letra.

Skeleton: Finalmente, creemos que sería interesante tratar pasar tanto el texto extraído como las imágenes del dataset por un algoritmo de skeleton que permitiera simplificar las formas de las letras y dejar únicamente la forma real de la letra.

Conclusion

Con tal de obtener datos fiables y de calidad, es necesario un riguroso tratamiento de imágenes. Se ha de tener muchas más cosas en cuenta que la iluminación: grosor de letra, tamaño, *spacing*, orientación, definición, etcétera. Muchos de los problemas se pueden afrontar fácilmente con morfología, pero otros requieren más detenimiento. Un ejemplo a lo anterior ha sido el thresholding. No nos ha sido fácil encontrar una forma óptima para todos los tipos de imágenes, pero sabemos que ahora tampoco lo tenemos.

En cuanto a la red neuronal, hemos ido observado, cómo aunque obtuvieramos buenos valores de accuracy, la red neuronal no era capaz de predecir correctamente los resultados, y como el uso primero de la red convolucional y segundo del data augmentation nos han ayudado a mejorar los resultados.

También nos hemos dado cuenta que unos buenos resultados de accuracy no quieren decir que obtengamos unos buenos resultados a la hora de predecir los datos extraídos, ya que las letras extraídas del OCR son diferentes al dataset “perfecto”, el utilizado por nosotros. Por esto el data augmentation ha sido muy importante para hacer que los datos del dataset, sean más similares a las imágenes reales y así mejorar el rendimiento de nuestro OCR.

Hemos comprobado, pues, que un OCR no es un problema trivial, y que muchísimas cosas pueden hacerse para paliar este problema. Aún así, creemos que un OCR perfecto que se adapte a todos los entornos (lo cual es muy difícil) no sólo tendrá su fuerza en una compleja red neuronal, sino que por igual peso, o incluso más, estará

Project Final Report

dotado de una gran fuerza en el tratamiento y extracción de caracteres en la imagen.

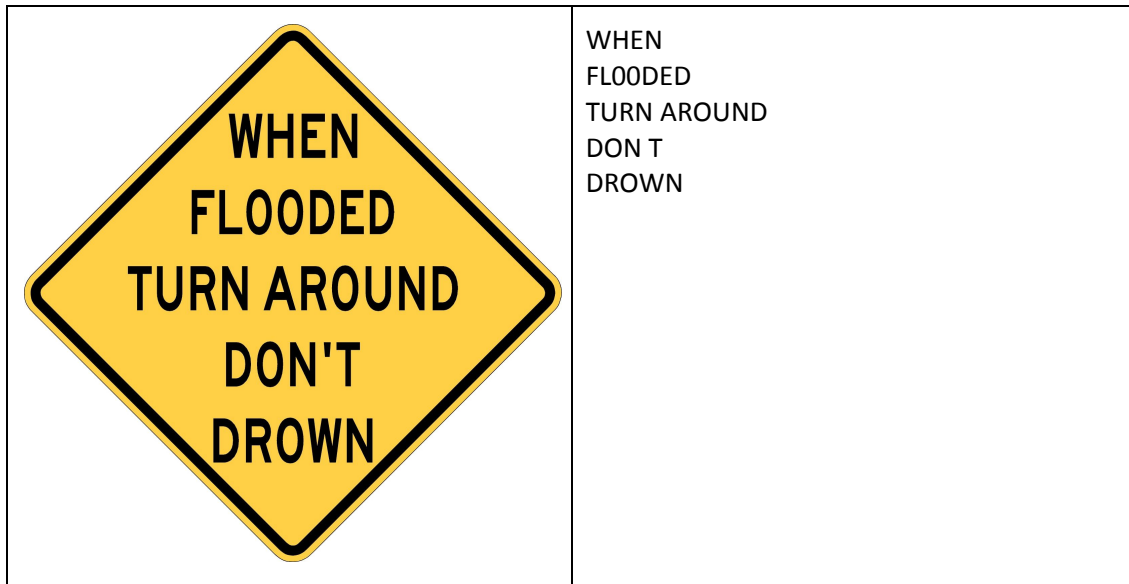
References

1. Theano, visitado 14/05/2017. <http://www.deeplearning.net/software/theano/>
2. Keras, visitado 28/05/2017. <https://keras.io/>
3. Pycuda, visitado 14/05/2017. <https://mathematician.de/software/pycuda/> y descargado de <http://www.lfd.uci.edu/~gohlke/pythonlibs/#pycuda>
4. Lumbreras, Felipe. "Visió per computador". cvc.uab.es , Universitat Autònoma de Barcelona 23/04/2017.
<http://www.cvc.uab.es/shared/teach/a102784/>
5. Cuccioli, Bryan y Amez, Renato. Optical Character Recognition via neural networks, 12 de diciembre de 2012.
<https://github.com/bcuccioli/neural-ocr/blob/master/papers/final.pdf>
6. OpenCV, visitado 14/05/2017. <http://opencv.org/>
7. Letras escritas a máquina, "The Chars74K dataset". Visitado el 14/08/2017.
<http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/>
8. Florian Muellerklein, Chars74k CNN, visitado el 28 de mayo de 2017.
https://github.com/FlorianMuellerklein/Chars74k_CNN
9. Andrej Karpathy y Justin Johnson. Stanford CS231n class notes: Convolutional Neural Networks for Visual Recognition. Visitado el 28/05/2017
<http://cs231n.github.io/neural-networks-3>
10. Fabien Tencé. First place using convolutional neural networks. Visitado el 28/05/2017.
<http://ankivil.com/kaggle-first-steps-with-julia-chars74k-first-place-using-convolutional-neural-networks/>

Annex

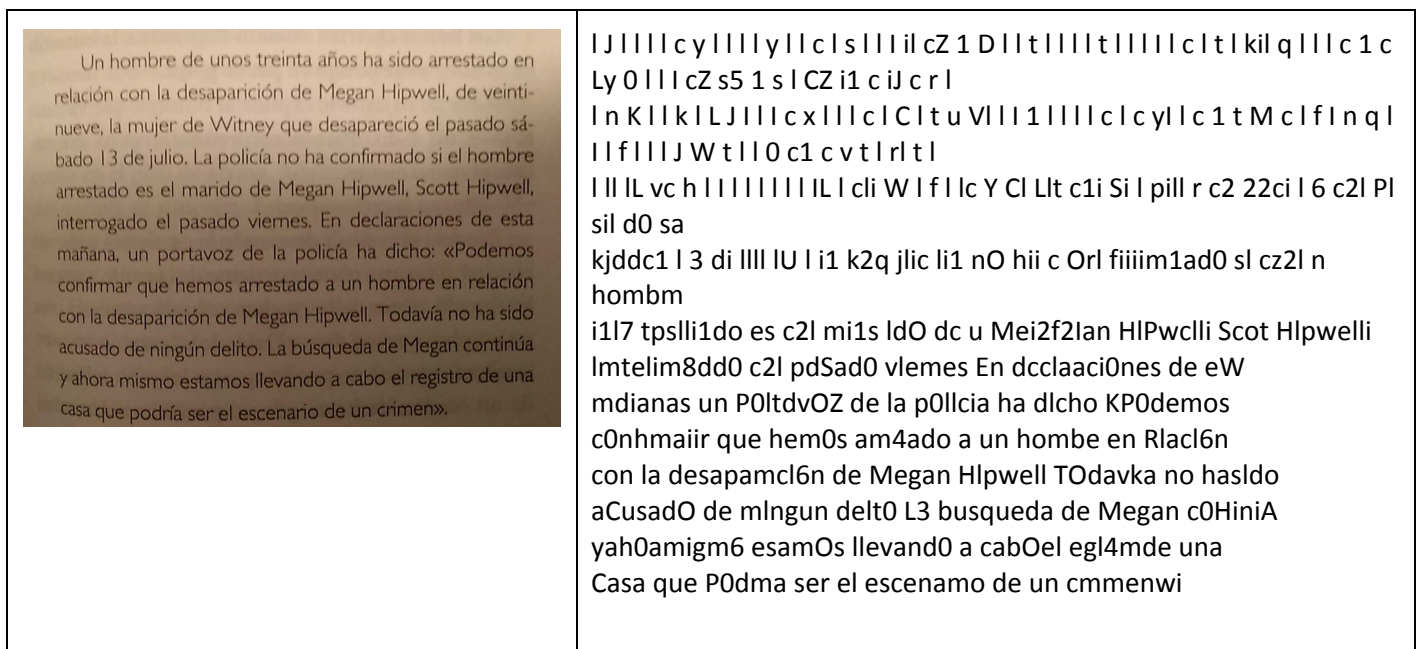
Algunos Resultados:

Resultado 1. Señal de Tráfico.



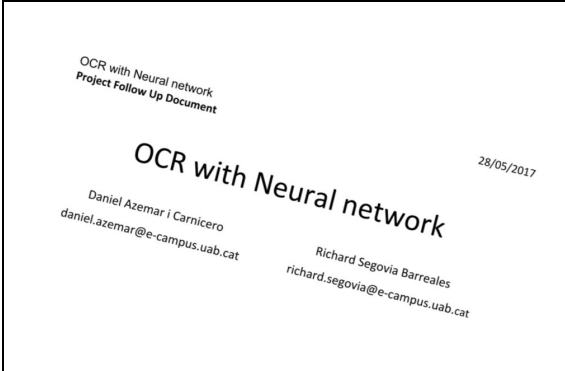
Explicación: El algoritmo ha podido eliminar correctamente el contorno del símbolo.

Resultado 2. Foto Natural sobre la página de un libro.



Explicación: El programa no ha detectado correctamente las primeras líneas debido a que están un poco borrosas. Las últimas, sin embargo, obtiene buenos resultados.

Resultado 3. Texto con diferentes tamaños y rotado.

	<p>28I05I2017 OcRwlth Neural neWoK PrQjedF0II0wUpDwumens OcR With Neural netW0rk Daniel A2emarI Grmicero Rlchard SeB0vla Barreales danlel a2emarQe campus uab cat rlchard seB0vlaeetampus uab cac</p>
---	---

Explicación: El programa ha conseguido girar la imagen correctamente, con lo que obtiene mejores resultados que si no lo hubiese hecho.

Otros datos almacenados:

Hemos guardado tanto los modelos convolucionales como los resultados del gradiente en durante el training y la validación. También hemos guardado los resultados de los textos a lo largo de todo la práctica. Se pueden encontrar en el siguiente enlace:

<https://drive.google.com/open?id=0B3mf8Ba2iBOWb0ZTMm9PcUhjV0E>

Telegram Bot:

Mientras la red neuronal se entrenaba, y en momentos de “procrastinación”, hemos implementado un bot en telegram para que sea más fácil la interactividad con nuestro OCR.

OCR Bot

@vcOCRbot

Lo encontrarás en [telegrambot.py](https://github.com/vicentecorbal/telegrambot.py)

Como que nuestro OCR no está muy preparado para imágenes de un entorno natural (básicamente un documento plano) no es muy útil. Pero es entretenido ver qué predice.

Project Final Report

Nota. El bot no funcionará a no ser que sea activado.

