Resolució d'un Sudoku a partir d'una imatge

David Martí Felip, Jordi Auñon Ansio, Arnau Busquets Domingo

Abstract— En aquest projecte tenim com a objectiu desenvolupar un sistema automàtic per reconèixer i resoldre un sudoku a partir d'una imatge d'entrada. Per aconseguir-ho, utilitzem tècniques estudiades i les seves respectives funcions a OpenCV, i a vegades numpy, per processar la imatge i extreure les caselles del sudoku.

Després de processar la imatge, s'entrena una xarxa neuronal utilitzant les llibreries TensorFlow i Keras per predir els dígits presents a cada casella del sudoku.

Una vegada que els dígits han estat predits i guardats en una matriu de numpy, s'aplica un senzill algoritme de backtracking per resoldre el sudoku. Aquest algoritme permet explorar totes les possibilitats i trobar la solució correcta. Es realitza una validació per assegurar-se que la solució és única i correcta.

Els resultats mostren la importància d'adaptar les tècniques de processament a les circumstàncies en la que es pren la foto, i la dificultat que comporta en entors diferents.

Keywords—Opencv, Image processing, Contour detecting, Tensorflow, Keras, Digit recognition, Backtracking algorithms.

1 INTRODUCCIÓ

Motivats pels programes que cada cop trobem més habitualment als dispositius mòbils i dispositius electrònics que permeten el tracking de diferents parts d'imatges (Live Text ~ APPLE, OCR ~ Samsung, ...) neix la nostra proposta de treball sobre la que treballarem en el desenvolupament d'un sistema de visió per computador en què tractarem de resoldre un Sudoku a partir d'una imatge proporcionada per un usuari.

Principalment, la nostra proposta de projecte es fonamenta en la creació d'un codi el més òptim possible fet a partir de Python i les seves llibreries que ens permeten efectuar les diferents manipulacions que necessitem fer a les diferents imatges.

El procés de proves i test a l'algoritme de detecció té una implementació basada en conjunts/datasets train/test que en alguns casos venen ja implementats dins llibreries que ens ofereix Python i d'altres son de fonts externes.

Objectius proposats:

- Creació d'una senzilla xarxa neuronal entrenada amb datasets d'imatges de nombres impresos per al seu posterior ús.
- Detecció de la secció amb informació rellevant per la resolució del Sudoku
- Reconeixement de caràcters numèrics (impresos) a partir de fotografies mitjançant una xarxa neuronal senzilla emprant eines com tensorflow.
- Implementació d'una interfície a terminal sobre la qual l'usuari podrà passar una direcció (path) d'un arxiu amb un Sudoku i aquesta mostrarà la solució proposada.
- Resolució del Sudoku.
- Implementar un sistema d'avaluació del nostre codi.

Respecte a la fiabilitat d'aquest tipus d'algoritmes n'hi ha diferents sistemes d'avaluació per qualificar el nivell de precisió que presenta el codi com són accuracy, precision, recall, f1score, presentats al llarg de l'assignatura. Nosaltres calcularem l'eficiència de la nostra xarxa neuronal dividint el nombre de prediccions encertades entre el nombre de prediccions totals.

2 ESTAT DE L'ART

2.1 Justin Pinkney (MATLAB)

MATLAB és una bona eina pel processament d'imatge i machine learing, gràcies a l'ample ventall de directives que té relacionades amb el filtratge, la segmentació, l'extracció de característiques... A més, és senzill, mitjançant algorismes, la detecció dels marges, correlació de les imatges, transformacions geomètriques... En definitiva, consta amb tot el que és necessari per abordar aquest projecte.

En aquest cas, Justin Pinkney (investigador de machine learning en Lambda Labs) utilitza aquest llenguatge per abordar aquest projecte de la següent manera:

1. En primer lloc, entrena el programa ensenyant on es troba el requadre del sudoku en un conjunt d'imatges.
2. Un cop entrenat veu si el programa detecta la graella netejant tot el soroll de la imatge, dilatant la màscara que envolta la zona on es troba la graella, ignorant tota la resta de la imatge que no sigui la màscara posant-li de valor 0, i trobant exactament la graella del sudoku que es busca.
3. Utilitzant deep learning digitalitza els nombres manuscrits. Per aconseguir-ho fa ús del conjunt de dades de MATLAB, MNIST.
4. Ja per acabar, un cop ja pot reconèixer els dígits crea la matriu i amb ella ja resol el problema de bon principi, el Sudoku.

2.2 Dmitrii Eliuseev (OpenCV, PyTesseract i PyTorch)

OpenCV i PyTorch ambdues son biblioteques de Python útils per processar imatges i extreure característiques, però en aquest cas, Dmitri Eliuseev (enginyer i científic de dades), ha centrat OpenCV per poder realitzar la part de processament de la imatge i detecció de la cel·la i seguidament amb PyTorch ha pogut llegir els dígits de les imatges.

Després tenim també la biblioteca Tesserarct la qual és utilitzada per llegir text en imatges i convertir-lo en text processable per la computadora.

Dmitrii aborda aquest projecte de la següent manera:

1. Per començar, a partir d'una imatge extreta d'internet crea un model d'un sudoku. Amb aquest model hi aplica filtres per poder especificar com és aquest requadre, per exemple que mai pot ser un rectangle, que sempre ha de tenir 4 costats, que entengui que és un quadrat encara que estigui en perspectiva... Així detecta el requadre en la imatge i el transforma en una

imatge plana i equidistant en tots els punts.
2. Gràcies a l'anterior pas és més còmode detectar els dígits de cada casella, i. utilitzant comandes de Tesseract i un dataset format per dígits de l'1 al 9, el programa entén quin nombre hi ha en cada casella i així la imatge passa a ser una matriu. Amb PyTorch aconsegueix el mateix resultat en 0.01 segons, pel fet que és una biblioteca més centrada en els dígits a diferència de Tesseract centrada més en text.
3. Per acabar, simplement resol la matriu del sudoku amb un algorisme en Python.

3 PROPOSTA

La nostra proposta per a resoldre el repte explicat consta de diferents processos definits. Hem decidit treballar amb llenguatge python per diverses raons. És un llenguatge que, per la nostra trajectòria a la universitat, se'ns fa molt familiar, i conté una gran quantitat de llibreries per a treballar en molts àmbits. Concretament, utilitzarem opencv per al processament d'imatges i tensorflow per a la creació i entrenament de la xarxa neuronal. També farem servir numpy per a algunes de les operacions amb les imatges, ja que en molts casos les tractarem com a matrius, matplotlib per a visualitzar els resultats i altres llibreries més bàsiques per a traslladar les dades.

3.1 Reconeixement de dígits

Per a poder entendre un Sudoku imprès en una imatge és clar que necessitarem ser capaços de reconèixer els diferents dígits que trobem impresos. És per això que hem crearem una xarxa neuronal usant tensorflow. La qual entrenarem amb dades de la base MNIST, concretament la part enfocada a dígits. Es tracta d'un conjunt d'imatges de nombres impresos i manuscrits etiquetades amb el nombre al qual corresponen. A aquesta base de dades hi podem accedir directament des de la llibreria de Tensorflow. Per tant, no hem hagut de descarregar-la i incloure-la dins del nostre directori. També crearem un data set de milers d'imatges creades per nosaltres i modificades amb diferents funcions. Aquest pas ens permetrà entendre els buits com a zeros.

Tenim com a objectiu aconseguir una precisió a prop del 99% d'encerts en les prediccions que ens proporciona la xarxa. Podem modificar certs valors en la xarxa, com per exemple en nombre d'epochs (iteracions) o les capes que afegim al entrenament de la xarxa, per tal d'obtenir un entrenament més fiable de les dades.

3.2 Processament de la imatge

Per a poder obtenir resultats reals en la predicció de nombres, necessitem obtenir imatges que continguin exclusivament els nombres. Això ho aconseguirem mitjançant la detecció de línies i contorns de la imatge. El que volem aconseguir és detectar primerament el quadrat gran que conté les 81 cel·les del Sudoku, per més endavant, separar-les i aconseguir 81 imatges diferents.

Aquestes són les imatges que hem de passar a la nostra xarxa neuronal perquè ens retorni la seva millor predicció. Perquè siguin aptes, cal també modificar-les en mida i tipus d'imatge. En el nostre cas hem fet servir imatges de mida 28x28 per als conjunts traint/test, així que modificarem les imatges obtingudes per tal que estiguin en la mateixa mida.

3.3 Transformació a matriu

Després d'obtenir les imatges processades, cal gestionar les prediccions, fer un control dels resultats i crear una matriu en la que guardar el resultat. En aquest pas, ens assegurarem que les prediccions retornen resultats útils, no generen errors, i omplirem una matriu numpy preparada per a l'últim pas, amb els dígits que identifiqui. Fins aquí arriba la feina que farem relacionada amb el processament de la imatge i la predicció dels nombres. Per a l'últim pas, hi treballarem de manera paral·lela ignorant tot el treball explicat fins ara.

3.4 Resolució del Sudoku.

L'etapa de resolució del Sudoku la podem considerar independent a les altres tres etapes. Aquí crearem una funció principal capaç de retornar la matriu corresponent a un Sudoku solucionat a partir d'una matriu inicial no completa.

Per fer-ho, hem implementat un algoritme de backtracking, concepte après a l'assignatura de Disseny d'Algorismes. Les matrius utilitzades són arrays de numpy, i no hem tingut cap problema especial. El codi d'aquest apartat és a un arxiu exclusiu que importem des del nostre arxiu principal per a fer-ne servir la funció de resolució.

4 EXPERIMENTS, RESULTATS I ANÀLISI

Respecte l'execució del codi hem aconseguit alguns dels objectius plantejats però en altres hem tingut dificultats.

La detecció del requadre que forma el sudoku funciona correctament ja que és senzill poder detectar el quadrat més gran que hi ha a la imatge, però el tractament de la imatge ja retallada per trobar les cel·les és més complexa ja que hem de crear una llista ordenada de totes les cel·les que hi ha i amb això poder seguidament detectar cada dígit que forma el sudoku.

Un cop processada tota la imatge i obtingut el que desitjem la resolució del sudoku funciona correctament i sense problema.

4.1 Detecció del contorn exterior

Abans de poder detectar el contorn exterior ha calgut filtrar la imatge de tal manera que poguesim reduir el soroll de la mateixa.

Per aconseguir això s'ha passat la imatge a escala de grisos, s'ha aplicat un filtre Gaussià i per últim s'ha fet una umbralització per així binaritzar la imatge mitjançant el métode de thresholding *Otsu.*

Ja reduit el soroll de la imatge, amb la detecció de *Canny* i altres funcions de la llibreria *cv2*, s'ha pogut detectar tots els contorns de la imatge i amb ells els hem pogut ordenar de més grans a més petit, quedar-nos amb els 10 més grans (per així asegurar que el quadrat estigui dins d'aquestes 10 opcions), i seguidament crear uns perímetres amb els contorns veïns formant així el polígon que busquem.

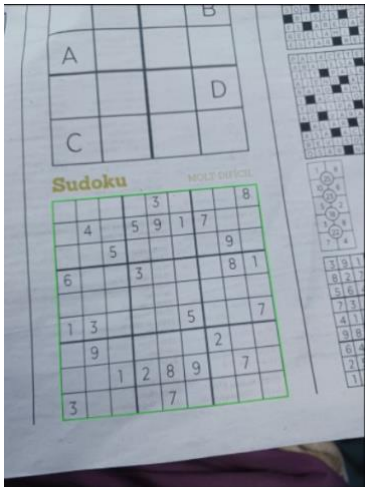


Fig. 1. Detecció del sudoku en la imatge original, gràcies a la detecció de Canny i el processament de desrés.

Com es pot veure en la Figura 1 el quadrat detectat és el del sudoku, el polígon amb més perímetre de la imatge.

4.2 Homografia de la graella

Un cop detectada la graella cal aplanar-la i així totes les cel·les son equidistants amb les seves veïnes.

Per aconseguir això primer s'ha creat un rectangle al voltant de la imatge inicial, així saber amb quins marges caldrà aplanar el requadra. Seguidament calcular la matriu de transformació de la presepctiva a la imatge inicial. Finalment aplicar aquesta transformació a la imatge inicial.

4.3 Detecció de les cel·les

Després de les funcions del sistema de detecció de quadricula ara cal retallar la matriu del *Sudoku* en les 81 cel·les.

Com la imatge processada de la graella és plana, i per tant equidistants les cel·les amb les seves veïnes, retallem l'imatge en 81 rectangles amb la mateixa àrea i guardem totes les imatges en una llista.

4.4 Detecció dels dígits

Per a la detecció dels dígits, hem usat una xarxa neuronal creada amb Tensorflow a la que hem aplicat quatre capes. Una d'elles per asegurar-nos de que la mida d'entarda de les imatges que volem fer servir es de 28x28, mida molt petita però suficient per a fer les proves i identificar-hi dígits, hem usat aquesta mida ja que és la mida de les imatges de la base de dades MNIST. Dues més per al propi entrenament de les dades i una última de mida 10, per a clasificar en els 10 possibles dígits.

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)

#només cal executar una vegada:

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=4)

model.save('digits.model')
```

Fig. 2. Creació de la xarxa neuronal.

Amb aquesta xarxa neuronal, hem conseguit excelents resultats, quan fem servir els nombres MNIST. (més del 98% d'encert).

```
Epoch 1/4
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2630 - accuracy: 0.9229
Epoch 2/4
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1075 - accuracy: 0.9668
Epoch 3/4
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0724 - accuracy: 0.9764
Epoch 4/4
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0506 - accuracy: 0.9836
```

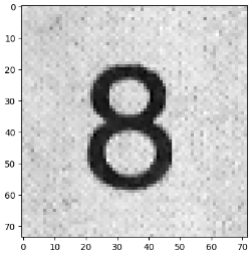
Fig. 3. Resultats del testing de la base

En canvi, els resultats són molt pitjors quan passem les cel·les retallades dels sudokus. És per això que, a part de millorar el processament de les imatges i la manera en que retallem les cel·les, també hem creat un altre dataset per a entrenar-la.

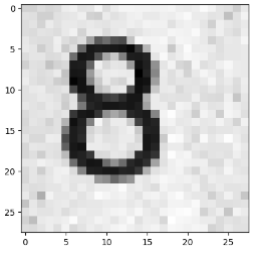
Si bé la configuración de la xarxa neuronal és la mostrada, hem fet noves imatges a partir de uns exemples de sudoku i 4 funcions per a modificar-les. D'aquesta manera hem creat fins a 5000 imatges noves de les quals hem dedicat 4500 a entrenar la xarxa neuronal i 500 les hem guardat per a comprobar-ne el seu encert. El qual arriba fins a més del 98 per cent.

Per a crear aquest dataset hem usat 4 funcions diferents que modifiquen les imatges. Hem aplicat el retall per a totes les imatges creades i les altres tres les hem aplicat de manera aleatoria. Per a mostrar-ho, la primera imatge de l'àlbum original i com queda després d'aplicar manualment cada funció individualment.

***Imatge original*:**

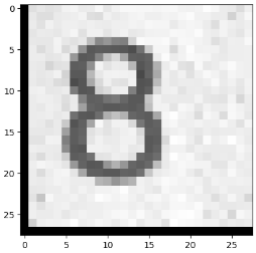


Després de retallar aleatòriament (Funció 1):



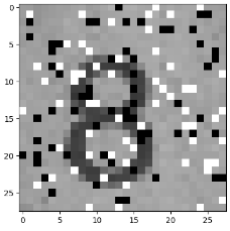
Vejem que l'imatge ha estat desplaçada cap a l'esquerra i sutilment alçada.

Després d'afegir contorns (Funció 1 i 2):



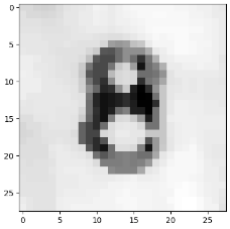
Vejem que han estat insertats contorns a l'esquerra i a la part Baixa de la imatge.

Després d'afegir soroll (Funció 1 i 3):



Vejem que alguns íxels han estat modificats aleatòriament i han canviat el seu color a un blanc o negre absolut.

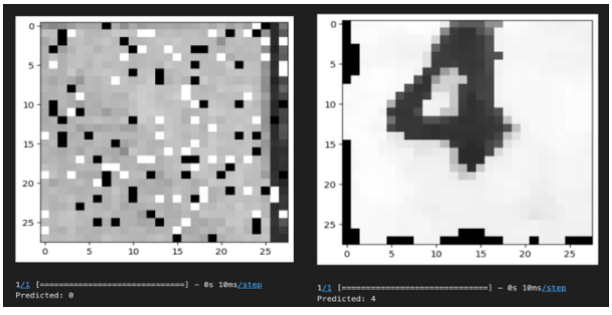
Després de difuminar l'imatge (Funció 1 i 4):



Vejem com després d'aplicar un filtre de mediana, el nombre es veu amb més dificultat.

També hem pres la decisió de esborrar les imatges de l'MNIST per a l'entrenament de la xarxa, nombres manuscrits que vam confiar en que funcionarien al plantejament del problema però hem vist que distorsionen el nostre entrenament, empitjorat la precisió. Això és en gran part perquè les dades MNIST entrenen la xarxa amb el nombre 0 ben dibuixat. En canvi, nosaltres entenem els buits com a zeros i així ho hem detallat al dataset per a entrenar la xarxa.

A continuació, les imatges despres de ser tractades per les anteriors funcions, i la predicció encertada de la xarxa neuronal:



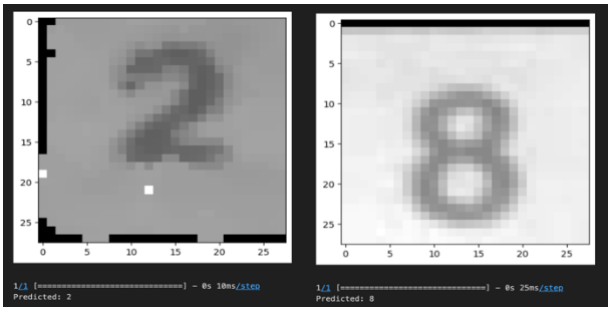


Fig. 3. Alguns exemples de les prediccions

4.5 Resolució del sudoku

La resolución del sudoku, com hem explicat anteriorment, és un apartat extern al processament de la imatge on només rep una matriu de numpy i aplicant un algorisme el resol.

Aquest algorisme funciona correctamente, demostrat gràcies a una funció que hem implementat, la qual comprova que cada posició sigui valida dins el sudoku, és a dir, que si la matriu que rep per paràmetre té algún dígit incorrecte, ja que té algun error en la lectura dels dígits i es repeteix algún valor en alguna fila, columna o graella, doni error, i en cas contrari que resolgui el sudoku i retorni aquesta matriu correcte.

5 CONCLUSIONS

Creiem que hem après molt en aquest projecte, tot i que no consegueix sempre el resultat esperat, ja que per a solucionar un sudoku hauria d'encertar 81 vegades la predicció, i la precissió que hem aconseguit ronda el 98 per cent. Considerem que hem emprat tècniques i eines apreses en l'assignatura, com l'aplicació de filtres gaussians i thresholds per reduïr el soroll o la detecció de Canny.

També cal aclarir que el problema de la detecció del sudoku apartir d'una imatge ens ha resultat senzill de poder buscar informació ja que en diverses pàgines hi ha bastanta informació, però no en el nivell de processament que nosaltres buscavem, ja que en varis casos la persona qui resolia el problema es centraba en ensenyar el codi I no explicar el perquè i això ens ha dificultat a l'hora de entendre com s'ha de fer cada part.

Sobre el llenguatge estem molt satisfets d'haver triat python i les llibreries OpenCV I Tensorflow ja que hem trobat tot tipus de funcions per poder, tant simplificar el codi, com fer exactament ampliar els passos per processar la imatge, com és el cas de el preprocessament necessari abans de poder començar a tractar amb el sudoku com a tal.

En conclusió, creiem que hem fet una gran feina en tots els àmbits del treball, i deixem com a possibles millores l'adaptació a escenaris que no hem treballat com Sudokus en paper arrugat, mullat, etc. Així com la maximització de l'efectivitat de la xarxa neuronal fins al 100% dels casos.

BIBLIOGRAFIA

[1] Raqueeb Shaik. Jun 22, 2020. "OpenCV (findContours) Detailed Guide" https://medium.com/analytics-vidhya/opencv-findcontours-detailed-guide-692ee19eeb18

[2] Akshay Gupta. May 26, 2021. "Solving Sudoku from Image Using Deep Learning – With Python Code" https://www.analyticsvidhya.com/blog/2021/05/solving-sudoku-from-image-using-deep-learning-with-python-code/

[3] NeuralNine. Aug 20, 2021. "Neural Network Python Project – Handwritten Digit Recognition" https://www.youtube.com/watch?v=bte8Er0QhDg&ab_channel=NeuralNine

[4] ProgramarFacil. Sep 17, 2020. "Detector de bordes Canny cómo contar objetos con OpenCV y Python" https://programarfacil.com/blog/vision-artificial/detector-de-bordes-canny-opencv/

[5] Mathworks. Nov 15, 2018. "Sudoku Solver: Image Processing and Deep Learning" https://blogs.mathworks.com/deep-learning/2018/11/15/sudoku-solver-image-processing-and-deep-learning/

[6] Mathworks. May 17, 2022. "Solving Sudoku in real-time using a Convolutional Neural Network and OpenCV" https://blog.devgenius.io/solving-sudoku-in-real-time-using-a-convolutional-neural-network-and-opencv-e47a92478dce

[7] UCM. Sept 18, 2013. "Reconocimiento de imagenes usando redes neuronales artificiales" https://eprints.ucm.es/23444/1/ProyectoFinMasterPedroPablo.pdf