Deocamdată, ne vom concentra în primul rând pe modul de instruire a unui model Keras / TensorFlow personalizat pentru a recunoaște caractere alfanumerice (adică cifrele 0-9 și literele A-Z).

Pe baza postării de astăzi, săptămâna viitoare vom afla cum putem folosi acest model pentru a clasifica corect caracterele scrise de mână în imagini de intrare personalizate.

Scopul acestei serii din două părți este de a obține o înțelegere mai profundă a modului în care învățarea profundă este aplicată clasificării scrisului de mână și, mai precis, obiectivul nostru este de a:

Familiarizați-vă cu unele seturi de date de scriere de mână bine cunoscute, ușor disponibile atât pentru cifre, cât și pentru litere

Înțelegeți cum să formați modelul de învățare profundă pentru a recunoaște cifre și litere scrise de mână

Câștigați experiență în aplicarea modelului nostru personalizat la câteva exemple de date din lumea reală

Înțelegeți unele dintre provocări cu datele zgomotoase din lumea reală și cum am putea dori să mărim seturile noastre de date de scriere de mână pentru a ne îmbunătăți modelul și rezultatele

Vom începe cu elementele fundamentale ale utilizării unor seturi de date de scriere de mână bine cunoscute și vom instrui un model de învățare profundă ResNet cu privire la aceste date.

Pentru a ne pregăti modelul personalizat Keras și TensorFlow, vom folosi două seturi de date:

Setul de date standard MNIST 0-9 de LeCun și colab.

Setul de date Kaggle A-Z de Sachin Patel, bazat pe baza de date specială NIST 19

Setul de date standard MNIST este încorporat în cadrul popular de învățare profundă, inclusiv Keras, TensorFlow, PyTorch etc. Un eșantion al setului de date MNIST 0-9 poate fi văzut în Figura 1 (stânga). Setul de date MNIST ne va permite să recunoaștem cifrele 0-9. Fiecare dintre aceste cifre este conținută într-o imagine în nuanțe de gri de 28 x 28. Puteți citi mai multe despre MNIST aici.

Dar ce zici de literele A-Z? Setul de date standard MNIST nu include exemple de caractere A-Z - cum le vom recunoaște?

Răspunsul este să folosiți baza de date specială NIST 19, care include caractere A-Z. Acest set de date acoperă de fapt 62 de caractere hexazecimale ASCII corespunzătoare cifrelor 0-9, literelor majuscule A-Z și literelor minuscule a-z.

Pentru a face setul de date mai ușor de utilizat, utilizatorul Kaggle Sachin Patel a lansat setul de date într-un fișier CSV ușor de utilizat. Acest set de date ia literele majuscule A-Z din baza de date specială 19 NIST și le redimensionează la 28 x 28 pixeli în tonuri de gri pentru a fi în același format cu datele noastre MNIST.

Pentru acest proiect, vom folosi doar setul de date Kaggle A-Z, ceea ce va face ca preprocesarea noastră să fie o briză. Un eșantion din acesta poate fi văzut în Figura 1 (dreapta).

Vom implementa metode și utilitare care ne vor permite:

Încărcați ambele seturi de date pentru cifrele MNIST 0-9 și literele Kaggle A-Z de pe disc

Combinați aceste seturi de date împreună într-un singur set de date de caractere unificat

Manevrați înclinarea / dezechilibrul etichetei clasei de la a avea un număr diferit de probe pe caracter

Instruiți cu succes un model Keras și TensorFlow pe setul de date combinat

Trasați rezultatele antrenamentului și vizualizați rezultatele datelor de validare.

//

Pentru a instrui modelul nostru personalizat Keras și TensorFlow OCR, trebuie mai întâi să implementăm două utilitare de ajutor care ne vor permite să încărcăm atât seturile de date Kaggle A-Z, cât și cifrele MNIST 0-9 de pe disc.

Aceste funcții de asistență I / O sunt denumite în mod corespunzător:

load\_az\_dataset: pentru literele Kaggle A-Z

load\_mnist\_dataset: pentru cifrele MNIST 0-9

Acestea pot fi găsite în fișierul helpers.py al submodulelor az\_dataset ale pyimagesearch.

Explicatie cod:

Funcția noastră load\_az\_dataset preia un singur argument datasetPath, care este locația fișierului CSV Kaggle A-Z (Linia 5). Apoi, inițializăm matricele noastre pentru a stoca datele și etichetele (liniile 7 și 8).

Fiecare rând din fișierul CSV al lui Sachin Patel conține 785 coloane - o coloană pentru eticheta clasei (adică „A-Z”) plus 784 coloane care corespund celor 28 x 28 pixeli în tonuri de gri. Să analizăm.

Începând de la Linia 11, vom parcurge fiecare rând al fișierului CSV și vom analiza eticheta și imaginea asociată. Linia 14 analizează eticheta, care va fi eticheta întreagă asociată cu litera A-Z. De exemplu, litera „A” are o etichetă corespunzătoare numărului întreg „0”, iar litera „Z” are o valoare a etichetei întregi „25”.

Apoi, Linia 15 analizează imaginea noastră și o aruncă ca o matrice NumPy de numere întregi de 8 biți nesemnate, care corespund valorilor în tonuri de gri pentru fiecare pixel din [0, 255].

Ne remodelăm imaginea (Linia 20) dintr-o matrice plană 784-dimensională într-una care este 28 x 28, corespunzătoare dimensiunilor fiecăreia dintre imaginile noastre.

Apoi vom adăuga fiecare imagine și etichetă la datele noastre și, respectiv, la matricile de etichete (liniile 23 și 24).

Pentru a finaliza această funcție, vom converti datele și etichetele în tablouri NumPy și vom returna datele și etichetele imaginii:

În prezent, datele și etichetele noastre de imagine sunt doar liste Python, așa că vom tasta cast-le ca matrice NumPy de float32 și respectiv int (Liniile 27 și 28).

Următoarea noastră funcție de asistență I/O, load\_mnist\_dataset, este considerabil mai simplă.

Linia 33 încarcă datele MNIST de 0-9 cifre folosind funcția de ajutor Keras, mnist.load\_data. Observați că nu trebuie să specificăm un set de date Path, așa cum am făcut pentru datele Kaggle, deoarece Keras, în mod convenabil, are acest set de date încorporat.

Mnist.load\_data de la Keras vine cu o împărțire implicită pentru date de antrenament, etichete de antrenament, date de testare și etichete de testare. Deocamdată, vom combina datele de testare și de testare pentru MNIST folosind np.vstack pentru datele noastre de imagine (Linia 38) și np.hstack pentru etichetele noastre (Linia 39).

Mai târziu, în train\_ocr\_model.py, vom combina datele noastre de 0-9 cifre MNIST cu literele Kaggle A-Z. În acel moment, vom crea propria noastră împărțire personalizată de date de testare și instruire.

În cele din urmă, Linia 42 returnează datele imaginii și etichetele asociate funcției de apelare.

Felicitări! Ați finalizat acum funcțiile de asistență I / O pentru a încărca atât eșantioanele de cifre, cât și cele de litere pentru a fi utilizate pentru OCR și învățare profundă. Apoi, vom examina fișierul nostru principal de driver utilizat pentru instruire și vizualizarea rezultatelor.

/// final I/O

În această secțiune, vom instrui modelul nostru OCR folosind Keras, TensorFlow și o implementare PyImageSearch a arhitecturii de învățare profundă foarte populară și de succes, ResNet.

Pentru a începe, localizați fișierul nostru driver principal, train\_ocr\_model.py, care se găsește în directorul principal, ocr-keras-tensorflow /. Acest fișier conține o referință la un fișier resnet.py, care se află în modelele / sub-directorul de sub modulul pyimagesearch.

Să luăm un moment să examinăm train\_ocr\_model.py. După aceea, ne vom întoarce și o vom descompune, pas cu pas.

Aceasta este o listă lungă de declarații de import, dar nu vă faceți griji. Înseamnă că avem o mulțime de pachete care au fost deja scrise pentru a ne face viața mult mai ușoară.

Începând cu Linia 5, vom importa matplotlib și vom configura backend-ul acesteia, scriind rezultatele într-un fișier folosind matplotlib.use („Agg”) (Linia 6).

Avem apoi câteva importuri din modulul nostru personalizat de căutare pyimages pentru arhitectura noastră de învățare profundă și funcțiile de asistență I / O pe care tocmai le-am analizat:

Importăm ResNet din pyimagesearch.model, care conține propria noastră implementare personalizată a popularei arhitecturi de învățare profundă ResNet (Linia 9).

Apoi, importăm funcțiile noastre de asistență I/O load\_mnist\_data (Linia 10) și load\_az\_dataset (Linia 11) din pyimagesearch.az\_dataset.

Avem câteva importuri din modulul Keras al TensorFlow, care simplifică mult creșterea și formarea noastră a datelor:

Linia 12 importă ImageDataGenerator pentru a ne ajuta să ne mărim în mod eficient setul de date.

Apoi importăm SGD, popularul algoritm de optimizare Stochastic Gradient Descent (SGD) (Linia 13).

În continuare, importăm trei funcții de asistență din scikit-learn pentru a ne ajuta să etichetăm datele noastre, să împărțim seturile de date de testare și instruire și să imprimăm un raport de clasificare frumos pentru a ne arăta rezultatele:

Pentru a converti etichetele noastre din numere întregi într-un vector în ceea ce se numește codare cu un singur hot, importăm LabelBinarizer (Linia 14).

Pentru a ne ajuta să împărțim cu ușurință seturile noastre de date de testare și instruire, importăm train\_test\_split din scikit-learn (Linia 15).

Din submodulul de valori, importăm classification\_report pentru a imprima un raport de clasificare frumos formatat (linia 16).

În continuare, vom folosi un pachet personalizat pe care l-am scris numit imutils.

Din imutils, importăm build\_montages pentru a ne ajuta să construim un montaj dintr-o listă de imagini (Linia 17). Pentru mai multe informații despre construirea montajelor, vă rugăm să consultați tutorialul Montaje cu OpenCV.

Vom importa în cele din urmă Matplotlib (Linia 18) și OpenCV (Linia 21).

Acum, să examinăm cele trei argumente ale liniei de comandă:

Avem trei argumente de analizat:

--az: Calea către setul de date Kaggle A-Z (Liniile 25 și 26)

--model: Calea către ieșirea modelului instruit de recunoaștere a scrisului de mână (Liniile 27 și 28)

--plot: Calea către ieșirea fișierului istoric de antrenament (Liniile 29 și 30)

Până în prezent, avem importurile, funcția de comoditate și argumentele pentru linia de comandă gata de pornire. Mai avem câțiva pași pentru a configura instruirea pentru ResNet, a o compila și a o antrena.

Acum, vom seta parametrii de instruire pentru ResNet și vom încărca datele cifrelor și literelor folosind funcțiile de asistență pe care le-am analizat deja:

Liniile 35-37 inițializează parametrii pentru instruirea modelului nostru ResNet.

Apoi, încărcăm datele și etichetele pentru datele Kaggle A-Z și MNIST 0-9 cifre, respectiv (Liniile 41 și 42), folosind funcțiile de asistență I / O pe care le-am analizat la începutul postării.

În continuare, vom efectua o serie de pași pentru a ne pregăti datele și etichetele pentru a fi compatibili cu modelul nostru de învățare profundă ResNet din Keras și TensorFlow:

Pe măsură ce combinăm literele și numerele într-un set de date cu un singur caracter, dorim să eliminăm orice ambiguitate în care există suprapuneri în etichete, astfel încât fiecare etichetă din setul de caractere combinat să fie unică.

În prezent, etichetele noastre pentru A-Z merg de la [0, 25], corespunzătoare fiecărei litere a alfabetului. Etichetele cifrelor noastre merg de la 0 la 9, deci există o suprapunere - ceea ce ar fi o problemă dacă ar fi să le combinăm direct.

Nici o problema! Există o soluție foarte simplă. Vom adăuga doar zece la toate etichetele noastre A-Z, astfel încât toate să aibă valori întregi mai mari decât valorile etichetei cifrelor noastre (Linia 47). Acum, avem o schemă de etichetare unificată pentru cifrele 0-9 și literele A-Z fără nicio suprapunere în valorile etichetelor.

Linia 50 combină seturile noastre de date pentru cifre și litere într-un set de date cu un singur caracter folosind np.vstack. De asemenea, Linia 51 unifică etichetele noastre corespunzătoare pentru cifrele și literele noastre folosind np.hstack.

Arhitectura noastră ResNet necesită ca imaginile să aibă dimensiuni de intrare de 32 x 32, dar imaginile noastre de intrare au în prezent dimensiunea de 28 x 28. Redimensionăm fiecare dintre imagini folosind cv2.resize (Linia 56).

Avem doi pași finali pentru a ne pregăti datele pentru utilizare cu ResNet. Pe Linia 61, vom adăuga o dimensiune „canal” suplimentară la fiecare imagine din setul de date pentru ao face compatibilă cu modelul ResNet din Keras / TensorFlow. În cele din urmă, ne vom scala intensitățile pixelilor de la un interval de [0, 255] până la [0,0, 1,0] (Linia 62).

Următorul nostru pas este să pregătim etichetele pentru ResNet, să cântărim etichetele pentru a ține cont de înclinarea în numărul de ori în care fiecare clasă (caracter) este reprezentată în date și să împărțim datele în fracțiuni de testare și antrenament:

Instanțiem un LabelBinarizer (Linia 65), iar apoi convertim etichetele din numere întregi într-un vector de binare cu o codare cu un singur hot (Linia 66) folosind le.fit\_transform. Liniile 70-75 cântăresc fiecare clasă, pe baza frecvenței de apariție a fiecărui caracter. Apoi, vom folosi utilitarul scikit-learn train\_test\_split (Liniile 79 și 80) pentru a partiția datele în formare de 80% și testare de 20%.

De acolo, ne vom mări datele folosind un generator de imagini de la Keras:

putem îmbunătăți rezultatele clasificatorului nostru ResNet prin mărirea datelor de intrare pentru instruire folosind un ImageDataGenerator. Liniile 82-90 includ diverse rotații, redimensionarea dimensiunii, traduceri orizontale, traduceri verticale și înclinări în imagini. Pentru mai multe detalii despre mărirea datelor, consultați tutorialul nostru Keras ImageDataGenerator și Data Augmentation.

Acum suntem gata să inițializăm și să compilăm rețeaua ResNet:

Folosind optimizatorul SGD și un program standard de decădere a ratei de învățare, ne construim arhitectura ResNet (Liniile 94-96). Fiecare caracter / cifră este reprezentat ca o imagine în tonuri de gri de 32 × 32 pixeli, așa cum este evident prin primii trei parametri ai metodei de construire ResNet.

Notă: Pentru mai multe detalii despre ResNet, asigurați-vă că consultați pachetul Practitioner of Deep Learning pentru computer Vision cu Python, unde veți afla cum să implementați și să reglați arhitectura puternică.

Liniile 97 și 98 compilează modelul nostru cu pierderea „categorical\_crossentropy” și optimizatorul SGD stabilit. Vă rugăm să aveți grijă că, dacă lucrați cu un set de date cu doar 2 clase (noi nu suntem), va trebui să utilizați funcția de pierdere „binary\_crossentropy”.

Apoi, vom instrui rețeaua, vom defini numele etichetelor și vom evalua performanța rețelei:

Ne instruim modelul folosind metoda model.fit (Liniile 102-108). Parametrii sunt după cum urmează:

flux august: stabilește creșterea datelor în linie (Linia 103)

validation\_data: testați imaginile de intrare (testX) și etichetele de testare (testY) (Linia 104)

steps\_per\_epoch: câte loturi sunt executate pe fiecare trecere a datelor complete de antrenament (Linia 105)

epoci: numărul de treceri complete prin setul complet de date în timpul antrenamentului (Linia 106)

class\_weight: greutăți din cauza dezechilibrului eșantioanelor de date pentru diferite clase (de exemplu, cifre și litere) în datele de antrenament (Linia 107)

detaliat: arată o bară de progres în timpul antrenamentului (Linia 108)

Notă: În trecut, TensorFlow / Keras solicita utilizarea unei metode numite .fit\_generator pentru a antrena un model folosind generatoare de date (cum ar fi obiecte de mărire a datelor). Acum, metoda .fit poate face față și generatoarelor / măririi datelor, făcând un cod mai consistent. Acest lucru se aplică și migrării de la .predict\_generator la .predict. Asigurați-vă că verificați articolele mele despre fit și fit\_generator, precum și despre mărirea datelor.

Apoi, stabilim etichete pentru fiecare personaj individual. Liniile 111-113 concatenează toate cifrele și literele noastre și formează o matrice în care fiecare membru al matricei este o singură cifră sau număr.

Pentru a ne evalua modelul, facem predicții asupra setului de testare și imprimăm raportul nostru de clasificare. Raportul îl vom vedea foarte curând în secțiunea următoare!

Linia 118 imprimă rezultatele folosind utilitarul convenabil scikit-learn classification\_report.

Vom salva modelul pe disc, vom grafica rezultatele istoricului antrenamentelor și vom salva istoricul antrenamentelor:

Pe măsură ce ne-am terminat antrenamentul, trebuie să salvăm modelul format din arhitectură și greutăți finale. Vom salva modelul nostru pe disc, ca fișier Hierarchical Data Format versiunea 5 (HDF5), care este specificat de save\_format (Linia 123).

Apoi, folosim plt-ul matplotlib pentru a genera un grafic liniar pentru pierderea antrenamentului și pierderea setului de validare împreună cu titluri, etichete pentru axe și o legendă. Datele pentru pierderile de antrenament și validare provin din istoria lui H, rezultatele model.fit de sus cu un punct pentru fiecare epocă (Liniile 127-134). Graficul curbelor de pierdere de antrenament este salvat în plot.png (Linia 135).

În cele din urmă, să codificăm procedura noastră de vizualizare, astfel încât să putem vedea dacă modelul nostru funcționează sau nu:

Linia 138 inițializează matricea noastră de imagini de testare.

Începând de pe linia 141, selectăm aleatoriu 49 de caractere (pentru a forma o grilă 7 × 7) și trecem la:

Clasificați personajul folosind modelul nostru bazat pe ResNet (Liniile 143-145)

Prindeți imaginea caracterului individual din datele noastre de testare (Linia 149)

Setați o culoare a textului adnotării ca verde (corect) sau roșu (incorect) prin liniile 150-154

Creați o reprezentare RGB a imaginii noastre cu un singur canal și redimensionați-o pentru a fi inclusă în montajul nostru de vizualizare (Liniile 159 și 160)

Adnotați eticheta text colorată (liniile 161 și 162)

Adăugați imaginea la matricea noastră de imagini de ieșire (Linia 165)

Pentru a închide, asamblăm fiecare imagine de caracter adnotată într-o grilă de vizualizare OpenCV Montage, afișând rezultatul până când este apăsată o tastă (Liniile 168-172).