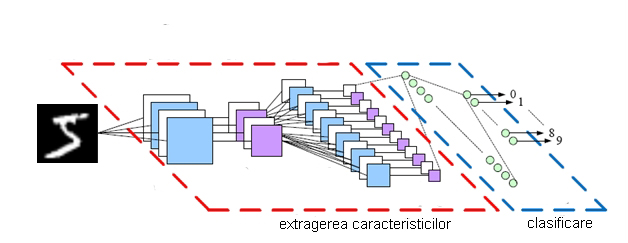
Documentație aplicație Digit Classifier

# **Inteligență artificială**

Pe măsură ce tehnologia evoluează, noile descoperiri vizează încercarea de a facilita experiența utilizatorilor. Astfel, modalități pentru a realiza o comunicare cât mai eficientă între om și dispozitivele care îl înconjoară, sunt în continuă căutare. Evident, realizarea unei astfel de comunicări reprezintă o problemă deosebit de dificilă, iar o aplicație capabilă să recunoască cifre scrise de mână este un pas în direcția bună. Ideea are la bază termenul de *Deep Learning* care, la rândul lui, se bazează pe foarte multe calcule matematice. Din fericire, procesoarele moderne sunt capabile să efectueze un număr impresionant de calcule, iar, în consecință, tehnologia va fi soluția multor probleme din ce în ce mai complexe.

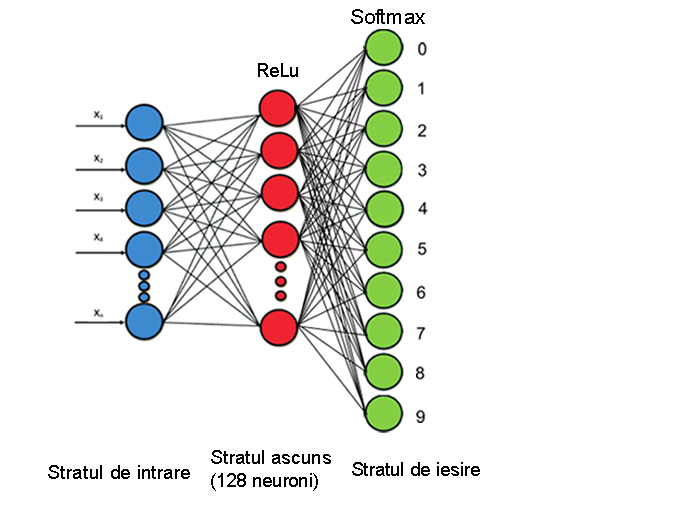
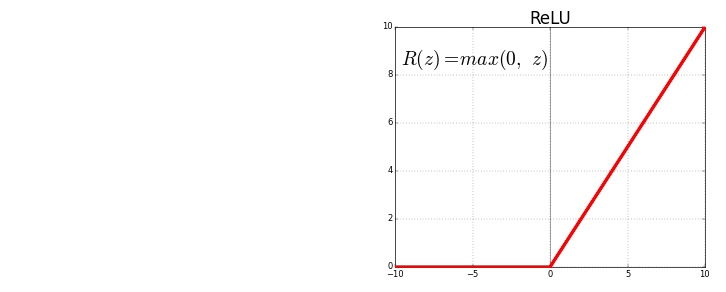
# **Prezentare generală**

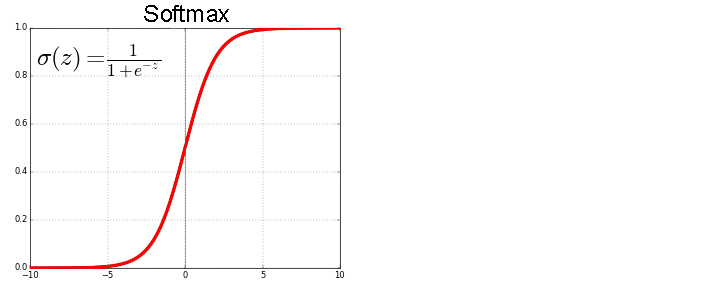
Programul este scris în Python 3 și folosește bibliotecile Keras, Tensorflow și Numpy pentru a crea și folosi ceea ce se numește o rețea neuronală convolutivă (*Convolutional Neural Network*) care clasifică cifre scrise de mână. Algoritmul folosește o bază de date de 10.000 de imagini alb-negru (*MNIST dataset*) pentru a găsi o generalizare aplicabilă datelor noi, reușind să identifice caracteristicile principale ale cifrelor și, chiar daca toată lumea scrie diferit, să le recunoască cu o acuratețe de peste 99%. Procesul are loc în două etape:

* Extragerea caracteristicilor, în care are loc preprocesarea imaginilor și ulterior crearea unor mape de caracteristici (modelul folosit repeta acest proces de 3 ori și creează mape cu 32, 64 și respectiv 128 de caracteristici)
* Clasificarea, în care ultima mapă de caracteristici este transformată în date de intrare pentru rețeaua neuronală propriu-zisă și rezultatul final este generat în ultimul strat de neuroni din rețea

# **Etapa Clasificării**

Aplicația devine capabilă să recunoască cifrele scrise de mână doar în urma unui proces de învățare care se folosește de imaginile din baza de date MNIST. În fișierele *CNN\_mnist*, *CNN\_mnist\_best* și *CNN\_mnist\_best\_improve* se găsesc algoritmi care încearcă să creeze o rețea neuronală cu o acuratețe cât mai mare, modificând parametrii acesteia. Primul algoritm determină, printre altele, că un model cu 128 de neuroni aranjați într-un singur strat ascuns este optim pentru această problemă. Cel de-al doilea algoritm se folosește de datele furnizate de către primul și antrenează modelul, la final, având o acuratețe de 99,27% pe imaginile de test.





Rețeaua poate fi văzuta ca un graf orientat în care nodurile transmit informații nodurilor din stratul următor, fiecare drum având o valoare care determină importanța acestuia. De asemenea, fiecare nod aplică o funcție de activare informațiilor primite, iar rezultatul acesteia este transmis mai departe. Datorită funcției de activare de tip Softmax aplicată ultimului strat, valoarea prezisă este numărul neuronului cu valoarea cea mai apropiată de 0.

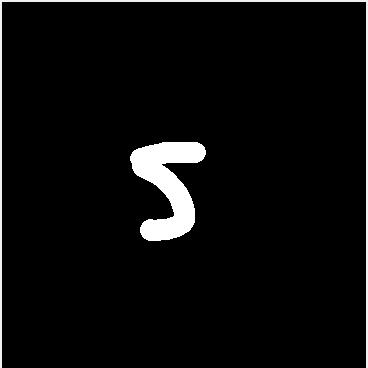
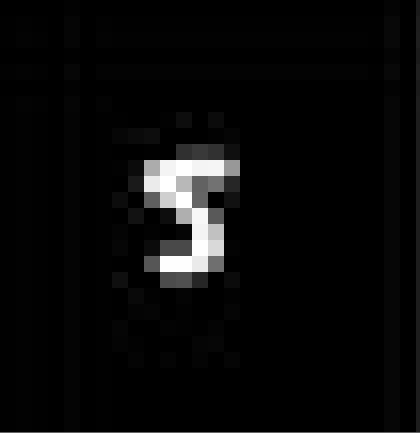
# **Probleme și rezolvări**

O problemă evidentă apărută imediat după partea de învățare este faptul că, deși acuratețea pe imaginile de test (aprox. 2.000) este de peste 99%, algoritmul încă greșește destul de mult când încearcă să clasifice imagini noi. Aceasta înseamnă ca modelul folosit cunoaște foarte bine cele 10.000 de imagini din baza de date MNIST, dar acuratețea generală, pe imagini nemaivăzute, este mult mai mică (s-a produs fenomenul de *OVERFITTING*).



Pentru a reduce acest fenomen, aplicația *HelpApp* salvează noi imagini și creează o bază de date pentru a îmbunătăți rezultatele modelului. Cu încă 2.000 de imagini noi, algoritmul CNN\_mnist\_best\_improve reantrenează rețeaua neuronală, obținând o acuratețe de 100% pe imaginile de test. Evident, procentajul acesta în realitate este mai mic, dar se poate observa o îmbunătățire clară.

Totuși, modelul este atât de precis doar în cazul în care cifrele sunt destul de mari în raport cu dimensiunea ferestrei (364 x 364 pixeli) și, pe cât posibil, centrate. Motivul pentru care cifra trebuie sa fie suficient de mare este pentru că, în etapa de preprocesare, imaginile sunt redimensionate la rezoluția de   
28 x 28 de pixeli. O cifră prea mică este, de obicei, prea greu de recunoscut și probabilitatea unei erori crește semnificativ. Redimensionarea este necesară pentru că arhitectura modelului este construită în așa fel încât să primească imagini de 28 x 28 pixeli pentru a se potrivi cu baza de date inițială.



364 x 364 28 x 28

# **Utilizare**

Aplicația este una de tip GUI (*Graphical User Interface*) construită cu ajutorul bibliotecii Tkinter și este formată dintr-un obiect te tip Canvas, doua butoane și un Label care conține valoarea prezisă. Modelul este încărcat în memorie dintr-un fișier (*best\_models/single\_models/Best.hdf5*), fără a mai fi nevoie de alocarea timpului pentru antrenament. Butonul Predict încarcă imaginea în memorie, o redimensionează și o transformă în date de intrare pentru rețeaua neuronală. Rezultatul este afișat apoi lângă textul “Prediction:”. La apăsarea butonului Reset, desenul se va șterge și o altă cifră poate fi introdusă.

Pornind de la arhitectura modelului se poate construi unul nou, mult mai complex care să poată recunoaște atât cifre, cât și litere. Un astfel de algoritm ar putea folosi o imagine scanată a unui text scris și l-ar putea transpune eficient în format digital. Cu o bază de date de test suficient de mare și suficient timp de antrenament, acuratețea generală ar putea ajunge aproape 100%. Scopul final este de a facilita experiența utilizatorilor, fapt care ar eficientiza lucrul pe dispozitivele electronice.

# **Bibliografie**

* https://www.google.ro
* https://keras.io
* <https://www.udemy.com/deeplearning/learn/lecture/6744030>
* https://ml-cheatsheet.readthedocs.io/en/latest/linear\_regression.html
* “Deep Learning With Python” - Jason Brownlee