RAPORT ACTIVITATE IUNIE - SEPTEMBRIE 2017

- ***Laurentiu-Gheorghe PICIU*** *(4E Software & Facultatea de Automatică și Calculatoare - UPB)*
- ***Andrei SIMION-CONSTANTINESCU*** *(4E Software & Facultatea de Automatică și Calculatoare - UPB)*

Prezentare generală

În domeniul Învățării Automate (Machine Learning), orice problemă are ca punct de plecare existența unor date. O pereche de forma $(x^{(i)}, y^{(i)})$ constituie un exemplu din setul de date, unde $x^{(i)}$ reprezintă variabilele de intrare (predictori), iar $y^{(i)}$ reprezintă variabila target. Astfel, dându-se un set de date, scopul este să se găsească o funcție $h : X \to Y$, cu proprietatea că $h(x)$ este un predictor bun pentru valoarea corespunzătoare - y. În momentul în care varibila target este continuă, atunci interacționăm cu o problemă de **regresie**. Altfel, dacă y aparține unei mulțimi cu un număr mic de valori discrete, atunci problema este una de **clasificare**.

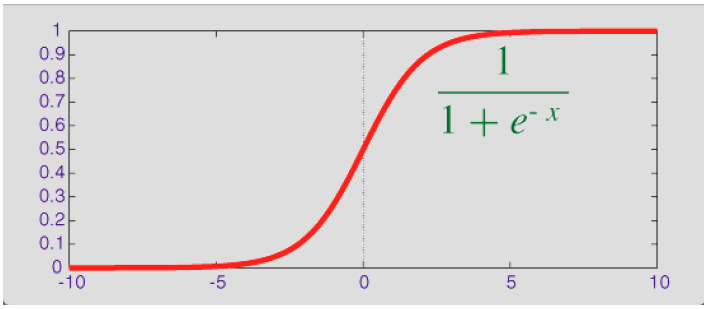
Atât în cazul regresiei liniare, cât și în cazul regresiei logistice (clasificare), se dorește găsirea unui **hiperplan** care să aproximeze cât mai bine punctele din setul de date. Ecuația hiperplanului este data de înmulțirea dintre parametrii θ (pe care dorim să îi îmbunătățim prin antrenament) și variabilele predictor:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \ldots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ .. \\ x_n \end{bmatrix} = \theta^T x, \text{ unde } n = \text{numărul}$$

predictorilor și $x_0 = 1$.

Problema clasificării

Pentru ca funcția ipoteză să întoarcă valori în intervalul $[0, 1]$ s-a introdus funcția **sigmoid**
$g(z) = \frac{1}{1+e^{-z}} \implies h_\theta(x) = g(\theta^T x)$.



Sigmoid-ul este de ajutor pentru clasificare binară, dar și pentru clasificare multinomială **One-vs-All**. Pentru generalizarea clasificării multinomiale, se folosește funcția **softmax**, care este o generalizare a sigmoid-ului. Pentru a putea fi folosit softmax-ul, trebuie ca variabila target sa fie **one-hot**, astfel încât funcția softmax să ofere K probabilități (K = numărul de clase):

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1 \mid x^{(i)}; \theta) \\ p(y^{(i)} = 2 \mid x^{(i)}; \theta) \\ .. \\ p(y^{(i)} = k \mid x^{(i)}; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ .. \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

Funcția de cost

Evaluarea convergenței unui model se face cu ajutorul **funcției de cost**. Pentru clasificare se folosește funcția de cost numită **cross-entropy**.

$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$, unde m =numărul de observații.

$Cost(h_\theta(x), y) = -log(h_\theta(x))$, dacă $y = 1$
$Cost(h_\theta(x), y) = -log(1 - h_\theta(x))$, dacă $y = 0$

$$\implies J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

Tunarea parametrilor

Pentru îmbunătățirea parametrilor, se va ține cont de derivatele parțiale ale funcției de cost în funcție de fiecare parametru θ_j, aplicându-se metoda **gradientului descendent**, care modifică parametrii astfel încât funcția de cost să descrească.

$Repeta$ {
$\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
}
$Repeta$ {
$\quad \theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
}

Varianta vectorizată: $\theta := \theta - \frac{\alpha}{m} X^T \cdot residual$

În această perioadă, activitatea noastră a avut ca scop înțelegerea conceptelor care stau în spatele modelelor de regresie/clasificare, precum și construirea unor astfel de modele. În principal, tot ceea ce s-a construit, a avut ca punct de plecare setul de date **MNIST**, care conține 70,000 de imagini de dimensiune 28x28, reprezentând cifrele de la 0 la 9 scrise de mână. Pe baza acestor imagini, s-au antrenat modele de clasificare din ce în ce mai complexe, pornind de la **regresie logistică**, **KNN** și **SVM**, avansând la **rețele neuronale complet conectate (fully-connected)** și terminând cu **rețele neuronale de convoluție (CNN)**. Modelele au fost antrenate/testate folosind limbajul de programare **Python**, fiind scrise inițial folosind numai operații vectorizate **NumPy**, urmând ca în final să rescriem totul cu ajutorul bibliotecii **TensorFlow**. Pentru fiecare model, setul de date a fost împărțit în date de antrenare (70%), date de validare în timpul antrenamentului (15%), precum și date de testare la finalul antrenamentului (15%). În plus, variabilele predictor (cei 784 de pixeli) au fost scalate între 0 și 1, folosind **scalarea (normalizarea) Min-Max**, ce face ca ele să "cântărească" la fel de mult și, în plus, crește abilitatea modelului de a învăța. Formula folosită pentru normalizare este: $X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$.

Modelele au fost antrenate **stochastic**, datele fiind procesate în mini-batch-uri și, în plus, s-au folosit metode care vizează:

- înlăturarea **overfitting-ului** ("învățare pe de rost"): regularizare, dropout;
- optimizarea procesului de găsire a minimului funcției de cost: momentum, descreștere a coeficientului de învățare (learning rate decay).

Conținut

Regresie logistică

Fiind vorba de 784 variabile predictor (784 pixeli) și de 10 categorii, s-au folosit 10x785 de parametri (câte un regresor pentru fiecare categorie). Aceștia au fost inițializați cu valoarea 0. Hiperparametrii folosiți pentru antrenament sunt urmatorii: **număr de epoci**: 25; **coeficient de învățare**: 0.001; **dimensiunea unui mini-batch**: 10; **coeficient regularizare**: 0.0001; **momentum**: 0.9; **factor decay**: 0.65.

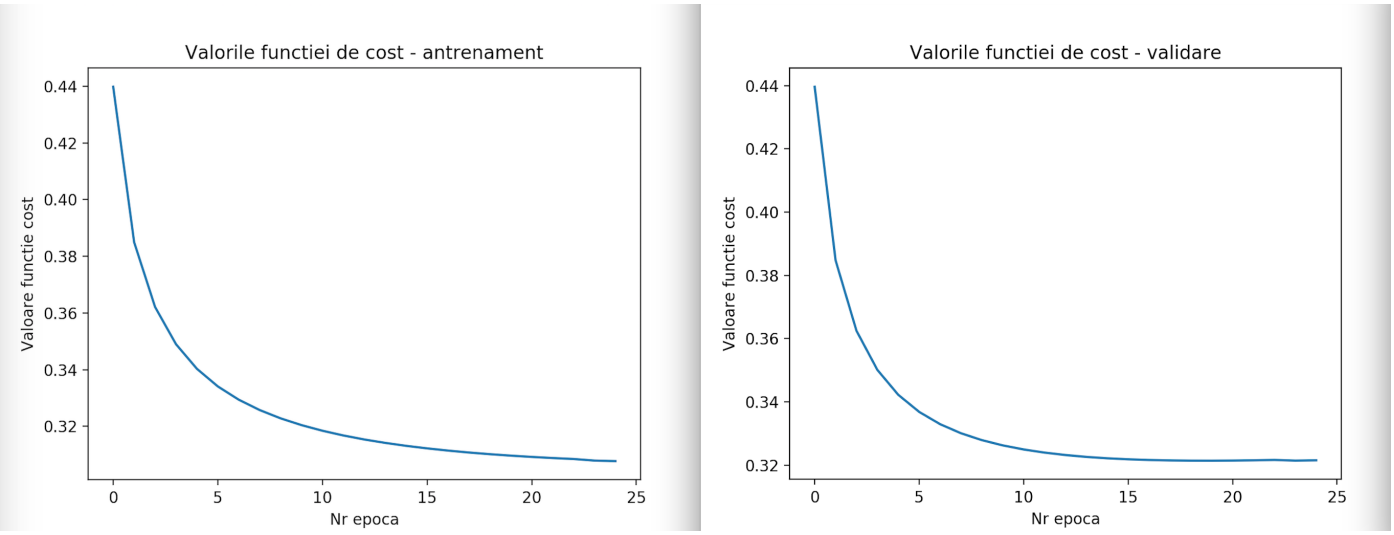
În timpul antrenamentului au fost înregistrate la fiecare epocă loguri relevante pentru a observa modul în care converge modelul (costul per mini-batch, target real vs target prezis, cost + acuratețe pe seturile de date de antrenament și validare, timpul scurs pe ecopa respectivă):

```
[LOGREG][2017-09-21 16:43:12] Training logreg model (initialized with 0)... epochs=25, alpha=0.001,
                              batch_sz=10, beta=0.0001, momentum=0.9, decay=0.65
[LOGREG][2017-09-21 16:43:12] Epoch 1/25
[LOGREG][2017-09-21 16:43:12]     [TRAIN Minibatch: 0] loss: 2.30
[LOGREG][2017-09-21 16:43:12]          yTrue:[8 3 1 4 3 0 7 2 0 8]
[LOGREG][2017-09-21 16:43:12]          yPred:[0 0 0 0 0 0 0 0 0 0]
[LOGREG][2017-09-21 16:43:12]     [TRAIN Minibatch: 1000] loss: 0.45
[LOGREG][2017-09-21 16:43:12]          yTrue:[7 8 9 7 3 1 0 7 8 8]
[LOGREG][2017-09-21 16:43:12]          yPred:[7 8 9 7 3 1 0 7 8 8]
[LOGREG][2017-09-21 16:43:13]     [TRAIN Minibatch: 2000] loss: 1.10
[LOGREG][2017-09-21 16:43:13]          yTrue:[1 2 8 4 8 1 3 4 0 6]
[LOGREG][2017-09-21 16:43:13]          yPred:[1 7 8 4 8 1 9 4 0 6]
[LOGREG][2017-09-21 16:43:13]     [TRAIN Minibatch: 3000] loss: 0.45
[LOGREG][2017-09-21 16:43:13]          yTrue:[1 3 7 4 1 0 8 7 9 9]
[LOGREG][2017-09-21 16:43:13]          yPred:[1 3 7 4 1 6 8 7 7 9]
[LOGREG][2017-09-21 16:43:14]     [TRAIN Minibatch: 4000] loss: 0.13
[LOGREG][2017-09-21 16:43:14]          yTrue:[0 8 1 1 4 4 2 6 5 3]
[LOGREG][2017-09-21 16:43:14]          yPred:[0 8 1 1 4 4 2 6 5 3]
[LOGREG][2017-09-21 16:43:15] 2.44s - loss: 0.44 - acc: 88.60% - val_loss: 0.44 - val_acc: 88.81%

[LOGREG][2017-09-21 16:43:15] Epoch 2/25
[LOGREG][2017-09-21 16:43:15]     [TRAIN Minibatch: 0] loss: 0.98
[LOGREG][2017-09-21 16:43:15]          yTrue:[8 3 1 4 3 0 7 2 0 8]
[LOGREG][2017-09-21 16:43:15]          yPred:[8 3 1 9 5 0 9 1 0 8]
[LOGREG][2017-09-21 16:43:15]     [TRAIN Minibatch: 1000] loss: 0.22
[LOGREG][2017-09-21 16:43:15]          yTrue:[7 8 9 7 3 1 0 7 8 8]
[LOGREG][2017-09-21 16:43:15]          yPred:[7 8 9 7 3 1 0 7 8 8]
[LOGREG][2017-09-21 16:43:16]     [TRAIN Minibatch: 2000] loss: 0.97
[LOGREG][2017-09-21 16:43:16]          yTrue:[1 2 8 4 8 1 3 4 0 6]
[LOGREG][2017-09-21 16:43:16]          yPred:[1 7 8 4 8 1 9 4 0 6]
[LOGREG][2017-09-21 16:43:16]     [TRAIN Minibatch: 3000] loss: 0.35
[LOGREG][2017-09-21 16:43:16]          yTrue:[1 3 7 4 1 0 8 7 9 9]
[LOGREG][2017-09-21 16:43:16]          yPred:[1 3 7 4 1 6 8 7 7 9]
[LOGREG][2017-09-21 16:43:17]     [TRAIN Minibatch: 4000] loss: 0.08
[LOGREG][2017-09-21 16:43:17]          yTrue:[0 8 1 1 4 4 2 6 5 3]
[LOGREG][2017-09-21 16:43:17]          yPred:[0 8 1 1 4 4 2 6 5 3]
[LOGREG][2017-09-21 16:43:17] 2.35s - loss: 0.39 - acc: 89.80% - val_loss: 0.38 - val_acc: 89.91%
```

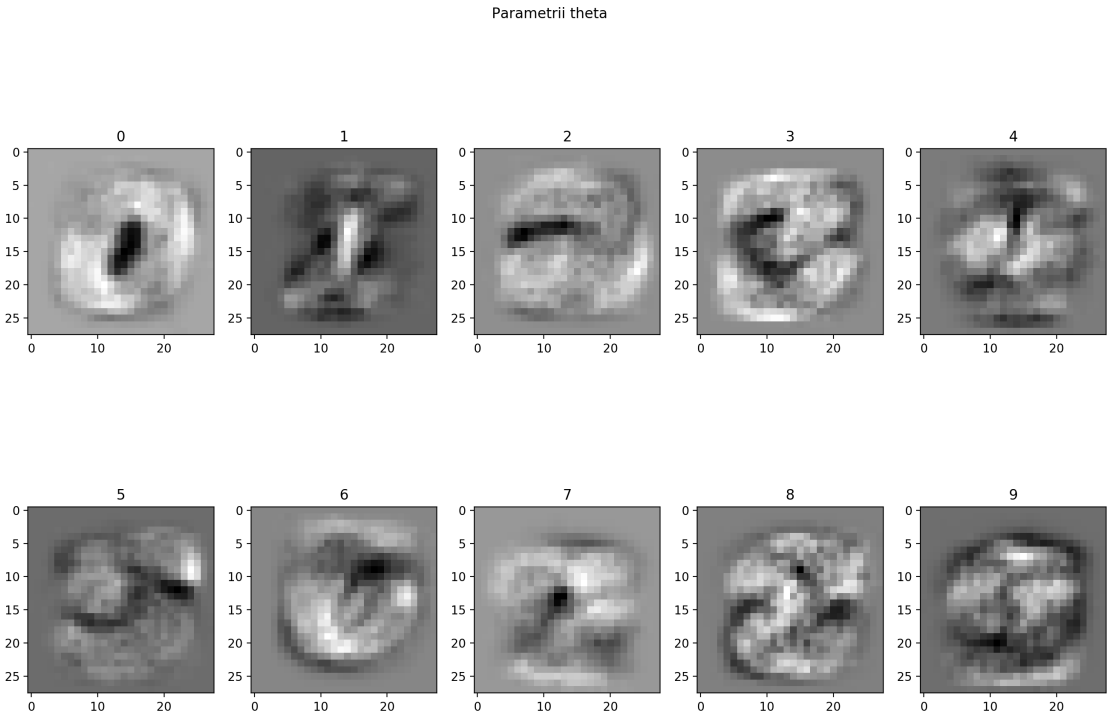
În urma antrenamentului modelului (care a durat aproximativ 60 de secunde), s-au obținut următoarele valori pentru acuratețe:

- pe setul de date folosit pentru antrenament: 92.45%
- pe setul de date de validare: 91.94%
- pe setul de date de test: 92.27%

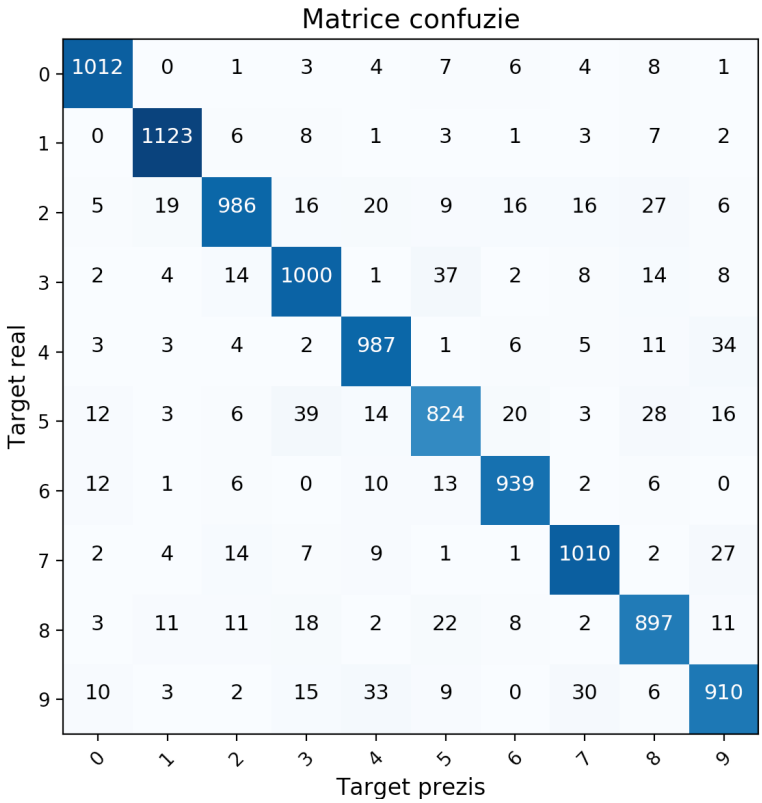
Convergența modelului se poate observa urmărind cele 2 grafice de mai jos:



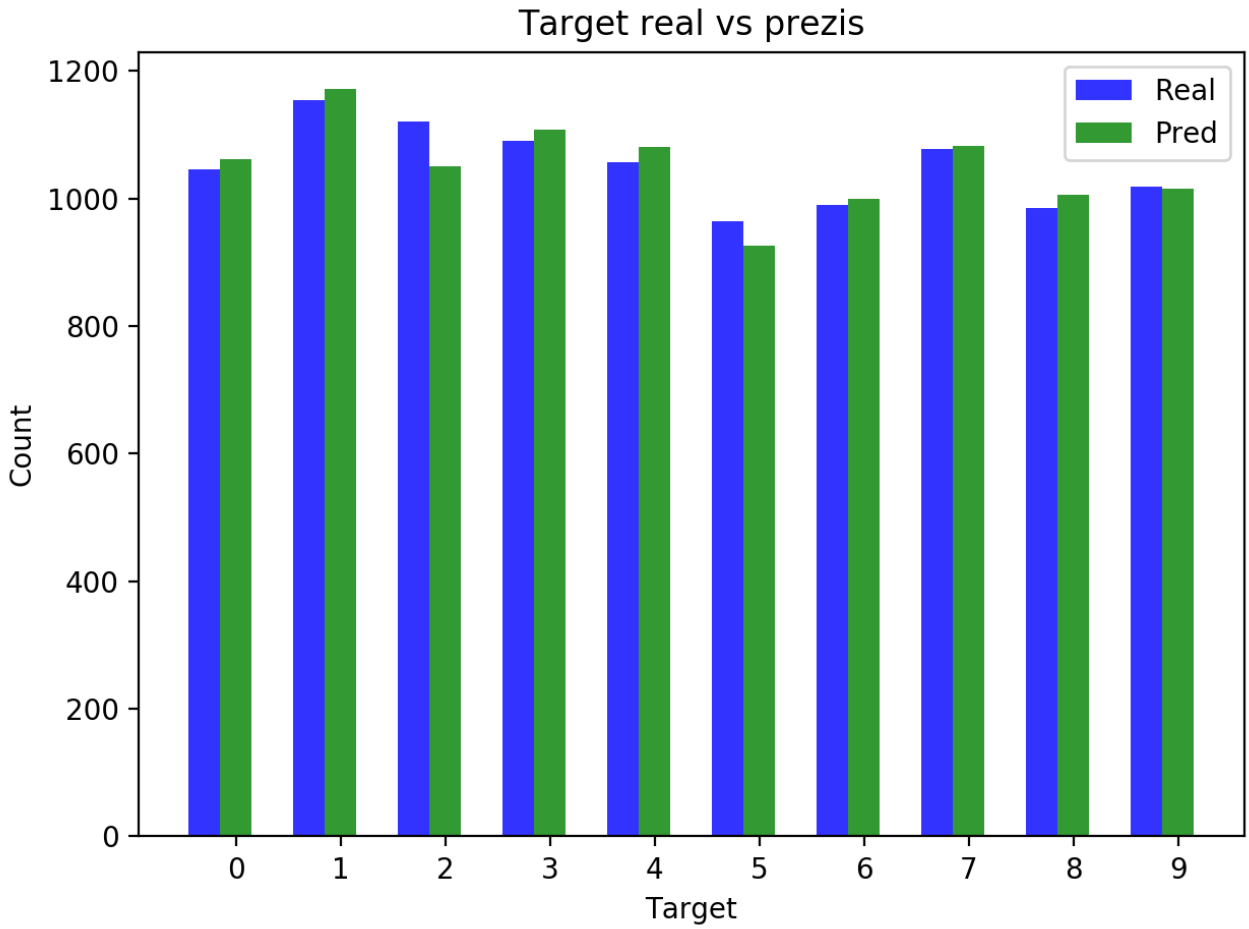
S-au folosit tehnici de afișare a celor 10 regresoare și s-a observat faptul că ele iau forma cifrei pe care doresc să o clasifice (adică parametrii vor avea valori foarte mari pozitive acolo unde pixelii sunt activați și valori foarte mari negative acolo unde pixelii sunt neactivați). Motivul apariției acestui fenomen este faptul că regresorul care încearcă să clasifice cifra $i, i = 0...9$ trebuie să întoarcă o valoare cât mai mare pentru o imagine care are target $= i$ și o valoare cât mai mică pentru o imagine cu target $! = i$, astfel încât aplicarea funcției softmax pentru predicție să scaleze aceste valori corespunzător între 0 și 1 pentru oferirea unei probabilități. Mai mult, odată cu afișarea celor 10 vectori θ ajungem la concluzia că **regresia logistică este o metodă de clasificare pixel-wise (nu învață trasături/caracteristici)**.



Pentru a observa în ce măsură s-a făcut predicția pe setul de date de test, s-au folosit procedee din domeniul statisticii precum: matricea de confuzie, precizie (cât din ce a fost prezis este și corect), recall (cât din ce este corect a fost prezis), scor F (media armonică a preciziei și a recall-ului):



	Precizie	Recall	F	N Obs
0	0.95	0.97	0.96	1046
1	0.96	0.97	0.97	1154
2	0.94	0.88	0.91	1120
3	0.90	0.92	0.91	1090
4	0.91	0.93	0.92	1056
5	0.89	0.85	0.87	965
6	0.94	0.95	0.94	989
7	0.93	0.94	0.94	1077
8	0.89	0.91	0.90	985
9	0.90	0.89	0.90	1018
avg	0.92	0.92	0.92	10500



KNN

KNN (K nearest neighbors) reprezintă unul din cei mai simpli algoritmi folosiți pentru clasificare/regresie, asta deoarece este *non-parametric*, astfel că datele de antrenare nu sunt folosite pentru a face o *generalizare*, ci pentru a face direct partea de predicție.

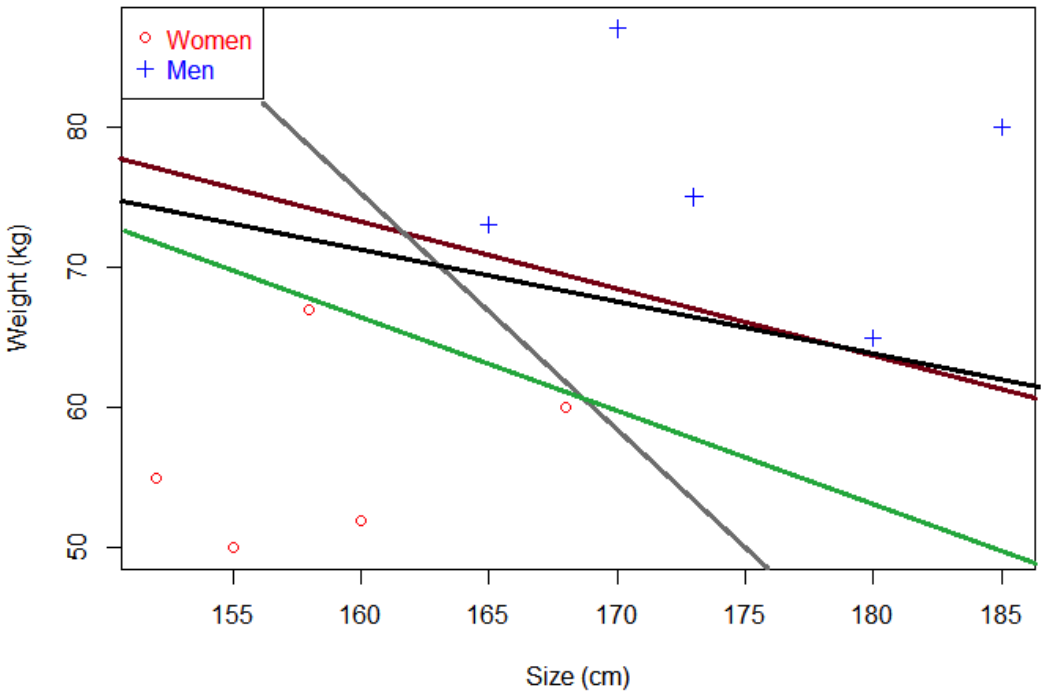
Ce înseamnă acest lucru? Algoritmul se uită la o observație din setul de date de test și calculează distanța dintre aceasta și toate observațiile din setul de date de antrenament. La final se aleg cele mai mici k sume și, în cazul clasficării, se alege categoria ca fiind cea care are majoritate între cele k descoperite cu ajutorul algoritmului.

Distanța dintre două imagini poate fi calculată în moduri diferite (dist. Euclidiană, Manhattan etc.), iar noi am folosit-o pe cea Manhattan, deoarece se comportă mai bine pentru date de dimensiuni mari (în cazul nostru dimensiunea este 784).

Deși **KNN** se descurcă foarte bine la clasificarea imaginilor din setul de date MNIST, **acuratețea fiind de 96.44%**, costul computațional este unul foarte mare. Rularea algoritmului pe MNIST (80% train, 20% test) a durat aproximativ 1 oră si 30 de minute (pentru o abordare iterativă - fiecare intrare de test era tratată separat) și în jur de 11 minute (folosind puterea de calcul a bibliotecii **sklearn**: pairwise distances (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise_distances.html) - ce calculează toate distanțele vectorizat).

SVM

Am discutat mai devreme despre faptul că regresiile găsesc un **hiperplan** care aproximează punctele pe care se face antrenamentul unui model. Cu toate acestea, găsirea lui nu înseamnă și faptul că el este cel mai bun. De exemplu, în figura de mai jos sunt reprezentate mai multe hiperplane, fiecare dintre ele făcând corect clasificarea sexelor:

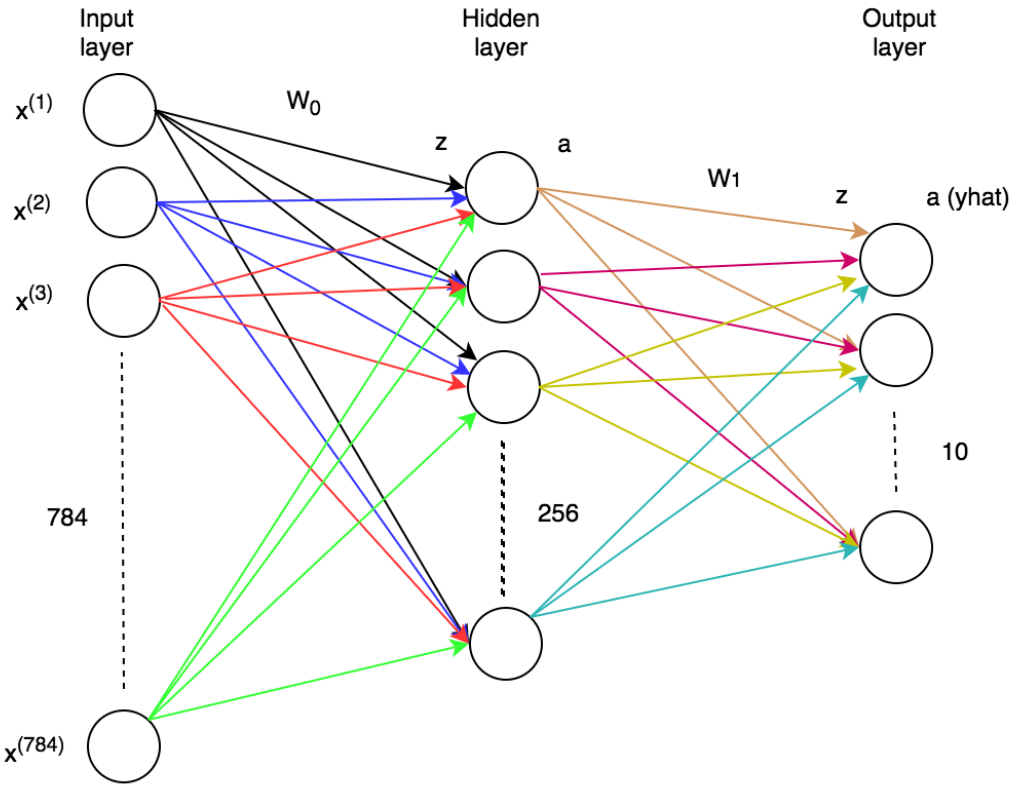


Algoritmul **SVM** (Support Vector Machine) caută să găsească hiperplanul care aproximează cel mai bine datele, prin introducerea noțiunii de **margine**: $2 * d$, unde d este distanța de la hipreplan la cel mai apropiat punct. Astfel, SVM consideră separația optimă ca fiind cea care maximizează marginea.

Pentru observarea modului în care se comportă acest algoritm la clasificarea cifrelor din MNIST, am folosit un clasificator din biblioteca **sklearn**: SVC (http://scikit-learn.org/stable/modules/svm.html#classification), a cărei acuratețe a fost 96.38% pentru datele de antrenament și 94.27% pentru datele de test.

Rețele neuronale complet conectate

Problema regresiei constă în faptul că nu surprinde deloc non-liniaritățile. Tot ce reușește să facă este să găsească o linie (hiperplan pentru $n > 2$) care să aproximeze cât mai bine punctele (observațiile). Pentru a scăpa de acest inconvenient, am creat o rețea neuronală complet conectată formată dintr-un nivel de intrare (cu 784 neuroni), un nivel ascuns (cu 256 neuroni) și un nivel de ieșire (cu 10 neuroni), care folosește o funcție de activare non-liniară (**ReLU**: $relu(z) = max(0, z)$, **sigmoid**) pentru a produce output-ul fiecărui nivel ascuns și, evident, **softmax** la nivelul de ieșire pentru ca rețeaua să returneze un vector de probabilități cu un număr de componente egal cu numărul de clase. Fiecare componentă a vectorului reprezintă probabilitatea ca imaginea de la intrare să se încadreze în clasa corespunzătoare.



1. Forward propagation

Pentru fiecare nivel, matricea de parametri care trebuie sa fie tunați va fi de dimensiune $(nrUnitsPrev + 1) \times nrUnitsCurrent$ (+1 pentru **bias**). Astfel, în cazul arhitecturii 784-256-10, au fost folosiți 203,530 parametri, ocupând 0.78MB din memoria totală a calculatorului.

Intrarea pentru primul nivel este chiar setul de date. Acesta reprezintă totodată și ieșirea nivelului. Întotdeauna ieșirea nivelului $i-1$ devine intrare pentru nivelul $i, i = 1..n-1$. Astfel, pentru nivelul ascuns se calculează produsul dintre intrare și matricea de parametri - obținând matricea z. Intervine însă introducerea non-liniarității care va genera ieșirea (matricea a). Asemănător, se calculează matricea z și pentru nivelul de ieșire, după care se aplică **softmax** pentru a genera cele 10 probabilități.

$$z^{(1)} = XW_0$$
$$a^{(1)} = f(z^{(1)})$$
$$z^{(2)} = a^{(1)}W_1$$
$$a^{(2)} = f(z^{(2)}) = \hat{y}$$

2. Backward propagation

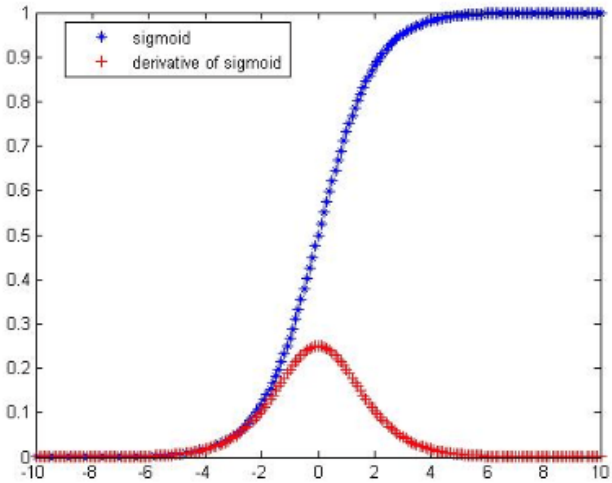
Tunarea parametrilor se face tot cu ajutorul metodei **gradientului descendent**.

$$\frac{\partial J}{\partial W_1} = \frac{1}{m} input^T \cdot \delta_2 = \frac{1}{m} a^{(1)T} \cdot (\hat{y} - y)$$
$$\frac{\partial J}{\partial W_0} = \frac{1}{m} input^T \cdot \delta_1 = \frac{1}{m} X^T \cdot [(\delta_2 \cdot W_1^T) * fAct']$$
$$W_1 := W_1 - \frac{\alpha}{m} \frac{\partial J}{\partial W_1}$$
$$W_0 := W_0 - \frac{\alpha}{m} \frac{\partial J}{\partial W_0}$$

Funcția de activare folosită este **ReLU**, a cărei derivată arată astfel:

$$f'(z) = \begin{cases} 1 & dacă\ z > 0 \\ 0 & altfel \end{cases}$$

În contrast, graficul derivatei funcției **sigmoid** arată faptul că pe masură ce valoarea sigmoidată este din ce în ce mai mare, derivata în acel punct devine din ce în ce mai mică, ceea ce înseamnă că gradientul va deveni nesemnificativ și astfel, rețeaua nu va fi capabilă să învețe foarte bine:



- ***Antrenarea modelului***

Parametrii modelului au fost inițializați folosind metoda **Xavier** (ei sunt aleși de pe o distribuție Gaussiană cu media = 0 și o variație = $\frac{1}{N}$, N = numărul neuronilor de pe nivelul anterior). Hiperparametrii folosiți pentru antrenament sunt urmatorii: **număr de epoci**: 10; **coeficient de învățare**: 0.001; **dimensiunea unui mini-batch**: 10; **coeficient dropout**: 0.7; **momentum**: 0.9.

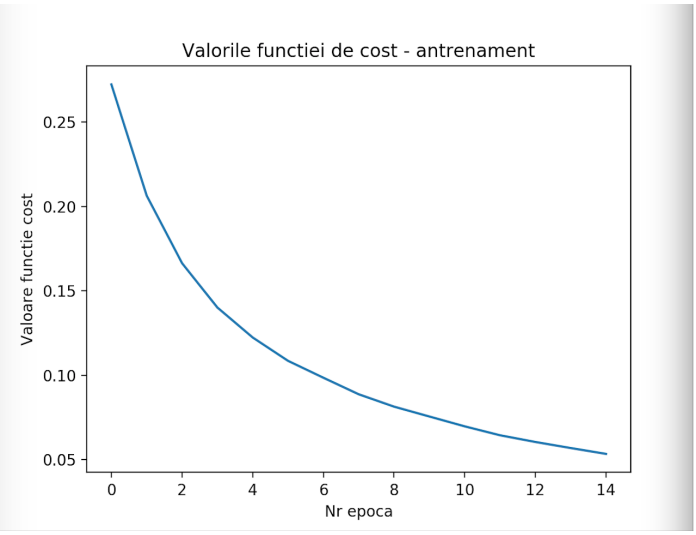
În timpul antrenamentului au fost înregistrate la fiecare epocă loguri relevante pentru a observa modul în care converge modelul (costul per mini-batch, target real vs target prezis, cost + acuratețe pe seturile de date de antrenament și validare, timpul scurs pe ecopa respectivă):

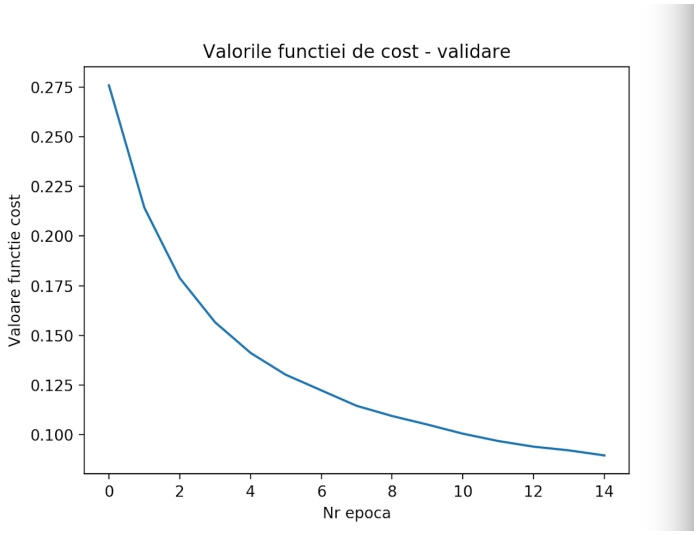
```
[DNN][2017-09-23 15:13:58] Layers:
   Layer:[0] Name:[inputlayer] Type:[input] Act:[] Units:[784]
   Layer:[1] Name:[hiddenlayer] Type:[hidden] Act:[relu] Units:[256]
   Layer:[2] Name:[outputlayer] Type:[output] Act:[softmax] Units:[10]
[DNN][2017-09-23 15:13:58] Training tf_dnn model (initialized using xavier method)... epochs=15, alpha=0.001,
                           batch_sz=10, beta=0, momentum=0.9, drop=0.7
[DNN][2017-09-23 15:14:18] Model capacity: 203,530 weights, 0.78MB
[DNN][2017-09-23 15:14:18] Epoch 1/15
[DNN][2017-09-23 15:14:18]    [TRAIN Minibatch: 0] loss: 2.41
[DNN][2017-09-23 15:14:18]         yTrue:[8 3 1 4 3 0 7 2 0 8]
[DNN][2017-09-23 15:14:18]         yPred:[5 5 7 5 3 5 3 5 3 5]
[DNN][2017-09-23 15:14:21]    [TRAIN Minibatch: 1000] loss: 0.33
[DNN][2017-09-23 15:14:21]         yTrue:[7 8 9 7 3 1 0 7 8 8]
[DNN][2017-09-23 15:14:21]         yPred:[7 8 9 7 3 1 0 7 8 8]
[DNN][2017-09-23 15:14:24]    [TRAIN Minibatch: 2000] loss: 1.12
[DNN][2017-09-23 15:14:24]         yTrue:[1 2 8 4 8 1 3 4 0 6]
[DNN][2017-09-23 15:14:24]         yPred:[1 7 8 4 8 1 9 4 0 6]
[DNN][2017-09-23 15:14:27]    [TRAIN Minibatch: 3000] loss: 0.41
[DNN][2017-09-23 15:14:27]         yTrue:[1 3 7 4 1 0 8 7 9 9]
[DNN][2017-09-23 15:14:27]         yPred:[1 3 7 4 1 6 8 7 7 9]
[DNN][2017-09-23 15:14:31]    [TRAIN Minibatch: 4000] loss: 0.06
[DNN][2017-09-23 15:14:31]         yTrue:[0 8 1 1 4 4 2 6 5 3]
[DNN][2017-09-23 15:14:31]         yPred:[0 8 1 1 4 4 2 6 5 3]
[DNN][2017-09-23 15:14:35] 15.91s - loss: 0.27 - acc: 92.47% - val_loss: 0.28 - val_acc: 92.27%

[DNN][2017-09-23 15:14:35] Epoch 2/15
[DNN][2017-09-23 15:14:35]    [TRAIN Minibatch: 0] loss: 0.63
[DNN][2017-09-23 15:14:35]         yTrue:[8 3 1 4 3 0 7 2 0 8]
[DNN][2017-09-23 15:14:35]         yPred:[8 3 1 9 5 0 7 1 0 8]
[DNN][2017-09-23 15:14:38]    [TRAIN Minibatch: 1000] loss: 0.08
[DNN][2017-09-23 15:14:38]         yTrue:[7 8 9 7 3 1 0 7 8 8]
[DNN][2017-09-23 15:14:38]         yPred:[7 8 9 7 3 1 0 7 8 8]
[DNN][2017-09-23 15:14:40]    [TRAIN Minibatch: 2000] loss: 0.68
[DNN][2017-09-23 15:14:40]         yTrue:[1 2 8 4 8 1 3 4 0 6]
[DNN][2017-09-23 15:14:40]         yPred:[1 7 8 4 8 1 9 4 0 6]
[DNN][2017-09-23 15:14:43]    [TRAIN Minibatch: 3000] loss: 0.21
[DNN][2017-09-23 15:14:43]         yTrue:[1 3 7 4 1 0 8 7 9 9]
[DNN][2017-09-23 15:14:43]         yPred:[1 3 7 4 1 6 8 7 9 9]
[DNN][2017-09-23 15:14:45]    [TRAIN Minibatch: 4000] loss: 0.03
[DNN][2017-09-23 15:14:45]         yTrue:[0 8 1 1 4 4 2 6 5 3]
[DNN][2017-09-23 15:14:45]         yPred:[0 8 1 1 4 4 2 6 5 3]
[DNN][2017-09-23 15:14:48] 11.15s - loss: 0.21 - acc: 94.22% - val_loss: 0.21 - val_acc: 94.05%
```

În urma antrenamentului modelului (care a durat aproximativ 185 de secunde), s-au obținut următoarele valori pentru acuratețe:

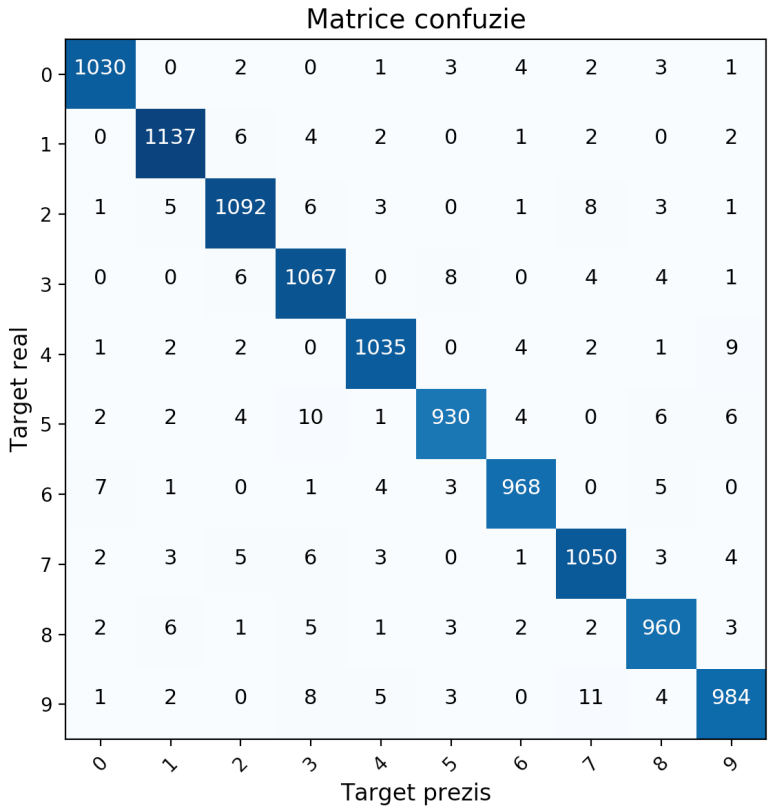
- pe setul de date folosit pentru antrenament: 98.32%
- pe setul de date de validare: 97.55%
- pe setul de date de test: 97.74%

Convergența modelului se poate observa urmărind cele 2 grafice de mai jos:





- ***Rezultatele predicției pe setul de date de test***



	Precizie	Recall	F	N Obs
0	0.98	0.98	0.98	1046
1	0.98	0.99	0.98	1154
2	0.98	0.97	0.98	1120
3	0.96	0.98	0.97	1090
4	0.98	0.98	0.98	1056
5	0.98	0.96	0.97	965
6	0.98	0.98	0.98	989
7	0.97	0.97	0.97	1077
8	0.97	0.97	0.97	985
9	0.97	0.97	0.97	1018
avg	0.98	0.98	0.98	10500

Rețele neuronale de convoluție (CNN)

Rețelele neuronale de convoluție (Convolutional Neural Networks) sunt clasificatori speciali concepuți pentru lucrul cu imagini.

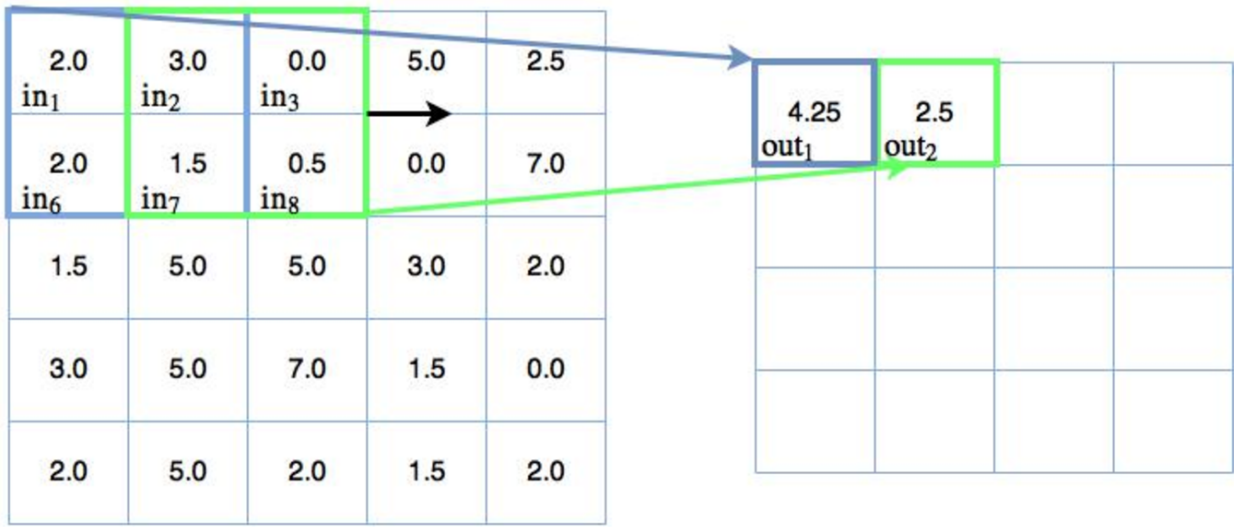
În general, CNN au două componente:

1. O componentă de extragere a trasăturilor din imagini. Aceasta se compune din mai multe nivele:

   - **Nivel de convoluție**

     În cadrul acestuia se realizează convoluția dintre imaginea de intrare și un filtru (matrice $n \times n, n \leqslant 5$). Spre deosebire de nivelele din rețelele neuronale clasice, în nivelele de convoluție numărul de conexiuni se reduce la dimensiunea filtrelor utilizate, și nu la cea a imaginii de la intrare. Nivelurile de convoluție produc una sau mai multe *feature maps* (în funcție de numărul de filtre utilizate), imagini care conțin anumite trăsături sau caracteristici ale imaginii de la intrare. Aceste hărți sunt obținute prin plimbarea fiecărui filtru asupra imaginii de intrare (mecanismul **sliding window**).

Plimbarea unui filtru de 2x2 (cu toate ponderile = 0.5) asupra unei imagini (stride = 1)



La antrenarea rețelei, valorile filtrelor se modifică după o logică similară cu cea a rețelelor clasice, ideea fiind de a minimiza eroarea dintre rezultatul de la ieșire dorit și cel obținut efectiv.

- **Nivel de activare**

  Hărțile obținute în urma convoluțiilor sunt supuse unei funcții de activare ce asigură comportamentul non-liniar al rețelei. Ca funcție de activare, de cele mai multe ori se folosește **ReLU**.

- **Nivel de agregare (pooling)**

  În cadrul acestui nivel se realizează o reducere a dimensiunilor hărților de activare. Astfel, operația de pooling 2x2 aplicată pe o hartă de dimensiune 28x28 generează o hartă de dimensiune 14x14. Reducerea dimensiunii pentru fiecare grup 2x2 de pixeli se poate realiza prin mai multe metode: de exemplu, se determină maximul dintre cei 2x2 pixeli (max pooling).

2. O componentă complet conectată (fully-connected), în cadrul căreia se realizează clasificarea propriu-zisă. Această componentă este o rețea neuronală clasică, la intrarea căreia se furnizează *feature map-urile* și la ieșirea căreia se aplică funcția de activare **softmax**, pentru a încadra imaginea în clasa corespunzătoare.

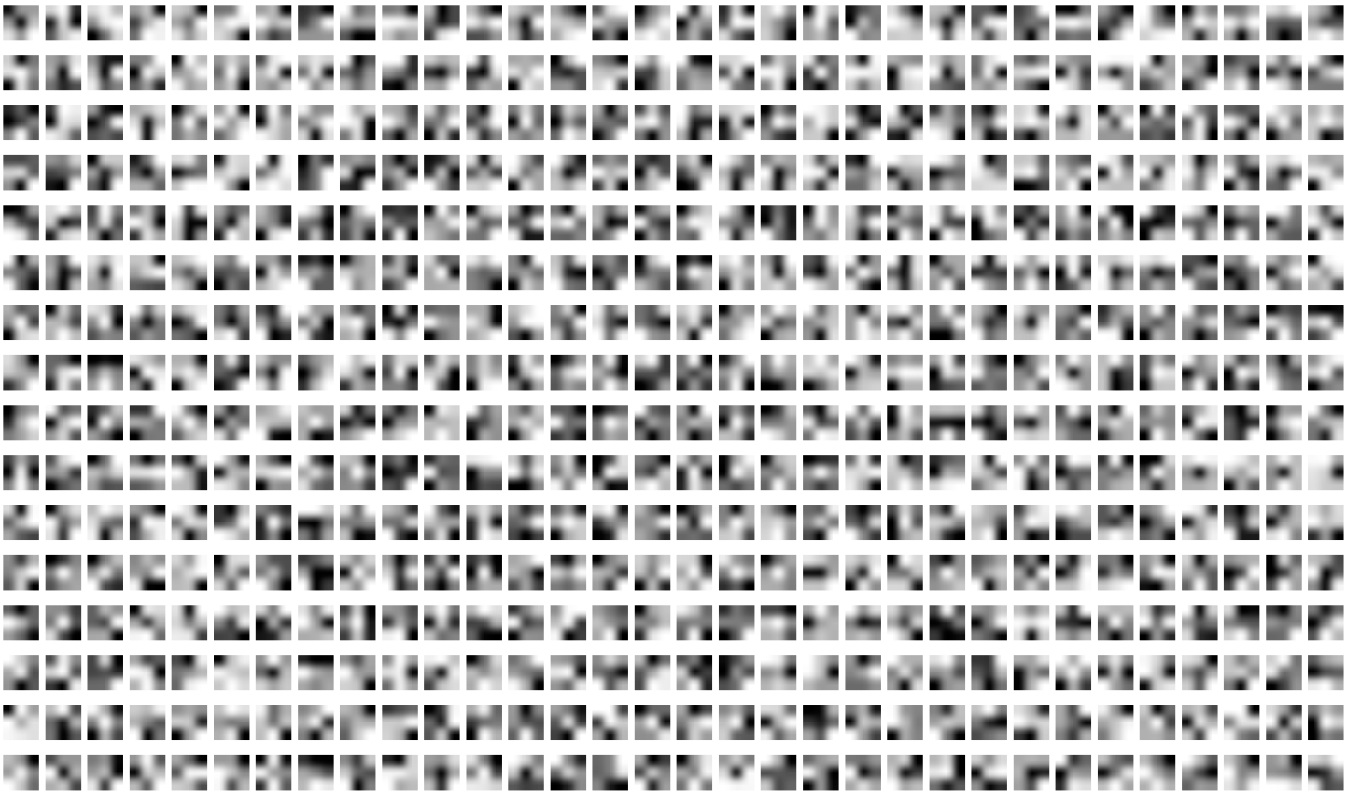
- **Antrenarea modelului**

  Arhitectura aleasă pentru rețeaua ce clasifică imaginile din MNIST este următoarea:

  - *INPUT* [28x28x1] - reprezintă pixelii imaginii: 28 pixeli înalțime, 28 pixeli lățime, un canal de culoare ("grayscale")
  - *CONV* - este responsabil cu crearea hărților despre care s-a discutat: am folosit 32 de filtre de 3x3 care se plimbă cu pas=1. Se folosește și o tehnică de bordare a imaginii cu 0-uri, astfel încât harta să aibă aceeași dimensiune. Volumul ce trebuie procesat este acum [28x28x32]
  - *RELU* - se aplică funcția de activare ReLU. Volumul rămâne același.
  - *MAXPOOL* - "downsampling"; fereastra de 2x2 se plimbă cu pas=2.
  - *CONV* - crearea unor hărți care să surprindă trăsături și mai abstracte. Am folosit 64 de filtre de 3x3 care se plimbă cu pas=1. Dimensiumea volumului devine [14x14x64]
  - *RELU* - se aplică funcția de activare ReLU.
  - *MAXPOOL* - din nou se face același "downsampling" pe noile hărți
  - *FC* - se creează o rețea neuronală cu 3 niveluri: [input, $7 * 7 * 64$ neuroni]; [hidden, 1000 neuroni, act=ReLU]; [output, 10 neuroni, act=softmax].

Modelul s-a descurcat semnificativ mai bine decât o rețea neuronală complet conectată, obținând următoarele rezultate dupa antrenarea și testarea modelului pe MNIST: - Acuratețe: train (99.49%), validare (98.87%), test (99.06%) - Timp total de rulare: 11 minute 08 secunde (pentru 3 epoci)

S-au folosit tehnici de afișare a câtorva filtre folosite pentru a extrage trăsăturilor imaginilor. La finalul antrenamentului, ele arătă în felul următor:



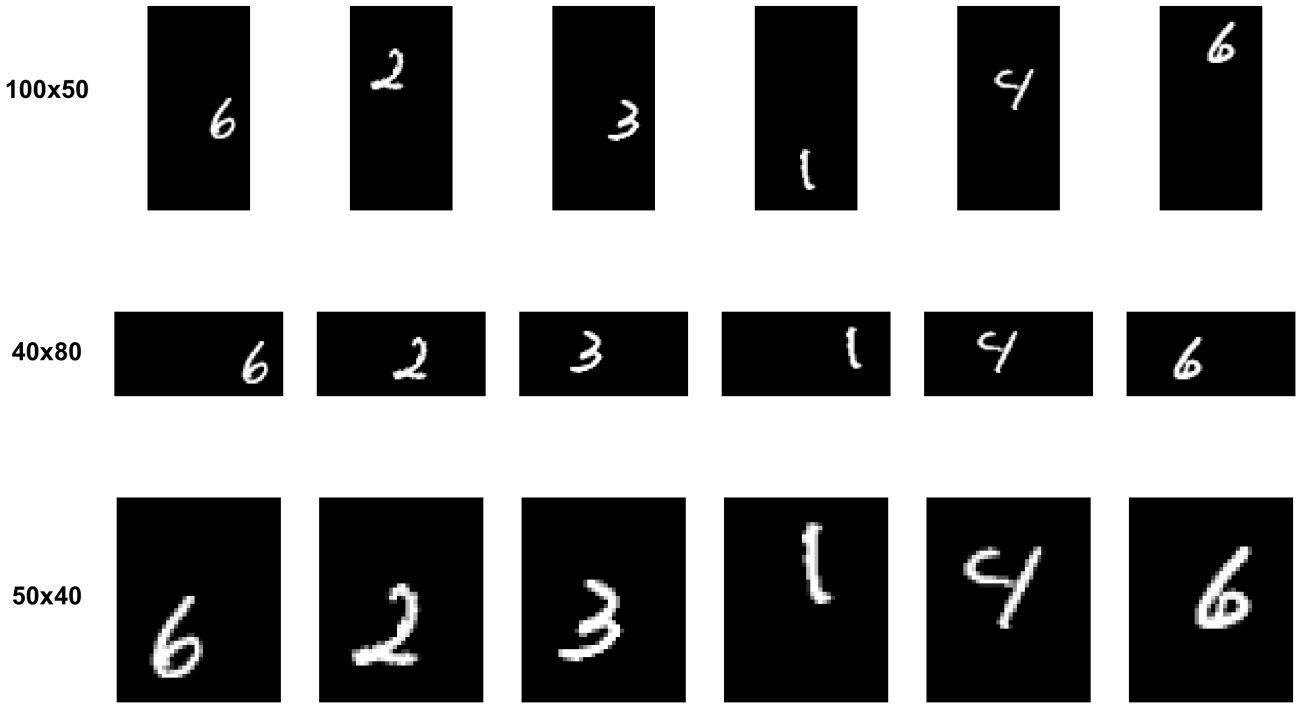
- **Inferența pe scene de dimensiuni variate**

  Deși am găsit o modalitate de recunoaștere a trăsăturilor din imagini, încă nu se poate face inferența pe scene cu dimensiuni diferite de 28x28, deoarece prin max-pooling se face doar un 'downsampling' al hărților, iar numărul de neuroni de pe layer-ul de intrare al componentei *fully-connected* va depinde de dimensiunea imaginilor pe care s-a făcut antrenamentul. Astfel, pentru ca rețeaua să poată face inferența pe orice scenă după ce a fost antrenată, am hotărât ca după ultimul nivel de convoluție să folosim **global max pooling** - alegem valoarea maximă din fiecare hartă. Este evident că vom avea un număr egal de hărți cu cel al filtrelor folosite pe ultimul nivel de convoluție și astfel numărul neuronilor de pe layer-ul de intrare al componentei FC va fi dependent doar de arhitectura aleasă.

  Arhitectura rețelei devine: *INPUT* [28x28x1] - *CONV* [32 filtre de dimensiune 3x3] - *ReLU* [asupra volumului de dimensiune 28x28x32] - *CONV* [256 filtre de dimensiune 3x3] - *ReLU* [asupra volumului de dimensiune 28x28x256] - *GMP* - *FC* [rețea neuronală cu 2 niveluri: (input, 256 neuroni); (output, 10 neuroni, act=softmax)]. Cu ajutorul acesteia, s-a obținut o acuratețe de peste 96% pe date de test de orice dimensiune (scene care arată precum cele prezentate în capitolul următor).

Fereastra glisantă

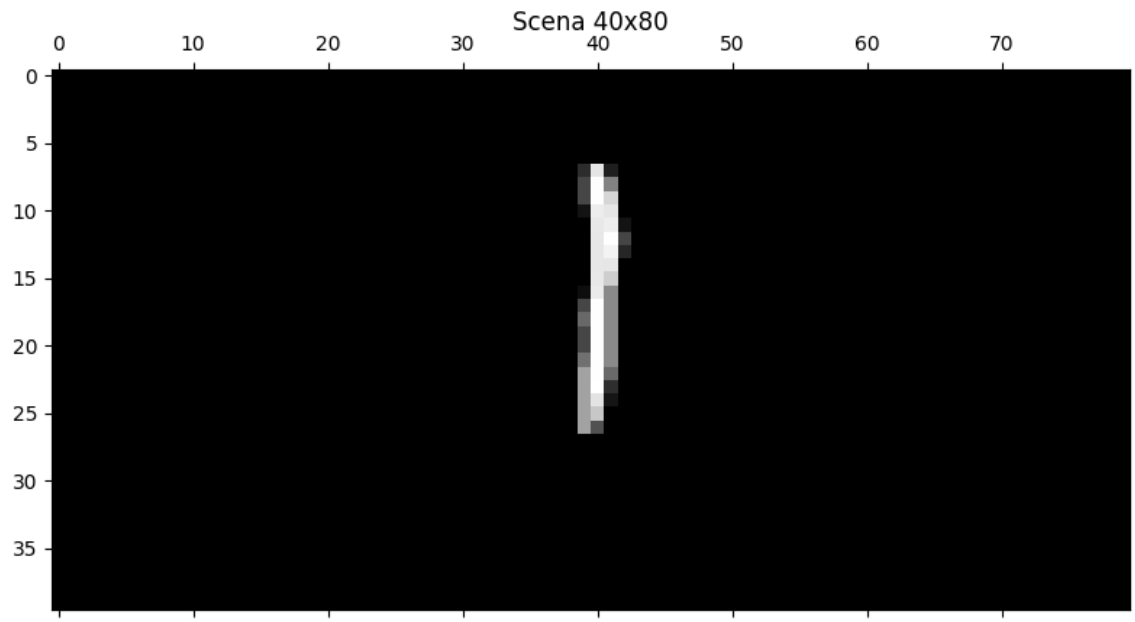
Pentru acest experiment, am înglobat toate cele 10500 de imagini din setul de date de test în scene de diverse dimensiuni (40x80, 50x40, 100x50), în poziții aleatoare. Colajul de mai jos conține 6 imagini din MNIST puse în cele 3 scene:



În continuare, am implementat mecanismul **sliding window**, plimbând o fereastră de 28x28 pe fiecare scenă. Scopul a fost găsirea target-ului, cât și poziția exactă a cifrei în scenă. Vom analiza rezultatele obținute prin utilizarea fiecărui model:

- **Regresie logistică**

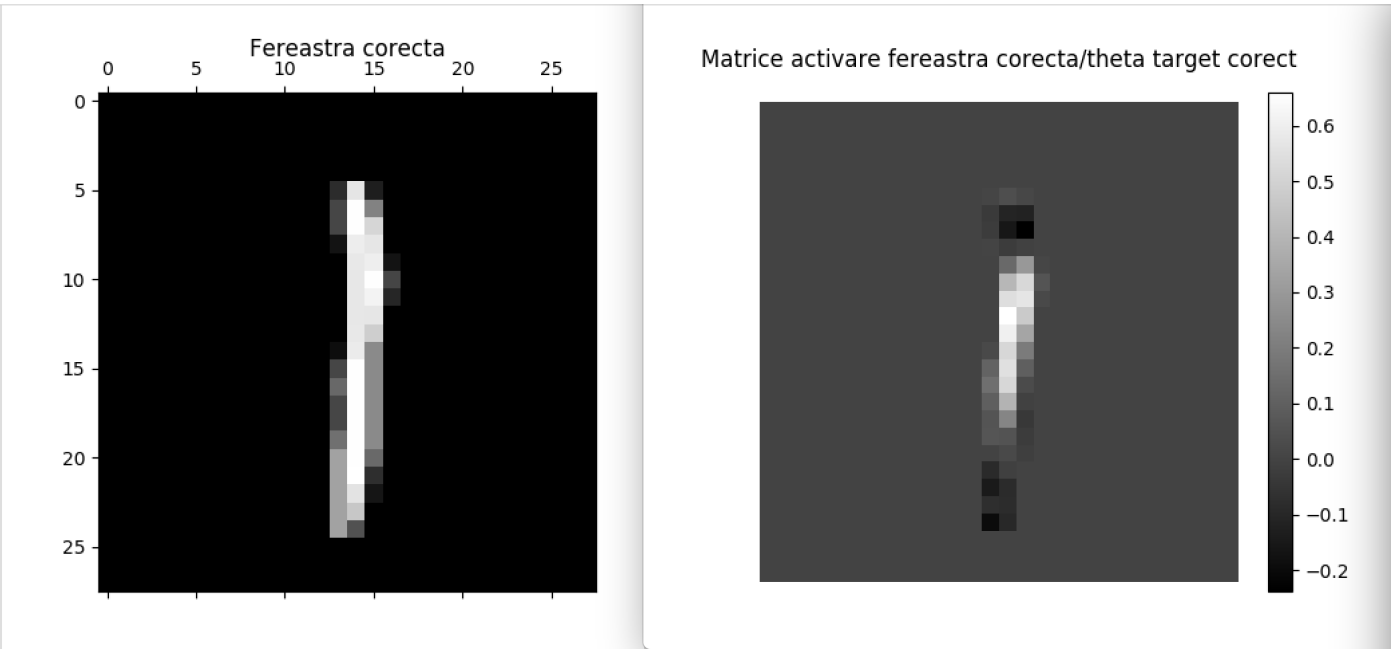
Vom analiza 2 scene de dimensiune 40x80, una în care modelul nostru a identificat corect cifra poziționată în scenă, iar alta în care acesta a generat o predicție falsă.



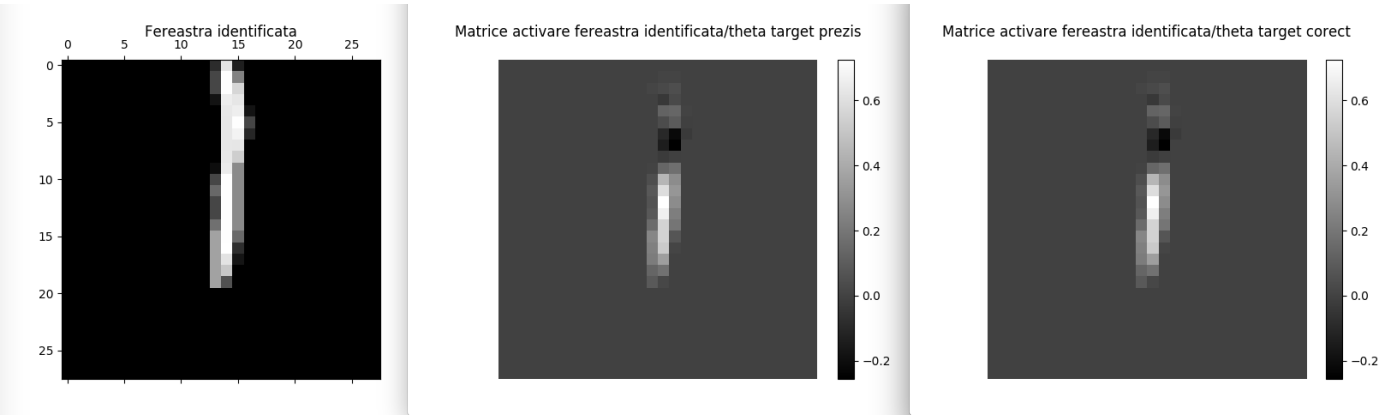
Scenă de 40x80 cu cifra 1 poziționată la (2, 26)

Modelul antrenat pe datele din MNIST a identificat corect cifra 1 din scena de mai sus, cu o siguranță de 98,97%. Fereastra de siguranță maximă s-a găsit la poziția (4, 26), aflându-se la o distanță rezonabilă de poziția corectă. Siguranța de la fereastra de pe poziția exactă a fost 97,48%, diferența de probabilitate între cele 2 poziții fiind de sub 1,5%.

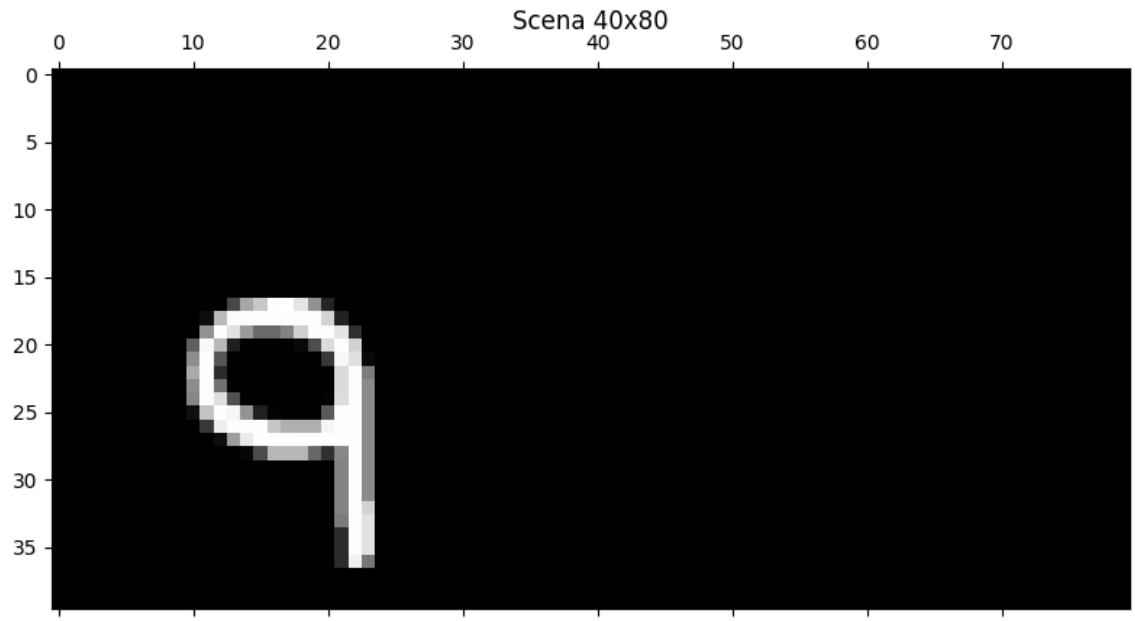
Acum ne vom orienta atenția către vizualizarea matricilor de activare, între ferestre (prezisă și corectă) și regresorii asociați cifrei prezise, respectiv cifrei corecte



Matrice de activare pentru fereastra de pe poziția corectă



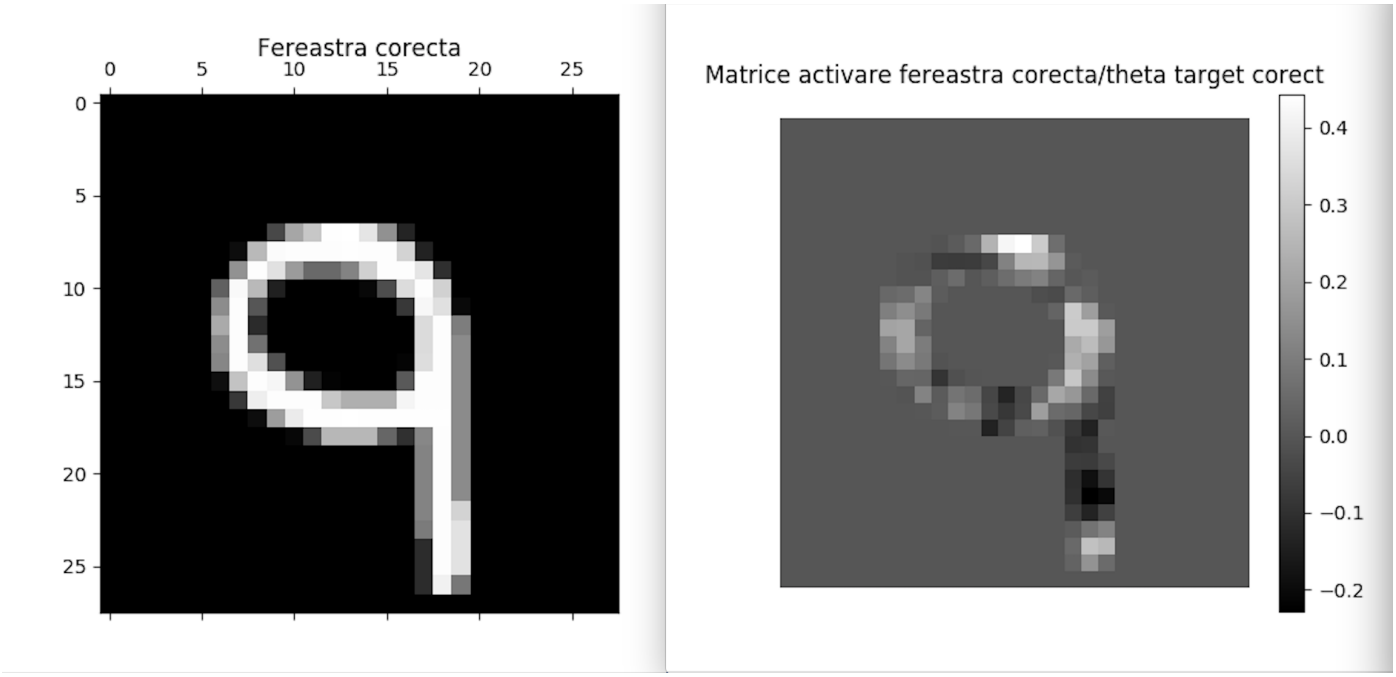
Matricile de activare pentru fereastra de pe poziția prezisă



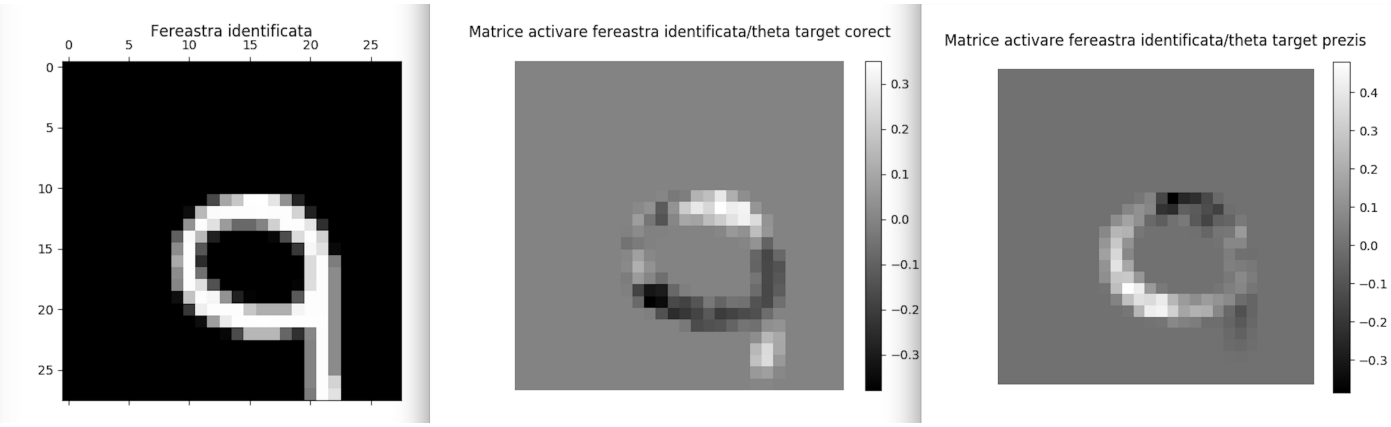
Scenă de 40x80 cu cifra 9 poziționată la (10, 4)

Modelul nostru a identificat în fereastra de la poziția (6, 1), cu siguranța maximă de 98,77%, cifra 6. Probabilitatea obținută pe fereastra de la poziția exactă a fost de 80,15%, prezicând corect cifra 9, dar cu o siguranță mult mai mică față de cifra 6 de la poziția (6,1).

Urmează să vizualizăm matricile de activare.



Matrice de activare pentru fereastra de pe poziția corectă



Matricile de activare pentru fereastra de pe poziția prezisă

Pentru a ne da seama de acuratețea modelului, am lăsat algoritmul să ruleze pe toate cele 3 seturi de date. Nu vom oferi o acuratețe clasică în procente, ci un număr de scene prezise corect sau greșit într-un anumit mod, pe fiecare set de scene în parte. Metrica folosită pentru fiecare set de scene este: în câte scene a identificat corect atât cifra, cât și poziția acesteia, în câte scene a identificat corect doar cifra și în câte a greșit. Trebuie menționat că în verificarea poziției s-a ținut cont de un epsilon de 2 pixeli, în toate direcțiile. \

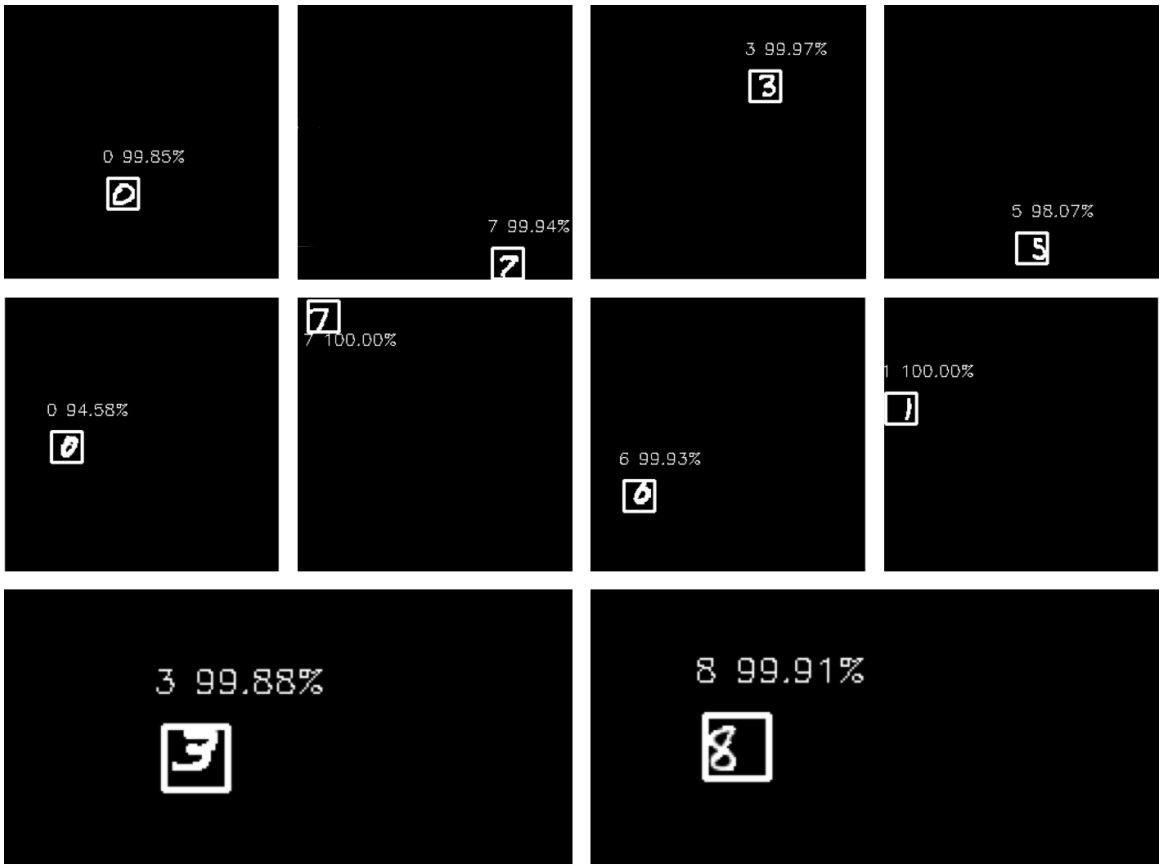
```
Rezultatele rulării regresiei logistice:
[2017.09.26-12:59:17] Scenes of size 40x80 - 427.57s -  corrects=1883,
partially_wrongs=2054, wrongs=6563
[2017.09.26-13:02:19] Scenes of size 50x40 - 182.16s - corrects=2346,
partially_wrongs=2166, wrongs=5988
[2017.09.26-13:19:30] Scenes of size 100x50 - 1031.35s - corrects=1524,
partially_wrongs=1852, wrongs=7124
```

- **Rețele neuronale FC**

```
Rezultatele rulării rețelei neuronale:
[2017.09.26-01:25:53] Scenes of sz 40x80 - 2653.00s - corrects=6241,
partially_wrongs=1684, wrongs=2575
[2017.09.26-01:45:04] Scenes of sz 50x40 - 1150.68s - corrects=6174,
partially_wrongs=1710, wrongs=2616
[2017.09.26-03:22:50] Scenes of sz 100x50 - 5865.57s - corrects=5359,
partially_wrongs=1665, wrongs=3476
```

- **CNN**

Pentru a observa mai bine cum se comportă modelul rezultat în urma antrenării folosind CNN, am folosit OpenCV pentru a vizualiza felul în care fereastra este baleată asupra scenei. Totodată, în momentul în care fereastra ajunge să fie deasupra unor pixeli activați, se face și inferența - prezicerea target-ului și probabilitatea cu care modelul prezice. Am ales aleator 10 scene din cele procesate pentru testare și pentru fiecare am desenat fereastra pentru care s-a obținut probabilitatea maximă.



Găsirea target-ului la într-o scenă

Se observă faptul că modelul nostru nu numai că își dă seama că într-o scenă este desenată o anumită cifră, ci detectează și locul unde ea se află. Vom vedea în ultima parte a raportului felul în care modelul se descurcă pe o scenă cu mai multe cifre desenate și tot acolo vom analiza și pozițiile la care sunt găsite.

Comportament CNN pe scene care înglobează mai mult de o imagine

Cifre de dimensiuni egale (28x28)

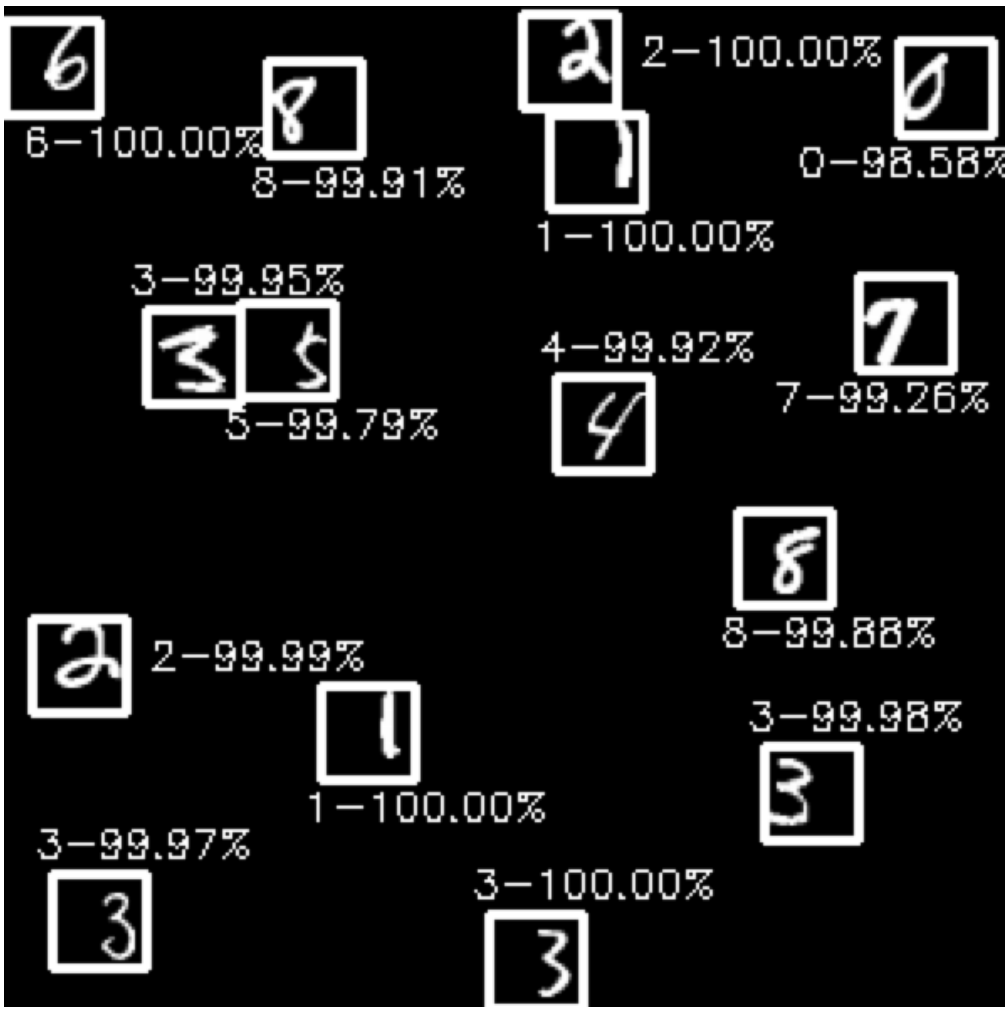
Am creat o scenă de dimensiune 300x300, pe care am desenat 15 cifre alese aleator din setul de date de test. Cifrele au fost dispuse astfel:

```
[{'object': 6, 'pos_left_up': (3, 4)},
 {'object': 8, 'pos_left_up': (15, 72)},
 {'object': 2, 'pos_left_up': (0, 158)},
 {'object': 7, 'pos_left_up': (80, 250)},
 {'object': 3, 'pos_left_up': (90, 43)},
 {'object': 2, 'pos_left_up': (180, 10)},
 {'object': 3, 'pos_left_up': (260, 20)},
 {'object': 1, 'pos_left_up': (200, 100)},
 {'object': 4, 'pos_left_up': (110, 165)},
 {'object': 3, 'pos_left_up': (220, 220)},
 {'object': 3, 'pos_left_up': (270, 150)},
 {'object': 5, 'pos_left_up': (90, 77)},
 {'object': 0, 'pos_left_up': (10, 260)},
 {'object': 8, 'pos_left_up': (150, 220)},
 {'object': 1, 'pos_left_up': (30, 170)}]
```

Am aplicat algoritmul care plimbă fereastra asupra întregii scene (din 2 în 2 pixeli), face inferența utilizând modelul antrenat cu CNN și am obținut următorul rezultat:

```
[{'object': 6, 'pos_left_up': (4, 0), 'probability': 100%},
 {'object': 8, 'pos_left_up': (16, 78), 'probability': 99.91%},
 {'object': 2, 'pos_left_up': (2, 154), 'probability': 100%},
 {'object': 7, 'pos_left_up': (80, 254), 'probability': 99.26%},
 {'object': 3, 'pos_left_up': (90, 42), 'probability': 99.95%},
 {'object': 2, 'pos_left_up': (182, 8), 'probability': 99.99%},
 {'object': 3, 'pos_left_up': (258, 14), 'probability': 99.97%},
 {'object': 1, 'pos_left_up': (202, 94), 'probability': 100%},
 {'object': 4, 'pos_left_up': (110, 164), 'probability': 99.92%},
 {'object': 3, 'pos_left_up': (220, 226), 'probability': 99.98%},
 {'object': 3, 'pos_left_up': (270, 144), 'probability': 100%},
 {'object': 5, 'pos_left_up': (88, 70), 'probability': 99.79%},
 {'object': 0, 'pos_left_up': (10, 266), 'probability': 98.58%},
 {'object': 8, 'pos_left_up': (150, 218), 'probability': 99.88%},
 {'object': 1, 'pos_left_up': (32, 162), 'probability': 100%}]
```

Pentru validarea rezultatului de mai sus, am folosit OpenCV, ce ne ajută să desenăm ferestrele exact la pozițiile menționate. Se observă clar că modelul găsește cifrele în vecinătatea pozițiilor care descriu imaginile din MNIST **centrate**, astfel încât putem trage concluzia că modelul are capacitatea de a distinge trăsături specifice unei imagini.



Inferența făcută pe o scenă cu mai multe obiecte (cifre)

Cifre de dimensiuni variate (scalate)

Până în acest moment nu am reușit să construim o arhitectură care să creeze un model ce poate face inferența la fel de bine precum o face pe cifre din MNIST care nu sunt scalate. Pentru a arăta progresul, am creat 4 noi seturi de date de test, dar de data aceasta cu imaginile scalate, având următoarele dimensiuni: 14x14, 28x28, 50x50, 84x84, 100x100. S-au antrenat 11 modele cu rețete ce au cu arhitecturile menționate în primul tabel. Am adăugat nivele noi de convoluție + nivele complet conectate, în speranța că vor fi depistate trăsături din ce în ce mai abstracte. În cel de-al doilea tabel pot fi observate acuratețile obținute în urma predicției făcute pe cele 4 seturi de date folosind cele 11 modele.

ID	Arhitectura
M1	INPUT - CNV 64 - CNV 256 - GMP - SM
M2	INPUT - CNV 32 - CNV 64 - CNV 512 - GMP - SM
M3	INPUT - CNV 32 - CNV 32 - CNV 64 - CNV 64 - CNV 512 - GMP - SM
M4	INPUT - CNV 32 - CNV 32 - CNV 64 - CNV 64 - CNV 128 - CNV 128 - GMP - SM
M5	INPUT - CNV 32 - CNV 64 - CNV 128 - CNV 128 - CNV 256 - CNV 256 - CNV 512 - GMP - SM
M6	INPUT - CNV 32 - CNV 32 - CNV 64 - CNV 64 - CNV 256 - CNV 256 - CNV 512 - GMP - SM
M7	INPUT - CNV 32 - CNV 32 - CNV 128 - CNV 128 - CNV 512 - GMP - FC 512 - SM
M8	INPUT - CNV 32 - CNV 32 - CNV 128 - CNV 128 - CNV 512 - GMP - FC 512 - FC 512 - SM
M9	INPUT - CNV 32 - CNV 32 - CNV 128 - CNV 128 - CNV 512 - GMP - FC 1024 - FC 1024 - SM
M10	INPUT - CNV 32 - CNV 32 - CNV 64 - CNV 128 - CNV 128 - CNV 256 - CNV 512 - GMP - FC 512 - SM
M11	INPUT - CNV 32 - CNV 32 - CNV 64 - CNV 128 - CNV 128 - CNV 256 - CNV 512 - GMP - FC 512 - FC 512 - SM

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11
14x14	45.11%	50.51%	45.77%	39.44%	40.31%	37.32%	53.75%	26.80%	50.07%	34.02%	46.99%
28x28	95.46%	98.18%	98.96%	98.86%	98.58%	97.87%	98.65%	98.72%	97.97%	98.98%	97.32%
50x50	42.23%	61.69%	75.97%	75.31%	71.44%	50.82%	58.10%	35.97%	36.53%	46.48%	43.41%
84x84	37.21%	39.76%	32.00%	28.04%	25.74%	32.17%	24.79%	20.35%	26.59%	26.06%	22.65%
100x100	11.49%	15.19%	20.99%	24.18%	14.94%	15.13%	14.09%	14.42%	12.35%	11.64%	10.70%