Clasificarea Semnelor de Circulatie

Inteligenta Artificiala

Alexandru Andrița
Alex Baciu

Contents

1 Convolutional Neural Networks(CNNs)

1.1 Introducere in Imagini, vazute ca obiecte ce urmeaza sa fie prelucrate

Cand vine vorba de seturi de date ce implica imagini, la fiecare dintre acestea, se observa un pattern: numarul mare de imagini cat si de labels folosite. Pentru a putea folosi eficient informatia din fiecare imagine, exista un tip de Retele Neuronale numit Convolutional Neural Networks, care sunt foarte utile cand trebuie sa ne axam pe structura locala in datele noastre: e.g. date de pixeli, date audio, date video s.a.m.d.

La o simpla analiza, putem calcula numarul dimensiunilor unui input. Presupunand ca imaginea noastra este 300X300 pixeli si este RGB (deci are 3 canale de culoare - rosu, verde si albastru), vom avea $300 \cdot 300 \cdot 3 = 270000$ de dimensiuni. Putem observa astfel cat de supra-dimensionate sunt imaginile.

Un avantaj al imaginilor este ca pixelii care se afla unii langa altii au un coeficient de corelatie mare (i.e. schimbarile intr-un pixel sunt asociate cu schimbarile in pixel-ul de langa), de unde reiese si structura locala de care am amintit mai devreme. In general, ne dorim sa avem invariante la unele variatii, adica modelul sa ramana consistent sau sa obtina aceleasi rezultate in ciuda modificarilor aplicate asupra datelor de intrare. Spre exemplu, invarianta la translatare este importanta pentru ca fara ea, ar insemna ca fiecare imagine sa contina acelasi obiect pus in aceiasi pozitie astfel incat el sa fie recunoscut (cu invarianta, modelul va recunoaste obiectul indiferent de pozitia in care se afla).

1.2 Campul receptiv local

In ceea ce priveste campul receptiv local, putem afirma ca pixelii care se afla unii langa altii au rata de corelatie mare. Printre patch-urile de baza, putem reaminti marginile sau colturile, care apar in toate pozitiile imaginii. Pentru a detecta aceste patch-uri, putem folosi o retea cu camp receptiv mic. De unde si reiese rolul campului local i.e. sa conecteze reteaua cu un patch al imaginii folosind o matrice de weights, numita si filtru sau kernel.

1.3 Weight Sharing (Sharing-ul de Greutati)

Cum patch-urile de baza apar la toate pozitiile unei imagini, putem folosi kernel-ul in fiecare pozitie din respectivele pozitii.

Weight Sharing - Sharing-ul de Greutati

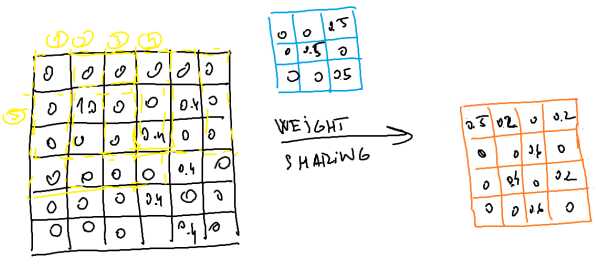

Figure 1

1.4 Convolutie

O convolutie este aplicarea kernel-ului la toate pozitiile imaginii. In general, la fiecare output (dupa o convolutie), aplicam o functie de activare. De obicei, este folosita Rectified Linear Units sau ReLU, pe care o vom explica in capitolele ce urmeaza. Un layer (strat) in CNN este reprezentat de aplicarea kernel-ului urmat de functia de activare.

Pentru a pastra shape-ul input-ului dupa convolutie, putem aplica padding. Padding-ul poate fi facut urmand una din urmatoarele tehnici:

- zero-padding: padding cu 0-uri;
- repeat-padding: padding cu valorile chenarului exterior;
- alte metode: padding cu suma valorilor, media valorilor etc.

1.5 Pooling

Prin intermediul pooling-ului, ajungem sa reducem dimensiunea input-ului. Pooling-ul va duce la pierdere de informatie, dar va contribui la marirea campului receptiv odata ce avansam in retea (mergem tot mai adanc). Cu alte cuvinte, chiar daca la final vom fi pierdut informatie, esentialul (cea mai importanta informatie) a ramas.

Exista mai multe tipuri de pooling:

- average pooling: dintr-o dimensiune *kxk*, va fi luata media valorilor;
- max-pooling: dintr-o dimensiune *kxk*, va fi luata valorea maxima dintre respectivele valori;
- n-max pooling: vom facem media dintre primele *n* valori in ordinea descrescatoare.

1.6 Flattening

Flattening-ul este o metoda prin care aducem vectorii multi-dimensionali la vectori unidimensionali.

```
numpy array cu shape (2,3): [[4,7,1],[9,5,2]]
dupa Flatenning: [4,7,1,9,5,2]
```

1.7 Dense

Dense sunt layerele pentru care unitatile sunt complet conectate cu layer-erele anterioare si/sau precedente. (Fig 3)

1.8 Rescaling

Rescaling are rolul de a reduce datele la un anumit interval specificat. Normalizarea datelor intr-un anumit interval, cum ar fi [0,1] faciliteaza training-ul cat si convergenta modelelor (din punct de vedere computational, operatiile vor fi mai rapide). Este foarte important ca toate datele sa fie normalizate, nu doar o parte din ele (e.g. atat output-ul modelului, cat si valorile adevarate cu care e comparat output-ul produs).

Max Pooling

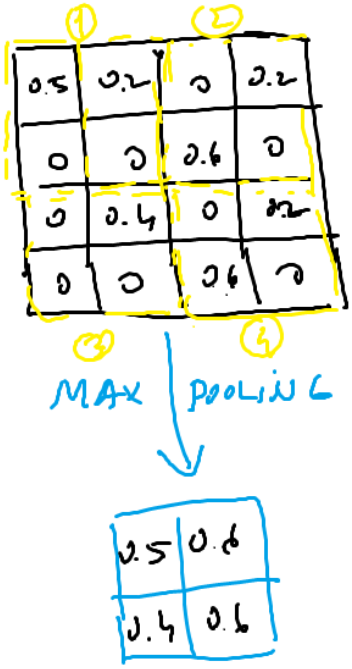

Figure 2

Exemplu Dense layer

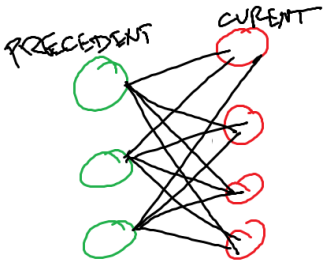

Figure 3

1.9 Niveluri Multiple in Convolutii

O retea de tip CNN are in general mai multe layere de convolutions, nonlinearitate sau pooling. Fiecare camp receptiv genereaza un nou channel (sau feature map) pentru urmatorul layer, complexitatea feature-urilor detectate crescand de la un layer la altul. Un simplu exemplu ar fi: la primul layer sunt identificate marginile, la urmatorul layer sunt combinate in forme complexe s.a.m.d.

1.10 Overfitting. Underfitting. Optimizatori. Functii de activare. Loss functions. Metode de Reguralizare

Overfitting se intampla atunci cand modelul invata fiecare detaliu al datelor de train si are performanta foarte slaba pe datele de test. In acest caz, modelul este prea complex.

Am inclus si cateva plot-uri in proiect pentru a monitoriza modelul si evoluatia acestuia. Pe grafic se poate observa foarte usor ca va ajunge in overfitting cand eroarea de validare creste brusc si eroarea de trainig scade.

La cealalta extrema, underfitting se intampla cand modelul este prea simplu si nu reuseste sa invete nici macar datele de train.

Rectified Linear Unit sau ReLU este o functie de activare asemenea lui Sigmoid, fiind definita astfel: $\forall x \in \mathbb{R} \quad ReLU(x) = \max(0, x)$

Functia de activare ReLU

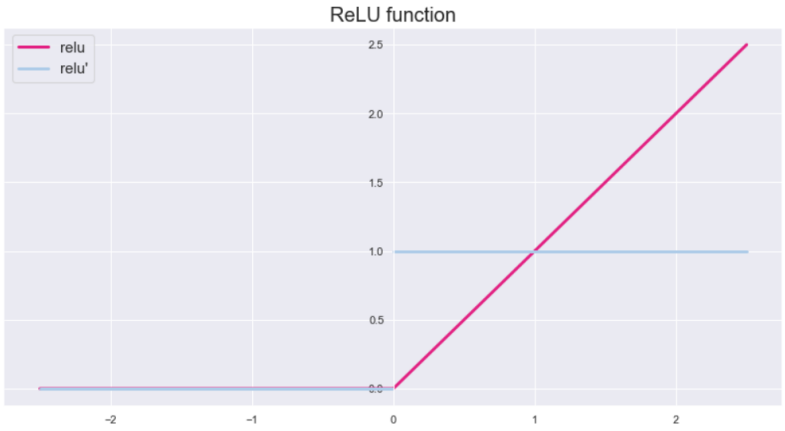

Figure 4

Softmax are rolul de a converti un set de valori intr-o distributie de probabilitati, fiind definita astfel: $Softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

K = numarul de clase

z_i = scorul clasei i

Functia de activare Softmax

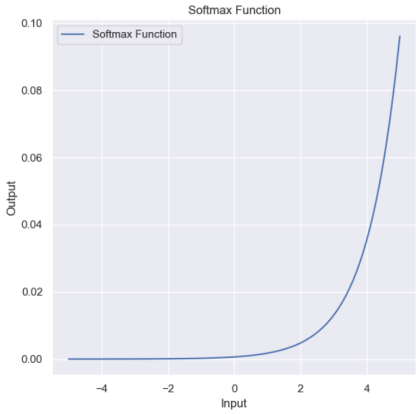

Figure 5

Loss functions sunt niste metrici matematice care compara output-ul modelului cu predictia reala. Printre ele, se numara si Cross Entropy Loss, o functie folosita atunci cand vine vorba de multiclasificare. Ea calculeaza probabilitati, iar cu cat probabilitatea pentru fiecare dintre clase este mai mare, cu atat predictia este mai buna.

Optimizatorii au rolul de a modifica valorile parametrilor in timpul training-ului, cu scopul minimizarii loss-ului. Am folosit Adam optimizer, fiind unul dintre cei mai cunoscuti algoritmi pentru un asa tip de task. Desigur, modelul poate functiona si fara optimizator, dar performanta ar fi destul de slaba.

Pentru ca overfitting-ul sa fie evitat, pot fi folosite tehnici de regulizare. Printre else se numara si dropout-ul. Ideea de baza la dropout este urmatoarea: printr-o selectie aleatorie, cateva unitati sunt excluse din training. Valorea procentului p poate sa difere, insa este recomandat ca dropout-ul sa fie aplicat la final pentru a ne asigura ca pana in acel punct, modelul a invatat informatiile esentiale.

In Figura 6, in partea stanga, se poate observa o retea neuronala standard in timp ce in partea dreapta se poate observa o retea neuronala dupa ce dropout-ul a fost aplicat. Un aspect important de mentionat este ca dropout-ul va fi aplicat numai intre 2 layere, nu va fi aplicat automat in toata reteaua.

Dropout

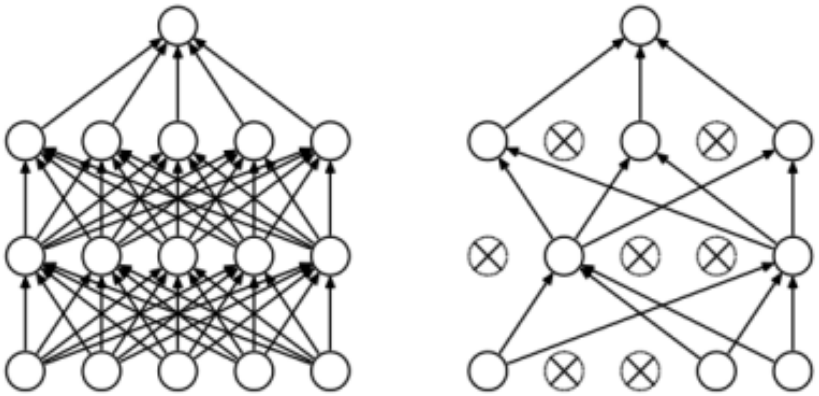

Figure 6

2 Setul de date

Imaginile au dimensiunea de 32x32 pixeli, iar in total sunt 43 de labels. Dupa ce am incarcat dataset-ul, am facut split in cele 3 categorii: train data, validation data si testing data. Am folosit validation data pentru a fi siguri ca modelul nu ajunge in overfitting cat si pentru hyperparameter tunning (modificarea 'fina' a valorilor parametrilor modelului).

In ceea ce priveste modul in care sunt distribuite label-urile, fiecare dintre ele are un indice. Spre exemplu, semnul de stop are label-ul 14, indicatorul cu limita de viteza 70 are label 4, iar indicatorul cu obligatoriu de virat la dreapta are label-ul 33 (asa cum reiese si din Figurile 7 si 8).

Labels investigare 1

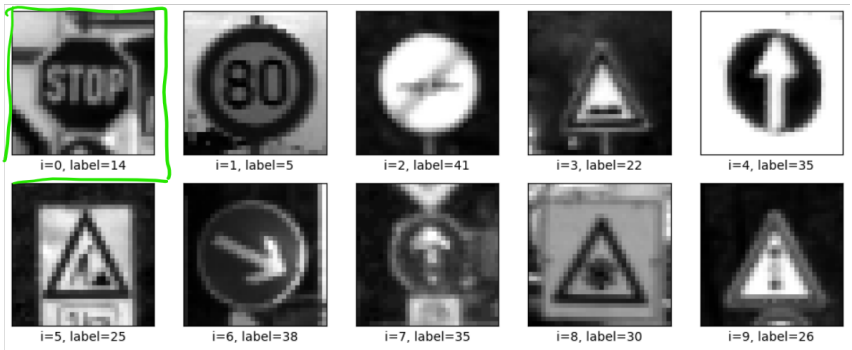

Figure 7

Labels investigare 2

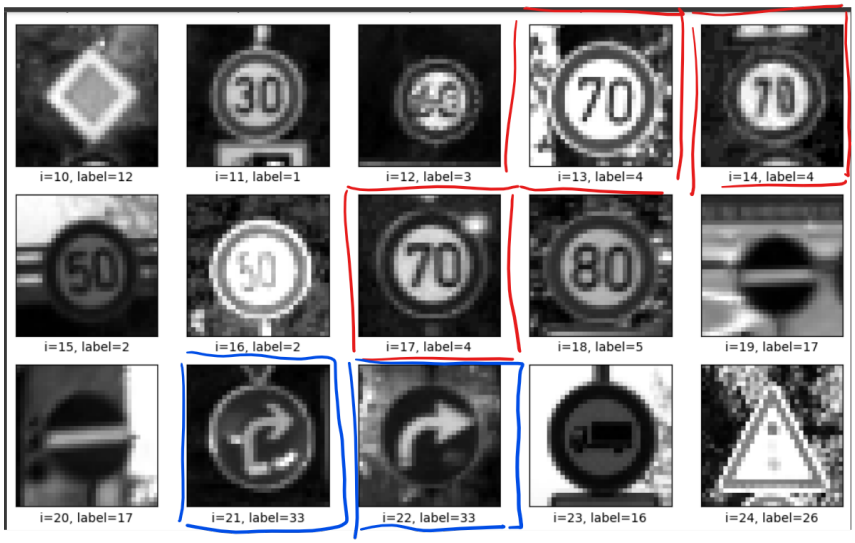

Figure 8

3 Arhitecturile folosite

Ne-am folosit de doua arhitecturi, plecand de la una simpla si ajungand la una complexa.

3.1 Prima arhitectura

Arhitectura 1

```
model = Sequential([
    Rescaling(1, input_shape=(32, 32, 1)),
    AveragePooling2D(pool_size=(4, 4)),
    Conv2D(filters=6, kernel_size=(5, 5), activation='relu'),
    AveragePooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(units=30, activation='relu'),
    Dense(units=43, activation='softmax')
])
```

Figure 9

Variatia shape-ului de la input la output. Numarul de parametrii ai modelului.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 32, 32, 1)         0

 average_pooling2d (Average  (None, 8, 8, 1)           0
 Pooling2D)

 conv2d (Conv2D)             (None, 4, 4, 6)           156

 average_pooling2d_1 (Avera  (None, 2, 2, 6)           0
 gePooling2D)

 flatten (Flatten)           (None, 24)                0

 dense (Dense)               (None, 30)                750

 dense_1 (Dense)             (None, 43)                1333

=================================================================
Total params: 2239 (8.75 KB)
Trainable params: 2239 (8.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 10

Sequential (secvential) are rolul de a executa pe rand fiecare layer trecut in lista. Numarul total de parametrii cat si numarul individual la fiecare layer poate raspunde la intrebari legate de complexitatea modelului.

Prima arhitectura are un accuracy de 68.46% la validare. Perfomanta superioara fata de random (50%), dar inca este destul loc pentru imbunatatire.

3.2 A doua arhitectura

Arhitectura 2

```
model = Sequential([
    Rescaling(1, input_shape=(32, 32, 1)),
    Conv2D(filters=6, kernel_size=(5, 5), activation='relu'),
    AveragePooling2D(pool_size=(2, 2)),
    Conv2D(filters=16, kernel_size=(5, 5), activation='relu'),
    AveragePooling2D(pool_size=(2, 2)),
    Conv2D(filters=120, kernel_size=(5, 5), activation='relu'),
    Dropout(0.2),
    Flatten(),
    Dense(units=120, activation='relu'),
    Dense(units=43, activation='softmax')
])
```

Figure 11

Variatia shape-ului de la input la output. Numarul de parametrii ai modelului.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 32, 32, 1)         0

 conv2d_1 (Conv2D)           (None, 28, 28, 6)         156

 average_pooling2d_2 (Avera  (None, 14, 14, 6)         0
 gePooling2D)

 conv2d_2 (Conv2D)           (None, 10, 10, 16)        2416

 average_pooling2d_3 (Avera  (None, 5, 5, 16)          0
 gePooling2D)

 conv2d_3 (Conv2D)           (None, 1, 1, 120)         48120

 dropout (Dropout)           (None, 1, 1, 120)         0

 flatten_1 (Flatten)         (None, 120)               0

 dense_2 (Dense)             (None, 120)               14520

 dense_3 (Dense)             (None, 43)                5203

=================================================================
Total params: 70415 (275.06 KB)
Trainable params: 70415 (275.06 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 12

Se poate observa ca avem un model mult mai complex (deci mai costisitor cand vine vorba de costuri si resurse), numarul de parametri a crescut substantial. Cu toate acestea, performanta modelului s-a imbunatatit considerabil, ajungand sa aiba un accuracy de 99.89% la validare.

3.3 Salvarea si incarcarea modelului

Ambele modele au fost salvate local si apoi incarcate pentru a fi folosite la predictie.

Salvarea primului model

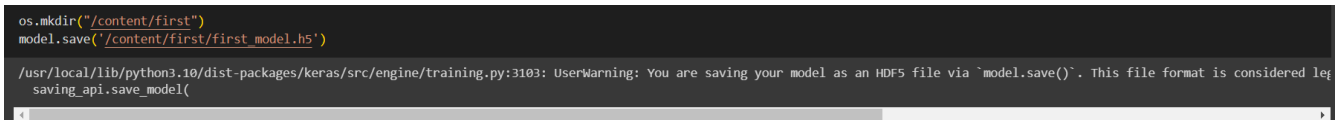

Figure 13

Salvarea celui de-al doilea model

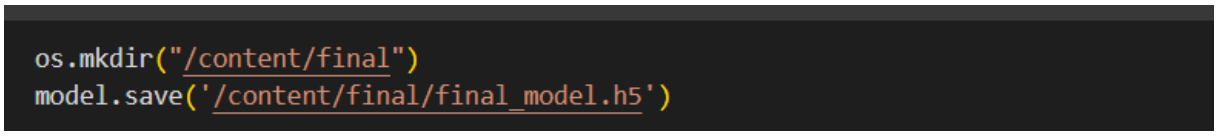

Figure 14

4 Predictie pe unseen data

Modele anterior antrenate au fost incarcate pe rand si au fost pe rand testate pe unseen data (imagini nemaivazute).

```
model = load_model("/content/first_model/first_model.h5")
preds = model.predict(test_images)

plot_preds(preds)
```

Figure 15

```
model = load_model("/content/final_model/final_model.h5")
preds = model.predict(test_images)

plot_preds(preds)
```

Figure 16

5 Bibliografie

- Prezentare Arhitectura LeNet
- Setul de date
- Setul de date exact descarcat
- Articol ce contine un model deja antrenat
- Understanding Dropout with the Simplified Math behind it | by Chitta Ranjan | Towards Data Science