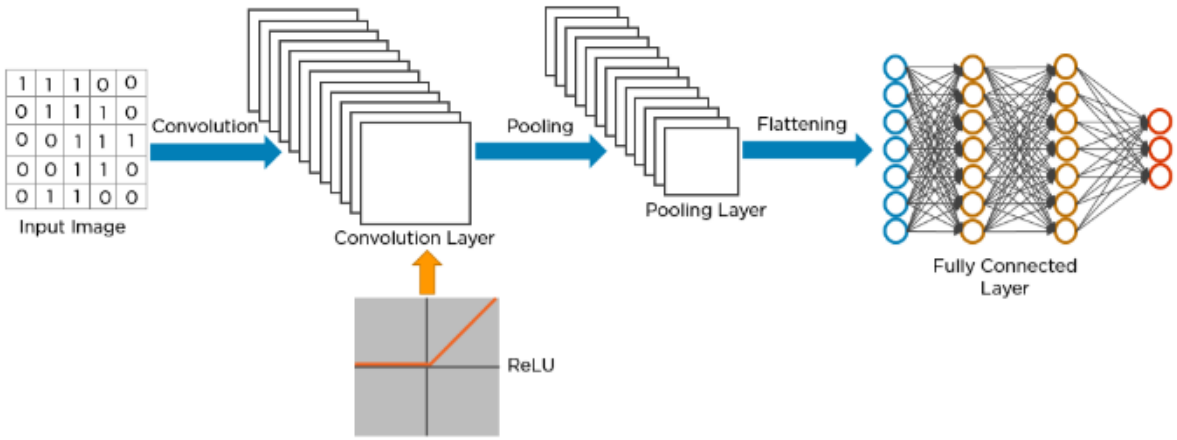
# Convolutional neural network

O rețea neuronală convoluțională este o rețea neuronală avansată care este în general utilizată pentru a analiza imaginile vizuale prin procesarea datelor cu topologie bazata pe matrice. Este, de asemenea, cunoscut sub numele de ConvNet. O rețea neuronală convoluțională este utilizată pentru a detecta și clasifica obiectele dintr-o imagine.

Se vor parcurge urmatoarele etape:

1. Etapa de convolutie
2. Etapa de pooling
3. Etapa de nivelare (flattening)
4. Etapa de conectare completa




*Imagine preluata de pe www.simplilearn.com*

Mindruleanu Matei Daniel

Citirea si procesarea datelor:

Avem fisierele de input train.txt, validation.txt si test.txt, le vom citi astfel:

```
f = open("train.txt", 'r')
v_imagini = []
v_indici = [] #label

for i in f:
    image = imread(f"train/{i.split(',')[0]}")
    image = list(image)
    indice = int(i.split(',')[1].split()[0])
    v_imagini.append(image)
    v_indici.append(indice)
f.close()
v_indici = keras.utils.to_categorical(v_indici)
```

In v_imagini salvam imaginile din train(cu ajutorul functiei imread), iar in v_indici – indicii aceatora. Cu ajutorul functiei keras.utils.to_categorical transformam v_indici intr-o matrice de clase binare.

```
f = open("validation.txt", 'r')
v_imagini_validare = []
v_indici_validare = [] #label

for i in f:
    image = imread(f"validation/{i.split(',')[0]}")
    indice = int(i.split(',')[1].split()[0])
    v_imagini_validare.append(image)
    v_indici_validare.append(indice)
f.close()
v_indici_validare = keras.utils.to_categorical(v_indici_validare)
```

In v_imagini_validare salvam imaginile din validation(cu ajutorul functiei imread), iar in v_indici_validare – indicii aceatora. Cu ajutorul functiei keras.utils.to_categorical transformam v_indici_validare intr-o matrice de clase binare.

```
v_imagini = np.asarray(v_imagini)
v_imagini = v_imagini.reshape(v_imagini.shape[0], 32, 32, 1)

v_imagini_validare = np.asarray(v_imagini_validare)
v_imagini_validare = v_imagini_validare.reshape(v_imagini_validare.shape[0], 32,
32, 1)
```

Transformam v_imagini intr-un assaray pentru a-i puteada reshape, acesta fiind incarcat cu imagini de forma 32x32. Facem acelasi lucru si pentru v_imagini_validare.

```
f = open("test.txt", 'r')
v_imagini_test = []
v_path = []

for i in f:
    image = imread(f"test/{i.split()[0]}")
    image = list(image)
    v_imagini_test.append(image)
    v_path.append(i.split()[0])
f.close()

v_imagini_test = np.asarray(v_imagini_test)
v_imagini_test = v_imagini_test.reshape(v_imagini_test.shape[0], 32, 32, 1)
```

Procedam la fel si cu datele din test, salvandu-le in v_imagini_test(fisierul de test contine numai imagini, fara labeluri). Dupa care dam reshape array-ului, cum am procedat si la cele 2 array-uri de imagini precedente.

Convolutional neural network:

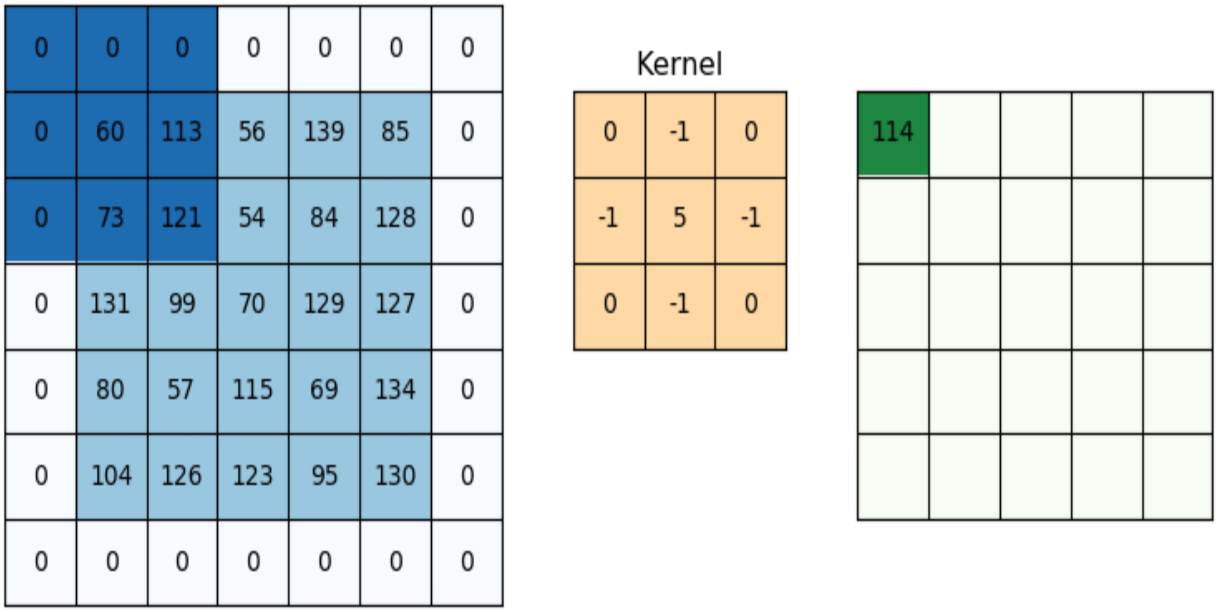
```
model = keras.models.Sequential() #creem obiectul model
model.add(keras.layers.Input((32, 32, 1))) #intantiem modelul cu
un input de 32x32 pixeli si greyscale
model.add(keras.layers.BatchNormalization()) #normalizam modelul pe batch
```

Convolutia:

```
model.add(keras.layers.Conv2D(600, 5, activation='relu'))
model.add(keras.layers.Conv2D(600, 5, activation='relu'))
```

In convolutie se iau matrice din imagini, care se trec prin 600 de filtre de dimensiune 5x5 astfel: se scot matrice de 5x5 care se inmultesc cu filtrul(element pe element, nu matrice cu matrice), apoi, suma elementelor din noile matrice(dupa inmultire) va fi un element din matricea rezultat. Functia ‘relu’ va transforma toate numerele negative in 0.

Reprezentare viziala:




*Imagine preluata de pe* https://www.pyimagesearch.com

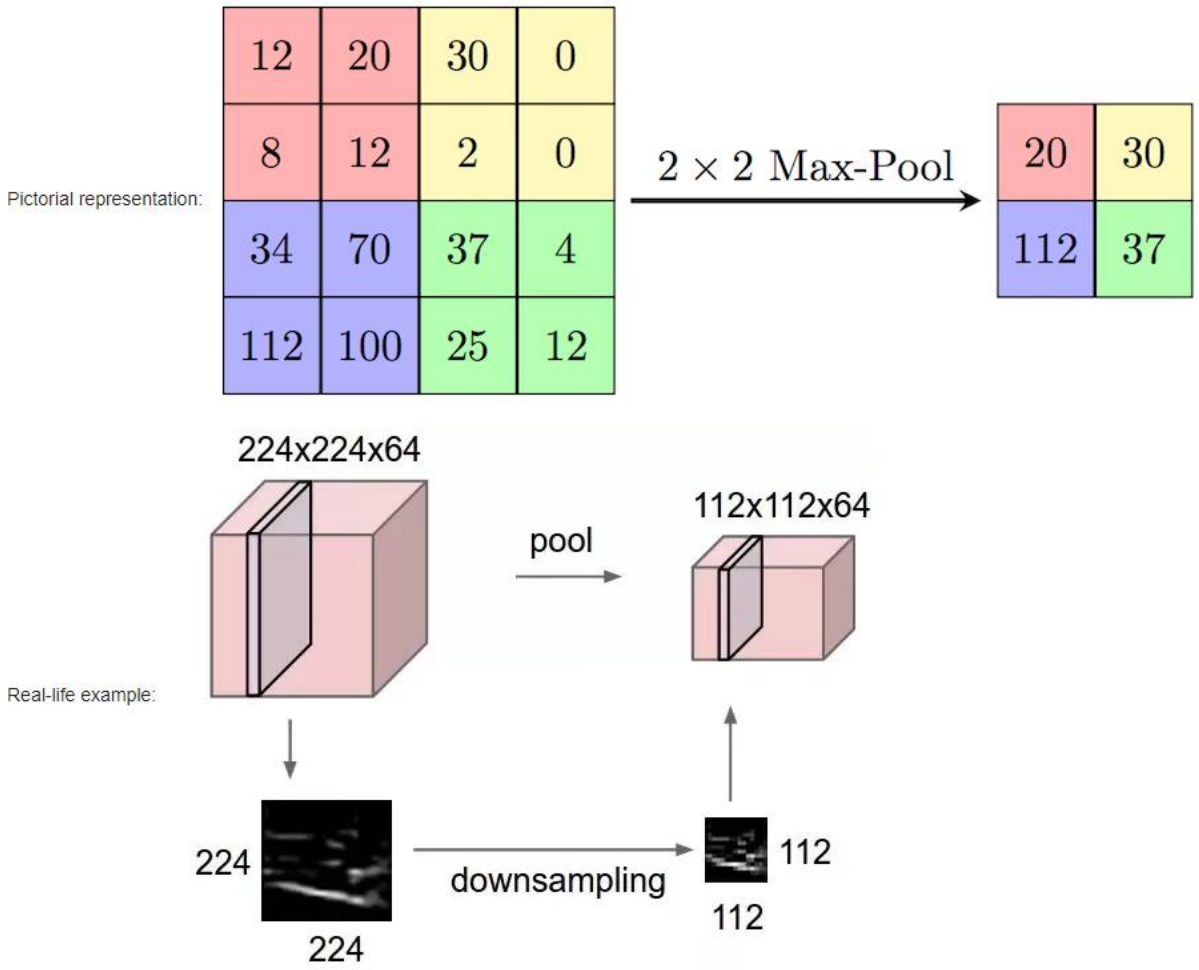
Mindruleanu Matei Daniel

Poolingul:

```
model.add(keras.layers.MaxPool2D())
```

Matricea rezultata dupa convolutionalele este trecuta prin pooling. Se iau mini-matrice de 2x2(default) din matricea mare, iar maximul dintre aceste 4 elemente este adaugat in matricea rezultat.

Reprezentare vizuala:




Imagine preluata de pe *https://computersciencewiki.org*

Dropout:

```
model.add(keras.layers.Dropout(0.4))
```

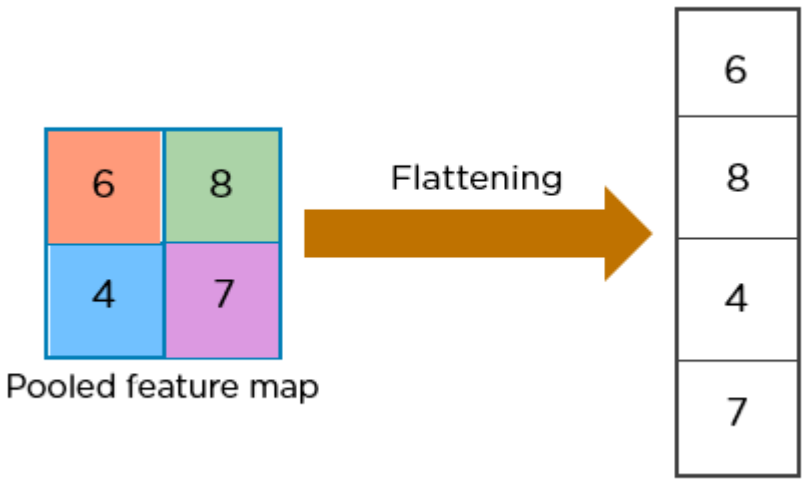
Folosim dropout pentru a renunta la 40% din datele adunate pana in acest moment(pentru a evita overfit-ul).

Flatten:

```
model.add(keras.layers.Flatten())
```

Cu ajutorul flatten-ului punem toate elementele din matrice sub forma unui array.

Reprezentare vizuala:



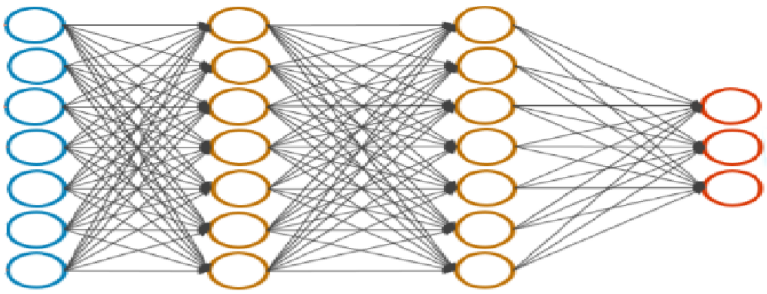

*Imagine preluata de pe* https://www.simplilearn.com

Dense(conectare completa):

```
model.add(keras.layers.Dense(350, activation='relu'))
model.add(keras.layers.Dense(350, activation='relu'))
```

In aceasta se va aplica un algoritm de Multilayer Perceptron pe array-ul ramas dupa flatten. Perceptronul este un algoritm folosit pentru invatarea supravegheata a clasificatorilor binari. MLP-ul conecteaza fiecare perceptron de pe primul layer cu toti de pe urmatorul.

Reprezentare vizuala:



*Imagine preluata de pe* www.simplilearn.com

Continuare:

```
model.add(keras.layers.Dropout(0.4))

model.add(keras.layers.Dense(9, activation='softmax'))
```

Mai folosim o data dropout, dupa care, folosim functia softmax pentru a alege label-ul din care face parte poza actuala.

```
model.compile(optimizer="adamax", loss="categorical_crossentropy", metrics=["acc"
])
```

Compilam modelul folosind optimizatorul adamax, pe functia de loss categorical_crossentropy, punand accentual pe acuratete.

```
model.fit(v_imagini, v_indici, epochs=30, verbose=2, validation_data=(
    v_imagini_validare, v_indici_validare), use_multiprocessing=True, batch_size=
64)
```

Antrenam modelul, setam 30 de epoci, verbose 2 pentru a afisa clar in consola acuratetea pe fiecare epoca si ii dam un batch_size de 64.

Output:

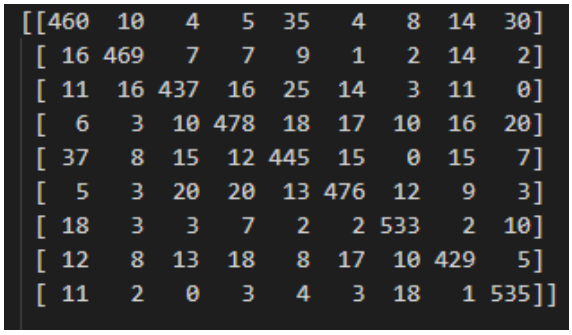
```
predictii = model.predict_classes(v_imagini_test)
o = {"id" : v_path,'label' : predictii}
output = pd.DataFrame(data=o)

output = output.set_index("id")

output.to_csv("output_final.csv")
```

Creem un dictionar in care tinem id-ul, ca path catre imagine(v_path a fost salvat mai sus in citire) si labelul pe care l-am aflat cu ajutorul metodei predict_classes din keras. Creem variabima output de tip pd(panda) cu ajutorul datelor din dictionar, ii punem ca index – idurile si returnam sub forma unui fisier csv(coma separated values).

Eficienta algoritmului propus este una de 0.86640 .

Timpul de rulare : 1246.919 secunde.

Matricea de confuzie:



```
[[460  10   4   5  35   4   8  14  30]
 [ 16 469   7   7   9   1   2  14   2]
 [ 11  16 437  16  25  14   3  11   0]
 [  6   3  10 478  18  17  10  16  20]
 [ 37   8  15  12 445  15   0  15   7]
 [  5   3  20  20  13 476  12   9   3]
 [ 18   3   3   7   2   2 533   2  10]
 [ 12   8  13  18   8  17  10 429   5]
 [ 11   2   0   3   4   3  18   1 535]]
```

# K-nearest neighbors

Metoda celor mai apropiați vecini este un algoritm neparametric de clasificare. Acesta clasifica fiecare obiect in functie de clasele vecinilor acestuia.

Citirea, procesarea datelor si outputul sunt implementate la fel ca la CNN.


*Imagine preluata de pe* https://towardsdatascience.com/

KNN:

```
neigh = KNeighborsClassifier(n_neighbors = 9)
neigh.fit(v_imagini, v_indici)
```

Construim neigh apeland KNeighborsClassifier cu 9 vecini. Dupa aceea antrenam masina folosind datele din train.

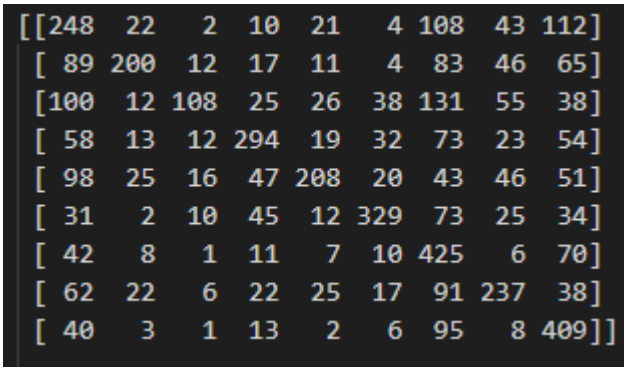
```
predictii_cm = neigh.predict(v_imagini_validare)
cm = confusion_matrix(v_imagini_validare, predictii_cm)
```

Cream matricea de confuzie atat cu ajutarul functiei predict din neighbors, cat si cu ajutorul functiei confusion_matrix din metrics.

Eficienta algoritmului propus este una de 0.4916 .

Timpul de rulare : 442.951secunde.

Matricea de confuzie:



```
[[248  22   2  10  21   4 108  43 112]
 [ 89 200  12  17  11   4  83  46  65]
 [100  12 108  25  26  38 131  55  38]
 [ 58  13  12 294  19  32  73  23  54]
 [ 98  25  16  47 208  20  43  46  51]
 [ 31   2  10  45  12 329  73  25  34]
 [ 42   8   1  11   7  10 425   6  70]
 [ 62  22   6  22  25  17  91 237  38]
 [ 40   3   1  13   2   6  95   8 409]]
```