Deep Hallucination Classification

Tindeche Alexandru - grupa 231

1. Introducere

Tema competitiei este clasificarea unor imagini produse de un model de tip "deep generative" in 96 de clase diferite folosind un model de tip "deep classification".

2. Descrierea setului de date

Imaginile apartin a 96 de clase diferite, iar fiecare imagine este de dimensiune 64x64x3. Setul de date este impartit in 2 folduri de date: train si test. Imaginile, desi uitandu-ma prin cateva par a fi 64x64 am ales sa le scalez pe toate la aceeasi dimensiune pentru a fi sigur. Imaginile au culori si contrast bune, asa ca am luat decizia de a nu le modifica in acest sens. Totusi, pentru ca vrem sa antrenam un model deep classification, trebuie sa augumentam datele aplicand transformari de tip rotatie, zoom, flip, etc. pentru a avea un set de date mai mare si mai divers si pentru a evita overfitting-ul.

3. Modelul de deep learning

Mai intai m-am gandit sa incerc ceva de tip NaiveBayes sau KNN pentru ca au fost incercate si explicate la laborator, dar cautand pe internet si facand putin research, am descoperit ca nu au o acuratete atat de buna cand vine vorba de clasificare de imagini, asa cum o au alte modele precum retelele neuronale convolutionare (CNN). Am ales sa nu folosesc un model de tip Multilayer Perceptron (MLP) pentru ca numeroase articole mi-au dat de inteles ca este mai putin eficient decat un model de tip CNN pe atat de multe clase si cu imagini relativ complexe.

Mod de lucru in constructia unui model de deep learning

Orice algoritm de deep learning trebuie sa urmeze urmatorii pasi:

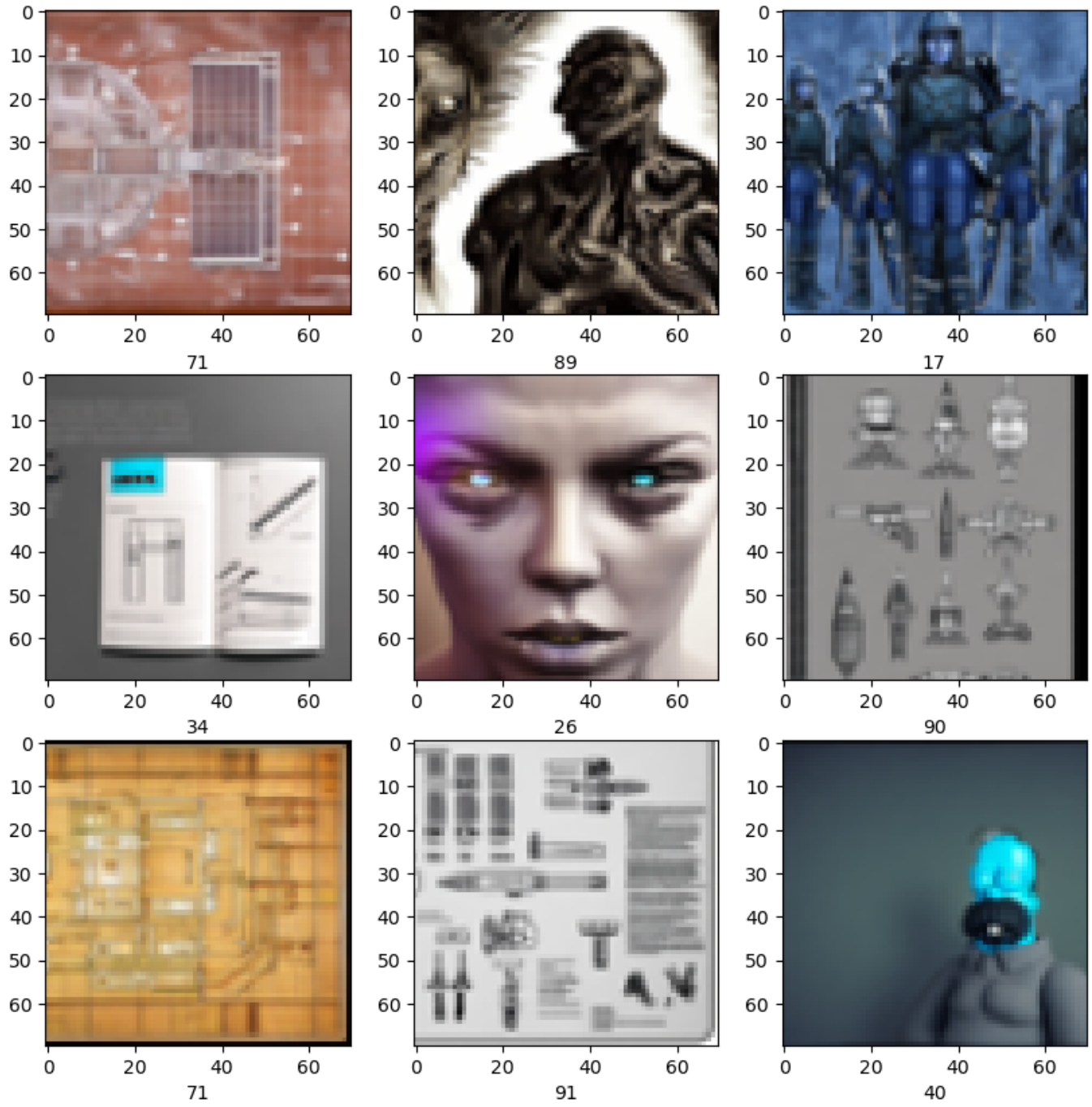
1. Citirea datelor
2. Preprocesarea datelor
3. Modelul in sine
4. Antrenarea modelului
5. Predictia

1. Citirea datelor

Primul lucru pe care l-am facut a fost sa citesc datele din csv, si anume sa colectez imaginile si etichetele lor. Pentru fiecare tip de date de intrare (training, validation, test) avem un fisier csv care contine numele imaginii si eticheta ei, bineinteles, mai putin pentru datele de test, unde avem doar numele imaginilor. In csv-urile "train.csv" si "val.csv" avem 2 coloane: numele imaginii si eticheta ei. Am folosit biblioteca "pandas" pentru a citi fisierul csv, apoi cu iloc (functie tot din pandas) am luat doar coloana cu numele imaginii si am salvat-o intr-o lista. Am facut acelasi lucru si pentru etichete. Apoi, intr-un for am luat fiecare nume de imagine si l-am

incarcat din folderul ce cuprinde toate imaginile, le-am citit folosing biblioteca "cv2", le-am facut resize la o dimensiune de 64x64, le-am normalizat impartind fiecare pixel la 255, si le-am dat append la lista de imagini. Am facut acelasi lucru si pentru etichete, bineinteles fara resize si normalizare, pentru ca sunt int-uri.

```
    train_data = pd.read_csv('/kaggle/input/unibuc-dhc-2023/train.csv')

# Shuffle the data

train_data = shuffle(train_data)

# Take the images and labels
# Images are on the first column and labels are on the second column

images = train_data.iloc[:,0]
labels = train_data.iloc[:,1]

train_images = []
train_labels = []

for image in images:
    img = cv2.imread('/kaggle/input/unibuc-dhc-2023/train_images/'+image)
    img = cv2.resize(img, (64, 64))
    img = np.array(img)
    img = img/255.0
    train_images.append(img)
```



2. Preprocesarea datelor

2.1 Data augmentation

Ce este data augumentation?

Data augmentation este o tehnica de preprocesare a datelor care consta in modificarea datelor de antrenare prin aplicarea unor transformari precum rotatie, zoom, flip, etc. pentru a avea un set de date mai mare si mai divers si pentru a evita overfitting-ul.

Am adaugat astfel la imaginile citite noile imagini generate prin data augmentation, astfel marind setul de antrenare.

2.2 Preprocesarea datelor

Pentru fiecare imagine am redus valorile pixelilor de la 0-255 la 0-1 pentru a avea o mai buna convergenta a modelului (normalizare), modelele de deep learning fiind sensibile la valori mari.

De asemenea am flosit LabelEncoder pentru a transforma etichetele din string-uri in numere (int-uri).

Nu am impartit imaginile de train in train, validation si test, pentru ca setul nostru de date de pe Kaggle, deja ne da 3 folduri cu date diferite, asa ca le-am folosit pe acelea pentru a maximiza numarul de date de antrenare.

3. Modelul de deep learning

Arhitectura modelului; Etapele de constructie a modelului

1. Extragem caracteristici din imaginile de antrenare folosind layere de convolutie si pooling

- Convolutie
- Functie de activare
- Pooling

2. Clasificam imaginile folosind layere dense

- Flatten
- Dense
- Un layer dense cu numarul de neuroni egal cu numarul de clase si functia de activare

Am folosit keras.Sequential care este un API ce ne ajuta sa construim un model CNN si sa punem layere in el mai usor.

3.1 Straturile de convolutie

Concepte

Ce inseamna convolutie?

Convolutie = o operatie matematica ce ne permite sa obtinem un nou set de date din combinarea a doua seturi de date. In matematica convolutia ne arata cum doua functii se influenteaza una pe cealalta. In deep learning, convolutia este o operatie ce presupune aplicarea unui filtru pe o imagine pentru a obtine o noua imagine care contine doar caracteristicile importante ale imaginii originale (Feature Map). In cazul nostru avem imagini reprezentate ca matrice de pixeli, iar filtrul este o matrice de numere.

Ce este un filtru?

Un filtru ne da posibilitatea de a extrage caracteristici dintr-o imagine, de exemplu o linie orizontala, un arc de cerc, etc. Cand aplicam un filtru pe o imagine obtinem o noua imagine care contine doar caracteristicile importante ale imaginii originale (Feature Map). Filtrul este o matrice de numere, cunoscuta ca si weight care la inceput are numere random, dar pe masura ce modelul invata, aceste numere se schimba pentru a obtine un feature map cat mai bun si cu caracteristici cat mai importante.

Pe aceste feature map-uri aplicam functia de activare precum relu, sigmoid, etc. pentru a obtine o noua imagine care sa contina doar caracteristicile importante ale imaginii originale.

Ce inseamna stride?

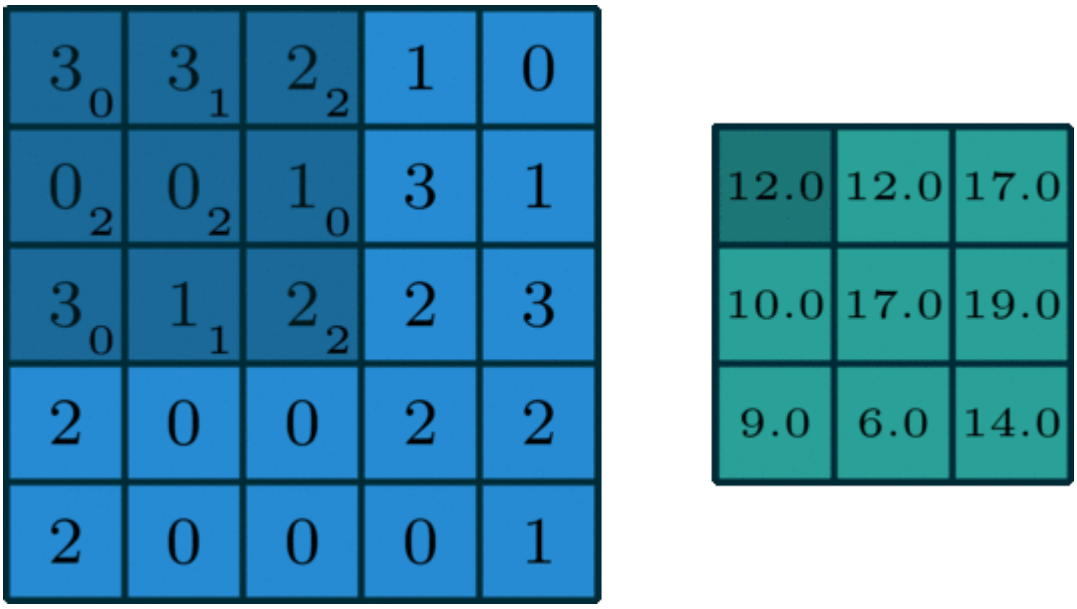
Stride-ul este numarul de pixeli cu care ne deplasam cand aplicam filtrul pe imagine.

Ce inseamna padding?

Padding-ul consta in adaugarea de pixeli in jurul imaginii pentru a nu pierde informatii cand aplicam filtrul pe imagine.

Concluzie: Straturile de convolutie dintr-un CNN iau kernel-ul (matricea - filtru) si il aplica in repetate randuri pe imaginea de intrare si calculeaza produsul scalar dintre kernel si valoarea pixelilor imaginii. Filtrul se schimba la ficare epoca "invatand" sa recunoasca anumite caracteristici din imagine.

Operatia de convolutie dupa aceste cunostiinte dobandite



Sursa imaginii: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Dupa cum se observa in acest GIF, filtrul se deplaseaza pe imagine cu un anumit stride, si cu padding corespunzator si calculeaza produsul scalar dintre kernel si valoarea pixelilor imaginii pentru a obtine un feature map.

Ce inseamna pooling?

Operatia de pooling reduce dimensiunile feature-map-ului extrase de straturile de convolutie si pentru a scadea capcitatea necesara pentru procesarea datelor. In majoritatea cazurilor in care avem nevoie de un CNN pentru clasificarea imaginilor, trebuie sa aplicam mai multe straturi de convolutie care extrag caracteristici intr-un feature map care poate avea o dimensiune destul de mare. Astfel ca operatia de pooling ne ajuta sa reducem dimensiunea feature-map-ului pentru a reduce numarul de parametrii si timpul de antrenare. Operatia de pooling imparte acest map in diferite zone mici pe care aplica diferite functii de agregare precum min, max, average, etc. Cele mai comune tipuri de pooling sunt max pooling si average pooling.

Ce inseamna max pooling?

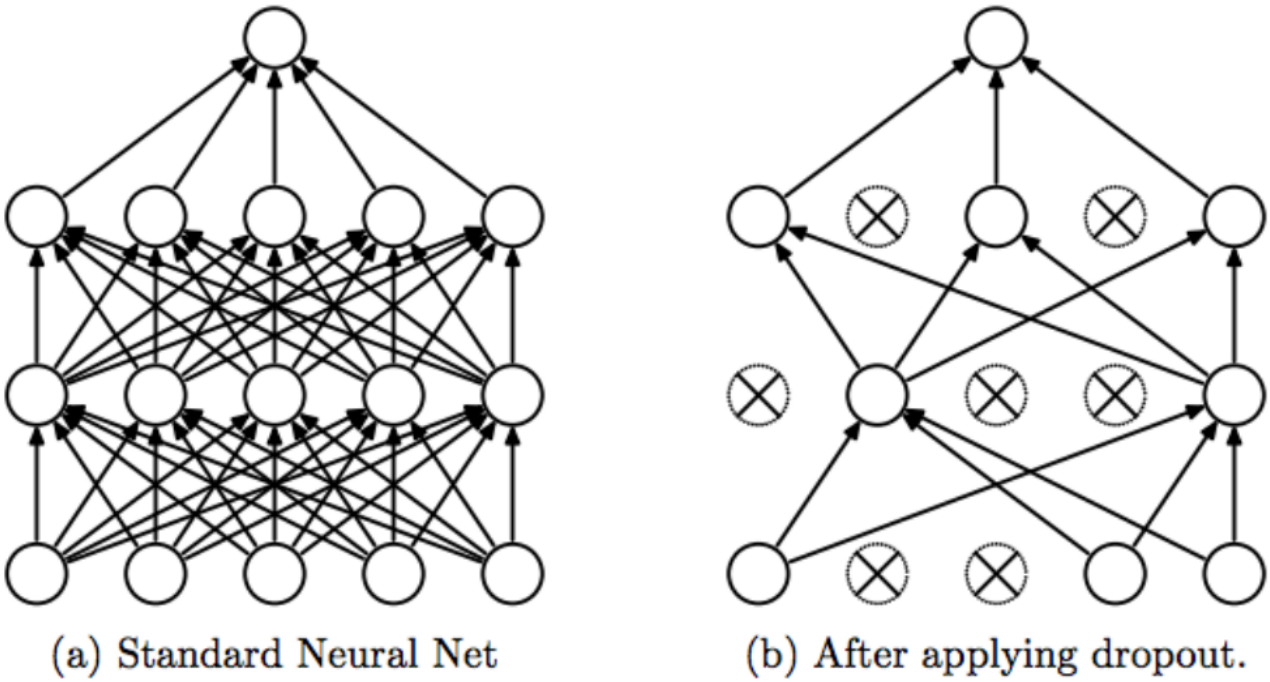
Max pooling-ul consta in reducerea dimensiunii imaginii prin selectarea valorii maxime sau dintr-o zona a imaginii. Astfel ne dam seama daca o anumita caracteristica este relevanta sau nu. Acest strat de max pooling cauta regiuni semnificative din imagine si le retine, cu alte cuvinte face modelul invariant la translatare.

Ce este global average pooling?

Spre deosebire de max pooling care selecteaza pixelii mai "stralucitori", average pooling face o medie a pixelilor dintr-o zona a imaginii, asadar o face mai "smooth".

Ce inseamna dropout?

Dropout-ul consta in eliminarea unor neuroni dintr-un layer cu o anumita probabilitate. Acesta seteaza weight-ul unei parti dintre neuroni la 0 (practic ii reseteaza) Astfel fortam modelul sa distribuie selectia de caracteristici cat mai egal intre neuroni si sa invete feature-uri cat mai independente. Acesta este unul dintre cele mai populare metode de regularizare a retelelor neuronale.



(a) Standard Neural Net (b) After applying dropout.

Sursa imaginii: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

Ce inseamna batch normalization?

Batch normalization-ul este o tehnica de preprocesare a datelor care consta in normalizarea output-ului unui layer pentru a avea o distributie normala. Astfel, modelul va fi mai stabil si va avea o convergenta mai rapida.

Ce inseamna flatten?

Flatten-ul este o tehnica de preprocesare a datelor care consta in transformarea unei matrici intr-un vector. Astfel, modelul nostru va avea un input de o dimensiune fixa. Acest layer este folosit pentru a trece de la un layer de convolutie la un layer de tip fully connected.

Concluzie

Am folosit 13 layere Conv2D cu aceasta configuratie:

- Toate layerele au kernel-ul de dimensiune (3, 3), stride-ul de 1 si functia de activare relu
- 2 straturi au 64 de filtre, 2 straturi au 128 de filtre, 3 straturi au 256 de filtre, 6 straturi au 512 de filtre
- Dupa fiecare strat avem un BatchNormalization si dupa fiecare calup din cele mentionate mai sus avem un MaxPooling2D si un Dropout de 0.6

Cele 13 layere gasesc caracteristici in imagine si le extrag, cu cat mai multe cu atat avem o extragere mai buna a caracteristicilor, dar si un timp de antrenare mai mare. De asemenea, am folosit si un dropout de 0.6 pentru a evita overfitting-ul. Numarul de filtre creste de la 64 la 512 pentru ca vrem sa extragem la inceput caracteristici mai simple si mai generale, iar la final caracteristici mai complexe si mai specifice. De asemenea, am folosit si un batch normalization pentru a normaliza output-ul unui layer pentru a avea o distributie normala. Astfel, modelul va fi mai stabil si va avea o convergenta mai rapida.

3.2 Straturile fully connected

Concepte

Ce este un layer dens?

Un layer dens este un layer in care fiecare neuron este conectat cu toti neuronii ce il preced. In acest layer fiecare output al unui neuron este legat prin weight-uri de fiecare input al neuronilor din alte layere, weight-urile fiind invatate in timpul antrenarii.

Layerele de mai devreme, de convolutie si pooling ne dau low-level features si texturi. Aceste feature-uri sunt combinate si agregate pentru a forma high-level features si le invata aplicand combinatii non-lineare. Aceste high-level features sunt folosite de layerele dense pentru a face clasificarea si a ne da probabilitatea ca o imagine sa apartina unei anumite clase.

Avem un layer dens final cu 96 de neuroni si functia de activare softmax pentru a ne da probabilitatea ca o imagine sa apartina unei anumite clase.

Functii de activare folosite

Functia de activare decide daca un neuron se va activa sau nu in functie de inputul primit.

Functia de activare softmax

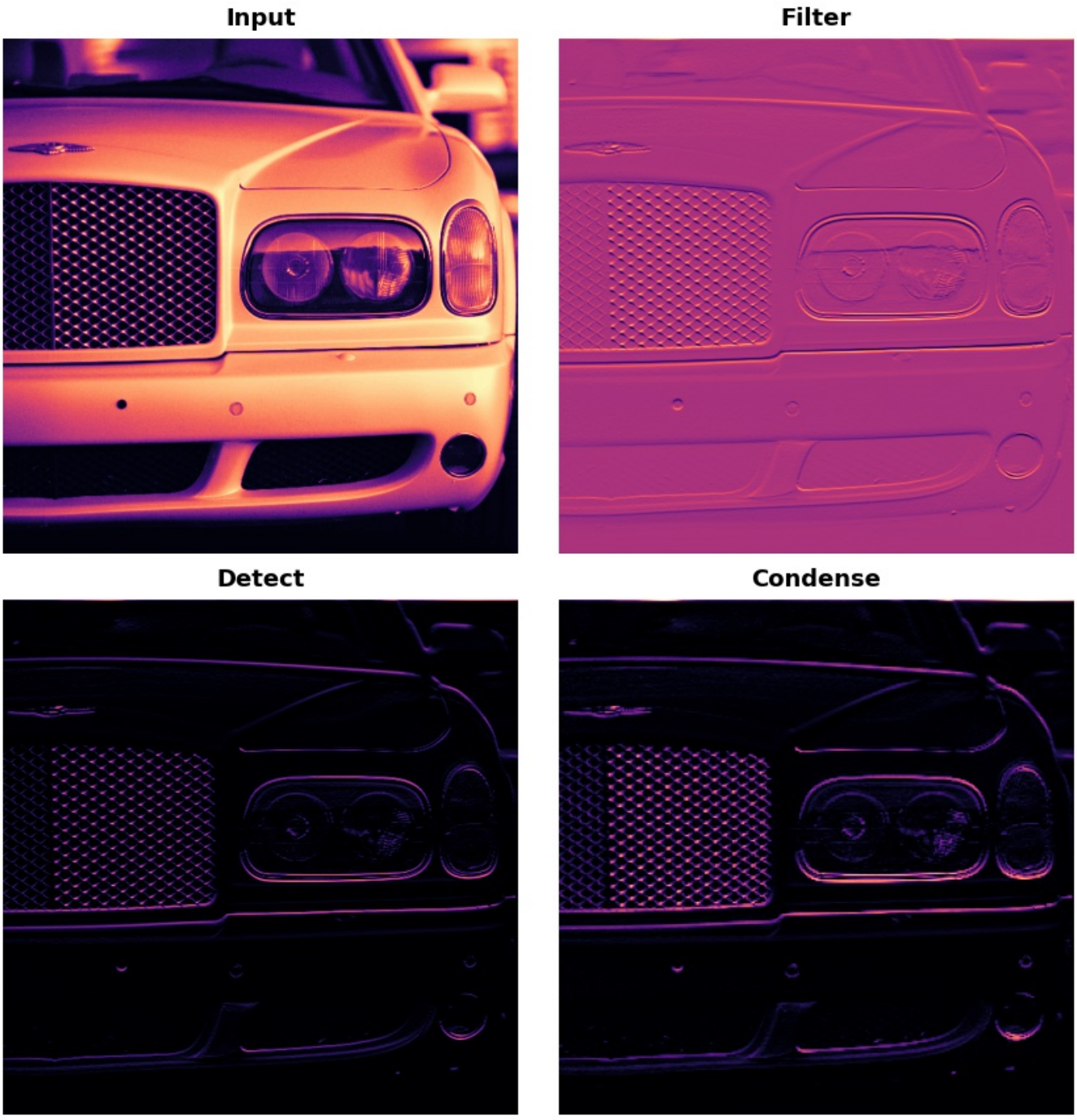
Functia softmax este o functie de activare ce ne da probabilitatea ca o imagine sa apartina unei anumite clase. Aceasta functie este folosita in layer-ul final al modelului nostru. Am ales sa folosesc aceasta functie pentru ca avem un task de clasificare cu mai multe clase.

Functia ReLU (Rectified Linear Unit)

In general, imaginile din realitate nu sunt lineare, adica intr-o imagine, caracteristicile unor pixeli sau regiuni nu depind liniar una de cealalta. Daca am o imagine cu un caine si dublez numarul de pixeli nu inseamna ca voi avea un caine de doua ori mai mare. Exista multe aspecte ale imaginilor care prezinta non-linearitate, precum variatiile de lumina si contrast, tranzitiile intre culori, texturi complexe, umbre, etc. De asemenea, intre pixelii adiacenti intr-o imagine (de exemplu tranzitia de la o zona luminata la umbra) poate fi neuniforma si pot exista relatii complexe intre valorile acestor pixeli. Transformarile liniare precum scalarea, rotatia sau translatia nu pot cuprinde caracteristicile complexe ale unei imagini, deci pentru a reusi sa facem acest lucru, avem nevoie de filtre non-lineare sau functii de activare non-lineare. In ML una dintre cele mai populare astfel de functii este ReLU. Aceasta functie este folosita pentru a introduce non-linearitate in modelul nostru si este esentiala pentru a invata relatiile complexe si nonlineare din cadrul unei imagini. Functia ReLu este buna pentru ca ne lasa doar numerele pozitive.

In concluzie, un comportament non-linear ne ajuta sa invatam caracteristici complexe.

[



In aceasta imagine preluata din articolul intitulat "Convolution and ReLU" (https://www.kaggle.com/code/ryanholbrook/convolution-and-relu) putem observa cum peste imaginea de input aplicam un filtru de convolutie, apoi ReLU pentru detectie. Vedem cum filtrul de convolutie ne da o imagine cu caracteristici importante, dar inca destul de complexa si cu tranzitii neuniforme, iar dupa ce aplicam ReLU, obtinem o imagine cu caracteristici mult mai clare si mai usor de detectat.

Concluzie

Am folosit 4 layere dense cu aceasta configuratie:

- 2 layere cu 4069 de neuroni si functia de activare relu
- 1 layer cu 512 neuroni si functia de activare relu
- BatchNormalization si Dropout de 0.7 dupa aceste 3 layere

- 1 layer cu 96 de neuroni si functia de activare softmax pentru output

Layerele au un numar de neuroni ce scade treptat pentru ca un numar mai mare de neuroni o sa invete si clasifice caracteristici mai generale, iar daca avem aceste numere converg spre numarul nostru de clase, este mai eficient in identificare si clasificarea corecta a imaginilor. Am mai adaugat si aici un dropout pentru a evita overfiting-ul. Este mai mare decat cel dintre layerele de convolutie pentru ca am observat o mica imbunatatire a acuratetii. Ultimul layer cu 96 de neuroni si functia de activare softmax este layerul final care ne da probabilitatea ca o imagine sa apartina unei anumite clase.

Cum am construit modelul? Care e evolutia lui?

Acum ca am inteles ce este un model de deep learning si cum functioneaza, am inceput sa construiesc modelul.

Initial am folosit 3 layere de convolutie, primele doua cu 32 de filtre si inca unul cu 64 si un layer dens care dadea direct outputul, dar acesta a obtinut o acuratete prea mica.

```
model.add(Conv2D(filters=16, kernel_size=(3, 3), strides = 1, activation='relu',
input_shape=(32, 32, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D())

model.add(Conv2D(filters=32, kernel_size=(3, 3), strides = 1, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())

model.add(Conv2D(filters=64, kernel_size=(3, 3), strides = 1, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(3000, activation='relu'))
model.add(Dense(1000, activation='relu'))
model.add(Dense(96, activation='softmax'))

# compile model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=
['accuracy'])
```

Acest model avea o acuratete de 50%.

Am observat ca obtin rezultate mai bune daca pornesc de la un numar mic de filtre si merg spre mare la straturile de convolutie, si invers la cele dense.

De asemenea am observat o imbunatatire a performantei daca adaug si un layer de max pooling dupa fiecare layer de convolutie, si un dropout. Dropout-ul a fost adaugat pentru a evita overfitting-ul, pentru ca obtineam acuratete de 0.98 pe setul de antrenare si 0.6 pe setul de testare, in kaggle, mai ales ca ajungea la acuratetea maxima in doar 10 epoci.

Testand, am observat ca obtin rezultate decente pe dropout de 0.3, 0.1 neavand efect, iar 0.5 scazand acuratetea, iar numeroase articole mi-au dat de inteles ca 0.3 este o valoare buna pentru dropout.

Totodata, am marit si stride-ul la 2 pentru a reduce dimensiunea imaginii mai repede si a reduce numarul de parametrii si timpul de antrenare.

```
model = Sequential()

model.add(Conv2D(filters=64, kernel_size=(3, 3), strides = 1, activation='relu',
input_shape=(64, 64, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Conv2D(filters=128, kernel_size=(3, 3), strides = 1, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(BatchNormalization())
model.add(Dropout(0.3))

# augumentation layer
model.add(RandomRotation(3))

model.add(Conv2D(filters=128, kernel_size=(3, 3), strides = 1, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Conv2D(filters=256, kernel_size=(2, 2), strides = 1, activation='relu',
padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(RandomFlip('vertical'))

model.add(Conv2D(filters=256, kernel_size=(2, 2), strides = 1, activation='relu',
padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Conv2D(filters=256, kernel_size=(1, 1), strides = (2, 2),
```

```
activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(1, 1), strides=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Conv2D(filters=512, kernel_size=(1, 1), strides = (2, 2),
activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(GlobalAveragePooling2D())
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Flatten())


model.add(Dense(3000, activation='relu'))
model.add(Dense(1000, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Dense(96, activation='softmax'))
```

Se poate observa ca am si destul de multe layere dense, asta este pentru ca am observat ca obtin rezultate mai bune daca am mai multe layere dense, care pornesc de la un numar mare de neuroni si merg spre un numar mic. Intre cele doua tipuri de layere am pus si un flatten pentru a trece de la un layer de convolutie la un layer de tip fully connected.

Pentru a varia si mai mult datele de antrenament, am adaugat si un layer de augumentation care aplica o rotatie de 3 grade pe imaginea de intrare si un layer de flip care aplica un flip vertical pe imaginea de intrare.

Am folosit si batch normalization pentru a normaliza output-ul unui layer pentru a avea o distributie normala. Astfel, modelul va fi mai stabil si va avea o convergenta mai rapida, adica va invata mai repede.

Am incercat sa folosesc si alte functii de activare precum LeakyRelu, dar chiar scadea acuratetea, asa ca am ramas la relu.

Acest model a avut o acuratete de 0.8.

Modelul final

Dupa ce am vazut ca mai multe layere de convolutie dau un model mai bun, chiar daca creste timpul de antrenare, am facut putin research si mi-am adus aminte ca in curs ne-a fost prezentat modelul VGG16 care are 16 layere de convolutie, de unde si numele, acesta fiind prezentat ca unul dintre cele mai bune si clasice modele dintre cele care sunt folosite pentru multiclass image classification. Am vazut ca seamana destul de mult cu ce am incercat eu sa fac (mai sus se afla modelul incercat inainte de VGG-16) asa ca am decis sa incerc aceasta "reteta" pentru a vedea daca obtin rezultate mai bune.

```
################## 3. Build Model #################

model = Sequential()

# ARHITECTURA VGG16 CARE ESTE LUATA DIN CURSUL DE ML

model.add(Conv2D(filters=64, kernel_size=(3, 3), strides = 1, activation='relu',
input_shape=(64, 64, 3)))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2 ), strides=(2, 2)))
model.add(Dropout(0.6))

model.add(RandomFlip('horizontal_and_vertical'))

model.add(Conv2D(filters=128, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2 ), strides=(2, 2)))
model.add(Dropout(0.6))

model.add(Conv2D(filters=256, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(filters=256, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(filters=256, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2 ), strides=(2, 2)))
model.add(Dropout(0.6))

model.add(Conv2D(filters=512, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(filters=512, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(filters=512, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2 ), strides=(2, 2)))
model.add(Dropout(0.6))

model.add(Conv2D(filters=512, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
```

```
model.add(Conv2D(filters=512, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(filters=512, kernel_size=(3, 3), strides = 1, activation='relu',
padding="same"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2 ), strides=(2, 2)))
model.add(Dropout(0.6))

# augumentation layer
model.add(RandomRotation(3))

model.add(Flatten())

model.add(Dense(4069, activation='relu'))
model.add(Dense(4069, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.7))
model.add(Dense(96, activation='softmax'))
# compile model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=
['accuracy'])

# print model summary
model.summary()
```

Am folosit adam ca optimizer pentru ca este unul dintre cei mai buni optimizatori, iar loss-ul este sparse_categorical_crossentropy pentru ca avem un task de clasificare cu mai multe clase. De asemenea am marit drop-ul la 0.6-0.7 pentru a avea un overfit mai mic.

4. Antrenarea modelului

Learning rate

Ce este learning rate?

Learning rate-ul este un hyperparametru ce ne spune cat de mult vrem sa invete modelul nostru. Daca learning rate-ul este prea mic, modelul va invata foarte incet, iar daca este prea mare, modelul va invata foarte repede si va sari peste minimul global. Daca avem un learning rate prea mare, modelul va face overfit, iar un learning rate prea mic va invata foarte incet si ne trebuie multe epoci pentru a converge.

Cum am ales learning rate-ul?

Am incercat mai multe tipuri de functii de learning rate, in aceasta ordine:

- exponential decay (acuratete buna, 0.8, dar invata foarte mult la inceput si deci facea overfit)
- warmup + exponential decay (acuratete si mai buna, 0.86-0.87; la sfarsit am reusit sa obtin o acuratete de 0.9 pe Kaggle pe setul de testare, dar nu am reusit sa o mai reproduc, probabil a fost o predictie buna pe datele de test si ar fi dat prost pe celalalt 70% din datele de test)
- ReduceLROnPlateau (constant 0.89 pe validare si 0.888 pe Kaggle) -> final

De asemenea am aplicat tot ca si callback functiei de fit, un checkpoint pentru a salva modelul cu cea mai buna acuratete pe setul de validare.

Ce este un checkpoint?

Un checkpoint este o functie de callback care salveaza modelul cu cea mai buna acuratete pe setul de validare. Astfel, daca modelul incepe sa faca overfit, putem sa ne intoarcem la modelul cu cea mai buna acuratete pe setul de validare.

Early stopping

Ce este early stopping?

Early stopping este o functie de callback care opreste antrenarea modelului daca acuratetea pe setul de validare nu creste de un anumit numar de epoci. Astfel, putem sa evitam overfitting-ul.

Nu am folosit early stopping pentru ca am observat ca acuratetea pe setul de validare creste constant, iar daca o opresc, nu mai ajunge la acuratetea maxima. Mai mult, avem checkpoint, iar daca dupa ce termin de antrenat observ ca face overfit, pot oricand sa ma intorc la modelul salvat.

Functia de loss

Ce este functia de loss?

Functia de loss este o functie care ne spune cat de bine se comporta modelul nostru. Cu cat functia de loss este mai mica, cu atat modelul nostru are o eroare la prezicere mai mica.

Am folosit sparse_categorical_crossentropy pentru ca avem un task de clasificare cu mai multe clase si este dovedibil mai eficienta decat squared error sau cea euclidiana.

Concluzie

```
from keras.callbacks import ReduceLROnPlateau

lr_scheduler = ReduceLROnPlateau(monitor='val_accuracy', factor=0.8, patience=22,
verbose=1, min_lr=0.0001, mode='auto', initial_lr=0.001)


# callbacks

from keras.callbacks import ModelCheckpoint

filepath="weights.best.hdf5"

checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,
save_best_only=True, mode='max', save_weights_only=False,
restore_best_weights=True)

callbacks_list = [checkpoint, lr_scheduler]
```

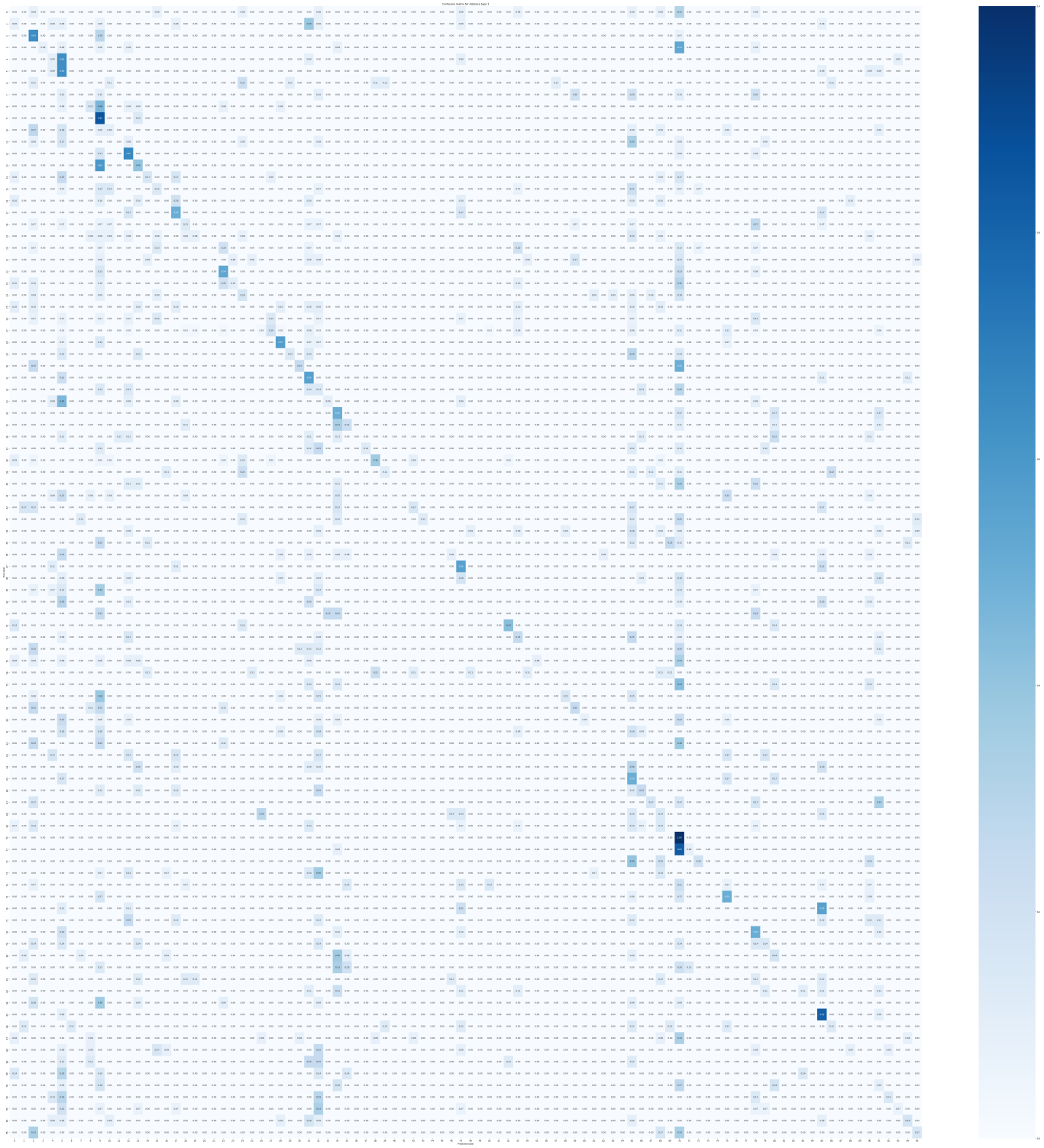
```
# train model

history = model.fit(train_dataset, epochs=250, batch_size=32,
validation_data=val_dataset, callbacks=callbacks_list)
```

Am folosit ReduceLROnPlateau cu monitorul de val_accuracy pentru a scadea learning rate-ul cand nu mai crestea acuratetea pe datele de validare. Am ales factorul de 0.8 pentru ca am observat ca acuratetea pe validare crestea constant, dar nu foarte mult, iar daca scadeam prea mult learning rate-ul, nu mai ajungea la acuratetea maxima. Am ales patience de 22 din acelasi motiv, am vrut sa scad "incet dar sigur" acuratetea. Am pornit de la 0.001 pentru ca aceasta este si acuratetea default, iar acuratetea minima este 0.0001 pentru ca daca ar fi mai mica, modelul ar invata mult prea putin pentru cele 250 de epoci cu care l-am testat.

Numarul de epoci este 250 pentru ca am observat ca acuratetea pe setul de validare creste constant pana la 250 de epoci, iar dupa acest numar, nu mai crestea deloc.

Batch size-ul este 32 pentru ca am observat ca daca il maresc, acuratetea pe setul de validare scadea, iar daca il micsoram, timpul de antrenare crestea foarte mult. Batch size-ul este numarul de imagini pe care le antrenam in acelasi timp.

Matricea de confuzie



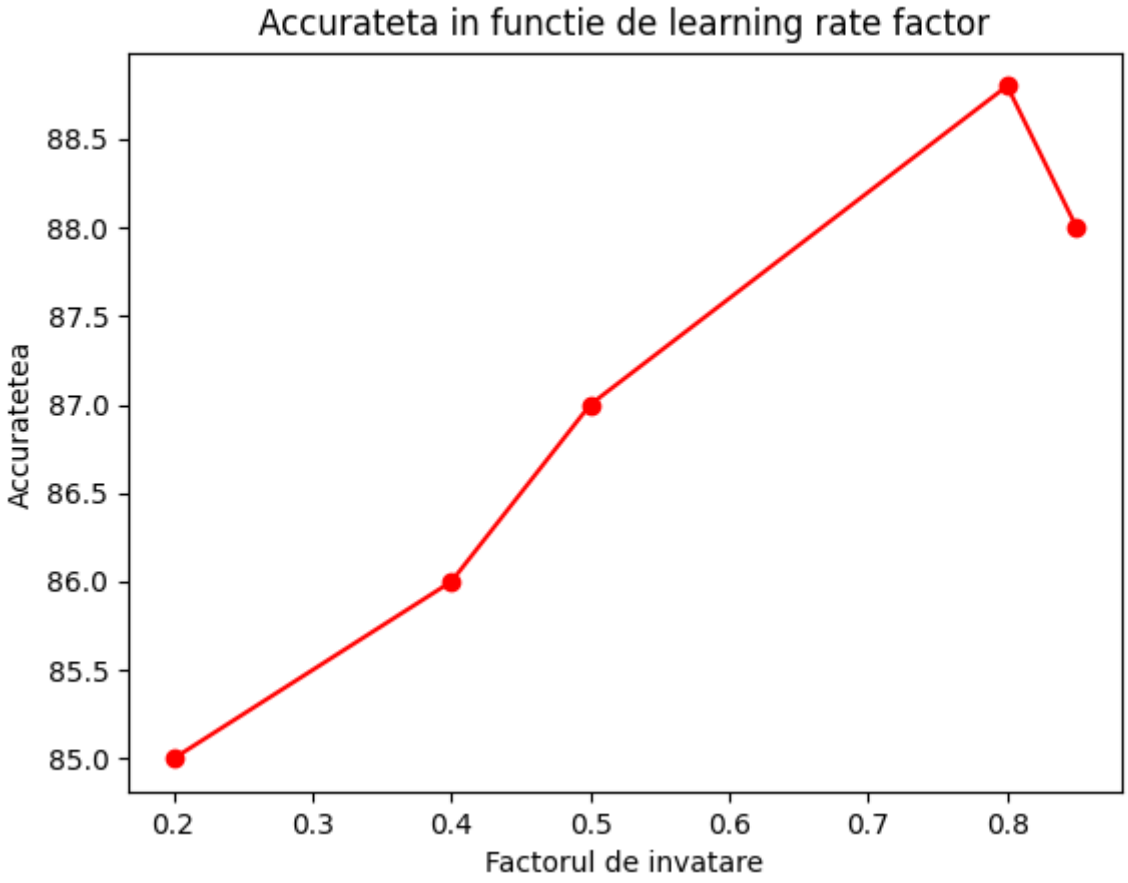
5. Testarea modelului

Model	Nr de layere conv	Nr de layere dense	Acuratete pe Kaggle
CNN (fara dropout)	3 (32, 32, 32)	1 (96)	0.42
CNN (fara dropout)	3 (16, 32, 64)	3 (3000, 1000, 96)	0.5

Model	Nr de layere conv	Nr de layere dense	Acuratete pe Kaggle
CNN	3 (16, 32, 64)	3 (3000, 1000, 96)	0.55
CNN	5 (64, 128, 128, 256)	3 (3000, 1000, 96)	0.67
CNN + Data augumentation + Scaderea dimensiunii kernel-ului	7 (64, 128, 128, 256, 256, 256, 512)	5 (3000, 1000, 512, 512, 96)	0.84
VGG + Data augumentation + learning rate (exponential)	13 (64, 64, 128, 128, 256, 256, 256, 512, 512, 512, 512, 512, 512)	5 (4096, 4096, 512, 96)	0.88
VGG + Data augumentation + learning rate (ReduceLrOnPlateau) + Dropout (0.6-0.7)	13 (64, 64, 128, 128, 256, 256, 256, 512, 512, 512, 512, 512, 512)	5 (4096, 4096, 512, 96)	0.888-0.89

-> Am incercat si cu GlobalAveragePooling2D, dar nu am sesizat nicio imbunatatire, ba chiar durata de antrenare a crescut, asa ca am renuntat la el

- Nu am facut foarte multe incercari pentru care sa fac vreun tabel, dar pentru acuratetea in functie de learning rate factor, care este unul dintre cei mai importanti hiperparametrii, am notat pe hartie cateva valori si am ales cea mai buna.



Optimizari pentru rulare

1. Rulare pe GPU

Am observat pe kaggle ca am optiunea de a activa un acelerator care imi punea la dispozitie 2 gpu-uri. Am ales sa folosesc unul dintre ele pentru a reduce timpul de antrenare. Cu "tf.config.experimental" am setat memomory growth la true. Acest lucru inseamna ca memoria va creste pe masura ce este nevoie, ceea ce este mai eficient decat alocarea intregii memorii de la inceput.

Apoi am incarcat imaginile pe GPU cu AUTOTUNE si prefetch functii din tensorflow.

```
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_images_tensor = tf.convert_to_tensor(train_images)
train_labels_tensor = tf.convert_to_tensor(train_labels)

val_images_tensor = tf.convert_to_tensor(val_images)
val_labels_tensor = tf.convert_to_tensor(val_labels)

train_dataset = tf.data.Dataset.from_tensor_slices((train_images,
train_labels)).batch(32).cache().prefetch(AUTOTUNE)
val_dataset = tf.data.Dataset.from_tensor_slices((val_images,
val_labels)).batch(32).cache().prefetch(AUTOTUNE)
```

Dupa cum se observa pentru a putea folosi functiile a trebuit sa transform imaginile si etichetele in dataset-uri de tip tensor.

KNN - 0.16

Ce este knn?

KNN este un algoritm de clasificare ce se bazeaza pe gasirea celor mai apropiati vecini ai unui punct si clasificarea lui in functie de vecinii lui.

Am incercat mai intai acest model, chiar daca stiam ca nu voi obtine o acuratete foarte decenta, pentru a ma familiariza cu constructia unui algoritm de clasificare si pentru a ma familiariza cu libraria numpy.

Acest model este unul dintre modelele care presupune ca exista o legatura intre date, dar nu este un model de deep learning. In esenta, pentru a prezice tipul unei date de intrare, el "se uita" la celelate date si alege tipul care "seamana" cel mai mult.

Cum functioneaza knn?

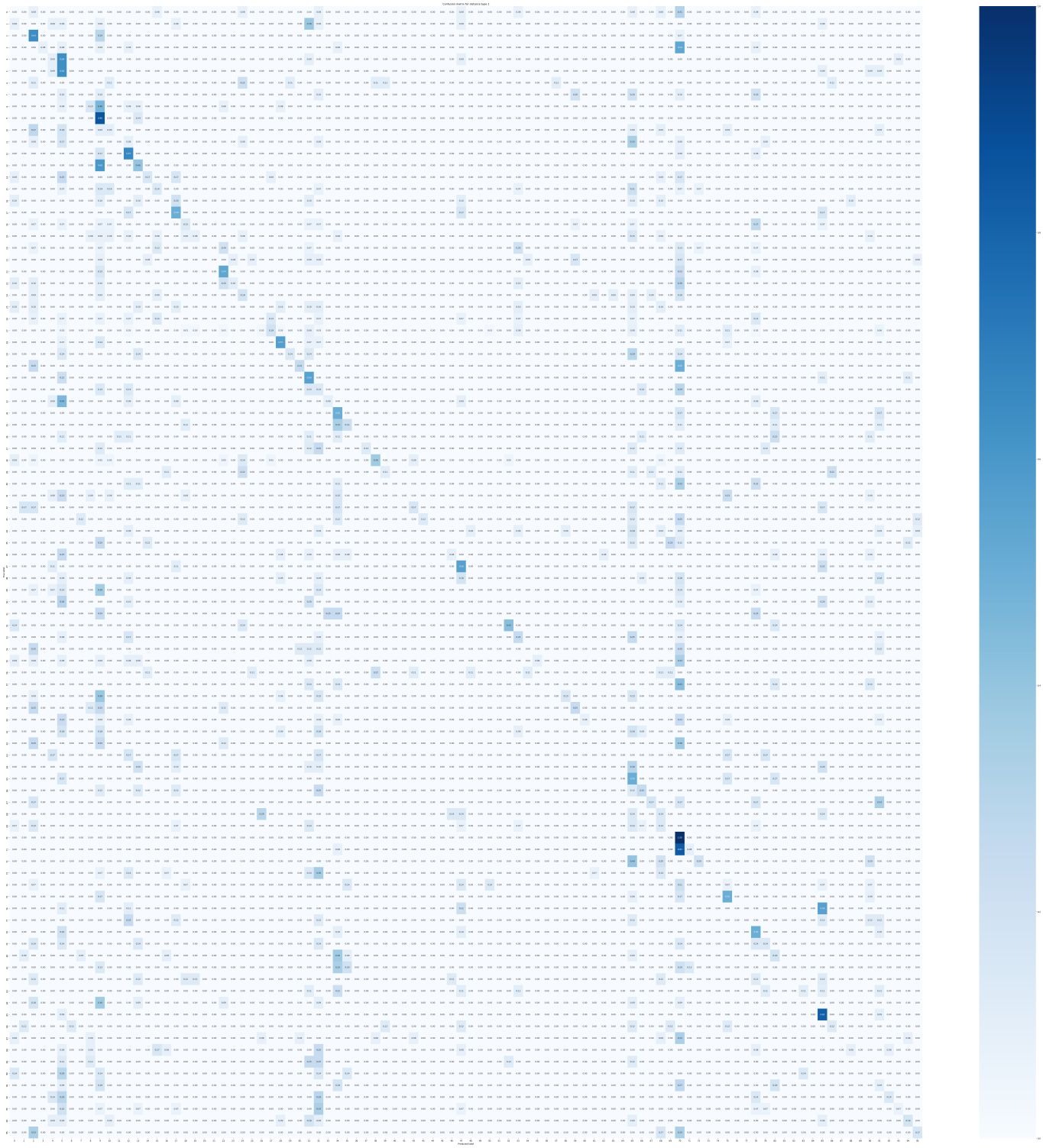
KNN gaseste datele care seamana cel mai mult, astfel: - Consideram ca fiecare imagine este pusa pe un grafic, unde OX este imaginea si OY este eticheta ei - Cand avem o imagine noua, o punem si pe ea pe grafic si calculam distanta dintre ea si celelalte imagini - Sortam imaginile dupa distanta si luam primele k imagini - Putem alege diferite distante, cele mai populare fiind distanta euclidiana, manhattan si cosinus

Cum am implementat knn?

Pasii sunt asemanatori cu cei de la CNN: - Am citit datele din csv - Am preprocesat datele; Aici trebuie mentionat ca imaginile au fost transformate intr-un array 2D pentru ca altfel nu puteam calcula distanta dintre ele - Am construit modelul - Am prezis etichetele imaginilor de testare

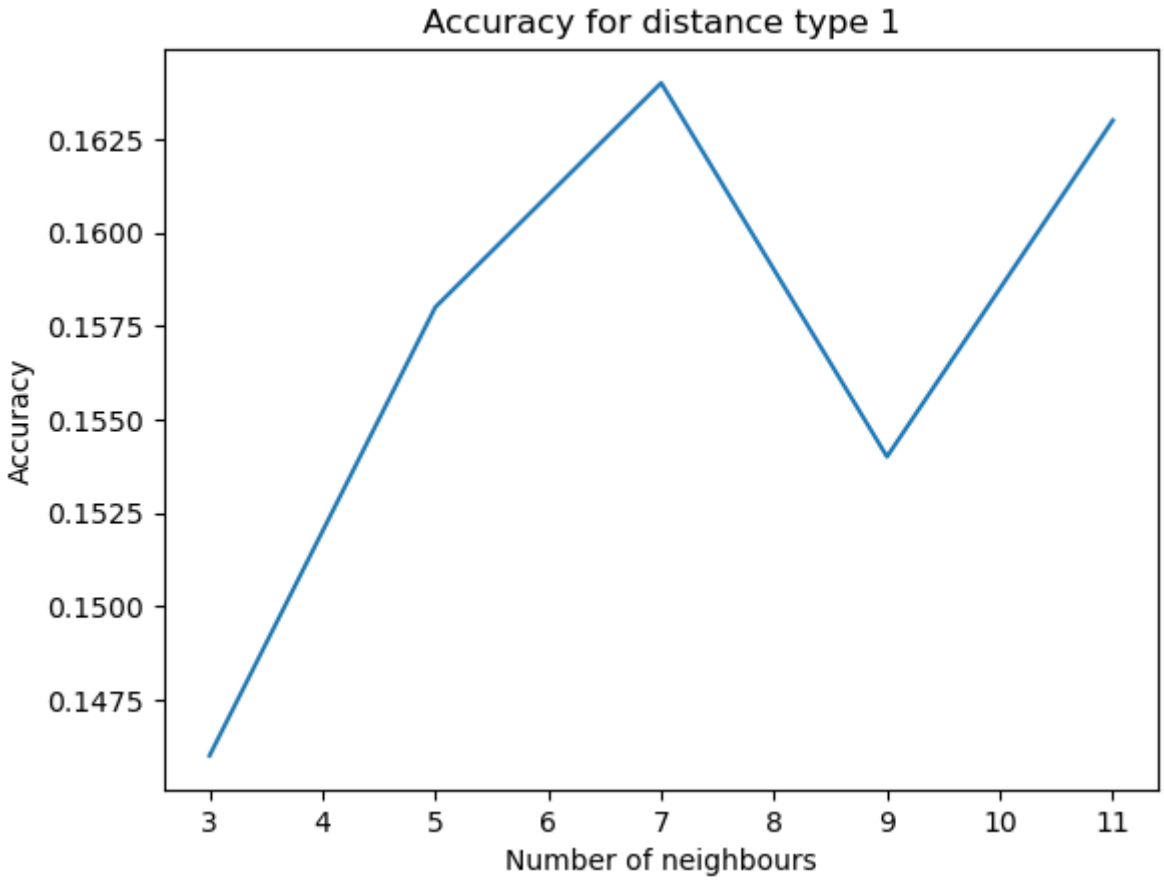
Modelul implementeaza 3 tipuri de distante, euclidiana si cosinus. Numarul de vecini a fost testat pe 3, 5, 6, 9 si 11.

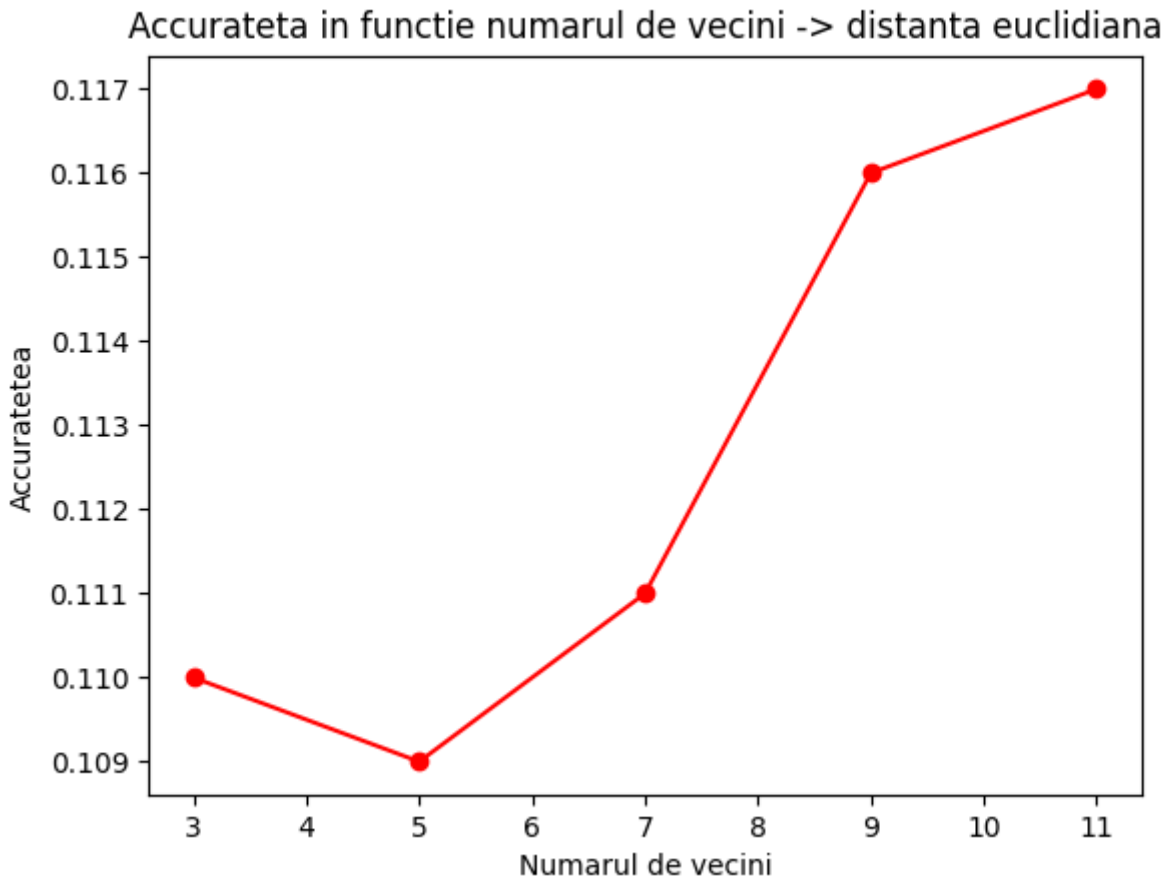
Matricea de confuzie pentru distanta manhattan si 5 vecini



5. Testarea modelului

Am testat pe distanta Euclidiana si Manhattan, cu 3, 5, 6, 9 si 11 vecini. Am obtinut o acuratete de 0.16.





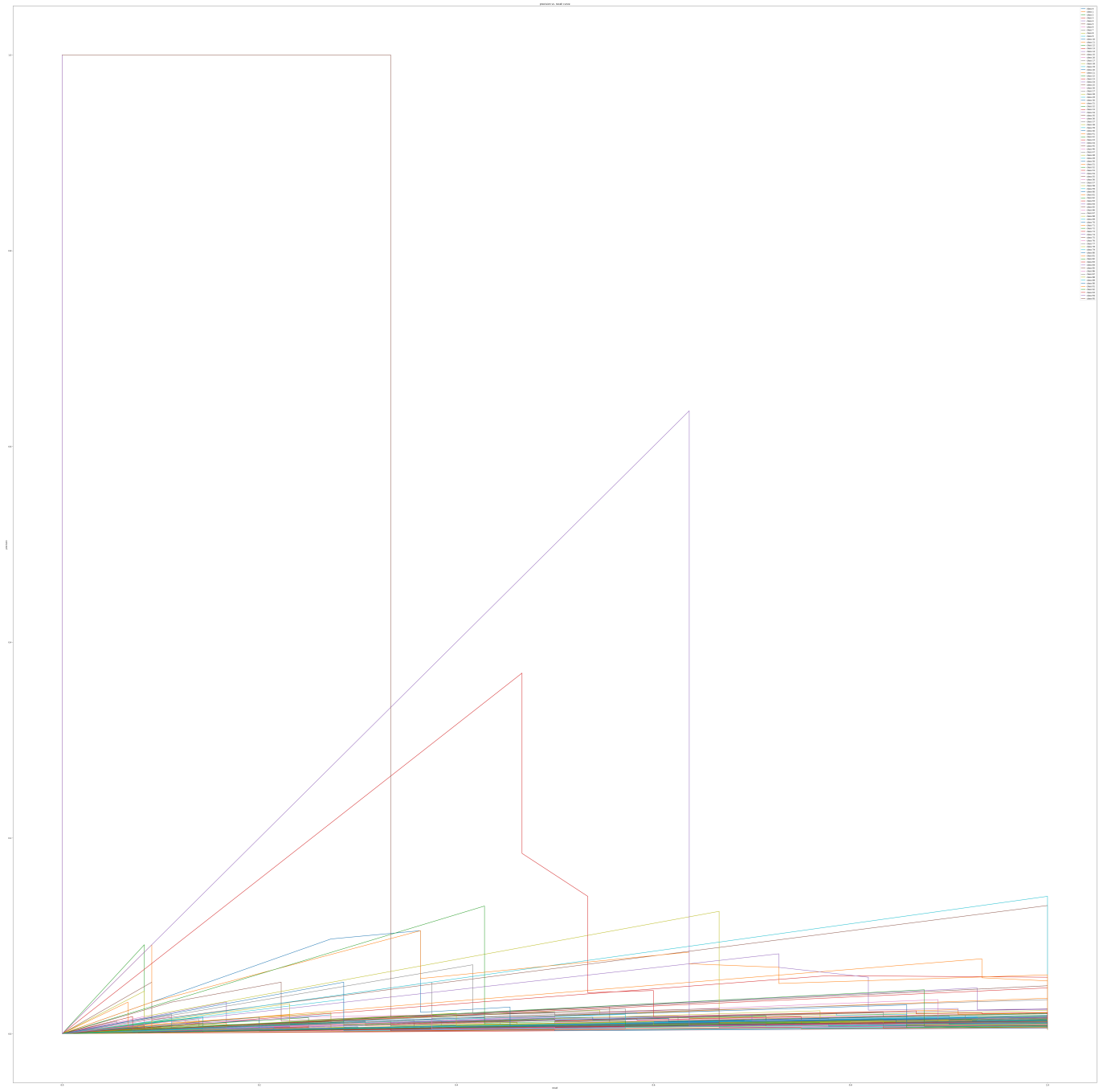
Precizie, recall si acuratete pt ficare clasa

Pentru CNN - VGG16 pe 30 de epoci:

	precision	recall	f1-score	support
0	0.86	0.46	0.60	13
1	0.21	0.54	0.30	13
2	1.00	0.14	0.25	14
3	0.80	0.92	0.86	13
4	0.67	0.36	0.47	11
5	0.60	0.55	0.57	11
6	0.33	0.22	0.27	9
7	0.33	0.70	0.45	10
8	0.73	0.62	0.67	13
9	1.00	0.71	0.83	7
10	0.40	0.73	0.52	11
11	1.00	0.17	0.29	12
12	0.89	0.67	0.76	12
13	0.67	0.80	0.73	5
14	0.10	0.08	0.09	12
15	0.33	0.07	0.12	14
16	1.00	0.60	0.75	10
17	0.86	1.00	0.92	6
18	1.00	0.73	0.85	15
19	0.89	0.62	0.73	13
20	1.00	0.13	0.24	15
21	0.90	0.75	0.82	12
22	0.65	1.00	0.79	13
23	0.43	0.90	0.58	10
24	1.00	0.18	0.31	11
25	0.50	0.40	0.44	10
26	0.33	0.07	0.12	14
27	0.69	0.86	0.77	36
28	0.93	0.93	0.93	14
29	0.20	0.29	0.24	7
30	1.00	0.50	0.67	4
31	0.44	0.44	0.44	9
32	0.00	0.00	0.00	7
33	0.70	0.64	0.67	11
34	0.00	0.00	0.00	6
35	0.44	0.44	0.44	9
36	0.43	0.33	0.38	9
37	0.67	0.25	0.36	8
38	0.29	0.09	0.14	22
39	0.00	0.00	0.00	9
40	0.38	0.33	0.35	9
41	0.53	0.77	0.62	13
42	0.00	0.00	0.00	6
43	0.71	0.62	0.67	8
44	0.55	1.00	0.71	11
45	0.50	0.25	0.33	8
46	0.41	0.92	0.56	12
47	0.78	0.78	0.78	9

```
           48       0.83      0.45      0.59        11
           49       0.31      0.29      0.30        14
           50       0.56      0.50      0.53        10
           51       0.43      0.75      0.55         4
           52       0.22      0.29      0.25         7
           53       0.33      0.08      0.13        12
           54       0.50      1.00      0.67         8
           55       1.00      0.17      0.29        12
           56       0.42      0.56      0.48         9
           57       0.20      0.43      0.27         7
           58       0.50      0.46      0.48        13
           59       1.00      0.25      0.40         8
           60       0.29      0.15      0.20        13
           61       0.50      0.82      0.62        11
           62       0.57      1.00      0.73         8
           63       0.22      0.33      0.27         6
           64       0.78      0.70      0.74        10
           65       0.83      0.83      0.83         6
           66       0.67      0.25      0.36         8
           67       0.33      0.33      0.33         6
           68       1.00      0.14      0.25         7
           69       0.00      0.00      0.00        14
           70       0.88      1.00      0.93         7
           71       0.62      0.83      0.71        12
           72       0.31      1.00      0.48         5
           73       0.62      0.93      0.74        14
           74       0.36      0.86      0.51        14
           75       0.73      0.92      0.81        12
           76       0.31      0.89      0.46         9
           77       0.31      0.50      0.38         8
           78       0.75      0.30      0.43        10
           79       0.67      0.29      0.40         7
           80       1.00      0.09      0.17        11
           81       0.48      0.87      0.62        15
           82       0.18      0.88      0.30         8
           83       0.35      0.78      0.48         9
           84       0.57      0.73      0.64        11
           85       0.77      0.91      0.83        11
           86       0.00      0.00      0.00         8
           87       0.71      0.42      0.53        12
           88       1.00      0.58      0.74        12
           89       0.42      1.00      0.59         8
           90       0.29      0.29      0.29         7
           91       0.12      0.09      0.11        11
           92       0.75      0.43      0.55         7
           93       0.88      0.47      0.61        15
           94       0.78      0.64      0.70        11
           95       1.00      0.33      0.50         6

    accuracy                           0.52      1000
   macro avg       0.57      0.51      0.48      1000
weighted avg       0.58      0.52      0.49      1000
```

Precizia si recall-ul:



Pentru KNN:

Clasa	Precizie	Recall	F1-score	Support
0	0.00	0.00	0.00	13
1	0.00	0.00	0.00	13
2	0.21	0.64	0.32	14
3	1.00	0.08	0.14	13
4	0.10	0.09	0.10	11
5	0.09	0.64	0.16	11

Clasa	Precizie	Recall	F1-score	Support
6	0.00	0.00	0.00	9
7	0.00	0.00	0.00	10
8	0.25	0.15	0.19	13
9	0.08	0.86	0.14	7
10	0.11	0.09	0.10	11
11	0.00	0.00	0.00	12
12	0.25	0.67	0.36	12
13	0.12	0.40	0.19	5
14	0.40	0.17	0.24	12
15	0.18	0.14	0.16	14
16	0.00	0.00	0.00	10
17	0.18	0.50	0.26	6
18	0.25	0.13	0.17	15
19	0.33	0.08	0.12	13
20	0.00	0.00	0.00	15
21	0.00	0.00	0.00	12
22	0.39	0.54	0.45	13
23	0.50	0.10	0.17	10
24	0.14	0.18	0.16	11
25	0.00	0.00	0.00	10
26	0.00	0.00	0.00	14
27	0.64	0.19	0.30	36
28	0.50	0.57	0.53	14
29	0.50	0.14	0.22	7
30	0.33	0.25	0.29	4
31	0.12	0.56	0.20	9
32	0.02	0.14	0.03	7
33	0.50	0.09	0.15	11
34	0.09	0.50	0.15	6
35	0.20	0.22	0.21	9

Clasa	Precizie	Recall	F1-score	Support
36	0.00	0.00	0.00	9
37	1.00	0.12	0.22	8
38	0.67	0.36	0.47	22
39	0.33	0.11	0.17	9
40	0.00	0.00	0.00	9
41	0.00	0.00	0.00	13
42	0.20	0.17	0.18	6
43	1.00	0.12	0.22	8
44	0.00	0.00	0.00	11
45	0.00	0.00	0.00	8
46	0.33	0.08	0.13	12
47	0.20	0.56	0.29	9
48	0.00	0.00	0.00	11
49	0.00	0.00	0.00	14
50	0.00	0.00	0.00	10
51	0.00	0.00	0.00	4
52	0.60	0.43	0.50	7
53	0.17	0.25	0.20	12
54	0.00	0.00	0.00	8
55	1.00	0.08	0.15	12
56	0.00	0.00	0.00	9
57	0.00	0.00	0.00	7
58	0.67	0.15	0.25	13
59	0.29	0.25	0.27	8
60	1.00	0.08	0.14	13
61	0.00	0.00	0.00	11
62	0.00	0.00	0.00	8
63	0.00	0.00	0.00	6
64	0.00	0.00	0.00	10
65	0.05	0.50	0.10	6

Clasa	Precizie	Recall	F1-score	Support
66	0.29	0.25	0.27	8
67	0.33	0.17	0.22	6
68	0.05	0.14	0.08	7
69	0.00	0.00	0.00	14
70	0.06	1.00	0.12	7
71	0.33	0.08	0.13	12
72	0.33	0.20	0.25	5
73	0.00	0.00	0.00	14
74	0.00	0.00	0.00	14
75	0.30	0.50	0.37	12
76	0.00	0.00	0.00	9
77	0.00	0.00	0.00	8
78	0.16	0.50	0.24	10
79	0.17	0.14	0.15	7
80	0.17	0.18	0.17	11
81	0.00	0.00	0.00	15
82	0.00	0.00	0.00	8
83	0.50	0.11	0.18	9
84	0.00	0.00	0.00	11
85	0.27	0.82	0.41	11
86	0.25	0.12	0.17	8
87	0.00	0.00	0.00	12
88	0.50	0.08	0.14	12
89	0.00	0.00	0.00	8
90	0.00	0.00	0.00	7
91	0.05	0.09	0.07	11
92	0.00	0.00	0.00	7
93	0.00	0.00	0.00	15
94	0.00	0.00	0.00	11
95	0.00	0.00	0.00	6

Clasa	Precizie	Recall	F1-score	Support
---	---	---	---	---
accuracy			0.16	1000
macro avg	0.18	0.16	0.13	1000
weighted avg	0.20	0.16	0.14	1000

```
[0.         0.         0.64285714 0.07692308 0.09090909 0.63636364
 1.         0.         0.15384615 0.85714286 0.09090909 0.
 0.66666667 0.4        0.16666667 0.14285714 0.         0.5
 0.13333333 0.07692308 0.         0.         0.53846154 0.1
 0.18181818 0.         0.         0.19444444 0.57142857 0.14285714
 0.25       0.55555556 0.14285714 0.09090909 0.5        0.22222222
 1.         0.125      0.36363636 0.11111111 0.         0.
 0.16666667 0.125      0.         0.         0.08333333 0.55555556
 1.         0.         0.         0.         0.42857143 0.25
 2.         0.08333333 0.         0.         0.15384615 0.25
 0.07692308 0.         0.         0.         0.         0.5
 0.25       0.16666667 0.14285714 0.         1.         0.08333333
 0.2        0.         0.         0.5        0.         0.
 0.5        0.14285714 0.18181818 0.         0.         0.11111111
 1.         0.81818182 0.125      0.         0.08333333 0.
 2.         0.09090909 0.14285714 0.06666667 0.18181818 0.16666667]
[0.         0.         0.20930233 1.         0.1        0.09333333
 1.         0.         0.25       0.07792208 0.11111111 0.
 0.25       0.125      0.4        0.18181818 0.         0.17647059
 0.25       0.33333333 0.         0.         0.38888889 0.5
 0.14285714 0.         0.         0.63636364 0.5        0.5
 0.33333333 0.125      0.01960784 0.5        0.09090909 0.2
 1.         1.         0.66666667 0.33333333 0.         0.
 0.2        1.         0.         0.         0.33333333 0.2
 1.         0.         0.         0.         0.6        0.16666667
 2.         1.         0.         0.         0.66666667 0.28571429
 3.         0.         0.         0.         0.         0.05357143
 0.28571429 0.33333333 0.05263158 0.         0.06422018 0.33333333
 0.33333333 0.         0.         0.3        0.         0.
 0.16129032 0.16666667 0.16666667 0.         0.         0.5
 1.         0.27272727 0.25       0.         0.5        0.
 2.         0.05263158 0.5        0.5        0.4        0.25      ]
```

Resurse folosite

1. Cursul de ML
2. Kaggle
3. Diferite forumuri pe StackOverflow
4. medium.com
5. https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

6. https://www.researchgate.net/publication/337401161_Fundamental_Concepts_of_Convolutional_Neural_Network