Klasifikacija hrane korišćenjem konvolucionih neuronskih mreža

Djordje Vujinović 481/2017
Lazar Bojanić 415/2016

February 4, 2019

Sadržaj

1 Uvod

Klasifikacija slika predstavlja problem označavanja slike klasom iz fiksnog skupa kategorija za svaku sliku iz ulaznog skupa podataka. Ona predstavlja problem nadgledanog učenja: definisati skup ciljnih klasa (objekata koje je potrebno identifikovati na slici), i trenirati model koji će prepoznavati slike koristeći već klasifikovane podatke za trening.

Klasifikacijom slika možemo rešiti neke od problema kompjuterskog vida (eng. Computer vision) kao što su prepoznavanje objekata na slici, segmentacija slike koji se mogu primeniti u situacijama kao što su autonomna vožnja, prepoznavanje lica, pametne prodavnice kao što je Amazon Go, analiza krvnih slika, vojno nadgledanje...

Iz ugla čoveka klasifikovanje slika dolazi prirodno, jer je naš mozak podsvesno treniran određenim skupovima slika što je rezultovalo dobijanjem sposobnosti kategorisanja istih. Međutim iz ugla računara, istom problemu je potrebno prići iz malo drugačijeg ugla. Pošto su slike u računaru predstavljene pomoću niza numeričkih vrednosti, potrebno je pronaći šablone koji se javljaju u nizovima i na osnovu njih svrstavati slike u određene kategorije.

Neki od izazova klasifikacije slika mogu biti:

- Orijentacija pogleda : Položaj objekata na slikama se uglavnom razlikuje i davanje računaru istih slika sa različitim položajem će rezultovati različitim kategorisanjem – potrebno je napraviti algoritam koji prepoznaje objekte bez obzira na njihov položajem
- Razlika u veličini : Skaliranje određenog objekta ne bi trebalo da promeni kategoriju slike, iz ugla računara dve slike su reprezentovane različitim numeričkim vrednostima tako da ih on vidi kao različite iako su suštinski identične slike
- Deformacije: U realnom svetu ne postoje identični objekti, blage deformacije postoje svuda. Ako bismo zanemarili deformacije objekata, računar bi možda pomislio da neki objekat mora da bude tačno određenog oblika rezultujući pogrešnim klasifikovanjem jako sličnih objekata.
- Varijacije unutar klasa: Neke klase ne moraju da imaju striktan oblik, na primer postoje različite vrste flaša, različite vrste stolica, mobilnih telefona i potrebno je napraviti model koji je invarijantan i na ovo.
- Zaklon vida: Objekti mogu biti zaklonjeni iza nekog drugog objekta i samim tim rezultovajuće klasifikovanje bi bilo neispravno. Na primer, mesec zaklonjen senkom od Zemlje bi trebalo ispravno klasifikovati.

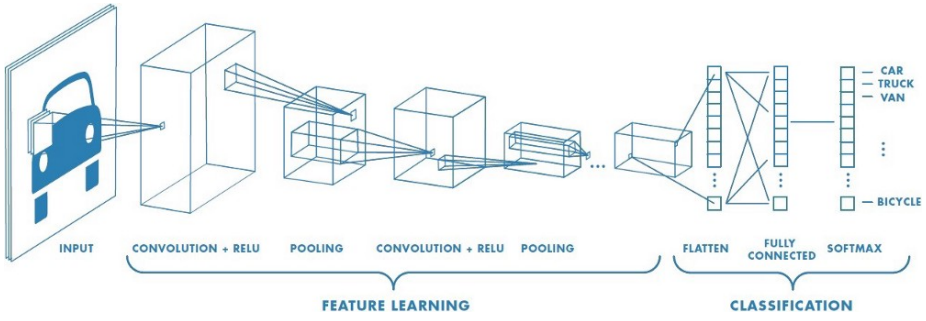
Konvolucione neuronske mreže su se pokazale kao idealan izbor metode za rešavanje ovog problema i njihova primena u rešavanju problema klasifikacije slika će biti detaljno objašnjena u nastavku.

Postoji nekoliko arhitektura u polju konvolucionih neuronskih mreža i neke od najpoznatijih su:

- LeNet – prva uspešna primena konvolucionih neuronskih mreža napravljena od strane Yann LeCunn 1990ih godina koja je korišćena za čitanje zip kodova, cifara itd.
- AlexNet je prvi rad koji je popularizovao korišćenje konvolucionih neuronskih mreža koju su napravili Alex Krizhevsky, Ilya Sutskever i Geoff Hinton. Imala je sličnu strukturu kao LeNet samo je bila dublja, veća i koristila nekoliko povezanih konvolucionih slojeva
- ZF Net
- GoogLeNet
- VGGNet
- ResNet

2 Konvoluciona neuronska mreža

Konvoluciona neuronska mreža je algoritam dubokog učenja (eng. Deep Learning) koji prima sliku kao ulazni parametar i kao izlaz ima vektor čiji elementi predstavljaju pripadnosti svake unapred definisane klase. Za problem klasifikacije slika izabrane su ispred regularnih neuronskih mreža iz razloga što regularne neuronske mreže i za slike malih dimenzija postaju jako kompleksne za izračunavanje. Naime za sliku CIFAR-10 baze podataka koje su dimenzija 32x32 piksela i RGB modelom boja vodilo bi tome da svaki neuron ima 32*32*3 = 3072 težina. Za primenljivije situacije ovakav pristup postaje neupotrebljiv, jer na primer za sliku 200x200px za svaki neuron bi bilo potrebno 120000 težina, a neurona je potrebno mnogo za konstrukciju kvalitetne neuronske mreže. Konvolucione neuronske mreže pretpostavljaju da će im ulaz biti slika i cela arhitektura neuronske mreže je adaptirana ovoj pretpostavci. One su u stanju da prepoznaju prostorne zavisnosti slike kroz primenu raznih filtera.



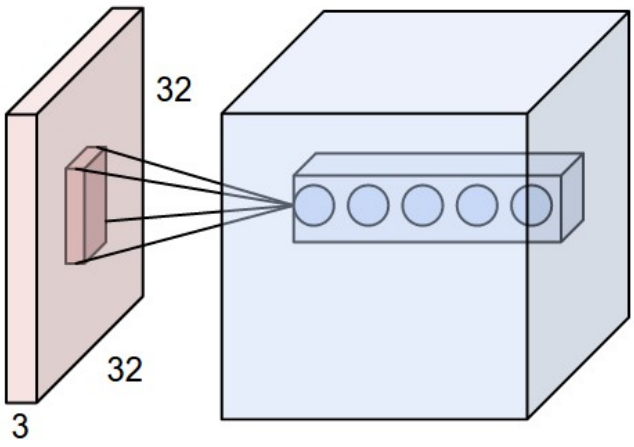
Slika 1: Tok podataka kroz konvolucionu neuronsku mrežu

Konvoluciona neuronska mreža sačinjena je od tri tipa slojeva:

- Konvolucionog sloja(eng. Convolutional layer)
- Sloja ujedinjavanja (eng. Pooling layer)
- Potpuno povezanog sloja (eng. Fully connected layer)

2.1 Konvolucioni sloj

Konvolucioni sloj je najbitniji deo KNN i on radi najviše izračunavanja u mreži. Parametri konvolucionog sloja su sačinjeni od skupa filtera koji mogu da uče. Ovi filteri su uglavnom malih dimenzija što se tiče širine i dužine, ali su iste dubine kao ulazni parametar. Tokom prvog prolaza pomeramo svaki filter po širini i visini i računamo skalarni proizvod ulaza i vrednosti filtera. Izlaz jednog filtera će biti dvodimenzioni niz. Ako imamo na primer 12 filtera, izlaz iz konvolucionog sloja će biti rezultati svakog filtera poređanih po dubini. Intuitivno, mreža će naučiti ove filtere da se aktiviraju kada vide neki prepoznatljiv deo mreže kao što je na primer ivica objekta ili određena boja ili neki prepoznatljivi obrazac. Kao što smo rekli nije praktično povezivati svaki neuron sa svim neuronima prethodnog sloja, tako da ćemo neurone povezivati sa regionima ulazne vrednosti. U slučajevima slika u RGB formatu, filter ima istu dubinu kao i ulaz, a izlazna vrednost predstavlja sumu rezultata po kanalima boja.

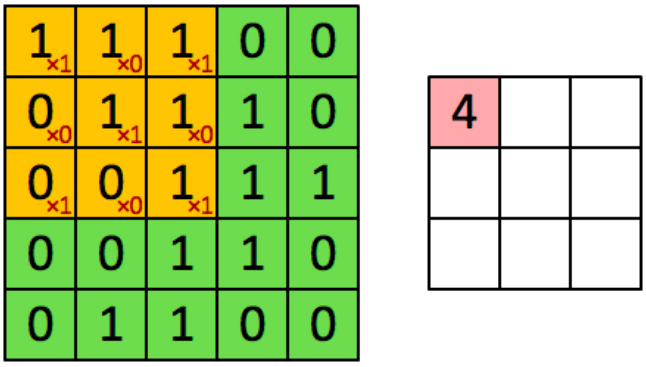


Slika 2: Tok podataka kroz konvolucionu neuronsku mrežu

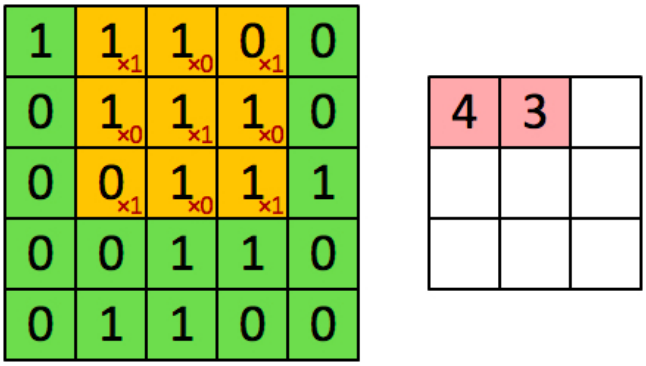
Na primer za filter

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

imali bismo sledeću konvoluciju:



Slika 3: Računanje skalarnog proizvoda filtera i prvog segmenta slike



Slika 4: Računanje skalarnog proizvoda filtera i drugog segmenta slike

Konvolucioni sloj ima nekoliko ključnih parametara na koje bi trebalo obratiti pažnju:

- Dubina izlaza nam predstavlja broj filtera koje će naš sloj sadržati. Na primer u prvom konvolucionom sloju mreže možemo staviti filtere koji prepoznaju boju i ivice.

- Dužina koraka (eng Stride) nam govori za koliko piksela ćemo pomerati filter udesno u svakom koraku (U slici iznad korak je 1).
- Padding će nam služiti prilikom definisanja dimenzije izlaza iz sloja, u nekim situacijama želimo da dimenziju izlaza smanjimo u odnosu na dimenziju ulaza, a u nekim da je povećamo ili ostavimo identičnom. U drugom slučaju popunili bismo ivice ulaza nulama.

Na svaki element iz konvolucionog sloja primenjivaće se RELU funkcija aktivacije max(0, x). Cilj konvolucionog sloja je da izvuče opšte osobine kao što su ivice. Naravno, mreža ne mora da sadrži jedan konvolucioni sloj, može ih imati nekoliko. Ako ih ima nekoliko, na primer prvi bi mogao da prepoznaje boju, ivice dok bi naredni mogli da prepoznaju neke osobine višeg nivoa koji bi davali našoj mreži sveobuhvatno znanje o slici.

2.2 Sloj ujedinjavanja

Sloj ujedinjavanja (eng. Pooling layer) je zadužen za smanjivanje dimenzija rezultata konvolucije. Cilj ovog sloja je da smanji procesirajuću snagu potrebnu za obradu podataka kroz redukciju dimenzionalnosti.

Postoje dva tipa ujedinjavanja:

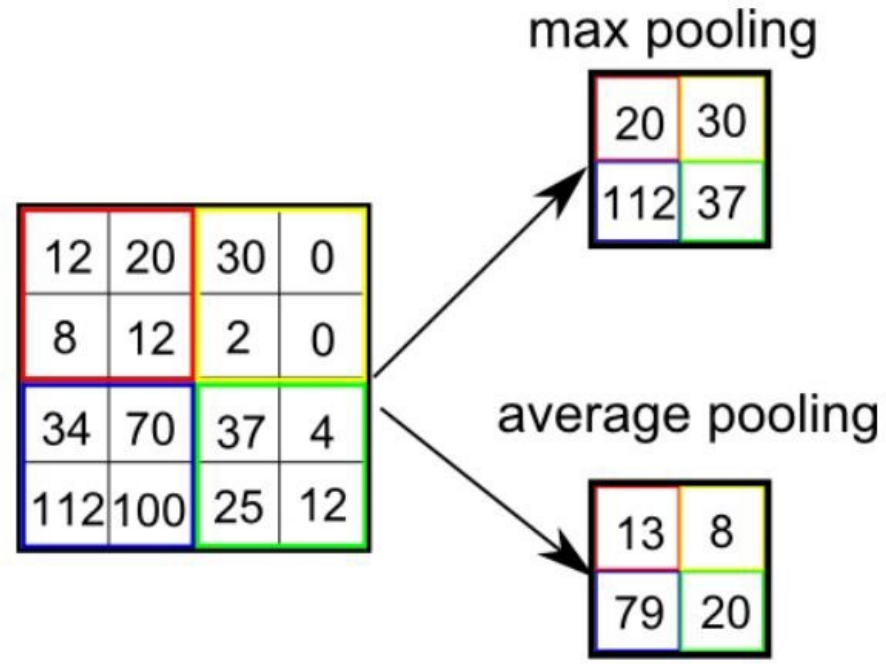
- Max pooling koje vraća maksimalnu vrednost dela slike pokrivenog jezgrom
- Average pooling koje vraća prosečnu vrednost dela slike pokrivenog jezgrom

Konvolucioni sloj zajedno sa slojem ujedinjavanja čine i-ti sloj konvolucione neuronske mreže. U zavisnosti od veličine slike i njene kompleksnosti broj ovakvih slojeva može biti povećan zarad prepoznavanja sitnih detalja, po ceni veće procesirajuće zahtevnosti.

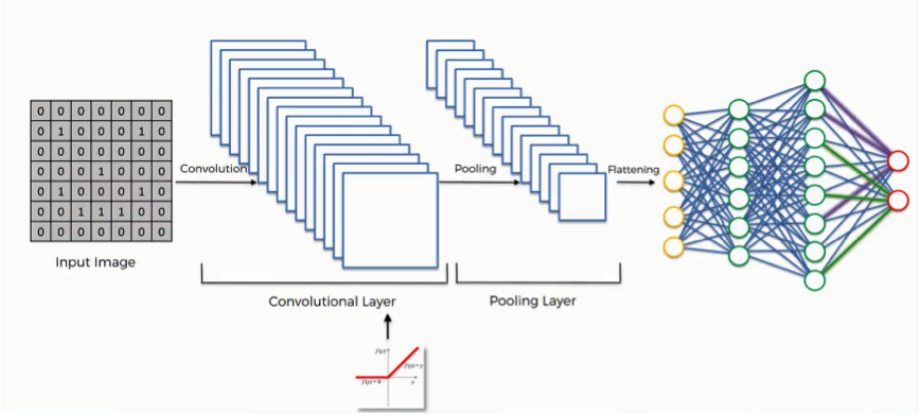
2.3 Potpuno povezani sloj

Potpuno povezani sloj (eng Fully connected layer) predstavlja poslednji sloj unutar konvolucione neuronske mreže. To je način prepoznavanja osobina koje su izlaz konvolucionog sloja. Prvi korak bi bio ‘ispravljanje‘ izlaza konvolucionog sloja u kolona vektor kako bismo. Taj vektor će predstavljati ulaz regularne neuronske mreže I backpropagation će se odvijati u svakoj iteraciji treninga.

Nakon serije epoha, ovakav model će biti u stanju da prepozna sitne detalje slike i da ih klasifikuje koristeći softmax klasifikacionu tehniku.



Slika 5: Primer smanjivanja dimenzionalnosti korišćenjem max i average pooling-a



Slika 6: Primer toka konvolucione neuronske mreže sa potpuno povezanim slojem (desno)

2.4 Funkcija aktivacije

Softmax funkcija je optimalno rešenje za funkciju aktivacije jer dobijamo da vrednosti pripadnosti svakoj klasi u zbiru daju 1. Na ovaj način možemo imati konstatacije tipa slika je 90% sushi, 5% hamburger I 5% je pica. Ako ne bismo koristili softmax imali bismo situaciju u kojoj bi ulazna slika bila 70% sushi 45% pica I 30% hamburger, što bi nam predstavljalo neupotrebljiv rezultat.

2.5 Funkcija greške

Za funkciju greške dobro rešenje je uzeti cross-entropy jer ima prednosti u odnosu na kvadratnu grešku. Naime na početku backpropagation procesa, izlazne vrednosti su obično minimalne. Gradijent je takodje nizak pa samim tim je i teško neuronskoj mreži da iskoristi te podatke u ispravljanju težina i optimizaciji. Cross-entropy zbog logaritamske prirode uspeva da iskoristi ovu situaciju i da služi kao bolje optimizaciono rešenje.

2.6 Podešavanja parametara

Preporučena podešavanja parametara prilikom pravljenja konvolucione neuronske mreže:

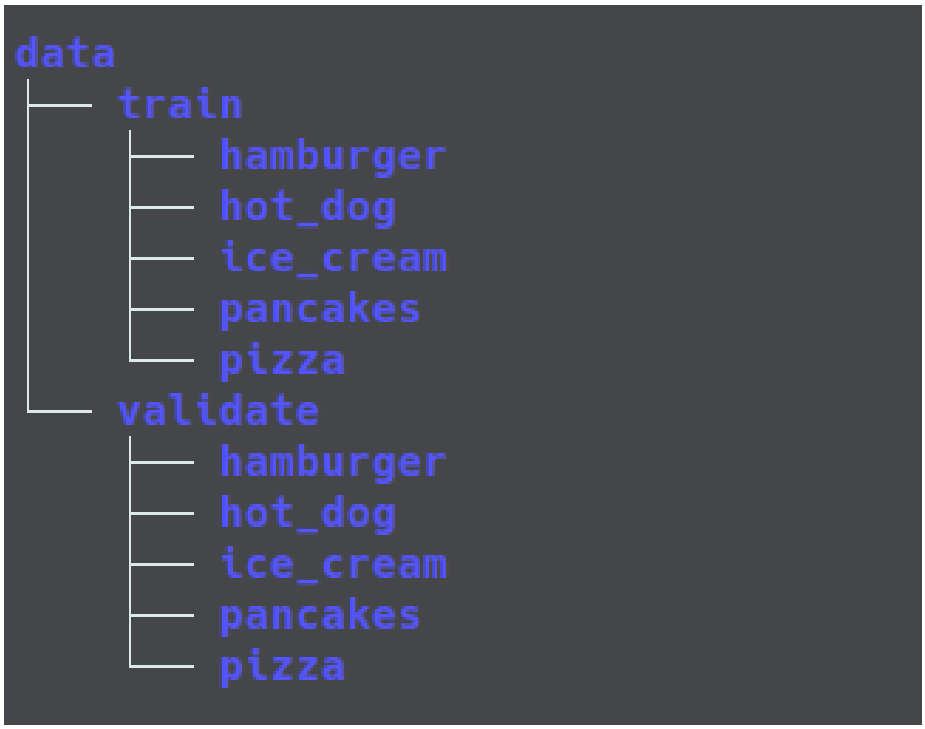
- Dimenzije ulazne slike bi trebalo da budu deljivi sa 2 nekoliko puta
- Poželjno je koristiti male filtere u konvolucionom sloju (3x3 ili 5x5) koristeći korak K=1 i koristiti padding kako konvolucioni sloj ne bi menjao dimenziju ulaza
- Sloj ujedinjavanja je zadužen za smanjivanje dimenzionalnosti ulaza. Najčešće se sreće max-pooling sa 2x2 poljima I korakom K=2

3 Implementacija i eksperimentalni rezultati

Za izradu projekta korišćen je programski jezik Python i biblioteka Keras. Projekat je rađen nad skupom podataka Food-101 koji se sastoji od 101 jela sa po 1000 slika za svako jelo.

Slika 7: Skup podataka Food-101: www.vision.ee.ethz.ch/datasets_extra

Zbog predugog izvršavanja redukovan je skup podataka tako da se u njemu nalazi samo pet klasa sa po 20 slika za trening i po 5 slika za test. Treniranje je izvšeno u 1000 epoha.



Slika 8: Klase za trening i test

```
img_width, img_height = 150, 150
# postavljamo parametre
train_data_dir = 'data/train'
validation_data_dir = 'data/validate'
nb_train_samples = 100
nb_validation_samples = 25
nb_epoch = 1000

model = Sequential()
```

```
# Konvolucija i sazimanje 1
model.add(Conv2D(32, 3, 3, input_shape=(3, img_width, img_height)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Konvolucija i sazimanje 2
model.add(Conv2D(32, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Konvolucija i sazimanje 3
model.add(Conv2D(64, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Transformacija 3D slike u 1D vektor
model.add(Flatten())
model.add(Dense(64)) # viseslojni perceptron
model.add(Activation('relu'))
model.add(Dropout(0.5)) # deaktiviramo polovinu neurona
model.add(Dense(5))  # broj klasa
model.add(Activation('softmax'))

model.summary()
```

Definisani model je sekvencijalan, što znači da su slojevi postavljeni jedan iza drugoga. Definisane su tri operacije konvolucije i tri operacije sažimanje slike. Konvolucija koristi jezgro veličine 3x3 dok sloj sažimanje koristi jezgro veličine 2x2, uzimajući maksimalnu vrednost unutar jezgra. U svakom od slojeva se kao aktivacijska funkcija koristi ReLU(engl. Rectified Linear Unit) koja je definisana na sledeći način: f(x) = max(0,x) pri čemu je x ulaz u neuron. Ona unosi nelinearnost u model i model najčešće znatno bolje opisuje stvarni objekat. Upoređujući sa ostalim aktivacijskim funkcijama kao na primer sigmoidnom, ReLU omogućava brže i efikasnije učenje dubokih neuronskih mreža na velikim i kompleksnim skupovima podataka. Nakon toga sledi transformacija 3D slike u 1D vektor pomoću funkcije Flatten(), a zatim sloj u kojem svaki neuron spajamo sa svakim neuronom u sledećem sloju korišćenjem funkcije Dense(). Pomoću funkcije Dropout() izbegavamo problem prenaučenosti modela. To za posledicu ima da se mreža neće oslanjati na određene neurone nego će učiti što različitije reprezentacije uzoraka. Argument funkcije definiše postotak neurona koji će biti deaktivirani. Koristimo vrednost 0.5 što znači da će polovina svih neurona nasumično da se deaktivira u toku procesa učenja. Na kraju koristimo funkciju Dense(5) koja sve neurone iz prethodnog sloja spaja sa 5 izlaznih klasa koje je potrebno klasificirati. Tu se kao aktivacijska funkcija koristi softmax. Ova funkcija preslikava ulaznu vrednost u interval [0, 1], čime se zapravo dobija verovatnoća pripadnosti određene ulazne slike određenoj klasi, što je zapravo i

izlaz iz mreže.

```
# Konfigurisanje modela
model.compile(
    loss='categorical_crossentropy',
    optimizer=rmsprop,
    metrics=['accuracy'])
```

Zatim je pre procesa treniranja potrebno konfigurisati model. Definišemo kriterijumsku funkciju kao categorical_crossentropy koja koristi algoritam optimizacije rmsprop. Metoda vrednovanja dobijenih rezultata služi da bi se ocenile performanse izgrađenog modela pomoću neke od kriterijumskih funkcija.

Kao rezultat dobijamo arhitekturu modela: broj i vrstu slojeva(Layer (type)), izlaz iz svakog sloja (Output Shape) i broj parametara koje je potrebno podesiti u svakom sloju (Param #).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 148, 148)      896
_________________________________________________________________
activation_1 (Activation)    (None, 32, 148, 148)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 74, 74)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 72, 72)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 32, 72, 72)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 36, 36)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 34, 34)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 64, 34, 34)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 64, 17, 17)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                1183808
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 325
_________________________________________________________________
activation_5 (Activation)    (None, 5)                 0
=================================================================
Total params: 1,212,773
Trainable params: 1,212,773
Non-trainable params: 0
_________________________________________________________________
Found 100 images belonging to 5 classes.
Found 25 images belonging to 5 classes.
```

Slika 9: Rezultat konfigurisanja modela

Koristili smo sledeće argumente funkcije za slučajne transformacije:

- rescale - originalne slike sadrže RGB vrednosti 0-255, koje su prevelike za procesiranje našeg modela, tako da ih skaliramo sa 1/255 faktorom da bi dobili vrednosti između 0 i 1
- shear_range - smicanje
- zoom_range - nasumično zumiranje unutar slike
- hortizontal_flip – slučajno okretanje slike u horizontalnom smeru

```
# slucajne transformacije
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)
```

Nakon što je definisana arhitektura i podešeni svi njeni parametri, možemo da pokrenemo i sam proces treniranja mreže. Funkcija flow_from_directory u beskonačnoj petlji dohvata slike iz objekta train_datagen tj. test_datagen koji generišu slike na koje su primenjene tranformacije. Funkciji fit_generator() prosleđuju se podaci za učenje, podaci za validaciju kao i broj koraka po epohi. Ispod možemo da vidimo kod koji pokreće proces učenja i na kraju sprema model i njegove težine.

```
# treniranje mreze
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')


model.fit_generator(
    train_generator,
```

```
        steps_per_epoch=nb_train_samples//batch_size,
        nb_epoch=nb_epoch,
        # validation_data=validation_generator,
        nb_val_samples=nb_validation_samples//batch_size)

# pamtimo model i tezine modela
model.save_weights('first_try.h5')
```

Na slici ispod prikazan je rezultat mreže u poslednjih pet epoha. Preciznost u poslednjoj epohi je 0.9792 tako da za ovu mrežu možemo da kažemo da je dobro istrenirana.

```
Epoch 996/1000
3/3 [==============================] - 8s 3s/step - loss: 0.1113 - acc: 0.9583
Epoch 997/1000
3/3 [==============================] - 8s 3s/step - loss: 0.0595 - acc: 0.9792
Epoch 998/1000
3/3 [==============================] - 6s 2s/step - loss: 0.1179 - acc: 0.9306
Epoch 999/1000
3/3 [==============================] - 8s 3s/step - loss: 0.0666 - acc: 0.9583
Epoch 1000/1000
3/3 [==============================] - 8s 3s/step - loss: 0.0328 - acc: 0.9792
```

Slika 10: Rezultat rada mreže

4 Zaključak

Konvolucione mreže predstavljaju idealno rešenje za problem klasifikacije slika, međutim ni one nisu savršene. Povećavajući broj mogućih klasa u koje slike mogu biti svrstane, potrebni hardverski resursi za izračunavanja i optimizaciju postaju veliki. Najveći problem koji se ovde javlja je memorijska zahtevnost zbog svih parametara o funkcija aktivacija. Da bi konvolucione neuronske mreže pokazale svoj pun potencijal potrebne su velike količine podataka, a možemo primetiti da i za male skupove slika malih dimenzija konvolucione neuronske mreže zahtevaju dosta računarskih resursa.

Reference

[1] http://cs231n.github.io/convolutional-networks/#conv

[2] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[3] https://www.superdatascience.com/convolutional-neural-networks-cnn-softmax-cross-entropy/