SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Diplomski studij računarstva

Sustav za raspoznavanje pasa i mačaka pomoću konvolucijske neuronske mreže (CNN)

Obrada slike i računalni vid

Seminarski rad

Matej Džijan

Osijek, 2019.

Sadržaj

1. Uvod

Konvolucijska neuronska mreža (Convolutional Neural Network - CNN) je klasa neuronskih mreža za duboko učenje. CNN predstavlja ogroman proboj u prepoznavanju slika. Najčešće se koriste za analizu slika i videa i često se koriste za klasifikaciju slika.

Ovaj seminarski rad opisuje korištenje CNN-a za klasifikaciju fotografija mačaka i pasa.

2. Pregled područja i problematike

Zadatak ovog seminara je napraviti binarnu klasifikaciju fotografija mačaka i pasa. Problem klasifikacije je jedan od osnovnih problema strojnog učenja, ali dodatan problem ovog zadatka je što se klasifikacija treba vršiti na nestrukturiranim podacima, slikama. Neuronske mreže inače uzimaju za ulaz određen broj značajki (*features*), a fotografija u boji veličine 200x200, što je vrlo mala veličina, ima ukupno 120 000 podataka pa uzimanje svakog od tih podataka za značajku nema previše smisla. Taj problem se rješava korištenjem konvolucijskih slojeva.

CNN se sastoji od nekoliko dijelova:

Konvolucijski slojevi

ReLU slojevi

Pooling slojevi

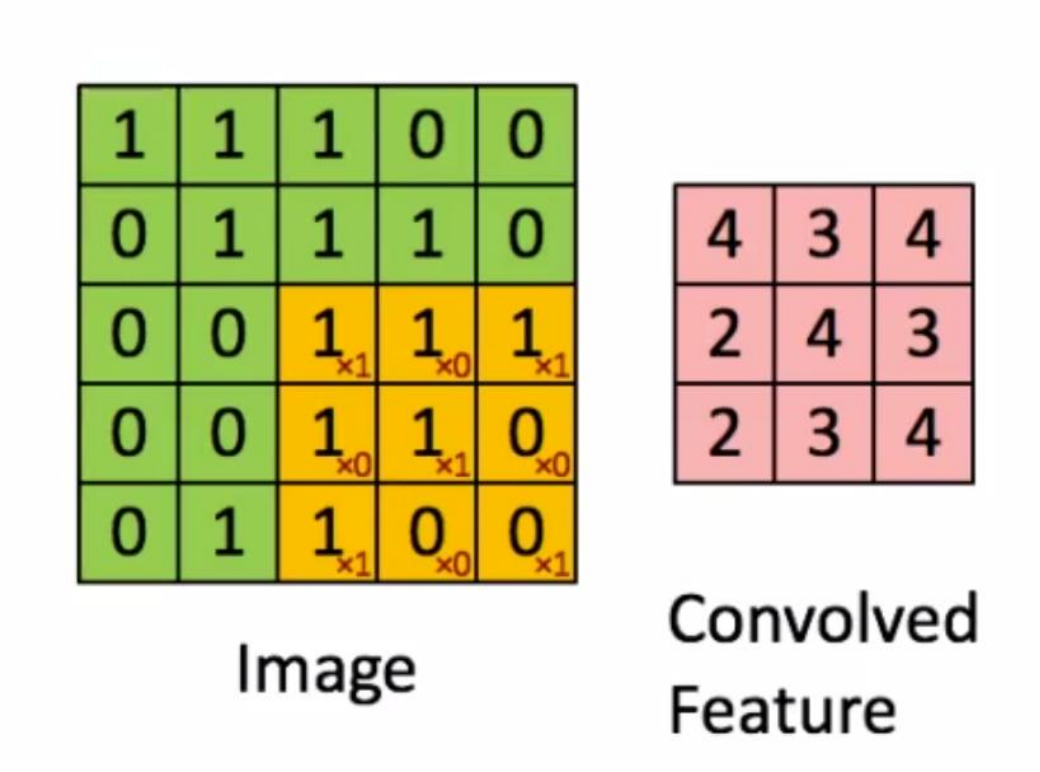
Potpuno spojeni sloj

Tako tipična CNN može izgledati ovako:

Ulaz -> Konvolucija -> ReLU -> Pooling -> Konvolucija -> ReLU -> Pooling -> Konvolucija -> ReLU -> Pooling -> Potpuno spojeni

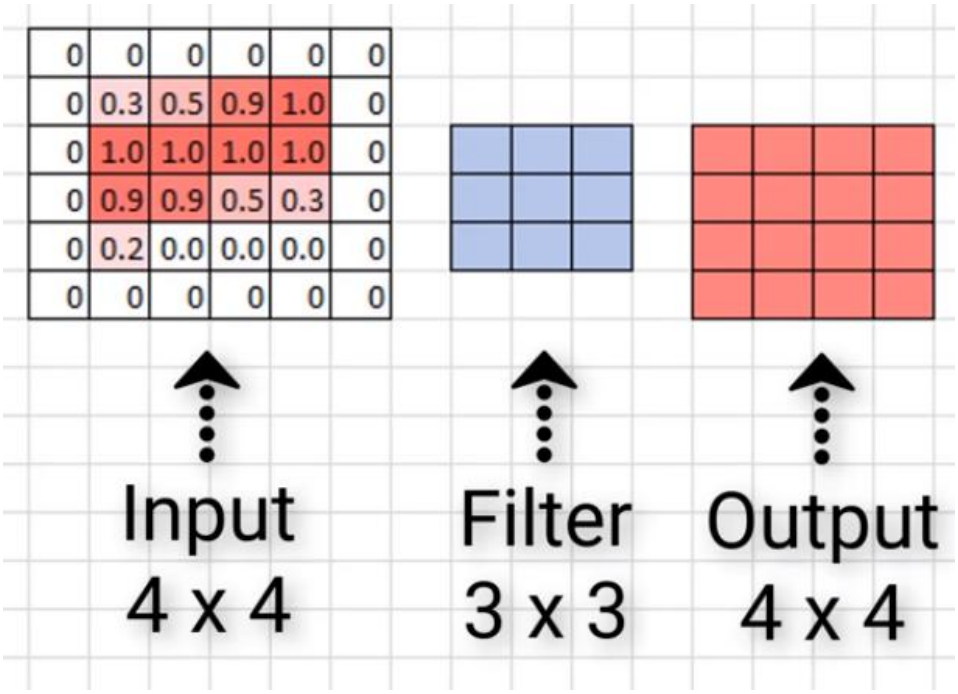
3. Konvolucija

Glavni cilj konvolucije je izvući značajke iz ulazne slike. Konvolucija je uvijek prvi korak u CNN-u. Imamo ulaznu sliku, detektor značajki i kartu značajki ili aktivacijsku kartu. Uzima se filter (detektor značajki) koji je obična matrica, često veličine 3x3, 5x5 i slično, te se taj filter primjenjuje blok piksela po blok piksela. Filter se primjenjuje na način da se uzme blok piksela i skalarno se pomnoži s filterom. Na ovaj način se dobije jedna ćelija izlazne matrice, karte značajki, te se filter pomiče za zadani broj ćelija (*stride*), promatra se novi blok piksela i proces se ponavlja dok se ne dobije gotova karta značajki kao izlaz.



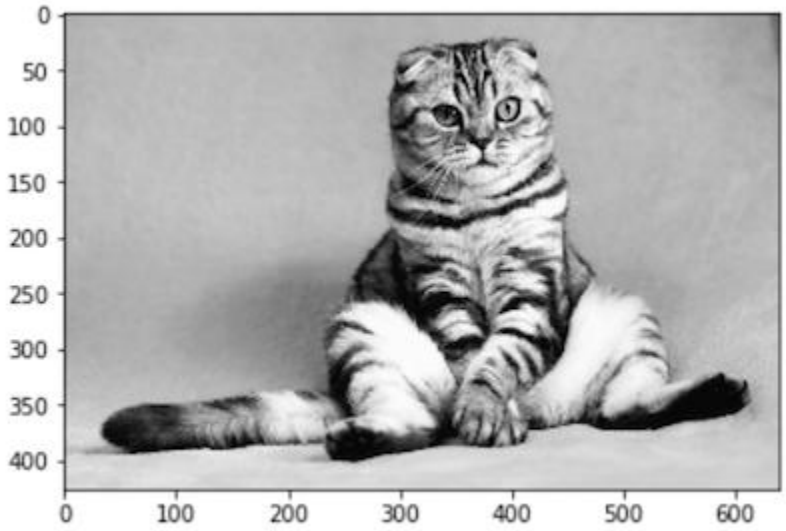
Slika 1. Prikaz konvolucije

Ovim postupkom se izgubi dio informacija, ali se, ovisno o filteru, prepoznaju različite značajke što je upravo i cilj detektora značajki. Vidimo da se za izlaz dobije slika manje veličine što možda ne želimo u ovom koraku učiniti, stoga se može proširiti ulaznu matricu nulama sa svih strana (*zero padding*) da se dobije izlaz jednake veličine ulazu.



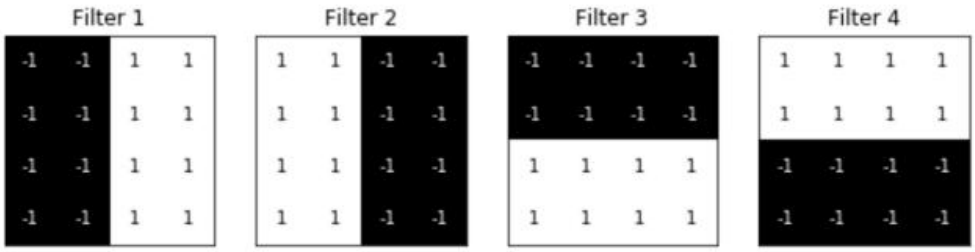
Slika 2. Prikaz *zero paddinga*

Možemo vizualizirati ovaj proces na primjeru fotografije mačke:



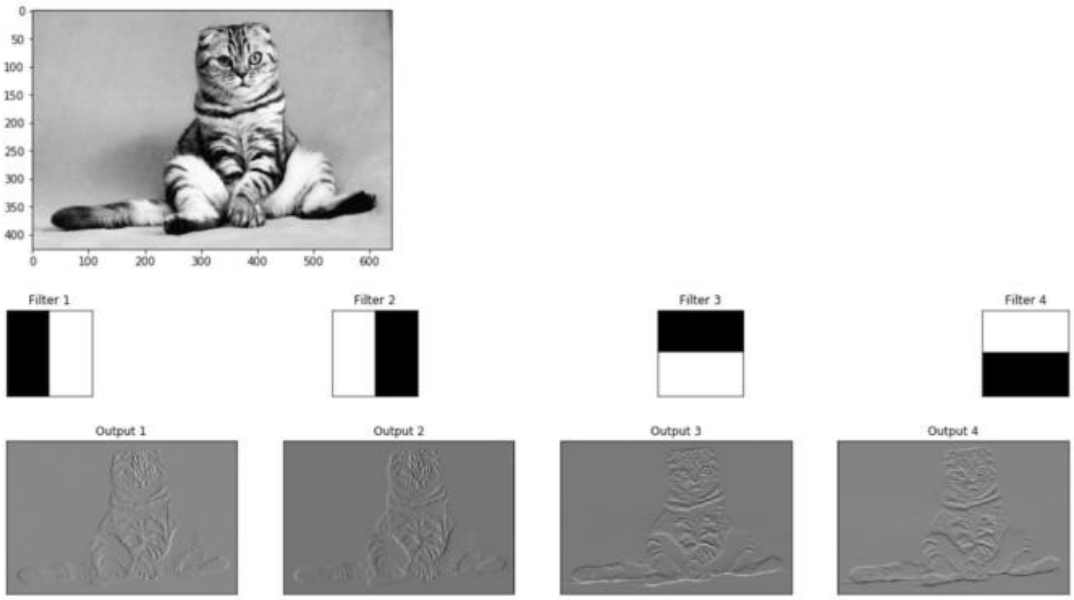
Slika 3. Fotografija mačke

Možemo definirati nekoliko različitih filtara koje ćemo primijeniti na ovu sliku:



Slika 4. Različiti filtri

Gore opisani proces se primjenjuje na svaki od blokova u originalnoj slici s pomicanjem za jedan piksel. Te se za izlaz dobije:



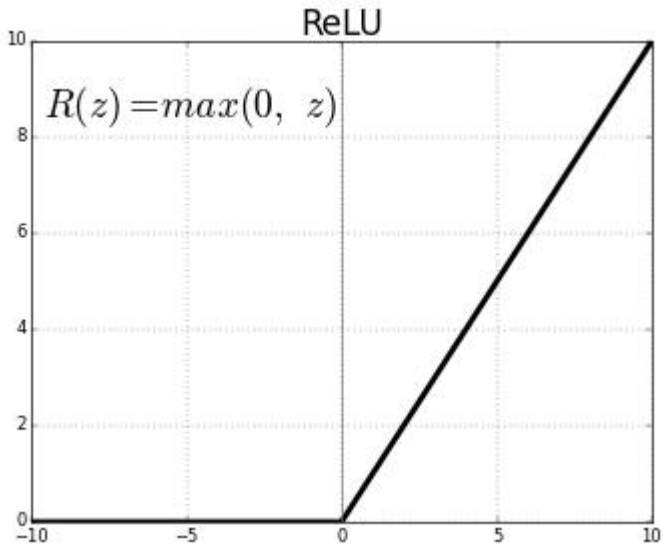
Slika 5. Rezultati konvolucije

Za rezultate dobijemo slike na kojima se mogu primijetiti iscrtani rubovi mačke. Sljedeći korak je primjena ReLU funkcije na obrađenu sliku.

3.1. ReLU funkcija

ReLU (*rectified linear unit*) funkcija je aktivacijska funkcija definirana kao pozitivan dio argumenta, odnosno, vraća pozitivne ulaze kakvim jesu, a negativne postavlja na nula:

$$f(x) = x^+ = \max(0, x)$$



Slika 6. Prikaz ReLU funkcije

ReLU sloj omogućava dodatno isticanje značajki kao što su rubovi. To možemo vidjeti na primjeru mačke gdje rubovi dodatno istaknu u odnosu na rezultat konvolucije:

Slika 7. Rezultat ReLU funkcije

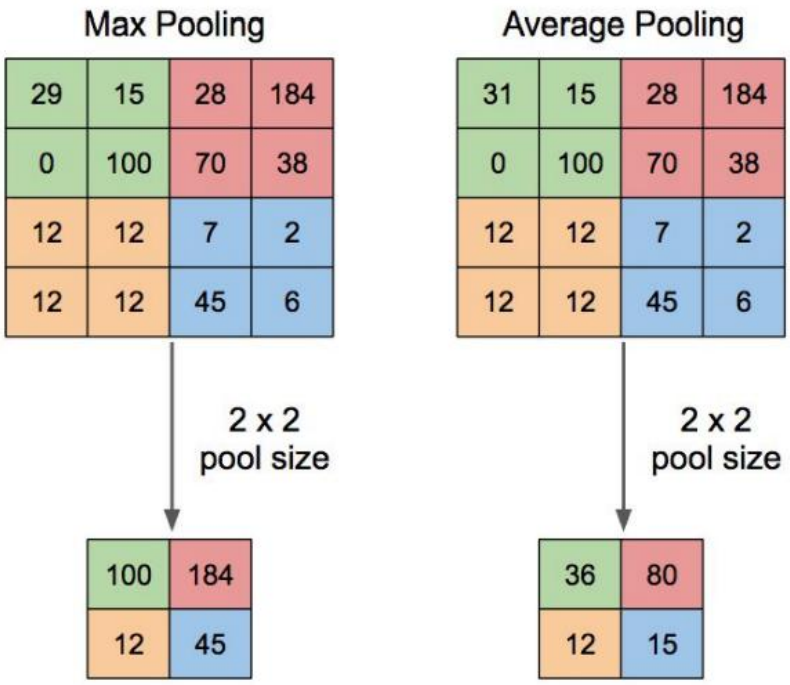
Na ovaj način se dobiju slike jednake veličine pa i dalje imamo problem velike količine podataka, ali sad ti podaci prikazuju nekakve značajke. Sljedeći korak može biti dodatno smanjivanje ovih podataka upotrebom *poolinga*.

3.2. Pooling

Nešto što nije dobro da neuronska mreža radi je da traži specifičnu značajku točne nijanse na točnom mjestu. Za dobar CNN je potrebno da radi na raznim fotografijama, rotiranim, zrcaljenim, odrezanim, spljoštenim i slično. Da bi se to izbjeglo, koristi se *pooling*. *Pooling* je proces u kojem se ulazna matrica progresivno smanjuje. Na ovaj se način postiže smanjivanje veličine ulaznih podataka i omogućuje detektiranje značajki bez obzira gdje se nalaze na slici. Također, *pooling* pomaže pri smanjivanju *overfittinga*. Za ovo je, također, potreban skup podataka za treniranje s raznim slikama, nije dobro imati slike samo jedne pasmine za pse i slično.

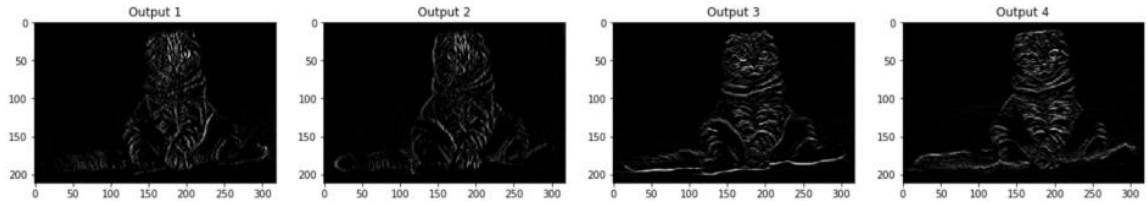
Najpopularniji oblik *poolinga* je *max pooling*. Ulazna slika se podijeli na manje dijelove koji se ne preklapaju. Izlaz svakog od tih manjih dijelova je maksimalna vrijednost tog dijela. Kao izlaz se dobije manja matrica. Ako se *pooling* vrši s podjelom na 2x2 dijelove, dobije se slika dva puta uža i dva put niža, odnosno četiri puta manja. Na ovaj način se izgubi 75% informacije koji nisu značajke. Na ovaj način se neutraliziraju iskrivljenja.

Drugi oblici *poolinga* su *average pooling* i *sum pooling*, odnosno *pooling* po prosjeku i *pooling* po zbroju. Oni nisu česti izbori jer *max pooling* ima bolje rezultate u praksi.



Slika 8. *Max pooling* i *Average pooling*

Tako će *max pooling* na slici mačke nakon ReLU funkcije izgledati ovako:



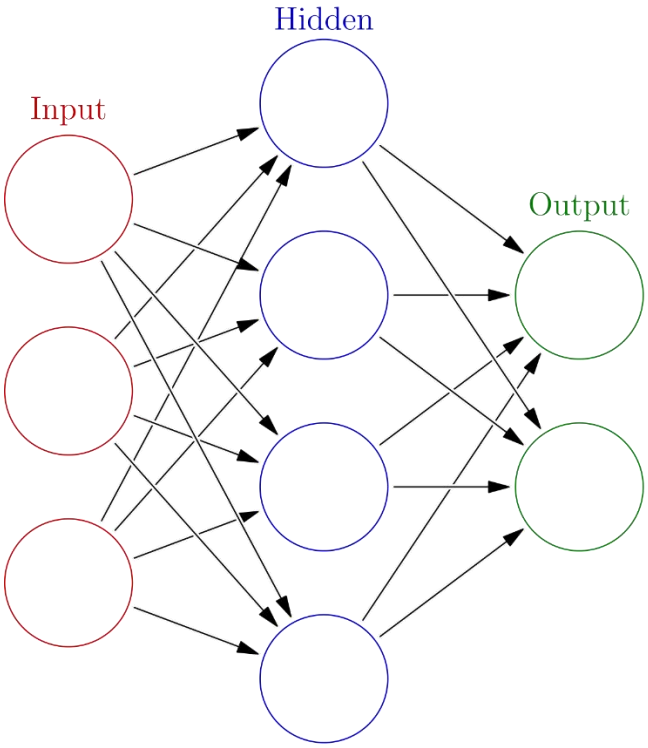
Slika 9. Slika mačke nakon *poolinga*

Rezultantna slika je duplo uža i duplo niža, odnosno četiri puta manja.

Sljedeći korak nakon *poolinga* može biti ravnanje (*flattening*). Ovo je vrlo jednostavan korak u kojem se mapa značajki nakon *poolinga* ravna u jedan sekvencijalni stupac brojeva (vektor). Ovo omogućava toj informaciji da postane ulazni sloj u umjetnu neuronsku mrežu za daljnju obradu.

3.3. Potpuno spojeni sloj

Nakon svih slojeva konvolucije, ReLU-a i *poolinga*, dodaje se umjetna neuronska mreža. Cilj ovog sloja je da uzme dane ulaze i preko skrivenog sloja neurona predvidi klasu za dani ulaz. Sastoji se od niza neurona i poveznica između njih, sinapsi. U ovom slučaju je riječ o potpuno spojenoj neuronskoj mreži, odnosno, svaki od izlaza iz prethodnog sloja je spojen na svaki neuron u sljedećem sloju.



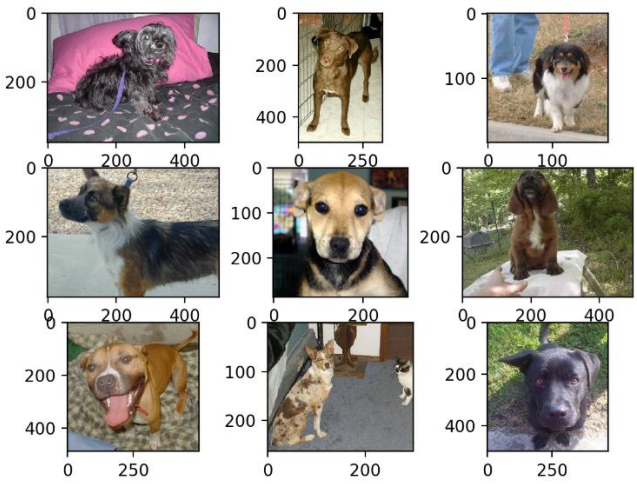
Slika 10. Prikaz potpuno spojene umjetne neuronske mreže

Svaka od sinapsi ima određenu težinu, između 0 i 1. Veće težine predstavljaju veću važnost prethodnog neurona za izlaz. Ulazni podatak prolazi kroz neuronsku mrežu, računaju se izlazi, te se iz izlaza računa pogreška. Na temelju te pogreške se zatim težine sinapsi mijenjaju i proces se ponavlja. Na ovaj način se dobije mreža koja daje veću vrijednost određenim značajkama za jednu klasu, a drugim značajkama za neku drugu klasu. Na kraju se primjenjuje *softmax* funkcija na izlazu te se dobivaju vjerojatnosti za svaku od zadanih klasa.

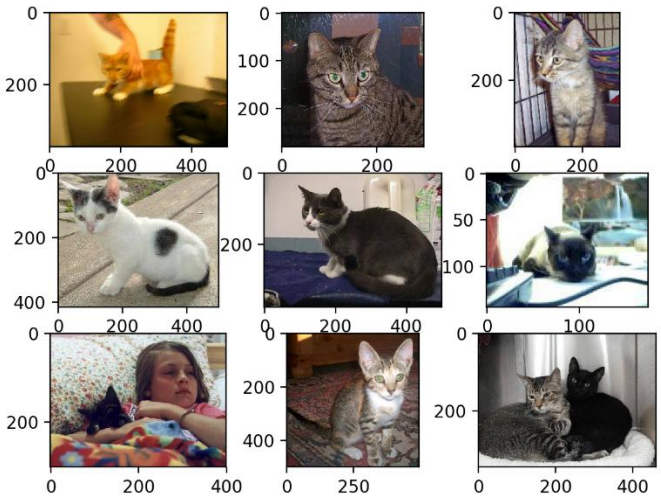
4. Rezultati zadatka

Zadatak ovog projekta je izgradnja konvolucijske neuronske mreže koja će razlikovati fotografije mačaka i pasa.

Prvi korak ovog projektnog zadatka je bilo prikupljanje podataka. To je učinjeno uzimanjem gotovog skupa podataka koji se sastoji od 12 500 fotografija mačaka i 12 500 fotografija pasa (link). Fotografije su raznih oblika i s raznim psima i mačkama. Mnoge fotografije imaju razne druge objekte na njima kao što su ljudi, namještaj i slično. Ovo je dobro za treniranje neuronske mreže jer smanjuje mogućnost *overfittinga* i generalno poboljšava rad.



Slika 11. Nekoliko fotografija pasa iz ulaznog skupa podataka



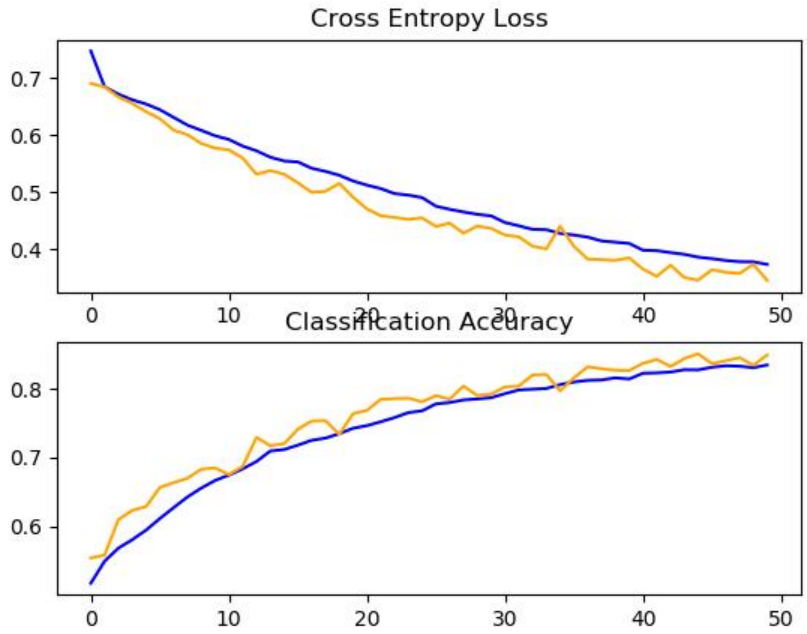
Slika 12. Nekoliko fotografija mačaka iz ulaznog skupa podataka

Za realizaciju CNN-a korištena je Python biblioteka Keras. Ova biblioteka omogućava izgradnju konvolucijskih neuronskih mreža sloj po sloj.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

Slika 13. Izgradnja CNN-a pomoću Kerasa

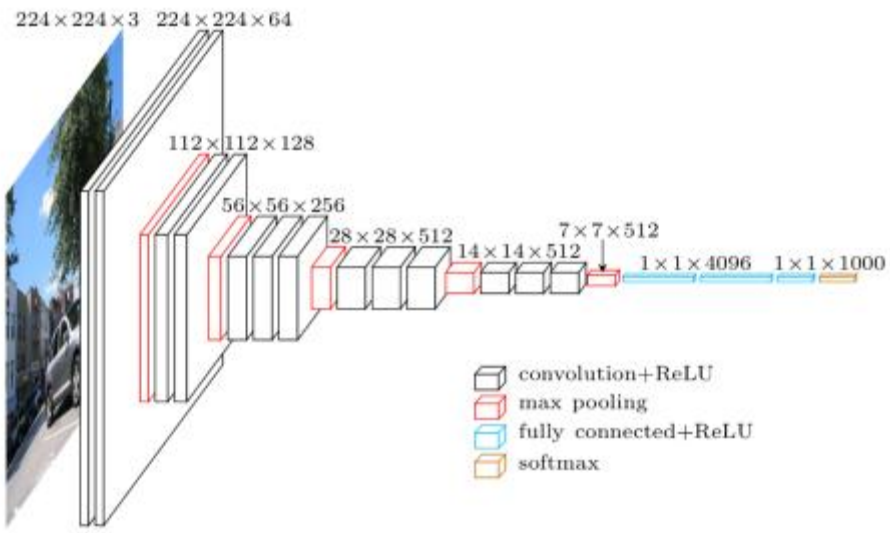
Na slici 13. se vidi izgradnja jedne konvolucijske neuronske mreže pomoću Kerasa. Ova mreža se sastoji od tri konvolucijska sloja, svaki sa svojim *poolingom*. Nakon svakog od tih slojeva, također se vrši i *dropout*, odnosno ispuštanje dijela podataka. Ovo se vrši kako bi se smanjio *overfitting*. Ovim modelom je postignuta točnost od 84.975%.



Slika 14. Rezultati evaluacije modela s tri konvolucijska sloja po epohama

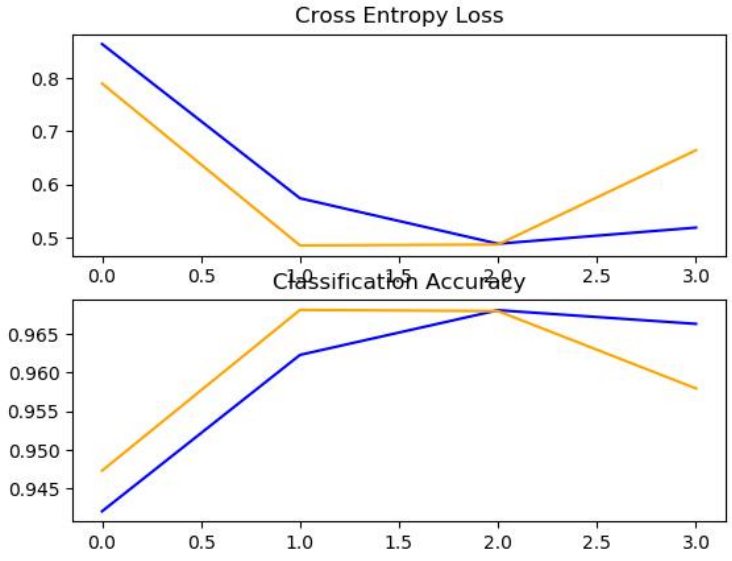
Iz rezultata se vidi da nije bilo *overfittinga*, ali rezultati se nisu znatno poboljšavali pa daljnje treniranje ne bi rezultiralo točnijim modelom.

Nakon toga je isproban VGG16 model koji je implementiran u Keras biblioteci. Sastoji se od niza konvolucijskih slojeva s *max pooling* slojevima između, na kraju su dva potpuno spojena sloja i *softmax* klasifikator kao što je prikazano na slici 15. Ogromni problemi ovog modela su to što je vrlo spor za treniranje i velika količina memorije koju zauzimaju arhitekture tog modela.



Slika 15. VGG16 model

Ovim modelom je postignuta točnost od 95.796% sa samo 4 epohe (nije izvođeno s više epoha zbog ograničenja korištenog sklopovlja).



Slika 16. Rezultati VGG16 modela

Ovaj model je zatim spremljen i može se koristiti za predviđanje klasa fotografija mačaka i pasa. Tako je za ovu fotografiju psa, ispravno predvidio klasu '1', odnosno da je pas.

Slika 17. Fotografija psa

5. Zaključak

Problem klasifikacije i prepoznavanja slika postaje sve prisutniji u današnjem svijetu interneta gdje je jako velika količina podataka upravo u obliku slike i videa. Konvolucijske neuronske mreže su donijele ogroman proboj u tom području. U ovom seminarskom radu je prikazana jednostavnost korištenja istih pomoću gotovih alata biblioteke Keras. Uz vrlo slabo sklopovlje su postignuti izvrsni rezultati s 95% točnosti.

Literatura

https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/

https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb

https://www.youtube.com/playlist?list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU

Dataset: https://www.kaggle.com/c/dogs-vs-cats/data

Prilozi

Izvorni kod (dostupan i na github link-u: https://gitlab.com/dzijo/dog-vs-cat-classification)

Izvorni kod

Prvi model

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator

import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))

# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
```

```
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    return model

# plot diagnostic learning curves
def summarize_diagnostics(history):
    # plot loss
    pyplot.subplot(211)
    pyplot.title('Cross Entropy Loss')
    pyplot.plot(history.history['loss'], color='blue', label='train')
    pyplot.plot(history.history['val_loss'], color='orange', label='test')
    # plot accuracy
    pyplot.subplot(212)
    pyplot.title('Classification Accuracy')
    pyplot.plot(history.history['acc'], color='blue', label='train')
    pyplot.plot(history.history['val_acc'], color='orange', label='test')
    # save plot to file
    filename = sys.argv[0].split('/')[-1]
    pyplot.savefig(filename + '_plot.png')
    pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
    # define model
    model = define_model()
    # create data generators
```

```
    train_datagen = ImageDataGenerator(rescale=1.0/255.0,
        width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
    test_datagen = ImageDataGenerator(rescale=1.0/255.0)
    # prepare iterators
    train_it = train_datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
        class_mode='binary', batch_size=32, target_size=(200, 200))
    test_it = test_datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
        class_mode='binary', batch_size=32, target_size=(200, 200))
    # fit model
    history = model.fit_generator(train_it, steps_per_epoch=len(train_it),
        validation_data=test_it, validation_steps=len(test_it), epochs=50, verbose=0)
    # evaluate model
    _, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=0)
    print('> %.3f' % (acc * 100.0))
    # learning curves
    summarize_diagnostics(history)
    model.save('final_model.h5')

# entry point, run the test harness
with tf.device('/device:GPU:0'):
    run_test_harness()
```

VGG16 model:

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator
```

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))


# define cnn model
def define_model():
	# load model
	model = VGG16(include_top=False, input_shape=(224, 224, 3))
	# mark loaded layers as not trainable
	for layer in model.layers:
		layer.trainable = False
	# add new classifier layers
	flat1 = Flatten()(model.layers[-1].output)
	class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
	output = Dense(1, activation='sigmoid')(class1)
	# define new model
	model = Model(inputs=model.inputs, outputs=output)
	# compile model
	opt = SGD(lr=0.001, momentum=0.9)
	model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
	return model


# plot diagnostic learning curves
def summarize_diagnostics(history):
	# plot loss
	pyplot.subplot(211)
	pyplot.title('Cross Entropy Loss')
	pyplot.plot(history.history['loss'], color='blue', label='train')
	pyplot.plot(history.history['val_loss'], color='orange', label='test')
```

```
        # plot accuracy
        pyplot.subplot(212)
        pyplot.title('Classification Accuracy')
        pyplot.plot(history.history['acc'], color='blue', label='train')
        pyplot.plot(history.history['val_acc'], color='orange', label='test')
        # save plot to file
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_plot.png')
        pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
    # define model
    model = define_model()
    # create data generator
    datagen = ImageDataGenerator(featurewise_center=True)
    # specify imagenet mean values for centering
    datagen.mean = [123.68, 116.779, 103.939]
    # prepare iterator
    train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
       class_mode='binary', batch_size=32, target_size=(224, 224))
    test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
       class_mode='binary', batch_size=32, target_size=(224, 224))
    # fit model
    history = model.fit_generator(train_it, steps_per_epoch=len(train_it),
       validation_data=test_it, validation_steps=len(test_it), epochs=4, verbose=1)
    # evaluate model
    _, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=0)
    print('> %.3f' % (acc * 100.0))
    # learning curves
    summarize_diagnostics(history)
    model.save('final_model_TL.h5')
```

```
# entry point, run the test harness
with tf.device('/device:GPU:0'):
    run_test_harness()
```

Predikcija klase:

```
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
import numpy as np

import os
import tensorflow as tf
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

def load_image(filename):
    # load the image
    img = load_img(filename, target_size=(224, 224))
    # convert to array
    img = img_to_array(img)
    # reshape into a single sample with 3 channels
    img = img.reshape(1, 224, 224, 3)
    # center pixel data
    img = img.astype('float32')
    img = img - [123.68, 116.779, 103.939]
    return img

def predict(image):
    # load the image
    img = load_image(image)
```

```
    # load model
    model = load_model('final_model_TL.h5')
    # predict the class
    result = model.predict(img)
    return result[0][0].astype(int)

print(predict('66832444_2275116232582234_7067699075927244800_n.jpg'))
```