SVEUČILIŠTE U RIJECI
ODJEL ZA INFORMATIKU
Preddiplomski jednopredmetni studij informatike

Matteo Kinkela

Klasifikacija s TensorFlow

Završni rad

Mentor: doc. dr. sc. Marina Ivašić - Kos

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 15.2.2018.

Zadatak za završni rad

Pristupnik: **Matteo Kinkela**

Naziv završnog rada: **Klasifikacija s TensorFlow**

Naziv završnog rada na eng. jeziku: **Classification with TensorFlow**

Sadržaj zadatka: Opisati osnovne termine vezane za raspoznavanje uzoraka kao što su metode strojnog učenja s naglaskom na neuronske mreže i duboko učenje. Proučiti TensorFlow okolinu za dubinsko učenje i opisati njeno korištenje za problem klasifikacije slika.

Opisati ekspriment klasifikacije znamenaka kao problem klasifikacije u više klasa i problem klasifikacije filmova u žanrove temeljem njihovih postera. U oba slučaja opisati bazu koja se koristi, arhitekturu mreže, metriku za evaluaciju i dobivene rezultate.

Mentor

doc. dr. sc. Marina Ivašić-Kos

Voditelj za završne radove
3624

dr. sc. Miran Pobar

Zadatak preuzet: 15.2.2018.

(potpis pristupnika)

Sadržaj

1 Sažetak

U ovom radu prikazano je korištenje TensorFlowa i Kerasa u izradi modela za duboko učenje. Rješavali smo problem klasifikacije znamenaka i problem klasificiranja filmskih postera u žanrove. Za rješavanje problema klasifikacije korištena je TensorFlow duboka neuronska mreža te je objašnjena njena arhitektura i aktivacijske funkcije. Prikazane su sve faze učenja koje su potrebne za stvoriti dobar model počevši od analize podataka, preko izrade i učenja modela i na kraju testiranje i analize rezultata. Problem klasificiranja zanemanaka riješen je na jedan a problem klasifikacije filmskih postera na dva načina te su detaljno opisane dobre i loše strane svakog od rješenja.

Ključne riječi: TensorFlow, Keras, Python, duboko učenje, neuronska mreža, klasifikacija.

2 Uvod

Možemo primjetiti da je umjetna inteligencija danas svuda oko nas. Primjerice, kada pišemo poruku na pametnom telefonu, možemo vidjeti da nam softver preporuča koju riječ napisati. Kroz vrijeme kako pišemo poruke, taj isti softver prepoznaje koje riječi koristimo češće i prilagođava nam se, uči od nas.

Posvuda možemo vidjeti razne reklame za dućane, proizvode i među ostalom možemo vidjeti filmske najave. Na ulici ćemo ih vidjeti u obliku postera. Posteri su slike većih dimenzija. Najčešće prikazuju glumce iz filma ili neku scenu. Također, možemo vidjeti naziv filma i često popis glumaca, redatelja, itd. Oni su postavljeni kako bi privukli publiku da pogledaju film u kinu.

Ljudi će iz postera većinom znati o kojem se žanru radi. Vjerojatno jer su negdje pročitali o filmu ili prepoznaju rad redatelja ili osoblja koji rade na njemu. Međutim, pitanje je hoće li netko tko nije upoznat sa tim filmom znati o kojem se žanru radi. Pretpostavljamo da postoje određena pravila kako izraditi poster za određeni žanr filma, npr. animirani film bi trebao biti šaren, posteru za neki horor bi trebala prevladavati crna boja ili možda krv na slici, itd. Naša računala ne znaju ništa o tim filmovima te smo se tako odlučili ispitati možemo li naučiti računalo kako pomoću slika filmskih postera odrediti kojeg je žanra film.

3 Raspoznavanje uzoraka i metode strojnog učenja

Raspoznavanje uzoraka[1] je grana strojnog učenja koja se bavi prepoznavanjem uzoraka i pravilnosti u podacima. Raspoznavanje uzoraka možemo podijeliti u kategorije prema načinu učenja koji se koristi za generiranje izlaza na nadzirano i nenadzirano učenje.

3.1 Nadzirano učenje

Nadzirano učenje (*engl. supervised learning*) pretpostavlja da postoji skup podataka za učenje modela. Taj skup se sastoji od podataka koji služe kao ulaz u model i podataka koji predstavljaju točan izlaz. Cilj nadziranog učenja je stvoriti model koji ćemo učiti na danim podacima tako da može stvarati dobre predikcije na skupu podataka koji još nije vidio.

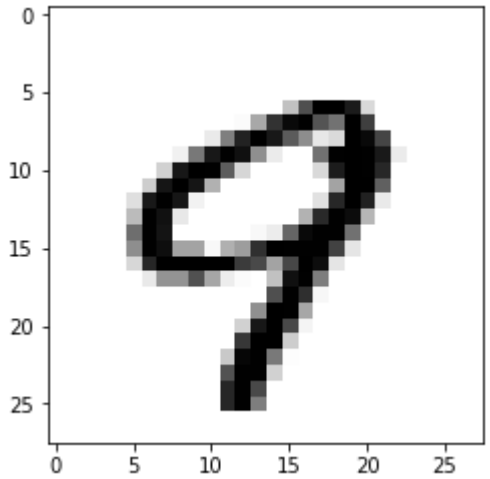
3.2 Nenadzirano učenje

Nenadzirano učenje[4] (*engl. unsupervised learning*) je grana strojnog učenja u kojoj imamo samo ulazne podatke, ne i izlazne. Ime nenadzirano učenje dobilo je po tome što ne postoji točno rješenje i ne postoji "učitelj" koji nadzire učenje. Cilj nenadziranog učenja je stvoriti model koji će učiti činjenice o podacima. Problemi nenadziranog učenja mogu biti podijeljeni u klasterizacijske probleme i probleme asocijacije. Kod klasterizacije želimo pronaći određene grupe u podacima, npr. želimo grupirati ljude po omiljenom žanru filma. S druge strane, kod problema asocijacije želimo pronaći pravila kako povezati veliku količinu podataka, npr. ljudi koji vole animirane filmove, voljet će i komedije.

4 Klasifikacija

Klasifikacija je problem nadziranog učenja u kojem je potrebno napraviti model koji dane ulazne podatke kategorizira po zadanim kategorijama. Za potrebe ovog rada, spomenut ćemo dvije vrste klasifikacije, to su single-label i multi-label klasifikacija.

Single-label klasifikacija je klasifikacija u kojoj dani podatak možemo svrstati u samo jednu ponuđenu kategoriju od većeg broja danih kategorija. U ovom radu, primjer za single-label klasifikaciju bio bi klasifikacija znamenaka kod kojeg se dana slika neke znamenke klasficira u njoj odgovarajuću oznaku znamenke. Broj različitih znamenki je 10, a svaka slika klasificira se u jednu od njih.



Slika 1: Primjer znamenke 9

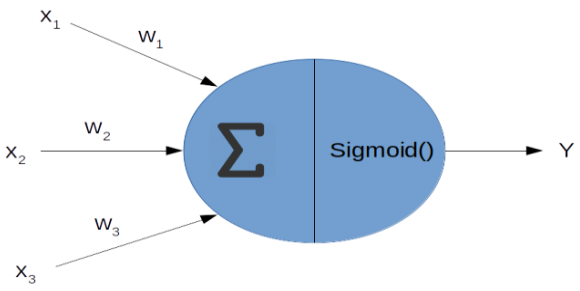
U slučaju kada postoje samo dvije kategorije kao što je npr. znamenka/ nije znamenka ili zdrav/bolestan, govorimo o binarnoj klasifiakciji.

Multi-label klasifikacija je klasifikacija u kojoj dani podatak možemo svrstati u više kategorija. U ovom radu, primjer za multi-label klasifikaciju bio bi klasifikacija postera po filmskim žanrovima kod kojih se neki poster može klasificirati u jedan do više žanrova. Npr. poster za film Innisifree pripada dokumentarnom filmu, a poster za film Brothert Bear 2 pripada akcijskom, animiranom, komediji, drami i obiteljskom filmu.

Slika 2: Innisifree pripada dokumentarnom filmu

Slika 3: Brother Bear 2 pripara akcijskom, animiranom, komediji, drami i obiteljskom filmu

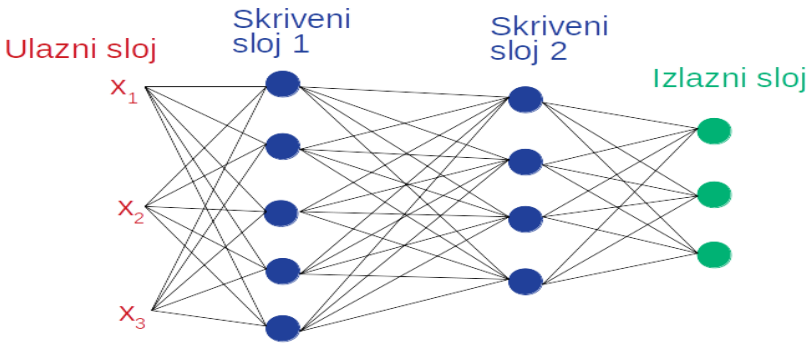
4.1 Neuronske mreže



Slika 4: Umjetni neuron

Neuronske mreže su jedna od metoda nadziranog učenja. Inspirirane su ponašanjem neurona u mozgu. Ideja bi bila oponašati neuron. On se sastoji od dendrita, jezgre, aksona te njegovih završetaka. Za mrežu, potrebno je najmanje dva neurona koji razmjenjuju informacije pomoću sinapse.

Iz slike 4 možemo vidjeti kako koncept izgleda u primjeni u računalnoj znanosti. Ulazne podatke čine x_1, x_2 i x_3. Oni se svaki posebno množe sa odgovarajućim težinama te se ti rezultati koriste kao ulaz u funkciju (u ovom slučaju sumiranje). Funkcija će stvoriti izlaz te poslati svoj izlaz na ulaz sljedeće funkcije koja vrši izlaz iz neurona.



Slika 5: Koncept neuronske mreže

Na slici 5 možemo vidjeti kako bi izgledala jedna neuronska mreža. Vidimo da je podijeljena u slojeve, svaki stupac je jedan sloj. Prvi stupac označava ulazni sloj, a posljednji stupac označava izlazni sloj. Svi slojevi unutar su skriveni slojevi. Duboka neuronska[2] mreža je umjetna neuronska mreža u kojoj nisu svi međusobno povezani čvorovi kao u regularnoj neuronskoj mreži. Potpuno su povezani obično posljednjih nekoliko slojeva (*dense layer*). Konvolucijska neuralna mreža ima filtere (objašnjeno u sljedećem poglavlju) koji se pomiču po slici i izdvajaju bitne značajke.

U današnje vrijeme za klasifikaciju slika često se koriste konvolucijske neuronske mreže gotove koje su prethodno naučene na velikim skupovima slika kao štoje COCO ili Imagenet. Takvi modeli koriste se za predviđanje, ekstrakciju značajki i fino podešavanje[3]. Neki od tih modela su Xception, VGG19, RestNet50, OmceptionV3, itd.

5 TensorFlow

5.1 O TensorFlowu

TensorFlow je biblioteka otvorenog koda za strojno i duboko učenje. Razvio ga je Google Brain tim koji se bavi dubokim učenjem i umjetnom inteligencijom. Prvotna misija bila mu je za interno korištenje u Google-u.

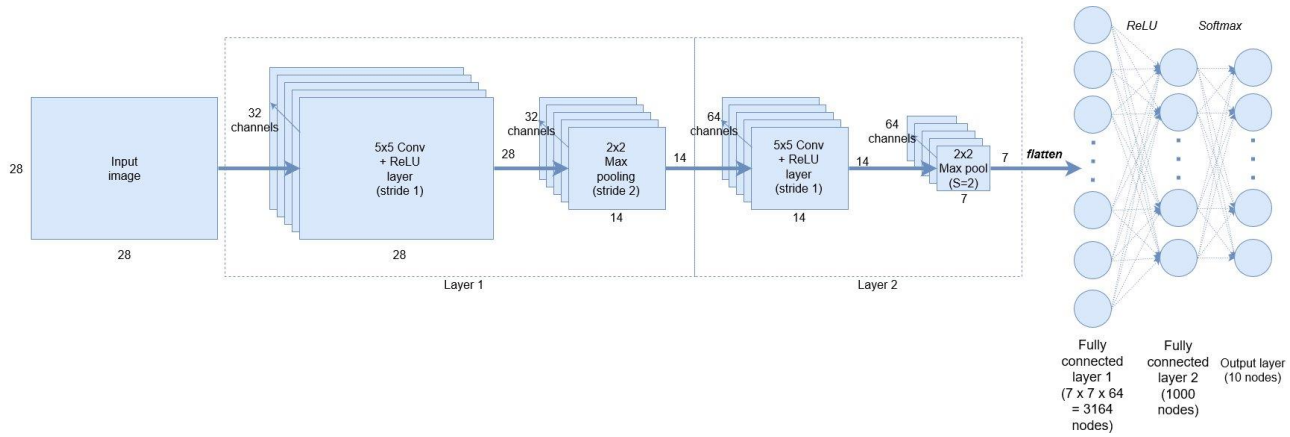
Slika 6: Logo TensorFlowa; preuzeto sa; [5]

Preteča TensorFlowa bila je DistBelief. Projekt[5] je počeo 2011. godine u Google-u. Koristio je strojno učenje za prepoznavanje karakterističnih uzoraka na slikama. Program je radio tako da je koristio pozitivno pojačanje. Ako program izbaci točno rješenja, znači da radi dobro, međutim, ako izbaci krivo rješenje, onda je sustav prilagođen tako da prepozna neke druge uzorke sa slike kako bi idući put izbacio bolje rješenje.

TensorFlow koristi taj koncept korištenjem dubokog učenja, tj. konvolucijske neuronske mreže sastavljene od mnogo slojeva. Početna verzija puštena je 11. veljače 2017. Ime je dobio po tome jer za računanje koristi multidimenzionalne matrice koje se nazivaju tenzori.

5.2 Arhitektura

Neuronska mreža građena je u slojevima koji mogu biti različite vrste. Ovdje ćemo spomenuti one slojeve koje smo koristili za rješavanje zadataka.

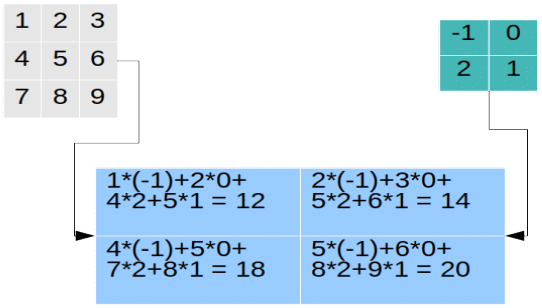


Slika 7: Primjer arhitekture neuronske mreže; [6]

Na slici 7 možemo vidjeti primjer arhitekture neuronske mreže. U skrivenim slojevima mreža ima dva konvolucijska sloja (Conv) sa MaxPool2D operacijama te dva potpuno povezana sloja (Fully connected). Ukupno ima 5 slojeva, 1 ulazni i 4 skrivena sloja.

5.2.1 Conv2D

Kroz dvodimenzionalni konvolucijski sloj uglavnom provlačimo slike. On radi tako da veliku matricu u kojoj je zapisana slika podijeli u više matrica veličine filtera. Veličinu filtera odabire programer, a najčešće se odabiru veličine 2x2, 3x3 i 5x5. Filtere možemo zamisliti kao težine bridova. Za svaku podmatricu glavne matrice množimo jedan njezin element sa elementom filtera na istoj poziciji. Sve umnoške zbrojimo te kada zapišemo rezultate dobijemo matricu na izlazu.



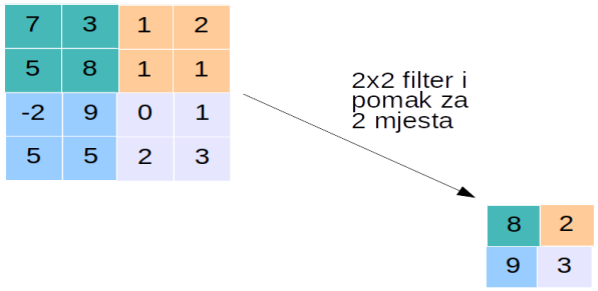
Slika 8: Način rada 2D konvolucijskog sloja

Na slici 8 možemo vidjeti kako radi konvolucijski dvodimenzionalni sloj. Dakle, imamo ulaznu matricu (obojana sivo) i filter matricu (obojana zeleno). Izlazna matrica je plave boje. Zamislimo da možemo staviti ulaznu matricu i filter matricu jednu preko druge. U prvom računanju, filter matricu bismo postavili u gornji lijevi kut, tako da se vrijednost 1 (ulazna

matrica) i vrijednost -1 (filter matrica) nalaze jedan preko drugoga. Elemente koje se nalaze na istim pozicijama pomnožimo i tako dobijemo 1*(-1), 2*0, 4*2 i 5*1. Te rezultate zbrojimo i dobijemo vrijednost 12. Tu vrijednost zapišemo u godnji lijevi kut izlazne matrice. Nakon toga prividno pomičemo filter matricu za jedno mjesto udesno i ponavljamo postupak. Kada završimo sa redom, pomičemo se jedan red prema dolje.

5.2.2 MaxPool2D

Ova operacija smanjuje dimenziju matrice tako da ju podijeli na regije, npr. 2x2. Zatim u svakoj regiji pronađe maksimalnu vrijednost te stvori novu matricu koja se sastoji samo od tih maksimalnih vrijednosti.



Slika 9: Način rada MaxPool2D operacije

Na slici 9 možemo vidjeti da imamo matricu veličine 4x4. Pretpostavimo da nam je veličina filtera 2x2 te da se filter pomiče za 2 mjesta. Da bi se lakše razumjelo kako MaxPool2D radi, obojali smo zadanu matricu u 4 boje, zelena, narančasta, plava, ljubičasta. Primjetimo da su ta obojana područja zadane matrice veličine filtera (2x2). MaxPool2D radi tako da za svako područje jedne boje pronađe najveći element i njega zapiše u izlaznu matricu na odgovarajuće mjesto.

5.2.3 Dropout

Dropout operacija nasumično odabere čvorove koje će odbaciti prilikom svakog ciklusa ažuriranja težine. Korisnici sami odabiru vjerojatnost odabiranja čvora. Ova operacija koristi se prilikom treniranja modela dok se prilikom testiranja zanemaruje.

Pretpostavimo da imamo model kojem se događa overfitting. Ideja dropouta je da korisnik zada vjerojatnost izbacivanja čvora, npr. možemo zamisliti da imamo nekoliko čvorova u sloju i vjerojatnost izbacivanja čvora 50%. Možemo za svaki čvor bacati kocku i ako padne paran broj onda taj čvor brišemo (skupa sa svim svojim vezama).

5.2.4 BatchNormalization

Pretpostavimo da model treniramo na crno-bijelim slikama. Kada bi test podaci bili također crno-bijele slike, dobili bismo dobre rezultate. Problem nastaje kada bismo testirali rezultate na slikama u boji. Jedno od mogućih rješenja bilo bi trenirati isti model na slikama u boji. Drugo rješenje bilo bi koristiti *BatchNormalization* sloj.

On povećava stabilnost neuronske mreže tako da normalizira serije podataka koje dobijemo izlazom proteklog sloja. Ovaj sloj normalizira podatke tako da od rezultata prijašnjeg sloja oduzme srednju vrijednost te tu razliku podijeli sa standardnom devijacijom te serije podataka.

5.2.5 Flatten

Ova operacija se koristi u slučajevima kada imamo višedimenzionalne matrice. U slučaju da se u jednom trenutku pojavi potreba za računanjem sa jednodimenzionalnom matricom, koristit ćemo ovaj sloj.

5.2.6 Fully connected

Potpuno spojeni sloj spaja svaki ulazni podatak sa svakim izlaznim podatkom te između njih postoji težina brida. Programer bira koliko postoji različitih izlaznih vrijednosti. Potrebno je paziti na količinu izlaznih podataka pošto se ukupan broj veza računa kao umnožak broja ulaznih podataka i broja izlaznih podataka.

5.3 Aktivacijske funkcije

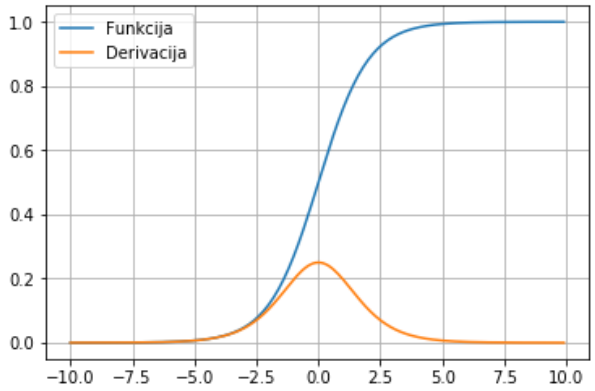
Svaki neuron dobiva podatke na ulaz. Te podatke množi sa pripadnim težinama i zbroji ih. Potreban je način da bismo dobili izlaz iz tog neurona. Upravo tome služe aktivacijske funkcije. Postoji mnogo aktivacijskih funkcija. Neke koje su se koristile u početcima neuronskih mreža zamijenjene su jednostavnijima. Ovdje ćemo opisati neke aktivacijske funkcije koje se često koriste danas.

5.3.1 Sigmoid

Sigmoid funkcija se koristi jako često kada nam je potrebna vjerojatnost da se jedan ulazni podatak može klasificirati kao određena klasa[7]. Znamo da je vjerojatnost realan broj između 0 i 1. Graf sigmoid funkcije izgleda kao slovo “S” kao na slici 6. Njezina domena su svi realni brojevi a kodomena realni brojevi izmedu 0 i 1 što možemo iskoristiti kao

vjerojatnost predviđanja. Ova funkcija je derivabilna što znači da možemo dobiti njezin nagib u svakoj točki.

```
def sigmoid(x):
   return 1.0/(1+np.exp(-x))

def d_sigmoid(x):
   return sigmoid(x) * (1-sigmoid(x))
```



Slika 10: Graf sigmoid funkcije i njezine derivacije

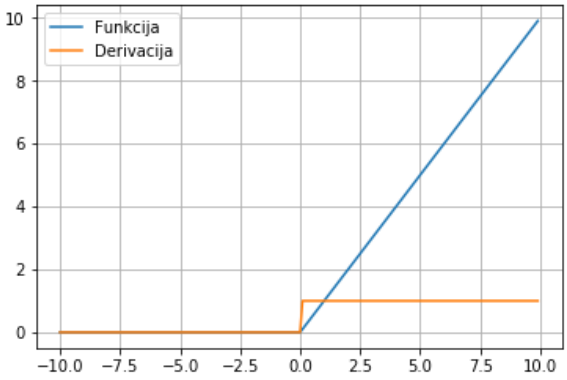
Iz grafa na slici 10 možemo vidjeti kako izgleda sigmoid funkcija (plava krivulja) i njezina derivacija (narančasta krivulja). Možemo vidjeti da je sigmoid funkcija monotona te da njezina derivacija nije. Zbog toga može nastati problem jer funkcija može zaglaviti neuronsku mrežu tijekom treninga.

5.3.2 ReLU

Ova aktivacijska funkcija je trenutno najkorištenija aktivacijska funkcija korištena za duboko učenje[8].

ReLU funkcija vraća 0 ako je ulaz manji od 0, inače vraća taj broj. Pogodna je za korištenje zbog brzine kojom se računa. Njezina derivacija je 0 za sva vrijednosti manje od 0, a 1 za sve vrijednosti veće od 0. Možemo primjetiti da je u točki 0 nemoguće izračunati derivaciju, ali se u praksi se vraća 0 radi lakšeg korištenja. Kodomena ReLU funkcije su svi pozitivni realni brojevi stoga se ne može koristiti za vjerojatnost predviđanja. Najveći problem ove aktivacijske funkcije je u tome što za negativne vrijednosti vraća 0 stoga ju se ne može koristiti za treniranje modela na negativnim vrijednostima.

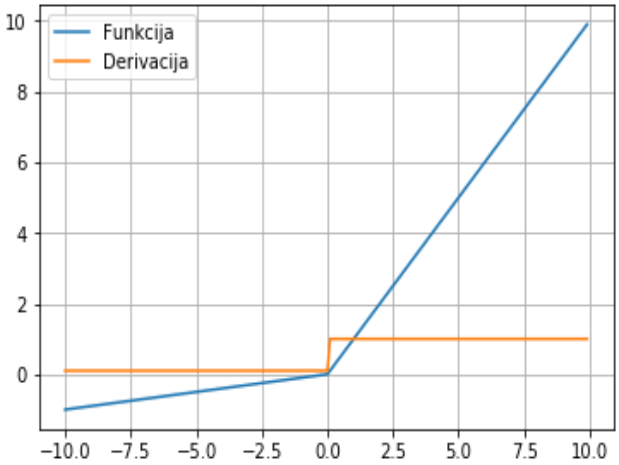
```
def relu(x):
    return max(0, x)

def d_relu(x):
    if(x == 0.0):
        return 0 #u ovoj točki je neodređeno ali stavljamo 0 zbog lakšeg računanja
    elif(x > 0):
        return 1
    else:
        return 0
```



Slika 11: Graf i derivacije ReLU funkcije

Rješenje za treniranje na negativnim vrijednostima je Leaky ReLU funkcija. Ona za pozitivne vrijednosti vraća vrijednost jedne funkcije a za negativne vrijednosti vraća vrijednost druge funkcije. Ideja za negativne vrijednosti bila je zamijeniti funkciju f(x)=0 gdje je x<0 sa funkcijom f(x) = ax gdje je x<0 i a se postavlja najčešće na vrijednost 0.01.

```
a = 0.1

def leaky_relu(x):
    if(x > 0):
        return x
    else:
        return a*x

def d_leaky_relu(x):
    if(x > 0):
        return 1
    else:
        return a
```



Slika 12: Graf i derivacija Leaky ReLU funkcije

Za primjer smo u napravili graf u kojem je parametar a=0.1 da bi se bolje vidio utjecaj.

5.3.3 Softmax

Kod problema klasifikacije prethodne funkcije ne mogu biti od pretjerane koristi. Softmax, kao i sigmoid funkcija, vraća brojeve između 0 i 1. Razlika izeđu njih dvije je u tome što softmax funkcija dijeli dobiveni rezultat tako da je zbroj svih izlaza jednak 1. Ova funkcija nam je neophodna za korištenje prilikom npr. klasificiranja znamenaka jer kaže za svaku sliku koja je vjerojatnost da ta slika prikazuje svaku od znamenaka.

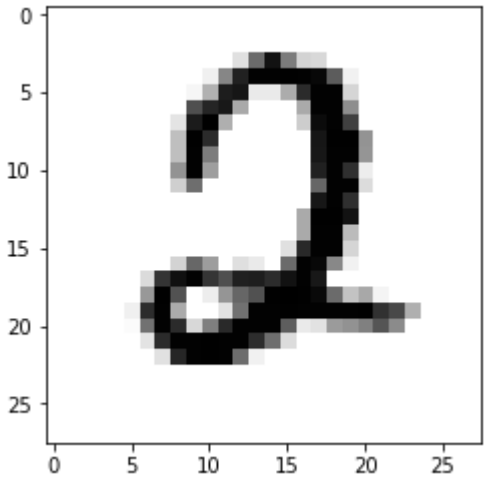
6 Klasifikacija znamenaka

Klasifikacija znamenaka tipičan je primjer single-label klasifikacije. MNIST[9] je baza ručno pisanih znamenaka koja broji 60000 trening podataka i 10000 test podataka. U upotrebu je puštena 1999. godine i otada se koristi kao jednostavan primjer za strojno učenje.

Kaggle[10] je platforma koja sadrži mnoštvo podataka iz različitih područja koji se koriste za proučavanja i izrađivanje modela za predviđanje. Platforma također drži natjecanja u kojima je cilj postići što bolju točnost. Jedno od takvih natjecanja, primarno namjenjeno učenju, je i zadatak klasifikacije znamenaka (engl. Digit recognizer). To natjecanje koristi dio MNIST-ove baze ručno pisanih znamenaka.

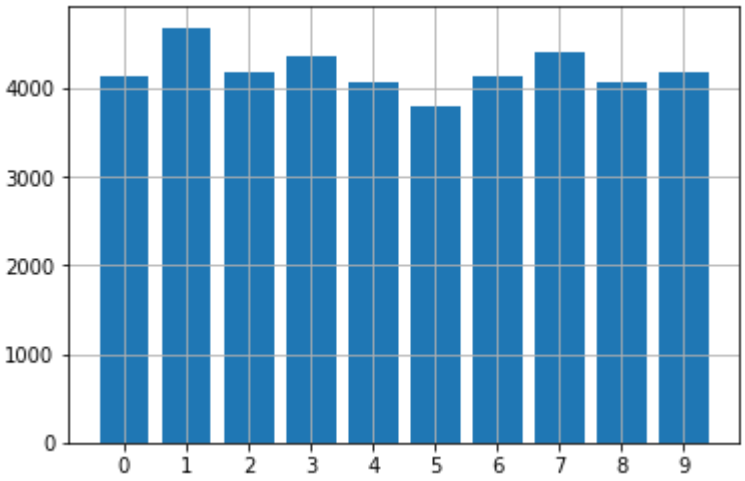
6.1 Podaci

Ulazni podaci za problem klasifikacije znamenaka su crno-bijele slike veličine 28px * 28px. Svaki piksel određen je brojem između 0 i 255, gdje 0 označava bijeli piksel, a 255 označava crni piksel.



Slika 13: Primjer znamenke

Podaci su podijeljeni tako da imamo 42000 slika za učenje modela i 28000 slika za testiranje modela. Na slici 13 možemo vidjeti jedan primjer kako izgleda znamenka 2 koju dobivamo kao ulazni podatak.



Slika 14: Zastupljenost znamenaka u skupu za učenje

Iz grafa na slici 14 možemo vidjeti kako je koja znamenka zastupljena u skupu za učenje. Vidimo da su sve znamenke podjednako zastupljene.

Skup podataka za učenje modela podijelili smo na 2 skupa u omjeru 80%:20%. Na prvom skupu ćemo učiti model, a drugi skup će služiti za validaciju modela. Skup podataka za testiranje ne možemo koristiti kao validacijske podatke jer rješenja ne vidimo.

6.2 Arhitektura mreže i učenje modela

Mreža se sastoji od 1 ulaznog sloja, 1 izlaznog sloja i 7 skrivenih slojeva. Ulazni sloj je dimenzije (28, 28, 1) što je veličina jedne slike. Zadnji broj nam govori koliko ima kanala slika. Crno-bijela slika ima 1 kanal, a slika u boji ima 3 kanala. Izlazni sloj ima 10 izlaza. Za svaku moguću znamenku 1 izlaz koji govori kolika je vjerojatnost da je na zadanoj slici određena znamenka.

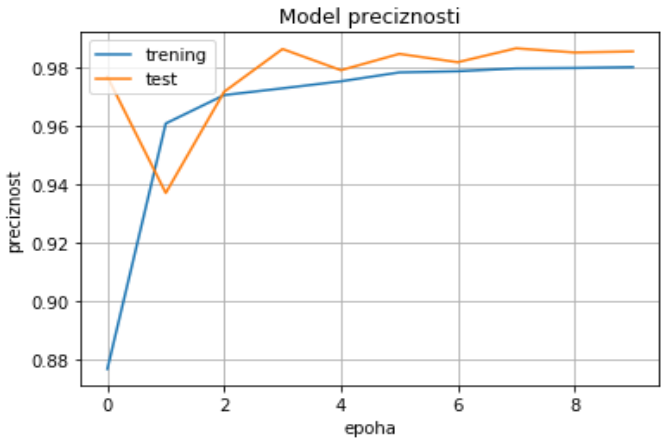
Mreža se sastoje od 4 *Conv2D()* sloja sa *MaxPool2D* operacijom *i 4 Dense()* sloja. U mreži se još koriste operacije *BatchNormalization(), Flatten() i Dropout()*.

```
model = Sequential()

model.add(Conv2D(16, (5, 5), activation='relu', input_shape=(image_width, image_height, channels)))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(32, (5, 5), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(BatchNormalization(axis=-1))

model.add(Conv2D(64, (2, 2), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(BatchNormalization(axis=-1))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10,activation='softmax'))
```

Kao optimizacijsku funkciju koristili smo funkciju *RMSprop()* sa parametrom za učenje 1e-3. Odabrali smo funkciju *fit_generator()* za učenje modela sa 10 prolaza kroz sve slike znamenaka u skupu za učenje (10 epoha).

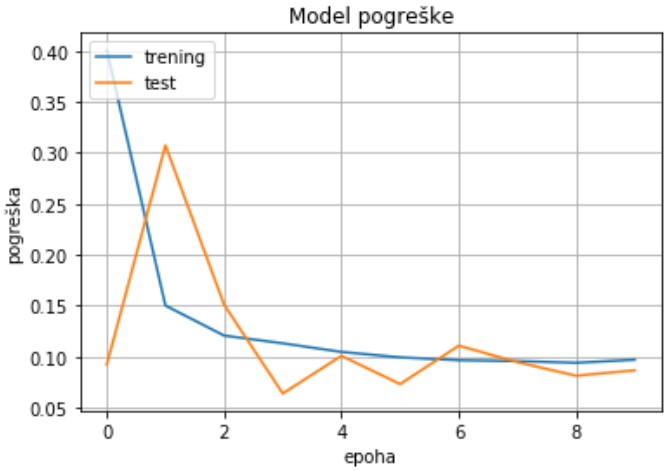
6.3 Analiza rezultata

Ovaj model dao je odlične rezultate. Model je naučio skup za učenje sa 98% točnosti, a kod validacijskog dijela dobili smo 98,5% nakon 10 epoha.



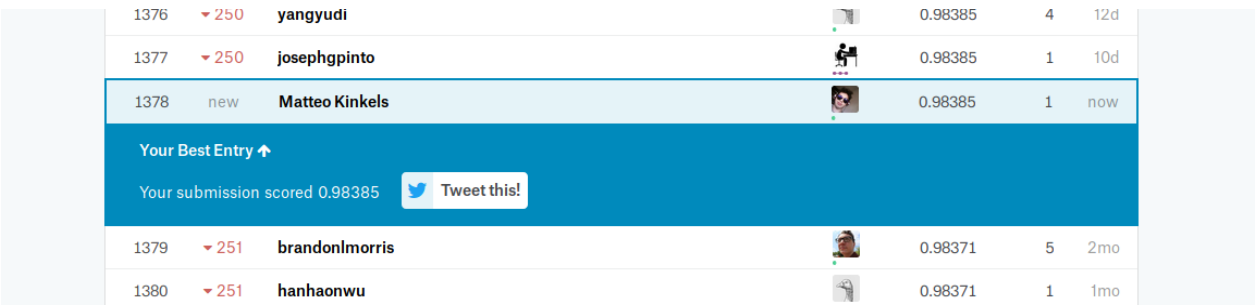
Slika 15: Model preciznosti

Graf na slici 15 prikazuje točnost modela nakon svake epohe. Iz grafa možemo zaključiti da model radi dobro te se ne događa overfitting. To također možemo potvrditi grafom na slici 16.



Slika 16: Model pogreške

Preostalo je jedino iz dobivenih test podataka generirati rješenja te poslati na Kaggle-ovu platformu. Naš rezultat iznosi 98,385% kao što možemo vidjeti na slici 17.



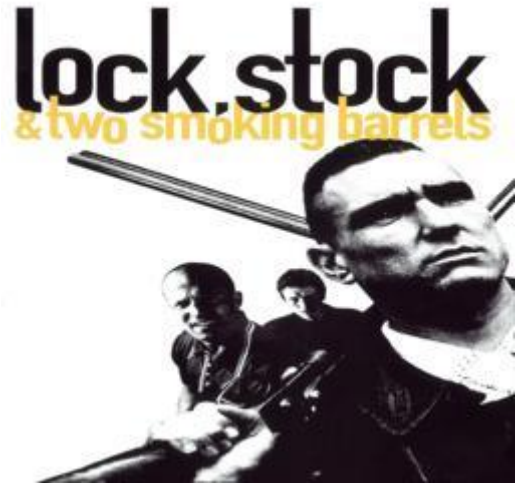
Slika 17: Rezultat koji smo dobili na Kaggle-u

7 Klasifikacija filmskih postera

Problem klasificiranja filmova u žanrove je klasični primjer multi-label klasifikacije. Prikazat ćemo dva načina za rješavanje problema te opisati njihove prednosti i nedostatke.

7.1 Podaci

Ulazni podatci za izradu ovog projekta su slike filmskih postera i njihova klasifikacija. Imena slika zapisana su u datoteci pod nazivom *imena.txt,* a njihove klasifikacije zapisane su u datoteci pod nazivom *klase.csv*.



Slika 18: Klasificira se kao komedija i kriminalistički film

Slika 19: Klasificira se kao komedija

Za unos podataka koristili smo biblioteku pandas. Ova iznimno moćna biblioteka omogućuje brzu i fleksibilnu analizu i manipulaciju podacima.

```
X = pd.read_csv('imena.txt', sep="\n", header=None)
X.columns = ["Ime datoteke"]

Y = pd.read_csv('klase.csv')
```

U varijablu *X* spremamo imena slikovnih datoteka, a u varijablu *Y* njihove klasifikacije. U kodu ćemo kao alias za pandas koristiti pd. Pandas za učitavanje koristi funkciju *read_csv()*. Kod učitavanja *.csv* datoteka, jedini parametar koji moramo poslati je ime datoteke. Što se tiče učitavanja ostalih vrsta datoteka, potrebno je navesti separator i zaglavlje.

7.1.1 Analiza podataka

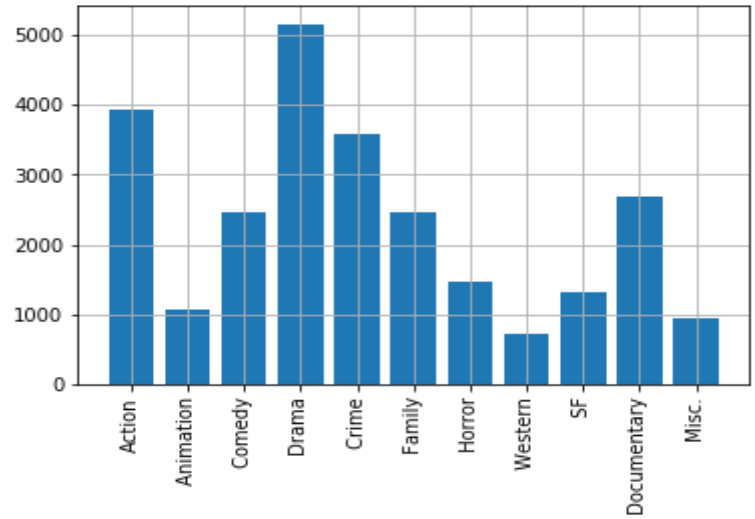
Sumiranjem svih stupaca dobit ćemo ukupan broj filmova koji pripadaju svakoj od kategorija.

```
Y.sum()

Action          3918
Animation       1057
Comedy          2474
Drama           5153
Crime           3572
Family          2462
Horror          1454
Western          719
SF              1329
Documentary     2691
Misc.           948
dtype: int64
```

Možemo vidjeti da imamo jedanaest filmskih žanrova. To su akcija, animirani film, komedija, drama, kriminalistički film, obiteljski film, horor, vestern, znanstvena fantastika, dokumentarni film i ostalo.

Iz podataka primjećujemo da žanrovi nisu ravnomjerno raspoređeni. To možemo najbolje vidjeti na stupičastom grafu. Sve grafove ćemo raditi pomoću biblioteke *matplotlib*[11].

```
plt.bar(range(11), list(Y.sum()))
plt.xticks(range(11), Y.columns, rotation='vertical')
plt.grid(True)
```

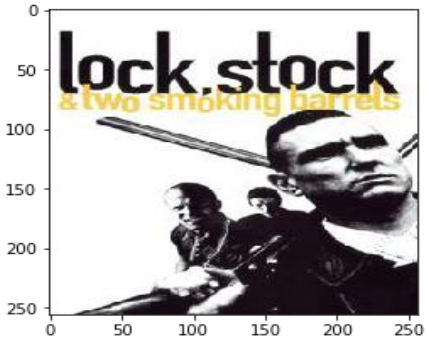


Slika 20: Distribucija slika postera po kategoriji

Iz grafikona na slici 20 vidimo da u podacima ima najviše slika filmskih postera koje pripradaju žanru drama, a slijede ga akcijski film, kriminalistički film te dokumentarni film. Nebalansiranost podataka se vidi iz činjenice da su posteri koji pripadaju dokumentarnim filmovima četvrti po brojnosti, a ima ih upola manje nego drama.

Ukupan broj filmskih postera je 10979. Svaki poster je u *.jpg* formatu. Svaka slika normalizirana je na visinu od 256 piksela i širinu od također 256 piksela. Biblioteka *matplotlib* nam osim izrade grafikona omogućuje i ispis slike na ekran. Ispisat ćemo jednu sliku za primjer.

```
image = Image.open('posteri/100.jpg')
plt.imshow(image)
```



Slika 21: Primjer slike filmskog postera

7.2 Podjela podataka na skup za učenje i skup za testiranje

Sve učitane podatke razvrstat ćemo u skup za učenje i skup za testiranje i to u omjeru 80%:20%. Skup za učenje služi za učenje modela, a skup testiranje za testiranje. To se jednostavno postiže funkcijom *train_test_split()* koja se nalazi u biblioteci *sklearn*[12].

```
X_train, X_test, Y_train, Y_test = train_test_split(
    X,
    Y,
    test_size=train_test_size,
    random_state=random_seed)
```

Funkcija prima 4 parametra: učitane X vrijednosti, učitane Y vrijednosti, veličinu testnog skupa i random state. X vrijednosti su imena slikovnih datoteka, a Y vrijednosti su vrijednosti njihovih klasifikacija. Random state uvijek postavljamo na isti broj tako da prilikom višestrukog ponavljanja koda uvijek dobijemo iste rezultate. Funkcija vraća 4 vrijednosti:

trening skup ulaznih vrijednosti, testni skup ulaznih vrijednosti, trening skup izlaznih vrijednosti i testni skup izlaznih vrijednosti.

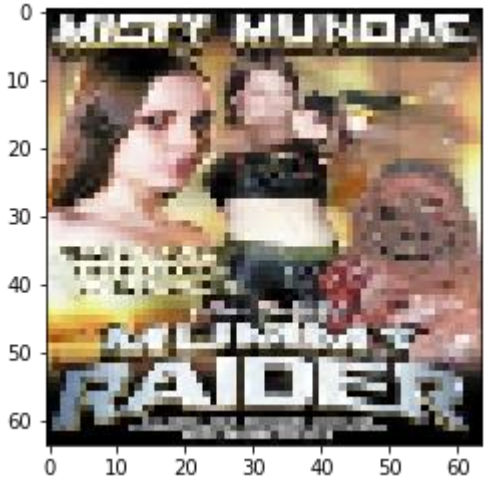
Na početku smo spomenuli da ćemo u ovom radu prikazati 2 načina rješavanja zadatka. U prvom načinu (u daljnjem tekstu: način A), nakon razvrstavanja podataka, stvaramo 2 direktorija pod imenom *train* i *test.* U svaku od njih potrebno je stvoriti 11 poddirektorija koje su imenove nazivima žanrova te u njih prebacimo slike postera kako im pripadaju. Kako jedan poster može pripadati većem broju žanrova, potrebno je primijetiti da se jedna slika postera može nalaziti u više direktorija.

```
os.mkdir('test')
os.mkdir('train')
for zanr in Y.columns:
    os.mkdir('train/'+str(zanr))
    os.mkdir('test/'+str(zanr))

def move(name, categories, folder):

    for i in range(len(categories)):
        if(categories[i]==1):
            os.system('cp posteri/'+str(name)+' '+folder+'/'+str.strip(Y.columns[i])
+'/'+str.strip(Y.columns[i])+'_'+str(name))

n = len(X_train)
for i in range(n):
    move(X_train.iloc[i].values[0], Y_train.iloc[i].values, 'train')

m = len(X_test)
for i in range(m):
    move(X_test.iloc[i].values[0], Y_test.iloc[i].values, 'test')
```

```
def imageToArray(name):

    image = Image.open('posteri/'+name)
    image = image.resize((image_width, image_height))
    image = np.array(image) / 255

    return image

def getInputX(X, Y):

    amount = Y.sum()

    x = np.zeros((amount*2, image_width, image_height, channels))
    cnt = 0

    for idx in range(len(X)):

        if(Y.iloc[idx] == 1):

            x[cnt] = imageToArray(X.iloc[idx].values[0])
            cnt += 1

        elif(amount > 0):

            x[cnt] = imageToArray(X.iloc[idx].values[0])

            amount -= 1
            cnt += 1

    return x

def getInputY(X, Y):

    amount = Y.sum()

    y = []

    for idx in range(len(Y)):

        if(Y.iloc[idx] == 1):

            y.append(1)

        elif(amount > 0):

            y.append(0)
            amount -= 1

    return y
```

Drugi način rješavanja problema je promijeniti veličinu slika tako da ih možemo zapisati u niz (u daljnjem tekstu: način B). Odlučili smo smanjiti slike na visinu 64 piksela i širinu 64 piksela. Također, odlučili smo ravnomjerno rasporediti broj slika po žanru. To smo napravili tako da smo nakon razdvajanja podataka izračunali za svaki žanr koliko mu slika pripada te smo u trening skup uzeli sve slike za taj žanr i jednako toliko slika koje nisu tog žanra. Kasnije ćemo pojasniti zbog čega je ovo bolja metoda.

Također, prednost kod ovog načina rješavanja je to što možemo odmah normalizirati podatke tj. slike. Kao što znamo, slike u *.jpg* formatu su zapisane tako da svaki piksel zapišemo u RGB obliku. Taj oblik nam za svaku od 3 boje zapiše intenzitet te boje. Intenzitet je broj između 0 i 255. Kako bi nakon nekoliko množenja rezultat bio jako velik i računanje bi bilo sve sporije, slike se normaliziraju. To radimo tako da brojeve između 0 i 255 pretvorimo u brojeve između 0 i 1 kao u funkciji *imageToArray()*. Još jedna pozitivna strana normaliziranja je to da će slične boje imati manju apsolutnu razliku stoga bi model mogao bolje učiti.



Slika 22: Slika filmskog postera kojoj smo smanjili dimenziju

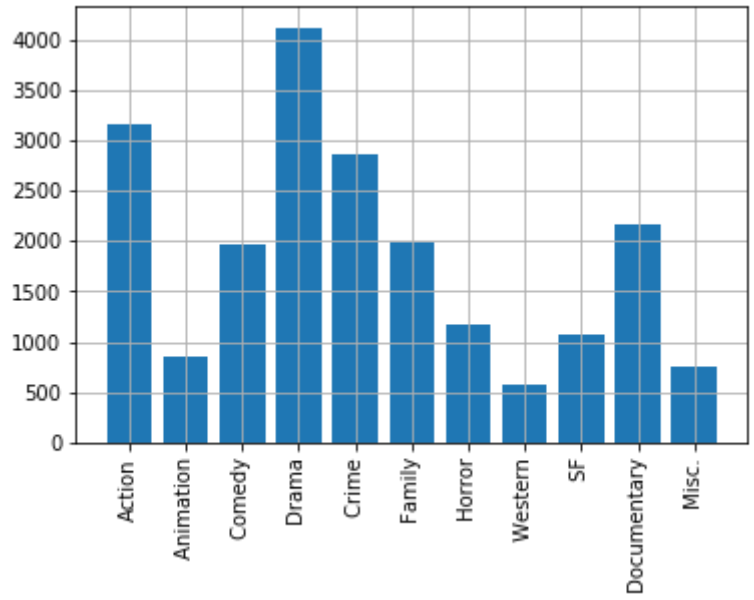
7.2.1 Analiza skupa za učenje

Ranije smo napravili analizu svih dobivenih podataka. Sada ćemo napraviti analizu podataka koje ćemo koristili samo za treniranje modela. Skup podataka za treniranje modela čini 80% podataka iz cijelog skupa ulaznih podataka. Kao i ranije, možemo jednostavnim korištenjem *pandasa* dobiti brojčane vrijednosti količine slika po žanrovima. Potrebno je napomenuti da se ova analiza radi samo za A način rješavanja pošto je B način bitno drukčiji.

```
Y_train.sum()

Action         3151
Animation       844
Comedy         1966
Drama          4114
Crime          2862
Family         1988
Horror         1171
Western         567
SF             1077
Documentary    2156
Misc.           754
dtype: int64
```

Očekivano opet vidimo nebalansiranost podataka. Stupičastim grafom ćemo to bolje vidjeti.

```
plt.bar(range(11), list(Y_train.sum()))
plt.xticks(range(11), Y.columns, rotation='vertical')
plt.grid(True)
```



Slika 23: Distribucija slika filmskih postera u trening skupu podataka

Iz grafikona na slici 23 vidimo da su podaci ostali u istom omjeru kao i kod analize svih danih podataka. Ova analiza je bitna iz razloga da uočimo postoji li neki žanr koji sadrži sve slike i postoji li neki žanr koji ne sadrži nijednu sliku jer bez podataka ne možemo naučiti model. Vidimo da vesterna ima najmanje ali ima više od 5 stotina slika koje pripadaju tom žanru i nisu sve slike klasificirane kao drama stoga je skup u redu za učenje modela..

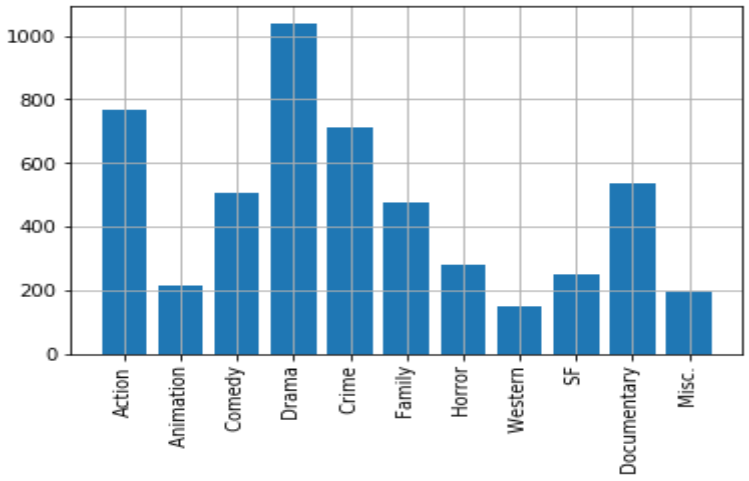
7.2.2 Analiza skupa za testiranje

I ova analiza odnosi se na A način podjele podataka u skupu za testiranje. Kako je kod podjele podataka cilj zadržati jednak omjer podataka u skupu za učenje i u skupu za testiranje, pokazati ćemo da je distribucija slika po žanru u skupu za testiranje ostao isti kao u skupu za učenje. Ovdje uzimamo u obzir preostalih 20% podataka koje nismo obradili u analizi podataka za učenje.

```
Y_test.sum()

Action          767
Animation       213
Comedy          508
Drama          1039
Crime           710
Family          474
Horror          283
Western         152
SF              252
Documentary     535
Misc.           194
dtype: int64
```

Grafički ćemo pokazati da je omjer ostao isti.

```
plt.bar(range(11), list(Y_test.sum()))
plt.xticks(range(11), Y.columns, rotation='vertical')
plt.grid(True)
```



Slika 24: Distribucija slika filmskih postera u testnom skupu podataka

Bitno je primjetiti da su svi žanrovi zastupljeni te da nisu sve slike klasificirane kao jedan žanr.

7.3 Arhitekture TensorFlow mreže

Za problem klasifikacije postera usporedili smo dvije varijante arhitekture TensorFlow mreže. Arhitekture mreža sastoje se od 8 *Conv2D() slojeva i 3 Dense()* sloja. Koriste se operacije *MaxPool2D(), Dropout(), BatchNormalization() i Flatten()*. Mreža se sastoji od 1 ulaznog sloja, 10 skrivenih slojeva i izlaznog sloja. Njihova funckija je objašnjena u poglavlju koje se tiče rada TensorFlow-a. Modeli koje smo koristili razliku se jedino u izlaznom sloju.

```
model = Sequential()

model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(img_width, img_height, channels)))
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(BatchNormalization(axis=-1))

model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(BatchNormalization(axis=-1))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(11,activation='sigmoid'))
```

Iz modela možemo vidjeti da ulazni sloj čini slika dimenzija 256x256 sa 3 kanala (crveni, zeleni i plavi). Za izlazni sloj koristimo potpuno spojeni sloj sa 11 izlaza. Svaki izlaz je neovisan o ostalim izlazima. Sigmoid funkcijom računamo vrijednost izlaza. Ona je povoljna zbog toga što ta funkcija za kodomenu ima vrijednost između 0 i 1.

Slično smo napravili model za B način. Razlika je u tome da ovdje umjesto da radimo jedan model koji će moći sliku klasificirati u 11 žanrova, radimo 11 identičnih modela koji će imati samo jedan izlaz, a to je pripada li slika tom žanru ili ne.

```
def Layers():


    model = Sequential()

    model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(image_width, image_height, channels)))
    model.add(Conv2D(16, (3, 3), activation='relu'))
    model.add(Conv2D(16, (3, 3), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))

    model.add(Conv2D(16, (3, 3), activation='relu'))
    model.add(Conv2D(16, (3, 3), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))

    model.add(BatchNormalization(axis=-1))

    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))

    model.add(BatchNormalization(axis=-1))

    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1,activation='sigmoid'))

    return model

models = {}
for x in Y.columns:
    models[x] = Layers()
```

Iz ova dva modela možemo vidjeti dva različita načina rješavanja multi-label klasifikacije. Prvo rješenje, za način A, pokušava naučiti kako klasificirati jednu sliku u svih 11 žanrova. Drugo rješenje, za način B, oslanja se na to da multi-label klasifikaciju pretvorimo u više single-label klasifikacija.

7.4 Učenje modela

Nakon definiranja modela, potrebno je konfigurirati proces učenja. Za optimizacijsku funkciju često se koriste *adam, RMSprop* i *SGD.* Isprobali smo sve tri. *Adam* je definitivno izbačen kao jedna od mogućnosti za naš model pošto iznimno brzo konvergira. što je dobro, međutim, loše je to što konvergira u lokalni optimum. *SGD* je također pokazivao isti

problem, ali je konvergirao malo sporije. Preostala je funkcija *RMSprop* koja je radila dobar posao kada je stopa učenja bila blizu 1e-5. Kada smo stavili manju stopu učenja, model je konvergirao kao i prijašnje optimizacijske funkcije, dok je kod manjih vrijednosti bilo potrebno više epoha za dobivanje iste točnosti.

```
opt = RMSprop(lr=1e-5)
for x in models:
   models[x].compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
```

Za loss funkciju odabrali smo *binary_crossentropy*, koja se i inače koristi za problem multi-label klasifikacije. Proučavanjem tuđih grešaka uočili smo da se većina grešaka događala upravo zbog biranja slične funkcije, *categorical_crossentropy*. Razlika je u tome što *categorical_crossentropy* daje izlaz kod izlaznog sloja tako da zbroj izlaza bude jednak jedan (uz korištenje aktivacijske funkcije *softmax*). U ovom problemu to ne možemo upotrijebiti. Primjer za korištenje te funkcije bio bi klasifikacija znamenaka što ćemo spomenuti kasnije. Dakle, iz očitih razloga, odlučili smo koristiti funkciju *binary_crossentropy*.

Prije je bilo spomenuto da smo za A način premjestili slike u direktorije. Ovdje ćemo sada vidjeti zašto.

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

train_generator = train_datagen.flow_from_directory(
        'train',
        target_size=(256, 256),
        batch_size=batch_size)

test_generator = test_datagen.flow_from_directory(
        'test',
        target_size=(256, 256),
        batch_size=batch_size)
```

Keras nam omogućuje funkcijom *flow_from_directory* dohvaćanje slika tek kada su one potrebne. Također, upotrijebili smo funkciju *ImageDataGenerator*[13] kako bi normalizirali slike (razlog opisan ranije). Ova funkcija se koristi kako bi se povećao skup podataka za učenje. To se čini tako da se modificiraju i transformiraju slike. Neke od mogućnosti su npr. rotiranje, zrcaljenje, povećavanje i smanjivanje slika. Svaka od opcija koju odaberemo za rad nad slikom se događa slučajnim odabirom. To bi značilo da u 2 različite epohe ista slika

neće biti ista, npr. u prvom prolasku će biti povećana za 10% i rotirana 15° ulijevo, dok će u drugoj epohi biti smanjena za 2% i biti rotirana udesno 26%. Samim time što se smanjuje šansa da će se model više puta učiti na jednoj slici, ova funkcija smanjuje mogućnost overfittinga.

U našem slučaju, nismo dobili bolji rezultat korištenjem tih opcija pa smo ih maknuli jer je modelu potrebno mnogo više vremena za obradu.

Za način A odabrali smo funkciju *fit_generator* za učenje modela i to u 50 prolaza kroz sve postere u skupu za učenje (50 epoha) kako bi mogli bolje naučiti model. U odnosu na funkciju fit(), funkcija fit_generator() kada imamo veliki skup ulaznih podataka[14]. Pošto smo kao generator slika koristili funkciju *ImageDataGenerator(),* kao generator, možemo koristiti funkciju *flow_from_directory()* za dohvaćanje slika. Ta funkcija dohvaća slike u njihovim direktorijima te po njima određuju kako ih klasificirati. Iz tog razloga smo na početku slike rasporedili po direktorijima koje smo nazvali po nazivima filmskih žanrova.

```
history = model.fit_generator(
        train_generator,
        steps_per_epoch=n_train // batch_size,
        epochs=50,
        validation_data=test_generator,
        validation_steps = n_test // batch_size,
)
```

Kod B načina rješavanja problema, morali smo koristiti funkciju *fit* za učenje. Ulazne podatke smo dobili funkcijama *getInputX* i *getInputY* koje smo definirali ranije. Pošto je model brže konvergirao kako imamo podjednak broj slika koje pripadaju određenom žanru i slika koje ne pripadaju žanru, odlučili smo broj epoha ograničiti na 10 iako u većini slučajeva rezultat počinje konvergirati puno prije.

```
for genre in models:

    x = getInputX(X_train, Y_train[genre])
    y = getInputY(X_train, Y_train[genre])

    print( "--------------------------------------------------------------------------TRAINING FOR "+str(genre) )
    models[genre].fit(x, y,
          batch_size=16,
          epochs=10)
```

Obje funkcije nam ispisuju parametre kako bi lakše mogli pratiti uspješnost učenja modela. Neki od podataka su redni broj epohe, vrijeme potrebno za učenje, vrijeme potrebno za obraditi jednu sliku, pogreška i točnost.

```
--------------------------------------------------------------------------TRAINING FOR Action
Epoch 1/10
7048/7048 [==============================] - 14s 2ms/step - loss: 0.9814 - acc: 0.4989
Epoch 2/10
7048/7048 [==============================] - 8s 1ms/step - loss: 0.9630 - acc: 0.4972
Epoch 3/10
7048/7048 [==============================] - 8s 1ms/step - loss: 0.9478 - acc: 0.4963
Epoch 4/10
7048/7048 [==============================] - 8s 1ms/step - loss: 0.9402 - acc: 0.5024
```

7.5 Testiranje modela

Model smo učili samo na skupu za učenje, skup za testirenje nismo gledali (u B načinu smo ga koristili samo da vidimo točnost za svaki žanr). Funkcijom *predict()* dobit ćemo rezultate klasifikacije i usporediti ih sa stvarnim podacima u skupu kako bismo mogli izračunati uspješnost klasifikacije.

```
x = misc.imread('posteri/'+str(X_test.iloc[index].values[0]))
x = x/255
x = np.reshape(x,[-1,img_width,img_height,3])

y = model.predict(x)
```

U kodu iznad prikazan je jedan način korištenja funkcije *predict()*.

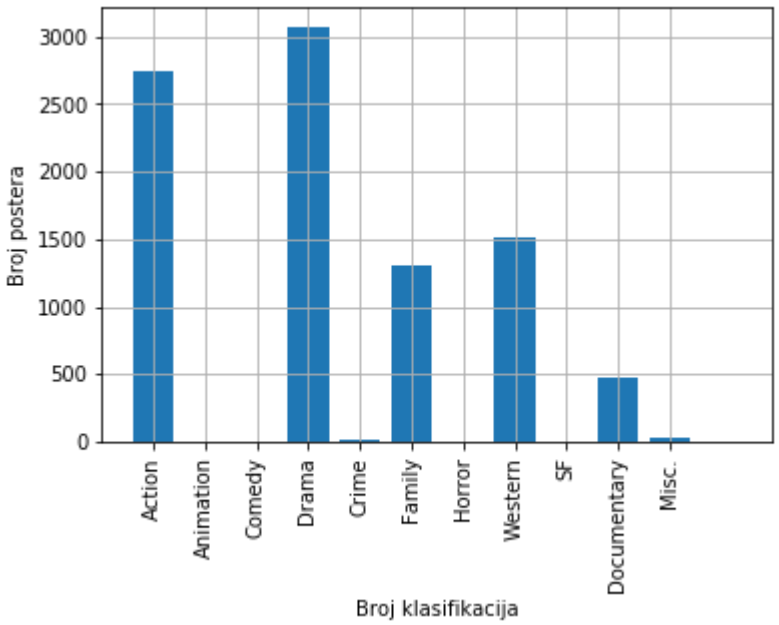
U B načinu bilo je potrebnan još jedan korak kako bi se dobio multi-label rezultat klasifikacije. Naime, kako smo raspodijelili podatke u 11 zasebnih kategorija, naučili smo model za svaku od kategorija posebno pa je na kraju potrebno spojiti sva dobivena rješenja.

```
X_valid3d = np.zeros((len(X_valid),image_width, image_height, channels))
for i in range(len(X_valid)):
    image = imageToArray(X_valid.iloc[i].values[0])
    X_valid3d[i] = image.reshape((image_width, image_height, channels))

p_valid = {}
for x in Y.columns:
    p_valid[x] = models[x].predict(X_valid3d, batch_size=128)

prediction = pd.DataFrame.from_dict({
        genre: list(models[genre].predict(X_valid3d, batch_size=128)) for genre in Y.columns

})

prediction = prediction.astype(np.float64)
prediction[prediction>=0.5] = 1
prediction[prediction<0.5] = 0
prediction = prediction.astype(np.int)
```

7.6 Analiza dobivenih podataka

Za računanje točnosti dobivenih podataka, koristili smo se funkcijom *fbeta_score* koja se nalazi u biblioteci *sklearn*.

```
fbeta_score(Y_valid, prediction, beta=2, average='samples')
```



Slika 25: Frekvencija rezultata klasifikacije A načinom rješavanja

A načinom rješavanja problema, postigli smo preciznost od 26,13%. Kao što možemo vidjeti na slici 25 model većinu slike prepoznaje kao drame i akcije pošto su oni najzastupljeniji.

Distribuciju klasificiranih slika po žanrovima možemo vidjeti u grafu na slici 25. Rezultati ovog načina rješavanja nisu dobri zbog velike razlike u zastupljenosti pojedinih žanrova.

Što se tiče B načina rješavanja, uspjeli smo donekle riješiti problem velike razlike u zastupljenosti pojedinih žanrova. Najbolji rezultat koji smo postigli ovim načinom rješavanja problema je bio 47,05%.

Žanr	Točnost
Action	36,25%
Animation	10,92%
Comedy	77,14%
Drama	50,55%
Crime	33,24%
Family	35,61%
Horror	75,05%
Western	35,61%
SF	64,75%
Documentary	42,99%
Misc.	89,97%

U tablici možemo vidjeti točnost za svaki žanr posebno. Modelima smo pokušali povećavati i smanjivati broj konvolucijskih slojeva. Kod smanjivanja smo zabilježili smanjenje točnosti od 2%, dok smo kod povećavanja broj konvolucijskih slojeva zabilježili otprilike jednake rezultate uz veće vrijeme izvršavanja programskog koda. Najmanji broj konvolucijskih slojeva bio je 6, dok je najveći broj bio 10.

7.7 Moguća poboljšanja

Kao što smo mogli vidjeti, B način rješavanja je dobio skoro duplo bolje rezultate. Stoga, smatramo da bi modeli postigli bolje rezultate kada bi za svaki žanr imali približno jednak broj slika filmskih postera.

Drugi način za poboljšanje rezultata bio bi srodne žanrove spojiti u jedan zajednički žanr pa time na istom skupu podataka dobiti manje klasifikacija.

8 Softver i hardver korišten u izradi

Od softvera, u ovom radu, korišteni su Anaconda[15], Jupyter notebook[16], TensorFlow[17] i Keras[18] te operativni sustav Ubuntu Linux 18.10[19].

Anaconda je besplatna open-source softver koji se često koristi za izradu programa strojnog i dubokog učenja. Prednosti su mu što se za izradu koda koristi Python i R koji su u današnje vrijeme programski jezici koji se najviše koriste kod strojnog i dubokog učenja. Također, omogućuje nam izradu okruženja za svaki projekt posebno. Prednost toga je što za svaki projekt možemo instalirati samo pakete koji taj projekt zahtjeva. Trenutno broji preko 6 milijuna korisnika te se može koristiti na Windows-u, Linuxu i MacOS-u.

Jupyter notebook dolazi automatski instalacijom Anaconde. Ono što ga izdvaja od ostalih IDE-a je to što može dijeliti kod po ćelijama pa tako umjesto da svaki puta iznova pokrećemo cijeli kod, Jupyter notebook nam dozvoljava pokretanje koda ćeliju po ćeliju. Takvo korištenje omogućuje nam lakše uočavanje i ispravljanje grešaka. Još jedna prednost mu je što se program otvara u Web pregledniku pa tako ne moramo kodirati spremaje grafova i slika nego ih možemo preuzeti kao i svaku drugu sliku na Web-u.

TensorFlow je framework korišten za strojno i duboko učenje. U ovom radu, koristili smo **Keras** koji pokreće TensorFlow u pozadini te mu on izvršava kalkulacije. Valja napomenuti da se u ovom radu koristi TensorFlow koji za računanje koristi grafičku karticu (razlog će biti obrazložen kasnije). Keras je API za korištenje kompleksnih neuralnih mreža. U svojoj pozadini može koristiti TensorFlow, Theano i CNTK. Za razliku od čistog TensorFlow-a, omogućuje nam pisanje čišćeg koda te dolazak do rezultata uz najmanje uloženog truda.

Od hardvera, u ovom radu, korišteni su:

- procesor: AMD HexaCore Ryzen 5 1600 (3.2GHz)
- grafička kartica: Nvidia GeForce GTX 1060 6GB
- memorija: 16GB DDR4, 2400 MHz
- SSD 120GB

8.1 CPU vs GPU

TensorFlow omogućuje obradu podataka na procesoru i na grafičkoj kartici. Za korištenje na procesoru potrebno je instalirati TensorFlow koji se može koristiti samo na procesoru.

Ova verzija je znatno lakša za instalirati te njezina instalacija kraće traje. Pogodna je za početnike i uglavnom se može koristiti kada imamo mali skup podataka.

Za obradu podataka na grafičkoj kartici potrebno malo duže vrijeme za konfiguraciju. Prva stvar na koju moramo paziti je da imamo grafičku karticu koja podržava CUDA-u[20]. Dakle, moramo nabaviti NVIDIA grafičku karticu. Nakon toga moramo imati instaliran upravljački program za nju (*engl. driver*). Kako smo projekt radili na operativnom sustavu Ubuntu Linux, bilo je moguće instalirati TensorFlow-gpu preko komandne linije (nismo isprobali za operativni sustav Windows). Obavezno se mora specificirati instalacija TensorFlow-gpu jer inače će biti instaliran TensorFlow koji koristi procesor za obradu podataka. Sav ostali softver će se instalirati sam (CUDA i ostalo).

Što se tiče brzine obrade podataka, TensorFlow će na grafičkoj kartici nekoliko stotina puta brže obavljati računske operacije nego na procesoru. To se događa zbog mogućnosti paralelnog programiranja pomoću CUDA-e. Bitno je napomenuti da se programski kod neće morati mijenjati za obradu na grafičkoj kartici. Postoje slučajevi kada TensorFlow brže završi sa radom prilikom korištenja procesora, međutim ti slučajevi su rijetki i događaju se jedino kada imamo jako male ulazne podatke. To se događa zbog vremena koje je potrebno da se podaci prebace na memoriju grafičke kartice.

Pošto u danom problemu imamo veliku količinu podataka, za izradu rješenja koristit ćemo TensorFlow koji koristi grafičku karticu za obradu podataka.

8.2 Keras

Keras je Python biblioteka otvorenog koda koja u svojoj pozadini može koristiti TensorFlow, CNTK, Theano ili MXNet. U izradi našeg rješenja, koristit ćemo TensorFlow u pozadini. Fokus Kerasa je na razumljivosti koda, modularnosti i fleksibilnosti. Stvorio ga je Google-ov inženjer Francois Chollet. 2017. godine, Google je odlučio podržati razvoj Kerasa kao temeljnu TensorFlow biblioteku. Keras je u studenom 2017. imao više od 200,000 korisnika.

9 Zaključak

Rješavanjem problema klasifikacije znamenaka i filmskih žanrova pomoću slika postera dobio sam uvid u mogućnosti TensorFlowa kao biblioteke za izradu modela neuronske mreže. Prije izrade rješenja bilo je potrebno temeljito proučiti dokumentaciju i teoriju. Također, za rješavanje problema bila je potrebna velika količina predznanja iz programiranja općenito, programskog jezika Python i matematike, a kao pomoć u razumijevanju neuronskih mreža bilo je potrebno osnovno znanje o dijelovima i radu živčanih stanica.

Umjetna inteligencija me počela zanimati tek nedavno. Glavni razlog je bio u tome što nam ovakav način programiranja i razmišljanja daje neograničene mogućnosti stvaranja sustava koji nam mogu pomoći u izvršavanju različitih svakodnevnih zadataka. Ovaj rad mi je bio prvi doticaj sa ovakvim načinom programiranja do sata. Iznimno sam zadovoljan sa dobivenim rezultatima i smatram da bih u budućnosti, kada naučim još metoda za klasificiranje slika, mogao poboljšati ovaj model.

10 Literatura i izvori

1. https://en.wikipedia.org/wiki/Pattern_recognition
2. https://en.wikipedia.org/wiki/Deep_learning
3. https://keras.io/applications/
4. https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/
5. https://en.wikipedia.org/wiki/TensorFlow
6. http://adventuresinmachinelearning.com/keras-tutorial-cnn-11-lines/
7. http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/
8. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
9. http://yann.lecun.com/exdb/mnist/
10. https://www.kaggle.com/
11. https://matplotlib.org/
12. http://scikit-learn.org/stable/index.html
13. https://machinelearningmastery.com/image-augmentation-deep-learning-keras/
14. https://towardsdatascience.com/keras-a-thing-you-should-know-about-keras-if-you-plan-to-train-a-deep-learning-model-on-a-large-fdd63ce66bd2
15. https://anaconda.org/anaconda/python
16. http://jupyter.org/
17. https://www.tensorflow.org/
18. https://keras.io/
19. https://www.ubuntu.com/
20. https://www.tensorflow.org/install/install_linux
21. https://en.wikipedia.org/wiki/Keras
22. http://ferko.fer.hr/ferko/EPortfolio!dlFile.action;jsessionid=42FAF2BB2A7E915516D462A818D598F3?id=350
23. http://www.zemris.fer.hr/~ssegvic/du/du2convnet.pdf
24. https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/
25. https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

26. https://stackoverflow.com/questions/44176982/how-flatten-layer-works-in-keras?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
27. https://www.quora.com/In-Keras-what-is-a-dense-and-a-dropout-layer

11 Popis slika

12 Prilozi

Uz ovaj rad priloženi su tri koda i izlazna datoteka za problem klasifikacije znamenaka. Jedan kod tiče se klasifikacije znamenaka, dok se druga 2 koda tiču klasifikacije filmskih postera.