SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5120

Klasifikacija slika dubokim konvolucijskim modelima

Krešimir Kralj

Zagreb, srpanj 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 10. ožujka 2017.

ZAVRŠNI ZADATAK br. 5120

Pristupnik: **Krešimir Kralj (0036483700)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Klasifikacija slika dubokim konvolucijskim modelima**

Opis zadatka:

Klasifikacija slika prirodnih scena je neriješen problem računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme najbolji rezultati u tom području postižu se pristupima utemeljenima na dubokim konvolucijskim modelima. Za ovaj rad posebno su zanimljivi nadzirani pristupi gdje je svaka slika skupa za učenje označena semantičkim razredima objekata koje slika sadrži.
U okviru rada, potrebno je proučiti dokumentaciju programskog okvira Tensorflow te biblioteke programskog jezika Python za rukovanje matricama i slikama. Izraditi izvedbu programskog sustava za učenje i primjenu jednostavne klasifikacijske arhitekture korištenjem konvolucije, sažimanja, normiranja i potpuno povezanih slojeva. Uhodati duboku arhitekturu za klasifikaciju slika. Prikazati i ocijeniti ostvarene rezultate.
Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 9. lipnja 2017.

Mentor:
Izv. prof. dr. sc. Siniša Šegvić

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Siniša Srbljić

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

Zahvaljujem se mentoru prof. dr. sc. Siniši Šegviću na pruženoj pomoći pri izradi ovog rada te svojoj obitelji na podršci tijekom cijelog školovanja.

SADRŽAJ

1. Uvod

Zadaća problema klasifikacije slika je određivanje ispravne klase iz fiksnog skupa kategorija za svaku sliku iz ulaznog skupa podataka. Problem klasifikacije jedan je od glavnih problema u području računalnog vida te se njegovim rješavanjem otvara širok spektar praktičnih primjena. Mnogi problemi računalnog vida (prepoznavanje objekata, segmentacija) mogu se svesti na problem klasifikacije slika. Iako postoje različite metode za rješavanje ovog problema, pokazalo se da konvolucijske neuronske mreže vrlo uspješno rješavaju probleme detekcije, segmentacije te prepoznavanja objekata i regija na slikama. Većina ovih problema ima dostupne velike količine prikupljenih podataka, kao na primjer: prepoznavanje prometnih znakova [3], segmentacija bioloških slika [13], detekcija lica [4] i pješaka [14]. Današnje arhitekture konvolucijskih mreža sastoje se od stotina milijuna težinskih parametara uz milijarde veza među neuronima [11]. Napredak u hardverskoj tehnologiji i softverskoj optimizaciji doveo je do toga da ovakve ogromne mreže uz korištenje paralelizacije mogu provesti proces učenja u prihvatljivom vremenskom razdoblju.

Nastanak neuronskih mreža inspiriran je mogućnostima učenja ljudskog mozga, koji je sastavljen od neurona povezanih sinapsama. Uz to, neuronske mreže omogućuju jednostavno ugrađivanje prethodnog znanja u svoju arhitekturu. U ovom radu razmatrane su konvolucijske neuronske mreže koje svoje operacije izvršavaju nad slikovnim podacima. Umjesto prepoznavanja predefiniranih obilježja, koriste se da bi samostalno naučile prepoznavati karakteristike koje se koriste u kasnijim fazama klasifikacijskog procesa. To je uspješno omogućeno pomoću konvoluiranja ulazne slike i naučenih filtara s ciljem izgradnje niza značajki.

Cilj ovog rada je izrada dvije konvolucijske neuronske mreže, od kojih će jedna biti testirana na skupu MNIST, a druga na skupu CIFAR-10. Arhitektura, dijelovi programske implementacije te rezultati za obje mreže prikazani su u nastavku.

2. Općenito o konvolucijskim mrežama

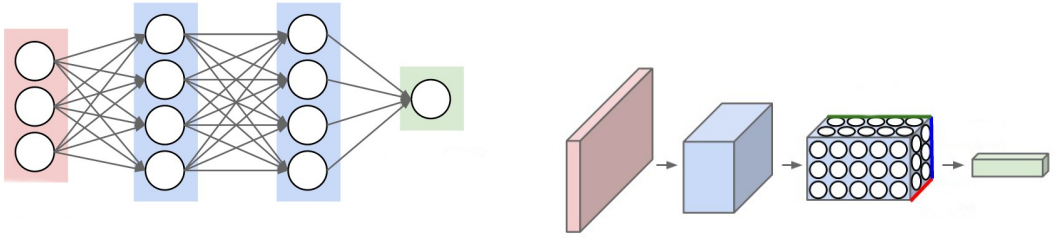
Trenutni pristupi raspoznavanju objekata na slikama uvelike ovise o metodama koje su proizašle iz područja strojnog učenja. Konvolucijske neuronske mreže vrlo su slične neuronskim mrežama, čak se na njih može gledati kao na proširenje višeslojnih neuronskih mreža. Analogno neuronskim mrežama sastoje se od neurona koji, ovisno o svojim težinama, računaju izlaz u ovisnosti o predanom ulazu.

Korištenje konvolucijskih neuronskih mreža zasniva se na pretpostavci da se njezini ulazni podaci mogu prikazati pomoću višedimenzionalnih polja (npr. rasterska grafika koja se obično sastoji od tri dvodimenzionalna polja), što im dozvoljava upotrebu specifičnih svojstava kojima se uvelike smanjuje broj potrebnih parametara te povećava efikasnost izračuna. Cijela mreža se i dalje može izraziti pomoću jedinstvene diferencijabilne funkcije dobitka, koja iz numeričkih RGB vrijednosti piksela (0-255) na ulazu računa ocjene klasa na izlazu. Stoga, u usporedbi s neuronskim mrežama konvolucijske neuronske mreže imaju puno manje veza i parametara pa su pogodnije za učenje i jednostavnije za korištenje. Iako su konvolucijske neuronske mreže zbog svojih svojstava efikasnije od neuronskih mreža i dalje je relativno skupo koristiti ih za klasifikaciju slika velikih rezolucija [10].

2.1. Arhitektura kovolucijske neuronske mreže

Za razliku od slojeva neuronskih mreža, konvolucijske neuronske mreže sastoje se od slojeva organiziranih tako da imaju tri dimenzije: dužinu, visinu te širinu.

Svaka konvolucijska neuronska mreža se sastoji od slojeva, a svaki sloj ima jednaku ulogu: transformacija ulaznih trodimenzionalnih podataka u izlazne trodimenzionalne podatke koristeći odabranu diferencijabilnu funkciju.



(a) Izgled neuronske mreže **(b)** Izgled konvolucijske neuronske mreže

Slika 2.1: Usporedba arhitekture neuronske i konvolucijske neuronske mreže. Svaki sloj neuronska mreže (prikazana lijevo) sastavljen je od jednodimenzionalno raspoređenih neurona, dok je svaki sloj konvolucijske neuronske mreže (prikazana desno) sastavljen od neurona koji zajedno čine volumen trodimenzionalnog oblika.

2.2. Slojevi konvolucijskih mreža

Pri izradi konvolucijske neuronske mreže najčešće se koriste tri glavna tipa slojeva: konvolucijski sloj, potpuno-povezani sloj i sloj sažimanja.

2.2.1. Konvolucijski sloj

Konvolucijski sloj pripada među glavne dijelove konvolucijske mreže i u njemu se izvršava najviše računskih operacija. Parametri konvolucijskog sloja sastoje se od skupa filtara čije težine treba naučiti. Svaki filter je po svojoj širini i dužini manji, ali po dubini jednak dimenziji ulaznog sloja (npr. filter dimenzija $5 \times 5 \times 3$ u prvom sloju konvolucijske mreže korišten za klasifikaciju slike dimenzija $32 \times 32 \times 3$). Tijekom unaprijednog prolaza (*eng. forward-pass*) pomičemo (konvoluiramo) filter po širini i dužini ulaza za svaku poziciju, što rezultira dvodimenzionalnim poljem koje predstavlja odzive tog filtra za svaku poziciju ulaznog sloja. Intuitivno, mreža će naučiti filtre da se aktiviraju kada prepoznaju određeni slikovni uzorak, na primjer: rubove, određene boje i slično. Obično se svaki konvolucijski sloj sastoji od više takvih filtara te svaki od njih generira prethodno opisanu dvodimenzionalnu aktivacijsku mapu. Skup svih aktivacijskih mapa u jednom konvolucijskom sloju čini njegov izlazni volumen.

Kada su dimenzije ulaza velike, nije praktično potpuno-povezivati neurone s ulazom. Umjesto toga, svaki neuron će biti povezan samo s manjim dijelom ulaznog volumena. Prostorna veličina ovakvog načina povezivanja je hiperparametar koji se naziva prihvatljivo polje neurona (veličina filtra). Ako pretpostavimo da nam je na ulazu slika iz skupa CIFAR-10 dimenzija $32 \times 32 \times 3$, a veličina filtra prvog sloja

konvolucijske mreže iznosi 5×5, onda će svaki neuron u prvom sloju mreže imati dimenzije svojih težina $5 \times 5 \times 3$ (dimenzija dubine ostaje jednaka).

U prethodnom dijelu objašnjen je način na koji se neuroni konvolucijskog sloja povezuju s ulazima, a sada ćemo proučiti način na koji su oni raspoređeni u konvolucijskom sloju. Da bi odredili veličinu izlaza konvolucijskog sloja, potrebna su nam tri parametra:

1. Dubina (*eng. depth*) - odgovara broju filtara sadržanih u konvolucijskom sloju
2. Korak (*eng. stride*) - broj piksela koje preskačemo pri svakoj iteraciji konvolucije
3. Nadopunjavanje nulama (*eng. zero-padding*) - najčešće se koristi kada želimo da izlazne dimenzije konvolucijskog sloja budu jednake ulaznima

Dimenzije izlaznog volumena moguće je izračunati ako su nam poznate dimenzije ulaznog volumena (oznaka W), veličina filtra (oznaka F), korak (oznaka S) te količina nadopunjavanja oko rubova (oznaka P). Tada je dimenzija izlaznog volumena jednaka: $(W - F + 2 \cdot P)/S + 1$. U slučaju ulaza dimenzija $7 \times 7 \times 1$, filtra veličine $3 \times 3 \times 1$ te koraka iznosa jedan ($S = 1$) bez korištenja nadopunjavanja ($P = 0$), izlazni sloj bi imao dimenzije $5 \times 5 \times 1$.

Temeljna svojstva konvolucijskog sloja su:

1. Prihvaćanje ulaza dimenzija $W_1 \times H_1 \times D_1$
2. Zahtijevanje četiri hiperparametra: broj filtara (K), dimenzije filtra (F), pomak (S), iznos nadopunjavanja nulama (P).
3. Korištenjem operacije konvolucije stvara izlaz dimenzija $W_2 \times H_2 \times D_2$ gdje je:
   (a) $W_2 = 1 + (W_1 - F + 2 \cdot P)/S$
   (b) $H_2 = 1 + (H_1 - F + 2 \cdot P)/S$
   (c) $D_2 = K$
4. Svaki filtar ima $F \cdot F \cdot D_1$ težina, što ukupno čini $(F \cdot F \cdot D_1) \cdot K$ parametara za učenje.
5. U izlaznom volumenu, i-ti dubinski sloj rezultat je operacije konvolucije i-tog filtra i ulaznog volumena uz korištenje prethodno zadanih hiperparametara.

2.2.2. Sloj sažimanja

Slojevi sažimanja u konvolucijskim mrežama najčešće se koriste između jednog ili više uzastopnih konvolucijskih slojeva, s ciljem smanjenja dimenzije mape te parametara potrebnih za izračun. Sažimanje se vrši na manjem dijelu ulaza koji se, nakon izvršavanja funkcije sažimanja (*eng. pooling function*), pretvara u jednu vrijednost. Funkcija sažimanja mapira skup prostorno bliskih značajki na ulazu u jednu značajku na izlazu, te obično računa neku statističku vrijednost skupa značajki na ulazu. Najčešći oblik sloja sažimanja ima filtre veličine 2×2, koristi pomak iznosa 2 po širini i visini i kao funkciju sažimanja koristi operaciju najveće vrijednosti, što rezultira odbacivanjem 75% aktivacijskih vrijednosti (od četiri vrijednosti odabire se najveća, ostale se odbacuju). Sažimanje ne utječe na dimenziju dubine. Sloj sažimanja svoje operacije ne temelji na nepoznatim parametrima već na fiksnoj funkciji koja računa izlaz ovisno o predanom ulazu (npr. funkcija MAX koja vraća najveći element iz ulaznog skupa). Formalno se proces rada sloja sažimanja može se opisati fazama:

1. Prihvati ulaz dimenzija $W_1 \times H_1 \times D_1$
2. Zatraži dva hiperparametra: dimenziju filtra (F) i pomak (S)
3. Korištenjem funkcije sažimanja stvori izlaz dimenzija $W_2 \times H_2 \times D_2$ gdje je:
   (a) $W_2 = 1 + (W_1 - F)/S$
   (b) $H_2 = 1 + (H_1 - F)/S$
   (c) $D_2 = D_1$

Korištenjem sloja sažimanja postiže se invarijantnost na pomak, što može biti vrlo korisno u slučajevima kad je za raspoznavanje bitnije detektirati prisutnost određene značajke od njezine točne lokacije [18]. Dimenzija filtra proporcionalna je invarijantnosti na pomake.

2.2.3. Potpuno-povezani sloj

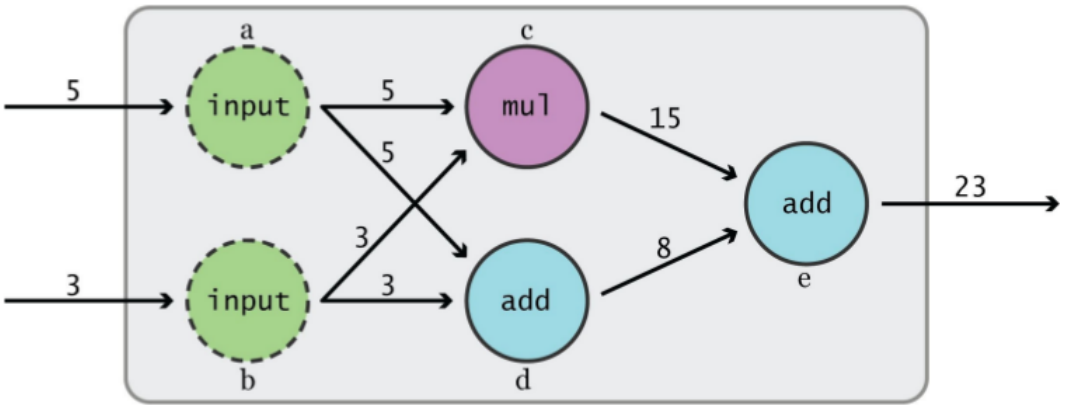
Općenito gledano, izlaz iz konvolucijskih slojeva predstavlja niz uzoraka koji su nastali učenjem ulaznih podataka. Potpuno-povezani sloj mreži daje potencijal za učenje prostornog rasporeda konvolucijskih značajki. Svaki neuron spojen je sa svim neuronima iz prethodnog sloja, a slojevi mogu se prikazati kao jednodimenzionalni, što znači da nakon njih u mreži ne mogu postojati konvolucijski slojevi. Računanje aktivacijske vrijednosti sastoji se od matričnog množenja ulaza i težina uz dodavanje konstantne početne vrijednosti, što je analogno računanju vrijednosti izlaza kod neuronskih mreža.

3. Općenito o Tensorflowu

Tensorflow je aplikacijski okvir otvorenog koda koji pruža implementaciju algoritama strojnog učenja te olakšava njihovo korištenje. Zbog jednostavne i organizirane arhitekture, efikasne implementacije te opširne dokumentacije, Tensorflow je, uz svoje konkurente (Caffe, Torch), trenutno jedan od najboljih izbora za rješavanje problema korištenjem algoritmima strojnog učenja. Tensorflow ima mnoge primjene, a jedna od njih je učenje i ispitivanje rezultata dobivenih korištenjem dubokih neuronskih modela, što će biti prikazano u daljnjim poglavljima.
Svaki program u napisan Tensorflow-u sastoji se od dvije odvojene faze:

1. Oblikovanje računskog grafa
2. Zadavanje računskih upita

Računski graf se sastoji od serije već implementiranih operacija koje se povezuju u jedinstvenu cjelinu, pa Tensorflow operacije možemo zamišljati kao čvorove računskog grafa. Odvajanje faza stvaranja računskog grafa te zadavanja upita omogućava efikasno računanje na različitim računalnim sustavima.



Slika 3.1: Primjer računskog grafa koji na temelju ulaznih parametara a i b računa izlaznu vrijednost funkcije: $a \cdot b + (a + b)$.

Programski kod koji stvara i pokreće računski graf s prethodne slike prikazan je u nastavku.

```
import tensorflow as tf
# Definicija racunskog grafa
a = tf.constant(5, name="input_a")
b = tf.constant(3,name="input_b")
c = tf.mul(a, b, name="mul_c")
d = tf.add(a, b, name="add_d")
e = tf.add(c, d, name="add_e")
# Racunanje izlazne vrijednosti
sess = tf.Session()
sess.run(e)  # 23
```

3.1. Primjena u strojnom učenju

Kad u Tensorflow-u oblikujemo postupak strojnog učenja, sastavljanje računskog grafa obično se sastoji od sljedećih koraka:

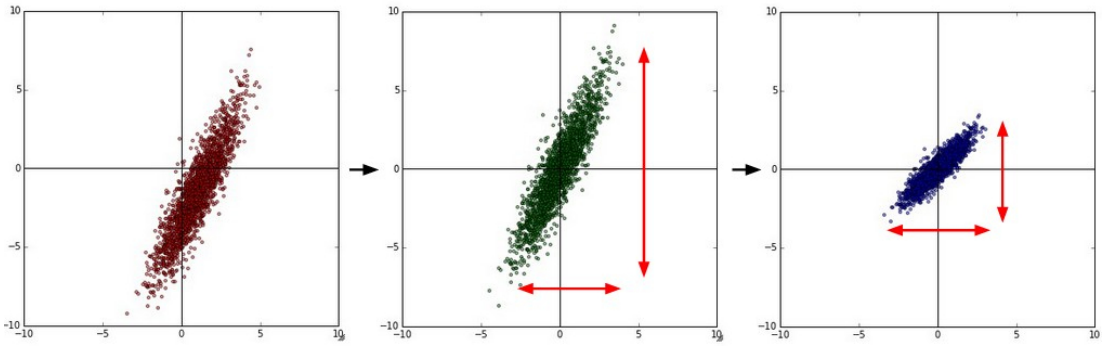
1. Definiranje ulaznih i izlaznih parametara
2. Definiranje modela strojnog učenja
3. Definiranje funkcije gubitka
4. Definiranje optimizacijskog postupka

Nakon što je računski graf sastavljen, pokreće se faza računanja, čija je zadaća evaluacija računskog grafa na temelju predanih podataka. Faza računanja obično se sastoji od učenja, provjere i ispitivanja mreže.

4. Ispitni skupovi

4.1. Predobrada podataka

Da bi lakše manipulirali podacima, uobičajeno je dvodimenzionalno polje piksela slike pretvoriti u jednodimenzionalni niz. U slučaju korištenja podataka iz skupa CIFAR-10, svaku sliku dimenzija $32 \times 32 \times 3$ bi pretvorili u niz od 3072 broja nad kojim bi vršili daljnje operacije. Kada radimo s algoritmima strojnog učenja, jedna od najčešćih predobrada podataka je njihovo centriranje. U slučaju slikovnih podataka, centriranje se postiže oduzimanjem aritmetičke sredine svih elemenata nekog skupa od svakog elementa u tom skupu (da bi učenje težina moglo "lagano" krenuti u zahtijevanom smjeru [16]). Osim centriranja podatkovnih vrijednosti, vrlo često se provodi i postupak normalizacije varijance s ciljem zadržavanja opsega podataka u sredini aktivacijske funkcije.



Slika 4.1: Grafički prikaz podataka prije i nakon predobrade. Na slici lijevo su prikazane izvorne vrijednosti podataka. Slika u sredini prikazuje centrirane podatke, a nakon normalizacije varijance rezultantni podaci su prikazani na slici desno. Slika je preuzeta sa službenih stranica predmeta CS231n [7].

4.2. MNIST

MNIST (*Modified National Institute of Standards and Technology*) predstavlja skup slika rukom pisanih znamenki (od nula do devet). Podaci MNIST skupa su podskup većeg skupa podataka pod nazivom NIST te su prikladni za korištenje u metodama strojnog učenja. Svaka slika iz skupa MNIST ima veličinu 28×28 piksela, a za pohranu svakog piksela koristi se 8 bita memorijskog prostora.

Slika 4.2: Primjer svakog od deset različitih razreda iz skupa MNIST

Podaci u MNIST-u se općenito dijele na tri dijela:

1. 55,000 podataka za treniranje (`mnist.train`)
2. 10,000 podataka za testiranje (`mnist.test`)
3. 5,000 podataka za validaciju (`mnist.validation`)

U nastavku je prikazan primjer učitavanja skupa MNIST koristeći Tensorflow:

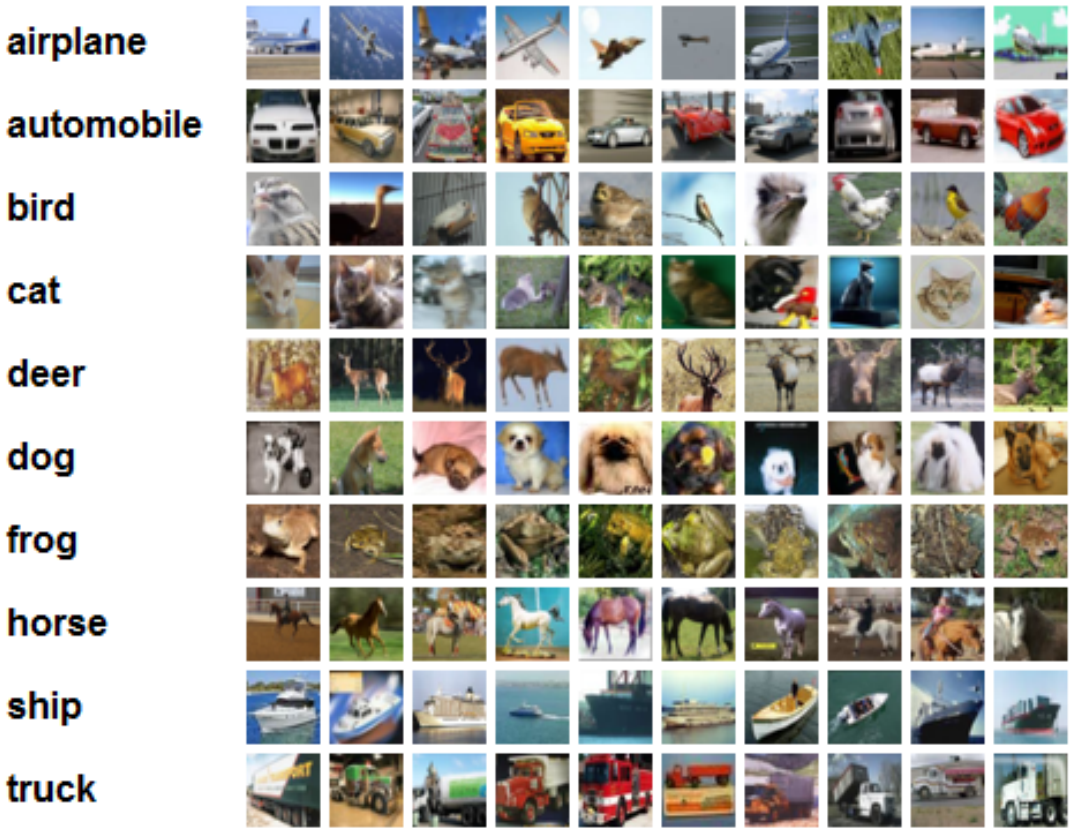
```
from tensorflow.examples.tutorials.mnist import input_data
tf.app.flags.DEFINE_string('data_dir',
'/tmp/data/', 'Direktorij za pohranu podataka')
mnist = input_data.read_data_sets(
  tf.app.flags.FLAGS.data_dir, one_hot=True)
```

Nakon učitavanja, skupovi slika i indeksi razreda predstavljeni su numpyjevim matricama te ih je moguće dohvatiti preko referenci `mnist.train.images` i `mnist.train.labels` koje će biti korištene u daljnjim fazama izračuna. Ovakva podjela

olakšava daljnje izračune jer je u procesima strojnog učenja vrlo važno razdvojiti podatke na temelju kojih učimo model od podataka koji služe za provjeru i ispitivanje, što u konačnici osigurava da predviđeni model daje generalizirane rezultate.

4.3. CIFAR-10

CIFAR-10 skup sastavljen je od ukupno 60,000 slika dimenzija $32 \times 32 \times 3$, s točno 6,000 slika po svakoj klasi. Skup za treniranje sastoji se od 50,000 slika, dok skup za testiranje čini 10,000 slika. Svih deset klasa CIFAR-10 skupa su potpuno neovisne jedna o drugoj.



Slika 4.3: Primjer slučajno odabranih uzoraka iz skupa CIFAR-10 za svaku od ukupno 10 klasa

5. Eksperimentalni rezultati

U ovom poglavlju prikazane su arhitekture dvaju konvolucijskih neuronskih mreža od kojih se jedna koristi za klasifikaciju slika skupa MNIST, a druga za klasifikaciju nad skupom CIFAR-10. Postignuti rezultati prikazani su grafički.

5.1. Rezultati na skupu MNIST

5.1.1. Arhitektura mreže

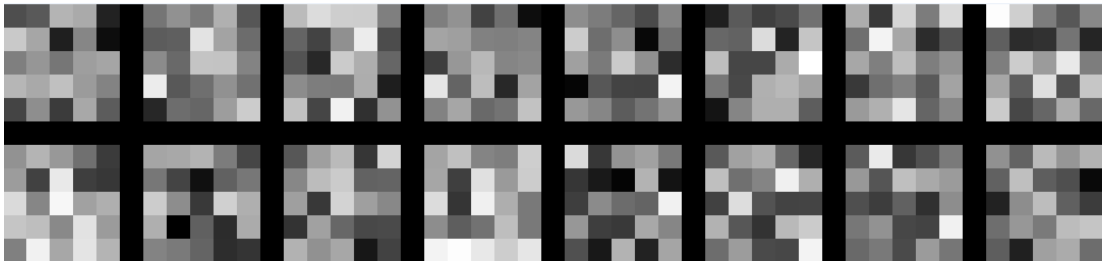
Za klasifikaciju skupa MNIST odabrana je arhitektura koja se sastoji od sljedećih slojeva:

1. Ulazni sloj - podatak iz skupa MNIST dimenzija $28 \times 28 \times 1$
2. Prvi konvolucijski sloj - Dimenzija filtra 5×5, 1 ulazni i 16 izlaznih kanala. Pomak po dužini i širini iznosi 1. Kao aktivacijska funkcija koristi se zglobnica (*eng. rectified linear unit*).
3. Prvi sloj sažimanja - Sadrži okno dimenzije 2×2, a pomak po dužini i širini iznosi 2.
4. Drugi konvolucijski sloj - Dimenzija filtra 5×5, 16 ulaznih i 32 izlazna kanala. Pomak po dužini i širini iznosi 1. Kao aktivacijska funkcija također se koristi zglobnica.
5. Drugi sloj sažimanja - Analogno prvom sloju sažimanja, sadrži okno dimenzije 2×2, a pomak po dužini i širini također iznosi 2.
6. Prvi potpuno povezani sloj - Sastavljen od 512 neurona.
7. Izlazni sloj - Svaki od 10 izlaznih neurona predstavlja jednu klasu iz skupa MNIST.

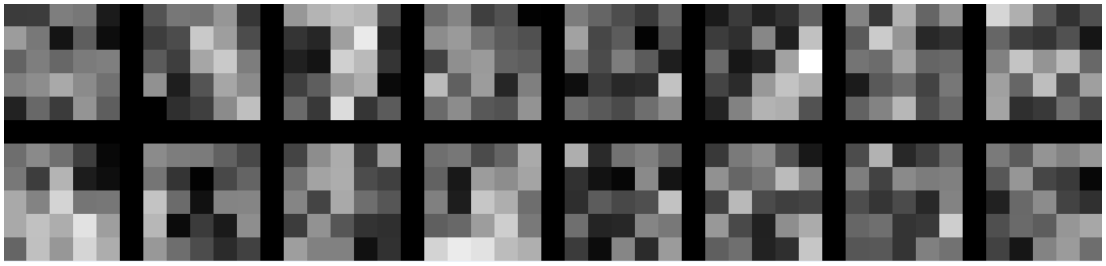
5.1.2. Rezultati

Navedena arhitektura trenirana je kroz osam epoha na skupu za učenje. Mreža je trenirana ukupno tri puta, svaki put s različitim regularizacijskim parametrom (u nastavku označenim sa λ). Regularizacijski parametar potiče mrežu da prilikom treniranja preferira manje, ali ravnomjernije raspodijeljene vrijednosti težina umjesto većih, dominirajućih težina zbog kojih dolazi do pretreniranosti (*eng. overfitting*) mreže.

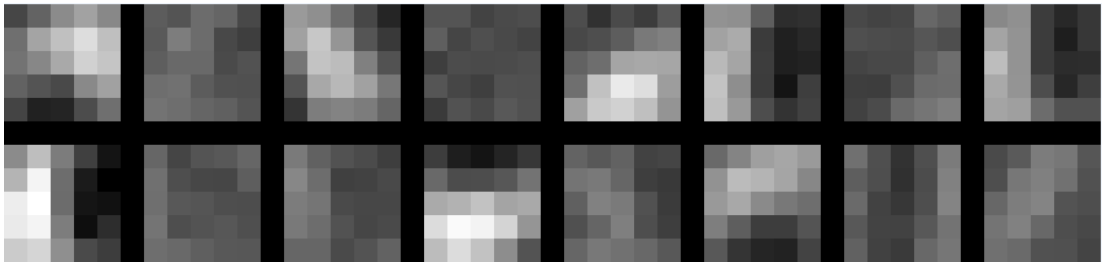
U nastavku slijedi prikaz i usporedba filtara prvog konvolucijskog sloja za svaki od ukupno tri regularizacijska parametra ($\lambda \in \{0.001, 0.01, 0.1\}$).



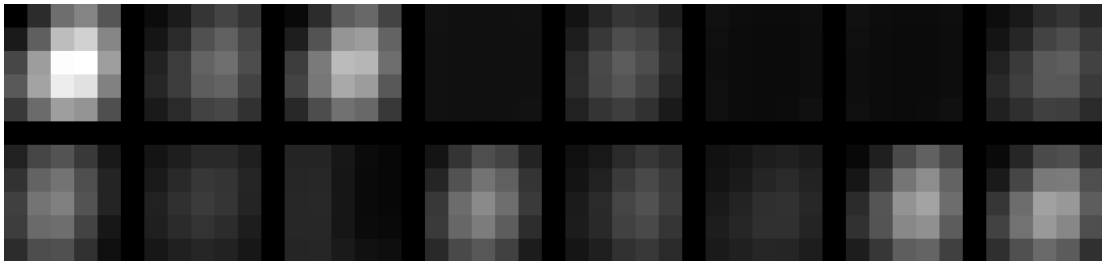
Slika 5.1: Slučano inicijalizirani filtri na početku učenja



Slika 5.2: Naučeni filtri s regularizacijom $\lambda = 0.001$

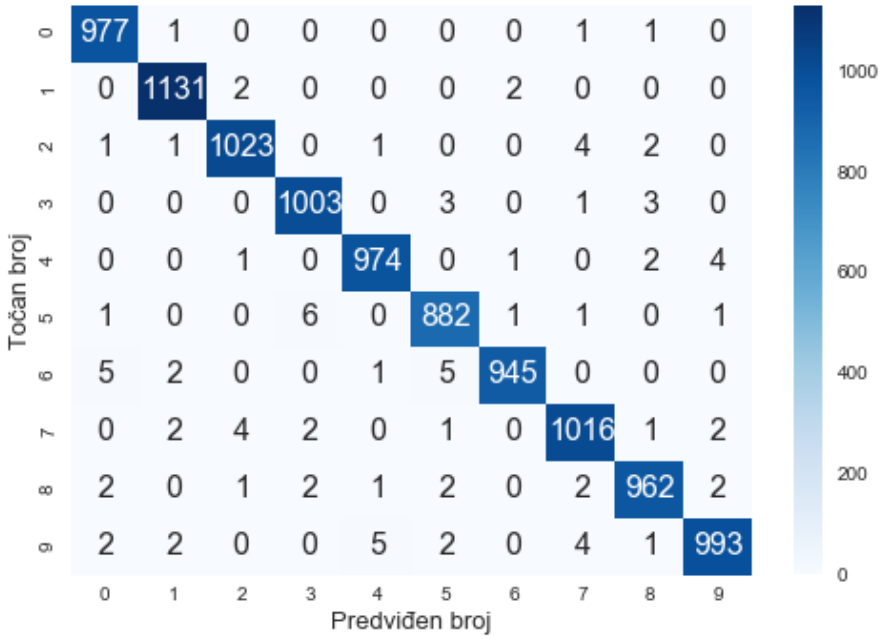


Slika 5.3: Naučeni filtri s regularizacijom $\lambda = 0.01$



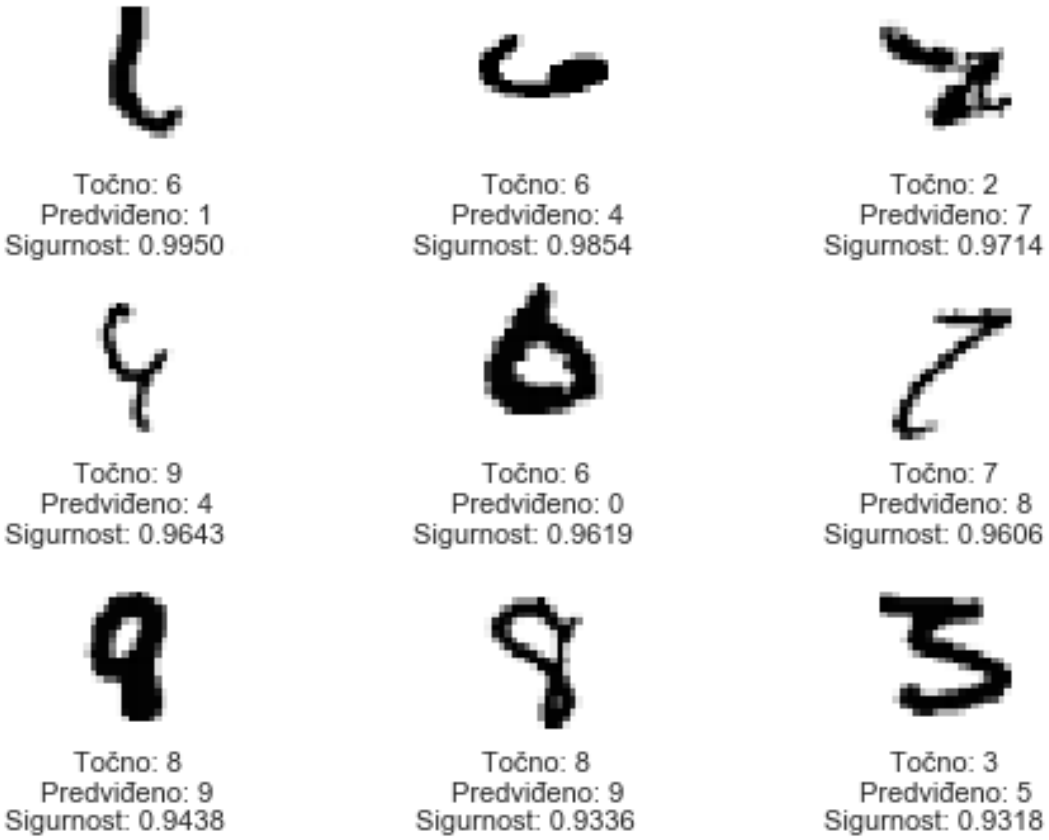
Slika 5.4: Naučeni filtri s regularizacijom $\lambda = 0.1$

Najveću točnost postiže mreža s najmanjim regularizacijskim parametrom $\lambda = 0.001$, te ona iznosi 99.06% na ispitnom skupu i 99.41% na skupu za učenje. Nešto lošiji rezultat postiže se s parametrom $\lambda = 0.01$ i on na ispitnom skupu iznosi 98.47%, a na skupu za učenje 98.52%, dok s točnosti 93.26% na ispitnom skupu te 92.98% na skupu za učenje najviše zaostaje mreža s visokim regularizacijskim parametrom $\lambda = 0.1$. Iz priloženih slika moguće je primijetiti kako postoji ovisnost izgleda filtra o veličini regularizacijskog parametra. Intuitivno je jasno da će malen regularizacijski parametar dozvoliti velike iznose pojedinih težina u filtrima, što se i prikazano slici 6.2 gdje je $\lambda = 0.001$, dok će veliki regularizacijski parametar zabraniti takva odstupanja, te će nam filtri izgledati zamućeno (što se može vidjeti sa slike 6.4). U nastavku slijedi prikaz matrice zabune za model s najvećim postotkom točnosti ($\lambda = 0.001$):



Slika 5.5: Matrica zabune ispitnih podataka iz skupa MNIST

Zanimljivo je pogledati nekoliko primjera za koje trenirana mreža daje krivi rezultat. Kao što se vidi na sljedećoj slici, određene uzorke iz skupa MNIST je vrlo teško ispravno klasificirati. Postoje slučajevi koje niti ljudska osoba ne može ispravno klasificirati s potpunom sigurnošću.



Slika 5.6: Prikaz grešaka s najvećom sigurnošću nakon treniranja mreže. Sigurnost je vjerojatnost kojom model predviđa ispravnost svoje klasifikacije.

5.2. Rezultati na skupu CIFAR-10

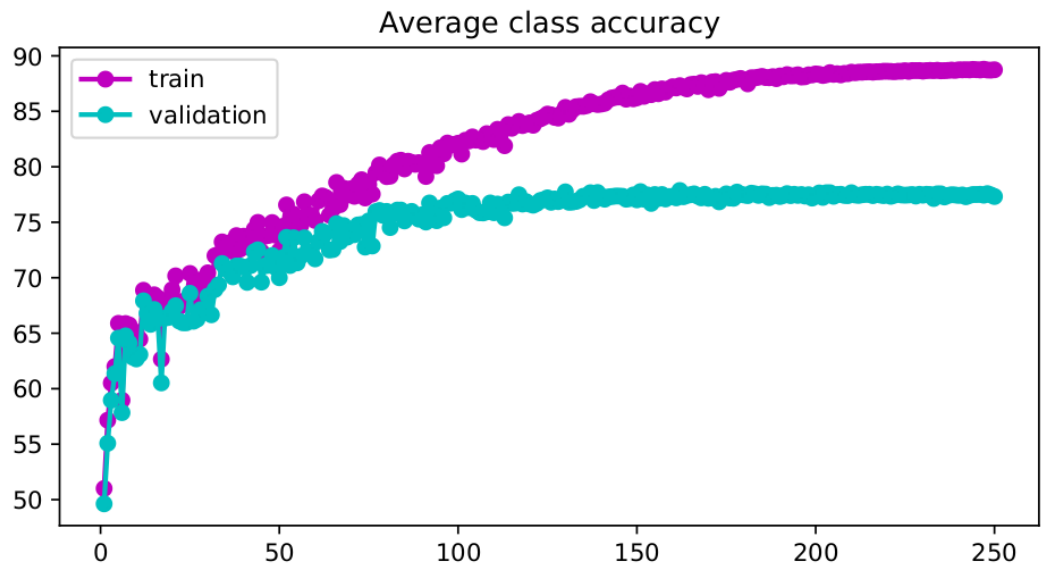
5.2.1. Arhitektura mreže

Za klasifikaciju skupa CIFAR-10 odabrana je arhitektura koja se sastoji od sljedećih slojeva:

1. Ulazni sloj - podatak iz skupa CIFAR-10 dimenzija $32 \times 32 \times 3$
2. Prvi konvolucijski sloj - Dimenzija filtra 5×5, 3 ulazna i 16 izlaznih kanala. Pomak po dužini i širini iznosi 1. Kao aktivacijska funkcija koristi se zglobnica.
3. Prvi sloj sažimanja - Sadrži okno dimenzije 3×3, a pomak po dužini i širini iznosi 2.
4. Drugi konvolucijski sloj - Dimenzija filtra 5×5, 16 ulaznih i 32 izlazna kanala. Pomak po dužini i širini iznosi 1. Kao aktivacijska funkcija također se koristi zglobnica.
5. Drugi sloj sažimanja - Sadrži okno dimenzije 3×3, a pomak po dužini i širini iznosi 2.
6. Prvi potpuno povezani sloj - Sastavljen od 256 neurona. Kao aktivacijska funkcija koristi se zglobnica.
7. Drugi potpuno povezani sloj - Sastavljen od 128 neurona. Kao aktivacijska funkcija koristi se zglobnica.
8. Izlazni sloj - Svaki od 10 izlaznih neurona predstavlja jednu klasu iz skupa CIFAR-10.

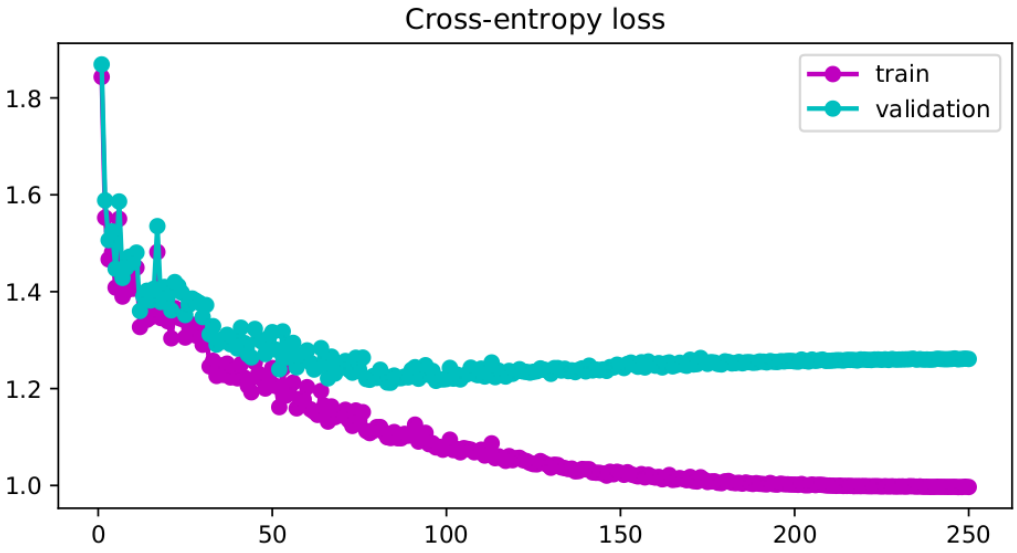
5.2.2. Rezultati

Mreža je trenirana s ukupno 250 prolaza kroz skup za učenje, koristeći eksponencijalno padajuću stopu učenja. Kao funkcija gubitka korištena je funkcija Softmax uz regularizaciju s parametrom $\lambda = 0.020$. Istrenirana mreža na ispitnom skupu postiže točnost iznosa 76.06%, slično kao i na validacijskom skupu, dok je na skupu za treniranje postignuta točnost iznosa 91.74%.

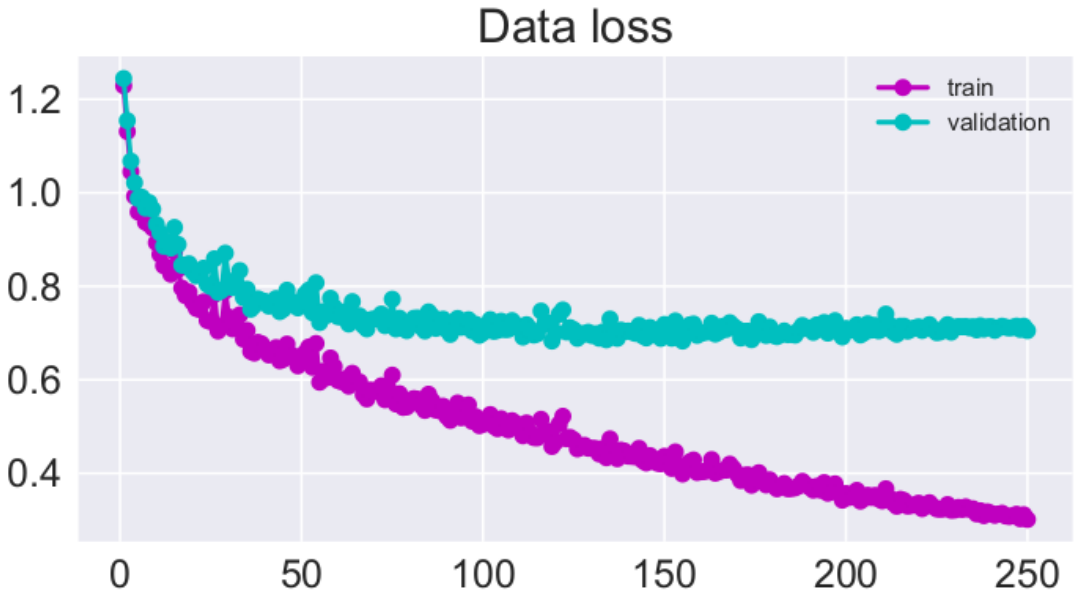


Slika 5.7: Prikaz promjene klasifikacijske točnosti na skupu za učenje i skupu za ispitivanje.

Iz gubitaka prikazanih na slici 5.8 vidimo da su na oba grafa linije validacijskog skupa te skupa treniranja relativno blizu jedna drugoj, što znači da, iako dolazi do pretreniranja (*eng. overfitting*) mreže, ono nije prevelikog iznosa pa naučena mreža i dalje postiže dobru generalizaciju. Nakon završetka učenja mreže, prosječna greška na skupu za učenje iznosi 0.97, na skupu za provjeru 1.32, a na skupu za ispitivanje iznosi 1.36, što je nešto lošije od greške na skupu za provjeru.

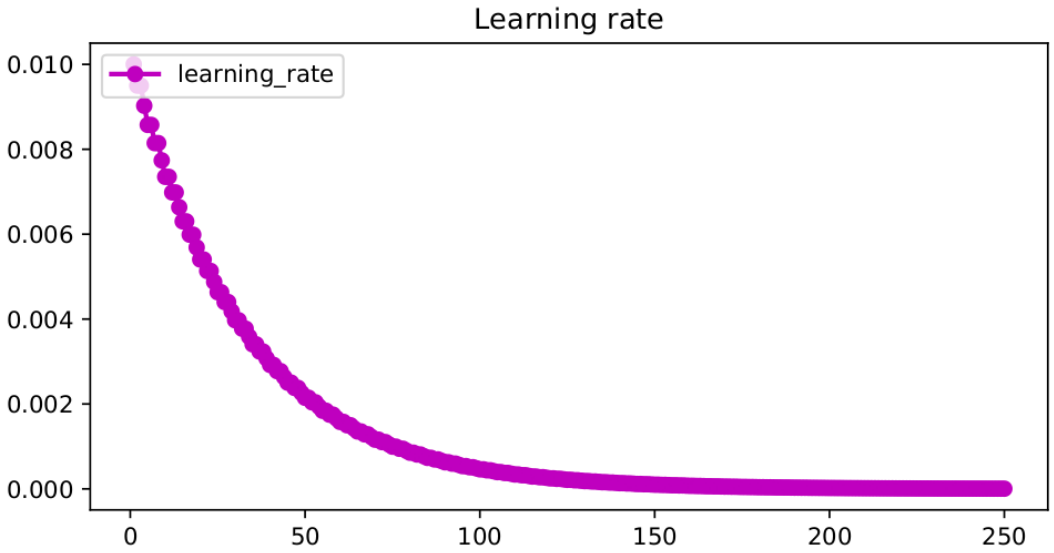


Slika 5.8: Prikaz promjene prosječne pogreške na skupu za učenje i skupu za ispitivanje.



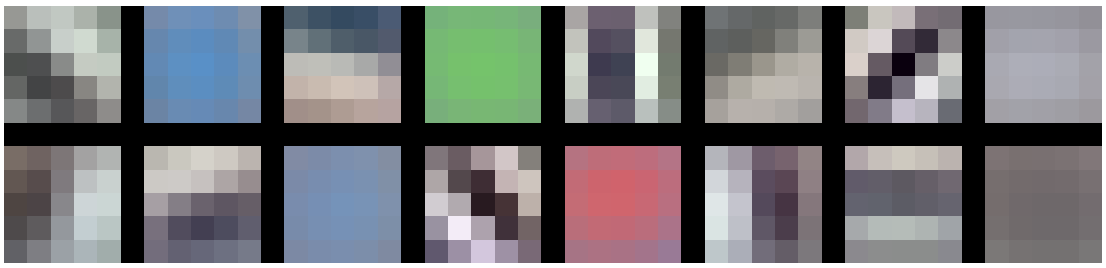
Slika 5.9: Prikaz promjene prosječne podatkovne pogreške na skupu za učenje i skupu za ispitivanje.

Često je preporučeno smanjivati stopu učenja tijekom procesa treniranja mreže. Intuitivno, uz visoku stopu učenja sustav daje velike promjene vrijednostima parametara što rezultira nemogućnošću pronalaženja boljih, ali bližih vrijednosti za koje je funkcija gubitka manja. Vrlo je teško predvidjeti najbolje trenutke za smanjenje stope učenja. U slučaju da se stopa učenja smanjuje sporo, proces računanja će biti beskoristan jer će sustav titrati oko boljih vrijednosti ali ih neće pogoditi. Suprotno tomu, ako stopu učenja smanjujemo brzo, postoji mogućnost da sustav završi u lokalnom optimumu zbog nemogućnosti dolaska u najbolju poziciju za koju mu trebaju velike promjene u vrijednostima.



Slika 5.10: Prikaz eksponencijalne promjene vrijednosti stope učenja za svaku epohu učenja mreže.

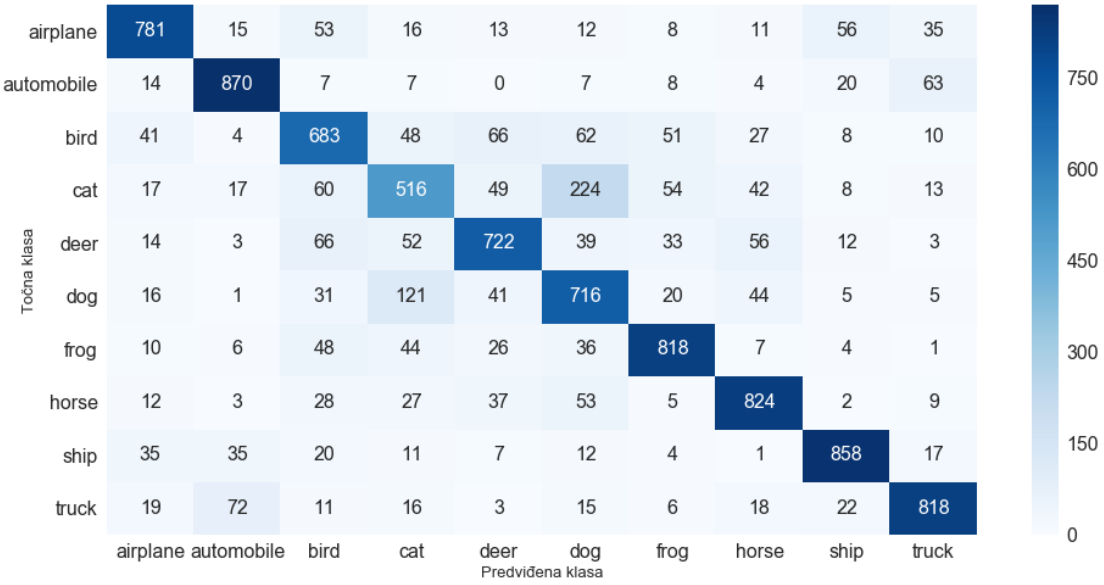
U nastavku slijedi prikaz filtara za koje se može primijetiti da nisu nasumični već slijede određeni uzorak. U ovom slučaju, filtri prvog sloja su naučeni da prepoznaju uzorke slične rubovima slika. Filtri koji sadrže puno šuma često upućuju na neke od sljedećih problema: mreža čiji parametri divergiraju, loše postavljena stopa učenja ili nizak regularizacijski parametar. U slučajevima kada su parametri učenja postavljeni pravilno, filtri izgledaju čisto, glatko i sadrže različite uzorke.



Slika 5.11: Prikaz filtara prvog konvolucijskog sloja naučene mreže.

Objekti na slikama iz skupa CIFAR-10 koji pripadaju istoj klasi izgledom mogu jako varirati. Kao primjer možemo promotriti klasu automobila koja sadrži velike, srednje i male automobile, koji su uslikani iz različitih kuteva, položaja i udaljenosti. Klasifikacija je još teža u slučajevima psa i mačke jer se te dvije vrste životinja mogu susresti na vrlo različitim mjestima, u vrlo različitim pozama te je čest slučaj da niti čovjek ne može biti siguran o kojoj se točno klasi radi ako na slici nisu jasno prikazana specifična svojstva te klase. U nastavku slijedi prikaz matrice zabune iz koje se može

vidjeti da naučeni model najviše griješi upravo na klasifikaciji podataka koji pripadaju klasama psa i mačke.



Slika 5.12: Prikaz matrice zabune naučene mreže na ispitnom skupu.

U nastavku su prikazane greške nastale prilikom klasifikacije podataka koji pripadaju klasi mačke, a za koje model predviđa da pripadaju klasi psa te devet grešaka s najvećom sigurnošću mreže na ispitnom skupu. Sigurnost je vjerojatnost kojom model predviđa ispravnost svoje klasifikacije.

Slika 5.13: Prikaz neispravno klasificiranih podataka koji pripadaju klasi ’mačka’, a koje je naučeni model svrstao u klasu ’pas’.

Slika 5.14: Prikaz devet neispravno klasificiranih primjera poredanih po sigurnosti naučenog modela u klasifikacijsku ispravnost.

6. Programska izvedba

Za programsku izvedbu odabran je programski jezik Python, uz korištenje biblioteke Tensorflow. Python je pogodan za brzo i jednostavno pisanje programa, a Tensorflow nudi već implementirane funkcije potrebne za stvaranje konvolucijskih modela. U nastavku slijedi prikaz programske izvedbe dva modela konvolucijske neuronske mreže čija arhitektura je opisana u prethodnom poglavlju.

6.1. Izvedba modela skupa MNIST

Izvedba modela na osnovi prethodno opisane arhitekture prikazana je u nastavku.

```
def cnn_model(input_layer):
    # Prvi konvolucijski sloj
    conv1 = tf.nn.conv2d(input_layer, conv1_weights,
        strides=[1, 1, 1, 1], padding='SAME')

    # Prvi aktivacijski sloj
    relu1 = tf.nn.relu(tf.nn.bias_add(conv1, conv1_biases))

    # Prvi sloj sazimanja
    pool1 = tf.nn.max_pool(relu1, ksize=[1, 2, 2, 1],
        strides=[1, 2, 2, 1], padding='SAME')

    # Drugi konvolucijski sloj
    conv2 = tf.nn.conv2d(pool1, conv2_weights,
        strides=[1, 1, 1, 1], padding='SAME')

    # Drugi aktivacijski sloj
    relu2 = tf.nn.relu(tf.nn.bias_add(conv2, conv2_biases))

    # Drugi sloj sazimanja
    pool2 = tf.nn.max_pool(relu2, ksize=[1, 2, 2, 1],
        strides=[1, 2, 2, 1], padding='SAME')
```

```
    # Prvi potpuno-povezani sloj
    flatten1 = tf.reshape(pool2, [-1, 7 * 7 * 32])
    fc1 = tf.matmul(flatten1, fc1_weights)
    relu3 = tf.nn.relu(tf.nn.bias_add(fc1, fc1_biases))

    # Drugi potpuno povezani sloj
    return tf.matmul(relu3, fc2_weights) + fc2_biases
```

Parametri koje mreža treba naučiti prikazani su u nastavku.

```
# Ulazni sloj
input_layer = tf.reshape(X, [-1, 28, 28, 1])

# Parametri konvolucijskih slojeva
conv1_weights = tf.Variable(tf.truncated_normal([5, 5, 1, 16],
    stddev=getstddev([5, 5, 1])))
conv1_biases = tf.Variable(tf.constant(0.1, shape=[16]))

conv2_weights = tf.Variable(tf.truncated_normal([5, 5, 16, 32],
    stddev=getstddev([5, 5, 16])))
conv2_biases = tf.Variable(tf.constant(0.1, shape=[32]))

# Parametri potpuno povezanih slojeva
fc1_weights = tf.Variable(tf.truncated_normal([7 * 7 * 32, 512],
   stddev=getstddev([7, 7, 32])))
fc1_biases = tf.Variable(tf.constant(0.1, shape=[512]))

fc2_weights = tf.Variable(tf.truncated_normal([512, 10],
    stddev=getstddev([512])))
fc2_biases = tf.Variable(tf.constant(0.1, shape=[10]))
```

Kao funkcija gubitka koristi se funkcija Softmax uz regularizacijski gubitak. Za poboljšavanje rezultata učenja korišten je algoritam gradijentnog spusta, uz promjenjive stope učenja ovisno o broju izvršenih epoha. U nastavku slijedi prikaz definicije funkcije gubitka i optimizatora koji se poziva u svakoj iteraciji učenja mreže.

```
# Stvori model
model = cnn_model()
# Definiraj gubitak
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
        labels=Y, logits=model))
# Dodaj regularizacijski gubitak.
regularizers = tf.nn.l2_loss(conv1_weights) +
               tf.nn.l2_loss(conv2_weights) +
               tf.nn.l2_loss(fc1_weights) +
```

```
                tf.nn.l2_loss(fc2_weights)
loss += WEIGHT_DECAY * regularizers
# Korak treniranja
learning_rate = tf.placeholder(tf.float32)
train_step = tf.train.GradientDescentOptimizer(learning_rate).
    minimize(loss)
```

6.2. Izvedba modela skupa CIFAR-10

Izvedba modela na osnovi prethodno opisane arhitekture prikazana je u nastavku.

```
def cnn_model(input_layer):
    # Prvi konvolucijski sloj
    conv1 = tf.nn.conv2d(input_layer, conv1_weights,
        strides=[1, 1, 1, 1], padding='SAME')
    # Prvi aktivacijski sloj
    relu1 = tf.nn.relu(tf.nn.bias_add(conv1, conv1_biases))
    # Prvi sloj sazimanja
    pool1 = tf.nn.max_pool(relu1, ksize=[1, 3, 3, 1],
        strides=[1, 2, 2, 1], padding='SAME')
    # Drugi konvolucijski sloj
    conv2 = tf.nn.conv2d(pool1, conv2_weights,
        strides=[1, 1, 1, 1], padding='SAME')
    # Drugi aktivacijski sloj
    relu2 = tf.nn.relu(tf.nn.bias_add(conv2, conv2_biases))
    # Drugi sloj sazimanja
    pool2 = tf.nn.max_pool(relu2, ksize=[1, 3, 3, 1],
        strides=[1, 2, 2, 1], padding='SAME')
    # Potpuno-povezani slojevi
    flatten1 = tf.reshape(pool2, [-1, 2048])
    fc1 = tf.matmul(flatten1, fc1_weights)
    relu3 = tf.nn.relu(tf.nn.bias_add(fc1, fc1_biases))
    fc2 = tf.matmul(relu3, fc2_weights)
    relu4 = tf.nn.relu(tf.nn.bias_add(fc2, fc2_biases))
    fc3 = tf.matmul(relu4, fc3_weights)
    return tf.nn.bias_add(fc3, fc3_biases)
```

Gdje su parametri učenja:

```
# Ulazni sloj
input_layer = tf.reshape(X, [-1, 32, 32, 3])

# Parametri konvolucijskih slojeva
conv1_weights = tf.Variable(tf.truncated_normal([5, 5, 3, 16],
```

```
stddev=getstddev([5, 5, 3])))
conv1_biases = tf.Variable(tf.constant(0.1, shape=[16]))

conv2_weights = tf.Variable(tf.truncated_normal([5, 5, 16, 32],
stddev=getstddev([5, 5, 16])))
conv2_biases = tf.Variable(tf.constant(0.1, shape=[32]))

# Parametri potpuno povezanih slojeva
fc1_weights = tf.Variable(tf.truncated_normal([2048, 256], stddev=
    getstddev([2048])))
fc1_biases = tf.Variable(tf.constant(0.1, shape=[256]))

fc2_weights = tf.Variable(tf.truncated_normal([256, 128], stddev=
    getstddev([256])))
fc2_biases = tf.Variable(tf.zeros([128]))

fc3_weights = tf.Variable(tf.truncated_normal([128, 10], stddev=
    getstddev([128])))
fc3_biases = tf.Variable(tf.zeros([10]))
```

Kao funkcija ukupnog gubitka koristi se funkcija Softmax uz dodatak regularizacijskog gubitka. Za poboljšavanje rezultata učenja korišten je algoritam gradijentnog spusta, uz promjenjive stope učenja ovisno o broju izvršenih epoha. U nastavku slijedi prikaz definicije stope učenja i optimizatora koji se poziva u svakoj iteraciji učenja mreže.

```
# Korak treniranja
global_step = tf.Variable(0, trainable=False)
learning_rate = tf.train.exponential_decay(0.01, global_step,
        4500, 0.92, staircase=True)
train_step = tf.train.GradientDescentOptimizer(learning_rate).
    minimize(cross_entropy_loss, global_step=global_step)
```

6.3. Treniranje i evaluacija mreže

Nakon što su definirani modeli, proces učenja, provjere i ispitivanja identičan je za obje mreže. U procesu učenja korisno je podijeliti skupove podataka da ne bi došlo do pretreniranja mreže. Brža konvergencija je zapažena u slučajevima kada se podaci slučajno permutiraju nakon svake epohe, što se može izvesti relativno efikasno [2]. Bolji rezultati učenja uzrokovani nasumičnim premještanjem podataka mogu se objasniti time što neuronske mreže najbrže uče iz najneočekivanijih podataka. U slučaju da su ulazni podaci raspoređeni po klasama te ih takve predamo mreži, ona će naučiti

jako dobro klasificirati početnu klasu, no prelaskom na sljedeću klasu greška u klasifikaciji postat će ogromna. Zatim će se mreža početi ispravljati te će naučiti dobro prepoznavati drugu klasu, no ujedno će ispravljati parametre dobivene učenjem prethodnih podataka što će u konačnici rezultirati gubitkom prethodno naučenih težina. Ako u svakoj epohi nasumično ispremještamo podatke, mreža će bolje učiti jer će na njene težine podjednako utjecati sve klase ulaznih podataka. U nastavku slijedi prikaz procesa učenja mreže.

```
for epoch in range(MAX_EPOCHS):
    # permutacija podataka
    permutation_idx = np.random.permutation(NUM_EXAMPLES)
    train_x_perm = train_x[permutation_idx]
    train_y_perm = train_y[permutation_idx]

    for batch in range(NUM_BATCHES_TRAIN):
        train_x_batch = train_x_perm[np.arange(BATCH_SIZE * batch,
    BATCH_SIZE * (batch + 1)), :]
        train_y_batch = train_y_perm[np.arange(BATCH_SIZE * batch,
    BATCH_SIZE * (batch + 1))]

        # ucenje
        softmax_model_out, ent_loss, t_step = sess.run(
                [softmax_model, cross_entropy_loss, train_step],
                feed_dict={X: train_x_batch, Y: train_y_batch})
```

Faze ispitivanja i provjere analogne su fazi učenja, osim što više nije potrebno pozivati optimizator jer je mreža gotova s procesom učenja:

```
# evaluacija
softmax_model_out, ent_loss = sess.run(
    [softmax_model, cross_entropy_loss],
    feed_dict={X: x_batch, Y: y_batch})
```

7. Zaključak

U ovom radu opisani su duboki konvolucijski modeli, s naglaskom na konvolucijske neuronske mreže. Objašnjen je način rada konvolucijske neuronske mreže te slojevi koji ju čine. Prikazana je implementacija mreže pomoću biblioteke Tensorflow, trenutno najmoćnijeg alata za izradu i korištenje konvolucijskih modela. Glavni zadatak ovog rada bio je stvoriti dva modela konvolucijske neuronske mreže te eksperimentalno analizirati njihovu točnost na dva različita klasifikacijska skupa: CIFAR-10 i MNIST. Na skupu MNIST postignuta točnost od 99.06%, a trenutno najbolji rezultat iznosi 99.79% [17], dok je na skupu CIFAR-10 klasifikacijska točnost oko 70% i trenutno najbolji postignut rezultat na svijetu ima čak 96.53% točnu klasifikaciju [6]. Procjenjuje se da na skupu CIFAR-10 čovjek postiže prosječnu točnost od 94% [8]. Pokazano je da korištenje alata Tensorflow omogućuje da se na vrlo jednostavan način programski implementira zamišljenu arhitektura mreže te prati proces njenog učenja i testiranja. Iz svega navedenog očituje se ogroman potencijal konvolucijskih neuronskih mreža koje postižu vrlo visoke klasifikacijske rezultate uz prikladnu arhitekturu mreže, te je stoga razumljivo da su one trenutno najpopularniji alat u području klasifikacije slika.

LITERATURA

[1] Sam Abrahams i Danijar Hafner et al. *TensorFlow For Machine Intelligence*. 2016.

[2] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. 2012. URL https://arxiv.org/pdf/1206.5533.pdf.

[3] D. Meier Ciresan, J. U. Masci, i J. Schmidhuber. Multi-column deep neural network for traffic sign classification. 2012. URL http://people.idsia.ch/~juergen/nn2012traffic.pdf.

[4] Christophe Garcia i Manolis Delakis. Convolutional face finder: A neural architecture for fast and robust face detection. 2004. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1335446.

[5] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Benjamin Graham. Fractional max-pooling. 2014. URL https://arxiv.org/abs/1412.6071.

[7] Andrej Karpathy. CS231n Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io.

[8] Andrej Karpathy. Lessons learned from manually classifying cifar-10. http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/, 2011.

[9] Damir Kopljar. Konvolucijske neuronske mreže. 2016. URL http://ferko.fer.hr/ferko/EPortfolio!dlFile.action?id=350.

[10] Alex Krizhevsky, Ilya Sutskever, i Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. 2012. URL https:

//www.nvidia.cn/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf.

[11] Quoc V. Le, Marc'Aurelio Ranzato, i Rajat Monga et al. Building high-level features using large scale unsupervised learning. 2011. URL https://arxiv.org/pdf/1112.6209.pdf.

[12] Yann LeCun, Yoshua Bengio, i Geoffrey Hinton. Deep learning. 2015. URL https://www.nature.com/nature/journal/v521/n7553/pdf/nature14539.pdf.

[13] Feng Ning, Damien Delhomme, Yann LeCun, Fabio Piano, Leon Bottou, i Paolo Emilio Barban. Toward automatic phenotyping of developing embryos from videos. 2005. URL http://leon.bottou.org/publications/pdf/tip-2005.pdf.

[14] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, i Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. 2013. URL http://soumith.ch/pedestrian-cvpr-13.pdf.

[15] Dario Smolčić. Raspoznavanje objekata konvolucijskim neuronskim mrežama. 2015. URL http://www.zemris.fer.hr/~ssegvic/multiclod/students/smolcic15bs.pdf.

[16] Vedran Vukotić. Raspoznavanje objekata dubokim neuronskim mrežama. 2013. URL http://www.zemris.fer.hr/~ssegvic/project/pubs/vukotic14ms.pdf.

[17] Li Wan i Matthew Zeiler et al. Regularization of neural networks using dropconnect. 2013. URL http://www.matthewzeiler.com/pubs/icml2013/icml2013.pdf.

[18] Siniša Šegvić. Konvolucijske neuronske mreže. Službena predavanja predmeta Duboko Učenje, FER Zagreb. URL http://www.zemris.fer.hr/~ssegvic/du/.

Klasifikacija slika dubokim konvolucijskim modelima

Sažetak

Klasifikacija slika prirodnih scena je neriješen problem računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme najbolji rezultati u tom području postižu se pristupima utemeljenima na dubokim konvolucijskim modelima. Za ovaj rad posebno su zanimljivi nadzirani pristupi gdje je svaka slika skupa za učenje označena semantičkim razredima objekata koje slika sadrži. U ovom radu, korištene su konvolucijske neuronske mreže, trenutno najpopularniji alat u području klasifikacije slika. U okviru rada, proučena je dokumentacija programskog okvira Tensorflow te biblioteke programskog jezika Python za rukovanje matricama i slikama. Izrađena je izvedba programskog sustava za učenje i primjenu jednostavne klasifikacijske arhitekture korištenjem konvolucije, sažimanja, normiranja i potpuno povezanih slojeva.

Ključne riječi: strojno učenje, računalni vid, klasifikacija slika, konvolucijske neuronske mreže, tensorflow

Image classification with deep convolutional models

Abstract

Image classification is currently popular computer vision problem with many interesting applications. Lately, best classification results are acquired by using deep convolutional models. For this paper, reinforcement learning approaches are considered in detail, specifically convolutional neural networks. Convolutional neural networks are currently most popular tool used for image classification. As a part of this paper, Tensorflow library has been combined with Python programming language and other libraries used for image data processing in order to create deep neural model. Two different convolutional neural network models have been created by using convolutional, pooling, activation and fully-connected layers.

Keywords: machine learning, computer vision, image classification, convolutional neural networks, tensorflow