

# Interaktivni projekt s pomočjo uporabe CNN

Matej Hrlec, Gal Kos

Fakulteta za računalništvo in informatiko,  
Večna pot 113, 1000 Ljubljana

**Povzetek** Ker obstaja precej knjižnic, ki uporabljajo konvolucijske nevronske mreže za obdelavo podatkov, smo v okviru tega projekta opravili pregled te panoge strojnega učenja ter implementirali sistem za zaznavo in obdelavo slik s pomočjo knjižnic CNN. Razvili smo klasificiranje objektov v slikah in stiliranje zaznanih regij s prenosom sloga umetniških slik.

## 1 Uvod

V zadnjih letih se je začel nagel razvoj sistemov strojnega učenja, ki temeljijo na simuliraju človeškega razmišljanja. To je v veliki možno zaradi velikega napredka na področju programerskih vmesnikih za izvajanje masovno paralelizirane kode na grafičnih karticah. Poleg tega so se kapacitete grafičnih pomnilnikov in pomnilniških povezav precej povečale, kar omogoča obdelavo velikih podatkovnih baz.

V okviru tega projekta smo se osredotočili predvsem na uporabo konvolucijskih nevronske mreže, bolj natančno na njihovo uporabo pri obdelavi in klasifikaciji slik. Z uporabo in povezovanjem knjižnic, ki podpirajo določene operacije nad slikami, smo implementirali sistem, ki predstavlja praktično uporabo zgornjih funkcionalnosti.

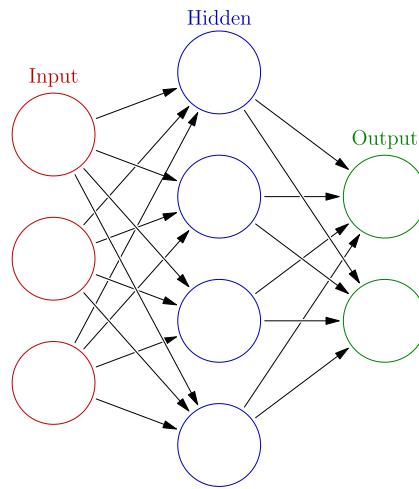
## 2 Opis tematike in pregled področja

### 2.1 Nevronske mreže

Umetne nevronske mreže (angl. *Artificial neural networks* - ANNs) so računski model, ki temelji na veliki zbirki povezav preprostih nevronske enot ozziroma nevronov (angl. *artificial neurons*), ki so po obnašanju sorodni možganskim aksonom[7].

Vsaka nevronska enota je povezana z mnogimi drugimi, povezave lahko povečajo ali zavirajo stanje aktiviranja sosednjih nevronske enot. Posamezna nevronska enota izračunava izhode s pomočjo funkcije seštevanja. Pogosto obstaja tudi pravovalna funkcija (angl. *threshold function*) na vsaki povezavi ali na enoti sami, ki določa stopnjo, ki jo mora signal preseči preden se lahko razširi do sosednjih nevronov. Nevronski sistemi se odločanja naučijo samostojno, tako da niso eksplicitno programirani in se odlikujejo predvsem na področjih, kjer je rešitev ali zaznavanje značilnosti težko izraziti v tradicionalnem računalniškem programu.

Nevronske mreže so običajno sestavljene iz več plasti, pri čemer se signalna pot razteza od prve (vhodne) do zadnje (izhodne) plasti nevronskih enot kot je vidno na sliki 1. Sodobna nevronska omrežja so v smislu simulacije pretočnejša, zato pri tovrstnih omrežjih povezave interaktirajo veliko bolj kaotično in kompleksno. Najnaprednejše so dinamične nevronske mreže, ki lahko na podlagi pravil tvorijo nove povezave in celo ustvarjajo nove nevronske enote ali jih onemogočajo.



**Slika 1.** Vhodna, skrita in izhodna plast pri umetnih nevronske mrežah

Cilj nevronske mreže je reševanje problemov na enak način, kot bi jih reševali človeški možgani, čeprav so veliko bolj abstraktne. Sodobni projekti uporabljajo nevronske mreže, ki so običajno sestavljene iz nekaj tisoč do nekaj milijonov povezav, kar je še vedno velikokrat manj kompleksno od človeških možganov in so bolj podobne računski moči črva.

Zanimiv vidik nevronskega sistemov je, da so nepredvidljivi glede tega, kako uspešni bodo pri samostojnjem učenju. Po tem, ko se naučijo na učnih podatkih, nekateri dobro rešujejo probleme, medtem ko drugi pri reševanju novih problemov niso tako uspešni. Proses učenja običajno zahteva več tisoč ciklov interakcije. Podobno kot pri ostalih metodah strojnega učenja, torej sistemih, ki se učijo iz podatkov, se tudi nevronske mreže uporabljajo za reševanje širokega spektra različnih nalog. Dober primer sta računalniški vid in prepoznavanje govora, ki sta težko rešljiva problema z uporabo tradicionalnega programiranja.

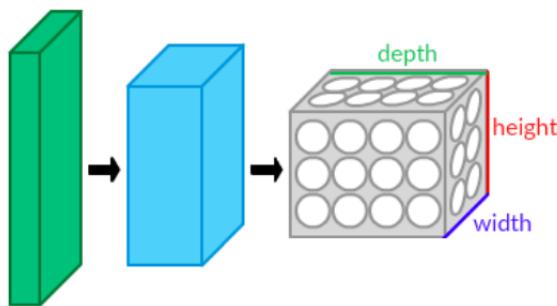
## 2.2 Konvolucijske nevronske mreže

Konvolucijske nevronske mreže (angl. *convolutional neural network* - CNN) so vrsta umetnih nevronske mreže, pri katerih se vzorec povezljivosti med nevron-

skimi enotami zgleduje po ureditvi vidnega korteksa pri živalih[9]. Posamezni kortikalni nevroni se odzivajo na dražljaje v omejeni prostorski regiji oziroma receptivnem polju. Tovrstna polja različnih nevronov se delno prekrivajo, tako da vidno polje razdelijo na ploščice. Odziv posameznega nevrona na dražljaje v svojem receptivnem polju lahko matematično aproksimiramo z operacijo konvolucije. Konvolucijska omrežja izhajajo iz bioloških procesov in so variacije večplastnih zaznav, ki katerih je predprocesiranje minimalno. Konvolucijske nevronske mreže se uporabljajo v aplikacijah za prepoznavo slik in videa, v priporočilnih sistemih in pri obdelavi naravnega jezika.

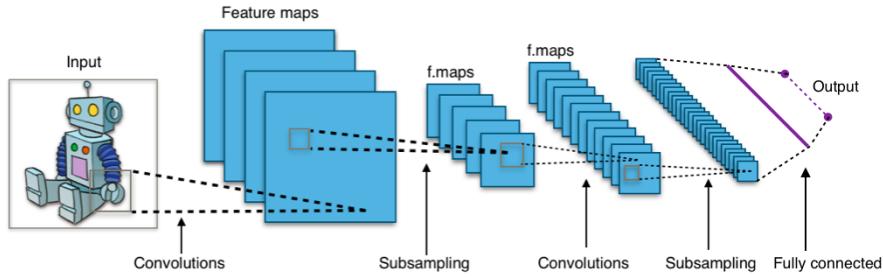
Konvolucijske nevronske mreže so torej različice večslojnih zaznavanj (angl. *multilayer perceptron* - MLP), ki so osnovane tako, da lahko posnemajo vidni kortex. Mreže CNN imajo naslednje značilne lastnosti:

- **3D volumni nevronov:** Sloji CNN imajo nevrone razporejene v treh dimenzijah: širina, dolžina in globina, kot je vidno na sliki 2. Nevroni znotraj sloja so povezani samo z majhnim območjem plasti pred njih - receptivnim poljem. Različni tipi plasti, tako lokalno kot globalno povezani, so nakopičeni, s čimer tvorijo arhitekturo CNN.
- **Lokalna povezljivost:** V skladu s konceptom receptivnih polj mreže CNN izkoriščajo prostorsko-lokalno korelacijo z uveljavljanjem lokalne povezljivosti med nevroni sosednjih plasti. Arhitektura tako zagotavlja, da naučeni filteri ustvarijo najmočnejši odziv na prostorsko-lokalni vhodni vzorec. Kopičenje več takih plasti vodi do nelinearnih filtrov, ki postajajo vedno bolj globalni, torej se odzivajo na vedno večje območje. To omogoča omrežju, da najprej ustvari dobre predstavitev manjših delov vhoda in nato te majhne dele združi v predstavitev večjih področij.
- **Deljene uteži:** Pri mrežah CNN je vsak filter repliciran v celotnem vidnem polju. Te replicirane enote si delijo enako parametrizacijo, torej vektor uteži in začetno vrednost, s čimer tvorijo mapo značilk. Tako vsi nevroni v dani konvolucijski plasti zaznavajo povsem enako funkcijo. Takšno podvajanje nevronskeh enot omogoča, da so lastnosti detektirane ne glede na njihov položaj v vidnem polju, kar predstavlja translacijsko invariantno.



**Slika 2.** Plasti CNN so razporejene v treh dimenzijah (širina, dolžina in globina)

Skupaj zgornje lastnosti omogočajo mrežam CNN doseganje boljše generalizacije problemov. Delitev uteži dramatično zmanjša število prostih parametrov, ki se jih je potrebno naučiti, s čimer znižuje pomnilniške zahteve za delovanje mreže. Zmanjšanje zahtev po virih omogoča učenje večjih in močnejših nevronskih omrežij. Na sliki 3 referenco je vidna tipična arhitektura CNN.



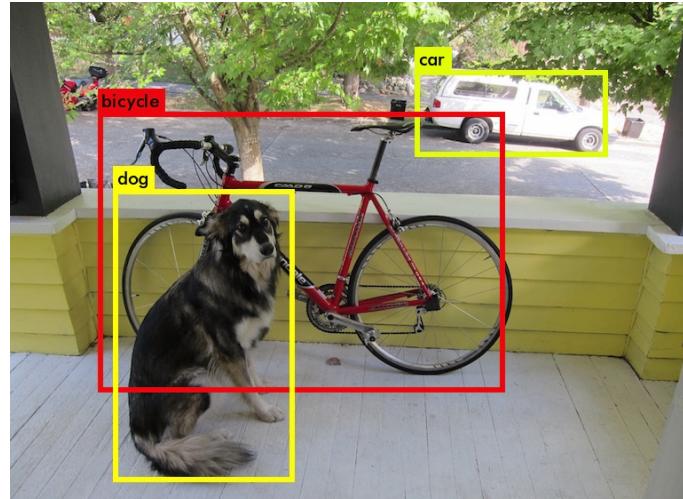
**Slika 3.** Tipična arhitektura konvolucijskih nevronskih mrež

### 2.3 Projekt Darknet

Darknet je odprtokodno ogrodje (angl. *framework*) za delo z nevronskimi mrežami sprogramirano v jeziku C in platformi CUDA (angl. *Compute Unified Device Architecture*)[5]. Ogródje podpira procesorsko in grafično podprto računanje.

Darknet – YOLO (angl. *you only look once*) je realnočasovni sistem za zaznavanje objektov na slikah s pomočjo mrež CNN, ki lahko zazna preko 9000 kategorij objektov.

Sistem uporablja enotno nevronsko omrežje na celotni sliki, s čimer razdeli sliko na regije, napove omejitvena polja in verjetnosti za vsako regijo. Omejitvena polja so utežena z napovedano verjetnostjo. Sistem detektira celotno sliko v času testiranja, zato so napovedi izračunane glede na globalni kontekst slike. Sistem natisne objekte, ki jih je zaznal in izračuna zaupanje (angl. *confidence*) in kako dolgo je trajalo da je objekta našel. V okviru projekta je na voljo več prednaučenih modelov. Primer detekcije je viden na sliki 4.



**Slika 4.** Primer detekcije z uporabo sistema Darknet - YOLO

## 2.4 TensorFlow

S pomočjo uporabe odprtokodne programske knjižnice lahko razvijamo sisteme, ki so sposobni grajenja in učenja nevronskih mrež. S tem lahko zaznavamo vzorce in povezave podobno kot smo to sposobni ljudje.

Arhitektura API-ja omogoča uporabo CPE (centralna procesna enota) ali GPE (grafična procesna enota) na stacionarnih računalnikih, strežnikih in mobilnih napravah. Z njeno uporabo zgeneriramo vozlišča in graf, kjer povezave predstavljajo matematične operacije. Robovi grafov predstavljajo večdimenzionalne nize podatkov, ki med seboj komunicirajo. Ta model lahko potem ostale knjižnjice uporablja za klasificiranje različnih vhodnih podatkov, kot je prepoznavanje govora, objektov v sliki,...

**CUDA** Za razvijalce in znanstvenike so postali moderni GPE zelo zanimivi, saj nudijo dovolj funkcionalnosti in moči, da jih lahko uporabljamo za računsko intenzivne aplikacije. *Nvidia* je za svoje grafične kartice razvila napreden jezik in orodje *CUDA*, ki temelji na standardu *OpenCV*, ki nudi povezavo med izvajanjem kode na CPE in GPE.

Pomemben del računalniškega vida je obdelava slik, področje, za katerega so bili grafični pospeševalci namenjeni. Druge uporabe tudi temeljijo na masovni paralelizaciji računanja in se jih zato pogosto da prenesti na GPE arhitekture z ohranjanjem konceptualne konsistence glede na funkcionalnost kode na CPE.

## 2.5 Prenos sloga slike z uporabo CNN

V zadnjih letih je bilo precej razvoja za prenos sloga z uporabo globokih nevronskih mrež. Gatys et al. [2] so predlagali pristop, ki uporabi inovativen pristop k



**Slika 5.** Primer uporabe prenosa sloga slike z uporabo CNN

zaznavi slog umetniške slike in njenega prenosa na druge slike. S pomočjo tega algoritma ohranjajo slog nanosov s čopičem, geometrijskih oblik in slikarskih struktur. Uporabljajo visoko nivojsko predstavitev lastnosti skrite plasti, ki jo dobimo s CNN. Vsebina in slog slike se podata kot dva ločena vhoda v algoritmom, zato lažje dobimo model umetniškega sloga. To dosežejo z oblikovanjem optimizacijskega problema, ki se začne reševat z belim platnom. Nato po korakih išče novo podobo, ki bi sprožila podobne nevronske aktivacije kot vsebina slike in podobne korelacije lastnosti, izraženo z Gramovo matriko.

## 2.6 Ohranjanje barve slike

Z uporabo prej opisanega postopka 2.5 dobimo kvalitetno preslikavo slikarskega sloga, vendar hkrati izgubimo tudi barve na originalni sliki. Gatys et al. [3] so se lotili tega problema z dvema pristopoma:

- **ujemanje barvnega histograma:** linearen prenos barve izbrane slike na umetniško sliko, preden jo podamo v nevronske mreže. Pri tem smo omejeni s kvaliteto prenosa barv, ki je težji pri bolj različnih slikah. Poleg tega se pri združevanju ohranijo vsebinske strukture slike, predvsem ostre poteze na umetniški sliki.
- **prenos sloga z uporabo svetlosti:** iz obej slik se prenese plast svetlosti, nad katerima se potem uporabi algoritmom nevronske mreže. Izhod potem združimo z barvo originalne slike, s čimer dobimo končno sliko. S tem se v celoti ohranijo barve, vendar se izgubi povezava med plastjo svetlosti in barve pri umetniški sliki. To se opazi predvsem pri močnih potezah čopiča, ki



**Slika 6.** Primer ohranjanje barve pri prenosu sloga slike z uporabo CNN. Na spodnji levi sliki je prikazan postopek z ujemanjem barvnega histograma, na sliki desno spodaj pa prenos sloga z uporabo svetlosti

jim ta postopek lahko da cel spekter barv. S tem postopkom poleg tega optimiziramo tudi grajenje nevronske mreže, saj zmanjšamo nabor parametrov za eno tretjino.

### 3 Implementacija

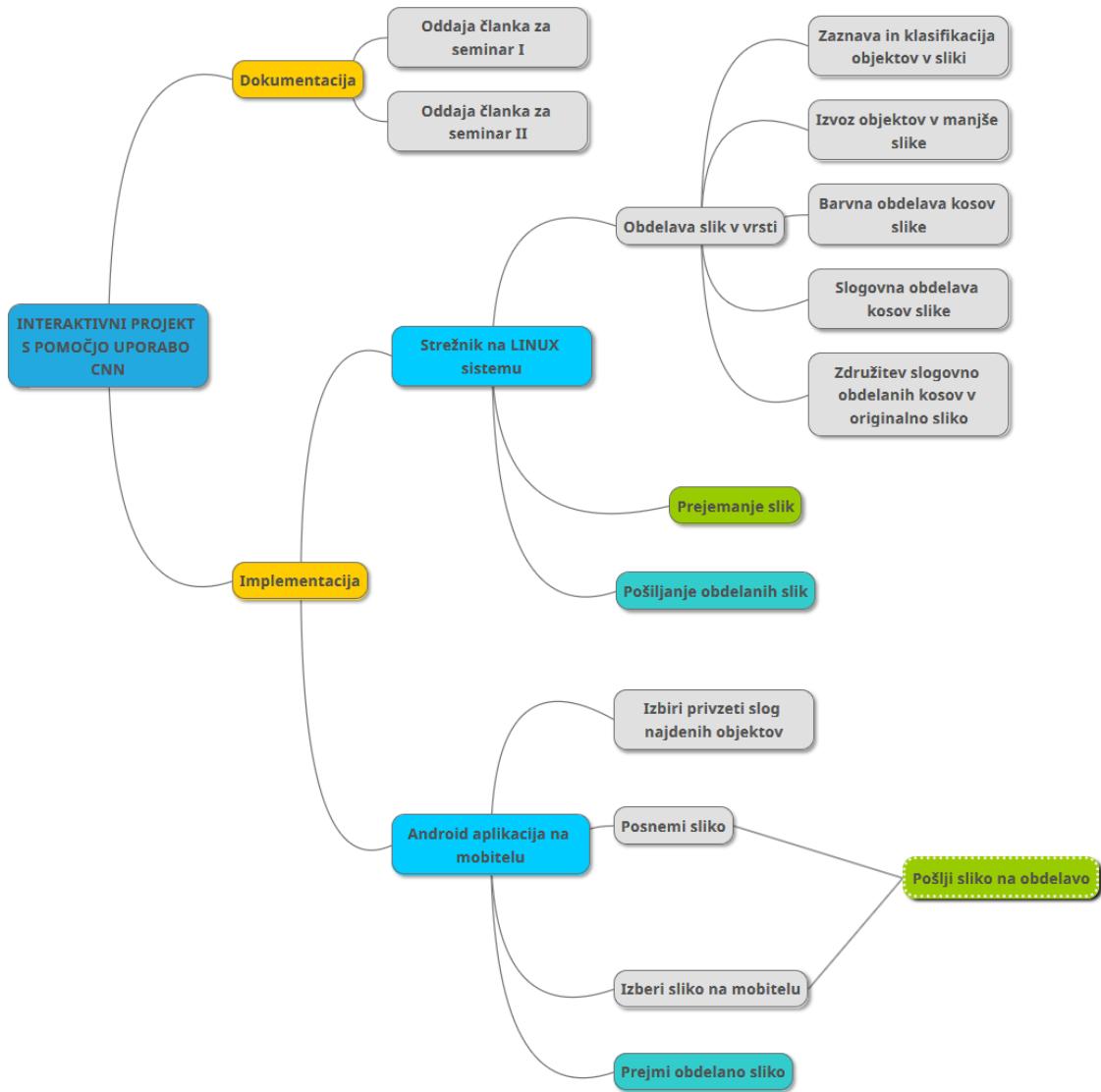
V sklopu tega projekta smo razvili sistem, ki bo s pomočjo knjižnic, ki uporablajo mreže CNN, obdeloval fotografije.

Na čelnem delu (angl. *frontend*) smo razvili *Android* aplikacijo, ki omogoča zajem fotografije s pomočjo kamere ali izbor slike iz galerije naprave.

Ko uporabnik zajame fotografijo oziroma jo izbere iz galerije, aplikacija fotografijo pošlje na strežnik. Tam je fotografija nadalje obdelana s pomočjo različnih zgoraj omenjenih knjižnic mrež CNN.

Sistem je v osnovi sestavljeni iz zalednega in čelnega dela. Zaledni del predstavlja strežnik in sistemi za obdelavo slik z uporabo mrež CNN. Členi del predstavlja aplikacija *Android*. V zaledju (angl. *backend*) smo implementirali strežnik, ki bo stregel zahtevke *Android* uporabnikov in koordiniral uporabo knjižnic CNN. Ko je fotografija na strežniku dokončno obdelana, jo strežnik vrne uporabniku. Rezultat uporabniške akcije je tako lastna fotografija uporabnika obdelana s pomočjo nevronskih mrež.

### 3.1 Miselni vzorec ideje izvedbe projekta



Slika 7. Miselni vzorec ideje izvedbe projekta

### 3.2 Načrt razvoja

Zastavili smo naslednji načrt razvoja:

1. postavitev okolja in namestitev potrebnih ogrodij,
2. implementacija strežnika *Butler*,
3. implementacija sistema *Detector*,
4. integracija sistema *Detector* in strežnika *Butler*,
5. implementacija sistema *Artist*,
6. integracija sistema *Artist*, sistema *Detector* in strežnika *Butler*,
7. implementacija čelne aplikacije *Magician* in
8. testiranje zgornjih sistemov.

### 3.3 Sistem *Butler*

Zaledje sistema smo postavili v okolju *Linux*, pri čemer smo izbrali distribucijo *Ubuntu 16.04*. Strežnik smo implementirali s pomočjo ogrodja *ASP.NET Core*[8], ki je odprtokodno ogrodje namenjeno zaledni strani spletnih aplikacij. Ogorode *ASP.NET Core* deluje na različnih platformah, namenjeno je programiranju s pomočjo programskega jezika C#. Strežnik, ki smo ga poimenovali *Butler*, in odjemalec (angl. *client*) bosta komunicirala preko zahtevkov protokola HTTP (angl. *HyperText Transfer Protocol*).

Ko odjemalec pošlje zahtevek HTTP POST s fotografijo, jo strežnik posreduje ustreznu sistemu CNN, ki fotografijo glede na odjemalčev zahtevek ustrezeno obdela. Ko se stiliranje fotografije zaključi, sistem CNN o uspehu procesiranja obvesti strežnik, ki obdelano fotografijo in metapodatke preko HTTP zahtevka vrne odjemalcu.

Da smo lahko strežnik objavili (angl. *publish*), smo uporabili odprtokodni spletni strežnik Nginx[10], ki podpira objavljanje aplikacij *ASP.NET Core* na platformi *Linux*.

### 3.4 Sistem *Detector*

Na strežniku smo poleg sistema *Butler* postavili dva sistema CNN. Prvi sistem, poimenovan *Detector*, uporablja jedro projekta *Darknet* in je implementiran s pomočjo programskega jezika C in platforme CUDA. Glavna naloga sistema *Detector* je prepoznavanje zanimivih regij oziroma objektov v sliki in predikcija kategorije zaznanega objekta.

Komunikacijo med sistemoma *Butler* in *Detector* smo implementirali s pomočjo preusmerjanja standardnega izhoda in vhoda. Ko se sistem *Butler* postavi, se obenem v ozadju inicializira storitev *DetectorService*, ki zažene sistem *Detector*. Sistem *Detector* ob zagonu naloži uteži v grafični pomnilnik in nato v neskončni zanki čaka na naloge sistema *Butler*. Zgoraj omenjena storitev *DetectorService*, ki skrbi za delegacijo nalog sistemu *Detector*, določa vrstni red operacij s pomočjo podatkovne strukture vrsta (angl. *queue*) po sistemu prvi noter, prvi ven (angl. *First In, First Out - FIFO*). S tem smo programsko zagotovili, da lahko sistem *Detector* naenkrat obdeluje največ eno fotografijo.

V primeru, ko sistem *Detector* v fotografiji zazna več regij, pri združevanju začnemo združevati z regijo pri kateri je bilo zaupanje napovedi tipa zaznanega objekta najmanjše in postopoma nadaljujemo do regij, pri katerih je bilo zaupanje večje. Tako na končni sliki vedno prevlada stil objekta z največjim zaupanjem.

Na sliki 8 je prikazana zaznava regij v sistemu *Detector* in na sliki 9 izrez zaznanih regij.



**Slika 8.** Zaznava regij v sistemu *Detector* iz vhodne slike



**Slika 9.** Izrez zaznanih regij v sistemu *Detector*

### 3.5 Sistem *Artist*

Sistem *Artist* temelji na implementaciji knjižnice TensorFlow v jeziku Python za treniranje hitre nevronske mreže za prenos stila [6]. Uporabljene so podobne

transformacije mreže kot pri [4], ki je sicer implementiran s knjižnico Torch. Normalizacija skupine vhodnih podatkov je narejena z Ulyanovo instanco. Za večjo zmogljivost je bila celotna knjižnica Tensorflow prevedena v binarno kodo na strežniku. Poleg tega smo omogočili še grafično pospeševanje s knjižnicama CUDA in CUDNN, ki ju nudi Nvidia, vendar zahtevata precej zmogljivo grafično kartico z vsaj 4GB grafičnega pomnilnika.

Za apliciranje stilov na zaznane regije sistem uporablja uteži, ki opisujejo potrebne transformacije za pretvorbo. Da stiliranje regij poteka dovolj hitro, sistem uporablja uteži, ki so naučene samo enega specifičnega stila. S sistemom se je sicer možno z predpripravljenimi podatki [1] naučiti novih transformacij s poljubno sliko, vendar ta proces vzame od 4 do 6 ur na *Maxwell Titan X* grafični kartici.

Podobno kot pri sistemu *Detector*, sistem *Butler* delegira naloge sistemu *Artist* s pomočjo vrste (angl. *queue*). Sistem *Artist* začne z apliciranjem sloga, ko mu sistem *Butler* posreduje informacijo, da je nov nabor zaznanih regij v čakalni vrsti pripravljen na obdelavo. Te regije ustvari sistem *Detector* po opravljeni detekciji artefaktov, kjer iz originalne slike izreže zanimive dele in jih posreduje v prej omenjeno čakalno vrsto. Sistemu *Artist* sistem *Butler* posreduje pot do direktorija, ki vsebuje zaznane regije, in pot do uteži želenega stila.

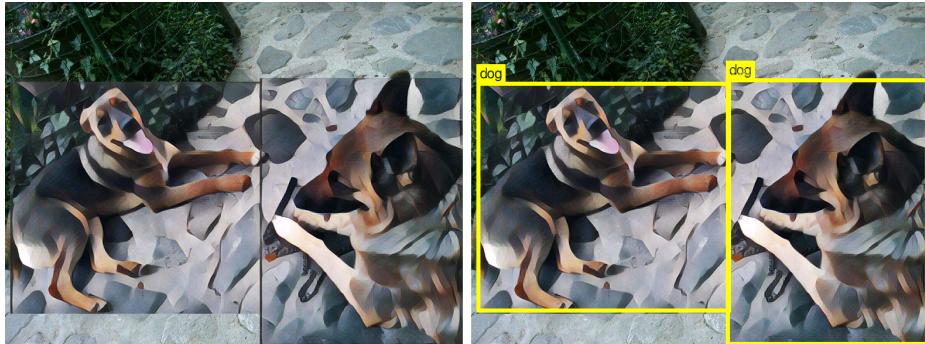
Obdelava posamezne regije se začne z nalaganjem nevronske mreže izbranega sloga v grafični pomnilnik, kjer ostane do konca procesa. Operacija stiliranja tipično traja okoli 5 sekund na grafični kartici *GTX 960M*, časovna zahtevnost je odvisna predvsem od velikosti slike. Poleg tega smo lahko hkrati še omejeni glede maksimalne resolucije, saj nam lahko pri manjši kapaciteti grafične kartice zmanjka grafičnega pomnilnika. Ko je delo zaključeno, se vse slike shranijo na svoje mesto v podatkovni bazi.

Ko je stiliranje zaključeno, sistem *Artist* o tem obvesti sistem *Butler*. Ta regije nato prilepi nazaj na njihovo mesto v originalni sliki in s pravokotniki označi zaznane objekte, ki se naj bi tam nahajali. To združeno sliko sistem *Butler* shrani v podatkovno bazo, s čimer se zaključi obdelovanje slike in obvesti sistem *Magician*, da lahko uporabniku vrne sliko.

Na sliki 10 je prikazano apliciranje sloga na izrezane zaznane regije v sistemu *Artist* in na sliki 11 končno združevanje originalne fotografije in zaznanih regij v sistemu *Butler*.



**Slika 10.** Apliciranje sloga na izrezane zaznane regije v sistemu *Artist*



**Slika 11.** Končno združevanje originalne fotografije in zaznanih regij v sistemu *Butler*

### 3.6 Podatkovna baza

Podatkovno bazo smo implementirali s pomočjo shranjevanja fotografij na disk. Ob vsakem prejetem zahtevku HTTP POST ustvarimo unikatno identifikacijsko številko zahtevka, ki jo določimo s pomočjo časovnega žiga. Nato v mapi, ki predstavlja podatkovno bazo, ustvarimo podmapo, katere ime ustreza zgornji identifikacijski številki. Tako po prejemu zahtevka sistem *Butler* v mapi shrani originalno fotografijo. Ko kasneje sistem *Detector* zaključi z detekcijo, v mapo zahtevka shrani fotografijo z zaznanimi regijami. Prav tako v podmapi *detector* shrani slike, ki predstavljajo posamezne zaznane regije, izrezane iz originalne fotografije.

Nato sistem *Artist* izrezane fotografije iz podmape *detector* eno za drugo obdelava, tako da nanje aplicira željeni slog in jih shrani v novo podmapo *artist*.

Na koncu sistem *Butler* iz originalne fotografije in posameznih fotografij iz podmape *artist* sestavi končno fotografijo, ki predstavlja rezultat obdelave.

V mapi vsakega zahtevka poleg fotografij hranimo tudi objekt *ImageTask*, ki vsebuje metapodatke o posameznem zahtevku. Zgornji objekt je shranjen v odprtokodnem formatu JSON (angl. *JavaScript Object Notation*).

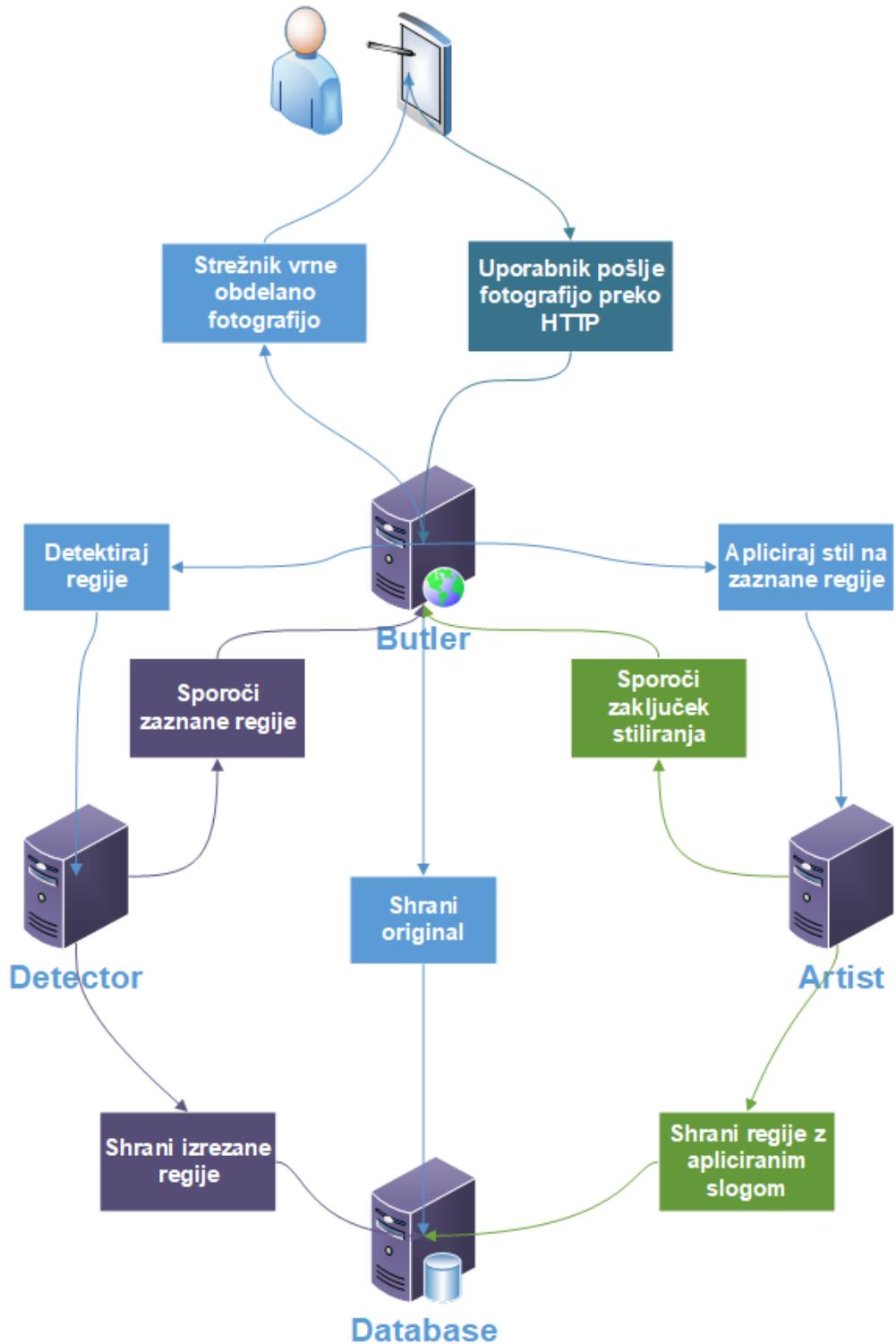
### 3.7 Sistem *Magician*

Uporabnik lahko izkoristi funkcionalnosti implementiranih algoritmov s pomočjo namenske aplikacije *Android*, poimenovano *Magician*. Razvita je z razvojnim okoljem *Android Studio*, in podpira večino naprav, ki uporabljujo ta operacijski sistem.

Ob zagonu aplikacije lahko uporabnik s pomočjo knjižnice za komunikacijo s kamero [11] posname sliko ali pa jo naloži iz galerije naprave. Izbrana slika se nato pošlje na strežnik preko HTTP POST zahtevka. Nato aplikacija *Magician* vsako sekundo pošlje zahtevek HTTP, s katerim jo sistem *Butler* obvešča o trenutnem napredku obdelave. Ko je slika obdelana, jo aplikacija pridobi s strežnika s pomočjo zahtevka HTTP GET in jo prikaže uporabniku. Rezultat operacije je tako obdelana slika, na kateri so razvidne zaznane regije in nanje apliciran slog. Na koncu lahko uporabnik sliko shrani v galerijo naprave.

### 3.8 Shema sistema

Na sliki 12 je vidna shema celotnega sistema in pot enega primera zahtevka uporabnika.



Slika 12. Shema celotnega sistema

## 4 Zaključek in nadaljnje delo

Implementacija strežnika in vseh ostalih sistemov, ki jih uporablja, je precej dobro zastavljena in verjetno ne potrebuje večjih nadgradenj. Ker se pri obdelavi slik uporablja čakalna vrsta, lahko naenkrat strežemo več uporabnikov brez zapletov. Večja omejitev je le pri zmogljivosti strojne opreme, kjer bi bila za hitrejše zaznavanje regij na sliki in njihovo stiliranje zaželjena močnejša grafična kartica. Poleg tega je z učenjem mogoče razširiti paletu stilov pri sistemu *Artist*, vendar je potrebna večurna obdelava s pomočjo grafične kartice.

S sistemom *Artist* je mogoče stilirati tudi videe, tako da bi lahko dodali to funkcionalnost v sistem *Butler*.

Na Android aplikaciji *Magician* bi lahko opravili še veliko dela, zato da bi uporabniku omogočili več izbire pri manipulaciji izgleda končne slike. Možne nadgradnje so:

- izbira stila za vsako najdeno regijo posebej
- možnost odstranitve pravokotnikov in napisov, ki označujejo regije
- spreminjanje meje občutljivosti za zaznavanje regij
- stiliranje celotne slike

## Literatura

1. Baraldi Lorenzo. Vgg-19 pre-trained model for keras. <https://gist.github.com/baraldilorenzo/8d096f48a1be4a2d660d>, 2016. [Dostopno na spletu; dostopano 27.3.2017].
2. L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, June 2016.
3. Leon A. Gatys, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Preserving color in neural artistic style transfer. *CoRR*, abs/1606.05897, 2016.
4. J.C. Johnson. Torch implementation of neural style algorithm. <https://github.com/jcjohnson/neural-style>, 2017. [Dostopno na spletu; dostopano 27.3.2017].
5. Joseph Redmon. Darknet: Open source neural networks in c. <http://pjreddie.com/darknet/>, 2013–2016. [Dostopno na spletu; dostopano 27.3.2017].
6. Lengstrom. Tensorflow cnn for fast style transfer. <https://github.com/lengstrom/fast-style-transfer>, 2017. [Dostopno na spletu; dostopano 27.3.2017].
7. Wikipedia. Artificial neural network — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/wiki/Artificial\\_neural\\_network](https://en.wikipedia.org/wiki/Artificial_neural_network), 2017. [Dostopno na spletu; dostopano 25.3.2017].
8. Wikipedia. Asp.net core — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/wiki/ASP.NET\\_Core](https://en.wikipedia.org/wiki/ASP.NET_Core), 2017. [Dostopno na spletu; dostopano 15.5.2017].
9. Wikipedia. Convolutional neural network — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network), 2017. [Dostopno na spletu; dostopano 25.3.2017].
10. Wikipedia. Nginx — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/wiki/Nginx>, 2017. [Dostopno na spletu; dostopano 15.5.2017].

11. Yuichi Araki. Cameraview. <https://github.com/google/cameraview>, 2017. [Dostopno na spletu; dostopano 15.5.2017].