Kolorizacija fotografija uporabom konvolucijske neuronske mreže
Tehnička dokumentacija
Verzija 1.0

Studentski tim: Leonarda Bojka
Barbara Kos
Matija Pavlović
Toni Polanec
Matea Teglović

Nastavnik: izv. prof. dr. sc. Tomislav Hrkać

Sadržaj

Tehnička dokumentacija

Opis razvijenog proizvoda

1.1 Kolorizator fotografija

Kolorizatori slika imaju širok spektar primjena, neki od primjera su kolorizacija astronomskih fotografija, snimki nadzornih kamera, slika nastalih elektronskom mikroskopijom.

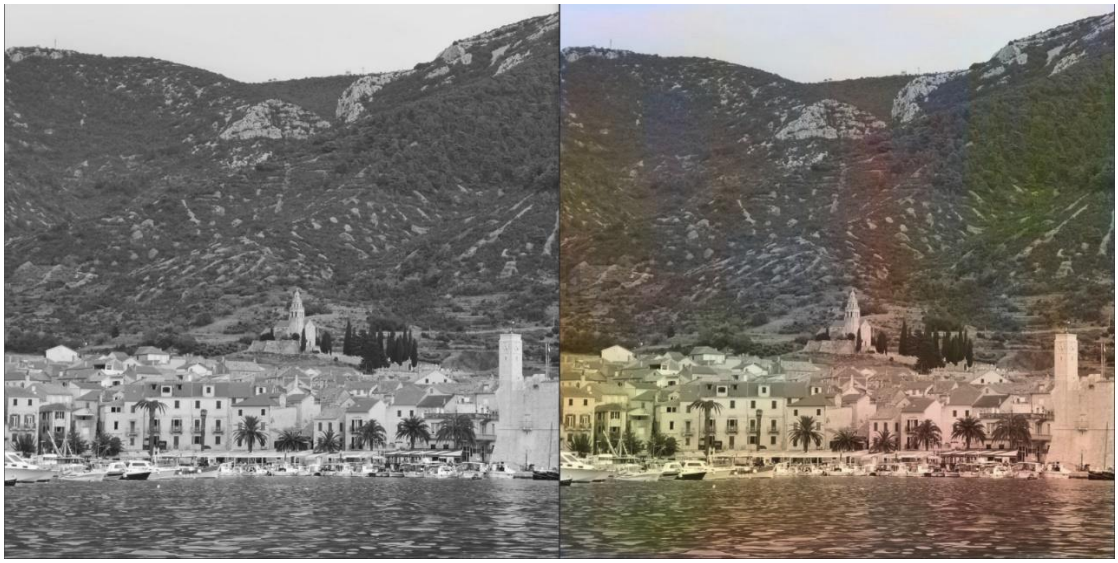
U sklopu ovog projekta razvijen je program u programskom jeziku Python koji se sastoji od modela konvolucijske neuronske mreže s U-Net arhitekturom, generatora skupa podataka za treniranje i testiranje modela, evaluatora dobivenih rezultata, te popratnih programa pokretača.

Razvijeni proizvod je sposoban primiti sliku u LAB formatu, odnosno sa samo jednim kanalom za boju, te iz dobivenog ulaza stvoriti izlaz s tri kanala za boju, odnosno sliku u RGB formatu.

Slika 1 - Prikaz RGB kanala

U RGB prostoru informacija o boji nalazi se unutar tri kanala (crvenom, zelenom, plavom), navedeni kanali se superponiraju te time dobivamo boje fotografije.

Slika 2 - Slika sa superponiranim RGB kanalima



Slika 3 - Jedan od naših prvih primjera kolorizirane slike

Primjer dobiven nakon treniranja dataseta od 217 fotografija kroz 100 epoha uz maksimalnu dubinu mreže od 64 sloja.

Slika 4 - Primjer kolorizirane slike našim modelom

Primjer dobiven nakon treniranja dataseta od 217 fotografija kroz 100 epoha uz maksimalnu dubinu mreže od 128 slojeva.

U nastavku slijede par najboljih generiranih slika pomoću našeg modela:

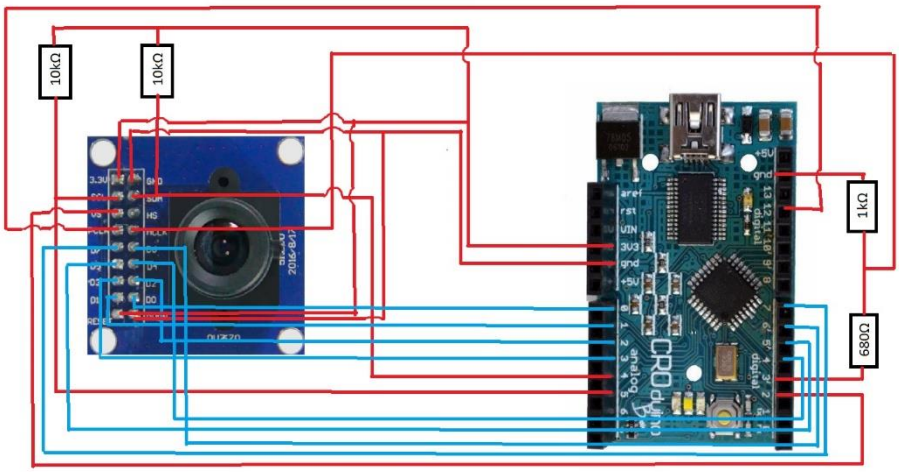
Slika 5 - Kolorizirana slika trave i neba

Slika 6 - Kolorizirana slika planine

Slika 7 - Kolorizirana slika pejzaža grada

1.2 Slike nastale korištenjem OV7670

Na ovom projektu osposobila se CMOS kamera OV7670 uz pomoć mikrokontrolera Croduino Basic. OV7670 je jednostavan niskonaponski sklop s 18 pinova koji radi na 3.3 V te generira slike rezolucije 640 x 480 piksela, dok je Croduino mikrokontroler analogan Arduinu Nanu.



Slika 8 - Shema spajanja OV7670 s Croduinom

3V3	GND	SIOC	SIOD	VSYC	PCKL	XCLK	D7-D4	D3-D0	RESET	PWDN
3V3	GND	10k - A5	10k - A4	D2	D12	1k - 680k	D7-D4	A3-A0	3V3	GND

Tablica 1 – Prikaz spajanja OV7670 s Croduinom

Spajanjem Croduina na računalo dobiva se RGB ili crno-bijela slika, ovisno o tome što smo postavili. Slika je niske rezolucije, dimenzije 640 x 480. Slike se brže generiraju u crno-bijelom načinu rada.

Slika 10 - Primjer dobivene slike s OV7670

Slika 9 - Primjer crno bijele slike

1.3 Kombinacija modela za kolorizaciju i kamere

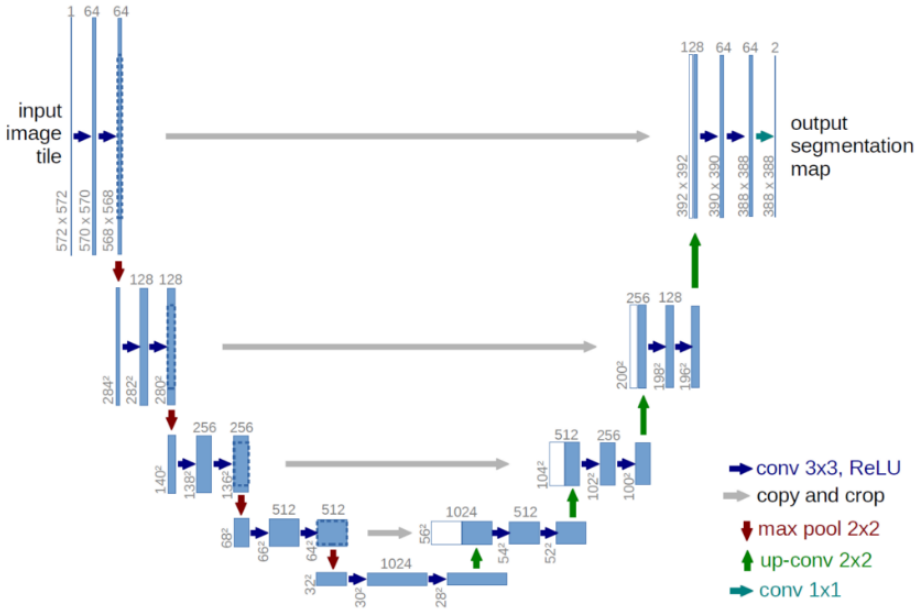
Slika 11 - Obojana slika dobivena kamerom

Kao što je vidljivo na slici, model za kolorizaciju nije baš najbolje rješenje kada su slike manje rezolucije. Međutim, pohvalno je i zanimljivo da je model svejedno uspio razaznati listove i obojati ih u zeleno.

2. Tehničke značajke

2.1 Kolorizator

Kolorizator fotografija koji smo razvili temelji se na UNet Keras modelu konvolucijske neuronske mreže.



Slika 12 - UNet model konvolucijske neuronske mreže

U-Net je konvolucijska neuronska mreža koja je razvijena za segmentaciju biomedicinskih slika na informatičkom odjelu Sveučilišta u Freiburgu.[1] Mreža se temelji na potpuno konvolucijskoj mreži[2], a njezina je struktura izmijenjena i proširena na rad s manje slika osposobljavanja i za postizanje preciznijih segmentacija. Segmentacija slike 512 × 512 slika traje manje od sekunde na suvremenom GPU-u.

[1] Ronneberger O, Fischer P, Brox T (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". arXiv:1505.04597

[2] Shelhamer E, Long J, Darrell T (April 2017). "Fully Convolutional Networks for Semantic Segmentation". IEEE Transactions on Pattern Analysis and Machine Intelligence

```
input = keras.Input(shape=(None, None, 1))
model = Conv2D(8, (3, 3), activation='relu', padding='same', strides=2)(input)
model = Conv2D(16, (3, 3), activation='relu', padding='same')(model)
model = Conv2D(16, (3, 3), activation='relu', padding='same', strides=2)(model)
model = Conv2D(32, (3, 3), activation='relu', padding='same')(model)
model = Conv2D(32, (3, 3), activation='relu', padding='same', strides=2)(model)
model = Conv2D(64, (3, 3), activation='relu', padding='same')(model)
model = Conv2D(64, (3, 3), activation='relu', padding='same', strides=2)(model)
model = UpSampling2D((2, 2))(model)
model = Conv2D(64, (3, 3), activation='relu', padding='same')(model)
model = UpSampling2D((2, 2))(model)
model = Conv2D(32, (3, 3), activation='relu', padding='same')(model)
model = UpSampling2D((2, 2))(model)
model = Conv2D(16, (3, 3), activation='relu', padding='same')(model)
model = UpSampling2D((2, 2))(model)
model = Conv2D(2, (3, 3), activation='tanh', padding='same')(model)
```

Iznad se nalazi implementacija konvolucijske mreže kojom smo trenirali model.

2.1.1 Generiranje dataseta i testnih primjera

2.1.2 Funkcija gubitka i optimizator

Prilikom izrade kolorizatora odlučili smo se za 'mse' funkciju gubitka. Srednja kvadratna pogreška (MSE) možda je najjednostavnija i najčešća funkcija gubitka, koja se često podučava na uvodnim tečajevima strojnog učenja. Da biste izračunali MSE, uzmete razliku između predviđanja vašeg modela i osnovne istine, kvadrirate je i izračunate prosjek za cijeli skup podataka. MSE nikada neće biti negativan, jer uvijek kvadriramo pogreške. MSE je formalno definiran sljedećom jednadžbom:

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Slika 13 - Funkcija gubitka

Optimizacija se vrši uporabom algoritma „Adam“. Na odabir algoritma optimizacije je ponajviše utjecao omjer brzine i memorijske učinkovitosti koja je bila od izrazite važnosti zbog ograničenja sustava na kojem smo vršili treniranje modela. 'Adam', punog naziva 'Adaptive Moment Estimation' je algoritam za optimizaciju gradijentnog spusta. Ova metoda je izrazito korisna pri radu s velikim skupovima podataka. Ako bi smo detaljnije razmatrali ovaj algoritam primjetili bi smo da se u stvarnosti radi o kombinaciji dvije metodologije gradijentnog spusta,

gradijentni spust s momentom i 'RMSP' algoritma.

2.2 Croduino

Budući da je Croduino Basic analogan Arduinu Nanu, za njegovo programiranje može se koristiti razvojna okolina Arduino IDE.

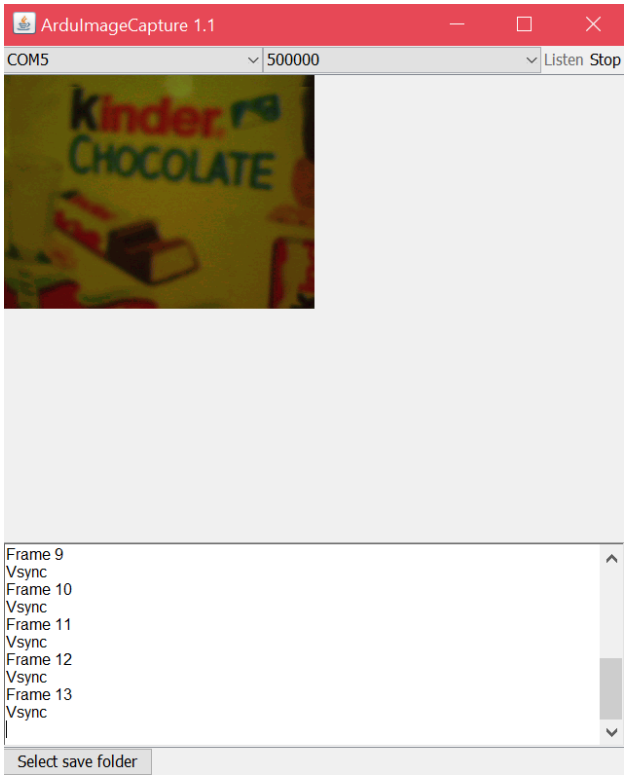
Moguće je odabrati čak 17 različitih načina rada u ovisnosti o rezoluciji i boji.

```
//  1 - 115200bps 160x120 rgb
//  2 - 115200bps 160x120 grayscale
//  3 - 500000bps 160x120 rgb
//  4 - 500000bps 160x120 grayscale
//  5 - 500000bps 320x240 rgb
//  6 - 500000bps 320x240 grayscale
//  7 - 1Mbps 160x120 rgb
//  8 - 1Mbps 160x120 grayscale
//  9 - 1Mbps 320x240 rgb
// 10 - 1Mbps 320x240 grayscale
// 11 - 1Mbps 640x480 grayscale
// 12 - 2Mbps 160x120 rgb
// 13 - 2Mbps 160x120 grayscale
// 14 - 2Mbps 320x240 rgb
// 15 - 2Mbps 320x240 grayscale
// 16 - 2Mbps 640x480 rgb
// 17 - 2Mbps 640x480 grayscale
#define UART_MODE 5
```

Slika 14 - Načini rada kamere

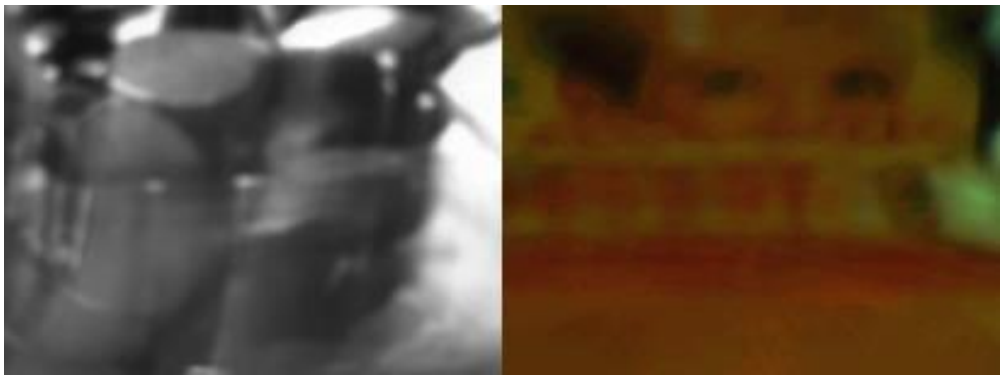
Međutim, vjerojatno zbog procesne snage Croduina, s načinima rada od 7 pa do 17 nije moguće raditi, jer je generirana slika previše deformirana.

Ukoliko se odabere način rada 5 (320 x 240, RGB) te se pokrene kod, kamera će u roku od nekoliko sekundi generirati sliku na računalo.



Slika 15 - Program za generiranje slike

Kameri je potrebno nekoliko sekundi za prilagodbu ekspozicije nakon svake nagle I jake promjene osvjetljenja.
Budući da se slike generiraju relativno sporo u odnosu na brzinu današnjih fotoaparata, tijekom snimanja je potrebno stabilizirati OV7670 ukoliko ne bi došlo do distorzije slike.



Slika 16 - Primjeri slika prilikom nepravilnog rukovanja OV7670

Ukoliko želimo spremiti slike, dovoljno je samo odabrati željenu mapu. U nju će se tijekom snimanja spremati svaka generirana slika.

3. Upute za korištenje

Proces kolorizacije započinje pretvaranjem obojanih slika iz foldera „dataset“ u crno-bijele slike. To radi program RGBToBW.py te slike sprema u folder „dataset_bw“.

```
for image in files:
    img = cv2.imread(os.path.join(path, image))
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    new_img_name = "bw_" + image
    cv2.imwrite(os.path.join(gray_path, new_img_name), gray)
```

Slika 17 – Pretvranje slika u crno-bijele pomoću RGBToBW.py

Nakon toga, Colorizer.py koristi novonastale slike iz foldera „dataset_bw“ i originalne slike iz foldera „dataset“ te od njih stvara dva numpy array-a. Zatim se poziva funkcija „create_model(input)“ iz NeuralNetwork.py koja stvara model na temelju našeg inputa, to jest neuronsku mrežu.

```
def create_model(input):
    model = Conv2D(8, (3, 3), activation='relu', padding='same', strides=2)(input)
    model = Conv2D(16, (3, 3), activation='relu', padding='same')(model)
    model = Conv2D(16, (3, 3), activation='relu', padding='same', strides=2)(model)
    model = Conv2D(32, (3, 3), activation='relu', padding='same')(model)
    model = Conv2D(32, (3, 3), activation='relu', padding='same', strides=2)(model)
    model = Conv2D(64, (3, 3), activation='relu', padding='same')(model)
    model = Conv2D(64, (3, 3), activation='relu', padding='same', strides=2)(model)
    # model = Conv2D(128, (3, 3), activation='relu', padding='same')(model)
    # model = Conv2D(128, (3, 3), activation='relu', padding='same', strides=2)(model)
    model = UpSampling2D((2, 2))(model)
    # model = Conv2D(128, (3, 3), activation='relu', padding='same')(model)
    # model = UpSampling2D((2, 2))(model)
    model = Conv2D(64, (3, 3), activation='relu', padding='same')(model)
    model = UpSampling2D((2, 2))(model)
    model = Conv2D(32, (3, 3), activation='relu', padding='same')(model)
    model = UpSampling2D((2, 2))(model)
    model = Conv2D(16, (3, 3), activation='relu', padding='same')(model)
    model = UpSampling2D((2, 2))(model)
    model = Conv2D(2, (3, 3), activation='tanh', padding='same')(model)

    model = tf.reshape(model, (1, 112, 112, 2))
    model = tf.image.resize(model, [100, 100])
    model = tf.reshape(model, (1, 100, 100, 2))

    return model
```

Slika 18 – Funkcija create_model iz NeuralNetwork.py

Funkcija vraća model i on se dovršava te kompajlira. Sprema se kao „model.h5“.

```
# Finish model
model = keras.Model(input, model)

model.compile(optimizer='Adam', loss='mse')
model.fit(X, Y, batch_size=1, epochs=200, verbose=1)
model.save('model.h5')
```

Slika 19 – Kompajliranje i spremanje gotovog modela

Model se testira pomoću TestModel.py. Učitava se istrenirani model.h5 te crno-bijele slike spremljene u „dataset_bw“ koje program boja i sprema u folder „results“. Na kraju kada imamo istrenirani i testirani model, možemo Demo.py dati path od slike koju želimo obojati pa se pomoću modela predviđaju odgovarajuće boje na slici i prikazuje nam se obojana slika.

```
# Predicting test image and producing output
output = loaded_model.predict(X)
output = np.reshape(output, (100, 100, 2))
output = cv2.resize(output, (ss[1], ss[0]))
```

Slika 20 – Bojanje crno-bijele slike na temelju istreniranog modela

4. Literatura

1. https://becominghuman.ai/auto-colorization-of-black-and-white-images-using-machine-learning-auto-encoders-technique-a213b47f7339
2. https://lukemelas.github.io/image-colorization.html
3. https://www.baeldung.com/cs/sigmoid-vs-tanh-functions
4. https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3
5. Ronneberger O, Fischer P, Brox T (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". arXiv:1505.04597
6. Shelhamer E, Long J, Darrell T (April 2017). "Fully Convolutional Networks for Semantic Segmentation". IEEE Transactions on Pattern Analysis and Machine Intelligence
7. https://www.geeksforgeeks.org/intuition-of-adam-optimizer/
8. http://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf

Slike