Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Проект по Дигитално Процесирање на Слика

Обојување на црно бели слики

Ментор:

Проф. Др. Ивица Димитровски

Изработено од:

Борис Џотов, 185022

Содржина

Вовед

Овој проект го демонстрира користењето и функционирањето на алгоритам за обојување на црно-бели слики, без разлика дали се тоа вештачки црно-бели слики или се така сликани. Со користење на модел кој овозможува предвидување на ab каналите на една слика според каналот L на истата крајниот резултат е предвиден обоен изглед на една црно-бела слика.

За овој проект користам постоечки модел кој ќе биде подетално објаснет понатаму во оваа документација. За развој на овој проект користев Python и Django и дополнителни библиотеки на Python за манипулирање на слики како што е Open-cv.

Пристапување кон проблемот

На прв поглед, да се замислат боите во една црно-бела слика изгледа прилично тешко/невозможно со оглед на тоа што многу од информациите се изгубени. Кога ќе се погледне поубаво може да забележиме дека од семантиката на сцената и нејзината текстура можеме да добиеме многу дополнителни помошни информации за секоја слика, на пример, тревата секогаш е зелена, небото секогаш сино.

Се користи простор на бои наречен Lab. Овој “систем“ од бои се состои од три канали.

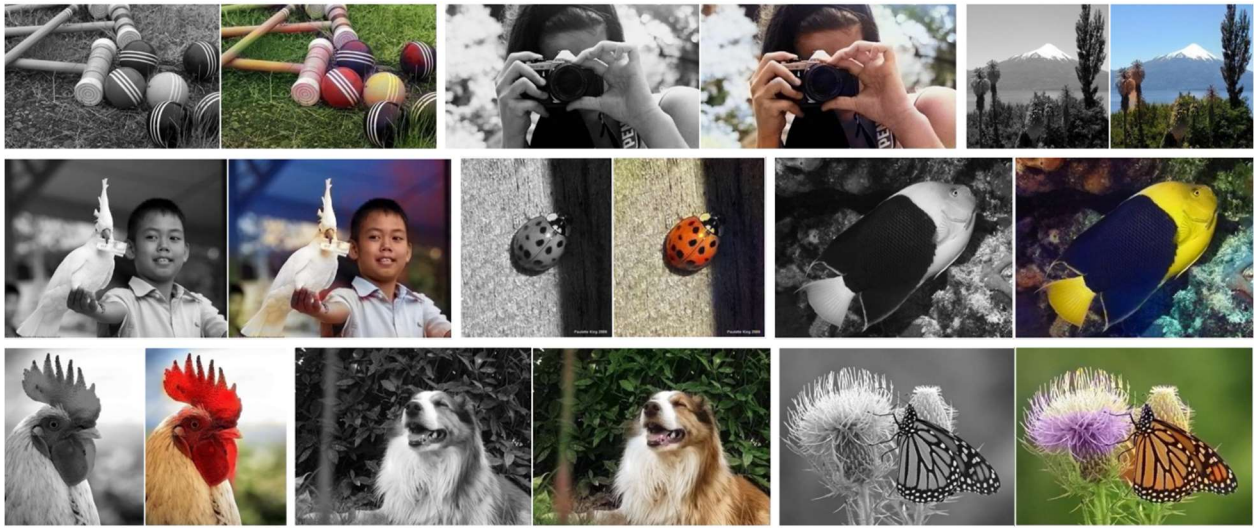
- *L* канал: интензитет на светлина
- *a* канал: зелено-црвено енкодирање
- *b* канал: сино-црвено енкодирање

Од една црно-бела слика се зима вредноста на L каналот и се дава како влез за моделот, кој потоа врз основа на таа вредност ги предвидува останатите вредности од просторот на бои – a и b.

За решавање на овој проблем до сега има повеќе обиди од разни истражувачи, но, повеќето од тие имаат бои кои се прилично ‘потиштени’ (desaturated). Моделот кој го користам јас од Richard Zhang, Phillip Isola, Alexei A. Efros, е малку поразличен и се добиваат подобри резултати со поживи бои и пореалистични за разлика од другите обиди.

На Фигура 1 се прикажани резултати од тестирањето на овој модел кои ги добиле R. Zhang, P. Isola и A. A. Efros.

На Фигура 2 и Фигура 3 се прикажани резултати кои јас ги добив по тестирање на алгоритмот врз мои слики со помош на open-cv.



Фигура 1 - Примери за црно-бели слики и нивните обоени слики – резултати од алгоритмот за обојување

Фигура 2

Фигура 3

Функција за решавање на проблемот

Со даден влез за каналот на интензитет на светлина $\boldsymbol{X} \in \mathbb{R}^{\mathrm{H\ x\ W\ x\ 1}}$ цел на моделот е да научи мапирање $\widehat{\boldsymbol{Y}} = \mathrm{F}(\boldsymbol{X})$ до двете кореспондирачки канали на бои $\boldsymbol{Y} \in \mathbb{R}^{\mathrm{H\ x\ W\ x\ 2}}$ *каде H,W* се димензиите на сликата. Со симболот ^ е означено предвидувањето а без тоа знак е означена оригиналната вистинската обоена слика. Другите моели за овој проблем користат равенка на Euclidean loss помеѓу предвидените и вистинските бои:

$$L_2(\hat{\mathrm{Y}}, Y) = \frac{1}{2}\sum_{h,w} \left|\left|Y_{h,w} - \hat{\mathrm{Y}}_{h,2}\right|\right|_2^2$$

Но, ова не е доволно робустно на природата на проблемот за обојување. Ако еден објект може да прими множество различни ab вредности, оптималното решение на равенката ќе биде средната вредност на множеството. Во предвидувањето на бои, овој ефект на земање средна вредност ги фаворизира сивите, “потиштени“ резултати.

Затоа, овој проблем во моделот кој го користев се решава со мултиномна класификација. Се квантизира ab излезот во множества со големина 10 и се зачувуваат *Q=313* вредности кои се *in-gamut*. За даден влез **X**, се учи мапирањето

$\hat{\mathrm{Z}} = G(X)$ на веројатносна дистрибуција врз можни бои $\hat{\mathrm{Z}} \in [0,1]^{HxWxQ}$ каде *Q* е бројот на квантизирани *ab* вредности.

$$L_{cl}(\hat{\mathrm{Z}}, Z) = -\sum_{h,w} v(Z_{h,w}) \sum_{q} Z_{h,w,q} log(\hat{\mathrm{Z}}_{h,w,q})$$

Каде *v* се користи за ребалансирање на загубата врз основа на реткост на класата на боја.

Евалуација на квалитет на обојување

Овој модел е трениран на 1.3 милиони слики, се валидираат првите 10 илјади од множеството и се тестира на други 10 илјади од истото множество. Квантитативни резултати се прикажани на табелата 1 подолу на три метрики.

За да се тестира специфично ефектот на функциите, конвулуционата невронска мрежа на проектот е тренирана со различни *губитоци*.

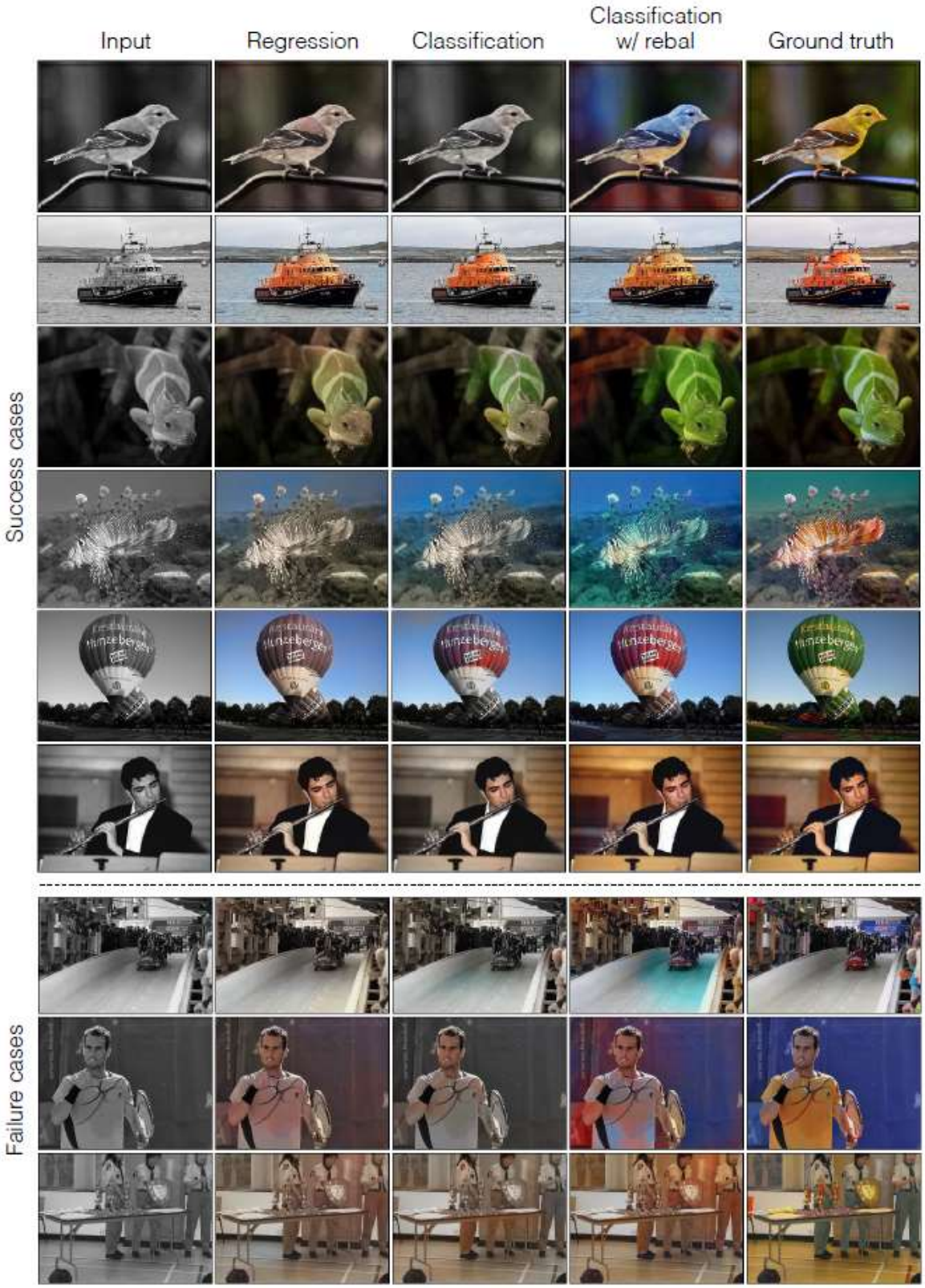
Colorization Results on ImageNet

Method	Model Params (MB)	Model Feats (MB)	Runtime (ms)	AuC non-rebal (%)	AuC rebal (%)	VGG Top-1 Class Acc (%)	AMT Labeled Real (%)
Ground Truth	–	–	–	100	100	68.3	50
Gray	–	–	–	89.1	58.0	52.7	–
Random	–	–	–	84.2	57.3	41.0	13.0±4.4
Dahl [2]	–	–	–	90.4	58.9	48.7	18.3±2.8
Larsson et al. [23]	588	495	122.1	**91.7**	65.9	**59.4**	**27.2±2.7**
Ours (L2)	129	127	17.8	91.2	64.4	54.9	21.2±2.5
Ours (L2, ft)	129	127	17.8	91.5	66.2	56.5	23.9±2.8
Ours (class)	129	142	22.1	91.6	65.1	56.6	25.2±2.7
Ours (full)	129	142	22.1	89.5	**67.3**	56.0	**32.3±2.2**

Табела 1 – Резултати од тестирање на 10 илјади слики од множеството ImageNet. AuC се однесува на плоштината под кривата на дистрибуцијата на кумулативната грешка врз ab просторот. Колона 2 од резултатите ги покажува варијантите на балансирани класи од оваа метрика. Колона 3 е класификациската прецизност по обојување со користење на VGG-16 мрежа. Колона 4 ги покажува резултатите на Amazon Mechanical Turk (AMT) real vs. fake тест.

На табелата погоре – *Табела 1*, се прикажани квантитативно резултатите од тестирањето врз множеството слики. Може да забележиме дека ако алгоритамот даде резултат еднаков на вистинскиот, односно предвидувањето на боите е идентично со оригиналната слика, тогаш алгоритамот ќе постигне 50% успех. За сите метрики поголема вредност е подобро. Може да заклучиме дека моделот на R. Zhang, P. Isola и A. A. Efros е поуспешен од другите со 32.3% точност.

Подолу на *Фигура 4* можеме и визуелно да ги разгледаме резултатите и да се увериме дека боите добиени со овој модел се најреални и најблиску до вистинската слика.



Фигура 4 – Визуелни резултати од тестирањето на множеството од ImageNet.

Заклучок

Обојувањето на црно-бели слики претставува тежок проблем на предвидување на пиксели. Со овој алгоритам е покажано дека обојувањето со длабоки CNN (Convolutional Neural Networks) и со добро избрана функција може да се приближеме до добивање резултати кои се скоро идентични со оние на вистински слики во боја.

Демо апликација

Демо апликацијата ја направив со Django.

Се состои од 3 контролери:

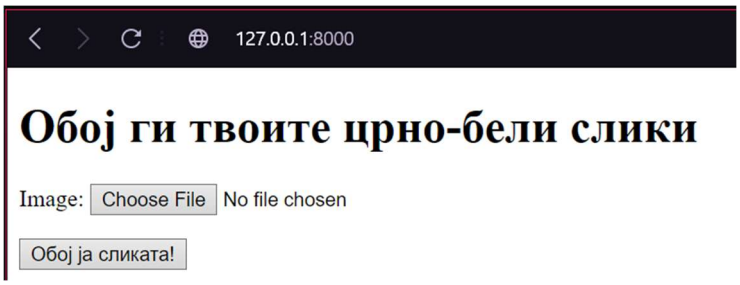
- color_image – функција за обојување на слики.
- color – акција која е одговорна за примање на прикачена слика од корисник и предавање на истата на функцијата.
- color_image_2 – идентична функција на првата со тоа што различно ги прикажува сликите. Овде се споени една до друга со променети димензии за полесен преглед на резултатот.

Апликацијата е поврзана со PostgreSQL база која се користи за чување на податоците за моделите од апликацијата.

Имам креирано еден модел – *Color* кој содржи едно поле – поле за слика.

```
4 usages  Boris Djotov
class Color(models.Model):
    image = models.ImageField(upload_to="colored")
```

Default рутирање на апликацијата води кон обична html страна на која корисникот може да прикачи слика по свој избор и со клик на копчето истата да се процесира низ функцијата и да се прикаже обоена. Прикачената слика се зачувува во фолдер на root на проектот.



Опис на функцијата

Главната замисла на функцијата е од дадената слика да се извади вредноста на каналот за интензитет на светлина – L која ќе се даде како влез на моделот кој ќе ги предвиди вредностите на ab каналите. По ова тие вредности се спојуваат заедно со L каналот и се генерира финалната слика.

На почеток на функцијата се внесуваат патеките до потребните датотеки за моделот - *.prototxt* и *.caffemodel* датотеки. Датотеката со наставка *prototxt* ја содржи структурата на невронската мрежа. Тежините на слоевите на невронската мрежа се зачувани во *caffemodel* фајл.

```
proto_file = r"color_bnw/bnw_assets/colorization_deploy_v2.prototxt"
hull_pts = r"color_bnw/bnw_assets/pts_in_hull.npy"
model_file = r"color_bnw/bnw_assets/colorization_release_v2.caffemodel"
```

Датотеката hull_pts ги претставува точките на Евклидовиот простор.

Се читаат параметрите на моделот.

Се чита сликата од патеката која се дава како аргумент на функцијата. Се скалира и се претвара од BGR во LAB простор на бои.

```
img = cv2.imread(path.image.path)
scaled = img.astype("float32") / 255.0
lab_img = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)
```

Следно се додаваат центрите на кластерите како 1x1 конволуции на моделот.

```
class8 = net.getLayerId("class8_ab")
conv8 = net.getLayerId("conv8_313_rh")
pts = kernel.transpose().reshape(2, 313, 1, 1)
net.getLayer(class8).blobs = [pts.astype("float32")]
net.getLayer(conv8).blobs = [np.full( shape: [1, 313], fill_value: 2.606, dtype="float32")]
```

Се менуваат димензиите на сликата да одговараат на мрежата. Се оддвојува каналот L од сликата и од него се одзима средната вредност.

```
resized = cv2.resize(lab_img, dsize: (224, 224))
# split the L channel
L = cv2.split(resized)[0]
# mean subtraction
L -= 50
```

Се предвидуваат ab каналите со помош на каналот L како влез на функцијата, па се променуваат димензиите на ab волуменот на иста димензија како прикачената слика.

```
# predicting the ab channels from the input L channel
net.setInput(cv2.dnn.blobFromImage(L))
ab_channel = net.forward()[0, :, :, :].transpose((1, 2, 0))
# resize the predicted 'ab' volume to the same dimension as input img
ab_channel = cv2.resize(ab_channel, dsize: (img.shape[1], img.shape[0]))
```

Се зима L каналот од lab сликата и се спојува со предвидените ab канали.

```
# Take the L channel from the image
L = cv2.split(lab_img)[0]
# Join the L channel with predicted ab channel
colorized = np.concatenate( arrays: (L[:, :, np.newaxis], ab_channel), axis=2)
```

Се конвертира сликата од LAB во BGR формат, се променува во ранг 0-255 вредности и се конвертира во int вредности. Се форматираат димензиите и се печатат споени со оригиналната прикачена црно-бела слика.

```
# Convert the img from Lab to BGR
colorized = cv2.cvtColor(colorized, cv2.COLOR_LAB2BGR)
colorized = np.clip(colorized, a_min: 0, a_max: 1)

# change the img to 0-255 range and convert it from float32 to int
colorized = (255 * colorized).astype("uint8")

# resize the image and show together
img = cv2.resize(img, dsize: (640, 640))
colorized = cv2.resize(colorized, dsize: (640, 640))

result = cv2.hconcat([img, colorized])

cv2.imshow( winname: "Grayscale → Color", result)

cv2.waitKey()
```

Се добива следното по кликање на копчето на страната.