

## Vedere Artificială - Tema 4

### Colorarea imaginilor

#### Obiectiv:

Scopul acestui proiect este implementarea (utilizând Tensorflow 2.0) și testarea unui algoritm de colorare a imaginilor (în tonuri de gri) folosind un autoencoder convoluțional.

Funcțiile Python care vă vor ajuta la implementarea proiectului sunt în directorul *cod*; imaginile pe care le veți folosi sunt în directorul *data*.

**Introducere.** Colorarea imaginilor constă în transformarea unei imagini în tonuri de gri în imagine RGB (BGR sau Lab) (Figura 1). Vom construi un autoencoder care va primi ca date de intrare canalul L, similar cu o imagine în tonuri de gri și vom prezice canalele *ab* ale reprezentării *Lab*, suficiente pentru a vizualiza imaginea în culori. Acest tip de autoencoder, se numește *cross-channel-autoencoder*.

**Reprezentarea Lab.** Pană acum, imaginile cu care am lucrat le prezentam în format *RGB* (sau *BGR* în OpenCV). Pentru acest proiect vom converti imaginile în spațiul *Lab*. *L* înseamnă luminozitatea imaginii de la negru (0) la alb 100; *a* de la verde (−128) la roșu (+127); *b* de la albastru (−128) la galben (+127). O reprezentare a unei imagini în spațiul

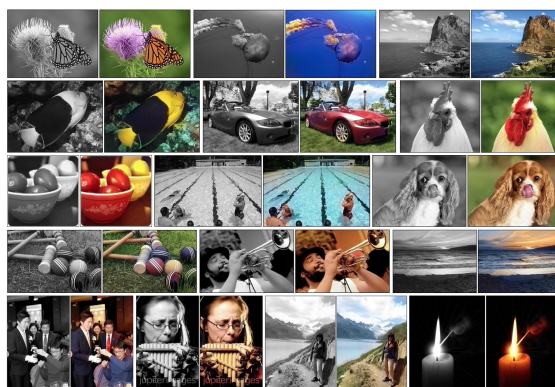


Figura 1: Rezultatele colorării imaginilor în tonuri de gri.



Figura 2: **Prima imagine este imaginea inițială în format *RGB*. A doua imagine reprezintă luminozitatea imaginii (*L*), apoi următoarele doua imagini sunt canalele *a* și *b* din reprezentarea *Lab*.**

*Lab* poate fi vizualizată în Figura 2. Prin urmare, autoencoderul primește canalul *L* și prezice canalele *ab*.

### Implementare:

Primul pas al implementării este instalarea bibliotecii *Tensorflow* rulând următoarea comandă în *Anaconda prompt*:

*pip install tensorflow*

**Autoencoder-ul** pe care-l vom antrena va fi format doar din straturi convoluționale și straturi de *upsampling*, prin urmare vom descrie cum se definește un strat convoluțional în *Tensorflow*.

```
import tensorflow.keras.layers as layers # importam pachetul cu straturi
conv_1 = layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', strides=(2, 2),
padding='same')
```

unde parametrii:

- *filters* reprezintă numărul de filtre aplicate imaginii,
- *kernel\_size* reprezintă dimensiunea filtrului convoluțional,
- *activation* reprezintă funcția de activare,
- *strides* stride-ul de aplicare al filtrului pe cele doua dimensiuni,
- *padding* setat cu valoarea **same** completează input-ului cu coloane/rânduri de 0 până când rezultatul aplicării filtrului va fi egal cu dimensiunea input-ului împărțită la *stride*.

Definirea stratului de *upsampling*:

```
up_1 = layers.UpSampling2D(size=(2, 2))
```

Acest strat redimensionează datele de intrare, folosind interpolarea cu cel mai apropiat vecin, cu factorul *size*.

Pentru antrenarea autoencoderului vom trece prin următoarele etape:

- **citirea datelor**

Imaginile sunt citite în funcția *read\_images* din clasa *DataSet*. Pentru fiecare imagine din mulțimea de antrenare/testare vom citi imaginea în format *BGR*, apoi o vom transforma în format *Lab* folosind următoarea instrucțiune:

```
lab_image = cv.cvtColor(np.float32(bgr_image) / 255, cv.COLOR_BGR2LAB),
```

apoi pentru datele de intrare vom păstra primul canal din formatul *Lab*, iar "etichetele" vor fi canalele *ab*. Canalele *ab* le vom împărți la 128, pentru că vom aplica funcția *tanh* pe ultimul strat al autoencoder-ului (funcția *tanh* ia valori între -1 și 1). Totodată, vom stoca și imaginea în format *BGR* pentru a observa diferențele între imaginea prezisă și cea corectă (ground-truth).

- **definirea autoencoderului**

Pentru a defini modelul, vom folosi clasa *Sequential* din *Tensorflow* și vom seta în constructorul clasei o listă cu straturile autoencoderului ca în exemplul următor:

```
import tensorflow as tf
import tensorflow.keras.layers as layers
```

```
model = tf.keras.models.Sequential([
    layers.InputLayer(input_shape=(32, 32, 1)), # setam input-ul rețelei
    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', strides=(2, 2), padding='same'), # adaugam un strat conv
    layers.UpSampling2D((2, 2))] # adaugam un strat de redimensionare
```

- **compilarea autoencoderului**

După definirea modelului, următorul pas este compilarea lui. Când compilăm modelul trebuie să setăm și *optimizer* – ul (algoritmul de optimizare a ponderilor rețelei), funcția de cost și metoda de performanță. În cazul nostru, nu vom seta și metoda de performanță, deoarece este aceeași cu funcția de cost și anume *MSE* (Mean Square Error). Vom optimiza media pătratelor diferențelor dintre canalele *ab* prezise de rețea și cele corecte.

```
from tensorflow.keras.optimizers import SGD, Adam # importam optimizer-ele
```

```
optimizer = Adam(lr=10 ** -4) # setăm rata de învățare
model.compile(optimizer=optimizer, loss='mse')
```

- **antrenarea autoencoderului**

Pentru antrenarea rețelei, vom defini un *checkpoint callback* prin care vom salva

ponderile rețelei după fiecare epoca, apoi vom antrena modelul apelând funcția *fit*.  
`checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_dir  
+ '/model.epoch:05d.hdf5')`

`model.fit(input_training_images, ground_truth_training_images,  
epochs=num_epochs, batch_size=batch_size, callbacks=[checkpoint_callback])`

Cand apelăm funcția *fit* trebuie să setăm următorii parametri:

- datele de intrare (`input_training_images`) și de ieșire (`ground_truth_training_images`)
- numărul de epoci de antrenare (`num_epochs`)
- numărul de exemple dintr-un batch (`batch_size`)
- callback

- **prezicerea culorilor**

După antrenarea modelului, acesta se va aplica pentru colorarea imaginile de testare. În funcția *evaluate\_the\_model*, vom încărca modelul utilizând funcția *load\_model* și vom specifica epoca pe care vrem să o încărcăm, apoi vom parcurge fiecare imagine din mulțimea de testare, iar cu ajutorul canalului *L* și al modelului antrenat, vom prezice valorile canalelor *ab*. Pentru prezicerea canalelor *ab* vom folosi funcția *predict* (`model.predict(images)`, funcția *predict* așteaptă un batch de imagini). După ce obținem valorile canalelor *ab*, trebuie să le înmulțim cu 128 (deoarece ele vor avea valori între [-1, 1] și trebuie să le aducem în intervalul [-128, 128], apoi vom reconstrui imaginea cu canalul *L* și canalele *ab* prezise. După ce obținem reprezentarea *Lab*, vom converti imaginea în *BGR* cu ajutorul instrucțiunii:

`pred_image = cv.cvtColor(lab_image, cv.COLOR_LAB2BGR) * 255.`

Scriptul *Demo.py* ilustrează antrenarea unui autoencoder cu o anumită arhitectură pe o singură imagine (cea din Figura 2). Mulțimea de antrenare fiind constituită dintr-un singur exemplu rețeaua practic va memora colorarea imaginii (apare fenomenul de overfitting) și nu va generaliza bine pe alte imagini de test.

## 1.1 Predarea proiectului

Arhitectura autoencoder-ului pe care trebuie să-l antrenați este următoarea:

- conv - 64 filtre de dimensiune  $3 \times 3$ , funcția de activare *relu*, stride-ul 2 și *padding* = *same*
- conv - 128 filtre de dimensiune  $3 \times 3$ , funcția de activare *relu*, stride-ul 2 și *padding* = *same*
- conv - 256 filtre de dimensiune  $3 \times 3$ , funcția de activare *relu*, stride-ul 2 și *padding* = *same*
- conv - 512 filtre de dimensiune  $3 \times 3$ , funcția de activare *relu*, stride-ul 1 și *padding* = *same*
- conv - 256 filtre de dimensiune  $3 \times 3$ , funcția de activare *relu*, stride-ul 1 și *padding* = *same*
- upsampling cu factorul 2
- conv - 128 filtre de dimensiune  $3 \times 3$ , funcția de activare *relu*, stride-ul 1 și *padding* = *same*
- upsampling cu factorul 2
- conv - 64 filtre de dimensiune  $3 \times 3$ , funcția de activare *relu*, stride-ul 1 și *padding* = *same*
- upsampling cu factorul 2
- conv - 2 filtre de dimensiune  $3 \times 3$ , funcția de activare *tanh*, stride-ul 1 și *padding* = *same*

Veți primi două baze de date de imagini "coast" și "forest". În clasa *DataSet* există parametrul *network\_input\_size* (setat implicit cu 64) care stabilește dimensiunea imaginilor (imaginile vor fi citite, apoi redimensionate conform acestui parametru; redimensionare se va face în funcția *read\_images*). În clasa *AeModel* există parametrii: *num\_epochs* care setează numărul de epoci pentru care va fi antrenat modelul (setat implicit cu 30); *batch\_size* care setează dimensiunea batch-ului pentru antrenare (setat implicit cu 8); *learning\_rate* care setează rata de învățare a modelului.

Puneți într-o arhivă cu numele *tema4\_cod\_nume\_prenume.zip* codul vostru Python și **checkpoint-urile modelelor**. Puneți într-un document cu numele *tema4\_rezultate\_nume\_prenume.pdf* următoarele:

- (a) **(1 punct)** 3 imagini colorate reușit și 2 imagini colorate nereușit de modelul vostru antrenat pe baza de date "coast" timp de minimum 20 de epoci pe imagini de dimensiune  $64 \times 64$ . Modelul va fi antrenat pe imaginile din subsetul training iar cele 5 imagini colorate colorate reușit sau nereușit vor fi selectate din imaginile din subsetul test.
- (b) **(1 punct)** 3 imagini colorate reușit și 2 imagini colorate nereușit de modelul vostru antrenat pe baza de date "coast" timp de minimum 20 de epoci pe imagini de dimensiune  $128 \times 128$ . Modelul va fi antrenat pe imaginile din subsetul training iar cele 5 imagini colorate colorate reușit sau nereușit vor fi selectate din imaginile din subsetul test.

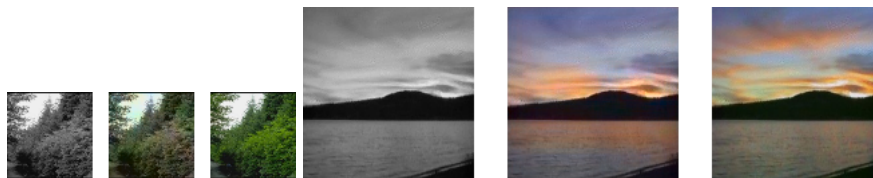


Figura 3: Prima imagine este imaginea grayscale (canalul *L*). A doua imagine reprezintă imaginea prezisă, iar ultima imaginea este imaginea corectă. (stânga) dimensiunea imaginii este de  $64 \times 64$  pixeli. (dreapta) dimensiunea imaginii este de  $128 \times 128$  pixeli.

- (c) **(1 punct)** 3 imagini colorate reușit și 2 imagini colorate nereușit de modelul vostru antrenat pe baza de date "forest" timp de minimum 20 de epoci pe imagini de dimensiune  $64 \times 64$ . Modelul va fi antrenat pe imaginile din subsetul training iar cele 5 imagini colorate colorate reușit sau nereușit vor fi selectate din imaginile din subsetul test.
- (d) **(1 punct)** 3 imagini colorate reușit și 2 imagini colorate nereușit de modelul vostru antrenat pe baza de date "forest" timp de minimum 20 de epoci pe imagini de dimensiune  $128 \times 128$ . Modelul va fi antrenat pe imaginile din subsetul training iar cele 5 imagini colorate colorate reușit sau nereușit vor fi selectate din imaginile din subsetul test.
- (e) **(2 puncte)** 3 imagini colorate reușit și 2 imagini colorate nereușit de modelul vostru antrenat pe un set de date ales de voi timp de minimum 20 de epoci pe imagini de dimensiune de minimum  $64 \times 64$ . La fel ca la punctele anterioare, va puteți împarti setul de date în training și test. Cele 5 imagini vor fi alese din subsetul de test. Setul de date trebuie să fie ales astfel încât să fie potrivit problemei de colorare a imaginilor.
- (f) **(2 puncte)** un video (cu durata recomandată de 5-10 secunde) colorat cât mai reușit de modelul vostru antrenat pe un set de date ales de voi timp de minimum 20 de epoci pe imagini de dimensiune de minimum  $64 \times 64$ . Colorarea întregului video se va face colorând individual fiecare frame din video. Setul de date trebuie să fie ales astfel încât să fie potrivit problemei de colorare a imaginilor.

**\* Fiecare imagine trebuie să fie însoțită de motivația alegerii ei.**

**\*\* Seturile de date alese de voi trebuie să contină minimum 200 de imagini.**

Vom acorda 1 punct din oficiu. Prezentarea temei și explicațiile care însoțesc exemplele din prezentare va fi punctată cu 1 punct.

Trimiteți cele două fișiere (*tema4\_cod\_nume\_prename.pdf* și *tema4\_rezultate\_nume\_prename.pdf*) și fișierul video la adresa de email a lui Bogdan, **bogdan.alexe@fmi.unibuc.ro**.

Termenul limită de predare a temei este sâmbătă, **9 ianuarie 2021**, ora 23:59. Fiecare zi de întârziere în predarea temei se penalizează cu 1 punct în minus. Nu vom accepta teme trimise mai târziu de ziua de luni, 11 ianuarie 2021.