Universitatea din Bucureşti
Facultatea de Matematică şi Informatică
Departamentul de Informatică

Iuliana Georgescu
Radu Ionescu
Bogdan Alexe
Informatică Opţional, anul 3

Vedere Artificială - Tema 5

Colorarea imaginilor

Obiectiv:

Scopul acestui proiect este implementarea (utilizând Tensorflow 2.0) şi testarea unui algoritm de colorarea imaginilor (în tonuri de gri) folosind un autoencoder convoluţional.

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

Introducere. Colorarea imaginilor constă în transformarea unei imagini în tonuri de gri în imagine RBG (BGR sau Lab) (Figura 1). Vom construi un autoencoder care va primi ca date de intrare imaginea în tonuri de gri şi vom prezice canelele *ab* ale reprezentării *Lab*. Acest tip de autoencoder, se numeşte *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 spaţiul *Lab*. L înseamnă luminozitatea imaginii de la negru (0) la alb 100; a de la verde (-128) la roşu ($+127$); b de la albastru (-128) la galben ($+127$). O reprezentare grafică poate fi văzută în imaginea 2.

Prin urmare, autoencoderul va primi canalul L şi vom prezice canalele *ab*.

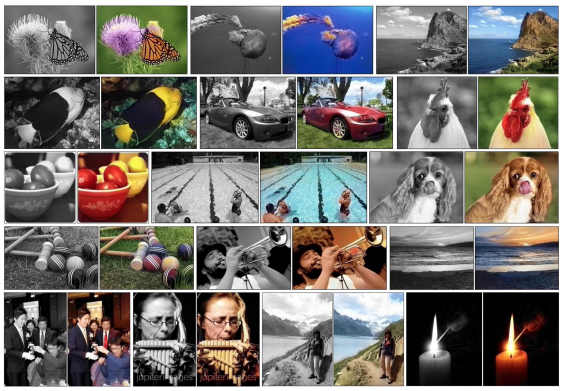

Figura 1: **Rezultatele colorării imaginilor în tonuri de gri.**

Implementare:

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

pip install tensorflow

Autoencoder-ul pe care-l vom antrena va fi format doar din straturi convoluţionale şi straturi de *upsampling*, prin urmare vom descrie cum se defineşte un strat convoluţional 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 convoluţional,
- *activation* reprezintă funcţia 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 împărţită 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:

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

- **citirea datelor:** Imaginile sunt citite în funcţia *read_images* din clasa *DataSet*. Pentru fiecare imagine din mulţimea de antrenare/testare vom citi imaginea în format *BGR*, apoi o vom transforma în format *Lab* folosind următoarea instrucţiune:

  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*. Canelele *ab* le vom împarţii la 128, pentru că vom aplica funcţia *tanh* pe ultimul strat al autoencoder-ului. Totodată, vom stoca şi imaginea în format *BGR* pentru a observa diferenţele între imaginea prezisă şi cea corectă.

- **definirea autoencoderului:** Pentru a defini modelul, vom folosi clasa *Sequential* din *Tensorflow* şi 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 retelei
  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 şi $optimizer - ul$ (algoritmul de optimizare a ponderilor reţelei), funcţia de cost şi metoda de performantă. În cazul nostru, nu vom seta şi metoda de performanţă, deoarece este aceeaşi cu funcţia de cost şi anume *MSE* (Mean Square Error). Vom optimiza media pătratelor diferenţelor dintre canelele *ab* prezise de reţea şi cele corecte.

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

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

- **antrenarea autoencoderului:** Pentru antrenarea reţelei, vom defini un *checkpoint callback* prin care vom salva ponderile reţelei după fiecare epoca, apoi vom antrena modelul apelând funcţia *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 funcţia *fit* trebuie să setăm următorii parametrii:

- datele de intrare (input_training_images) şi de ieşire (ground_truth_training_images)
- numărul de epoci de antrenare (num_epochs)
- numărul de exemple dintr-un batch (batch_size)
- callback

- **prezicerea culorilor:** După antrenarea modelul, vom colora imaginile de testare. În funcţia *evaluate_the_model*, vom încărca modelul utilizând funcţia *load_model* şi vom specifica epoca pe care vrem să o încarcăm, apoi vom parcurge fiecare imagine din mulţimea de testare, iar cu ajutorul canalului L şi al modelului antrenat, vom prezice valorile canalelor ab. După ce obţinem valorile canalelor ab, trebuie să le înmulţim cu 128 (deoarece ele vor avea valori intre [-1, 1] şi trebuie să le aducem în intervalul [-128, 128], apoi vom reconstrui imaginea cu canalul L şi canalele ab prezise. După ce obţinem reprezentarea Lab, vom converti imaginea în BGR cu ajutorul instrucţiunii:

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

1.1 Predarea proiectului

Arhitectura autoencoder-ul pe care trebuie să-l antrenaţi este următoarea:

- conv - 64 filtre de dimensiune 3×3, funcţia de activare $relu$, stride-ul 2 şi $padding = same$
- conv - 128 filtre de dimensiune 3×3, funcţia de activare $relu$, stride-ul 2 şi $padding = same$
- conv - 256 filtre de dimensiune 3×3, funcţia de activare $relu$, stride-ul 2 şi $padding = same$
- conv - 512 filtre de dimensiune 3×3, funcţia de activare $relu$, stride-ul 1 şi $padding = same$
- conv - 256 filtre de dimensiune 3×3, funcţia de activare $relu$, stride-ul 1 şi $padding = same$
- upsampling cu factorul 2
- conv - 128 filtre de dimensiune 3×3, funcţia de activare $relu$, stride-ul 1 şi $padding = same$
- upsampling cu factorul 2
- conv - 64 filtre de dimensiune 3×3, funcţia de activare $relu$, stride-ul 1 şi $padding = same$
- upsampling cu factorul 2
- conv - 2 filtre de dimensiune 3×3, funcţia de activare $tanh$, stride-ul 1 şi $padding = same$

Veţi primi două baze de date de imagini "coast" şi "forest". În clasa *DataSet* există parametrul $network\_input\_size$ (setat implicit cu 64) care stabileşte dimensiunea imaginilor

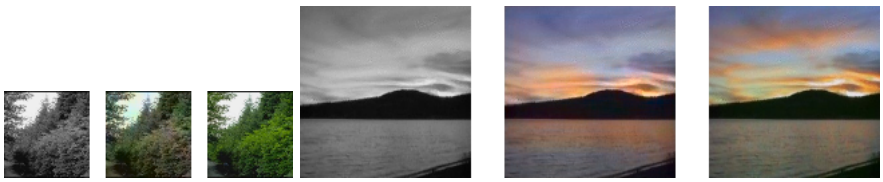

Figura 3: **Prima imagine este imaginea greyscale (canalul L). A doua imagine reprezintă imaginea prezisă, iar ultima imaginea este imaginea corectă. (stânga) dimensiunea imaginii este de 64×64 pixeli. (dreapta) dimensiunea imaginii este de 128×128 pixeli.**

(imaginile vor fi citite, apoi redimensionate conform acestui parametru; redimensionare se va face în funcţia *read_images*). În clasa *AeModel* există parametrii: *num_epochs* care setatează 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 invaţăre a modelului.

Puneţi într-o arhivă cu numele *tema5_cod.zip* codul vostru Python şi **checkpoint-urile modelelor**. Puneţi într-un document cu numele *tema5_rezultate.pdf* următoarele:

(a) (**1 puncte**) 3 imagini colorate corect şi 2 imagini colorante greşit de modelul vostru antrenat pe baza de date "coast" timp de minimum 20 de epoci pe imagini de dimensiune 64×64.

(b) (**1 puncte**) 3 imagini colorate corect şi 2 imagini colorante greşit de modelul vostru antrenat pe baza de date "coast" timp de minimum 20 de epoci pe imagini de dimensiune 128×128.

(c) (**1 puncte**) 3 imagini colorate corect şi 2 imagini colorante greşit de modelul vostru antrenat pe baza de date "forest" timp de minimum 20 de epoci pe imagini de dimensiune 64×64.

(d) (**1 puncte**) 3 imagini colorate corect şi 2 imagini colorante greşit de modelul vostru antrenat pe baza de date "coast" timp de minimum 20 de epoci pe imagini de dimensiune 128×128.

(e) (**2 puncte**) 3 imagini colorate corect şi 2 imagini colorante greşit de modelul vostru antrenat pe o bază de date aleasă de voi timp de minimum 20 de epoci pe imagini de dimensiune de minimum 64×64. Această bază trebuie să fie corespunzătoare task-ului de colorare a imaginilor.

(f) (**2 puncte**) 5 imagini colorate de modelul vostru antrenat pe o bază de date aleasă de voi timp de minimum 20 de epoci pe imagini de dimensiune de minimum 64×64. Această bază trebuie să nu fie corespunzătoare task-ului de colorare a imaginilor.

(g) (**1 puncte**) 3 imagini colorate corect şi 2 imagini colorante greşit de modelul (**definiţi propria voastră arhitectură**) vostru antrenat timp de minimum 20 de epoci pe imagini de dimensiune de minimum 64×64 (puteţi folosi orice bază de date).

** Bazele de date alese de voi trebuie să conţină minimum 200 de imagini.

Trimiteţi cele două fişiere (*tema5_cod.zip* şi *tema5_rezultate.pdf*) la adresa de email a Iulianei Georgescu, **georgescu_lily@yahoo.com** sau **mariana-iuliana.georgescu@my.fmi.unibuc.ro** .

Termenul limită de predare a proiectului este joi, 9 ianuarie 2020, ora 23:59. Fiecare zi de întârziere în predarea proiectului se penalizează cu 1 punct în minus.