

# Kategoriziranje fotografija

Maria Crnjak,  
Luka Jurić-Grgić,  
Antonija Pantar

*Prirodoslovno-matematički fakultet  
Sveučilište u Zagrebu*

**Sažetak – U ovom radu vam predstavljamo rezultate projekta u sklopu kolegija**

**Strojno učenje na Prirodoslovno-matematičkom fakultetu u Zagrebu. Tema je kategoriziranje fotografija koje ćemo raspodijeliti u pet kategorija.**

***Ključne riječi - yelp dataset challenge, photo categorisation, multi-class categorisation, food and drink categorisation, machine learning, deep learning, convolutional neural network.***

## 1. UVOD

U ovom radu odlučili smo spojiti dvije iznimno bitne stvari za cijelo čovječanstvo, tehnologiju i hranu. Sve više ljudi počinje se baviti stvarima vezanim za razvoj same tehnologije i aplikacija. Da bismo upamtili svaki važan trenutak našeg života te ga kasnije sa veseljem gledali, potrebne su nam kvalitetne fotografije. Danas nam je vrlo lako doći do kvalitetnih fotografija. Čak su i kamere mobilnih telefona danas vrlo kvalitetne. To se posebno odnosi na najnovije modele mobilnih telefona koji nas „bacaju“ s nogu

sa svojom kvalitetom slike. Kada vam netko pokaže fotografiju na svome mobilnom uređaju obično odmah znate što se na toj slici nalazi. Zna li to i vaš mobilni uređaj? Dosta ljudi, između ostalog, voli fotografirati hranu i mjesta gdje jedu. Naime, danas postoji mnogo aplikacija koje na temelju fotografije mogu reći nešto o tome što se na fotografiji nalazi - ili ne nalazi. Za naš projekt odlučili smo napraviti sličnu aplikaciju.

Na stranici Yelp objavljen je „Dataset challenge“. Ovaj dataset nas je motivirao na izradu ovakvog projekta. Yelp ovim putem daje priliku studentima da se okušaju u analizi slika ili pak teksta, a najbolji i zarade novac. Ovaj dataset challenge bio je predmet brojnih istraživanja, a neka od njih možemo pronaći na navedenoj stranici. Neki primjeri odgovora na navedeni izazov su projekti poput generatora koji transformira latentne vektore u realistične slike, modela učenja koji su vrlo učinkoviti u modeliranju podataka o tekstu ili govoru, aplikacije koja prepoznaje hot dog-ove i slično. Osim hrane, ovakve aplikacije mogu analizirati i kategorizirati bilo što, što možemo slikati.

Naša namjera je napraviti program koji prepoznaje što se od pet kategorija (*inside*, *outside*, *food*, *drink*, *menu*) nalazi na danoj fotografiji.

Slične komercijalne aplikacije imaju točnost čak do 99% dok većina kodova koje smo našli na „GitHub-u“ postiže između 70 i 99% točnih rezultata. Pretpostavka je da ćemo se pronaći u sredini s obzirom na dosta veliki train set o kojem ćemo kasnije reći nešto više.

## 2. OPIS DATASETA I PROBLEMA ISTRAŽIVANJA

Sa stranice Yelp skinuli smo dataset u kojem se nalazi 200 000 fotografija hrane, pića, restorana i razni podaci o restoranima, korisnicima tih restorana i samim fotografijama. S obzirom na ogroman broj fotografija podrazumijeva se da će našem programu trebati puno vremena da obradi sve slike (ovisno o kvaliteti našeg računala).

Ovaj dataset se sastoji od dva dijela: mape „*photos*“ u kojoj se nalazi 200 000 slika te *.json* datoteke u kojoj su zapisani atributi za svaku pojedinu sliku.

Dimenzije slika se nalaze u rangu od maksimalno 600x600 piksela, s time da je velika većina fotografija dimenzija 400x400 ili 533x400 piksela, što samim time što su dimenzije fotografija ujednačene olakšava posao pri povlačenju fotografija za treniranje i testiranje našeg modela za kategoriziranje. Svim fotografijama je pridruženo 3-4 atributa: *business\_id*, *caption*,

*label*, *photo\_id*, od čega je atribut *caption* (koji sadrži „naslov“ pojedine fotografije) opcionalan; odnosno, pridružen je samo manjem broju fotografija, što povlači da taj atribut sam po sebi nije bitan za točnu kategorizaciju fotografija.

caption	label	photo_id
	inside	MlIA1nNpcp1kDteVg6OGUw
	inside	YjxBE88Bf6CmTEF2LP1UNA
Outside	outside	-bpyOFpGiJsOzh_y17cTMQ
	food	7EcAwdZhYsdz-CSq8ROEBw
"last neighborhood bar in Vegas"	outside	YkW51dD0Hzw1572XLzrV5w

Slika 1: nedostatak "caption" atributa

Atribut *business\_id* sadrži jedinstveni broj identifikacije tvrtke/restorana/mjesta kojima pripadaju fotografije. Analizom dataset-a je utvrđeno da postoji 30 488 različitih *business\_id*-ova, iz čega se može zaključiti da je na nekim mjestima snimljeno više fotografija.

Atribut *photo\_id* očito sadrži jedinstven naziv svake fotografije.

Atribut *label* se dijeli na 5 kategorija: *inside*, *outside*, *food*, *drink*, *menu*, te nam je on najbitniji atribut jer želimo fotografije iz dataset-a točno rasporediti u tih 5 kategorija. Analizom dataset-a je utvrđeno da je svakom od 5 *label*-a, pridružen različit broj fotografija:

<i>Label</i>	<i>No.of photos</i>
<i>drink</i>	18 121
<i>food</i>	114 874
<i>inside</i>	52 448
<i>menu</i>	3 023
<i>outside</i>	11 534

Tablica 1: Udio fotografija po label-ima

U nastavku ćemo navesti primjere fotografija za pojedinu kategoriju, kako bi pokazali raznolikost dimenzija fotografija i onoga što je

prikazano na fotografiji za svaku pojedinu kategoriju:

Slika 2: label - food



Slika 3: label - drink



Slika 4: label - menu



Slika 5: label - inside



Slika 6: label - outside



Naš cilj je napraviti program koji što točnije prepoznaje da li slika koju smo mu dali prikazuje restoran

izvana, iznutra, hranu, piće ili meni restorana.

Kako bismo mogli testirati podatke morat ćemo podijeliti naš dataset na dva dijela: train set (75%) i test set (25%). Train set ćemo koristiti kako bismo "naučili" program kakve se slike nalaze u kojoj kategoriji, a zatim ćemo testirati "znanje" uz pomoć test seta. Na kraju ćemo analizirati našu točnost te ju usporediti s rezultatima popularnih aplikacija. Svi bismo htjeli da to bude idealan problem bez greške, ali s obzirom da je riječ o računalu i velikom dataset-u to je ipak nemoguća misija.

Kada smo započeli istraživanje, prvo što smo uočili je uobičajena veličina data setova koji se proučavaju u ovakvim problemima. Naš data-set je puno veći od bilo kojeg primjera koji smo našli na internetu, a uz to ima samo 5 kategorija. Ovo je prvo što nas je navelo da na neki način smanjimo dataset. Osim toga, od početka smo znali da će nam trebati „jako“ računalo (ali nismo znali koliko jako).

Još jedan problem u našem dataset-u je neizbalansiranost broja slika u pojedinoj kategoriji (Tablica 1).

Na kraju smo odlučili pokušati na tri različita dataset-a (podskupa našeg dataset-a):

Prvi dataset na kojem smo pokušali testirati naš kod je prvih 10% fotografija u mapi *photos* (mapa fotografija iz našeg dataset-a). Train set je prvih 75% ovog dataset-a, a test set posljednjih 25%. Ukupan broj fotografija u ovom dataset-u je 20000 (*train\_small*: 15 000, *test\_small*: 5 000).

Drugi dataset na kojem smo željeli testirati svoj kod je „samo“ 10 000 fotografija. Ovo je i dalje više od mnogo dataset-ova koji su testirani na internetu. Htjeli smo vidjeti što će se dogoditi ako ubacimo balansirani skup podataka. Ovaj dataset je taj balansirani skup. U njemu je točno 1500 fotografija svake skupine u train setu (*train\_balanced*; ukupno 7 500 fotografija) i 2500 fotografija u test setu (*test\_balanced*).

Na kraju, odlučili smo pokušati testirati početni dataset u cijelosti (uz train: prvih 75%, test: posljednjih 25% fotografija u mapi *photos*).

### 3. OPIS METODE

Izbor samih metoda je iznimno bitan, ponajviše radi velikog broja fotografija i vremena potrebnog računalu da ih „izvrti“. Odlučili smo se detaljno posvetiti analizi podataka kako bismo mogli što bolje odrediti parametre funkcija koje smo koristili u kodu.

Nismo si mogli dopustiti prevelik broj testiranja zbog manjka vremena i prikladne tehnologije.

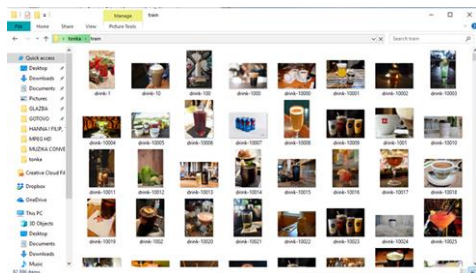
Kako smo prije spomenuli, naš dataset treba biti podijeljen na train i test set, međutim, Yelp dataset fotografija je samo jedan veliki skup fotografija, a ne dva. Dakle, prvo što trebamo je razdvojiti ovaj skup u train i test set. To smo učinili na tri načina. Posebno za svaki od gore spomenutih podskupova cijelog dataset-a. Prvo smo morali razdvojiti cijeli dataset na dva velika train i test seta kako ne bismo miješali podatke (a i kasnije ćemo koristiti ovu podjelu za testiranje cijelog skupa). Ovo smo učinili

pomoćnim programom *naming\_images.py*. Ostala dva naming programa su namijenjena za druga dva, manja, dataseta. *naming\_small.py* razdvaja prvih 10% foldera *photos* na *train\_small* i *test\_small* (20 000 slika). *naming\_balanced.py* uzima prvih 1500 slika od svake kategorije iz velikog train seta i sprema ih u *train\_balanced* (ukupno 7500 fotografija) te prvih 2500 fotografija iz velikog test seta i sprema ih u *test\_balanced*.

Osim train i test seta imamo i *validation set* koji je uvijek 20% od train seta. Validation set se koristi kako bismo validirali (potvrdili) naše trenirane podatke. Malo pre kasno smo shvatili da nam nije bio potreban validation data ako imamo test data čiju točnost rezultata možemo provjeriti, ali pokazalo se korisnim ipak imati validation data. Naime, naš kod se nije proveo do kraja zbog jedne greške u učitavanju podataka, ali nismo imali vremena za više testova (izvršavanje koda trajalo je ukupno oko 12 sati + još jedan neuspjeli pokušaj na pre slabom računalu). Međutim, zato što smo imali validation data, mogli smo vidjeti točnost svoje procjene.

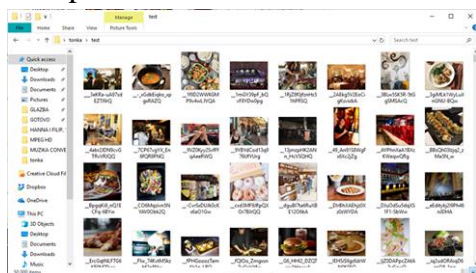
Vratimo se pomoćnim *naming* programima. Zovemo ih *naming* programi jer, osim što razvrstavaju fotografije u prikladne datoteke, odmah daju i prikladno ime svakoj slici. Svaka slika imat će ime oblika: *label-br* (gdje je *label* jedna od kategorija, a *br* redni broj slike u toj kategoriji).

Na primjer, mapa train izgleda ovako:



Slika 7: Train mapa

a mapa test ovako:



Slika 8: Test mapa

Nakon toga posvetili smo se standardizaciji veličine slika kako bismo ih lakše učitali u model. Također, dobro je smanjiti dimenzije fotografija kako ne bi došlo do *overfitting-a* (model je pre dobro naučen na train data).

S druge strane, premali broj značajki dovodi u opasnost od *underfitting-a*. Pošto smo imali vrlo velik dataset nismo se bojali *underfitting-a* - naravno - što veći skup podataka imamo, to ćemo bolje moći procijeniti. Cilj nam je bio smanjiti fotografije na veličinu koja će biti "dobrih" proporcija za sve fotografije. Odlučili smo se za veličinu između prosječne visine i širine. Ovo smo računali uz pomoć programa *prosjecna\_velicina.py*.

Kad smo izveli cijelu pripremu možemo pokrenuti naš program koji učitava train data te automatski čita *label* iz imena svake slike, na svaku sliku primijeni par promjena (antialiasing, crno bijelo, promijeni veličinu na prosječnu (430)). Nakon

toga se postavlja i pokreće *keras Sequential* model o kojemu ima govora u idućem poglavlju. Na kraju se učitava test data i izračunava točnost procjene. Također se ispisuje *matrica konfuzije*. (Posljednje dvije rečenice nismo uspjeli ostvariti. Na žalost, na kraju izvođenja javila nam se greška.)

## 4. CNN MODEL

Za naš problem klasifikacije fotografija u 5 kategorija, odlučili smo trenirati *CNN model*, odnosno, model zasnovan na konvolucijskoj neuronskoj mreži. Razlog zbog kojeg smo se odlučili baš za konvolucijsku neuronsku mrežu jest taj što se s otkrićem iste ubrzao i poboljšao proces klasifikacije slika jer za takav *CNN model* nije potrebno preprocesirati podatke i izvlačiti bitne *feature*, već sama mreža koristi piksele uvezene slike kako bi naučila što oni predstavljaju i u kakvoj su vezi svi pikseli koji su prošli kroz nju.

Treniranje i testiranje našeg modela proveli smo koristeći programski jezik *Python*, *Jupyter* te *Anacondu* sa *Spyder* sučeljem koristeći *Keras framework*.

Funkcije koje smo koristili iz *Keras libraryja*:

- `from keras.models import Sequential`
- `from keras.layers import Dense, Dropout, Flatten`
- `from keras.layers import Conv2D, MaxPooling2D`
- `from keras.layers.normalization import BatchNormalization`



Uzeli smo *sequential model* jer se radi o linearno naslaganim slojevima. Dodali smo *Dropout layer* kako bi prevenirali *overfitting* train seta. *Conv2D layer* stvara 2D konvolucijski sloj, a *MaxPooling2D layer* služi za smanjenje dimenzionalnosti prostornih podataka. *BatchNormalization layer* služi za normalizaciju svih slojeva neuralne mreže kako bi mreža mogla brže učiti.

S obzirom na ograničenost dostupne tehnologije (privatni laptop i računala), odlučili smo provesti 3 neovisna treniranja i testiranja:

Ograničen	Balansiran	
1 337	1 500	Drink
8 586	1 500	Food
3 990	1 500	Inside
225	1 500	Menu
862	1 500	Outside
5 000	2 500	Test
<b>20 000</b>	<b>10 000</b>	<b>Ukupno</b>

Tablica 2: Udio fotografija po label-ima i dataset-ovima

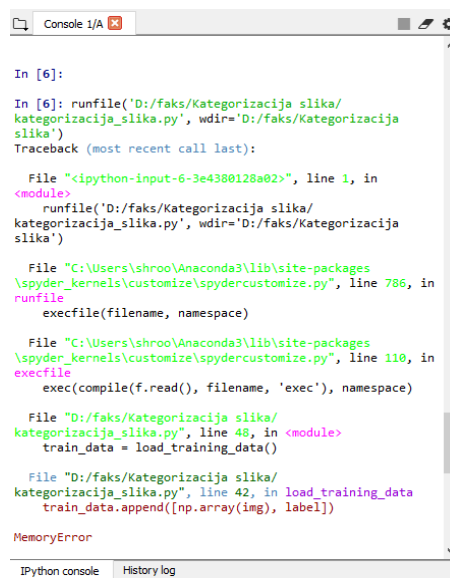
### 1. Ograničen dataset

Originalan dataset od 200 000 slika smo ograničili na 20 000 slučajno odabranih slika iz dataset-a, kako bi mogli istrenirati naš model i provjeriti radi li te s kolikom točnošću kategorizira zadane fotografije. Pri tome smo se držali omjera 3:1 za train i test set (15 000 train i 5 000 test fotografija) koji su bili podijeljeni u mape *train\_small* i *test\_small*.

### 2. Originalan dataset

U prethodno treniranu mrežu na manjem dataset-u smo planirali ubaciti čitav dataset od 200 000 slika kako bi

usporedili točnost kategorizacije s obzirom na količinu dostupnih informacija; međutim treniranje modela nije bilo moguće na nama dostupnoj tehnologiji jer je treniranje prethodnog manjeg testa na *high-end* laptopu (Asus rog strix scar II) trajalo više od nekoliko sati. Na slabijem je računalu izbacio *Memory error*.



```

In [6]:
In [6]: runfile('D:/faks/Kategorizacija slika/
kategorizacija_slika.py', wdir='D:/faks/Kategorizacija
slika')
Traceback (most recent call last):

  File "<ipython-input-6-3e4380128a02>", line 1, in
<module>
    runfile('D:/faks/Kategorizacija slika/
kategorizacija_slika.py', wdir='D:/faks/Kategorizacija
slika')

  File "C:\Users\shroo\Anaconda3\lib\site-packages
\spyder_kernels\customize\spydercustomize.py", line 786, in
runfile
    execfile(filename, namespace)

  File "C:\Users\shroo\Anaconda3\lib\site-packages
\spyder_kernels\customize\spydercustomize.py", line 110, in
execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "D:/faks/Kategorizacija slika/
kategorizacija_slika.py", line 48, in <module>
    train_data = load_training_data()

  File "D:/faks/Kategorizacija slika/
kategorizacija_slika.py", line 42, in load_training_data
    train_data.append([np.array(img), label])

MemoryError

```

Slika 9: Memory error

### 3. Balansirani dataset

S obzirom na neproporcionalne veličine kategorija, napravili smo poseban balansirani dataset u kojem je svaka kategorija sadržavala 1 500 slika, ukupno 7 500 slika za treniranje (u mapi *train\_balanced*) te dodatnih 2 500 za testiranje modela (u mapi *test\_balanced*) kako bi usporedili točnost klasificiranja našeg modela obzirom na proporcionalne i

neproporcionalne veličine dostupnih podataka.

## 5. REZULTATI

Arhitektura računala na kojem smo trenirali i testirali naš model je sljedeća: *procesor Intel CORE i7 8750H, 2.20GHz; 16GB RAM-a memorije; grafička kartica Nvidia GForce RTX 2070 s 8GB VRAM-a; Asus rog strix scar II laptop.*

Trening i testiranje našeg modela kroz 5 epoha na ograničenom dataset-u trajalo je oko 4 sata.

```
Train on 12000 samples, validate on 3000 samples
Epoch 1/5
- 1840s - loss: 0.7906 - acc: 0.7446 - val_loss: 0.7447 - val_acc: 0.7777
Epoch 2/5
- 1794s - loss: 0.5030 - acc: 0.8258 - val_loss: 0.9657 - val_acc: 0.7600
Epoch 3/5
- 1779s - loss: 0.3811 - acc: 0.8644 - val_loss: 1.5221 - val_acc: 0.6600
Epoch 4/5
- 1797s - loss: 0.2862 - acc: 0.8979 - val_loss: 0.4350 - val_acc: 0.8567
Epoch 5/5
- 1809s - loss: 0.2125 - acc: 0.9242 - val_loss: 0.6491 - val_acc: 0.8123
```

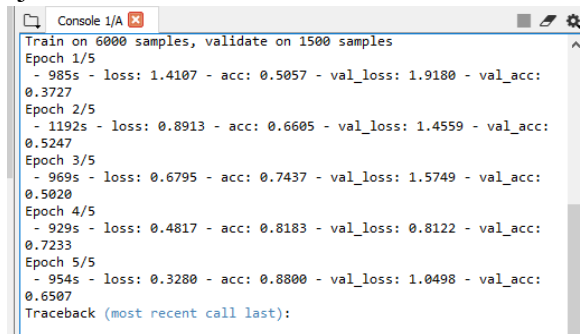
Slika 10: Točnost kategorizacije 1. dataseta

Kao što je vidljivo sa slike iznad, točnost (*acc*) train seta se povećava sa svakom sljedećom epohom, dok točnost (*val\_acc*) validacije drastičnije opada u trećoj i petoj epohi. U trećoj se epohi dogodio *underfitting* u validaciji zbog funkcije *dropout*, dok u petoj epohi možemo pretpostaviti da se dogodio *overfitting* podataka iz train seta, što je onemogućilo dobru kategorizaciju dosad našem modelu neviđenih fotografija u odgovarajuće kategorije.

Valja zaključiti kako bi u konačnom modelu trebalo biti samo 4 epohe kako bi naš model najbolje kategorizirao podatke (točnost train seta: 90%, točnost validacije: 86%).

Nadalje, nakon treninga modela na balansiranom dataset-u, na istoj

arhitekturi računala, dobiveni su sljedeći rezultati:



```
Console 1/A
Train on 6000 samples, validate on 1500 samples
Epoch 1/5
- 985s - loss: 1.4107 - acc: 0.5057 - val_loss: 1.9180 - val_acc: 0.3727
Epoch 2/5
- 1192s - loss: 0.8913 - acc: 0.6605 - val_loss: 1.4559 - val_acc: 0.5247
Epoch 3/5
- 969s - loss: 0.6795 - acc: 0.7437 - val_loss: 1.5749 - val_acc: 0.5020
Epoch 4/5
- 929s - loss: 0.4817 - acc: 0.8183 - val_loss: 0.8122 - val_acc: 0.7233
Epoch 5/5
- 954s - loss: 0.3280 - acc: 0.8800 - val_loss: 1.0498 - val_acc: 0.6507
Traceback (most recent call last):
```

Slika 11: Točnost kategorizacije 3. dataseta

Očekivali smo da će rezultati balansiranijeg skupa podataka biti točniji, no iznenadili smo se kada smo vidjeli koliko su ovi rezultati lošiji od prvog pokušaja.

Kao što je vidljivo sa iznad, točnost (*acc*) train seta se opet povećava sa svakom sljedećom epohom, međutim, ovaj puta raste samo do 88% (i to u 5. epohi kada je već došlo do *overfitting-a*). Ovdje bismo također stali na 4. epohi (točnost train seta: 81.83%, točnost validacije: 72.33%).

## 6. ZAKLJUČAK

Rezultati testiranja govore nam da je bolje trenirati model na većem neuravnoteženom skupu podataka nego na manjem potpuno uravnoteženom skupu podataka. Iz ovoga na kraju ne možemo puno zaključiti, osim možda da smo uzeli premali uravnoteženi skup za usporedbu. Na žalost nismo imali mogućnost napraviti veći skup uravnoteženih podataka. Naime, u cijelom dataset-u bilo je premalo fotografija sa oznakom *menu* i bili smo time ograničeni. Mogli smo uzeti manji skup podataka za početni (neuravnoteženi) skup tako da odgovara uravnoteženom skupu. Na taj način dobili bismo točnija

mjerenja. Jedino što možemo zaključiti iz danih podataka je da će nam manji skup podataka dati lošije rezultate.

Kako zbog ograničene tehnologije nismo bili u stanju testirati originalan dataset s puno više podataka, nismo uspjeli dobiti bolje rezultate. Na to je utjecala i vremenska ograničenost: nismo stigli pokrenuti program više puta jer nismo imali stalan pristup prigodnom računalu. U međuvremenu smo testirali svoj kod za greške na osobnim laptopima i računalima. Ova posljednja greška nam je promakla jer bi se na našim računalima dogodio *memory error* prije no što bi kod došao do te etape. Sigurni smo da bismo uspjeli do kraja izvršiti željeni kod da smo na vrijeme uspjeli doći do potrebne tehnologije, odnosno kada bismo mogli testirati kod još nekoliko puta.

Bez obzira na grešku uspjeli smo dobiti rezultate. Na kraju nismo dobili rezultate za test data već za validation data. Prvi pokušaj, 20 000 slika ispaio je prilično dobro. Kao što smo i predvidjeli, veličina skupa podataka pomogla nam je u točnosti i uspjeli smo postići više no što smo očekivali. Zadovoljni smo rezultatima jer smo uspjeli puno zaključiti o ponašanju početnog Yelp dataset-a i kodirati relativno učinkovit program za kategorizaciju slika.

## 7. LITERATURA

- [1] <https://www.yelp.com/dataset/challenge>
- [2] <https://github.com/stratospark/food-101-keras/blob/master/Food%20Classification%20with%20Deep%20Learning%20in%20Keras.ipynb>
- [3] <https://towardsdatascience.com/image-classification-python-keras-tutorial-kaggle-challenge-45a6332a58b8>
- [4] <https://www.kaggle.com/gab-orfodor/seedlings-pretrained-keras-models>
- [5] <https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6>
- [6] <https://keras.io/getting-started/sequential-model-guide/>
- [7] <https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/>
- [8] <https://towardsdatascience.com/hacking-your-image-recognition-model-909ad4176247>
- [9] <https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks>