

Konvolucionna Neuralna Mreza za Prepoznavanje Dominatne Boje

Petar Nesic

November 24, 2024

1 Problem koji pokusavamo da resimo

Prepoznavanje boja na slikama je problem koji se bavi prepoznavanjem najdominantnije boje na slici. Jedan od zahteva je da se koristi datasete. [House Plant Species](#). Ovaj dataset nije namenjen za prepoznavanja, vec je namenjen za klasifikaciju raznih biljaka. Sto nam odmah govori da ceo dataset zapravo treba da se modifikuje tako da se uklapa u nas problem.



Figure 1: Primer dominantnih boja slike

Kako pristupiti ovom problemu? Mozda je bolje pitanje kako pristupiti bilo kom problemu koji uci iz podataka. Najbolji savet koji sam ja nasao na netu je *"Ako planirate da trenirate model 2 sata, pripremite podatke 6 sati"*. Pa da pocnemo sa podacima. Sve sto opisujem u ovom izvestaju se segmentisano nalazi u jupyter notebook-u gde se nalaze kratki opisi i kod, tako da je najbolje da se prati uz kod.

2 Obrada i gruba analiza podataka

2.1 Preuzimanje dataset-a

Za pocetak preuzmemmo podatke sa [kaggle.com House Plant Species](#), to radimo tako sto prvo generisemo API key za preuzimanje, uploadujemo ga iz fajl sistema u notebook. Posle odredjenog vremena ce se preuzeti dataset. Dataset stize zipovan pa moramo i da ga unzipujemo.

2.2 Kako izgleda nas dataset?

To mozemo lako da proverimo ili da procitamo iz deskripcije dataset-a. Ali da bi proverili da smo sve preuzeli i posto nas obicno mrzi da citamo, bolje da iteriramo kroz fajl sistem i vidimo sta se desava. Utvrđujemo da postoji **14790** slika raznih biljaka koje su spakovana u **47** razlicitih foldera. Ti folderi nam označavaju klase/labele npr. (Aloe Vera -i 252 slike). Zasto je ovo bitno a zapravo nebitno. Jer

se bavimo problemom prepoznavanja dominatne boje nama ne trebaju nazivi tih biljaka, sto odmah implicira da mi moramo da promenimo label-e ovih slika.

2.3 Kako izgleda neki random uzorak iz dataset-a?

To barem mozemo lako da proverimo.



Figure 2: random sample

Prvo sta mozemo da vidimo da su slike **OGROMNE** iako se razlikuju u velicinama, ovo nasoj budućoj neuralnoj mrezi neće da prija a ni remote grafickama koje trebaju da manipulisu ovolikim brojem piksela. Tako da bi možda bilo najpametnije da transformisemo ove slike.

2.4 Transform

Da bi pripremili slike za buduci CNN moramo da im promenimo dimenzije. To mozemo da radimo preko torch transforms. Pritom cemo ih prebaciti u tensore kad smo vec tu.

Listing 1: Pytorch Transform

```

transform = transforms.Compose([
    transforms.Resize(size=(128, 128)),
    #transforms.RandomHorizontalFlip(p=0.5), nema potrebe posto radimo sa bojama
    transforms.ToTensor(),
])

```

Da bi videli sta smo uradili plotovacemo originalnu i novu verziju slike i to preko matplotlib.pyplot biblioteke. Nase slike sada izgledaju ovako.



Figure 3: Original vs New

Nase slike su sada 128x128 i one su tensori ciji je shape u formatu [RGB, H, W].

2.5 Splitovanje podataka

Posto nam labeli tj. nazivi ovih slika nebitni mi mozemo da uzmemos random broj slika iz dataset-a i da na njima treniramo i testiramo nas model. Zato cemo odmah to uraditi. Tokom rada na ovom projektu sam konstantno menjao velicinu dataset-a da vidim kako bi to najbolje uticalo na predikcije. Zato sam krenuo od malog broja slika (1000 slika) da bih video sa koliko najmanje slika moze da nas model da nauči da prepoznaže. Isao sam logikom da bi svako hteo da sa vrlo malo provezbanih zadataka iz matematike zna celu zbirku. Dok pisem ovaj izvestaj model trenira sa 4000 slika. Evo parce koda za train test split.

Listing 2: Train Test Split

```

from torch.utils.data import Subset

dataset = datasets.ImageFolder(root=dataset_path, transform=transform)

# Biramo random n slika
subset_size = 4000
random.seed(42)
subset_indices = random.sample(range(len(dataset)), subset_size)
dataset = Subset(dataset, subset_indices)

train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size

# 80/20 split
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

```

3 Relabelovanje podataka

Kako mozemo nasim slikama dobre klase, posto su nam nazivi biljaka beskorisni. Tako sto prodjemo kroz svaku sliku u dataset-u i odredimo koja je dominantna boja. Prvo sam probao pesackim putem tako sto sam prolazio kroz sve piksele i trazio boju koja se najvise puta ponavlja. Kada sam hteo

da proverim koja je moja dominantna boja dosadilo mi je vise bilo da ulazim na gugl i preko nekih sumnjivih sajtova proveravam koje boje je moja slika, trajalo je mnogo dugacko pritom sam morao da skidam sliku i da je uploadujem.

3.1 Color Thief

Tako sam naisao na **ColorThief** biblioteku koja moze lagano da utvrdi dominantnu boju. Pa sam hteo da poredim moje i njegovo resenje (supervised learning). Dobio sam ove rezultate. Ali pesacki algoritam je davao i mnogo gore od ovog resenja.



Figure 4: ColorThief vs Pesacki

Malo sam razmislio i rekoh daj da koristim lepo color thief za relabelovanje sigurno je bolje od mog resenja. Napravio sam dataset i normalizovao boje da bi model lakse ucio. Kada sam prosao kroz sve slike hteo sam da plotujem da vidim kako je on to uradio umesto mene, jedini dobar lopov je lopov koji krade boje. U uglu svake slike su dominantne boje.

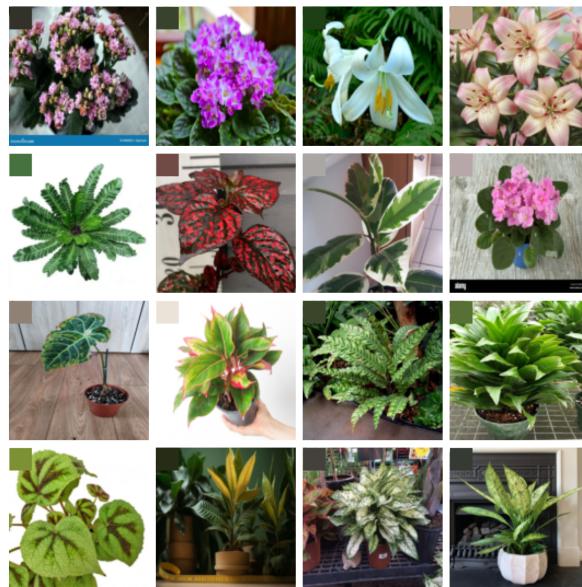


Figure 5: Lopov

Prosao sam kroz sve slike i dodelio im nove klase, tako sto sam koristio colorthief-a koji je za svaku sliku nasao dominantnu boju, pri kreiranju custom dataset-a koji nasledjuje PyTorch dataset.

4 Analiza podataka

Ovu sekciju je najbolje da pogledate u samom notebook-u. U sustini podaci koji su plotovani na drugacije nacine da bi se izvukli drugaciji zaključci. Bilo bi previse i da njih stavim ovde kao slike ovaj izvestaj bi vise licio na enciklopediju.

5 Dizajniranje modela

Pri ucenju gledao sam tutorijale i pratio vezbe na fakultetu. Na internetu sam nasao stranicu [CNN Explainer](#) gde mozes da se igras sa neuralnom mrezom koja je Tiny VGG arhitekture, koja sto sam kasnio saznao se karisti za obrazovne svrhe jer je laka za objasnjanje. TinyVGG koristi slojeve konvolucije i Max Pooling-a, svaki sloj konvolucije pracen ReLU aktivacionom funkcijom. Tako da sam nju iskoristio kao bazni model koji cu kasnije odbaciti ili mozda optimizovati. Model prihvata slike u RGB formatu sa dimenzijam [3, 128, 128] (C,H,W) prolazenjem kroz slojeve prepoznaće dominantnu boju.

5.1 Treniranje TinyVGG-a

Slike su vec organizovane u train i test datasetove, da bi ubrzali proces koristimo DataLoader-e za efikasno ucitavanje podataka u mini batch formatu od 32 slike. MSELoss (Mean Squared Error Loss) koristimo kao loss function i merimo razliku izmedju predvidjenih i stvarnih vrednosti, po cemu azuriramo tezine. Koristi se adam optimizator sa razlicitim stopama ucenja da bi azurirao tezine. Model se trenira kroz 10 epoha ali i to je promenljivo

5.2 Eksperimentisanje TinyVGG-a

Napravio sam prost model i trenirao ga sa 1000 slika (800 train, 200 test). Lose se pokazao sa toliko malo podataka, ali sam htio da izguram sa sto manje podataka. Pa sam se igrao sam hiperparametrima kao sto je lr, dodavao sam DropOut i cak sam i kernel size menjao ali predikcije nisu bile bas najbolje. Tako sam odustao od toga da istreniram sa 1000 slika pa sam povecao na 2000 slika. Sto je znatno poboljsalo predikcije. Dodatno sam se igrao sa hiperparametrima i dodavao konvolucione blokove. Sve ove eksperimente imam opisano u notebook-u da ne zalazim previse u detalje.

6 Drugi CNN

Da bih video koliko je los TinyVGG napravio sam jos jedan model. Ovaj model prihvata slike u istom format [3,128,128] ima 4 konvolucionala sloja za sad, posle svakog konvolucionog sloja koristi se 2x2 max pooling koji polovi dimenzije slike. Nakon konvolucionih slojeva podaci se transformisu u vektor (flatten), FC2 ima ulaz od 512 vorova i izlaz od 3 cvora koji predstavljaju RGB vrednosti. Posle svakog konvolucionog sloja i prvog Fully Connected (FC) se koristi ReLU aktivaciona funkcija za nelinearnost. Proces treniranja ovog modela je gotovo isti kao i TinyVGG.

References