Neuronske mreže – Treći domaći zadatak

ResNet50 klasifikacija otpada

Contents

Uvod

Svet generiše 2 milijarde tona komunalnog čvrstog otpada godišnje, od čega se najmanje 33% ne upravlja na ekološki bezbedan način. Širom sveta, otpad koji se stvara dnevno po osobi u proseku iznosi 0,74kg, ali se kreće u širokom rasponu, od 0,11 do 4,54kg. Sakupljanje otpada je kritičan korak u upravljanju otpadom. Segregacija smeća uključuje odvajanje otpada prema tome kako se njime rukuje ili kako se obrađuje. Važno je za reciklažu jer se neki materijali mogu reciklirati, a drugi ne.

Rešićemo problem kalsifikacije otpada uz pomoć ***PyTorch*** biblioteke u ***Python-u***. Dataset preuzet sa kaggle.com sajta, sadrži sledeće klase otpada:

['metal', 'cardboard', 'glass', 'trash', 'plastic', 'paper']

Svaka od nabrojanih klasa sadrži veliki broj različitih fotografija odgovarajuće klase.

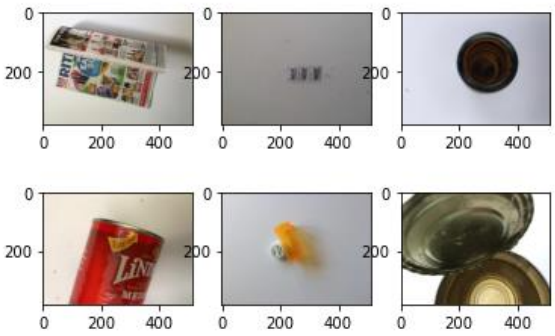

Figure 1 - Fotografije otpada

Najpre vršimo transformaciju fotografija, kroz smanjenje veličine i konverziju u PyTorch tensor:

transformations = transforms.Compose([transforms.Resize((256, 256)), transforms.ToTensor()])

Podelićemo skup podataka na skupove za obuku, validaciju i testove i kreirati učitavače podataka za obuku i validaciju koristeći ***DataLoader***.

Resnet50 model

Resnet je skraćeni naziv za rezidualnu mrežu koja podržava ***rezidualno učenje***. 50 označava broj slojeva koje ima. Dakle, ***Resnet50*** je skraćenica za Residual Netvork sa ***50 slojeva***. Duboke konvolucione mreže dovele su do brojnih otkrića za klasifikaciju slika. Uopšteno govoreći, trend je da se ide u dublji broj slojeva da bi se rešili složeni zadaci i povećala klasifikacija kao i tačnost prepoznavanja. Ali kako idemo dublje sa neuronskim mrežama, tačnost dolazi do zasićenja, a zatim takođe degradira. Residualna obuka pokušava da reši ovaj problem, zbog čega ću upravo ***Resnet50*** model primeniti za rešavanje problema klasifikacije otpada.

```
# Model
network = models.resnet50(pretrained=True)
# Replace last layer
num_ftrs = network.fc.in_features
network.fc = nn.Linear(num_ftrs, len(dataset.classes))
```

Figure 2 - ResNet50 model

Kreiramo funkcije za korak treniranja, validacije, kao i kraj epohe validacije, koje su u suštini u osnovi modela.

```
def training_step(self, batch):
    images, labels = batch
    out = self(images)
    loss = F.cross_entropy(out, labels)
    return loss

def validation_step(self, batch):
    images, labels = batch
    out = self(images)
    loss = F.cross_entropy(out, labels)
    acc = accuracy(out, labels)
    return {'val_loss': loss.detach(), 'val_acc': acc}

def validation_epoch_end(self, outputs):
    batch_losses = [x['val_loss'] for x in outputs]
    epoch_loss = torch.stack(batch_losses).mean()
    batch_accs = [x['val_acc'] for x in outputs]
    epoch_acc = torch.stack(batch_accs).mean()
    return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}
```

Figure 3 - funkcije

Sada možemo pristupiti treniranju modela tokom 8 epoha.

```
num_epochs = 8
opt_func = torch.optim.Adam
lr = 5.5e-5

model_result = fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

Figure 4 - Treniranje modela

Obrada velikog broja fotografija, zahteva jako puno resursa. Tako je za treniranje ***Resnet50*** modela tokom ***8 epoha*** bilo neophodno čitavih ***sat vremena*** koristeći samo ***CPU***. Model je nakon treniranja postigao ***tačnost*** od ***95,14%***. Tokom 8 epoha dolazimo do platoa, procenat tačnosti stagnira.

```
Epoch 1: train_loss: 1.4674, val_loss: 1.2824, val_acc: 0.8108
Epoch 2: train_loss: 1.1899, val_loss: 1.1688, val_acc: 0.9080
Epoch 3: train_loss: 1.0951, val_loss: 1.1445, val_acc: 0.9097
Epoch 4: train_loss: 1.0725, val_loss: 1.1269, val_acc: 0.9497
Epoch 5: train_loss: 1.0636, val_loss: 1.1209, val_acc: 0.9514
Epoch 6: train_loss: 1.0619, val_loss: 1.1293, val_acc: 0.9479
Epoch 7: train_loss: 1.0573, val_loss: 1.1005, val_acc: 0.9583
Epoch 8: train_loss: 1.0542, val_loss: 1.1049, val_acc: 0.9514
```

Figure 5 - Rezultat

Zarad ***ubrzanja*** treniranja modela, vršimo portovanje modela na ***GPU***. ***PyTorch*** olakšava korišćenje ***GPU-a*** u ogromnoj meri. Kada bismo pisali ***GPU kernel u C++*** okruženju, to bi zahtevalo daleko više posla, ali bismo imali daleko veće mogućnosti za optimizaciju, kako bismo dobili na brzini.

```
def get_default_device():
    """Use GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    """Move tensor(s) to the device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)
```

Figure 6 - GPU implementacija

Nvidia GTX 1650 grafička karta je uz pomoć ***Cuda*** arhitekture, isti model, za koji je ***CPU*** jedinici bilo potrebno ***sat vremena***, utrenirala za samo ***6 minuta***. ***Cuda*** je izuzetno pogodna za ***Tensorflow*** modele, jer znatno obrzava rešavanje paralelnih problema.

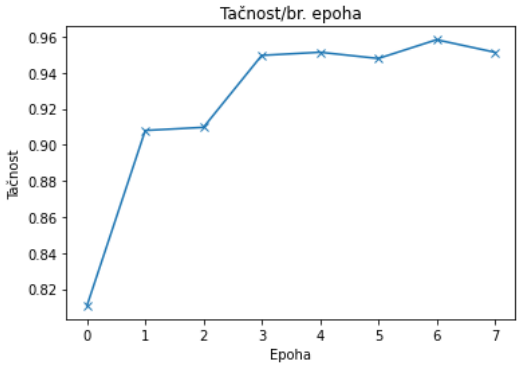

Figure 7 - Tačnost / Epoha

Gubitak validacije je ovde relativno konstantan i gubitak obuke nastavlja da opada. To sugeriše da naše poboljšanje u setu treninga ima tendenciju da se previše uklapa i da se ne može generalizovati na nevidljive podatke. Da bismo dalje poboljšali, mogli bismo istražiti dodavanje više regularizacije na model.

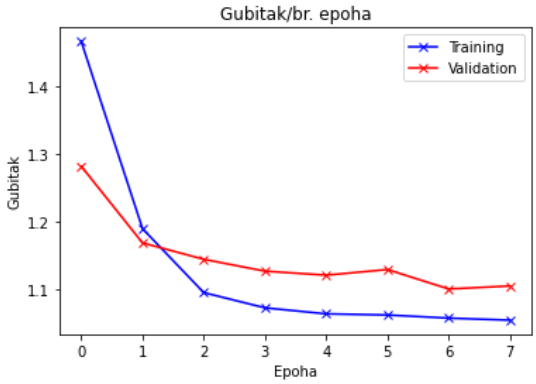

Figure 8 - Gubitak / Epoha

Na kraju sam uradio test sa novim nasumično preuzetim slikama, nakonm čega sam došao do zaključka da model odlično klasifikuje većinu klasa objekata. Velika borba je između stakla i metala jer imaju dosta sličnosti u izgledu.

predicted	actual
paper	paper
plastic	plastic
paper	paper
glass	glass
metal	glass
metal	glass
cardboard	cardboard
glass	glass
plastic	plastic
plastic	plastic
paper	paper
plastic	plastic
glass	metal
metal	metal
paper	paper
paper	paper

Figure 9 - Externi test

Literatura

- https://arxiv.org/pdf/1512.03385.pdf
- https://towardsdatascience.com/beginners-guide-on-image-classification-vgg-19-resnet-50-and-inceptionresnetv2-with-tensorflow-4909c6478941
- https://pythonprogramming.net/introduction-deep-learning-python-tensorflow-keras/
- https://arxiv.org/abs/1512.03385
- https://towardsdatascience.com/understand-and-implement-resnet-50-with-tensorflow-2-0-1190b9b52691
- Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch - Vishnu Subramanian
- Introduction To Neural Networks - Prof. George Papadourakis
- Veštačke neuronske mreže - Prof. dr Leonid Stoimenov