UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
DEPARTMAN ZA MATEMATIKU I
INFORMATIKU

Prepoznavanje lica pomoću Dubokih Konvolucionih Neuronskih Mreža

-Praktični projekat iz predmeta Veštačka Inteligencija-

Marko Mirković, 351/21

Novi Sad, jun 2024.

Sadržaj

1 Uvod

U ovom radu su prikazani rezultati poređenja tri modela na bazi neuronskih mreža i tri tradicionalna ML modela sa zadatkom prepoznavanja lica. Modeli na bazi neuronskih mreža su duboke konvolucione neuronske mreže, dok su ML modeli ZeroRule, NaiveBayes i SupportVectorMachine.

Konvolucione neuronske mreže (CNN) se najčešće koriste u oblasti computer vision-a. Za razliku od običnih neuronskih mreža, sadrže konvolucione slojeve koji pomažu mreži da zaključi karakteristične oblike ili šablone unutar slike. Slike koje ulaze u CNN su predstavljene u obliku matrice piksela i mogu biti RGB ili monohromatske. Proces se odvija na sledeći način: filter se postavi na početak slike, matričnim množenjem filtera i piksela se računa nova vrednost koja se upiše u drugu matricu i filter se pomera za određeni broj piksela. Ovaj postupak se ponavlja dok se ne dođe do kraja slike. Periodično se javljaju slojevi sažimanja (pooling layer) koji smanjuju broj parametara kako bi olakšali računske operacije. Najčešće se koristi sažimanje maksimumom ili sažimanje srednjom vrednošću. Na samom kraju mreže se nalaze potpuno povezani slojevi kao i kod tradicionalnih neuronskih mreža.

2 Modeli

2.1 ZeroRule

Korišćen je ručno implementiran **ZeroRule** klasifikator sa ciljem utvrđivanja minimalnih očekivanih performasi modela.

2.2 NaiveBayes

Korišćen je **Gaussian Naive Bayes** iz **Scikit-learn** biblioteke sa podrazumevanim parametrima.

2.3 SupportVectorMachine

SupportVectorMachine implementacija korišćena u projektu je **SVC** klasifikator iz **Scikit-learn** biblioteke sa **„rbf“** kernel parametrom.

2.4 Custom Convolutional Neural Network

```
def forward(self, x):
    x = self.pool(F.relu(self.bn1(self.conv1(x))))
    x = self.pool(F.relu(self.bn2(self.conv2(x))))
    x = self.pool(F.relu(self.bn3(self.conv3(x))))
    x = self.pool(F.relu(self.bn4(self.conv4(x))))
    x = x.view(-1, 256 * 8 * 8)
    x = F.relu(self.fc1(x))
    x = self.dropout(x)
    x = self.fc2(x)
    return x
```

Slika 2.1: „forward“ funkcija neuronske mreže

Na slici 2.1 se nalazi **forward** funkcija koja pripada mreži sa četri **konvoluciona** sloja nakon kojih se nalaze **BatchNorm2d** slojevi praćeni **ReLU** slojevima. Sledi prvi **potpuno povezani** sloj praćen **ReLU** slojem, nakon kog ide **Dropout** sloj i poslednji **potpuno povezani** sloj.

```
def __init__(self, num_classes):
    super(FaceRecognitionCNN, self).__init__()
    self.conv1 = nn.Conv2d( in_channels: 1,  out_channels: 32, kernel_size=5, padding=2)
    self.bn1 = nn.BatchNorm2d(32)
    self.conv2 = nn.Conv2d( in_channels: 32,  out_channels: 64, kernel_size=5, padding=2)
    self.bn2 = nn.BatchNorm2d(64)
    self.conv3 = nn.Conv2d( in_channels: 64,  out_channels: 128, kernel_size=3, padding=1)
    self.bn3 = nn.BatchNorm2d(128)
    self.conv4 = nn.Conv2d( in_channels: 128,  out_channels: 256, kernel_size=3, padding=1)
    self.bn4 = nn.BatchNorm2d(256)
    self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
    self.fc1 = nn.Linear(256 * 8 * 8,  out_features: 1024)
    self.fc2 = nn.Linear( in_features: 1024, num_classes)
    self.dropout = nn.Dropout(0.4)
```

Slika 2.2: Konstruktor neuronske mreže sa vrednostima hiperparametara

2.5 ResNet18 backbone

```
def __init__(self, num_classes):
    super(FaceRecognitionResNet18, self).__init__()
    self.resnet = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
    self.resnet.conv1 = nn.Conv2d( in_channels: 1,  out_channels: 64, kernel_size=7, stride=2, padding=3, bias=False)
    self.resnet.fc = nn.Linear(self.resnet.fc.in_features, num_classes)
    # Optional: Freeze earlier layers
    for param in self.resnet.parameters():
        param.requires_grad = False

    for param in self.resnet.layer3.parameters():
        param.requires_grad = True

    for param in self.resnet.layer4.parameters():
        param.requires_grad = True

    for param in self.resnet.fc.parameters():
        param.requires_grad = True
```

Slika 2.3: Konstruktor druge neuronske mreže

Na slici 2.3 se nalazi konstruktor neuronske mreže koja koristi **ResNet18** backbone, odnosno navedeni pretrenirani model. Promenjen je prvi sloj mreže kako bi prihvatao monohromatske slike, kao i poslednji kako bi izlazni sloj imao isti broj izlaza kao i broj klasa. Svi slojevi mreže su zamrznuti kao prevencija overfit-a. Odmrznuta su poslednja tri sloja mreže.

2.6 ResNet50 backbone

Konstruktor ovog modela je identičan **ResNet18** konstruktoru osim što se koristi **ResNet50** backbone i njegove početne težine.

3 Experimentalna postavka

3.1 Dataset

Korišćen dataset je **Casia-WebFace**, redukovan na 500 klasa sa najvećim brojem instanci. Ukupan broj instanci u redukovanom dataset-u je 129902. Podela dataset-a na **trening** i **validacioni** je izvedena **train_test_split** funkcijom iz **Scikit-learn** biblioteke i korišćen je **random_state** parametar 42.

Za ML modele korišćen je **PCA** algoritam za redukciju dimenzionalnosti. Svaka instanca je redukovana na 150 komponenti. Parametar **solver** je „randomized“, a **whiten** je True.

Za treniranje neuronskih mreža koristimo **DataLoader** iz **PyTorch** biblioteke radi treniranja u paketu (**batch**) od 32 instance.

3.2 Evaluacija

Kod ML modela, kao metod evaluacije koristi se preciznost (**accuracy**) nad validacionim setom.

Kod neuronskih mreža koristi se više različitih metrika. Za kriterijum treniranja se koristi **Cross Entropy Loss**. **Epoch Loss/Training Loss** se koristi za svaku epohu treniranja, kao i gubitak (**Validation Loss)** i preciznost (**Validation Accuracy**) prilikom validacije. Za optimizator se koristi **Adam** optimizator sa parametrima **lr** (learning rate/stopa učenja) 0.001 i **weight_decay** 1e-4. **Scheduler** koristimo za promenu stope učenja, kao meru prevencije overfit-a, sa parametrima **step_size** 10 i **gamma** 0.1.

3.3 Programski jezik i biblioteke

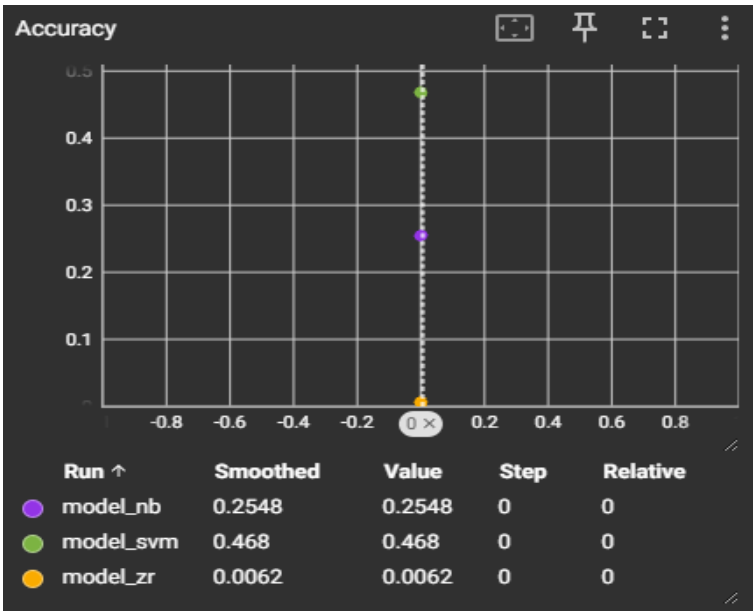
U projektu je korišćen programski jezik **Python** (verzija 3.12) sa bibliotekama **PyTorch**, **Scikit-learn** i **Numpy**. Za kontrolu verzija, virtuelno okruženje i instalaciju paketa korišćen je **Conda** alat.

ML modeli su trenirani na CPU-u, dok su neuronske mreže trenirane na GPU-u. Korišćena je **Nvidia Cuda** platforma, kao i **CUDNN** biblioteka radi bolje optimizacije treninga. Takođe, prilikom instalacije **PyTorch** biblioteke potrebno je dobaviti verziju koja podržava rad sa **Cuda** platformom.

Za praćenje i vizualizaciju metrika korišćen je **TensorBoard** iz **TensorFlow** biblioteke, koji je takođe podržan u **PyTorch** biblioteci.

4 Rezultati

4.1 ML modeli



Slika 4.1: Grafikon preciznosti ML modela

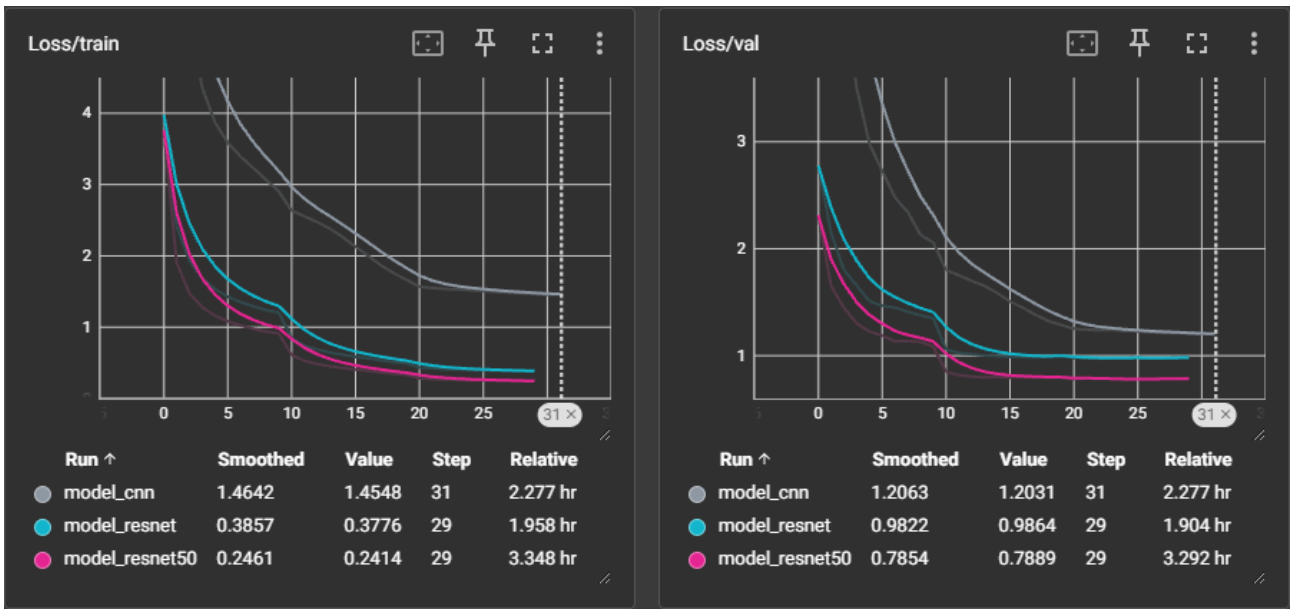
Kao što se vidi na slici 4.1, preciznosti **ZeroRule** klasifikatora je 0.62%, **NaiveBayes** klasifikatora je 25.48%, a **SupportVectorMachine** klasifikatora je 46.80%.

4.2 Neuronske mreže

U ovom delu je prikazana stabilnost treninga kao i sama preciznost različitih arhitektura.

4.2.1 Stabilnost treninga

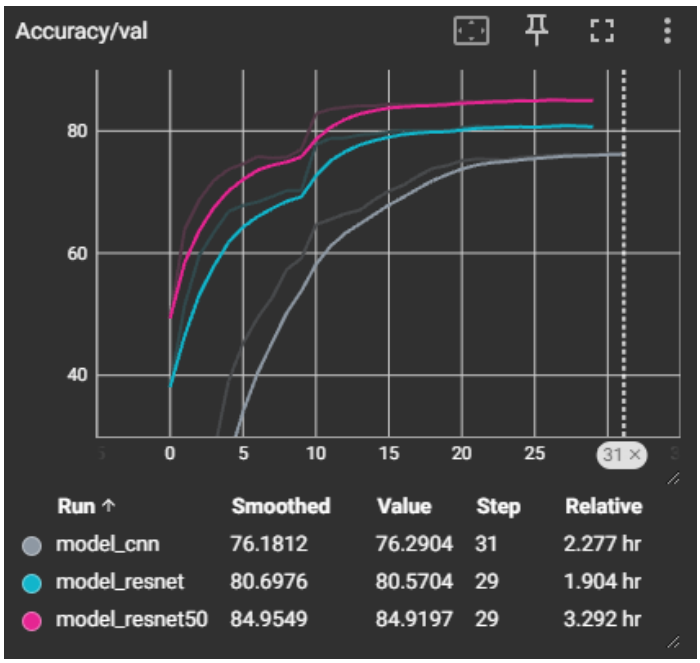
Praćenjem krivih gubitaka na trening setu i validacionom setu, prati se stabilnost treninga. Navedene krive ne moraju imati iste vrednosti, ali je poželjno da obe krive imaju slično ponašanje i da konvergiraju u nekoj bliskoj vrednosti. U trenutku kada validacioni gubitak krene da raste, potrebno je zaustaviti trening (**early stopping**) kako ne bi došlo do overfit-a.



Slika 4.2: Vrednosti metrika gubitka neuronskih mreža

X-osa predstavlja epohe, a Y-osa vrednost metrika gubitka. Levi grafik predstavlja gubitak u treningu, a desni u validaciji. Trening **CNN**-a je potrebno prekinuti oko epohe 27, **ResNet18** mreže oko epohe 18, a **ResNet50** mreže oko epohe 16.

4.2.2 Preciznost



Slika 4.3: Vrednost metrika preciznosti neuronskih mreža

CNN dostiže preciznost od 76.18%, **ResNet18** od 80.69%, a **ResNet50** od 84.91%.

5 Zaključak

U ovom radu je obrađen zadatak prepoznavanja lica pomoću dubokih konvolucionih neuronskih mreža, drugih neuronskih arhitektura i poređenje sa rezultatima ML modela.

Rezultati pokazuju da duboke konvolucione neuronske mreže postižu drastično bolje performanse od ML modela, što je i očekivano. Takođe, prikazano je da korišćenje pretreniranih modela može dalje poboljšati performanse.

Jedan od velikih izazova na ovom projektu su računarski resursi, iz razloga što pomenute mreže zahtevaju velike količine podataka i računskih operacija. Treniranje pomoću GPU-a je drastično ubrzalo proces, koji je moguće dalje ubrzati hardverom predviđenim za treniranje ovakih modela.

Tema daljeg istraživanja može biti dodatno povećanje performansi ovih mreža, što se može realizovati nekom od sledećih tehnika: odmrzavanje više slojeva pretreniranih mreža, stratifikacija dataset-a, detaljnija obrada instanci, bolje prilagođavanje hiperparametara mreže, kao i druge napredne tehnike u oblasti mašinskog učenja.