

YILDIZ TEKNİK ÜNİVERSİTESİ GÖRÜNTÜ İŞLEME PROJESİ



Hazırlayan: Yunus Karatepe 17011051

Ders: Görüntü İşleme – BLM4540

Konu: Derin Öğrenme İle Görüntü Sınıflandırması

YÖNTEM-)

Öncelikle kullandığım kütüphane: Pythonun Pytorch isimli bir derin öğrenme kütüphanesi. Yazılımı ilk kez kullandığım için “sentdex” isimli youtube kanalındaki Pytorch ile derin öğrenme videolarından (8 adet video) yola çıkarak yapmaya çalıştım.

1-) Data Oluşturma:

```
class Data:
    ALL_SOULS = "data/all_souls"
    ASHMOLEAN = "data/ashmolean"
    BALLIOL = "data/balliol"
    BODLEIAN = "data/bodleian"
    CHRIST_CHURCH = "data/christ_church"
    CORNMARKEt = "data/cornmarket"
    HERTFORD = "data/hertford"
    JESUS = "data/jesus"
    KEBLE = "data/keble"
    MAGDALEN = "data/magdalen"
    NEW = "data/new"
    ORIEL = "data/oriel"
    OXFORD = "data/oxford"
    PITT_RIVERS = "data/pitt_rivers"
    RADCLIFFE_CAMERA = "data/radcliffe_camera"
    TRINITY = "data/trinity"
    WORCESTER = "data/worcester"
    LABELS = {ALL_SOULS: 0, ASHMOLEAN: 1, BALLIOL: 2, BODLEIAN: 3, CHRIST_CHURCH: 4, CORNMARKEt: 5, HERTFORD: 6, JESUS: 7, KEBLE: 8, MAGDALEN: 9, NEW: 10, ORIEL: 11, OXFORD: 12, PITT_RIVERS: 13, RADCLIFFE_CAMERA: 14, TRINITY: 15, WORCESTER: 16}
    data = []
```

```
def make_data(self):
    for label in self.LABELS:
        i = 0
        for path in tqdm(glob.glob(label + '/*.jpg')):
            img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
            img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
            self.data.append([np.array(img), np.eye(17)[self.LABELS[label]]])

    np.random.shuffle(self.data)
    np.save("data.npy", self.data)
    print(len(self.data))
```

Data classımızı kullanarak datamızı shuffle işleminden geçtikten sonra oluşturmuş olduk.

Şimdi sırada modelimizi oluşturmak var.

2-) Modeli Oluşturmak:

```
62 class Net(nn.Module):
63     def __init__(self):
64         super().__init__()
65         self.conv1 = nn.Conv2d(1, 32, 3)
66         self.conv2 = nn.Conv2d(32, 64, 3)
67         self.conv3 = nn.Conv2d(64, 128, 3)
68
69         x = torch.randn(IMG_SIZE, IMG_SIZE).view(-1, 1, IMG_SIZE, IMG_SIZE)
70         self._to_linear = None
71         self.convs(x)
72
73         self.fc1 = nn.Linear(self._to_linear, 512)
74         self.fc2 = nn.Linear(512, 17)
75
76     def convs(self, x):
77         x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
78         x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))
79         x = F.max_pool2d(F.relu(self.conv3(x)), (2, 2))
80
81         if self._to_linear is None:
82             self._to_linear = x[0].shape[0] * x[0].shape[1] * x[0].shape[2]
83         return x
84
```

```
84
85     def forward(self, x):
86         x = self.convs(x)
87         x = x.view(-1, self._to_linear)
88         x = F.relu(self.fc1(x))
89         x = self.fc2(x)
90         return F.softmax(x, dim=1)
91
92
```

Modelimiz Net isimli classımız olmuş oluyor.

3 adet convolutional layer ve 2 adet fully-connected layer kullanmış olduk.

Sırada ise eğitme işlemi var.

SORU 1-)

3-) Modelin Eğitilmesi:

İlk olarak datamızın 0.8 ini train etmede 0.2 sini ise test etmede kullanıyoruz.

Batch_size = 64 ve Epoch = 10 seçip öncelikle training üzerindeki accurcymizi ölçtüm.

```
98 X = torch.Tensor([i[0] for i in data]).view(-1, IMG_SIZE, IMG_SIZE)
99 X = X / 255.0
100 y = torch.Tensor([i[1] for i in data])
101
102 VAL_PCT = 0.2
103 val_size = int(len(X) * VAL_PCT)
104 print(val_size)
105
106 train_X = X[:-val_size]
107 train_y = y[:-val_size]
108 test_X = X[-val_size:]
109 test_y = y[-val_size:]
110
111 print(len(train_X))
112 print(len(test_X))
113
114 BATCH_SIZE = 64
115 EPOCHS = 10
116
```

Eğitme Döngüsü:

```
for epoch in range(EPOCHS):
    for i in tqdm(range(0, len(train_X), BATCH_SIZE)):
        batch_X = train_X[i: i + BATCH_SIZE].view(-1, 1, IMG_SIZE, IMG_SIZE)
        batch_y = train_y[i: i + BATCH_SIZE]

        net.zero_grad()
        outputs = net(batch_X)
        loss = loss_function(outputs, batch_y)
        loss.backward()
        optimizer.step()
    print(loss)

torch.save(net, 'model2.pt')
```

Ölçtüğüm ilk training accuracy değerim 0.834, loss değerim 0.0003 çıktı. Daha sonra bu işlemi 2 kere daha tekrarladım. Yani toplamda bu modeli 30 epoch ile eğitmiş oldum ve son training üzerinde aldığım accuracy değerim 0.957 ve loss değerim 6.9362e-06 çıktı. Yani aldığım sonucu daha iyileştirmiş oldum.

Test datası üzerindeki accuracy (yani bizim için asıl önemli olan) değerim ise 0.234 çıkmış oldu. 17 classımız olduğunda random sınıf seçen bir modelin başarı oranı 1/17 olacaktır. Bu da 0.058 dir. Yani bizim modelimiz başarılı bir model olmuş oluyor.

Başarı oranını ölçtüğümüz döngü:

```
129 torch.save(net, 'model.pt')
130
131 correct = 0
132 total = 0
133 with torch.no_grad():
134     for i in tqdm(range(len(test_X))):
135         real_class = torch.argmax(test_y[i])
136         net_out = net(test_X[i].view(-1, 1, IMG_SIZE, IMG_SIZE))[0]
137         predicted_class = torch.argmax(net_out)
138         if predicted_class == real_class:
139             correct += 1
140         total += 1
141
142
143 print("Accuracy: ", round(correct / total, 3))
144
145
```

SORU 2-)

Bu soruyu yaparken ise herhangi bir yerde kaynak bulamadım. Ben de çözümü kodu modifiye etmekte buldum.

Bu soruda bizden istenen modelimizin en son katmanları olan fully-connected katmanlarından bir tanesini kullanarak test ve training resimleri arasında en yakın ilişkisini bulmak.

Bu işlemi yapabilmek için öncelikle model kodunu 512'lik layerin sonuçlarını döndürecek şekilde ayarlıyorum. Sonrasında modelimi kaydettiğim yerden load ediyorum. Sonrasında traindeki dönen her 512 elemanlı diziyi training_fc isimli listeye ekliyorum. Aynısını test_fc için de yapıyorum. Daha sonra bu listeyi numpy matrisine çeviriyorum ve elimde training_size x 512 lik bir matris ve test_size x 512 lik bir matris oluyor. Daha sonrasında yaptığımız işlem ise bildiğimiz minimum bulma işlemi. Matrislerden testteki bir satırı traindeki tüm satırlarla karşılaştırıp minimum hangisi ise onun indisini (örn: trainingdeki 342. resme en yakındır) şeklinde bir txt dosyasına yazıyorum.

Kodumdan örnekler vermem gerekirse:

Modifiye ettiğim model kodum:

```
61 class Net(nn.Module):
62     def __init__(self):
63         super().__init__()
64         self.conv1 = nn.Conv2d(1, 32, 3)
65         self.conv2 = nn.Conv2d(32, 64, 3)
66         self.conv3 = nn.Conv2d(64, 128, 3)
67
68         x = torch.randn(IMG_SIZE, IMG_SIZE).view(-1, 1, IMG_SIZE, IMG_SIZE)
69         self._to_linear = None
70         self.convs(x)
71
72         self.fc1 = nn.Linear(self._to_linear, 512)
73
74     def convs(self, x):
75         x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
76         x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))
77         x = F.max_pool2d(F.relu(self.conv3(x)), (2, 2))
78
79         if self._to_linear is None:
80             self._to_linear = x[0].shape[0] * x[0].shape[1] * x[0].shape[2]
81         return x
82
83     def forward(self, x):
84         x = self.convs(x)
85         x = x.view(-1, self._to_linear)
86         x = F.relu(self.fc1(x))
87         return F.softmax(x, dim=1)
88
```

Fully-connected katmanları train ve testteki her resim için alıp matris elde ettiğim bölüm:

```
141
142 net = torch.load('model.pt')
143 with torch.no_grad():
144     for i in tqdm(range(len(train_X))):
145         net_out = net(train_X[i].view(-1, 1, IMG_SIZE, IMG_SIZE))[0]
146         a = net_out.numpy()
147         training_fc.append(a)
148
149 training_fc = np.asarray(training_fc)
150 print(training_fc.shape)
151
152 with torch.no_grad():
153     for i in tqdm(range(len(test_X))):
154         net_out = net(test_X[i].view(-1, 1, IMG_SIZE, IMG_SIZE))[0]
155         a = net_out.numpy()
156         test_fc.append(a)
157
158 test_fc = np.asarray(test_fc)
159 print(test_fc.shape)
160
```

```

160
161 for i in tqdm(range(0, 1012)):
162     min = 99999.0
163     for j in range(0, 4051):
164         value = 0.0
165         for k in range(0, 512):
166             value = value + (abs(test_fc[i][k] - training_fc[j][k]))
167         if (value < min):
168             min = value
169             min_index = j
170
171 f = open("result.txt", "a")
172 f.write("Test datasındaki " + str(i) + ". resim ---> Train datasındaki " + str(min_index) + ". resme en yakındır.\n")
173
174
175

```

training_fc.shape ve test_fc.shape çıktısı:

```
100%|██████████| 4051/4051 [02:22<00:00, 28.41it/s]
(4051, 512)
100%|██████████| 1012/1012 [00:34<00:00, 28.96it/s]
(1012, 512)
```

[illegible]

test_fc numpy matrixinin bir bölümü:

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
12	0.0000024409	0.0000024409	0.0000024409	0.0000024409	0.0000024409	0.0000024409	0.0000024409	0.0000024409	0.0000024409	0.0000024409	0.0000024409	0.7408075929	0.0000024409	0.0000024409	0.0000024409
13	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709	0.0000001709
14	0.0000000123	0.0000000123	0.0000000123	0.0000000123	0.0000000123	0.0000000123	0.0000000123	0.0000000123	0.0000000123	0.0000000123	0.0000000123	0.0012448174	0.0000000123	0.0000000123	0.0000000123
15	0.0000045282	0.0000045282	0.0000045282	0.0000045282	0.0000045282	0.0000045282	0.0000045282	0.0000045282	0.0000045282	0.0000045282	0.0000045282	0.00004801244	0.0000045282	0.0000045282	0.0000045282
16	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627	0.0000020627
17	0.0000000007	0.0000000007	0.0000000007	0.0000000007	0.0000000007	0.0000000007	0.0000000007	0.0000000007	0.0000000007	0.0000000007	0.0000000007	0.9745998383	0.0000000007	0.0000000007	0.0000000007
18	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988	0.0001679988
19	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002
20	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924	0.0000078924
21	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325	0.00000311325
22	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408	0.0001439408
23	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869	0.0015010869
24	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124	0.0000001124
25	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565	0.0000091565
26	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268	0.0000027268
27	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969	0.0000468969
28	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227	0.0001214227
29	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920	0.0007434920
30	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465	0.0000001465
31	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367	0.0000000367
32	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793	0.0000435793
33	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990	0.0000065990
34	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000
35	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314	0.0000048314
36	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002	0.0000000002

result.txt dosyasının bir bölümü:

Test	datasındaki	147. resim	Train	datasındaki	1712. resme en yakindir.
Test	datasındaki	148. resim	Train	datasındaki	3688. resme en yakindir.
Test	datasındaki	149. resim	Train	datasındaki	3162. resme en yakindir.
Test	datasındaki	150. resim	Train	datasındaki	672. resme en yakindir.
Test	datasındaki	151. resim	Train	datasındaki	362. resme en yakindir.
Test	datasındaki	152. resim	Train	datasındaki	2580. resme en yakindir.
Test	datasındaki	153. resim	Train	datasındaki	1573. resme en yakindir.
Test	datasındaki	154. resim	Train	datasındaki	980. resme en yakindir.
Test	datasındaki	155. resim	Train	datasındaki	3750. resme en yakindir.
Test	datasındaki	156. resim	Train	datasındaki	2966. resme en yakindir.
Test	datasındaki	157. resim	Train	datasındaki	1739. resme en yakindir.
Test	datasındaki	158. resim	Train	datasındaki	3033. resme en yakindir.
Test	datasındaki	159. resim	Train	datasındaki	34. resme en yakindir.
Test	datasındaki	160. resim	Train	datasındaki	3714. resme en yakindir.
Test	datasındaki	161. resim	Train	datasındaki	3946. resme en yakindir.
Test	datasındaki	162. resim	Train	datasındaki	1146. resme en yakindir.
Test	datasındaki	163. resim	Train	datasındaki	1014. resme en yakindir.
Test	datasındaki	164. resim	Train	datasındaki	3927. resme en yakindir.
Test	datasındaki	165. resim	Train	datasındaki	3984. resme en yakindir.
Test	datasındaki	166. resim	Train	datasındaki	3990. resme en yakindir.
Test	datasındaki	167. resim	Train	datasındaki	1319. resme en yakindir.
Test	datasındaki	168. resim	Train	datasındaki	3628. resme en yakindir.
Test	datasındaki	169. resim	Train	datasındaki	3125. resme en yakindir.
Test	datasındaki	170. resim	Train	datasındaki	3663. resme en yakindir.

Sonuç:

Sonuç olarak derin öğrenmenin genel anlamıyla mantığını, görüntünün modellere nasıl verilir, sonuçlar alınabileceğini, convolutional neural network kullanımını, fully-connected vektörlerini bir özellik vektörü olarak kullanmayı öğrenmiş olduk. Verimiz çok düzgün olmadığı, ve yeterli sayıda fotoğraftan oluşmadığı için aldığımız sonuçlar çok iç açıcı olmadı. Yine de %5 in biraz üzerinde olan random başarı değerini %24 lere kadar çıkarmayı başardık.