# 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, HERTFO
              JESUS: 7, KEBLE: 8, MAGDALEN: 9, NEW: 10, ORIEL: 11, OXFORD: 12, PITT_RIVERS: 13, RADCLIFFE_
              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:

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, 3)
        self.conv2 = nn.Conv2d(32, 64, 3)
        self.conv3 = nn.Conv2d(64, 128, 3)

        x = torch.randn(IMG_SIZE, IMG_SIZE).view(-1, 1, IMG_SIZE, IMG_SIZE)
        self._to_linear = None
        self.convs(x)

        self.fc1 = nn.Linear(self._to_linear, 512)
        self.fc2 = nn.Linear(512, 17)

    def convs(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv3(x)), (2, 2))

        if self._to_linear is None:
            self._to_linear = x[0].shape[0] * x[0].shape[1] * x[0].shape[2]
        return x
```

```
    def forward(self, x):
        x = self.convs(x)
        x = x.view(-1, self._to_linear)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.softmax(x, dim=1)
```

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 accurcy 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 accurcy değerim 0.957 ve loss değerim 6.9362e-06 çıktı. Yani aldığım sonucu daha iyileştirmiş oldum.

Test datası üzerindeki accurcy ( 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
```

**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 sonda 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 yakindir ) şeklinde bir txt dosyasına yazıyorum.

Kodumdan örnekler vermem gerekirse:

Modifiye ettiğim model kodum:

```python
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, 3)
        self.conv2 = nn.Conv2d(32, 64, 3)
        self.conv3 = nn.Conv2d(64, 128, 3)

        x = torch.randn(IMG_SIZE, IMG_SIZE).view(-1, 1, IMG_SIZE, IMG_SIZE)
        self._to_linear = None
        self.convs(x)

        self.fc1 = nn.Linear(self._to_linear, 512)

    def convs(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv3(x)), (2, 2))

        if self._to_linear is None:
            self._to_linear = x[0].shape[0] * x[0].shape[1] * x[0].shape[2]
        return x

    def forward(self, x):
        x = self.convs(x)
        x = x.view(-1, self._to_linear)
        x = F.relu(self.fc1(x))
        return F.softmax(x, dim=1)

```

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

```python

net = torch.load('model.pt')
with torch.no_grad():
    for i in tqdm(range(len(train_X))):
        net_out = net(train_X[i].view(-1, 1, IMG_SIZE, IMG_SIZE))[0]
        a = net_out.numpy()
        training_fc.append(a)

training_fc = np.asarray(training_fc)
print(training_fc.shape)

with torch.no_grad():
    for i in tqdm(range(len(test_X))):
        net_out = net(test_X[i].view(-1, 1, IMG_SIZE, IMG_SIZE))[0]
        a = net_out.numpy()
        test_fc.append(a)

test_fc = np.asarray(test_fc)
print(test_fc.shape)

```

Karşılaştırma işlemini yapıp en yakın olanı “result.txt” isimli dosyaya yazdığım kod bölümü:

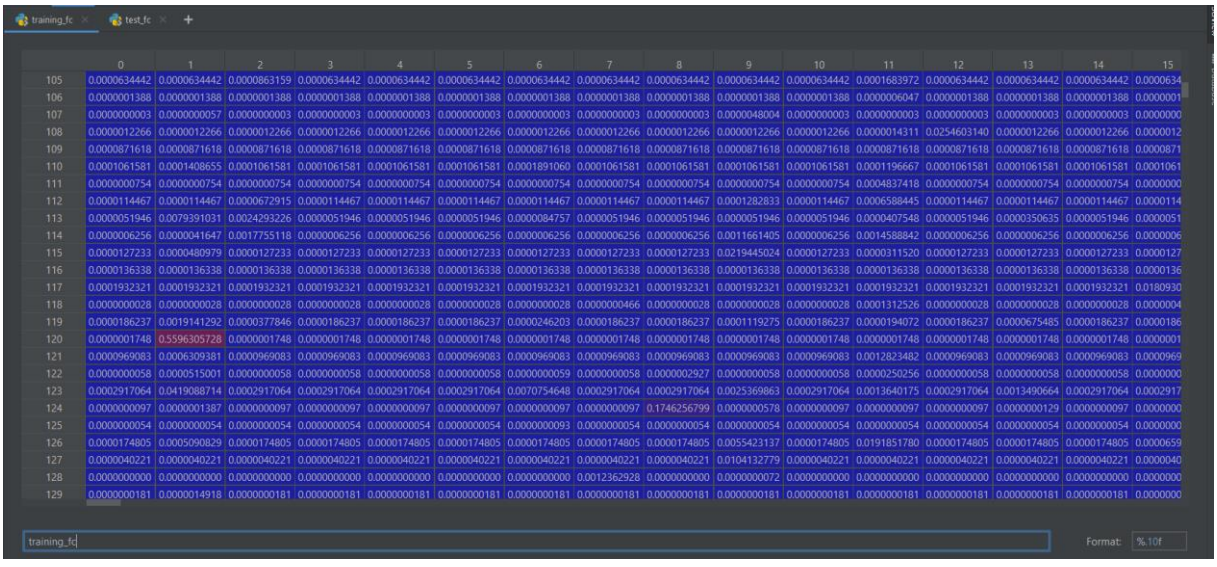
```
160
161 for i in tqdm(range(0, 1012)):
162     min = 99999.0
163     for j in range(0, 4051):
164         value = 0.0
165         for k in range(0, 512):
166             value = value + (abs(test_fc[i][k] - training_fc[j][k]))
167         if (value < min):
168             min = value
169        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

**Çıktılar:**

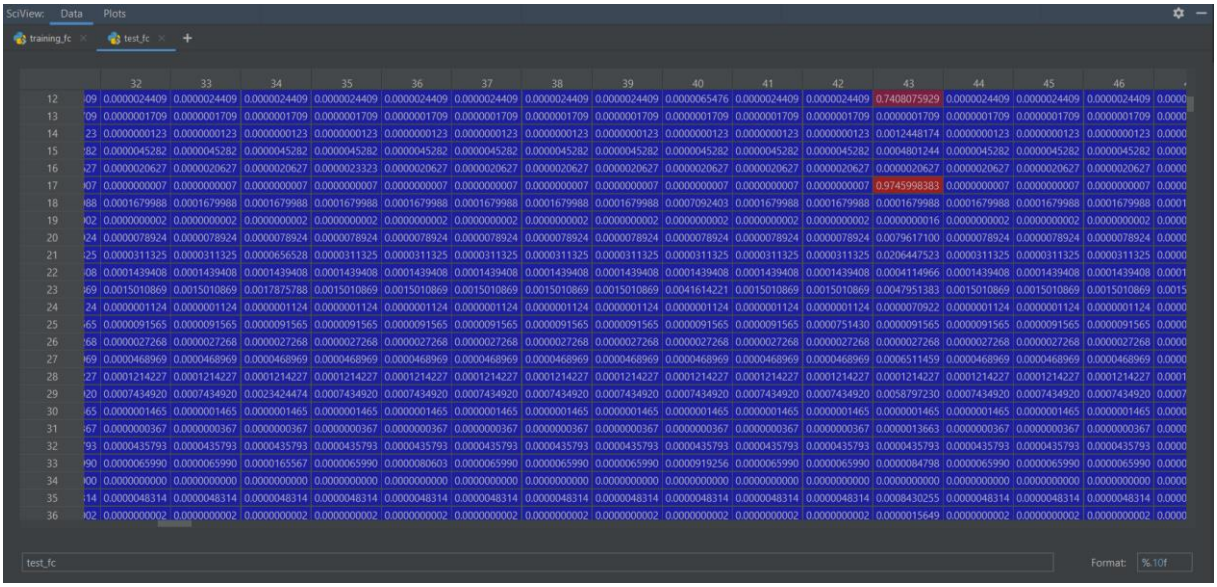
training_fc.shape ve test_fc.shape çıktısı:

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

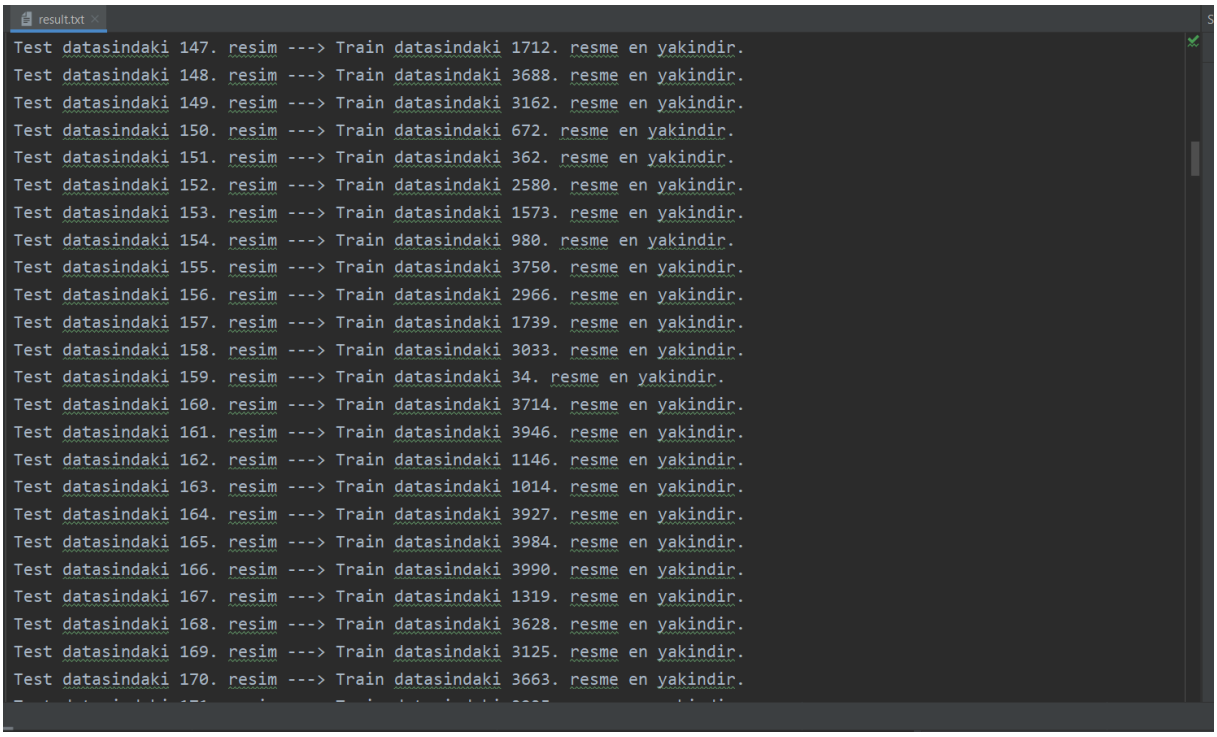
training_fc numpy matrixinin bir bölümü:



test_fc numpy matrixinin bir bölümü:




result.txt dosyasının bir bölümü:




**Sonuç:**

Sonuç olarak derin öğrenmenin genel anlamıyla mantığını, görüntünün modellere nasıl verilip, 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.