

**KASTAMONU ÜNİVERSİTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ GÖRÜNTÜ İŞLEME DERSİ ÖDEV RAPORU**

**ÖDEV**

**Görüntü İşleme 3. Grup**

**Dataset C (flowers-recognition)**

**İş Akışı-3 Uygulamaları**

**ÖDEV TARİHİ**

**22.01.2021**

**DERSİN SORUMLUSU**

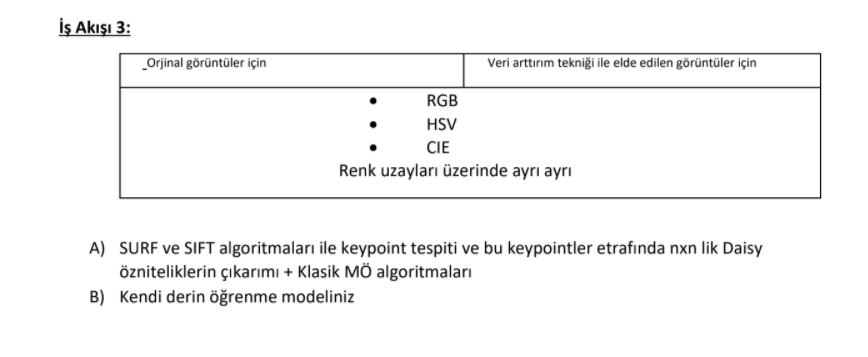
**Kemal AKYOL**

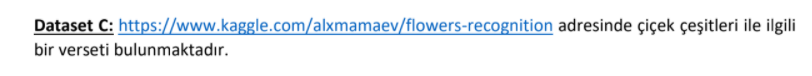
**RAPORU YAZAN ÖĞRENCİ**

**174410037**

**Mustafa Said ÇELİK**

**İş Tanımı**





**A)Ekran Görüntüleri**



**Arayüz Açıklaması**

-Veri Çek butonuna basıldığında ilgili klasör seçilir ve veriler yazdırılır.

-İlk comboBox renk uzayı seçimi içindir.

-İkinci comboBox algoritma seçimi içindir.

-Sift butonu yukarıda bulunan veri sayısı, n\*n daisy değeri, key point sayısı, test size ve fold size’a göre işlem yapar.

-Orb butonu yukarıda bulunan veri sayısı, n\*n daisy değeri, key point sayısı, test size ve fold size’a göre işlem yapar.

-Cnn butonu cnn işlemini gerçekleştirir.

**1)Sift Butonuna Basıldığında Yürtülen İşlem**

**RGB İçin**



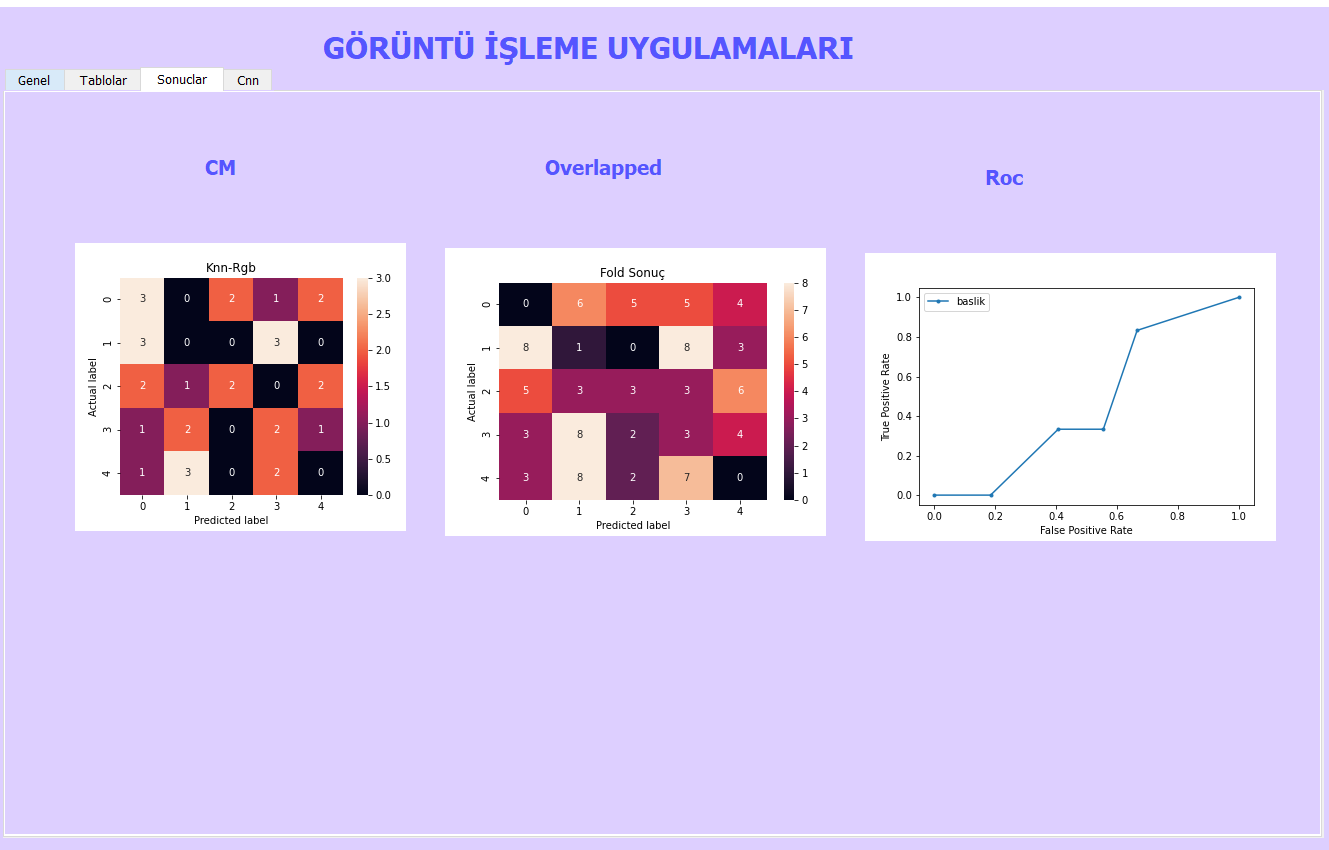
**HSV İçin**



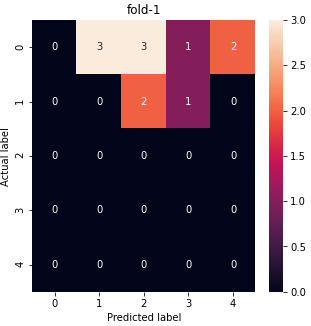
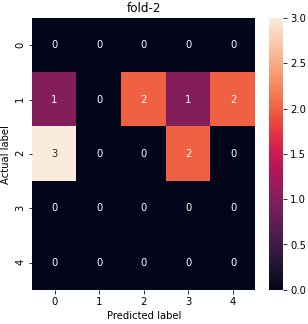
**CİE İçin**

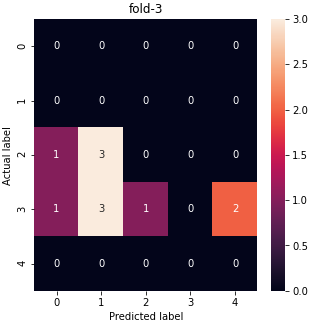
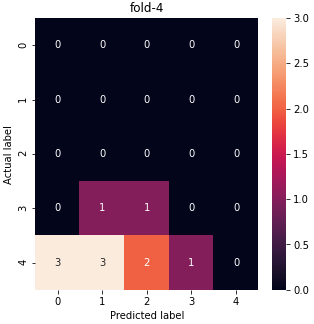


-Sift algoritması çalışır ve sonucu labellara yazılır. Tablodan seçim yapılarak model doğruluğu görüntüde olduğu gibi sınanır.



-Hold-out ile yapılan Confusion matrix ve Roc eğrisi, k-fold ile yapılan Overlapped matrix ekranda gösterilir.

-Her Fold değeri gösterilmiştir.



-Daisyden dönen verilerin test ve train olarak ayrımı gösterilmiştir.

-Orb butonu sonucuda ayni işlemler yapılır ve sonuçları gösterir.

**2)Cnn Butonuna Basıldığında Yürtülen İşlem**



-Cnn sonucunda Confusion matrix, acc-loss grafikleri ve Roc eğrisi gösterilir.

**B)KODLAR VE AÇIKLAMALARI**

**1)Veri Yükleme İşlemi**

**def** Yukle**(**self**,**veridoldur**):**

**from** pandas **import** DataFrame

c**=**len**(**veridoldur**.**columns**)**

r**=**len**(**veridoldur**.**values**)**

self**.**tableWidget**.**setColumnCount**(**r**)**

self**.**tableWidget**.**setRowCount**(**c**)**

self**.**tableWidget**.**setHorizontalHeaderLabels**(**self**.**labels**)**

# print(genelList)

**for** i**,**row **in** enumerate**(**veridoldur**):**

**for** j**,**cell **in** enumerate**(**veridoldur**.**values**):**

self**.**tableWidget**.**setItem**(**i**,**j**,** QtWidgets**.**QTableWidgetItem**(**str**(**cell**[**i**])))**

**def** ekleme**(**self**):**

**import** os

**from** pandas **import** DataFrame

file **=** str**(**QFileDialog**.**getExistingDirectory**(**self**,** "Select Directory"**))**

path**=**file**+**"/"

self**.**liste**=[]**

self**.**labels**=[]**

directories**=**os**.**listdir**(**path**)**

gecici **=** **[]**

sayi **=**int**(**self**.**lineEdit**.**text**())**

directories **=** os**.**listdir**(**path**)**

**for** label\_no**,** directory **in** enumerate**(**directories**):**

gecici**=[]**

self**.**labels**.**append**(**directory**)**

files **=** os**.**listdir**(**path **+** directory**)**

random**.**shuffle**(**files**)**

**for** i**,**j **in** enumerate**(**files**):**

**if** i **==** sayi**:**

**break**

gecici**.**append**(**j**)**

self**.**liste**.**append**(**gecici**)**

self**.**df **=** DataFrame**(**self**.**liste**)**

self**.**Yukle**(**self**.**df**)**

-Veri Seç butonuna basıldığında ilk olarak ekleme fonksiyonu çalışır. Seçilen klasör alınır ve bu klasör içindeki veriler random karıştırılır. Arayüzde istenilen veri sayısı kadar veri çekilir, tabloya yazdırlır.

**2)Tablodan Veri Seçme İşlemi**

**def** cekk**(**self**):**

column **=** self**.**tableWidget**.**currentItem**().**column**()**

row **=** self**.**tableWidget**.**currentItem**().**row**()**

yol **=** **(**self**.**tableWidget**.**item**(**row**,** column**).**text**())**

photo\_path2 **=** "./flowers/"**+**self**.**labels**[**column**]+**"/"**+**yol

self**.**label\_2**.**setPixmap**(**QPixmap**(**photo\_path2**))**

# photo = cv2.imread(photo\_path2)

# print(photo\_path2)

# photo = cv2.resize(photo,(20,30),interpolation = cv2.INTER\_AREA)

sec**=**self**.**comboBox**.**currentText**()**

**if** sec**==**'Rgb'**:**

self**.**label\_2**.**setPixmap**(**QPixmap**(**photo\_path2**))**

**if** sec**==**'Hsv'**:**

hsv **=**cv2**.**imread**(**photo\_path2**)**

hsv**=**color**.**rgb2hsv**(**hsv**)**

hsv **=** img\_as\_ubyte**(**hsv**)**

cv2**.**imwrite**(**"hsv.jpg"**,**hsv**)**

photo\_path2 **=** "./hsv.jpg"

self**.**label\_2**.**setPixmap**(**QPixmap**(**photo\_path2**))**

**if** sec**==**'Cie'**:**

cie **=**cv2**.**imread**(**photo\_path2**)**

cie**=**color**.**rgb2rgbcie**(**cie**)**

cie **=** img\_as\_ubyte**(**cie**)**

cv2**.**imwrite**(**"cie.jpg"**,**cie**)**

photo\_path2 **=** "./cie.jpg"

self**.**label\_2**.**setPixmap**(**QPixmap**(**photo\_path2**))**

image **=** cv2**.**imread**(**photo\_path2**)**

image **=** cv2**.**cvtColor**(**image**,** cv2**.**COLOR\_RGB2GRAY**)**

image **=** cv2**.**resize**(**image**,** **(**28**,**28**))**

image **=** image**.**flatten**()**

#image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

test **=** **[]**

test**.**append**(**image**)**

test **=** np**.**array**(**test**)/**255.0

tahmin **=** self**.**Algdeger**.**predict**(**test**)**

# if self.Algdegercnn!= None:

# tahmin1 = self.Algdegercnn.predict(test).round()

# print(tahmin[0])

self**.**label\_28**.**setText**(**self**.**labels**[**tahmin**[**0**]])**

self**.**label\_26**.**setText**(**self**.**labels**[**column**])**

**if** **(**self**.**labels**[**column**]==**self**.**labels**[**tahmin**[**0**]]):**

self**.**label\_31**.**setText**(**"Doğru Tahmin"**)**

self**.**label\_31**.**setStyleSheet**(**"color: Green"**)**

**else:**

self**.**label\_31**.**setText**(**"Yanlış Tahmin"**)**

self**.**label\_31**.**setStyleSheet**(**"color: Red"**)**

-İlgili renk uzayına göre görüntü arayüzde gösterilir. Her algoritma sonrasında model, self.Algdeger değişkenine aktarılır. Sonrasında cekk fonksiyonunda bulunan self.Algdeger ile predict işlemi yapılır. Tablodan seçilen değere göre sonuç olarak doğru veya yanlış yazdırılır.

**3)Orb Butonuna Basıldığında Yürütülen Kodlar**

**def** OrbAlg**(**self**):**

self**.**Algsayi**=**1

self**.**Sift**()**

-self.Algsayi değeri 1 olur. Bu işlemin amacı sift algoritmasıyla aynı işlevleri olduğu için kodda self.Algsayi’sina bakılarak yapılan değişiklik ile kod tekrarını önlemektir.

**4)Sift Butonuna Basıldığında Yürütülen Kodlar**

**def** Sift**(**self**):**

# crop\_image = cv2.cvtColor(crop\_image, cv2.COLOR\_BGR2GRAY)

# crop\_image=crop\_image.reshape(crop\_image.shape[0],crop\_image.shape[1]\*crop\_image.shape[2])

# print(crop\_image.shape)

**from** pandas **import** DataFrame

**from** skimage**.**transform **import** resize

**from** skimage**.**feature **import** daisy

**from** skimage **import** io**,**color

sec**=**self**.**comboBox**.**currentText**()**

**if** sec**==**'Rgb'**:**

**print(**"---------RGB----------------------"**)**

Xlist**=[]**

Ylist**=[]**

deslist**=[]**

**for** label\_no**,** directory **in** enumerate**(**self**.**labels**):**

**for** i **in** self**.**liste**[**label\_no**]:**

sayac2 **=**int**(**self**.**lineEdit\_2**.**text**())**

img **=**cv2**.**imread**(**'./flowers/'**+**directory**+**'/'**+**i**)**

# try:

img\_width **=** img**.**shape**[**1**]**

img\_height **=** img**.**shape**[**0**]**

# except:

# print("hata")

# break

# img=img.reshape(img.shape[0],img.shape[1]\*img.shape[2])

**if** self**.**Algsayi**==**1**:**

sift **=** cv2**.**ORB\_create**()**

self**.**Algsayi**=**0

**else:**

sift **=** cv2**.**SIFT\_create**()**

kp**,**descss **=** sift**.**detectAndCompute**(**img**,None)**

# img1=cv2.drawKeypoints(gray,kp,img1)

img**=** cv2**.**cvtColor**(**img**,**cv2**.**COLOR\_BGR2GRAY**)**

random**.**shuffle**(**kp**)**

**for** s1**,**i **in** enumerate**(**kp**):**

**if** s1**==**sayac2**:**

**break**

x**,**y **=** int**(**i**.**pt**[**0**]),** int**(**i**.**pt**[**1**])**

n**=**int**(**self**.**lineEdit\_3**.**text**())**

# print(x,y)

**if** **(**x**-**n**)>**0 **and** **(**y**-**n**)>**0 **and** **(**x**+**n**)<**img\_width **and** **(**y**+**n**)<**img\_height**:**

a**=**x**-**n

b**=**x**+**n

c**=**y**-**n

d**=**y**+**n

crop\_image **=** img**[**c**:**d**,** a**:**b**]**

# crop\_image = cv2.rectangle(crop\_image, start\_point, end\_point, color, thickness)

descs**,** descs\_img **=** daisy**(**crop\_image**,** step**=**90**,** radius**=**3**,** rings**=**2**,** histograms**=**5**,**

orientations**=**5**,** visualize**=True)**

# fig, ax = plt.subplots()

# ax.axis('off')

# ax.imshow(descs\_img)

# ax.set\_title('DAISY')

# plt.show()

descs**=**descs**.**reshape**(**descs**.**shape**[**0**],**descs**.**shape**[**1**]\***descs**.**shape**[**2**])**

descs**=**resize**(**descs**,** **(**28**,** 28**))**

descs**=**descs**.**flatten**()**

deslist**.**append**(**descs**)**

Ylist**.**append**(**label\_no**)**

**else:**

sayac2**+=**1

Xlist**=**np**.**array**(**deslist**)**

Ylist**=**np**.**array**(**Ylist**)**

self**.**Xdegerler**=**Xlist

self**.**Ydegerler**=**Ylist

sec1**=**self**.**comboBox\_2**.**currentText**()**

**if** sec1**==**'Knn'**:**

**from** sklearn**.**model\_selection **import** train\_test\_split

tast\_size1**=**float**(**self**.**lineEdit\_4**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_5**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**Xlist**,** Ylist**,** test\_size **=** tast\_size1**,** random\_state **=** 42**)**

self**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

# from sklearn.preprocessing import StandardScaler

# sc\_X=StandardScaler()

# x\_train=sc\_X.fit\_transform(x\_train)

# x\_test=sc\_X.fit\_transform(x\_test)

**from** sklearn**.**neighbors **import** KNeighborsClassifier

knn **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

knn**.**fit**(**x\_train**,** np**.**ravel**(**y\_train**))**

y\_pred**=** knn**.**predict**(**x\_test**)**

self**.**Algdeger**=**knn

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_16**.**setText**(**str**(**round**(**acc**,**2**)))**

**print(**"knn"**,**acc**)**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Knn-Rgb"**)**

self**.**pltRoc**(**y\_test**,**y\_pred**,**"Knn"**)**

# y\_test = y\_test.reshape(-1, 1)

# y\_pred=y\_pred.reshape(-1, 1)

**from** tensorflow**.**keras**.**utils **import** to\_categorical

# y\_test = to\_categorical(y\_test)

# y\_pred = to\_categorical(y\_pred)

# y\_pred=np.argmax(y\_pred, axis=1)

# y\_test=np.argmax(y\_test, axis=1)

# print(y\_test)

# print(y\_pred)

# self.pltRoc2(y\_test,y\_pred,"KnnRoc")

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

x\_deger**=** DataFrame**(**Xlist**)**

y\_deger**=** DataFrame**(**Ylist**)**

X **=** x\_deger**.**values

y **=** y\_deger**.**values

# X = Xlist

# y = Ylist

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

NBG**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

y\_pred **=** NBG**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

**print(**y\_test**.**shape**)**

**print(**y\_pred**.**shape**)**

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

**print(**acc**)**

model **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_17**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

**if** sec1**==**'Rf'**:**

**from** sklearn**.**model\_selection **import** train\_test\_split

tast\_size1**=**float**(**self**.**lineEdit\_4**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_5**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**Xlist**,** Ylist**,** test\_size **=** tast\_size1**,** random\_state **=** 42**)**

self**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

**from** sklearn**.**ensemble **import** RandomForestClassifier

rnd **=** RandomForestClassifier**(**random\_state**=**26**,** n\_jobs **=** **-**1**,**n\_estimators**=**100**)**

rnd**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

y\_pred **=** rnd**.**predict**(**x\_test**)**

self**.**Algdeger**=**rnd

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_16**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Rf-Rgb"**)**

self**.**pltRoc**(**y\_test**,**y\_pred**,**"Rf"**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

x\_deger**=** DataFrame**(**Xlist**)**

y\_deger**=** DataFrame**(**Ylist**)**

X **=** x\_deger**.**values

y **=** y\_deger**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** RandomForestClassifier**(**random\_state**=**26**,** n\_jobs **=** **-**1**,**n\_estimators**=**100**)**

NBG**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

y\_pred **=** NBG**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

**print(**acc**)**

model **=**RandomForestClassifier**(**random\_state**=**26**,** n\_jobs **=** **-**1**,**n\_estimators**=**100**)**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_17**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

**if** sec1**==**"Dt"**:**

**from** sklearn**.**model\_selection **import** train\_test\_split

tast\_size1**=**float**(**self**.**lineEdit\_4**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_5**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**Xlist**,** Ylist**,** test\_size **=** tast\_size1**,** random\_state **=** 42**)**

self**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

**from** sklearn**.**tree **import** DecisionTreeClassifier

c **=** DecisionTreeClassifier**()**

c**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

self**.**Algdeger**=**c

y\_pred**=**c**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_16**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Dt-Rgb"**)**

self**.**pltRoc**(**y\_test**,**y\_pred**,**"Dt"**)**

**print(**"DT"**,**acc**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

x\_deger**=** DataFrame**(**Xlist**)**

y\_deger**=** DataFrame**(**Ylist**)**

X **=** x\_deger**.**values

y **=** y\_deger**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** DecisionTreeClassifier**()**

NBG**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

y\_pred **=** NBG**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

**print(**acc**)**

model **=** DecisionTreeClassifier**()**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_17**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

**if** sec**==**'Hsv'**:**

Xlist**=[]**

Ylist**=[]**

deslist**=[]**

**print(**"-----------HSV--------------------"**)**

**for** label\_no**,** directory **in** enumerate**(**self**.**labels**):**

**for** i **in** self**.**liste**[**label\_no**]:**

sayac2 **=**int**(**self**.**lineEdit\_2**.**text**())**

img **=**cv2**.**imread**(**'./flowers/'**+**directory**+**'/'**+**i**)**

img**=**color**.**rgb2hsv**(**img**)**

img**=** img\_as\_ubyte**(**img**)**

img\_width **=** img**.**shape**[**1**]**

img\_height **=** img**.**shape**[**0**]**

# img=img.reshape(img.shape[0],img.shape[1]\*img.shape[2])

**if** self**.**Algsayi**==**1**:**

sift **=** cv2**.**ORB\_create**()**

self**.**Algsayi**=**0

**else:**

sift **=** cv2**.**SIFT\_create**()**

kp**,**descss **=** sift**.**detectAndCompute**(**img**,None)**

# img1=cv2.drawKeypoints(gray,kp,img1)

img**=** cv2**.**cvtColor**(**img**,**cv2**.**COLOR\_BGR2GRAY**)**

random**.**shuffle**(**kp**)**

**for** s1**,**i **in** enumerate**(**kp**):**

**if** s1**==**sayac2**:**

**break**

# print(s1)

# print(len(kp))

# for i in kp:

x**,**y **=** int**(**i**.**pt**[**0**]),** int**(**i**.**pt**[**1**])**

n**=**int**(**self**.**lineEdit\_3**.**text**())**

# print(x,y)

**if** **(**x**-**n**)>**0 **and** **(**y**-**n**)>**0 **and** **(**x**+**n**)<**img\_width **and** **(**y**+**n**)<**img\_height**:**

a**=**x**-**n

b**=**x**+**n

c**=**y**-**n

d**=**y**+**n

crop\_image **=** img**[**c**:**d**,** a**:**b**]**

# crop\_image = cv2.rectangle(crop\_image, start\_point, end\_point, color, thickness)

# print("asd" ,crop\_image.shape[0])

descs**,** descs\_img **=** daisy**(**crop\_image**,** step**=**90**,** radius**=**3**,** rings**=**2**,** histograms**=**5**,**

orientations**=**5**,** visualize**=True)**

# print("ddd",descs.shape)

# print("resm",descs\_img)

# fig, ax = plt.subplots()

# ax.axis('off')

# ax.imshow(descs\_img)

# ax.set\_title('DAISY')

# plt.show()

descs**=**descs**.**reshape**(**descs**.**shape**[**0**],**descs**.**shape**[**1**]\***descs**.**shape**[**2**])**

descs**=**resize**(**descs**,** **(**28**,** 28**))**

descs**=**descs**.**flatten**()**

deslist**.**append**(**descs**)**

Ylist**.**append**(**label\_no**)**

**else:**

sayac2**+=**1

Xlist**=**np**.**array**(**deslist**)**

Ylist**=**np**.**array**(**Ylist**)**

self**.**Xdegerler**=**Xlist

self**.**Ydegerler**=**Ylist

sec1**=**self**.**comboBox\_2**.**currentText**()**

**if** sec1**==**'Knn'**:**

**from** pandas **import** DataFrame

**from** sklearn**.**model\_selection **import** train\_test\_split

tast\_size1**=**float**(**self**.**lineEdit\_4**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_5**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**Xlist**,** Ylist**,** test\_size **=** tast\_size1**,** random\_state **=** 42**)**

self**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

# from sklearn.preprocessing import StandardScaler

# sc\_X=StandardScaler()

# x\_train=sc\_X.fit\_transform(x\_train)

# x\_test=sc\_X.fit\_transform(x\_test)

**from** sklearn**.**neighbors **import** KNeighborsClassifier

knn **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

knn**.**fit**(**x\_train**,** np**.**ravel**(**y\_train**))**

self**.**Algdeger**=**knn

y\_pred**=** knn**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_16**.**setText**(**str**(**round**(**acc**,**2**)))**

**print(**"knn"**,**acc**)**

# from sklearn.preprocessing import StandardScaler

# sc\_1=StandardScaler()

# y\_test=sc\_1.fit\_transform(y\_test)

# y\_pred=sc\_1.fit\_transform(y\_pred)

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Knn-HSV"**)**

self**.**pltRoc**(**y\_test**,**y\_pred**,**"Knn"**)**

# y\_test = y\_test.reshape(-1, 1)

# y\_pred=y\_pred.reshape(-1, 1)

# print(y\_test)

# print(y\_pred)

# self.pltRoc2(y\_test,y\_pred,"KnnRoc")

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

x\_deger**=** DataFrame**(**Xlist**)**

y\_deger**=** DataFrame**(**Ylist**)**

X **=** x\_deger**.**values

y **=** y\_deger**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

NBG**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

y\_pred **=** NBG**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

**print(**acc**)**

model **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_17**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

**if** sec1**==**'Rf'**:**

**from** sklearn**.**model\_selection **import** train\_test\_split

tast\_size1**=**float**(**self**.**lineEdit\_4**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_5**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**Xlist**,** Ylist**,** test\_size **=** tast\_size1**,** random\_state **=** 42**)**

self**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

**from** sklearn**.**ensemble **import** RandomForestClassifier

rnd **=** RandomForestClassifier**(**random\_state**=**26**,** n\_jobs **=** **-**1**,**n\_estimators**=**100**)**

# rnd.fit(x\_train, np.ravel(y\_train))

rnd**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

self**.**Algdeger**=**rnd

y\_pred **=** rnd**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_16**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Rf-Hsv"**)**

self**.**pltRoc**(**y\_test**,**y\_pred**,**"Rf"**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

x\_deger**=** DataFrame**(**Xlist**)**

y\_deger**=** DataFrame**(**Ylist**)**

X **=** x\_deger**.**values

y **=** y\_deger**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** RandomForestClassifier**(**random\_state**=**26**,** n\_jobs **=** **-**1**,**n\_estimators**=**100**)**

NBG**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

y\_pred **=** NBG**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

**print(**acc**)**

model **=**RandomForestClassifier**(**random\_state**=**26**,** n\_jobs **=** **-**1**,**n\_estimators**=**100**)**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_17**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

**if** sec1**==**"Dt"**:**

**from** sklearn**.**model\_selection **import** train\_test\_split

tast\_size1**=**float**(**self**.**lineEdit\_4**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_5**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**Xlist**,** Ylist**,** test\_size **=** tast\_size1**,** random\_state **=** 42**)**

self**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

**from** sklearn**.**tree **import** DecisionTreeClassifier

c **=** DecisionTreeClassifier**()**

c**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

self**.**Algdeger**=**c

y\_pred**=**c**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Dt-Hsv"**)**

self**.**pltRoc**(**y\_test**,**y\_pred**,**"Dt"**)**

self**.**label\_16**.**setText**(**str**(**round**(**acc**,**2**)))**

**print(**"DT"**,**acc**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

x\_deger**=** DataFrame**(**Xlist**)**

y\_deger**=** DataFrame**(**Ylist**)**

X **=** x\_deger**.**values

y **=** y\_deger**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** DecisionTreeClassifier**()**

NBG**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

y\_pred **=** NBG**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

**print(**acc**)**

model **=** DecisionTreeClassifier**()**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_17**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

**if** sec**==**'Cie'**:**

**print(**"---------CIE----------------------"**)**

Xlist**=[]**

Ylist**=[]**

deslist**=[]**

**for** label\_no**,** directory **in** enumerate**(**self**.**labels**):**

**for** i **in** self**.**liste**[**label\_no**]:**

sayac2 **=**int**(**self**.**lineEdit\_2**.**text**())**

img **=**cv2**.**imread**(**'./flowers/'**+**directory**+**'/'**+**i**)**

img **=** color**.**rgb2rgbcie**(**img**)**

img **=** img\_as\_ubyte**(**img**)**

img\_width **=** img**.**shape**[**1**]**

img\_height **=** img**.**shape**[**0**]**

# img=img.reshape(img.shape[0],img.shape[1]\*img.shape[2])

**if** self**.**Algsayi**==**1**:**

sift **=** cv2**.**ORB\_create**()**

self**.**Algsayi**=**0

**else:**

sift **=** cv2**.**SIFT\_create**()**

kp**,**descss **=** sift**.**detectAndCompute**(**img**,None)**

# img=cv2.drawKeypoints(img,kp,img)

img**=** cv2**.**cvtColor**(**img**,**cv2**.**COLOR\_BGR2GRAY**)**

random**.**shuffle**(**kp**)**

# print("img",img.shape)

# print("kp",len(kp))

# print(img)

**for** s1**,**i **in** enumerate**(**kp**):**

**if** s1**==**sayac2**:**

**break**

x**,**y **=** int**(**i**.**pt**[**0**]),** int**(**i**.**pt**[**1**])**

n**=**int**(**self**.**lineEdit\_3**.**text**())**

**if** **(**x**-**n**)>**0 **and** **(**y**-**n**)>**0 **and** **(**x**+**n**)<**img\_width **and** **(**y**+**n**)<**img\_height**:**

a**=**x**-**n

b**=**x**+**n

c**=**y**-**n

d**=**y**+**n

crop\_image **=** img**[**c**:**d**,** a**:**b**]**

# crop\_image = cv2.rectangle(crop\_image, start\_point, end\_point, color, thickness)

# print("asd" ,crop\_image.shape[0])

descs**,** descs\_img **=** daisy**(**crop\_image**,** step**=**90**,** radius**=**3**,** rings**=**2**,** histograms**=**5**,**

orientations**=**5**,** visualize**=True)**

# print("ddd",descs.shape)

# print("resm",descs\_img)

# fig, ax = plt.subplots()

# ax.axis('off')

# ax.imshow(descs\_img)

# ax.set\_title('DAISY')

# plt.show()

descs**=**descs**.**reshape**(**descs**.**shape**[**0**],**descs**.**shape**[**1**]\***descs**.**shape**[**2**])**

descs**=**resize**(**descs**,** **(**28**,** 28**))**

descs**=**descs**.**flatten**()**

deslist**.**append**(**descs**)**

Ylist**.**append**(**label\_no**)**

**else:**

sayac2**+=**1

Xlist**=**np**.**array**(**deslist**)**

Ylist**=**np**.**array**(**Ylist**)**

self**.**Xdegerler**=**Xlist

self**.**Ydegerler**=**Ylist

sec1**=**self**.**comboBox\_2**.**currentText**()**

**if** sec1**==**'Knn'**:**

**from** pandas **import** DataFrame

**from** sklearn**.**model\_selection **import** train\_test\_split

tast\_size1**=**float**(**self**.**lineEdit\_4**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_5**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**Xlist**,** Ylist**,** test\_size **=** tast\_size1**,** random\_state **=** 42**)**

self**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

# from sklearn.preprocessing import StandardScaler

# sc\_X=StandardScaler()

# x\_train=sc\_X.fit\_transform(x\_train)

# x\_test=sc\_X.fit\_transform(x\_test)

**from** sklearn**.**neighbors **import** KNeighborsClassifier

knn **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

knn**.**fit**(**x\_train**,** np**.**ravel**(**y\_train**))**

self**.**Algdeger**=**knn

y\_pred**=** knn**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**label\_16**.**setText**(**str**(**round**(**acc**,**2**)))**

**print(**"knn"**,**acc**)**

# from sklearn.preprocessing import StandardScaler

# sc\_1=StandardScaler()

# y\_test=sc\_1.fit\_transform(y\_test)

# y\_pred=sc\_1.fit\_transform(y\_pred)

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Knn-Cie"**)**

self**.**pltRoc**(**y\_test**,**y\_pred**,**"Knn"**)**

# y\_test = y\_test.reshape(-1, 1)

# y\_pred=y\_pred.reshape(-1, 1)

# print(y\_test)

# print(y\_pred)

# self.pltRoc2(y\_test,y\_pred,"KnnRoc")

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

x\_deger**=** DataFrame**(**Xlist**)**

y\_deger**=** DataFrame**(**Ylist**)**

X **=** x\_deger**.**values

y **=** y\_deger**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

NBG**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

y\_pred **=** NBG**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

**print(**acc**)**

model **=** KNeighborsClassifier**(**n\_neighbors**=**5**)**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_17**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

**if** sec1**==**'Rf'**:**

**from** sklearn**.**model\_selection **import** train\_test\_split

tast\_size1**=**float**(**self**.**lineEdit\_4**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_5**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**Xlist**,** Ylist**,** test\_size **=** tast\_size1**,** random\_state **=** 42**)**

self**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

**from** sklearn**.**ensemble **import** RandomForestClassifier

rnd **=** RandomForestClassifier**(**random\_state**=**26**,** n\_jobs **=** **-**1**,**n\_estimators**=**100**)**

rnd**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

self**.**Algdeger**=**rnd

y\_pred **=** rnd**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Rf-Cie"**)**

self**.**label\_16**.**setText**(**str**(**round**(**acc**,**2**)))**

self**.**pltRoc**(**y\_test**,**y\_pred**,**"Rf"**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

x\_deger**=** DataFrame**(**Xlist**)**

y\_deger**=** DataFrame**(**Ylist**)**

X **=** x\_deger**.**values

y **=** y\_deger**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** RandomForestClassifier**(**random\_state**=**26**,** n\_jobs **=** **-**1**,**n\_estimators**=**100**)**

NBG**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

y\_pred **=** NBG**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

**print(**acc**)**

model **=**RandomForestClassifier**(**random\_state**=**26**,** n\_jobs **=** **-**1**,**n\_estimators**=**100**)**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_17**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

**print(**"rn"**,**acc**)**

**if** sec1**==**"Dt"**:**

**from** sklearn**.**model\_selection **import** train\_test\_split

tast\_size1**=**float**(**self**.**lineEdit\_4**.**text**())**

fold\_size**=**int**(**self**.**lineEdit\_5**.**text**())**

self**.**foldsizee**=**fold\_size

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**Xlist**,** Ylist**,** test\_size **=** tast\_size1**,** random\_state **=** 42**)**

self**.**csvYukle**(**x\_train**,**y\_train**,**x\_test**,**y\_test**)**

**from** sklearn**.**tree **import** DecisionTreeClassifier

c **=** DecisionTreeClassifier**()**

c**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

self**.**Algdeger**=**c

y\_pred**=**c**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**Cmatrix**(**y\_test**,**y\_pred**,**"Dt-Cie"**)**

self**.**pltRoc**(**y\_test**,**y\_pred**,**"Dt"**)**

self**.**label\_16**.**setText**(**str**(**round**(**acc**,**2**)))**

**print(**"DT"**,**acc**)**

**from** sklearn**.**model\_selection **import** KFold

**from** numpy **import** mean

**from** sklearn**.**model\_selection **import** cross\_val\_score

x\_deger**=** DataFrame**(**Xlist**)**

y\_deger**=** DataFrame**(**Ylist**)**

X **=** x\_deger**.**values

y **=** y\_deger**.**values

kf **=** KFold**(**n\_splits**=**fold\_size**)**

kf**.**get\_n\_splits**(**X**)**

sayma**=**0

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

sayma**+=**1

# print("TRAIN:", train\_index, "TEST:", test\_index)

x\_train**,** x\_test **=** X**[**train\_index**],** X**[**test\_index**]**

y\_train**,** y\_test **=** y**[**train\_index**],** y**[**test\_index**]**

NBG **=** DecisionTreeClassifier**()**

NBG**.**fit**(**x\_train**,**np**.**ravel**(**y\_train**))**

y\_pred **=** NBG**.**predict**(**x\_test**)**

acc**=**accuracy\_score**(**y\_test**,** y\_pred**)\***100

self**.**CmatrixFold**(**y\_test**,**y\_pred**,**"fold-"**+**str**(**sayma**))**

self**.**kfoldCmatrix**(**y\_test**,** y\_pred**,**"Fold Sonuç"**)**

**print(**acc**)**

model **=** DecisionTreeClassifier**()**

scores **=** cross\_val\_score**(**model**,** X**,** y**,** scoring**=**'accuracy'**,** cv**=**kf**,** n\_jobs**=-**1**)**

self**.**label\_17**.**setText**(**str**(**round**(**mean**(**scores**\***100**),**2**)))**

**print(**'Accuracy: %.3f (%.3f)' **%** **(**mean**(**scores**),** scores**.**max**()))**

**print(**"--------------------"**)**

**def** Cmatrix**(**self**,**y\_test**,**y\_pred**,**isim**):**

cm **=** confusion\_matrix**(**y\_test**,** y\_pred**)**

# classNames = ['0','1',"2","3","4"]

cm\_data **=** pd**.**DataFrame**(**cm**)**

plt**.**figure**(**figsize **=** **(**5**,**4**))**

sns**.**heatmap**(**cm\_data**,** annot**=True,**fmt**=**"d"**)**

plt**.**title**(**isim**)**

plt**.**ylabel**(**'Actual label'**)**

plt**.**xlabel**(**'Predicted label'**)**

plt**.**savefig**(**'cm1.png'**)**

plt**.**show**()**

photo\_path2 **=** "./cm1.png"

self**.**label\_11**.**setPixmap**(**QPixmap**(**photo\_path2**))**

**def** pltRoc2**(**self**,**y\_test**,**y\_pred**,**baslik**):**

**from** sklearn**.**metrics **import** roc\_curve

**from** sklearn**.**metrics **import** roc\_auc\_score

**from** matplotlib **import** pyplot

lr\_auc **=** roc\_auc\_score**(**y\_test**,** y\_pred**)**

# summarize scores

**print(**'ALGRTM: ROC AUC=%.3f' **%** **(**lr\_auc**))**

# calculate roc curves

lr\_fpr**,** lr\_tpr**,** \_ **=** roc\_curve**(**y\_test**,** y\_pred**)**

# plot the roc curve for the model

pyplot**.**plot**(**lr\_fpr**,** lr\_tpr**,** marker**=**'.'**,** label**=**baslik**)**

# axis labels

pyplot**.**xlabel**(**'False Positive Rate'**)**

pyplot**.**ylabel**(**'True Positive Rate'**)**

# show the legend

pyplot**.**legend**()**

pyplot**.**show**()**

**def** CmatrixFold**(**self**,**y\_test**,**y\_pred**,**isim**):**

cm **=** confusion\_matrix**(**y\_test**,** y\_pred**)**

# classNames = ['0','1',"2","3","4"]

cm\_data **=** pd**.**DataFrame**(**cm**)**

plt**.**figure**(**figsize **=** **(**5**,**5**))**

sns**.**heatmap**(**cm\_data**,** annot**=True,**fmt**=**"d"**)**

plt**.**title**(**isim**)**

plt**.**ylabel**(**'Actual label'**)**

plt**.**xlabel**(**'Predicted label'**)**

plt**.**show**()**

**def** pltRoc**(**self**,**y\_test**,**y\_pred**,**baslik**):**

**from** sklearn**.**metrics **import** roc\_curve

**from** sklearn**.**metrics **import** roc\_auc\_score

**from** sklearn**.**linear\_model **import** LogisticRegression

**from** sklearn **import** metrics

y\_test**=**np**.**array**(**y\_test**)**

y\_pred**=**np**.**array**(**y\_pred**)**

postotal**=**0

**for** i **in** range**(**4**):**

**if** np**.**count\_nonzero**(**y\_pred **==** i**)!=**0**:**

postotal**+=**1

lr\_fpr**,** lr\_tpr**,** thresholds **=**metrics**.**roc\_curve**(**y\_test**,** y\_pred**,** pos\_label**=**postotal**)**

plt**.**plot**(**lr\_fpr**,** lr\_tpr**,** marker**=**'.'**,** label**=**'baslik'**)**

#axis labels

plt**.**xlabel**(**'False Positive Rate'**)**

plt**.**ylabel**(**'True Positive Rate'**)**

#show the legend

plt**.**legend**()**

plt**.**savefig**(**'roc\_klasik.png'**)**

plt**.**show**()**

photo\_path2 **=** "./roc\_klasik.png"

self**.**label\_33**.**setPixmap**(**QPixmap**(**photo\_path2**))**

-Arayüzde belitilen veri sayısı kadar değer alınır. Self. Algsayi değeri varsayılan 0 olarak belirlenmiştir. Orb butonuna basıldığında bu değer 1 olur ve ilgili kod çalışır.

-Arayüzde belirtilen miktarda key point alınma işlemi için ilk olarak key pointler random sıralanır ve belirtilen sayı kadar alınır.

-Her keypoint için crop noktaları belirlenir ve croplu resimler daisy için işleme alınır.

-Bu işlem sonucunda çıkan öznitelikler X listesine, Labller ise y listesine alınır.

-Arayüzden seçilen renk uzayına ve algoritmaya göre ilgili kod kısmı

işlettirilir. Bu değerler hold-out ile Confusion matrix, Roc eğrisi ve Overlapped matrix olarak ekranda gösterilir.

**5)Cnn Butonuna Basıldığında Yürütülen Kodlar**

**def** CnnAlgrt**(**self**):**

x**=** self**.**Xdegerler

y**=** self**.**Ydegerler

**from** sklearn**.**preprocessing **import** StandardScaler

sc\_X**=**StandardScaler**()**

x**=**sc\_X**.**fit\_transform**(**x**)**

# x\_test=sc\_X.fit\_transform(x\_test)

tast\_size1**=**float**(**self**.**lineEdit\_4**.**text**())**

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**x**,** y**,** test\_size **=**tast\_size1 **,** random\_state **=** 42**)**

**from** keras**.**utils **import** to\_categorical

y\_train **=** to\_categorical**(**y\_train**,** 5**)**

y\_test**=** to\_categorical**(**y\_test**,** 5**)**

**from** keras**.**models **import** Sequential

**from** keras**.**layers **import** Dense**,**Dropout**,**BatchNormalization**,**Activation

#modeli oluşturalım

model **=** Sequential**()**

#eğitim verisinde kaç tane stun yani model için girdi sayısı var onu alalım

n\_cols **=** x\_train**.**shape**[**1**]**

#model katmanlarını ekleyelim

model**.**add**(**Dense**(**16**,** input\_shape**=(**n\_cols**,)))**

model**.**add**(**Activation**(**"relu"**))**

model**.**add**(**BatchNormalization**())**

model**.**add**(**Dropout**(**0.5**))**

model**.**add**(**Dense**(**9**))**

model**.**add**(**Activation**(**"relu"**))**

model**.**add**(**BatchNormalization**())**

model**.**add**(**Dropout**(**0.5**))**

model**.**add**(**Dense**(**6**))**

model**.**add**(**Activation**(**"relu"**))**

model**.**add**(**BatchNormalization**())**

model**.**add**(**Dropout**(**0.5**))**

model**.**add**(**Dense**(**5**,** activation**=**'softmax'**))**

model**.**summary**()**

model**.**compile**(**optimizer**=**'adam'**,** loss**=**'categorical\_crossentropy'**,** metrics**=[**'accuracy'**])**

history **=**model**.**fit**(**x\_train**,**

y\_train**,**

validation\_data**=(**x\_test**,** y\_test**),**

batch\_size**=**16**,**

shuffle**=True,**

verbose**=**1**,**

epochs**=**10**)**

score **=** model**.**evaluate**(**x\_test**,** y\_test**,** verbose**=**0**)**

**print(**'Test loss:'**,** score**[**0**])**

**print(**'Test accuracy:'**,** score**[**1**])**

**from** matplotlib **import** pyplot **as** plt

# Plot training & validation accuracy values

# plt.figure(figsize=(14,3))

plt**.**subplot**(**1**,** 2**,** 1**)**

plt**.**plot**(**history**.**history**[**'accuracy'**])**

plt**.**plot**(**history**.**history**[**'val\_accuracy'**])**

plt**.**title**(**'Model accuracy'**)**

plt**.**ylabel**(**'Accuracy'**)**

plt**.**xlabel**(**str**(**round**(**score**[**1**]\***100**,**3**)))**

plt**.**legend**([**'Train'**,** 'Test'**],** loc**=**'upper left'**)**

# Plot training & validation loss values

plt**.**subplot**(**1**,** 2**,** 2**)**

plt**.**plot**(**history**.**history**[**'loss'**])**

plt**.**plot**(**history**.**history**[**'val\_loss'**])**

plt**.**title**(**'Model loss'**)**

plt**.**ylabel**(**'Loss'**)**

plt**.**xlabel**(**str**(**round**(**score**[**0**],**3**)))**

plt**.**legend**([**'Train'**,** 'Test'**],** loc**=**'upper left'**)**

plt**.**savefig**(**'acc\_loss.png'**)**

photo\_path3 **=** "./acc\_loss.png"

self**.**label\_23**.**setPixmap**(**QPixmap**(**photo\_path3**))**

plt**.**show**()**

self**.**Algdegercnn**=**model

**print(**'----Sonuç-----'**)**

score **=** model**.**evaluate**(**x\_test**,** y\_test**,** verbose**=**0**)**

**print(**'Test loss:'**,** score**[**0**])**

**print(**'Test accuracy:'**,** score**[**1**])**

y\_pred **=** model**.**predict**(**x\_test**)**

y\_test **=** y\_test**.**reshape**(-**1**,** 1**)**

y\_pred**=**y\_pred**.**reshape**(-**1**,** 1**)**

# print(confusion\_matrix(y\_test, y\_pred.round()))

y\_pred2**=**y\_pred**.**round**()**

self**.**Cmatrixcnn**(**y\_test**,**y\_pred2**,**"Derin Öğrenme"**)**

self**.**pltRocCnn**(**y\_test**,**y\_pred**,**"Derin Öğrenme"**)**

-Cnn ile model oluşturulur epoch sayısı 10, batch size 16 olan bir eğitim gerçekleştirilir. Sonuç olarak acc-loss grafikleri, Confusion matrixi ve Roc eğrisi gösterilir.