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

İş Akışı 3:

_Orjinal görüntüler için	Veri arttırım tekniği ile elde edilen görüntüler için
• RGB • HSV • CIE Renk uzayları üzerinde ayrı ayrı	

A) SURF ve SIFT algoritmaları ile keypoint tespiti ve bu keypointler etrafında nxn lik Daisy özniteliklerin çıkarımı + Klasik MÖ algoritmaları
B) Kendi derin öğrenme modeliniz

Dataset C: https://www.kaggle.com/alxmamaev/flowers-recognition adresinde çiçek çeşitleri ile ilgili bir verseti bulunmaktadır.

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.

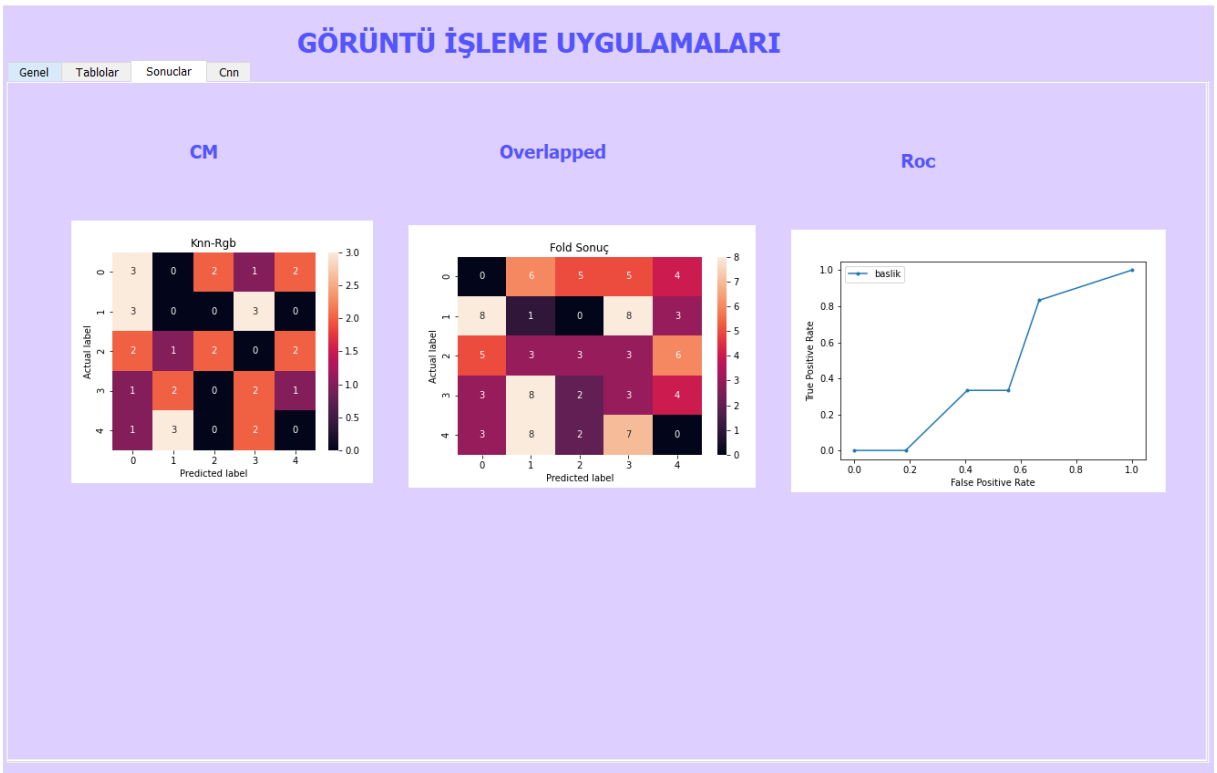
1)Sift Butonuna Basıldığında Yürtülen İşlem

RGB İçin

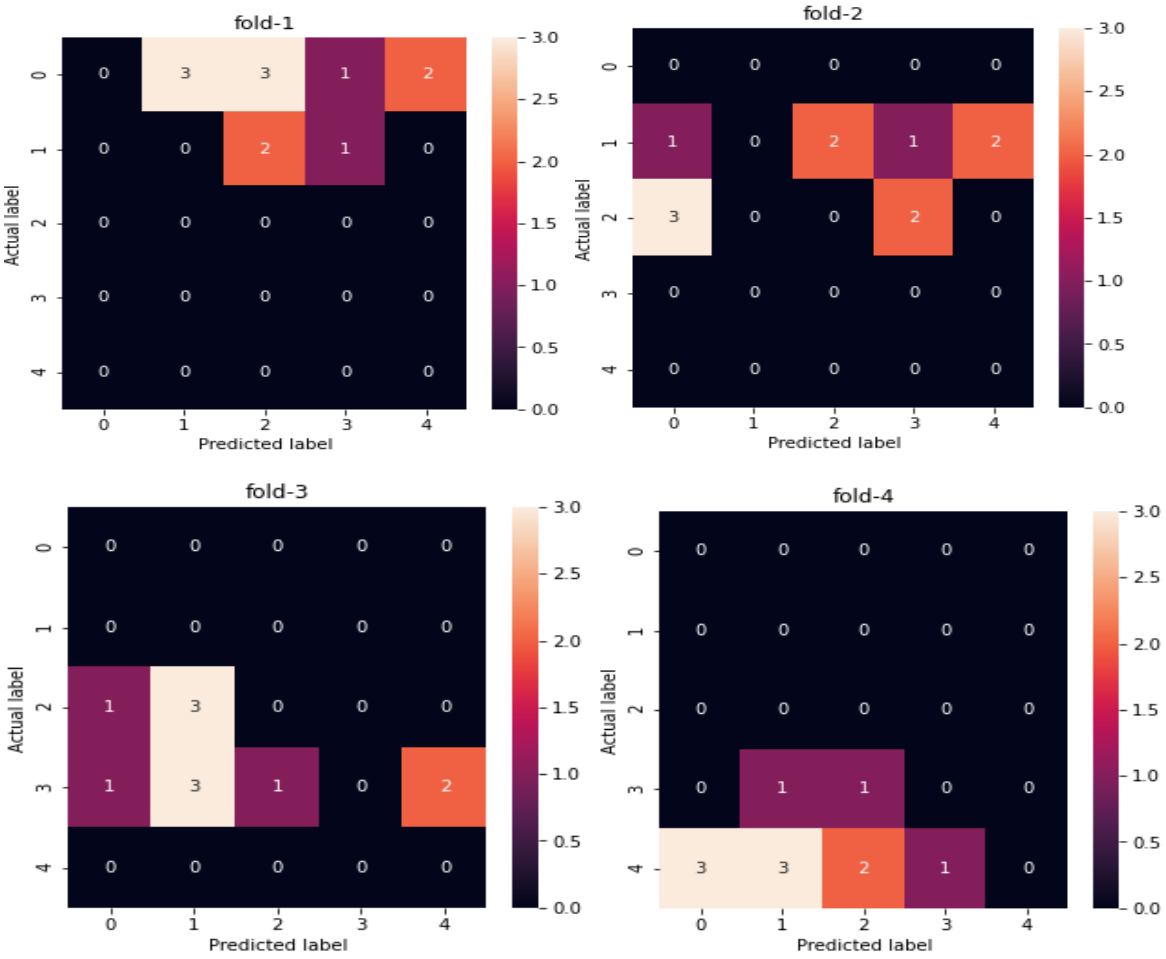
HSV İçin

CİE İçin

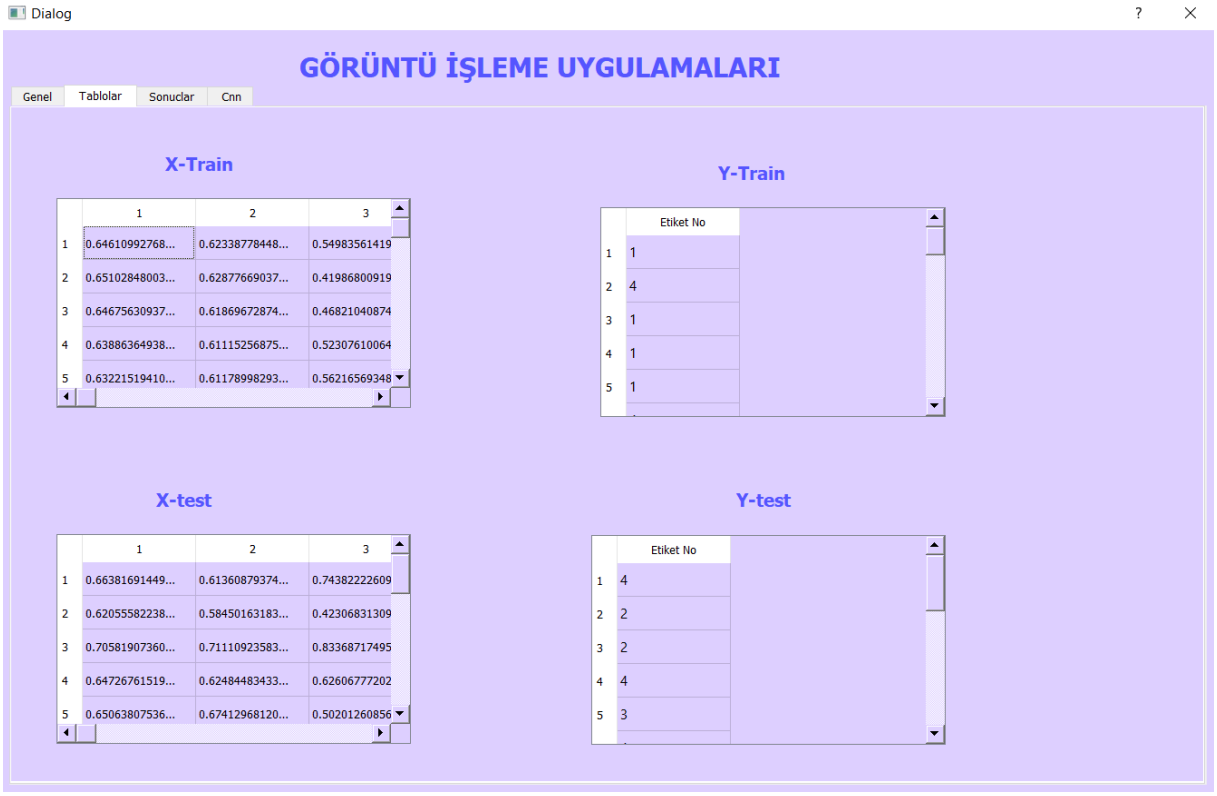
-Sift algoritması çalışır ve sonucu labellara yazılır. Tablodan seçim yapılarak model doğruluğu görüntüde olduğu gibi sınanır.



-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.