KASTAMONU ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
GÖRÜNTÜ İŞLEME DERSİ ÖDEV RAPORU

ÖDEV

DataSetB'yi İşakışı 2'ye Göre Uygulanması

ÖDEV TARİHİ

22.01.2020

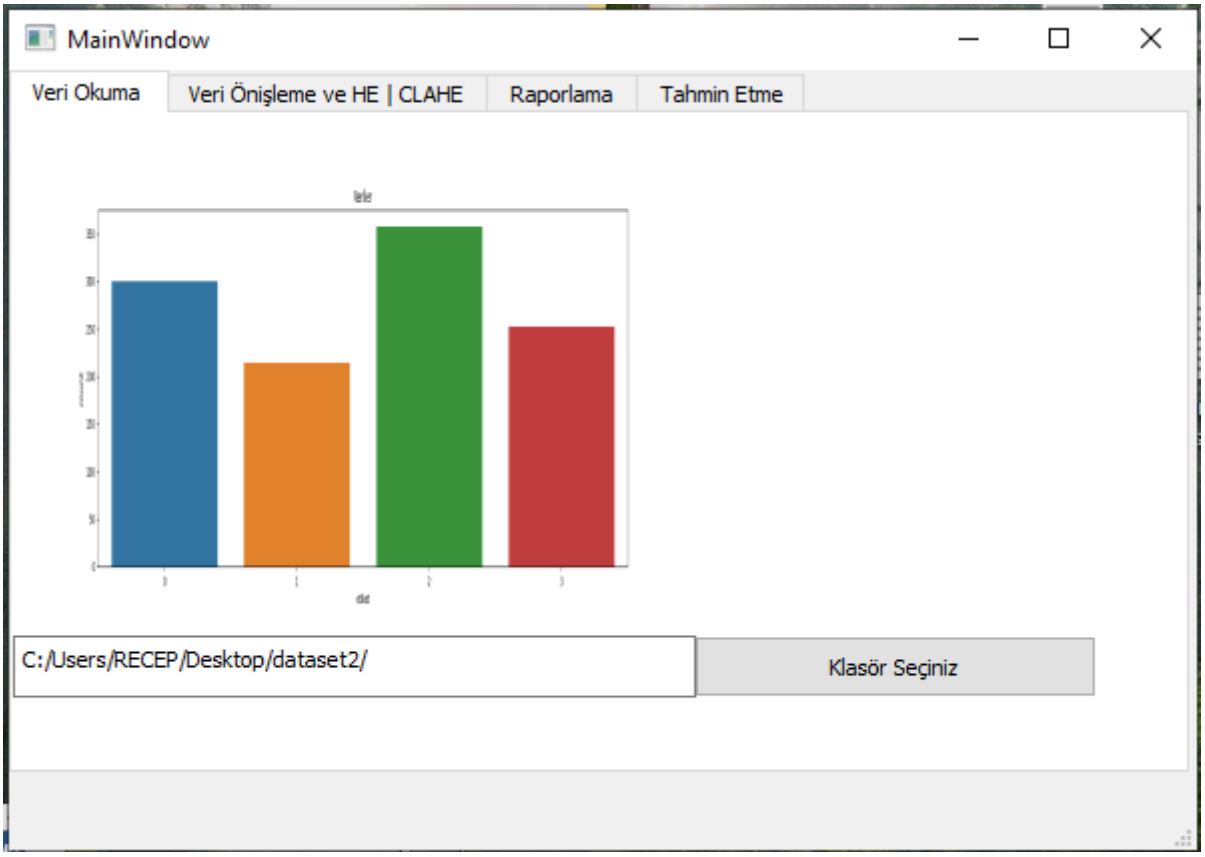
DERSİN SORUMLUSU

Kemal AKYOL

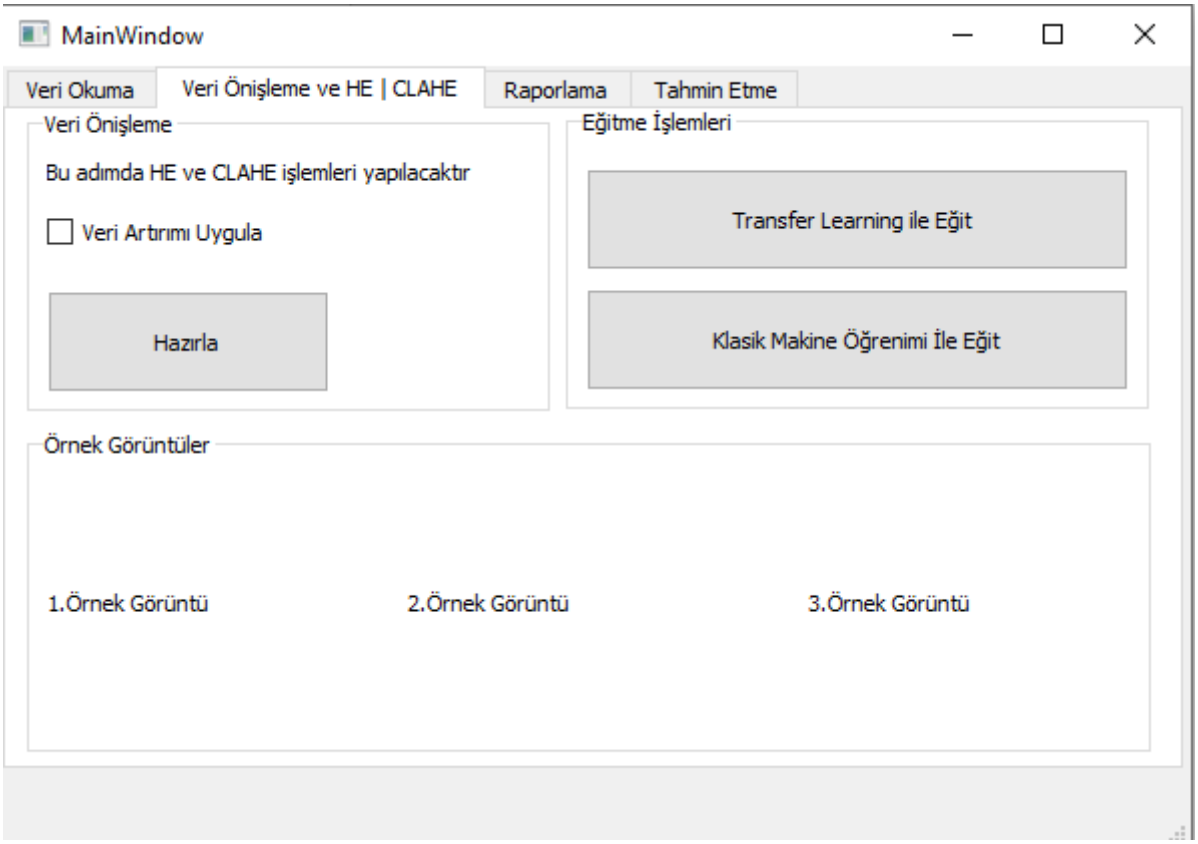
RAPORU YAZAN ÖĞRENCİ

Recep POLAT

EKRAN RESİMLERİ

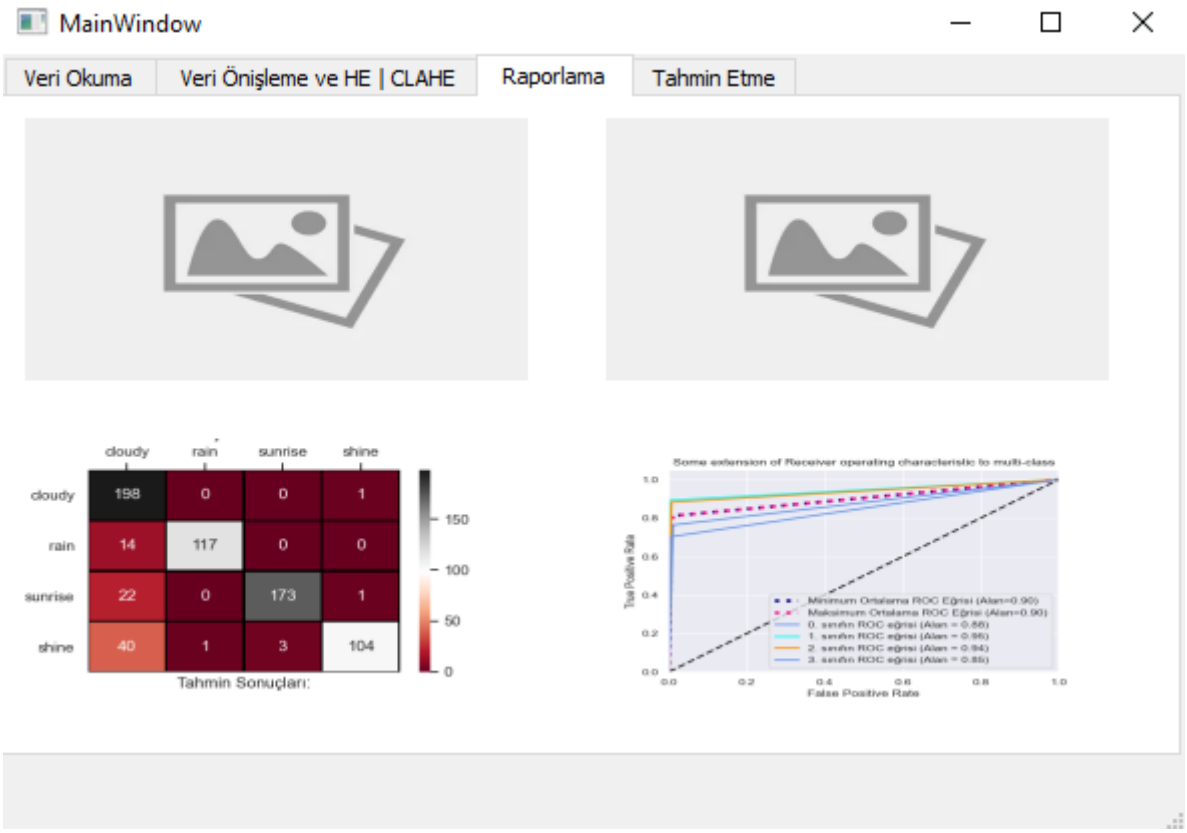


- Buradan işlem yapacağımız veri setini seçiyoruz.
- Seçtiğimiz veri setindeki sınıf sayısını görüyoruz.

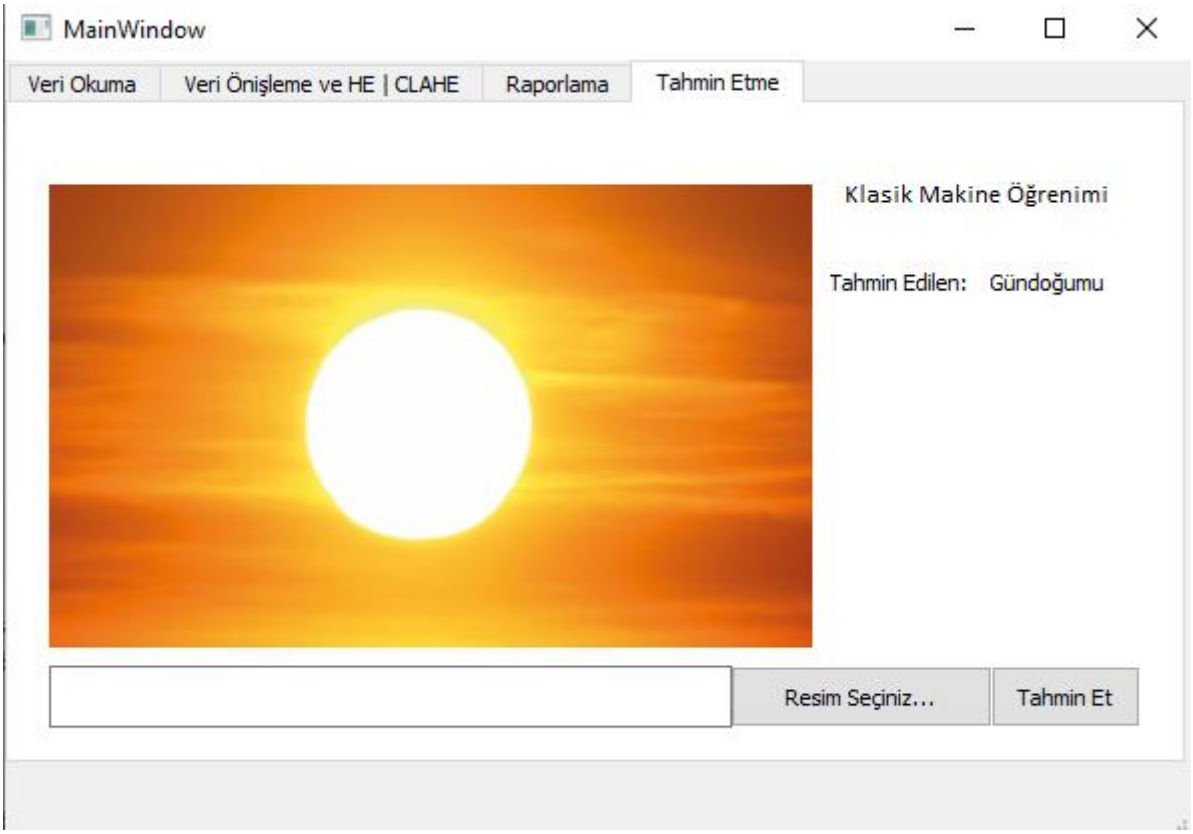


- Bu adımda HE ve CLAHE işlemleri Hazırla butonuna basıldığında gerçekleşir ve eğer Veri artırımı uygula seçeneği seçili ise önce veri artırma uygular ve bu görsellere de HE ve CLAHE uygular.
- Transfer Learning ile Eğit butonuna basılırsa derin öğrenme ve yapay sinir ağları ile eğitim gerçekleşir.
- Klasik Makine Öğrenimi İle Eğit butonuna basıldığında ise Random Forest Algoritması ile sınıflandırma işlemi yapılır.
- Derin Öğrenme modeli olarak ImageNet ağırlıkları ve Xception modeli kullanılmıştır.
- Örnek görüntüler kısmına ise HE, CLAHE ve normal görüntülerden rasgele birer tane seçilir ve yazdırılır.

A)Klasik Makine Öğrenimi

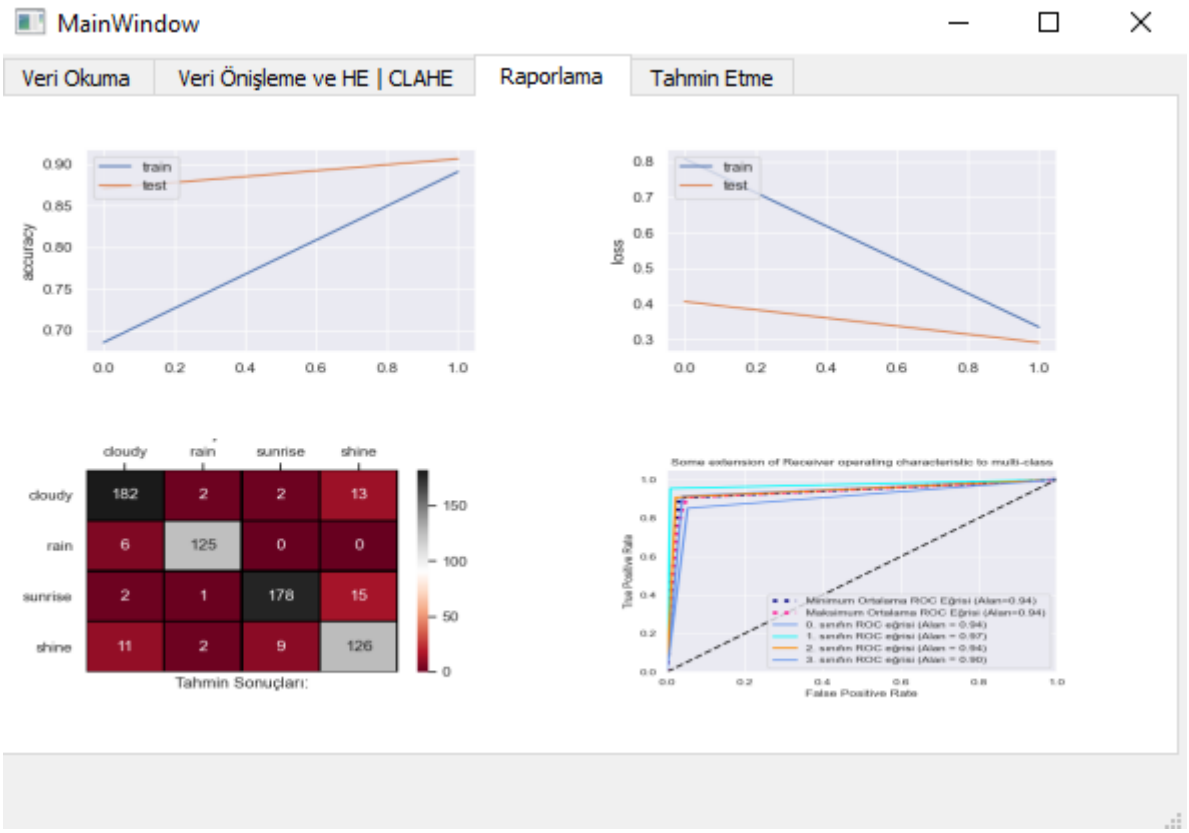


Bir önceki sayfada eğitilen Random Forest Modelinin Test verisi üzerinde çıkardığı sonuçların Karışıklık Matrisi ve ROC Eğrisi grafikleri gösterilmiştir.



Burada Random Forest modelinin görsel üzerinde yaptığı tahmin işlemi yapılır.

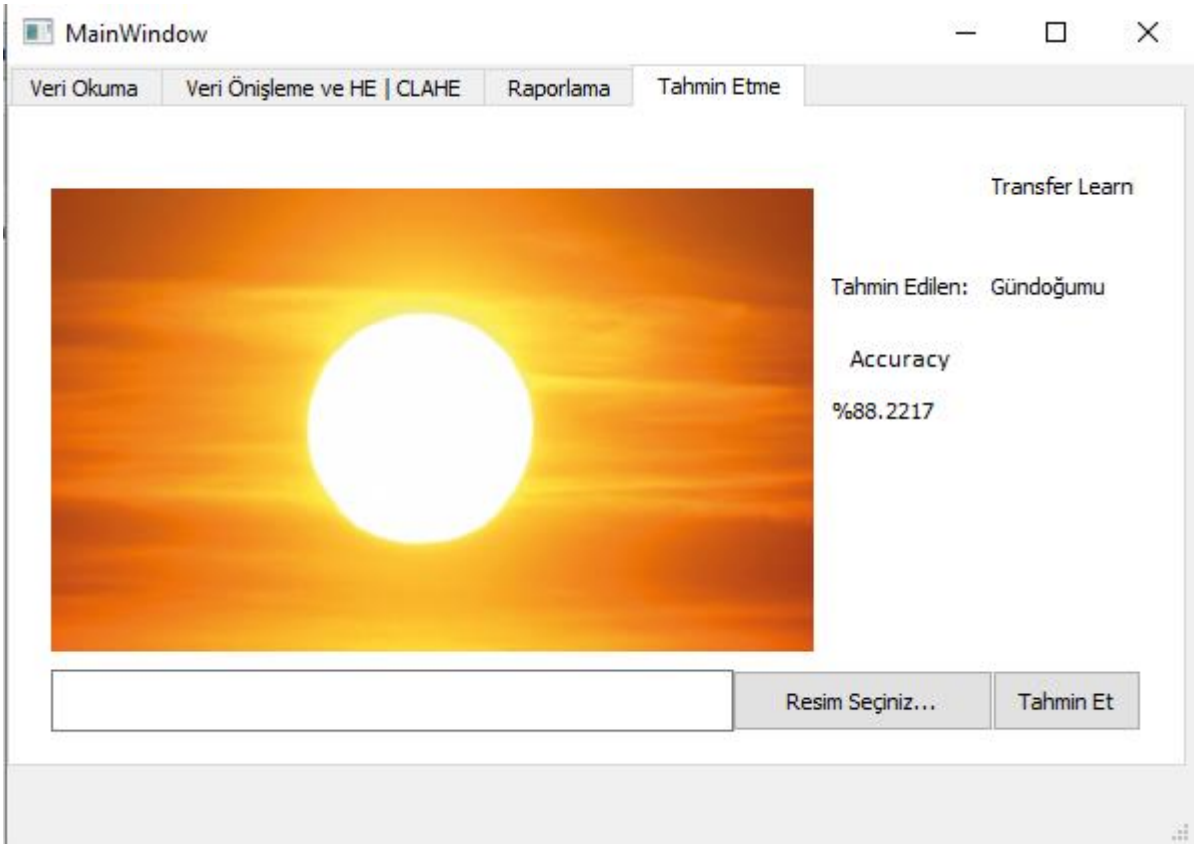
B)Derin Öğrenme



Burada da Derin Öğrenme modelinin test verisi üzerinde çıkan sonuçları ve Accuracy ve Loss Grafiklerini gösteriyorum.

```
Transfer Learning İşlemi Başladı.
Epoch 1/2
22/22 [==============================] - 64s 3s/step - loss: 1.1246 - accuracy: 0.5406 - val_loss: 0.4072 - val_accuracy: 0.8709
Epoch 2/2
22/22 [==============================] - 62s 3s/step - loss: 0.3609 - accuracy: 0.8805 - val_loss: 0.2925 - val_accuracy: 0.9065
Transfer Learning İşlemi Bitti.
```

Burada ise konsolda aldığım Epoch başı accuracy ve loss değerlerini gösteriyorum.



Burada Derin Öğrenme modelinin görsel üzerinde yaptığı tahmin işlemi yapılır.

KOD PARÇALARI

```
#Nesne Events
self.pbVeriOkuma.clicked.connect(self.pbVeriOkumaClick)
self.pbVeriOnisleme.clicked.connect(self.pbVeriOnislemeClick)
self.pbTrainTransferLearning.clicked.connect(self.pbTrainTransferLearning
Click)
self.pbCML.clicked.connect(self.pbCMLClick)
self.pbTahminEtGorsel.clicked.connect(self.pbTahminEtGorselClick)
self.pbTahminEt.clicked.connect(self.pbTahminEtClick)
#Değişkenler
self.labeled = []
self.he = []
self.CLAhe = []
self.array = []
self.veriler = []
self.etiketler = []
self.uzanti = []
self.X_train = []
self.X_test = []
self.y_train = []
self.y_test = []
self.im_Size = (150,150)
self.isClassic = False
self.TahminEdilecekFotografinUzantisi = None
```

```
#Buttonların Clickleri
#Veri Okuma
def pbVeriOkumaClick(self):
    try:
        folder = str(QtWidgets.QFileDialog.getExistingDirectory(self,
"Resim klasörünüzü seçin"))
        folder = folder + "/"
        self.teFolderName.setPlainText(folder)
        self.dataRead(folder)
    except Exception as e:
        print(e)
#Burada veri setini seçip değişkene uzantıyı alıyoruz.
#Veri ÖnİŞleme
def pbVeriOnislemeClick(self):
    if self.cbVeriArtirimi.isChecked():
        self.veriArtirma()
    self.histogramE()
    self.CLAhistogramE()
    self.birlestirme()
#Burada HE ve CLAHE yapıp veri artırma yapıp yapılmayacağını
sorguluyoruz.
```

```
#Transfer Learning İşlemini burada yapıyoruz
def pbTrainTransferLearningClick(self):
    print("Transfer Learning İşlemi Başladı.")
    self.isClassic = False
    inputs = keras.Input(shape=(150, 150, 3))
    x = self.base_model(inputs, training=False)
    x = keras.layers.GlobalMaxPooling2D()(x)
    outputs = keras.layers.Dense(4, activation="softmax")(x)
    model = keras.Model(inputs, outputs)
    model.compile(optimizer="adam",  # hep Adam
                  loss="categorical_crossentropy",
                  metrics=["accuracy"])
    result = model.fit(self.X_train, self.y_train, batch_size=128,
epochs=10, verbose=1,
                       validation_data=(self.X_test, self.y_test))
    self.GlobalModel = model

    # B
    y_pred = model.predict(self.X_test, batch_size=128)
    y_pred = np.argmax(y_pred, axis=1)
    y_test_encoded = np.argmax(self.y_test, axis=1)
    classificationResult = classification_report(y_test_encoded, y_pred,
                                                 target_names=["cloudy",
"rain", "sunrise", "shine"])
    confusionMatrix = confusion_matrix(y_test_encoded, y_pred)
    pyp.figure(figsize=(5, 3))
    sns.set(font_scale=1)
    canvas = sns.heatmap(confusionMatrix, annot=True,
xticklabels=["cloudy", "rain", "sunrise", "shine"],
                         yticklabels=["cloudy", "rain", "sunrise",
"shine"],
                         cmap="RdGy", linewidths=1, linecolor="black",
fmt=".0f")
    pyp.yticks(rotation=0)
    pyp.xlabel("Tahmin Sonuçları:")
    pyp.ylabel("Gerçek Sonuçlar:")
    canvas.xaxis.set_ticks_position("top")
    pyp.title("Karışıklık Matrisi")
```

```
    pyp.savefig("karmasiklikMatris.png")
    pyp.cla()
    pyp.clf()

    falsePositive = dict()
    truePositive = dict()
    aucRoc = dict()
    y_pred_oneHot = to_categorical(y_pred)
    for i in range(4):
        falsePositive[i], truePositive[i], c = roc_curve(self.y_test[:,
i], y_pred_oneHot[:, i])
        aucRoc[i] = auc(falsePositive[i], truePositive[i])

    falsePositive["min"], truePositive["min"], c =
roc_curve(self.y_test.ravel(), y_pred_oneHot.ravel())
    aucRoc["min"] = auc(falsePositive["min"], truePositive["min"])
    fp_all = np.unique(np.concatenate([falsePositive[i] for i in
range(4)]))
    tp_ort = np.zeros_like(fp_all)
    for i in range(4):
        tp_ort += np.interp(fp_all, falsePositive[i], truePositive[i])
    tp_ort = tp_ort / 4
    falsePositive["max"] = fp_all
    truePositive["max"] = tp_ort
    aucRoc["max"] = auc(falsePositive["max"], truePositive["max"])
    pyp.plot(result.history["accuracy"])
    pyp.plot(result.history["val_accuracy"])
    pyp.ylabel("accuracy")
    pyp.xlabel("epoch")
    pyp.legend(["train", "test"], loc="upper left")
    pyp.savefig("Accuracy.png")
    pyp.cla()
    pyp.clf()

    pyp.plot(result.history["loss"])
    pyp.plot(result.history["val_loss"])
    pyp.ylabel("loss")
    pyp.xlabel("epoch")
    pyp.legend(["train", "test"], loc="upper left")
    pyp.savefig("Loss.png")
    pyp.cla()
    pyp.clf()

    pyp.figure()
    pyp.plot(falsePositive["min"], truePositive["min"],
             label="Minimum Ortalama ROC Eğrisi
(Alan={0:0.2f})".format(aucRoc["min"]), color="navy", linestyle=":",
             linewidth=4)
    pyp.plot(falsePositive["max"], truePositive["max"],
             label="Maksimum Ortalama ROC Eğrisi
(Alan={0:0.2f})".format(aucRoc["max"]), color="deeppink",
             linestyle=":", linewidth=4)
    colors = cycle(["cornflowerblue", "aqua", "darkorange"])

    for i, color in zip(range(4), colors):
        pyp.plot(falsePositive[i], truePositive[i], color=color, lw=2,
                 label='{0}. sınıfın ROC eğrisi (Alan = {1:0.2f})'
                       ''.format(i, aucRoc[i]))

    pyp.plot([0, 1], [0, 1], 'k--', lw=2)
    pyp.xlim([0.0, 1.0])
```

```
    pyp.ylim([0.0, 1.05])
    pyp.xlabel('False Positive Rate')
    pyp.ylabel('True Positive Rate')
    pyp.title('Some extension of Receiver operating characteristic to
multi-class')
    pyp.legend(loc="lower right")
    pyp.savefig("ROCegrisi.png")
    pyp.cla()
    pyp.clf()
    self.lblROC.setPixmap(QtGui.QPixmap("./ROCegrisi.png"))

self.lblKarmasiklikMatrisi.setPixmap(QtGui.QPixmap("./karmasiklikMatris.p
ng"))
    self.lblLoss.setPixmap(QtGui.QPixmap("./Loss.png"))
    self.lblAccuracy.setPixmap(QtGui.QPixmap("./Accuracy.png"))
    print("Transfer Learning İşlemi Bitti.")
```

#Burada Xception modelini kullanarak makineyi öğretiyoruz ve Karışıklık Matrisi, ROC eğrisi, Accuracy ve Loss grafiklerini oluşturup bu grafikleri ".png" formatında proje path'imize kaydediyorum. Ve Raporlamada gösteriyorum.

```
#Klasik Makine Öğrenmesi
def pbCMLClick(self):
    print("Klasik Makine Öğrenmesi İşlemi Başladı.")
    self.isClassic = True
    xfeatureExtractor = self.base_model.predict(self.X_train)  # FE
    X_train_features =
xfeatureExtractor.reshape(xfeatureExtractor.shape[0], -1)
    xTefeatureExtractor = self.base_model.predict(self.X_test)  # FE
    X_test_features =
xTefeatureExtractor.reshape(xTefeatureExtractor.shape[0], -1)
    rfModel = RandomForestClassifier(random_state=30, n_estimators=40)  #
klasik makine öğrenmesi algoritması
    self.mainExtractor = self.base_model
    rfModel.fit(X_train_features, self.y_train)
    self.GlobalModel = rfModel

    y_Tahmin = rfModel.predict(X_test_features)
    yTest = np.argmax(self.y_test, axis=1)
    yTahmin = np.argmax(y_Tahmin, axis=1)
    confusionMatrix = confusion_matrix(yTest, yTahmin)
    pyp.figure(figsize=(5, 3))
    sns.set(font_scale=1)
    canvas = sns.heatmap(confusionMatrix, annot=True,
xticklabels=["cloudy", "rain", "sunrise", "shine"],
                         yticklabels=["cloudy", "rain", "sunrise",
"shine"],
                         cmap="RdGy", linewidths=1, linecolor="black",
fmt=".0f")
    pyp.yticks(rotation=0)
    pyp.xlabel("Tahmin Sonuçları:")
    pyp.ylabel("Gerçek Sonuçlar:")
    canvas.xaxis.set_ticks_position("top")
    pyp.title("Karışıklık Matrisi")
    pyp.savefig("karmasiklikMatrisCML.png")
    pyp.cla()
    pyp.clf()

    falsePositive = dict()
    truePositive = dict()
    aucRoc = dict()
```

```
    for i in range(4):
        falsePositive[i], truePositive[i], c = roc_curve(self.y_test[:,
i], y_Tahmin[:, i])
        aucRoc[i] = auc(falsePositive[i], truePositive[i])

    falsePositive["min"], truePositive["min"], c =
roc_curve(self.y_test.ravel(), y_Tahmin.ravel())
    aucRoc["min"] = auc(falsePositive["min"], truePositive["min"])
    fp_all = np.unique(np.concatenate([falsePositive[i] for i in
range(4)]))
    tp_ort = np.zeros_like(fp_all)
    for i in range(4):
        tp_ort += np.interp(fp_all, falsePositive[i], truePositive[i])
    tp_ort = tp_ort / 4
    falsePositive["max"] = fp_all
    truePositive["max"] = tp_ort
    aucRoc["max"] = auc(falsePositive["max"], truePositive["max"])

    pyp.figure()
    pyp.plot(falsePositive["min"], truePositive["min"],
             label="Minimum Ortalama ROC Eğrisi
(Alan={0:0.2f})".format(aucRoc["min"]), color="navy",
             linestyle=":", linewidth=4)
    pyp.plot(falsePositive["max"], truePositive["max"],
             label="Maksimum Ortalama ROC Eğrisi
(Alan={0:0.2f})".format(aucRoc["max"]), color="deeppink",
             linestyle=":", linewidth=4)
    colors = cycle(["cornflowerblue", "aqua", "darkorange"])

    for i, color in zip(range(4), colors):
        pyp.plot(falsePositive[i], truePositive[i], color=color, lw=2,
                 label='{0}. sınıfın ROC eğrisi (Alan = {1:0.2f})'
                       ''.format(i, aucRoc[i]))
    pyp.plot([0, 1], [0, 1], 'k--', lw=2)
    pyp.xlim([0.0, 1.0])
    pyp.ylim([0.0, 1.05])
    pyp.xlabel('False Positive Rate')
    pyp.ylabel('True Positive Rate')
    pyp.title('Some extension of Receiver operating characteristic to
multi-class')
    pyp.legend(loc="lower right")
    pyp.savefig("ROCegrisiCML.png")
    pyp.cla()
    pyp.clf()
    self.lblROC.setPixmap(QtGui.QPixmap("./ROCegrisiCML.png"))

self.lblKarmasiklikMatrisi.setPixmap(QtGui.QPixmap("./karmasiklikMatrisCM
L.png"))
    self.lblAccuracy.setPixmap(QtGui.QPixmap("./bos.png"))
    self.lblLoss.setPixmap(QtGui.QPixmap("./bos.png"))
    print("Klasik Makine Öğrenmesi İşlemi Bitti.")
```

#burada Random Forest Sınıflandırma Algoritması ile çalışıp Karışıklık Matrisi ve ROC Eğrisi oluşturuyoruz ve bunları ".png" formatında proje path'ine kaydediyorum. Ve Raporlamada gösteriyorum.

```
#Derin Öğrenme İçin Tahmin Etme Fonksiyonu
def prediction(self,image):
    image = cv.imread(image)
    image = cv.cvtColor(image,cv.COLOR_BGR2RGB)
    image = Image.fromarray(image,"RGB")
    image = image.resize(self.im_Size)
    image = np.array(image)
    image = image / 255
    veri = []
    veri.append(image)
    veri = np.array(veri)
    point = self.GlobalModel.predict(veri,verbose=1)
    etiket = np.argmax(point)
    accuracy = np.max(point)
    self.lblAcc.setText("%"+str(accuracy*100))
    if etiket == 0:
        self.lblTahminEtiket.setText("Bulutlu")
    elif etiket == 1:
        self.lblTahminEtiket.setText("Yağmurlu")
    elif etiket == 2:
        self.lblTahminEtiket.setText("Gündoğumu")
    else:
        self.lblTahminEtiket.setText("Güneşli")
    self.lblGercekEtiket.setText("Transfer Learning")
```

#Burada okuduğumuz fotoğraf üzerinde Derin Öğrenme Modeline uygun değişiklikler yapıp modele bu görseli tahmin ettiriyoruz dönen etiket değerine göre ise görselin hangi hava durumunda olduğunu yazdırıp, accuracy değerini de yazdırıyoruz.

```
#Klasik Makine Öğrenmesi Algoritması ile Tahmin Etme
def ClasicPrediction(self,image):
    image = cv.imread(image)
    image = cv.cvtColor(image,cv.COLOR_BGR2RGB)
    image = Image.fromarray(image,"RGB")
    image = image.resize(self.im_Size)
    image = np.array(image)
    image = image / 255
    input = np.expand_dims(image,axis=0)
    features = self.mainExtractor.predict(input)
    features = features.reshape(features.shape[0],-1)
    tahmin = self.GlobalModel.predict(features)[0]
    etiket = np.argmax(tahmin)
    if etiket == 0:
        self.lblTahminEtiket.setText("Bulutlu")
    elif etiket == 1:
        self.lblTahminEtiket.setText("Yağmurlu")
    elif etiket == 2:
        self.lblTahminEtiket.setText("Gündoğumu")
    else:
        self.lblTahminEtiket.setText("Güneşli")
    self.lblAcc.setText("")
    self.lblGercekEtiket.setText("Klasik Makine Öğrenmesi")
```

#Burada okuduğumuz fotoğraf üzerinde Random Forest Modeline uygun değişiklikler yapıp modele bu görseli tahmin ettiriyoruz dönen etiket değerine göre ise görselin hangi hava durumunda olduğunu yazdırıyoruz.

```
#Veri Okuma İşlemleri
def dataRead(self,folder):
    data = glob.glob(folder + "*")
    data = [x.replace('\\', '/') for x in data]
    folder = folder.replace("\\","/")
    print("Veri Okuma İşlemi Başlatıldı.")
    print("Veri Sayısı:"+str(len(data)))
    for i, x in enumerate(data):
        try:
            image = cv.imread(x)
            imArray = Image.fromarray(image, "RGB")
            image =np.array(imArray.resize(self.im_Size))
            if data[i].startswith(folder + "cloudy"):
                self.labeled.append([image, 0, os.path.basename(x)])
            elif data[i].startswith(folder + "rain"):
                self.labeled.append([image, 1, os.path.basename(x)])
            elif data[i].startswith(folder + "sunrise"):
                self.labeled.append([image, 2, os.path.basename(x)])
            elif data[i].startswith(folder + "shine"):
                self.labeled.append([image, 3, os.path.basename(x)])
        except Exception as e:
            print("Bu veri okunamıyor!!")
    data_count = pd.DataFrame(self.labeled, columns=["image", "etiket",
"filename"])
    pyp.figure(figsize=(20, 5))
    pyp.subplot(1, 1, 1)
    sns.countplot(data_count["etiket"])
    pyp.title('Veriler')
    pyp.savefig("./EtiketGrafigi/Etiketler.png")

self.labelDataRead.setPixmap(QtGui.QPixmap("./EtiketGrafigi/Etiketler.png
"))
    print("Veri Okuma İşlemi Bitti.")
```

#Burada verinin okunması ve etiketlenme işlemleri yapılmaktadır. Veri setindeki sınıf sayılarını da grafik oluşturup ekrana yazdırıyoruz.

```
#Histogram Eşitleme İşlemleri
def histogramE(self):
    print("Histogram Eşitleme işlemi başlatıldı-->")
    for im in self.labeled:
        try:
            image = im[0]
            img_yuv = cv.cvtColor(image, cv.COLOR_RGB2YUV)
            img_yuv[:, :, 0] = cv.equalizeHist(img_yuv[:, :, 0])
            he_img = cv.cvtColor(img_yuv, cv.COLOR_YUV2RGB)
            self.he.append([he_img, im[1], None])
        except Exception as e:
            print("Histogram Eşitlemede bu veri okunamıyor!!")
    print("Histogram Eşitleme işlemi bitti<--")
#Burada HE işlemlerini yapıp dönen resimleri ayrı bir dizide tutuyoruz
(daha sonra birleştireceğiz.)
```

```
# CLAHistogram Eşitleme
def CLAhistogramE(self):
    print("Contrast Limited Adaptive Histogram Eşitleme(CLAHE) işlemi
başlatıldı-->")
    for im in self.labeled:
        try:
            image = im[0]
            img_yuv = cv.cvtColor(image, cv.COLOR_RGB2YUV)
            clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
            img_yuv[:, :, 0] = clahe.apply(img_yuv[:, :, 0])
            CLAhe_img = cv.cvtColor(img_yuv, cv.COLOR_YUV2RGB)
            self.CLAhe.append([CLAhe_img, im[1], None])
        except Exception as e:
            print("CLAHE'de bu veri okunamıyor!!")
    print("Contrast Limited Adaptive Histogram Eşitleme(CLAHE) işlemi
bitti<--")
```

#Burada CLAHE işlemlerini yapıp dönen resimleri ayrı bir dizide tutuyoruz (daha sonra birleştireceğiz.)

```
#Veri Birleştirme İşlemleri
def birlestirme(self):
    self.labeled = np.array(self.labeled)
    self.he = np.array(self.he)
    self.CLAhe = np.array(self.CLAhe)
    self.array = np.array(np.concatenate((self.he, self.CLAhe)))
    birlestirilmis = np.concatenate((self.labeled, self.array))
    indisler = np.arange(len(birlestirilmis))
    np.random.shuffle(indisler)
    birlestirilmis = birlestirilmis[indisler]
    self.labeled = birlestirilmis
    for i, x in enumerate(self.labeled):
        self.veriler.append(np.array(self.labeled[i][0]))
        self.etiketler.append(self.labeled[i][1])
        self.uzanti.append(self.labeled[i][2])
    self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(self.veriler, self.etiketler, test_size=0.2)
    self.X_train = np.array(self.X_train)
    self.X_test = np.array(self.X_test)

    self.y_train = to_categorical(self.y_train)
    self.y_test = to_categorical(self.y_test)
    self.X_train = self.X_train.astype("float32") / 255
    self.X_test = self.X_test.astype("float32") / 255

    self.base_model = keras.applications.Xception(
        weights='imagenet',  # Load weights pre-trained on ImageNet.
        input_shape=(150, 150, 3),
        include_top=False)
    self.base_model.trainable = False
```

#Burada HE'den ve CLAHE'den dönen imageleri tuttuğumuz dizileri ve orijinal görüntülerimi tek bir dizide birleştirme işlemini yapıyoruz. Ve daha sonrasında Hold-out işlemlerini ve Train_test_split işlemlerini gerçekleştiriyoruz.

```
#Veri Artırma
def veriArtirma(self):
    augmentated_path = "augDatas"
    if os.path.exists(augmentated_path):
        shutil.rmtree(augmentated_path)
    os.mkdir(augmentated_path)
    for img in self.labeled:
        try:
            image = img[0]
            datagen = ImageDataGenerator(
                rotation_range= 60, width_shift_range= 0.2,
height_shift_range= 0.2, shear_range=0.4, zoom_range=0.5,horizontal_flip
= True,
                vertical_flip=True,fill_mode= "nearest"
            )
            a = img_to_array(image)
            a = a.reshape((1,)+a.shape)
            i = 0
            img_format, img_name = img[2][::-1].split(".", 1)
            img_name = img_name[::-1]
            img_format = img_format[::-1]
            for batch in datagen.flow(a, batch_size=1,
save_to_dir=(augmentated_path),
                                      save_prefix=img_name,
save_format=img_format):
                i += 1
                if i > 1:
                    break
        except Exception as e:
            print(e)
    folder =
os.path.join(os.path.dirname(os.path.abspath(__file__)),augmentated_path+
"/")
    self.dataRead(folder)
```

Burada orijinal verilerimiz üzerinde veri artırma işlemleri uygularız ve bunu orijinal resimlerle birlikte bu resimleri proje path'ine kaydederiz.