KASTAMONU ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
GÖRÜNTÜ İŞLEME DERSİ ÖDEV RAPORU

ÖDEV

GÖRÜNTÜ İŞLEME VE TRANSFER LEARNİNG TEKNİKLERİ İLE
HAVA DURUMU SINIFLANDIRMA

GRUP : 2
İŞ AKIŞI : 2
DATASET : B
RAPORLAMA REVİZESİ
V2

DERSİN SORUMLUSU

Doç. Dr. Kemal AKYOL

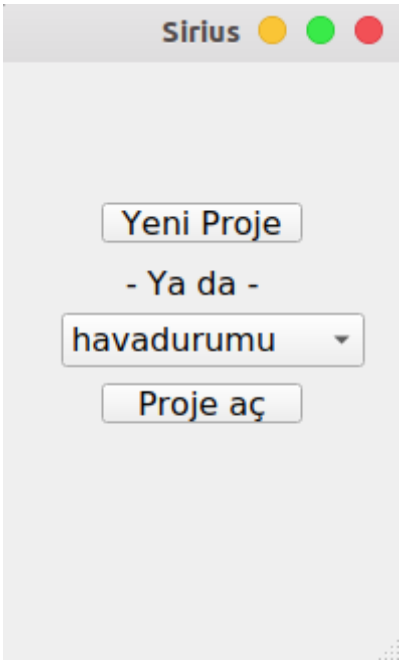
RAPORU YAZAN ÖĞRENCİ

194410065

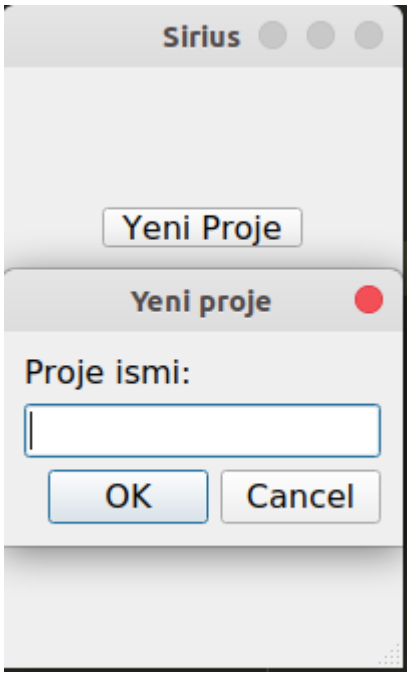
Emre EREN

EKRAN GÖRÜNTÜLERİ

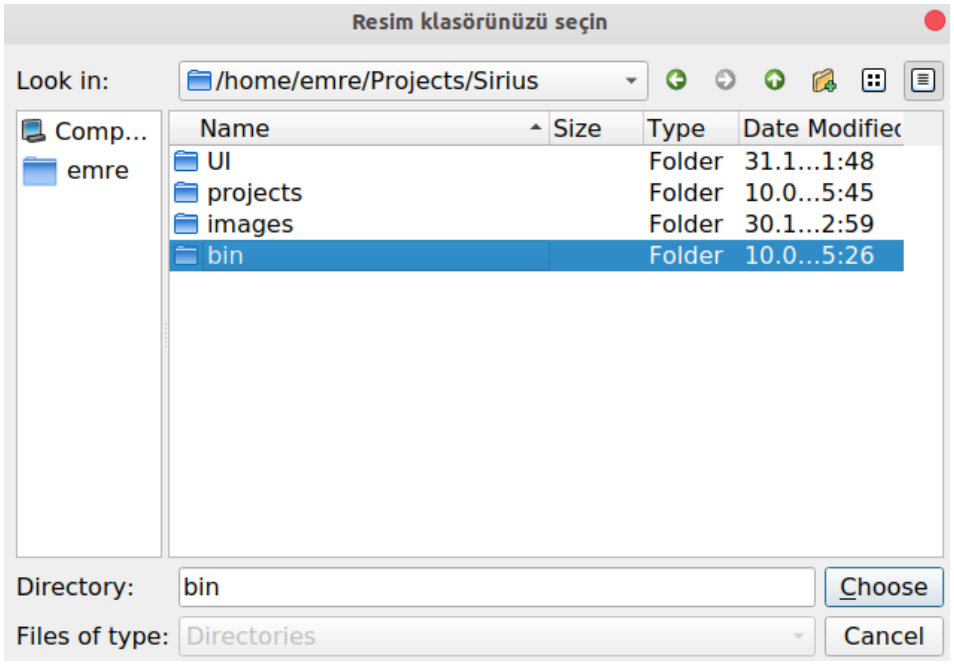
Ana Menü

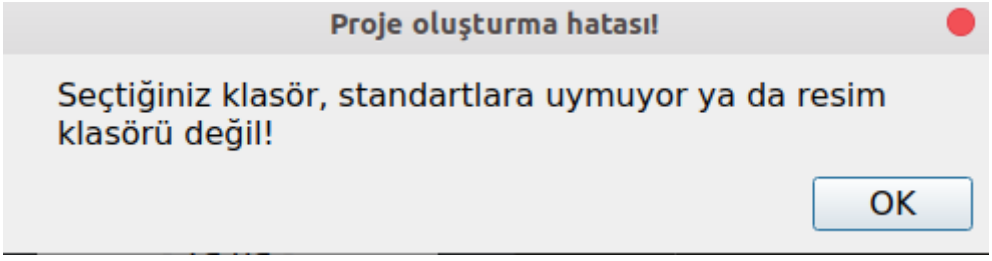


Sirius ana menüsünde yeni bir proje oluşturabilir veya daha öncesinde üzerinizde çalıştığınız bir projeden devam edebilrisiniz. Eğer daha önce Sirius ile yaptığınız bir ön işleme projeniz var ise, proje açma ekranı, bir daha ön işleme yapmadan bu projeden devam edebilme olanağı sağlar.

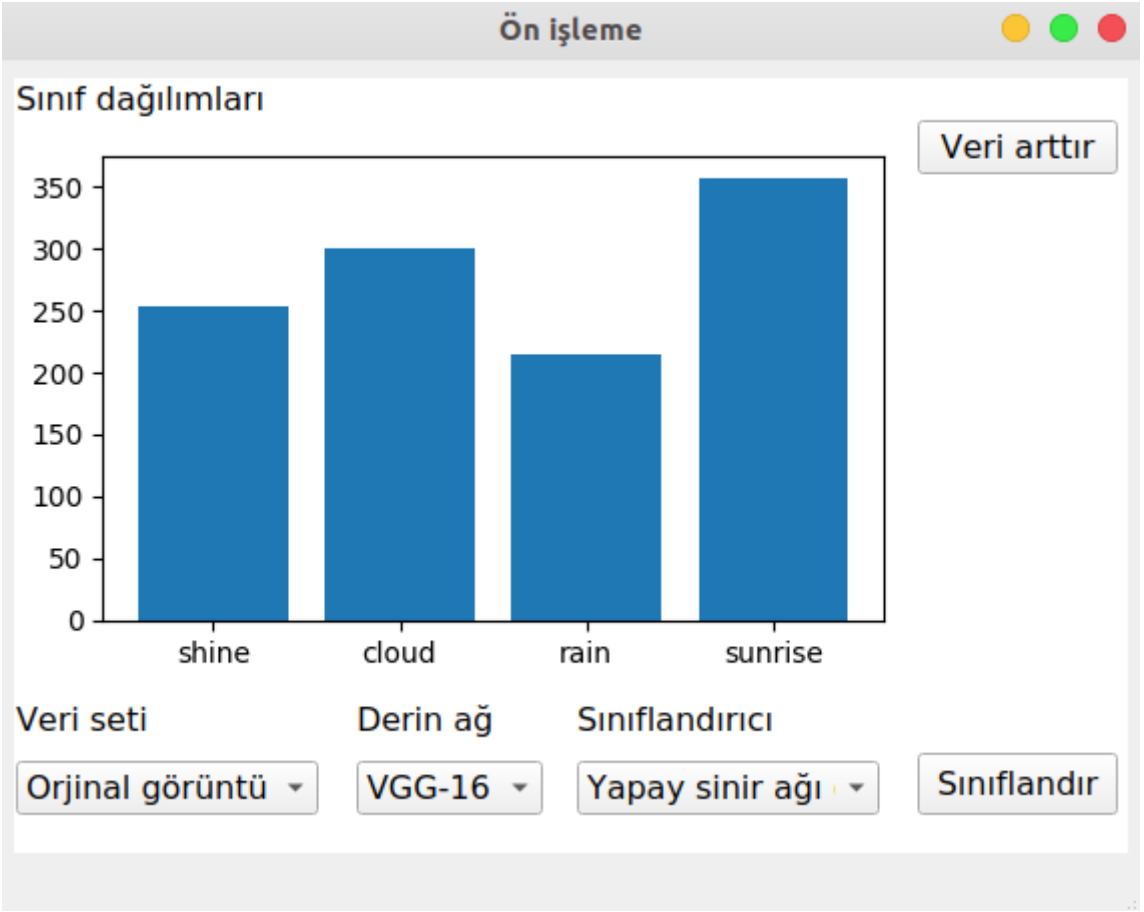


Yeni proje açtığınızda sizden önce proje isminizi, daha sonra çalışacağınız veri setini seçmenizi isteyecektir. Program veri seti konusunda hassas davranır. Seçeceğiniz çalışma klasörü içinde resimler etiketlerine göre klasörlerde olmalı ve bu klasörlerde sadece resim dosyaları olmak zorunda veya bir klasör içinde sadece resim dosyaları olmalı. Bu durumda resimleri sınıflarına ayırarak okuyacaktır.



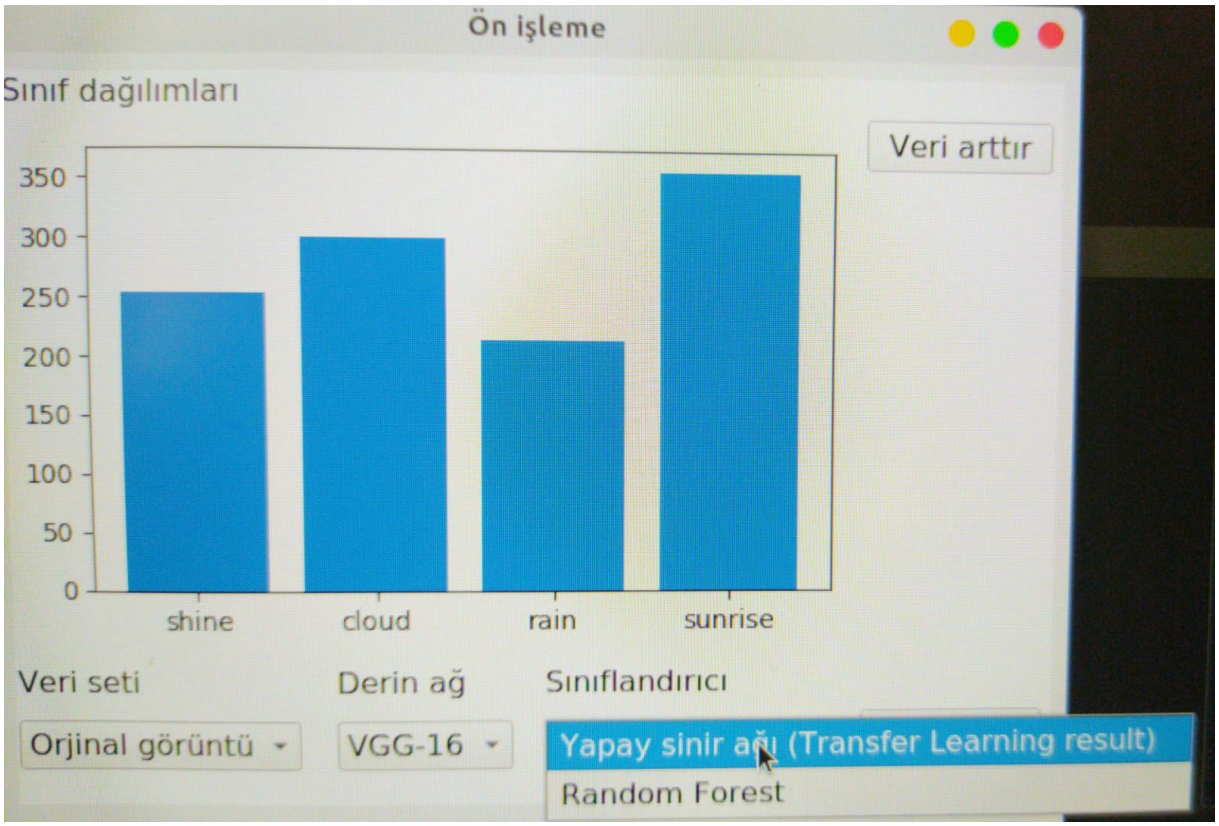


Ön İşleme Ekranı



Ön işleme ekranında sınıf dağılım grafiğinizi görebilir, Veri arttırma tekniğini uygulayabilir ve eğer veri arttırma yaptıysanız orijinal görüntü ile mi yoksa hem orijinal hem de veri arttırma yaptığınız resimler ile çalışmak istediğinize **Veri seti** bölümünden karar verebilirsiniz.

Eğer isterseniz VGG16 veya RESNET50 mimarisi arasında seçim yapabilirsiniz.



Sınıflandırıcı olarak VGG16 modelinden gelen featurelar ile Random Forest veya klasik yapay sinir ağı sınıflandırıcısını seçebilirsiniz.

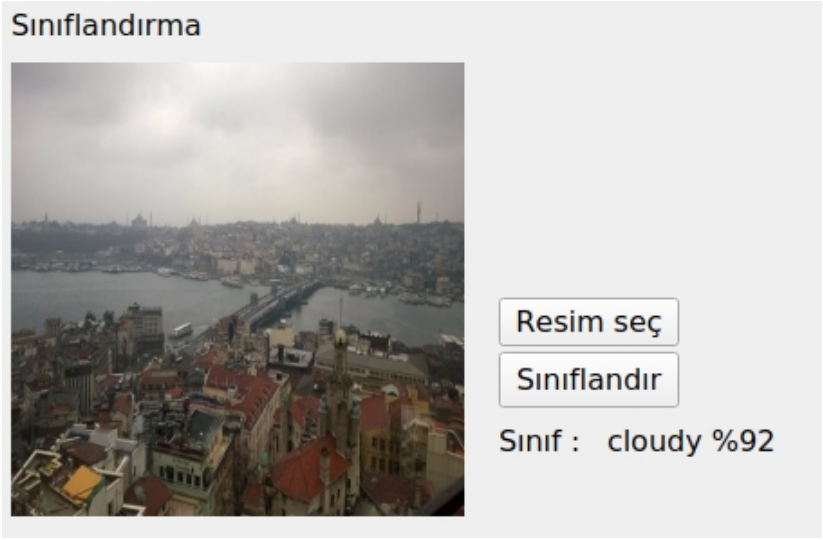
Sınıflandırma Raporu (Transfer Learning)

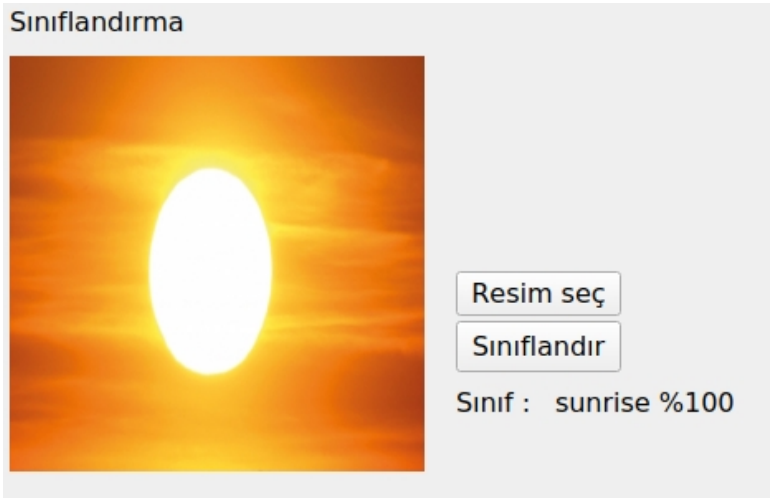
Sınıflandırma Raporu

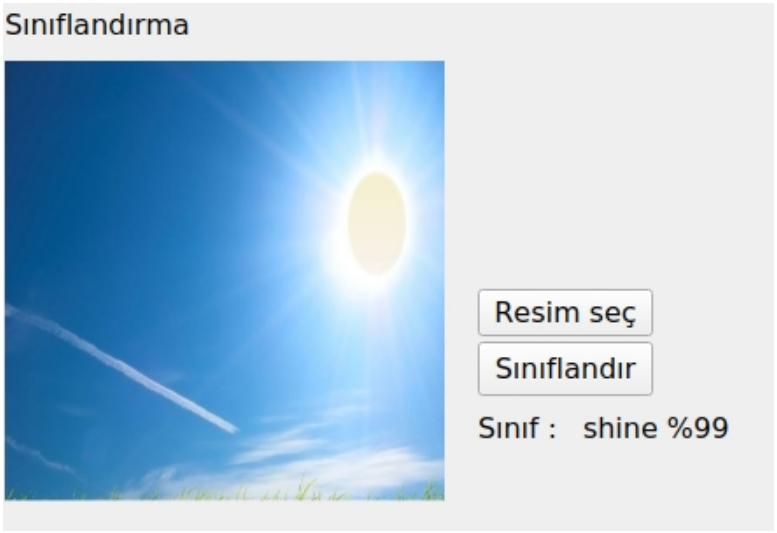
Sınıflandırma sonuçlarını bu ekranda görebiliriz. Sınıflandırma raporu bölümünden precision, recall, f1-score gibi metrikleri görebiliriz.

İstatistikler bölümünden accuracy-loss grafiklerini görebiliriz.

Karışıklık matrisinin ısı haritasının grafiklerini ve her sınıf için ROC eğrilerini görebiliriz.

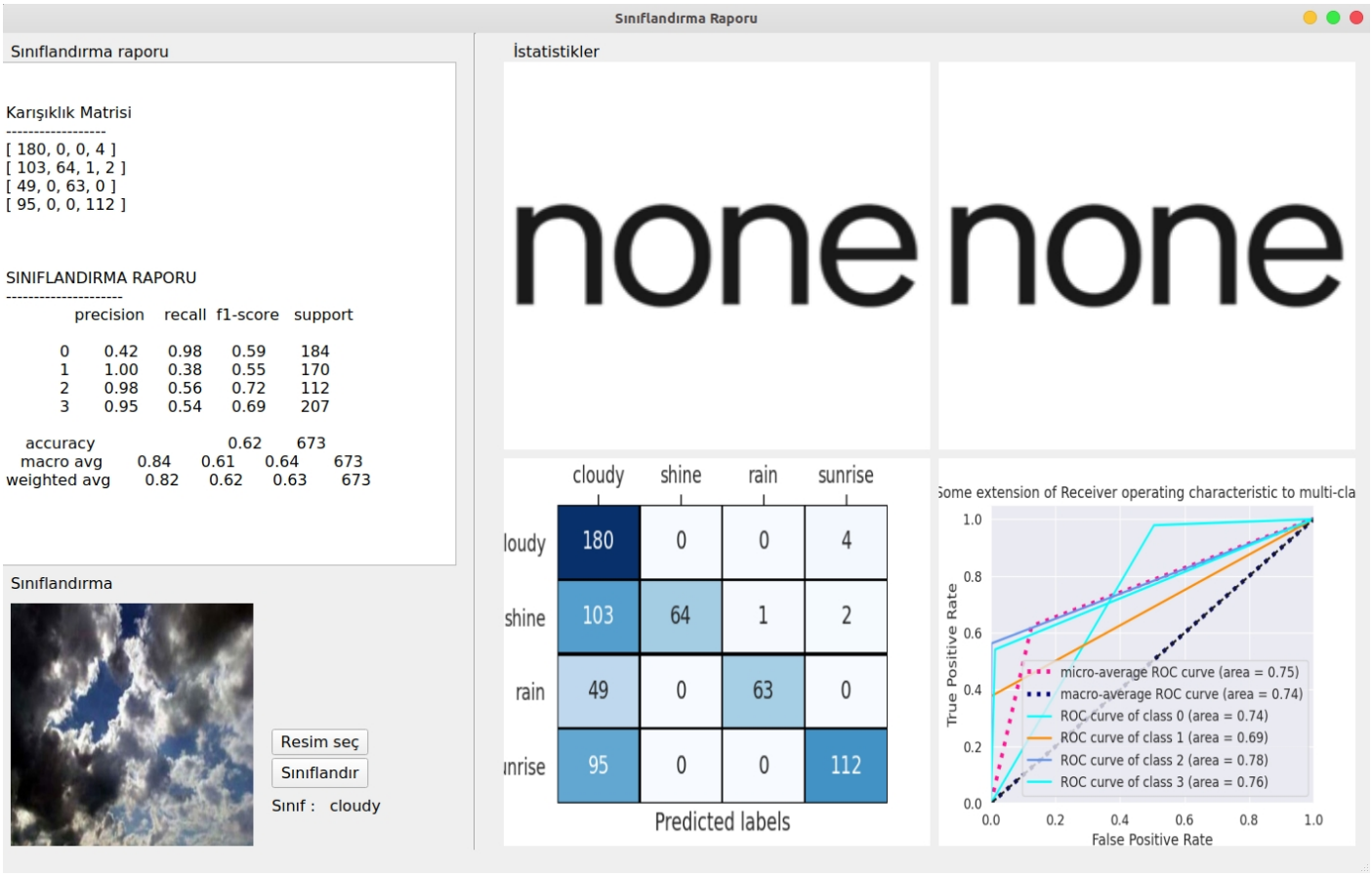






Yine sınıflandırma raporu penceresinde sol alt bölümünden ekstra resim seçip sınıflandırma sonuçlarını görebiliriz.

SINIFLANDIRMA RAPORU (KLASİK M.Ö)



Makine öğrenmesi kısmında accuracy-loss grafikleri olmadığı için sadece karışıklık matrisini, sınıflandırma raporunu ve ROC eğrisini görüyoruz.

Deep Learning Model Katmanları

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 150, 150, 3)]	0
vgg16 (Functional) *VEYA* resnet50 (Functional)	(None, 4, 4, 512)	14714688
max_pooling2d (MaxPooling2D)	(None, 2, 2, 512)	0
conv2d (Conv2D)	(None, 2, 2, 64)	131136
max_pooling2d_1 (MaxPooling2	(None, 1, 1, 64)	0
dropout (Dropout)	(None, 1, 1, 64)	0
flatten (Flatten)	(None, 64)	0
dense (Dense)	(None, 500)	32500
dropout_1 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 4)	2004

KOD PARÇACIKLARI

Veri Ön İşleme

Veri Ön işleme sınıfı:

```
class ImagePreProcessor:
    def __init__(self,workdir,imagedata):
        self.workdir = workdir
        self.imagedata = imagedata
```

Veri ön işleme için ImagePreProcessor sınıfı. Parametre olarak çalışma dizini (Proje klasörü oluşturulduğu için) ve görüntü kümesi ismi alıyoruz. Görüntü kümesi orijinal veriler için mi yoksa özel bir görüntü kümesi (Arttırılmış görüntüler vb.) üzerinde mi işlem yapılacağını belirler.

Veri arttırma:

```
    def image_augmentation(self,augmentation_count = 10):
    path = self.workdir
    augmentated_path = os.path.join(path, "augmentated_images")
    if not os.path.exists(augmentated_path):
            os.mkdir(augmentated_path)
        workdir = os.path.join(path, "images")

        labels = os.listdir(workdir)
        for label in labels:
            images_path = os.listdir(os.path.join(workdir, label))
            os.mkdir(os.path.join(augmentated_path, label))
            for image_path in images_path:
                try:
                    img_path = os.path.join(workdir, label, image_path)
                    img = cv2.imread(img_path)
                    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    datagen = ImageDataGenerator(
                        rotation_range=40,
                        width_shift_range=0.2,
                        height_shift_range=0.2,
                        shear_range=0.2,
                        zoom_range=0.2,
                        horizontal_flip=True,
                        fill_mode="nearest"
                    )
                    x = img_to_array(img)
                    x = x.reshape((1,) + x.shape)
                    i = 0
                    img_format, img_name = image_path[::-1].split(".", 1)
                    img_name = img_name[::-1]
                    img_format = img_format[::-1]
                    for batch in datagen.flow(x, batch_size=1,
save_to_dir=os.path.join(augmentated_path, label),
                                              save_prefix=img_name, save_format=img_format):
                        i += 1
                        if i > augmentation_count - 1:
                            break
                except Exception as e:
                    print(e)
                    print("loss : ", image_path)
        return True
    else:
        return False
```

Veri arttırma metodu, parametre olarak çoğaltma adedini alır. Keras kütüphanesinin ImageDataGenerator modulünü veri arttırmak için kullanır. augmentated_data isimli bir klasör oluşturur ve her bir resme arttırım uygulayarak kaydeder.

Histogram Eşitleme:

```
def extract_he_images(self):
    if not os.path.exists(os.path.join(self.workdir,"preprocessed")):
        os.mkdir(os.path.join(self.workdir,"preprocessed"))
    he_images_path =
os.path.join(self.workdir,"preprocessed",self.imagedata,"he_images")
    ajax = os.path.join(self.workdir,"preprocessed",self.imagedata)
    if not os.path.exists(ajax):
        os.mkdir(ajax)
    if not os.path.exists(he_images_path):
        os.mkdir(he_images_path)

        labels = os.listdir(os.path.join(self.workdir,self.imagedata))
        for label in labels:
            os.mkdir(os.path.join(he_images_path,label))
            images = os.listdir(os.path.join(self.workdir,self.imagedata,label))
            for image in images:
                try:
                    img_path =
os.path.join(self.workdir,self.imagedata,label,image)
                    img = cv2.imread(img_path)
                    img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
                    img_yuv[:, :, 0] = cv2.equalizeHist(img_yuv[:, :, 0])
                    he_img = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)
                    cv2.imwrite(os.path.join(he_images_path,label,"he_" +
image),he_img)
                except Exception as ex:
                    print("loss : ",image)
        return True
    else:
        return False
```

Histogram eşitleme metodu, her bir görüntü için önce yuv uzayına dönüştürme ardından histogram eşitleme uygulama (0. kanala) daha sonra tekrar bgr moda çevirme işlemi uygular. Bunun sebebi, çok kanallı görüntülerde histogram eşitlemenin cv2 ile yapılamamasıdır. Her bir görüntü üzerine histogram eşitleme uygulandıktan sonra pre_processed klasörü altına çalışma klasörü altına he_images klasörü içerisine kaydedilir.

CLA Histogram Eşitleme:

```
def extract_clahe_images(self):
    if not os.path.exists(os.path.join(self.workdir,"preprocessed")):
        os.mkdir(os.path.join(self.workdir,"preprocessed"))
    he_images_path = os.path.join(self.workdir, "preprocessed",
self.imagedata, "clahe_images")
    ajax = os.path.join(self.workdir,"preprocessed",self.imagedata)
    if not os.path.exists(ajax):
        os.mkdir(ajax)
    if not os.path.exists(he_images_path):
        os.mkdir(he_images_path)

        labels = os.listdir(os.path.join(self.workdir, self.imagedata))
        for label in labels:
            os.mkdir(os.path.join(he_images_path, label))
            images = os.listdir(os.path.join(self.workdir, self.imagedata,
label))
            for image in images:
                try:
                    img_path = os.path.join(self.workdir, self.imagedata,
label, image)
                    img = cv2.imread(img_path)
                    img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)

                    clahe = cv2.createCLAHE(clipLimit=2.0,
tileGridSize=(8,8))
                    img_yuv[:, :, 0] = clahe.apply(img_yuv[:, :, 0])


                    he_img = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)
                    cv2.imwrite(os.path.join(he_images_path, label,"clahe_"
+ image), he_img)
                except Exception as ex:
                    print("loss : ", image)
        return True
    else:
        return False
```

Histogram eşitleme metodu ile aynı mantıkta çalışır. Sadece CLA işlemi uygulanır.

Sınıflandırma sınıfı:

```
class Classification:
    def __init__(self,model_dict):
        self.model_dict = model_dict

        self.global_image_data = []
        self.global_label_data = []
        self.X_train = []
        self.X_test = []
        self.y_train = []
        self.y_test = []

         self.global_epoch_lim = 2

        self.global_image_size = (150,150)
        self.global_labels =
os.listdir(os.path.join(self.model_dict["workdir"],"images"))
        self.num_classes = len(self.global_labels)
        self.model = None
```

parametre olarak model_dict isimli bir sözlük alır. Bu sözlük içerisinde proje dizini, sınıflandırma için makine öğrenmesi algoritması, çalışılacak veri klasörleri (arttırılmış veriler, orijinal veriler vb.) gibi parametreler taşınır.

Globalde tanımlanacak sabit veriler de burada tanımlanır.

```
def get_image_data(self, data_path = "images"):
    data = []
    labels = []
    original_path = os.path.join(self.model_dict["workdir"],data_path)
    original_labels = os.listdir(original_path)
    for label_no,original_label in enumerate(original_labels):
        images_path = os.path.join(original_path,original_label)
        image_names = os.listdir(images_path)
        for image_name in image_names:
            try:
                image_path = os.path.join(images_path,image_name)
                image = cv2.imread(image_path)
                image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
                image_from_array = Image.fromarray(image,'RGB')
                resized_image =
image_from_array.resize(self.global_image_size)
                data.append(np.array(resized_image))
                labels.append(np.array(label_no))
            except Exception as ex:
                print(ex)
                print("Loss : ", image_name)
    return np.array(data),np.array(labels)
```

Verilen klasör üzerinden veriyi okur. RGB renk uzayına dönüştürür, yeniden boyutlandırır ve numpy dizisi olarak hem okuduğu resimleri hem de bu resimlerin etiketlerini geriye döner. Eğer resim okuma sırasında hata meydana gelirse bunları kayıp olarak işaretler.

```
def shuffle_data(self):
    s = np.arange(self.global_image_data.shape[0])
    np.random.shuffle(s)
    self.global_image_data = self.global_image_data[s]
    self.global_label_data = self.global_label_data[s]
```

Resimleri eğitim ve test verisi olarak ayırmadan önce 0’dan veri sayısına kadar sayıların olduğu bir indisler dizisi yaratır ve bu diziyi rastsal bir şekilde karıştırır. Bu sayede eğitim ve test sırasında rassallık sağlanır

```
def image_data_split(self):
    data_length = len(self.global_image_data)
    self.X_train,self.X_test = self.global_image_data[(int)
(0.2*data_length):],self.global_image_data[:(int)(0.2*data_length)]
    self.X_train = self.X_train.astype("float32")/255
    self.X_test = self.X_test.astype("float32")/255

    self.y_train,self.y_test = self.global_label_data[(int)
(0.2*data_length):],self.global_label_data[:(int)(0.2*data_length)]
    self.y_train = to_categorical(self.y_train,self.num_classes)
    self.y_test = to_categorical(self.y_test,self.num_classes)
```

Resimleri %20 test %80 eğitim olarak ayırdığımız kısımdır.

```
def create_model(self):
    base_model = None
    if self.model_dict["data_informations"]["DeepLearningAlgorithm"]
== DataInformations.DL_VGG16.value:
        print("vgg16")
        base_model = vgg16.VGG16(weights = "imagenet", include_top
= False, input_shape =
(self.global_image_size[0],self.global_image_size[1],3))
    elif self.model_dict["data_informations"]
["DeepLearningAlgorithm"] ==
DataInformations.DL_RESNET50.value:
        print("resnet50")
        base_model = resnet50.ResNet50(weights = "imagenet",
include_top = False, input_shape =
(self.global_image_size[0],self.global_image_size[1],3))
    base_model.trainable = False
    inputs = keras.Input(shape=(self.global_image_size[0],
self.global_image_size[1], 3))
    x = base_model(inputs, training=False)
    x = keras.layers.MaxPooling2D(pool_size=2)(x)
    x = keras.layers.Conv2D(filters=64, kernel_size=2,
padding="same", activation="relu")(x)
    x = keras.layers.MaxPooling2D(pool_size=2)(x)
    x = keras.layers.Dropout(0.2)(x)
    x = keras.layers.Flatten()(x)
    x = keras.layers.Dense(500, activation="relu")(x)
    x = keras.layers.Dropout(0.2)(x)
    outputs = keras.layers.Dense(self.num_classes,
activation="softmax")(x)
    model = keras.Model(inputs, outputs)
    self.model = model
    print(model.summary())
```

Seçilen (VGG16/RESNET50) derin öğrenme modeli, Transfer learning için ayarlanır. Üzerine performansı arttırmak için bir katman daha eklenir ve unutma işlemine tabi tutulup düzleştirme uygulanır. En sonunda yapay sinir ağına resim featureları aktarılır ve karar verilir.

```
def model_fit(self):
    self.model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
    callback = keras.callbacks.EarlyStopping(monitor="loss",
patience=3)
    history = self.model.fit(self.X_train, self.y_train, batch_size=32,
epochs=self.global_epoch_lim, verbose=1, callbacks=[callback],
                             validation_data=(self.X_test, self.y_test))
    self.history = history
```

Transfer learning sınıflandırma için modelin hazırlandığı kısımdır. Burada Early stopping için ayarlamalar yapılır ve model eğitilir.

Optimizer olarak adam kullanılır.

```
def create_rf_model(self):
    base_model = None
    if self.model_dict["data_informations"]["DeepLearningAlgorithm"]
== DataInformations.DL_VGG16.value:
        base_model = vgg16.VGG16(weights="imagenet",
include_top=False,
                                 input_shape=(self.global_image_size[0],
self.global_image_size[1], 3))
    elif self.model_dict["data_informations"]
["DeepLearningAlgorithm"] ==
DataInformations.DL_RESNET50.value:
        base_model = resnet50.ResNet50(weights="imagenet",
include_top=False,
                                       input_shape=(self.global_image_size[0],
self.global_image_size[1], 3))
    for layer in base_model.layers:
        layer.trainable = False
    X_train_feature_extractor = base_model.predict(self.X_train)
    self.X_train_features =
X_train_feature_extractor.reshape(X_train_feature_extractor.shape[0
],-1)

    X_test_feature_extractor = base_model.predict(self.X_test)
    self.X_test_features =
X_test_feature_extractor.reshape(X_test_feature_extractor.shape[0],
-1)

    self.TL_extractor = base_model
    self.model =
RandomForestClassifier(n_estimators=50,random_state=42)
```

Klasik makine öğrenmesi modeli için VGG16/RESNET50 Transfer learning modeli dahil edilir. Featurelar çıkarıldıktan sonra model Random Forest algoritması ayarlanır.

```
def rf_model_fit(self):
    self.model.fit(self.X_train_features,self.y_train)
```

Random forest modelinin eğitildiği kısımdır.

```
def get_confusion_matrix(self): #Global function
    y_pred = self.model.predict(self.X_test,batch_size = 32)
    y_pred = np.argmax(y_pred,axis=1)
    y_test_cm = np.argmax(self.y_test,axis=1)
    conf = confusion_matrix(y_test_cm,y_pred)
    clrp =
classification_report(y_test_cm,y_pred,target_names=self.global_labels)

    #roc curve
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    y_pred_e = to_categorical(y_pred)

    for i in range(len(self.y_test[0])):
        fpr[i], tpr[i], _ = roc_curve(self.y_test[:, i], y_pred_e[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    fpr["micro"], tpr["micro"], _ = roc_curve(self.y_test.ravel(),
y_pred_e.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(len(self.y_test[0]))]))

    mean_tpr = np.zeros_like(all_fpr)
    for i in range(len(self.y_test[0])):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

    mean_tpr /= len(self.y_test[0])
    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    return {
        "confusion" : conf,
        "classification_report" : clrp,
        "fpr" : fpr,
        "tpr" : tpr,
        "roc_auc" : roc_auc
    }
```

Bu kısımda sınıflandırma sonuçlarının görüntülenmesi için gerekli metriklerin çıkarıldığı kısımdır. Karışıklık matrisi ve sınıflandırma raporu burada üretilir. Ardından Roc eğrisinin çoklu sınıflandırmalarda kullanılabilmesi için her bir sınıf adına metrikler hesaplanır.

```
def classification_report(self):
    #Confusion matrix
    conf_matrix_str = "\n\nKarışıklık Matrisi\n-----------------\n"
    cm = self.classification_rep["confusion"]
    for i in range(cm.shape[0]):
        conf_matrix_str += "[ "
        for j in range(cm.shape[1]):
            if j != cm.shape[1] - 1:
                conf_matrix_str += str(cm[i, j]) + ", "
            else:
                conf_matrix_str += str(cm[i, j]) + " "
        conf_matrix_str += "]\n"

    #Classification report
    cr = "\n\nSINIFLANDIRMA RAPORU\n--------------------\n"
    cr += self.classification_rep["classification_report"]

    #Epoch steps
    epoch_steps_str = ""
    if self.model_dict["data_informations"]["ClassifierAlgorithm"] ==
DataInformations.CLASSIFIER_ANN.value:
        epoch_steps_str = "\n\nEpoch başına Accuracy - Loss oranları\
n-----------------------------------\n"
        count = len(self.history.history["accuracy"])
        for i in range(count):
            epoch_steps_str += "Adım " + str(i + 1) + "/" + str(count) +
" -> Accuracy : " + str(
                self.history.history["accuracy"][i]) + ", Loss : " +
str(self.history.history["loss"][i]) + "\n"
    return {
        "confusion_matrix_str" : conf_matrix_str,
        "classification_report_str" : cr,
        "epoch_steps_str" : epoch_steps_str
    }
```

Sınıflandırma raporunun textBox üzerine aktarılmak üzere String çıktılarına dönüştürüldüğü kısım burasıdır. Burası her iki sınıflandırma metodu içinde çalışır. Eğer model yapay sinir ağları ile sınıflandırma metodu ise epoch başına sonuçlarında çıkarıldığı kısım burasıdır.

```
def get_rf_confusion_matrix(self):
    y_pred = self.model.predict(self.X_test_features)
    y_train_decoded = np.argmax(self.y_train, axis=1)
    y_test_decoded = np.argmax(self.y_test, axis=1)
    y_pred = np.argmax(y_pred, axis=1)
    conf = confusion_matrix(y_test_decoded,y_pred)
    clrp = classification_report(y_test_decoded,y_pred)
    # roc curve
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    y_pred_e = to_categorical(y_pred)

    for i in range(len(self.y_test[0])):
        fpr[i], tpr[i], _ = roc_curve(self.y_test[:, i], y_pred_e[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    fpr["micro"], tpr["micro"], _ = roc_curve(self.y_test.ravel(),
y_pred_e.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(len(self.y_test[0]))]))

    mean_tpr = np.zeros_like(all_fpr)
    for i in range(len(self.y_test[0])):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

    mean_tpr /= len(self.y_test[0])
    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    return {
        "confusion": conf,
        "classification_report": clrp,
        "fpr": fpr,
        "tpr": tpr,
        "roc_auc": roc_auc
    }
```

Klasik makine öğrenmesi için metriklerin çıkarıldığı kısım burasıdır.

```
def predict(self,im):
    img = cv2.imread(im)
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    img = Image.fromarray(img,'RGB')
    image = img.resize(self.global_image_size)
    image = np.array(image)
    image = image / 255
    if self.model_dict["data_informations"]["ClassifierAlgorithm"] ==
DataInformations.CLASSIFIER_ANN.value:
        a = []
        a.append(image)
        a = np.array(a)
        score = self.model.predict(a,verbose=1)
        label_index = np.argmax(score)
        acc = np.max(score)
        weather = self.global_labels[int(label_index)]
        return weather,round(acc*100)
    else:
        input_img = np.expand_dims(image,axis=0)
        input_img_feature = self.VGG16_extractor.predict(input_img)
        input_img_feature =
input_img_feature.reshape(input_img_feature.shape[0],-1)
        prediction_RF_ARR = self.model.predict(input_img_feature)
        prediction_RF = prediction_RF_ARR[0]
        label_index = np.argmax(prediction_RF)
        weather = self.global_labels[int(label_index)]
        return weather,-1
```

Modelin tahmin yaptığı metot buradadır. Sınıflandırma işlemi bittikten sonra herhangi bir yeni resim üzerinde tahmin yapmak ve başarımı görmek için burası kullanılır.

```python
class DataInformations(Enum):
    #Classifiers
    CLASSIFIER_ANN = 0
    CLASSIFIER_RF = 1
    #Datasets
    DATASET_ORIGINAL = 0
    DATASET_AUGMENTATED_AND_ORIGINAL = 1
    #Deep learning algorithms
    DL_VGG16 = 0
    DL_RESNET50 = 1
```

Global sabitlerin tanımlandığı sınıflardır. Veri ön işleme ekranında seçilen Combobox seçeneklerinin if-else sorgularında daha okunabilir bir kod ortaya çıkarması ve müdahale edilebilirliği arttırmak için yapılan bir yöntemdir.

```
def start(self):
    original_data, original_labels = self.get_image_data()
    original_he_data,original_he_labels =
self.get_image_data(data_path="preprocessed/images/he_images")
    original_clahe_data,original_clahe_labels =
self.get_image_data(data_path="preprocessed/images/clahe_images
")

    self.global_image_data =
np.concatenate([original_data,original_he_data,original_clahe_data])
    self.global_label_data =
np.concatenate([original_labels,original_he_labels,original_clahe_labe
ls])

    if self.model_dict["data_informations"]["Dataset"] ==
DataInformations.DATASET_AUGMENTATED_AND_ORIGINAL.value:
        augmentated_data, augmentated_labels =
self.get_image_data()
        augmentated_he_data, augmentated_he_labels =
self.get_image_data(data_path="preprocessed/augmentated_images
/he_images")
        augmentated_clahe_data, augmentated_clahe_labels =
self.get_image_data(data_path="preprocessed/augmentated_images
/clahe_images")
        self.global_image_data =
np.concatenate([self.global_image_data,augmentated_data,augment
ated_he_data,augmentated_clahe_data])
        self.global_label_data =
np.concatenate([self.global_label_data,augmentated_labels,augment
ated_he_labels,augmentated_clahe_labels])

    self.shuffle_data()
    self.image_data_split()
    if self.model_dict["data_informations"]["ClassifierAlgorithm"] ==
DataInformations.CLASSIFIER_RF.value:
        self.create_rf_model()
        self.rf_model_fit()
        self.classification_rep = self.get_rf_confusion_matrix()
        plot_path = os.path.join(self.model_dict["workdir"], "plots")
        if os.path.exists(plot_path):
            shutil.rmtree(plot_path)
        os.mkdir(plot_path)
        plt.figure(figsize=(5, 3))
```

```
        sns.set(font_scale=1.2)
        ax = sns.heatmap(self.classification_rep["confusion"],
annot=True, xticklabels=self.global_labels,
                         yticklabels=self.global_labels, cbar=False,
                         cmap='Blues', linewidths=1, linecolor='black',
fmt='.0f')
        plt.yticks(rotation=0)
        plt.xlabel('Predicted labels')
        plt.ylabel('True labels')
        ax.xaxis.set_ticks_position('top')
        plt.title('Confusion matrix')
        plt.savefig(os.path.join(plot_path, "rf_confusion_matrix.png"))
        plt.cla()
        plt.clf()
        plt.figure()
        plt.plot(self.classification_rep["fpr"]["micro"],
self.classification_rep["tpr"]["micro"],
                 label='micro-average ROC curve (area = {0:0.2f})'
                       ''.format(self.classification_rep["roc_auc"]["micro"]),
                 color='deeppink', linestyle=':', linewidth=4)

        plt.plot(self.classification_rep["fpr"]["macro"],
self.classification_rep["fpr"]["macro"],
                 label='macro-average ROC curve (area = {0:0.2f})'
                       ''.format(self.classification_rep["roc_auc"]["macro"]),
                 color='navy', linestyle=':', linewidth=4)

        colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
        for i, color in zip(range(len(self.y_test[0])), colors):
            plt.plot(self.classification_rep["fpr"][i],
self.classification_rep["tpr"][i], color=color, lw=2,
                     label='ROC curve of class {0} (area = {1:0.2f})'
                           ''.format(i, self.classification_rep["roc_auc"][i]))

        plt.plot([0, 1], [0, 1], 'k--', lw=2)
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Some extension of Receiver operating characteristic to
multi-class')
        plt.legend(loc="lower right")
        plt.savefig(os.path.join(plot_path, "rf_roc_graph.png"))
```

```
        plt.cla()
        plt.clf()

    elif self.model_dict["data_informations"]["ClassifierAlgorithm"] ==
DataInformations.CLASSIFIER_ANN.value:
        self.create_model()
        self.model_fit()
        self.classification_rep = self.get_confusion_matrix()

        plot_path = os.path.join(self.model_dict["workdir"], "plots")
        if os.path.exists(plot_path):
            shutil.rmtree(plot_path)

        os.mkdir(plot_path)

        plt.plot(self.history.history["accuracy"])
        plt.plot(self.history.history["val_accuracy"])
        plt.title("model accuracy")
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')

        plt.savefig(os.path.join(plot_path,"accuracy_graph.png"))
        plt.cla()
        plt.clf()

        plt.plot(self.history.history['loss'])
        plt.plot(self.history.history['val_loss'])
        plt.title('model loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.savefig(os.path.join(plot_path, "loss_graph.png"))
        plt.cla()
        plt.clf()

        plt.figure(figsize=(5, 3))
        sns.set(font_scale=1.2)
        ax = sns.heatmap(self.classification_rep["confusion"],
annot=True, xticklabels=self.global_labels,
yticklabels=self.global_labels, cbar=False,
                         cmap='Blues', linewidths=1, linecolor='black',
fmt='.0f')
```

```
        plt.yticks(rotation=0)
        plt.xlabel('Predicted labels')
        plt.ylabel('True labels')
        ax.xaxis.set_ticks_position('top')
        plt.title('Confusion matrix')
        plt.savefig(os.path.join(plot_path,"confusion_matrix.png"))
        plt.cla()
        plt.clf()

        plt.figure()
        plt.plot(self.classification_rep["fpr"]["micro"],
self.classification_rep["tpr"]["micro"],
                 label='micro-average ROC curve (area = {0:0.2f})'
                       ''.format(self.classification_rep["roc_auc"]["micro"]),
                 color='deeppink', linestyle=':', linewidth=4)

        plt.plot(self.classification_rep["fpr"]["macro"],
self.classification_rep["fpr"]["macro"],
                 label='macro-average ROC curve (area = {0:0.2f})'
                       ''.format(self.classification_rep["roc_auc"]["macro"]),
                 color='navy', linestyle=':', linewidth=4)

        colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
        for i, color in zip(range(len(self.y_test[0])), colors):
            plt.plot(self.classification_rep["fpr"][i],
self.classification_rep["tpr"][i], color=color, lw=2,
                     label='ROC curve of class {0} (area = {1:0.2f})'
                           ''.format(i, self.classification_rep["roc_auc"][i]))

        plt.plot([0, 1], [0, 1], 'k--', lw=2)
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Some extension of Receiver operating characteristic to
multi-class')
        plt.legend(loc="lower right")
        plt.savefig(os.path.join(plot_path,"roc_graph.png"))
        plt.cla()
        plt.clf()
    self.global_image_data = []
    self.global_label_data = []
    self.X_train = []
```

```
    self.X_test = []
    self.y_train = []
    self.y_test = []
```

Sınıflandırmanın başladığı ana metot burasıdır. Raporlama ekranına geçilirken start metodu çağırılır. Diğer tüm yardımcı metotları çağırarak sınıflandırma ve eğitim işlemlerini gerçekler. Klasik makine öğrenmesi için ayrı, Transfer learning için ayrı çalışır.

Aşama 1 : Önce orijinal verileri okur daha sonra arttırılmış verileri okur. Her bir küme için HE ve CLAHE veri kümelerini de okur. Verileri global bir listeye ekler.

Aşama 2 : Veri okuma işlemi bittikten sonra önce verileri karıştırır sonra X_train,y_train,X_test,y_test şeklinde parçalara ayırır.

Aşama 3 : Burada klasik makine öğrenmesi ve Transfer learning yapay sinir ağları olmak üzere program 2 parçaya dallanır.

Aşama 4 : Model eğitimi gerçekleştirilir

Aşama 5 : Model eğitim sonucunda test verileri ile metrikler hesaplanır.

-> Rapor çıktıları

-> Grafik resimleri

olarak kaydedilirler. PyQT arayüzünden bu grafik resimleri okunur ve ekrana yansıtılır. Rapor çıktıları ise bir textBox üzerinde gösterilir.

Aşama 6 (Opsiyonel) : Veri setinden bağımsız görseller ile tahminler yapılabilir.