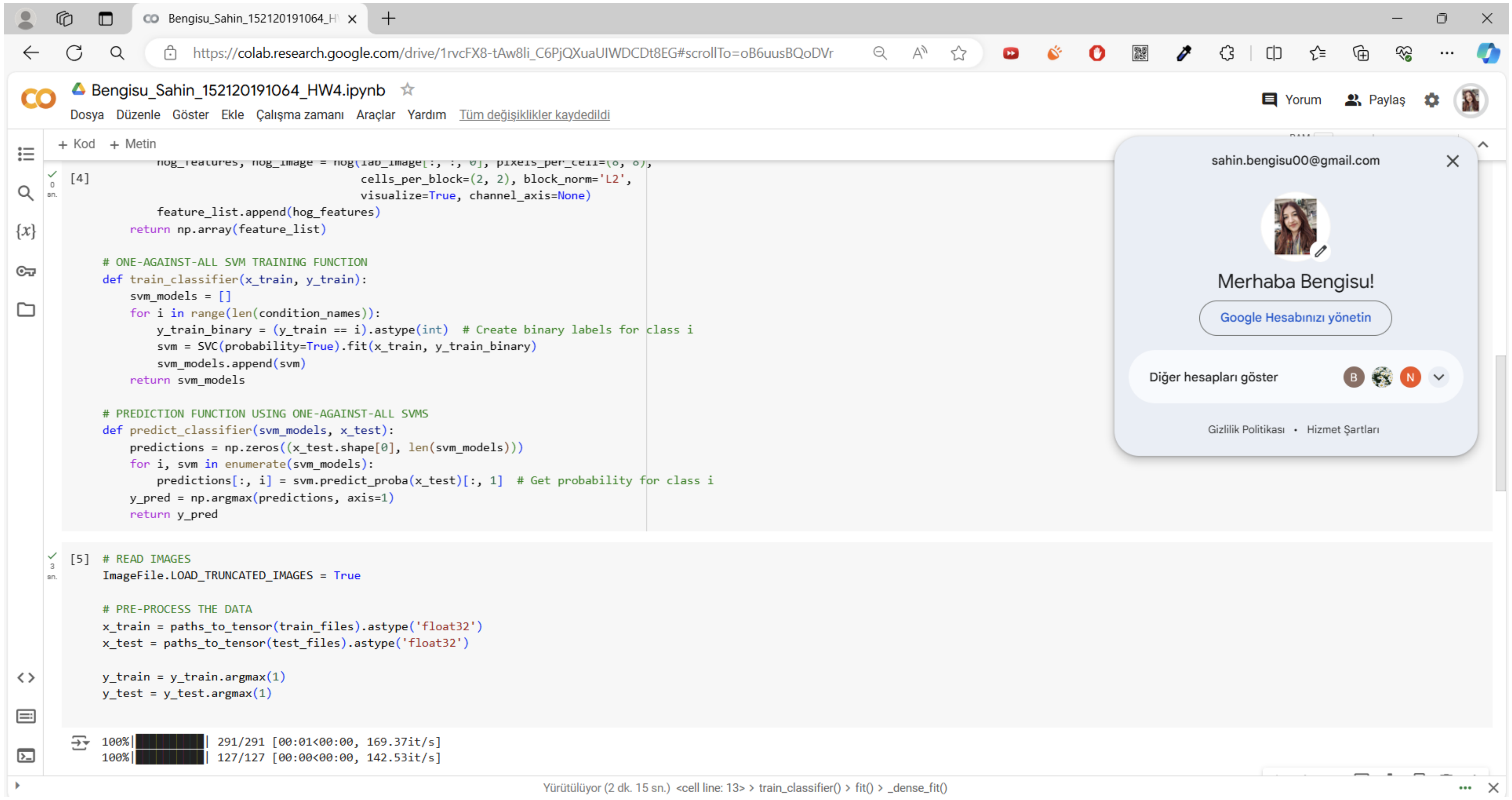
Bengisu Şahin

152120191064

Patter Recognition HW4 Report

- Take a snapshot of your all computer screen, which is related to the google Colab or your Spyder, as given below. Just Pres the **PrntScr** button and paste it to below. As you can see, at the top-right corner, your face must be available.


Bengisu_Sahin_152120191064_H
https://colab.research.google.com/drive/1rvcFX8-tAw8li_C6PjQXuaUIWDCDt8EG#scrollTo=oB6uusBQoDVr
Bengisu_Sahin_152120191064_HW4.ipynb
Dosya
Düzenle
Göster
Ekle
Çalışma zamanı
Araçlar
Yardım
Tüm değişiklikler kaydedildi
Yorum
Paylaş
+ Kod
+ Metin
sahin.bengisu00@gmail.com
Merhaba Bengisu!
Google Hesabınızı yönetin
Diğer hesapları göster
Gizlilik Politikası
Hizmet Şartları

```
[4]     hog_features, hog_image = hog(lab_image[:, :, 0], pixels_per_cell=(8, 8),
                                      cells_per_block=(2, 2), block_norm='L2',
                                      visualize=True, channel_axis=None)
            feature_list.append(hog_features)
        return np.array(feature_list)

    # ONE-AGAINST-ALL SVM TRAINING FUNCTION
    def train_classifier(x_train, y_train):
        svm_models = []
        for i in range(len(condition_names)):
            y_train_binary = (y_train == i).astype(int)  # Create binary labels for class i
            svm = SVC(probability=True).fit(x_train, y_train_binary)
            svm_models.append(svm)
        return svm_models

    # PREDICTION FUNCTION USING ONE-AGAINST-ALL SVMS
    def predict_classifier(svm_models, x_test):
        predictions = np.zeros((x_test.shape[0], len(svm_models)))
        for i, svm in enumerate(svm_models):
            predictions[:, i] = svm.predict_proba(x_test)[:, 1]  # Get probability for class i
        y_pred = np.argmax(predictions, axis=1)
        return y_pred
```

```
[5] # READ IMAGES
    ImageFile.LOAD_TRUNCATED_IMAGES = True

    # PRE-PROCESS THE DATA
    x_train = paths_to_tensor(train_files).astype('float32')
    x_test = paths_to_tensor(test_files).astype('float32')

    y_train = y_train.argmax(1)
    y_test = y_test.argmax(1)
```

```
100%|██████████| 291/291 [00:01<00:00, 169.37it/s]
100%|██████████| 127/127 [00:00<00:00, 142.53it/s]
```

Yürütülüyor (2 dk. 15 sn.) <cell line: 13> > train_classifier() > fit() > _dense_fit()

1. This code defines a function load_dataset(path) that loads datasets from the specified path using the load_files function. It then converts the filenames and targets into numpy arrays, where the targets are one-hot encoded using to_categorical function. The function returns the filenames and their corresponding one-hot encoded targets.

2. Functions and their explanations:

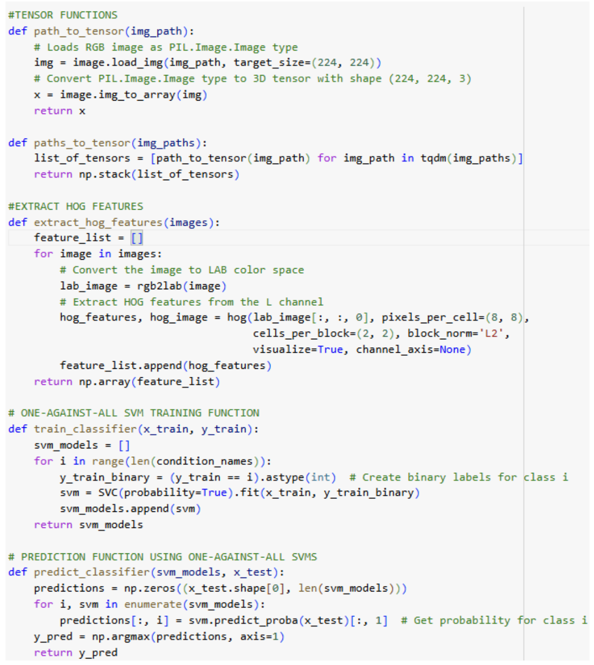
   path_to_tensor: Loads an RGB image from a given path and converts it into a 3D tensor with shape (224, 224, 3).

   paths_to_tensor: Converts a list of image paths into a stack of 3D tensors.

   extract_hog_features: Extracts Histogram of Oriented Gradients (HOG) features from the L channel of images converted to the LAB color space.

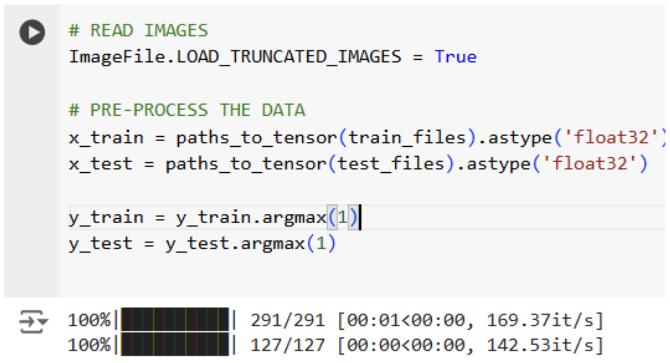
   train_classifier: Trains one-against-all SVM classifiers for each class in the dataset using the provided training data and labels.

   predict_classifier: Predicts the class labels for the test data using the trained one-against-all SVM classifiers by aggregating their probability outputs.



```python
#TENSOR FUNCTIONS
def path_to_tensor(img_path):
    # Loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # Convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    return x

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    return np.stack(list_of_tensors)

#EXTRACT HOG FEATURES
def extract_hog_features(images):
    feature_list = []
    for image in images:
        # Convert the image to LAB color space
        lab_image = rgb2lab(image)
        # Extract HOG features from the L channel
        hog_features, hog_image = hog(lab_image[:, :, 0], pixels_per_cell=(8, 8),
                                      cells_per_block=(2, 2), block_norm='L2',
                                      visualize=True, channel_axis=None)
        feature_list.append(hog_features)
    return np.array(feature_list)

# ONE-AGAINST-ALL SVM TRAINING FUNCTION
def train_classifier(x_train, y_train):
    svm_models = []
    for i in range(len(condition_names)):
        y_train_binary = (y_train == i).astype(int)  # Create binary labels for class i
        svm = SVC(probability=True).fit(x_train, y_train_binary)
        svm_models.append(svm)
    return svm_models

# PREDICTION FUNCTION USING ONE-AGAINST-ALL SVMS
def predict_classifier(svm_models, x_test):
    predictions = np.zeros((x_test.shape[0], len(svm_models)))
    for i, svm in enumerate(svm_models):
        predictions[:, i] = svm.predict_proba(x_test)[:, 1]  # Get probability for class i
    y_pred = np.argmax(predictions, axis=1)
    return y_pred
```

3. The code prepares image data and labels for other step. It ensures truncated images are loaded, converts image file paths into 3D tensors with `paths_to_tensor`, and casts these tensors to `float32`. It also converts one-hot encoded labels into class indices using `argmax`, making the data ready for feature extraction and model training.



```
# READ IMAGES
ImageFile.LOAD_TRUNCATED_IMAGES = True

# PRE-PROCESS THE DATA
x_train = paths_to_tensor(train_files).astype('float32'
x_test = paths_to_tensor(test_files).astype('float32')

y_train = y_train.argmax(1)
y_test = y_test.argmax(1)
```

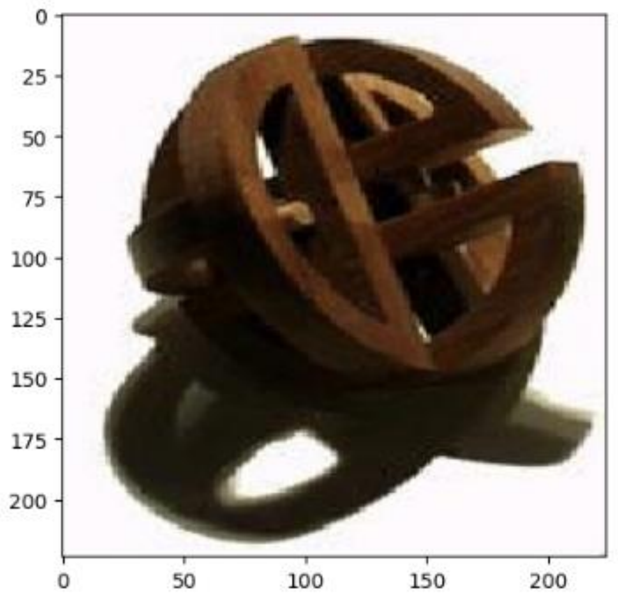
```
100%|██████████| 291/291 [00:01<00:00, 169.37it/s]
100%|██████████| 127/127 [00:00<00:00, 142.53it/s]
```

4. The provided code performs a complete machine learning workflow for image classification using HOG features and one-against-all SVM models. Initially, HOG features are extracted from both the training and test images. These features are then normalized using a standard scaler to ensure consistent scaling. SVM models are trained on the normalized training features, each focusing on distinguishing one class against all others. The trained models are used to predict class labels for the test set, and the accuracy and classification report are printed to evaluate performance. Additionally, the code visualizes a sample test image, extracts and normalizes its features, and uses the trained models to predict and print its actual and predicted labels, demonstrating the prediction process on an individual sample.

```python
# STEP 1: FEATURE EXTRACTION FOR TRAIN DATA
x_train_features = extract_hog_features(x_train)

# STEP 2: FEATURE EXTRACTION FOR TEST DATA
x_test_features = extract_hog_features(x_test)

# Normalize the HOG features
scaler = StandardScaler().fit(x_train_features)
x_train_features = scaler.transform(x_train_features)
x_test_features = scaler.transform(x_test_features)

# STEP 3: TRAIN ONE-AGAINST-ALL SVM MODELS
svm_models = train_classifier(x_train_features, y_train)

# STEP 4: MAKE PREDICTIONS
y_pred = predict_classifier(svm_models, x_test_features)

# STEP 5: COMPUTE ACCURACY
print('Accuracy: ' + str(accuracy_score(y_test, y_pred)))
print('\n')
print(classification_report(y_test, y_pred, zero_division=1))  # Adjust zer

# STEP 6: MAKE PREDICTION FOR SAMPLE TEST
sample_test = x_test[0]
plt.imshow(sample_test.reshape(224, 224, 3).astype('uint8'))  # Ensure valu
plt.show()

sample_test = np.expand_dims(sample_test, axis=0)  # Add a new axis to make

sample_test_features = extract_hog_features(sample_test)
sample_test_features = scaler.transform(sample_test_features)
y_pred_sample_test = predict_classifier(svm_models, sample_test_features)

# Print the actual and predicted labels
actual_label = condition_names[y_test[0]]
predicted_label = condition_names[y_pred_sample_test[0]]

print('Actual class label of test sample is:', actual_label)
print('Predicted class label of test sample is:', predicted_label)
```

5. Result is below:

```
Accuracy: 0.6850393700787402

              precision    recall  f1-score   support

           0       0.62      0.78      0.69        46
           1       1.00      1.00      1.00         4
           2       1.00      0.85      0.92        13
           3       1.00      0.25      0.40         4
           4       0.00      0.00      0.00         4
           5       0.33      0.50      0.40         4
           6       0.50      0.50      0.50         4
           7       0.00      0.00      0.00         4
           8       0.00      0.00      0.00         2
           9       0.73      0.73      0.73        11
          10       0.88      0.88      0.88         8
          11       1.00      0.33      0.50         3
          12       0.89      1.00      0.94         8
          13       1.00      0.62      0.77         8
          14       0.67      0.50      0.57         4

    accuracy                           0.69       127
   macro avg       0.64      0.53      0.55       127
weighted avg       0.70      0.69      0.67       127
```



```
Actual class label of test sample is: BACKGROUND_Google
Predicted class label of test sample is: BACKGROUND_Google
```