Computer Vision Project Report
TEXTURE SAMPLES RECOGNITION

Szymon Ślusarz

Faculty of Mathematics and Information Science, Warsaw University of Technology

January 20, 2022

1 Introduction

This project's goal is to acquire knowledge about techniques frequently used in image processing. This time we will be focusing on texture recognition using image classification with different types of texture features.

During this project, I will work on public dataset called *KTH-texture-Data*. We can find there two directories: train and valid, each of them contain 11 subdirectories with different types of textures:

- *KTH_aluminum_foil*
- *KTH_brown_bread*
- *KTH_corduroy*
- *KTH_cork*
- *KTH_cotton*
- *KTH_cracker*
- *KTH_linen*
- *KTH_orange_peel*
- *KTH_sponge*
- *KTH_styrofoam*
- *KTH_wool*

Figure 1: Examples of the above textures

First of all, I had to load images from *train* directory and train the classification model on them, to later on use the prepared classification in predicting the texture name to each image from *valid* directory.

2 Description of solution

2.1 Choosing features

One of the most important things, when it comes to texture recognition is a choice of texture features, based on which we will do the classification. In case of this project, I decided to use 3 different feature descriptors, and perform classification based on each of them, and one more based on combination of two of them. I implemented each of the features in corresponding functions.

First feature I picked is *Heralick texture feature*. It is used to describe a texture of the image, it is useful to distinguish between for example smooth and rough surfaces.

```
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick
```

Second feature descriptor I used is *Color Histogram feature*. It is a representation of the distribution of colors in an image, so it can be useful in distinguishing textures based on their colors.

```
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0,
   256, 0, 256])
    cv2.normalize(hist, hist)
    return hist.flatten()
```

Last but not least feature descriptor I chose is *Local Binary Pattern feature (LBP)*. It computes a local representation of texture. This local representation is constructed by comparing each pixel with its surrounding neighborhood of pixels. So it is definitely very useful in texture recognition.

```
def fd_lbp(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    lbp = skifeat.local_binary_pattern(gray, 24, 8, method="uniform")
    (hist, _) = np.histogram(lbp.ravel(),
                             bins=np.arange(0, 24 + 3),
                             range=(0, 24 + 2))
    # normalize the histogram
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    # return the histogram of Local Binary Patterns
    return hist
```

2.2 Choice of classifier

After choosing a few feature descriptors, it is necessary to find a good classifier model, that will learn how to distinguish different textures based on the feature descriptors given above. After testing few of them, for example: *KNeighborsClassifier*, *SVC*, *GaussianProcessClassifier* etc., I decided to go for ***RandomForestClassifier***, as it turned out to be the most accurate between the tested ones.

2.3 Classifier model training

This part will be focused on the most important thing of the whole topic, which is preparation for classification model training and the training itself.

Firstly, I initialized 2 arrays needed for training, *train_labels*, which stores texture label of each image from the train directory, and *global_features*, which stores feature descriptor of images corresponding to the mentioned labels.

In order to fill the arrays with necessary data, I had to loop through every image in the *train* directory, and append its labels and feature descriptors to the mentioned variables.

```
global_features = []
train_labels = []

for training_name in train_filenames:
    # join the training directory with the texture name
    dir = os.path.join(train_path, training_name)
    # get the current training texture label
    current_label = training_name[4:]
    # loop over the images in each texture subfolder
    for img in os.listdir(dir):
        # image file name
        file = dir + "/" + str(img)
        # reading image
        image = cv2.imread(file)

        # Here we pick the feature that we want to test and comment
        # the remaining ones:

        feature = fd_haralick(image)
        # feature = fd_histogram(image)
        # feature = np.hstack([fd_haralick(image), fd_histogram(image)])
        # feature = fd_lbp(image)

        # appending current texture label and its feature to corresponding
    arrays
        train_labels.append(current_label)
        global_features.append(feature)
```

Afterwards, I could proceed to the main part of this section - model training. As I mentioned I chose textit**RandomForestClassifier**, so I assigned it with value of 100 trees to a variable *clf*. Then I executed *clf.fit()* function, with *global_features* and *test_labels* as its arguments. Now the classifier model is being trained based on the features and corresponding labels.

```
clf = RandomForestClassifier(n_estimators=100, random_state=9)
clf.fit(global_features, train_labels)
```

2.4 Testing

As soon as my model is trained, I could move on to testing it. Again I had to iterate through all images in valid directory, which is made to test our classifier. I checked what is the real file descriptor of each image, and at the same time executed *predict()* function on them, which predicts what texture is it based on the pretrained classifier. Afterwards I compared the label given by the prediction, and with the actual label of the texture. If both outputs were the same, I incremented counter of hits, which I used to calculate accuracy percentage for each texture and for the whole dataset, by dividing it by the number of images for each texture or the total number of images in *valid* directory.

```
test_labels = []
total_hits = 0

for testing_name in test_filenames:
    # join the testing directory with the texture name
    dir = os.path.join(valid_path, testing_name)
    # get the current training texture label
    current_label = testing_name[4:]
    hits = 0
    count = 0
    for image in os.listdir(dir):
        # appending correct label of the texture to the array
        test_labels.append(current_label)
        # image file name
        file = dir + "/" + str(image)
        # reading file
        img = cv2.imread(file)

        # Here we pick the same feature as in the previous loop
        # and comment the remaining ones:

        feature = fd_haralick(img)
        # feature = fd_histogram(img)
        # feature = np.hstack([fd_haralick(img), fd_histogram(img)])
        # feature = fd_lbp(img)

        # using pretrained classifier to predict the label of the texture
        prediction = clf.predict(feature.reshape(1, -1))[0]

        count += 1
        # calculating accuracy percentages:
        if prediction == current_label:
            total_hits += 1
            hits += 1
    print(str(current_label) + ' correct percentage: ', 100*hits/count)

print('Percentage correct: ', 100*total_hits/len(test_labels))
```

3 Results

3.1 Mean results

I decided to present the results in form of a table with accuracy percentages for each texture separately, and total accuracy percentage from all textures. I did it for every fea-

ture descriptor described in section 2.1, and as it was also mentioned in this section - feature descriptor created with connection of Haralick and Color histogram texture features.

	Haralick	***Color Histogram***	***Haralick&Histogram***	***LBP***
KTH_aluminium_foil	90.74%	99.53%	99.53%	94.44%
KTH_brown_bread	48.78%	78.04%	90.24%	90.24%
KTH_corduroy	70.70%	95.31%	92.57%	74.60%
KTH_cork	83.79%	92.59%	91.20%	92.59%
KTH_cotton	81.71%	96.88%	96.88%	82.10%
KTH_cracker	58.82%	76.47%	88.23%	58.82%
KTH_linen	85.60%	92.99%	93.38	80.93%
KTH_orange_peel	84%	100%	100%	72%
KTH_sponge	68.29%	82.92%	85.36%	97.56%
KTH_styrofoam	63.41%	100%	100%	87.80%
KTH_wool	79.62%	99.07%	99.07%	75%
TOTAL	**79.68%**	**94.93%**	**95%**	**82.93%**

As we can see on the table above, in total the most accurate feature descriptor turned out to be the one made out from *Haralick* and *Color Histogram*. What is interesting, the usage of *Color Histogram* feature descriptor resulted in 100% accuracy in recognition orange peel and styrofoam. That shouldn't be a surprise as this feature descriptor distinguishes the textures based on their colors, and orange peel as well as styrofoam, had noticeably distinctive colors, and moreover, their texture was quite smooth, so it was very easy for this descriptor to recognise them.

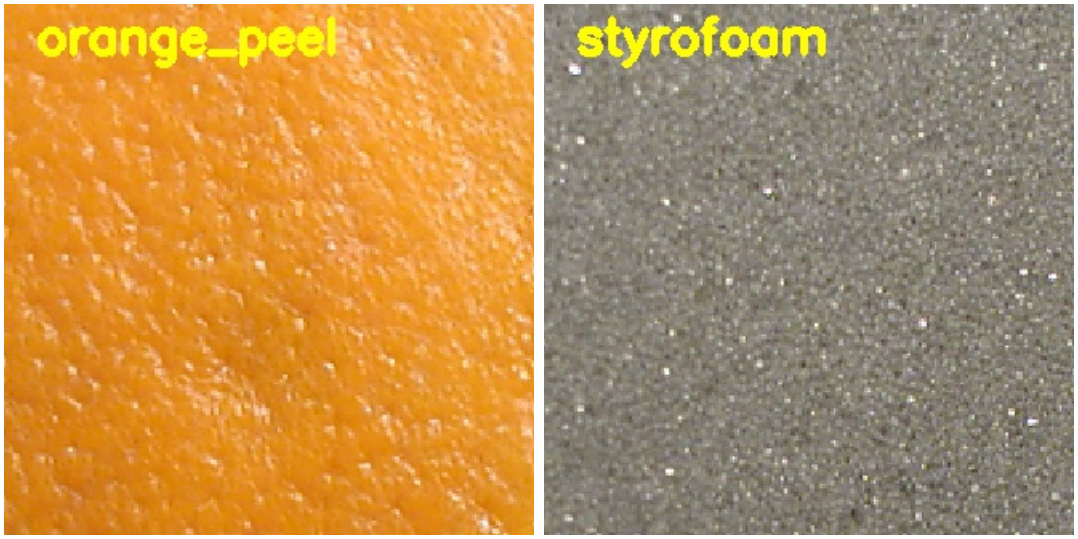

Figure 2: Orange peel and styrofoam with Color Histogram

What is also very intuitive, usage of *LBP* feature descriptor, which is based on comparison of pixels on the image, on orange peel and styrofoam, whose texture, as it was just said, is very smooth, should result in low accuracy, and indeed that is what we can read from the table. The following images present bad predictionns on the above textures.

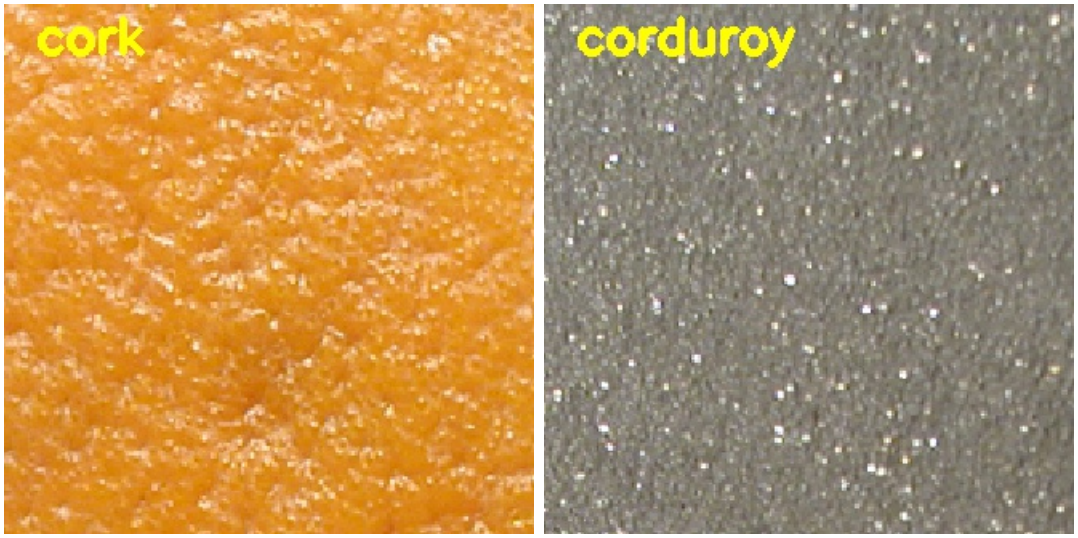

Figure 3: Orange peel and styrofoam with LBP

In case of *Haralick* feature descriptor, the rougher the texture was, the more accurate it turned out to be. As we can see, it returned the highest accuracy percentage while processing aluminium foil images, that are in most cases very rough. However for some of them that were noticeably less creased, the prediction was not correct.

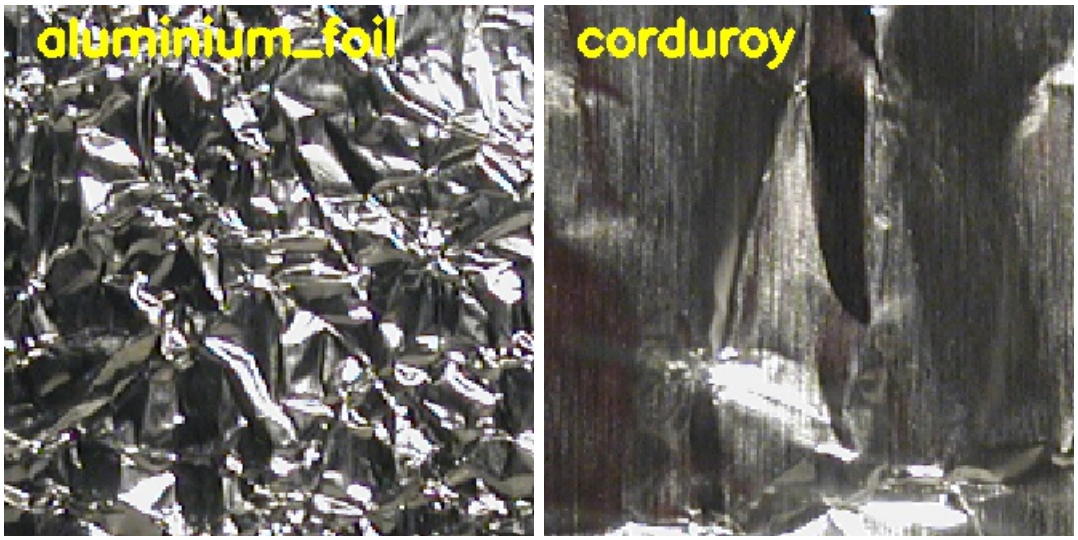

Figure 4: Aluminium foil with Haralick

3.2 Bad and good classification

In this section, I will present few of the bad and good classification results, apart from the examples given in the previous section.

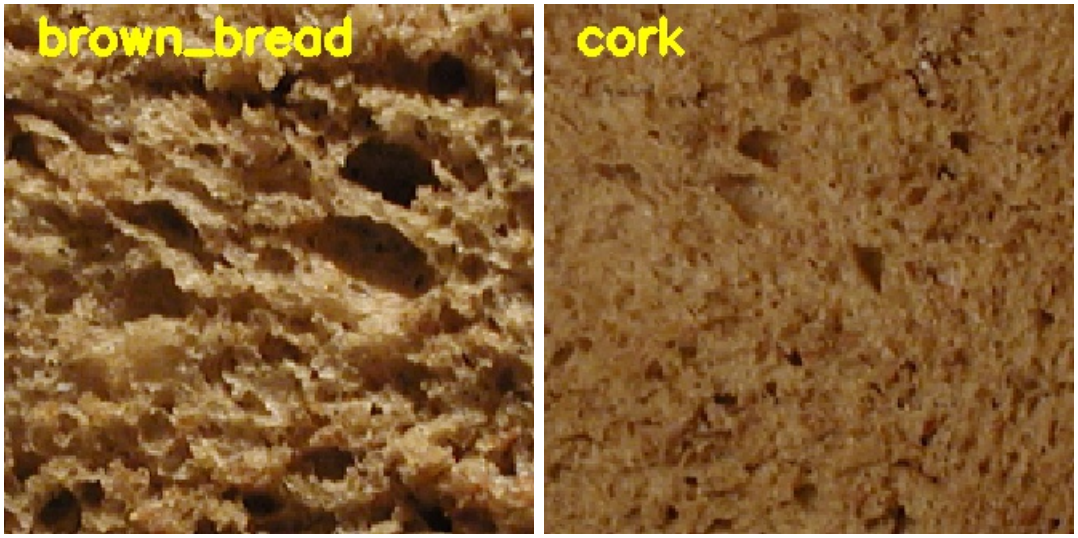

Figure 5: Two examples of brown bread with haralick

As we can see on the above images, haralick did well with brown bread with bigger holes, however, it did not predict the texture on the second image correctly, because it is very similar to cork texture.

Figure 6: Two examples of wool with LBP

On this example we can clearly see that both images are almost the same, however left texture is noticeably more fibrous, and that is probably what helped the classifier to identify this texture. On the right, we can see that it detected wool texture as linen, it shouldn't be a big surprise actually because the texture on the right image could be mistaken with the proper linen texture for example these one presented on fig: 7:

Figure 7: Example of linen texture

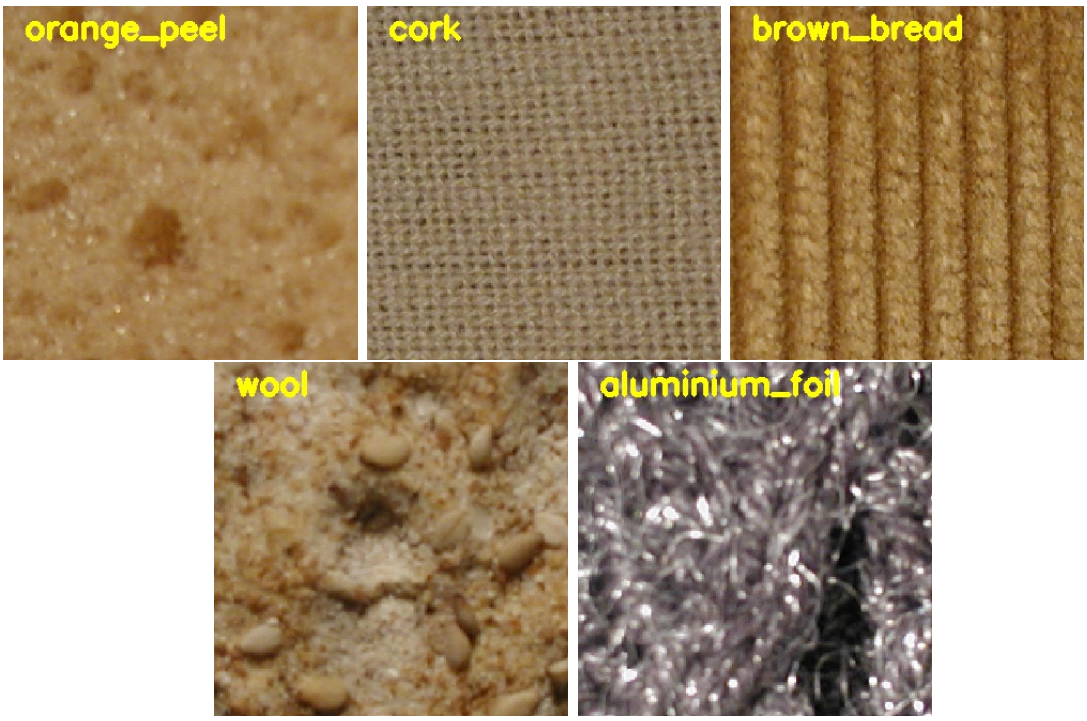

Figure 8: More examples of incorrect predictions

4 Conclusions

Summarising, I learned a lot of useful stuff in terms of image processing while working on this project, I also had a chance to implement the basics of machine learning. It was very interesting to analyse the functionalities, feature descriptors and a variety of classifiers. The obtained results turned out to be quite accurate, which means that my choices was justified. I also encountered a few problems, that took me a long time to fix, but finally I managed to deal with all of them.

References

[1] https://pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/

[2] https://gogulilango.com/software/image-classification-python

[3] https://kapernikov.com/tutorial-image-classification-with-scikit-learn/