Deep Learning versus Classical Image Features

CS904 Assignment 3, Annika Stechemesser

I. Introduction

In the first two questions of this assignment we use the Kather2016 images provided on the module website to train classifiers. This data set splits into a training set, containing 4136 images, and a validation set, containing 836 images. The images are categorized in the eight different classes shown in Figure 1 (One example is shown for every class.).

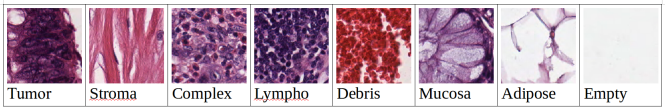

Figure 1: Eight different classes, example pictures.

The number of instances per class differ in the training set as well as in the validation set (Figure 2). This could influence the classification and we should be aware of it.

Class	Number of instances in training	Number of instances in validation
Tumor	424	201
Stroma	508	117
Complex	508	117
Lympho	536	89
Debris	536	89
Mucosa	524	101
Adipose	560	65
Empty	568	57

Figure 2: Number of instances per class per set.

In the last question of the assignment we apply the methods developed in the questions before to the Whole-Slide-Image (WSI) shown in figure 3, which was stained with Hematoxilyn and Eosin (H&E). The image has a width of 79328 pixels and a height of 199368 pixels. It comes with 11 wavelet decomposition levels. In Task 3 we will be using the levels 3 and 1.

The code for Task 1 and 2 of this assignment was written in Python, the code for Task 3 in Matlab. Details regarding the implementation can be found as comments in the code.

II. Task 1: Transfer Learning for Tissue Classification

In this task we want to use the deep learning to classify the images in the Kather2016 data set. Deep learning means using a special type of neural networks which typically use a cascade of multiple layers of non-linear processing units for

Figure 3: Whole Slide Image(level 3)

feature extraction and classification. Specifically, we want to use the VGG16 network whose structure is outlined in Figure 4.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Figure 4: Structure of the VGG16 network.

We download a version of the VGG16 architecture in

Keras which was pretrained on ImageNet. ImageNet is an image database containing over 14 billion natural images. The pretrained VGG16 model is therefore already performing very well on classification tasks involving natural images. However we are handling medical images, more specifically histopathology images, which are significantly different from natural images in their structure. Therefore, to achieve good classification results, we need to fine tune the VGG16 model with respect to our data. The classic approaches to fine tuning are to either just train the softmax layer of the network, the dense layers and the softmax layer or to retrain the whole network. Our approach is to keep the pretrained VGG16 network up to the "flatten" layer and then train a smaller fully connected model on top of these stored features. We expect the performance of the layers to be better the "deeper" the networks gets as more parameters were trained. As we keep the weights up to "flatten", we want to see how distinctive the extracted features from the flatten layer already are. Extracting the features from the flatten layer outputs a feature vector of size 25088 for each image. We use principal component analysis to reduce the length of the vector to 50 and then we apply TSNE to embed this high dimensional data in the two dimensional space so that we can plot it. The results are shown in Figure 5.

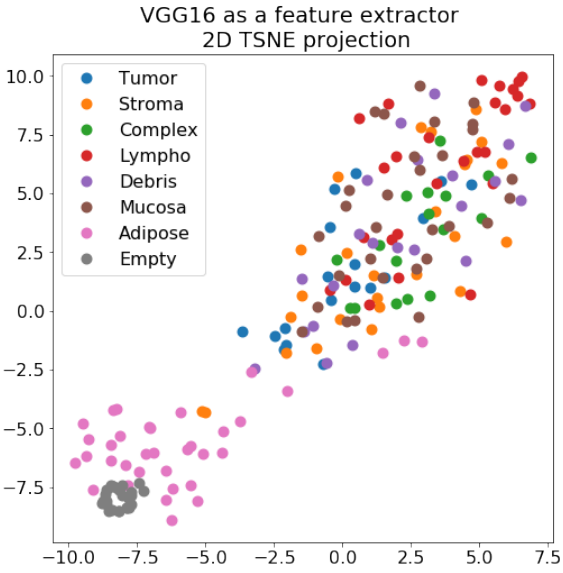

Figure 5: VGG as a feature extractor.

We see that two classes got separated out very well, the "adipose" and the "empty" class. This makes sense as they have a high structural difference to the other classes. All the pictures in the "empty" class consist of background only and are therefore in color and structure completely different from the tissue images. The "adipose" class consists of images of the fat cells in the tissue. These are translucent and relatively large and therefore in color and shape also very different from the other tissue. The TSNE plot suggests that the other classes don't get separated very well. The real performance though might vary as we heavily reduced the dimensions and only look at a batch of data for this plot.

The model we train on top of these frozen layers consists of a dense layer with a rectified linear unit activation function, a dropout layer with probability 0.5 and a softmax layer with output of size eight. The total number of parameters in this model is 2099464. It is shown in Figure 6. We use categorical cross entropy as a loss function and the build in Keras optimizer "ADAM" as an optimizer. The parameters are set like suggested in the paper "ADAM - A Method For Stochastic Optimization". The learning rate is very slow, which is important for fine tuning.

flatten_1 (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 256)	2097408
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 8)	2056

Figure 6: Changed Layers.

We then import the training and validation data using data generators. We specify a batch size of 128 and run 5 epochs. The final accuracy that was achieved is 0.8684 percent (Figure 7). Considering the output from the flatten layer, this is a very satisfying performance.

```
Epoch 1/5
33/33 [==============================] - 526s 16s/step - loss: 1.0154 - acc: 0.6520 - val_loss: 0.6102 - val_acc:
0.7955
Epoch 2/5
33/33 [==============================] - 533s 16s/step - loss: 0.5070 - acc: 0.8236 - val_loss: 0.4269 - val_acc:
0.8624
Epoch 3/5
33/33 [==============================] - 535s 16s/step - loss: 0.4188 - acc: 0.8555 - val_loss: 0.4546 - val_acc:
0.8553
Epoch 4/5
33/33 [==============================] - 535s 16s/step - loss: 0.3619 - acc: 0.8751 - val_loss: 0.3588 - val_acc:
0.8828
Epoch 5/5
33/33 [==============================] - 526s 16s/step - loss: 0.3167 - acc: 0.8894 - val_loss: 0.3705 - val_acc:
0.8684
```

Figure 7: Performance for different epochs.

This model could be improved by for instance running more epochs, by retraining the convolutional layers or by using data augmentation to generate a bigger training set.

Now we want to evaluate how good the classification performance is for the individual tissue types. We use our model to predict the labels of the validation set and then use the predicted labels as well as the true labels of the validation set to compute a confusion matrix. Each row of the matrix corresponds to a predicted label and each column to a true label. If the true label and the predicted label match the value on the diagonal increases. If a class i is predicted and the true class is j the matrix element [i,j] is increased. The results are shown in Figure 8.

We see that in every row (and column) the diagonal element has the highest value. The difference in magnitude of the diagonal values is due to the different number of images for each class (see Figure 2). The prediction for most classes worked very well, which is in comformity with the high accuracy. The class images were most likely to get mislabelled for is "Complex", the classes that were slightly harder to predict are "Debris" and "Complex". The overall performance is good.

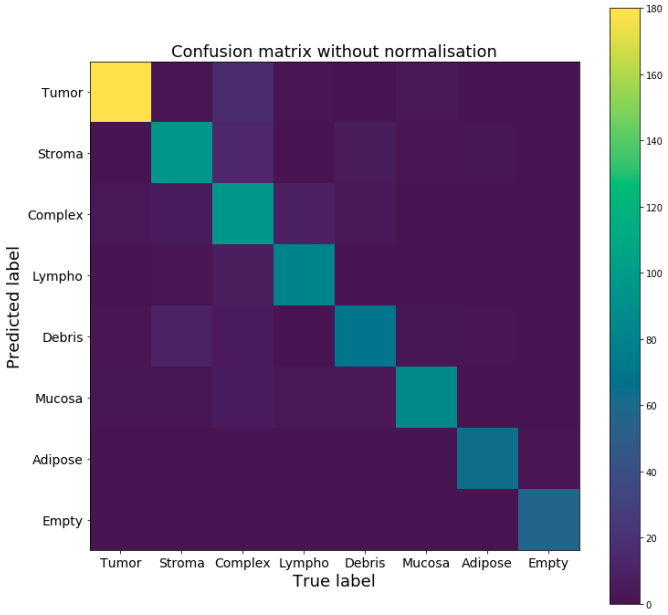

Figure 8: Confusion Matrix.

III. Task 2: Classical Image Features for Tissue Classification

In this question we want to classify our data using hand-crafted features. This means instead of going through the process of designing a deep neural network which takes input images and then first extracts low level features and then mid- to high level features, we now extract features and then manually select them. This process is suitable for smaller data sets and gives the user full control (therefore also a high risk of errors in the feature selection). In the paper "Multiclass texture analysis in colorectal cancer histology" was shown that textural features achieve good accuracy in multiclass texture classification. We're using the code that was published with the paper to extract some handcrafted features from the dataset and then train our own classifiers based on them. The paper groups features into six different sets of distinctors and suggests that the feature combination "Best5" works most efficiently. Therefore we will concentrate on the "Best5"features, which are described below.

A. *Examples for handcrafted features, "Best5"*

1) Lower-oder Histogram Features: Lower-oder histogram features basically mean the application of basic statistic measures to the gray level histogram of the image such as mean, variance, skewness [measures lack of symmetry], kurtosis [measures if the data is heavy-tailed or light-tailed in comparison to a normal distribution] and the 5th central moment of the histogram.

2) Higher-oder Histogram Features: Higher-order histogram features include in this context the second to the eleventh central moment of the gray level histogram. If we apply lower-order and higher-order histogram features in combination we obviously remove the overlapping features (2nd moment = variance, 3rd moment = skewedness, 4th moment = kurtosis, 5th moment included in both) and are then left with 11 histogram features in total.

3) Local Binary Patterns (LBP): Local binary patterns concentrate on evaluating whether points around a central point have a lower or higher pixel intensity than the central point. The answer is binary. To detect texture, a collection of LBPs over an image patch is required. The histogram of this LBP results can then be used to detect textures. In our specific case, we considered neighborhoods of eight equally-spaced points arranged along a circle of radius 1px. The resulting histograms were reduced to the 38 rotationally-invariant Fourier features proposed by "Ahonen et al". We therefore get 38 features based on the LBPs.

4) Gray-level co-occurrence matrix (GLCM): To compute a GLCM, an offset is fixed, in our case the offset consists of 5 displacement vectors in four different directions (average over these directions), and then a histogram of co-occurring greyscale values is computed. This gives five GLCM matrices, one for each of the offsets. For each of these contrast, correlation, energy and homogeneity is computed which results in 20 overall features per image.

5) Perception-like features: Perception-like features are modeled based on how the human visual system discriminates texture. Studies showed, that several specific attributes are important. The attributes that were used here are coarseness, contrast, directionality, line-likeness and roughness.

Considering these "best5" groups of features we get 11+38+20+5 = 74 features in total we extract for each image. Once we obtain the training and target sets for the training- and validation data, we can train our classifiers. Before we preprocess the data, meaning we replace missing values with the mean and normalize. We're then interested to see how distinctive these hand-crafted features are. To do that we apply TSNE to embed the high dimensional data in the two dimensional space (Figure 9)

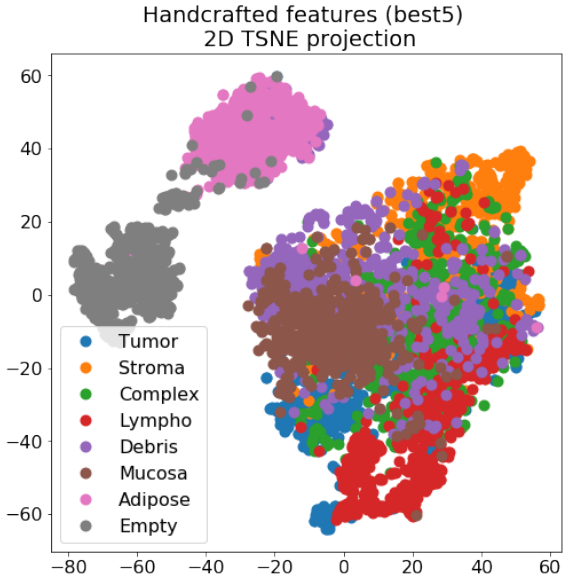

Figure 9: Handcrafted features (best5), TSNE

In contrast to Figure 5 here the whole data set was used.

Overall we can see a similar result as the extracted features of Flatten layer of the VGG16 gave. The classes "Empty" and "Adipose" separate out very nicely, the rest of the classes still overlap heavily.

B. Training Classifiers on classical image features

We train three different classifiers on the data set, a linear Support Vector Machine, a radial basis function Support Vector Machine and a random forest classifier using the Python package scikit-learn.

A support vector machine is interpreting data items as points in an n-dimensional space where n is the number of features. Each feature corresponds to one particular entry in the vector. The classifier is then attempting to find a set of hyperplanes that separates the classes as good as possible.

A random forest classifier uses a set of decision tree classifiers on different subsets of the dataset. It is averaging over these different fits which improves accuracy.

The accuracies of these approaches can be seen in Figure 10. They are similar to the results in the paper and also as good as the deep learning approach in Task 1, even though the deep learning approach could be extended very easily to improve performance whereas this approach probably doesn't have much room for improvement.

Classifier	Linear SVM	Radial basis function SVM	Random Forest
Accuracy	0.877	0.828	0.877

Figure 10: accuracy reached with the different classifiers

We now want to evaluate which selected features perform the best. To do that we compute the feature importances on the random forest and display the results in a bar chart (Figure 11). The features are labeled 0 to 73 and belong to the features in the order they are listed above. We see that the histogram features (first features) perform best. This is consistent with the results in the paper.

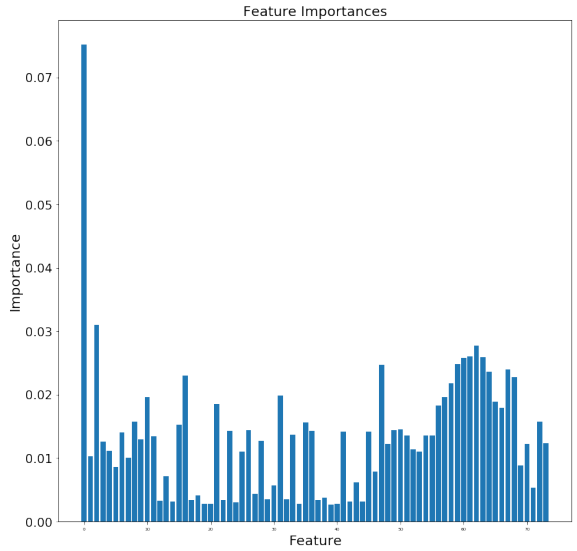

Figure 11: Feature Importances

IV. Task 3: Local Cell Density Estimation

In this question we want to combine the approaches from tasks one and two to perform a patch classification on the whole slide image shown in figure 3. As the image is in jp2 format, we can't read it in Python. Therefore the code for this question is written in Matlab. To classify regions in the whole slide image we split the image in non-overlapping patches of size 150x150 [same as Kather2016 data]. We then apply our classifiers to each patch of the image. We introduce a color code for every one of the eight classes and color every patch according to it's class color. The result is then a picture of the same size as the original image which indicates the class for each block.

The results in this task are heavily limited by memory issues. As the whole slide image is very big the Laptop could only process it on a low wavelet decomposition level (level 3). The magnification on level three is not the same as the magnification of the training images, therefore we can't really expect the classification to be sensible. We will nevertheless apply both approaches to the whole slide image to proof that the method works. To get more sensible results we will then in a second step apply the approaches to a subsection of the image on a higher magnification level.

A. Application of the two classification approaches on the WSI (level 3)

First we want to classify our patches using our fine-tuned VGG16 model. We import the model from Python and classify the patches in the manner described above. Our first result is displayed in figure 12. We see that it is not a good idea to use an unprocessed whole slide image, as some non-tissue regions got picked up as well (writing). Most of the background was classified correctly. Therefore, to improve performance and running time, we want to identify regions that are purely background in advance and then only classify patches that contain tissue. To this purpose, we use a standard deviation filter with Otsu thresholding to compute a binary mask in the same manner as it was done in Assignment 2. We also prune out regions that don't contain any tissue or are smaller than a certain threshold in the binary mask. For every patch we then take the same patch of the binary mask and only start the process if the total sum of the binary patch is greater than 0, indicating that it contains some tissue. This approach was a huge improvement in terms of running time. The results are shown in Figure 12

Next we want to classify the patches using the SVM classifier. We export the features for each patch in Matlab and use our previously trained linear SVM classifier in Python to classify the patches. To shorten the running time we again only classify patches that aren't purely background. We extract the best five feature sets from each patch and classify with the linear SVM. The results are shown in Figure 13.

We see that the classification gives very different results for both approaches even though they had a similar accuracy on the validation set (see Task 2). This could be due to using the wrong magnification. To get some more reliable results we know move on to considering a section of the WSI at level 1.

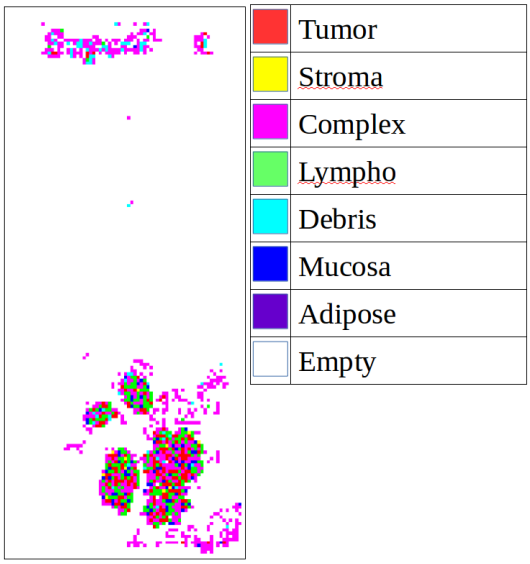

Figure 12: first attempt of classification, WSI

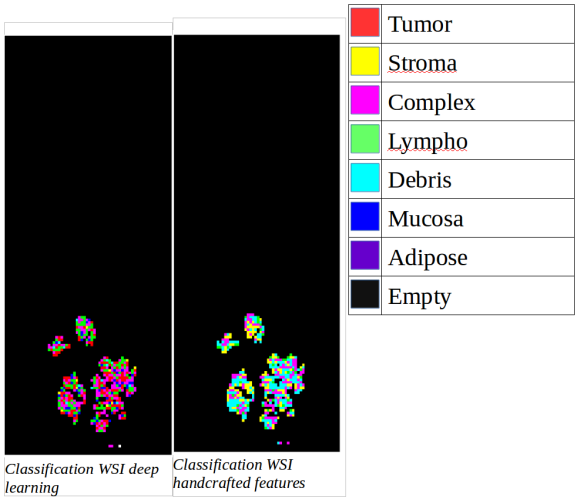

Figure 13: classification with both approaches, WSI

B. *Application of the two classification approaches on a section of the WSI (level 1)*

We read the image now at level 1, which I believe to have the same magnitude as the training images, and pick a section that has approximately the same number of pixels as the whole slide image on level 3 (246510000) and contains a lot of tissue. We decide for a section of size 15000x15000. We process this section in the same way we processed the WSI before. The results are shown in Figure 14.

We see that even on level 1 both methods lead to very different classifications. The deep learning classifier predicts that nearly all of the cells in the given section are lymphocytes, whereas the support vector machine predicts mostly debris. As we don't know the ground truth, we can't tell which classifier is right in this case (if any). However, if we compare to the results on level 3 we see that the prediction of the SVM is consistent on both levels, whereas the prediction of the deep learning classifier is completely different (mostly complex on level 3, mostly lympho on level 1). This could be an indication for the SVM to be working better in this case. Furthermore the SVM was used directly in Python, whereas the pretrained deep learning model was imported from Keras into Matlab to classify these images. It is possible that the import doesn't work correctly, some preprocessing is different or the network got damaged in some other way. However, I couldn't find proof for that.

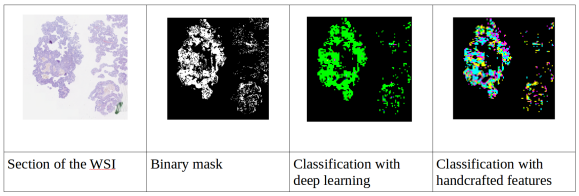

Figure 14: classification with both approaches, section (level 1)

C. *Redundancy of features*

As a last step, we want to check if any of the handcrafted "Best5" features we extracted are redundant. To do that we compute the feature correlation on the WSI (level 3) and the section (level 1) and display the results in a heat map (Figures 15,16). The features are labeled 1 to 74 in the order introduced in Task 2.

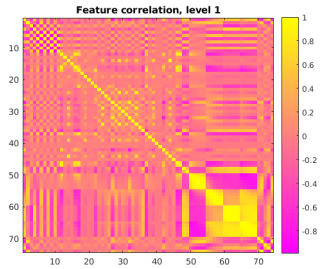

Figure 15: Feature correlation, level 1

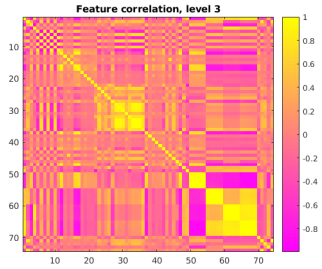

Figure 16: Feature correlation, level 3

We see that between the five different feature groups almost no correlation is observed, whereas within several feature groups some features are highly correlated (for example histogram lower, GLCM, parts of LBP on level 3). These results are widely consistent with the correlation analysis done in the paper. It suggests that the feature groups could possibly be reduced by some features.