CNN for automatic process supervision

Ertugrul Furkan Düzenli
Laboratory for Process Automated Systems
TU Dortmund university
Dortmund, Germany
furkan.duezenli@tu-dortmund.de

Marvin Daney Schwing
Laboratory for Process Automated Systems
TU Dortmund university
Dortmund, Germany
marvin.schwing@tu-dortmund.de

***Abstract*—For the operation of an extraction column it is crucial to know the state of the system. For this task we have used convolutional neural networks, transfer learning, machine learning with random forest classification by using Gabor and Sobel filters and tried a mathematical approach to distinguish between the flooded and normal state. For all methods we calculated the accuracy, recall and precision and compared the results. Further we discussed the drawbacks and advantages, and it was found out that all the methods except the mathematical approach were competitive to each other.**

I. Introduction

The extraction column can be considered as an important unit process in the chemical industry which enables one component to be extracted out by using a solvent that can be dissolved in the main solution. So, for an optimal operation care should be taken to neither use too much solvent nor to use an insufficient amount which would not yield an efficient separation process. Therefore, it can be differentiated between the undesirable flooding and desirable regular states in its operation. However, sometimes it is hard to distinguish between these two states because of the similarity of their appearance. Therefore, efforts have been done to develop an image recognition system to be able to distinguish between these two states in order to ensure an optimal operation.

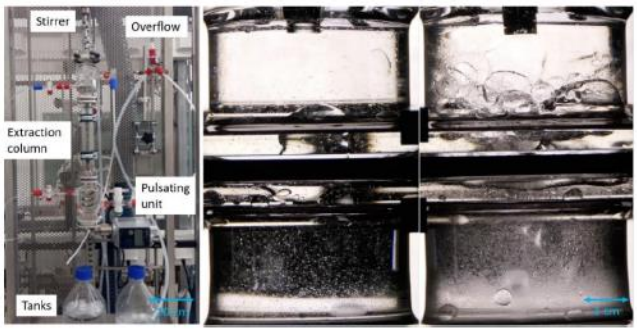

Figure 1: Extraction column (left image) with desired operating state "normal" (central image) and undesired flooding state (right image).

The motivation of this project was to use an AI, Machine Learning, and a mathematical approach to be able to identify the two different operational states and to make a comparison between their advantages and disadvantages. Additionally, to improve the performance, we implemented data augmentation for the AI-based methods.

The following table illustrates the methods which were governed in order to identify between the two states.

Table 1: Illustration of different methods that were used for image classification

AI Method		**Machine Learning**	**Mathematical approach**
CNN own net	Transfer learning with VGG16	Random Forest Model with Gabor and Sobel Filters	Bubble detection with circle Hough Transform

After having implemented the aforementioned methods, metrics which quantify the results were obtained as the accuracy, precision, and recall on the test data set. Consecutively, efforts have been done to evaluate the results on a sample image by using the models of the methods by stating its model prediction and compare it to the true state namely regular or flooding.

II. Image Classification using CNN

A. Data Preprocessing

The first steps in every Neural Network task are the collection of data, preprocessing of the data and arrange the data into a readable format for the "model.fit" command. For this task there are several usable toolboxes which all have their benefits and drawbacks. For the own net, we decided to safe the image data in a numpy array. The strongest argument for this decision is the speed of the calculations. We compared the Keras command "image_dataset_from_directory" with a numpy approach. With the command it is possible to load data directly in the right format and use it right away for the model. But it is necessary to preprocess the images by its function variables. First, the whole picture must be resized to 120x80 pixel and in the model the first line should be "AveragePooling2D" or "MaxPooling2D" to condense several pixel values into one to reduce the output shape of the layer. Further we have assumed, that colors are not needed and loaded greyscale images. With these operations we are down to 2.514 trainable parameters and each epoch takes around 154 s. A similar model with numpy data is a lot faster in the calculations but takes more time in the preprocessing. First it is not possible to load the whole dataset with the native resolution without overfilling the memory, because there the whole data is stored. The solution is to reduce the image size and convert the RGB data into grayscale. We also decide to focus on a smaller area in the bottom half of the images. The result of the preprocessing is shown in Fig. 2. With this processed data we have trained our model. The model differs from the previous model. It does not have the "AveragePooling2D" command, the input size of the images is 200x100 and the kernel size of the one and only convolutional layer is smaller. This results in a model with 640.274 trainable parameters. Even though we increase the number of parameters by more than 250-times the calculation

of each epoch is reduced to 6 seconds. This time improvement enables the effective search for better hyperparameters and model layouts. For this unoptimized model we got an accuracy of 85 %, a recall of 100 % and a precision of 73.5 % after five epochs.

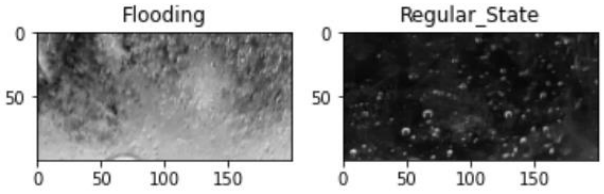

Figure 2: Cropped flooded and regular state in grayscale.

B. *Model optimisation*

For the optimization we used the Keras Tuner [1]. With this tool it is possible to try different parameter combinations without the need to create a new model every time. It is also possible to determine the epoch number and the executions per trial. Both parameters were set to two to reduce calculation time. That means for each parameter combination the Keras Tuner will calculate two epochs two times to incorporate the randomness in the weights. It is also possible to get the best overall model and the best hyperparameters with respect to the validation accuracy and epoch. In the first step we vary the filter-size and the learning rate. The highest validation accuracy we got with 24 filters and a learning rate of 0.001. With the "get_best_models()" command we got a model which had an accuracy of 94 %, a recall of 100 % and a precision of 89.7 % without optimizing the epoch. We assume the best model parameters were taken after the second epoch but cannot verify that. For a fair comparison with the first simple model, we create a new model with the optimized parameters and set the epoch to five. After the calculations we got an accuracy of 83.8 %, a recall of 100 % and a precision of 76.2 % which are like the previous numbers. This implies that we have overfitted our model which can be seen in Fig. 3. The validation loss is minimal after two epochs which explains the great results of the Keras Tuner best model. The next step is the Kernel size optimization. We vary between the kernel size of three to eight and let the Keras Tuner find the optimal value which was four. The model does not change because four was the initial value. In the next optimization we add a convolutional layer and again vary the filter size. The optimal filter size was 40 but the "get_best_models()" command return a model which was inferior to the previous. There are several possible reasons for this behavior. One of the two reasons we found was the epoch size. With an additional CNN layer there is much more to calculate, and the number of parameters massively increase. The second reason could be the lack of a pooling layer. With such a layer the model should have it easier to highlight important features for the state identification. It also reduces the number of trainable parameters with the result of a shorter calculation time. With these two additions we get an optimal epoch size of around 9 (Fig. 4). To get the best possible model out of our optimization we use the Keras Tuner without varying any data. The epoch is set to ten and the number of executions is two. The best model out of the Tuner lead to an accuracy of 96 %, a recall of 100 % and a precision of 92.7 %.

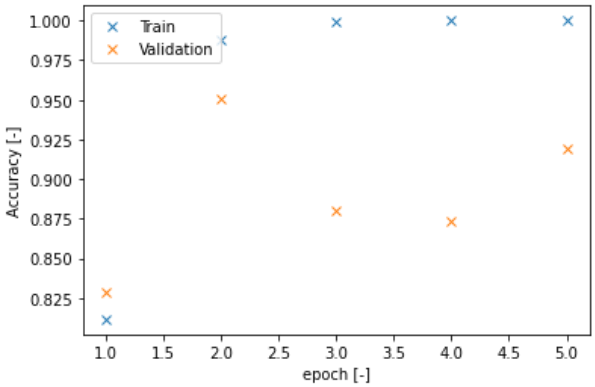

Figure 3: Plot of the train and validation accuracy over the number of epochs.

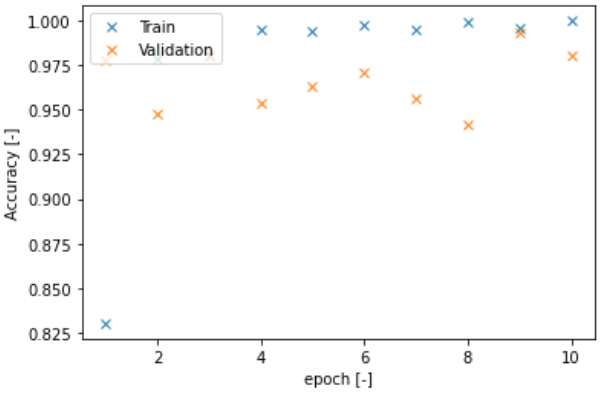

Figure 4: Plot of the train and validation accuracy over the number of epochs of the optimized model.

C. *Data Augmentation*

To further improve the model, we tried data augmentation [2]. In short it is a way to generate more data out of the existing images without the need of additional ones. For this task we use the ImageDataGenerator(). It takes the existing pictures and has a variety of methods to generate new pictures. Rotating, stretching, or zooming in are only some of the possible methods. The data generator is then used for the training dataset which will increase the overall number of images to train with. The optimal epoch size will increase. Therefore, we tried an epoch of 50 as our first guess. In Fig. 4 the validation loss is oscillating around 0.5 and there is, despite some spikes, no indication for overfitting. We conclude that the epoch is still too small and run the algorithm with an epoch of 150. Fig. 5 shows similar results with growing spikes. Because there was very little improvement, we decide to find the best model within the 50 epochs. We set up the Keras Tuner and evaluate the model. The model has an accuracy of 97.5 %, a recall of 100 % and a precision of 95.4 %. So, it was indeed possible to improve the model with the help of data augmentation.

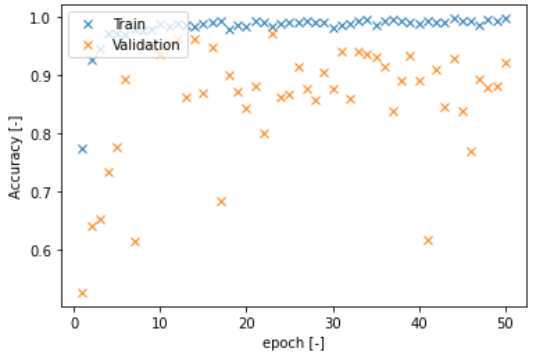

Figure 5: Plot of training and validation accuracy over the epoch size.

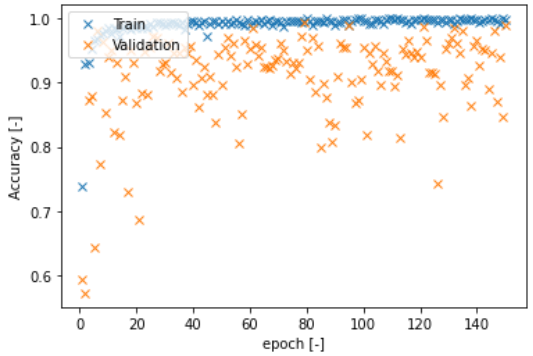

Figure 6: Plot of training and validation accuracy over the epoch size.

III. IMAGE CLASSIFICATION BY USING A PRETRAINED CNN

A. *Data Preprocessing*

After having implemented the necessary TensorFlow libraries the VGG16 model was chosen as the pretrained CNN. The aim of using a pretrained CNN was to achieve an overall better performance on the test dataset and keep the computational effort for training and building a neural network architecture from scratch low. With only a trivial replacement of the prediction layer of VGG16 to fit it to a binary classification problem with two outputs and keeping the previously trained weights unchanged, the model was able to being fitted. After doing necessary arrangements 8,194 trainable parameters were yield. Consecutively, the definition of the loss function to be binary cross entropy and optimizer to be Adam were chosen for the model compilation. A reasonably small value of 0.01 for the learning rate was selected. After that, the preprocessing and rescaling of the pixel values of the images were done by using preprocessing functions of TensorFlow's Keras library. The images were imported by using the "flow_from_directory" function. At this stage it was crucial to import the images in the suitable format in which the VGG16 model was able to work. So, the color mode was set to be RGB and the dimensions of the images were set to be (224,224). Also shuffling has been included for the training data set to allow randomization. The class mode was selected to be categorical as it was required for the VGG16 input format. Afterwards, model fitting with an early stopping criterion was done to finish training after a non-incremental validation loss difference of 0.001 for at most three consecutive epochs. The maximum number of epochs was set to be ten. The dataset consists of 5158 images from which 70 % of them were used for training and 30 % for testing.

B. *Model Analysis and Evaluation*

Promising results were obtained by the transfer learning module. It can be seen from Fig. 7 that after the first epoch a validation accuracy of 100 % was reached whereas the training accuracy picked up slowly, starting from 87 %.

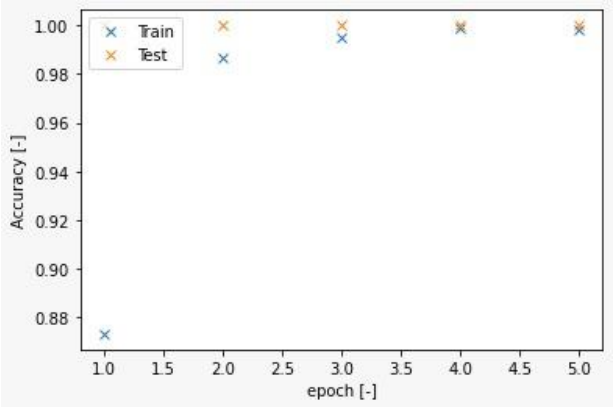

Figure 7: Plot of training and test accuracy over the epoch size.

The early stopping criterion was fulfilled after the fifth epoch, where the validation loss difference between three epochs was smaller than 0.001 as seen in Fig. 8. Apart from that, a precision and recall of 100 % was obtained for both the flooding and regular state.

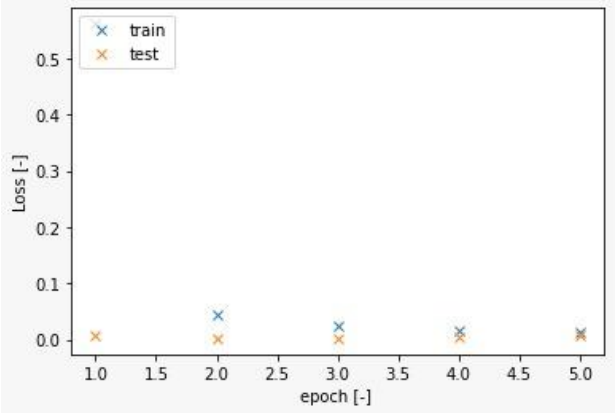

Figure 8: Plot of training and test loss over the epoch size.

C. *Data Augmentation for the pretrained Model*

The data augmentation was performed with the Keras function ImageDataGenerator, the rotation range was set to ten and the images and the zoom range to 0.2, thereby the data was amplified.

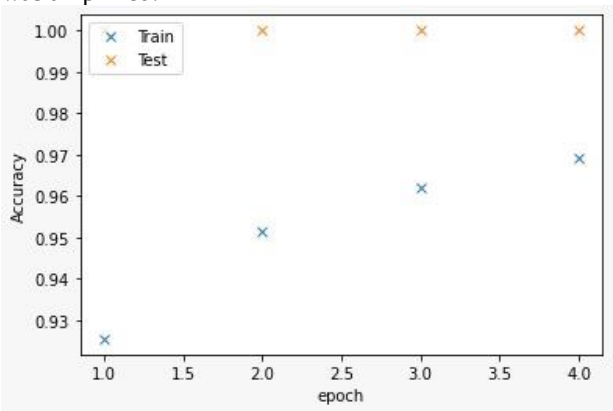

Figure 9: Plot of the training and test accuracy over the epoch size.

As a result, after the first three epochs the early stopping criterion was fulfilled with a minimum validation loss difference for at most three consecutive epochs whereas the training accuracy had still a bit more potential to grow. The precision and recall of both the flooding and regular state were obtained as 100 %. The results are shown in Fig. 9 and 10.

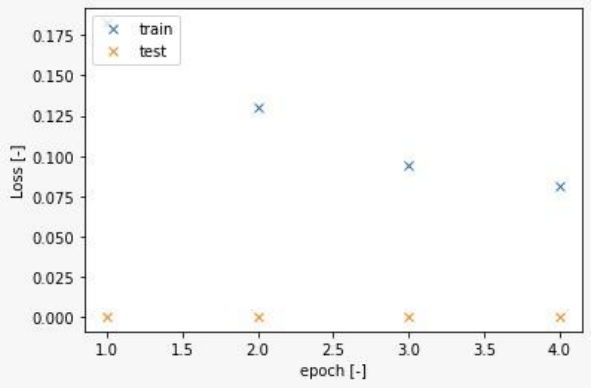

Figure 10: Plot of the training and test loss over the epoch size.

IV. Classification with a Machine Learning Method

As another approach, the classification of the regular and flooding states was done with a machine learning method called the random forest model by using Gabor and Sobel filters and the pixel values for feature extraction [3]. Gabor filters are particularly used for texture analysis where it observes specific frequency contents in regions of interest to determine texture representation and discrimination whereas Sobel filters are used for an edge detection which is a computationally cheap option to perform calculations. [4] [5]

A. Preprocessing of Images

The preprocessing steps of the images were done by creating two folders of training and testing where regular and flooding state pictures were divided into a 70 % training and 30 % test ratio. Afterwards images were imported and then converted to numpy arrays to enable faster computation. The advantage of the machine learning method compared to deep learning approaches was that it could handle even very big data files, whereas by deep learning approaches after conversion to numpy arrays, a memory error could happen due to the fact that TensorFlow functions might not handle large dataset files during the model fitting. But this happens only if images are converted to numpy arrays. In the machine learning case, there is not this kind of problem, therefore it could be benefitted from the conversion of the images into numpy arrays. Another advantage compared to the VGG16 model was to be able to resize the images to a desired size and also to change the color mode into grayscale which additionally have dropped the computational effort immensely. Additionally, with the aid of the sklearn prepocessing library, it was enabled to encode labels from text (folder names) to integers, and the pixel values were normalized between 0 and 1. Finally the function definition for the feature extraction was done by creating an empty pandas dataframe and allowing to adding up features as desired. The pixel values, Gabor and Sobel filter responses were used as the features for the pandas dataframe. Moreover, a for loop was created to change the θ parameter of the Gabor filter to allow for variability of filter responses whereas λ, α, σ were kept constant.

B. Model Fitting and best Parameter Estimation with GridSearchCV

The model fitting occurred with the random forest function. As the random forest model was prone to overfitting only the variable n_estimator was chosen for the random forest classifier and GridSearchCV was used for the optimal selection of the best value which was determined to 50. The random_state parameter of GridSearchCV was kept by 42 to avoid variability of the random forest function predictions. This allows reproducible parameter estimations of GridSearchCV.

C. Model Analysis and Evaluation

Surprisingly, the random forest model has shown itself as a competitive approach instead of deep learning methods. The following Fig. 11 demonstrates the precision and recall values.

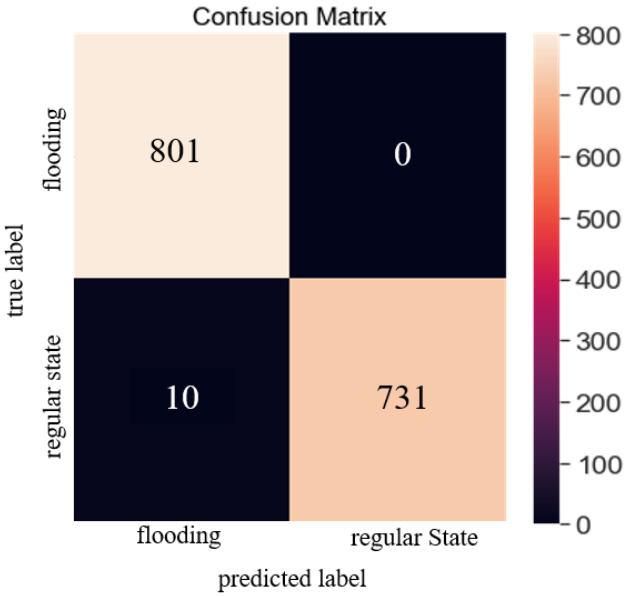

Figure 11: Confusion matrix of the flooding state and the regular state.

As a result, for flooding, a precision of 0.99 and 1.00 for recall, for the regular state, a precision of 1.00 and recall of 0.99 was obtained. For the accuracy we got 99.3 %. As can be seen from Fig. 10 only ten images have been misclassified as flooding where the true label is regular. Also, for demonstrative purposes the image of the resized extraction column is shown below.

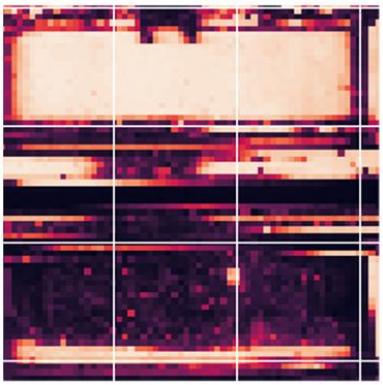

Figure 12: Preprocessed image by Gabor and Sobel filters of the regular state with 64 by 64 pixels.

It shows that the extraction column state could be barely distinguished by human vision. However, the algorithm can handle the image resizing and can perform a correct classification.

V. Classification with a Mathematical Approach

The idea that we want to implement is to count the dark pixels in an image and correlate it to one of the two states. As to be seen in Fig. 2 there are much more bubbles in the flooded picture. That is the reason that the number of dark pixels should be higher in the normal state images. Unfortunate that is not true for all the data. As shown in Fig. 13 there are images with a huge number of dark pixels but belong to the flooded category. Additionally, there are images with less dark pixels but are still in a normal state. That leads to an insufficient classification.

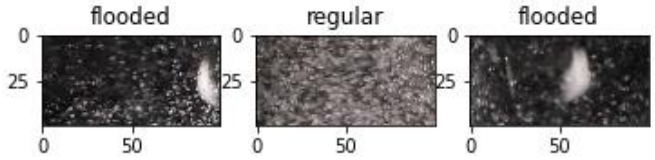

Figure 13: Exemplary images of two flooded pictures and one regular state.

Another idea is to detect the bubbles and count them. For this approach preprocessing of the images is required. A possible way is to use the scharr filter from skimage.filters. It is an edge detector and lead to Fig. 14. To remove some of the noise we programmed a simple algorithm which set the bright pixels to white and the darker pixels to black. This process leads to Fig. 15. The lower half of the column is still noisy but irrelevant because the bubble detection will only take place in the half with more darker pixels. The result of the circle detection is shown in Fig. 16. If the detector counts one or more bubbles the state is set to flooded, otherwise the image is classified as a regular state image. With that split we get an accuracy of 88 %, a recall of 76.8 % and a precision of 100 %.

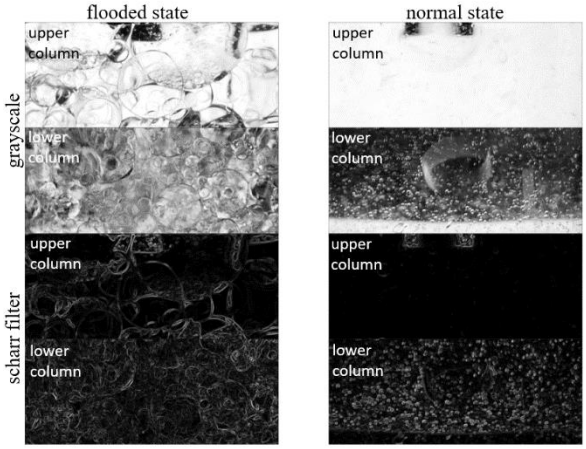

Figure 14: Grayscale and scharr filter images of the extraction column.

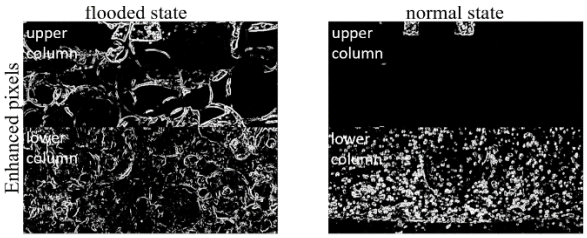

Figure 15: Enhanced pixels of the scharr filter images.

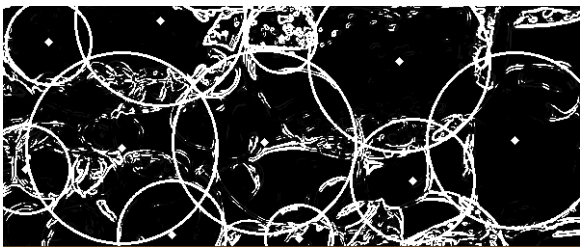

Figure 16: Bubble detection on the flooded state.

VI. Conclusion and Future Remarks

In summary we have tried six different approaches to classify the states of the extraction column. The mathematical approach with the circle detection has the worst metrics as seen in Table 2. For the recall it only achieves 76.8 % which means that the method misclassified almost a quarter of the flooded images. In an actual industrial process that would lead to an insufficient operating column which causes a lower product yield. Our own CNN has an overall good performance and data augmentation was able to improve the model even more. When training with data augmentation it is important to increase the epoch size since the amount of data rises. It was seen that by using the transfer learning method, data augmentation leads to nearly the same performance. There is only a slight difference in the training accuracy and the required epoch size which are lower with data augmentation. The Random Forest classification was also an excellent performing method. Overall, it has achieved a great run time performance after execution of the script which makes it attractive to use.

Table 2: Summary of the metrics of all used methods of the test dataset.

Method	Accuracy	Recall	Precision
Optimized CNN own net	96 %	100 %	92.7 %
Optimized CNN own net with data augmentation	97.5 %	100 %	95.4 %
Transfer learning with VGG16	100 %	100 %	100 %
Transfer learning with VGG16 with data augmentation	100 %	100 %	100 %
Random Forest Model with Gabor and Sobel Filters	99.3 %	100 %	99 %
Bubble detection with circle Hough Transform	88 %	76.8 %	100 %

The mathematical methods with the circle hough transform are in this form not advisable. It is required to improve the image preprocessing and optimize the parameters of the HoughCircles function. To improve the own net to the level of the transfer learning model further optimizations or increasing the layer depth is needed. The effort to improve the own net is not worthwhile, because we already achieved an accuracy of 100 % with the transfer learning approach with a reasonable amount of work.
As a future work localizing the flooding area and detecting the dimensions of all the bubbles is possible. With this additional information it could be possible to estimate the future states and set up a controller for automated process supervision.

VII. REFERENCES

[1] O' Malley, Tom and Bursztein, Elie and Long, James and Chollet, Francois and Jin, Haifeng and Invernizzi, Luca and others, "Keras Tuner", in https://github.com/keras-team/keras-tuner, 2019.

[2] Chollet, F. & others, "Keras Image data preprocessing", in https://keras.io/api/preprocessing/image/, 2015.

[3] Dr.Sreenivas Bhattiprolu, https://github.com/bnsreenu, 2020

[4] https://en.wikipedia.org/wiki/Gabor_filter

[5] https://de.wikipedia.org/wiki/Sobel-Operator