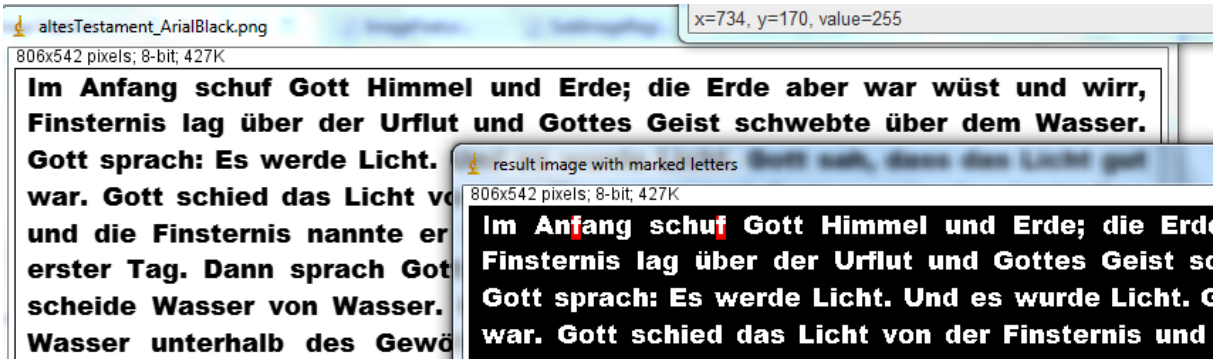
BVA2 – **Exercise 06:** ***Pattern Recognition - OCR***

[18 points]

Application of OCR pattern recognition to mark target letter in a text

Implement an ImageJ plugin to mark all occurrences of a target letter within an input text provided as grayscale image as illustrated below:



Example with reference character set to row=1, column=5, i.e. “f”. Some/most occurrences of “f” are marked in red as final result.

1. Basic Implementation [14 points]

For implementation of this exercise utilized the provided template files and implement the defined interfaces. The following steps need to be implemented. Always evaluate all of the achieved intermediate results:

- At first the input image needs to be binarized utilizing a proper threshold value.
- Separate the input image horizontally by splitting into sub-images that are containing the letters, i.e. lines. Then further separate the lines by vertical splitting to separate into character/letter images. Therefore, the sub-image data structure (`class SubImageRegion`) needs to be utilized. To separate the characters, the method `splitCharacters` needs to be implemented utilizing “fire-through” strategy. As an alternative, common region growing could be applied too (*cons: no ordering of letters in rows and columns, consider problems related to “i”,…*).
  ```
  public Vector<Vector<SubImageRegion>> splitCharacters(int[][] inImg, int width, int height, int BG_val, int FG_val)
  public Vector<SubImageRegion> splitCharactersVertically(SubImageRegion rowImage, int BG_val, int FG_val, int[][] origImg)
  ```
- Define the reference character (user input) and calculate the feature vector: `double[] calcFeatureArr(SubImageRegion region, int FGval, Vector<ImageFeatureBase> featuresToUse)`
  All the utilized features are derived from abstract base class `abstract class ImageFeatureBase` and necessitate implementation of the evaluation function `public abstract double CalcFeatureVal(SubImageRegion imgRegion, int FG_val)`. Implement at least all of the pre-defined features.
- For normalization, the mean feature values need to be calculated utilizing all characters in the provided text image:
  ```
  double[] normArr = calculateNormArr(splittedChars, BG_VAL, featureVect);
  ```

- Finally, classify all characters that show some level of similarity compared to the provided reference character. For comparison of the feature vectors, utilize correlation coefficient calculation and normalization.
- All letters that show a correlation coefficient above a certain confidence level are marked in the final result image. To allow for good results due to parameter tuning, the correlation coefficient threshold should be defined/adapted by the user.

Extensively test your implementation and evaluate the classification accuracy.

- Which letters can be detected in a confident way and which letters lead to problems – and why?
- Are all fonts applicable to this kind of OCR strategy – why or why not?
- Does classification accuracy depend on the other characters in the specific line – if that's the case: why and how to overcome this issue?

2. Advanced Implementation [4 points]

- Implement at least 3 additional features that improve robustness of the basic implementation.
- Ensure that the split characters image region is shrinked to its bounding box. How can that help to improve result quality?
- Discuss the normalization process – how does character occurrence probability influence the results?
- Discuss how the classification process itself could be improved. Are there better strategies for feature-based classification?
- How does correlation of some of the features itself influence the achievable classification results (*e.g.* ***max-distance-from-centroid*** *will somehow be correlated to the specific* ***width***)?