ARCHIVES
of
FOUNDRY ENGINEERING

ISSN (2299-2944)
Volume 2025
Issue 2/2025

173 – 182

19/2

10.24425/afe.2025.153807

Published quarterly as the organ of the Foundry Commission of the Polish Academy of Sciences

Analysis of the Possibility of Using Selected Artificial Intelligence Algorithms for the Assessment of the Microstructure of Vermicular Cast Iron

U. Janiszewska [a], Ł. Marcjan [a], S. Gajoch [a], K. Jaśkowiec [b], A. Bitka [b], M. Małysza [a], D. Wilk-Kołodziejczyk * [a, b]

[a] AGH University of Krakow, Faculty of Metals Engineering and Industrial Computer Science, Poland
[b] Łukasiewicz Research Network-Krakow Institute of Technology, Poland
* Corresponding author: E-mail address: dwilk@agh.edu.pl

Received 01.12.2024; accepted in revised form 12.03.2025; available online 30.06.2025

Abstract

This paper presents an analysis of artificial intelligence algorithms in the context of their applicability to the automatic analysis of microstructure images. In the example presented, reference is made to exemplary images of the microstructure of vermicular cast iron. A characteristic feature of this alloy is the shape of the graphite separations. The microstructure consists of elements that humans can learn to recognise quite simply. Developing an application that recognises 'dark colour' and 'worm shape' is no longer so straightforward. The determination of the 'dark' colour in the algorithm becomes problematic, because depending on the conditions under which the photo was taken (e.g. time: day/night), the actual intensity values are altered. A similar situation occurs in the determination of shape, which varies from case to case. Such a classification is very general and results in large differences between instances of the class. Even a term like 'relatively large' can change depending on the size of the graphite separation itself. A dark colour can be represented as a sudden change in image intensity, i.e. large values of the gradient modulus. The question arises: what happens if 'dark' can be more than one microcomponent, for example graphite and perlite. A good solution would be to define an associated set of features that would more precisely define just this component of the microstructure - that is, its shape, colour and surroundings. The paper uses the local feature paradigm to do this. Referring to the literature, it can be pointed out that [1] local features are referred to as non-small and specific parts of an image. Distinctive image features need to be distinguished in order to detect these places of interest. In this case, they are: edges, spots and ridges.

Keywords: Microstructure images, Vermicular cast iron, Artificial intelligence HOG algorithm

1. Introduction

Searching microstructure elements or other local features is a current research topic, especially in the context of using artificial intelligence methods. Creating a new method of representing images using numerical vectors based on the bag of features concept was indicated in literature, especially in [2]. Images are divided and converted into visual words, in order to be able to use them later in the image identification process. Works focusing on local features of images. Feature extraction is performed, which

means that characteristic places in the image are determined. Features are described using descriptors (SIFT and SURF algorithm). In the final phase, you can compare key points and recognize whether there is an interesting element in the image. An extension of the bag of features model is the spatial pyramidal representation, which takes into account the spatial arrangement of words in the image. This allows for a better description of the structure and relationships between image elements. Initially, hierarchical image partitioning is performed, then the bag of features for each window, histograms are combined and compared. This algorithm takes into account spatial information in the image representation, allowing the detection of certain regularities in the image composition, which is useful in classification tasks. A more efficient algorithm for constructing a pyramidal representation was implemented, using binary tree structures and arrays. This makes the algorithm more efficient than a naive algorithm that would check the windows in order. For each visual word in the image, a histogram is computed in such a way that it is not necessary to check all the windows. The code contains operations that define a window with a given visual word and calculate the offset in the histogram for that word. The algorithm uses binary trees containing edges of all windows on a given level of the pyramid. By selecting the appropriate edges, one can effectively identify windows containing a given visual word. Another example is the automatic analysis of microstructures [3], in which the authors use the backpropagation algorithm to train an artificial neural network. For this purpose, images of materials such as ductile, gray and malleable cast iron were used to segment and quantify various microstructures in metallographic and microphotographic images. In this case, segmentation includes dividing images into specific microstructure components, i.e.: pearlite, cementite, graphite or martensite. The multilayer perception network belonging to the feedforward network was used. This type of network consists of multiple layers, including one or more hidden layers that process input data and lead to an output layer that gives a solution. The neural network used in the work consists of 39 neurons, distributed over two layers: an input layer with three inputs (colors) and a hidden layer with 30 neurons and an output layer with nine neurons, corresponding to nine types of microstructures to be recognized. The training process consisted of manual selection of pixels from the micrographic image. Separate network training was performed for each type of material to be analyzed. Each pixel of the input image is classified based on colors (RGB) and assigned to the appropriate element of the microstructure using the neural network.

Methods for microstructural image segmentation, particularly for complex textures used in metallographic analysis, have been described in research studies. Traditional segmentation techniques often struggle with such images due to their rich details and irregular structures. Article [4] presents a hybrid algorithm combining multiple methods: Gaussian filter, mean shift method, FloodFill algorithm, and an enhanced flow-based difference-of-Gaussians algorithm. This approach enables more effective segmentation of microstructural elements in metallographic images. Experiments demonstrated higher effectiveness of this method compared to existing segmentation techniques, particularly for complex texture images with a limited number of samples. Article [5] introduces a metal microstructure analysis method that integrates deep learning with traditional image segmentation techniques. The main objective of the study was to develop a hybrid approach (HADMA) to improve the accuracy and repeatability of microstructural analysis. A boundary class semantic segmentation (BCSS) method was proposed, allowing for precise differentiation of grain and phase boundaries in the material. The BCSS method was then combined with the Watershed algorithm, enabling a more accurate determination of material structure. The method was tested on microstructure images of the Ti6Al4V alloy, obtained using a scanning electron microscope (SEM), achieving high accuracy compared to standard manual methods. The results highlight the potential of deep learning in automating microstructural analysis, which can significantly improve quality control and materials research processes. Automated structure identification plays a crucial role in the control and diagnostics of melting and casting processes for cast iron materials. This approach offers several advantages over traditional methods, enhancing efficiency, accuracy, and consistency.

1.1. Descriptors in Image Analysis

The methods of identifying vermicular graphite in the microstructure are known, but the problem remains the tools that will allow for finding microstructural features in practice. The solution may be the use of descriptors, i.e. tools used in image analysis to describe important features of objects and image fragments. Image descriptors are a key element of the image analysis and feature detection process [8]. As mentioned in [2], they are numerical representations of specific features of objects in images, which allows for more precise comparison and identification of these objects. Recognition of the structure of vermicular cast iron using descriptors plays an important role in the extraction of characteristic microstructural features. In the case of analysis of the structure of vermicular cast iron, local feature descriptors are crucial. Local features focus on the description of small, characteristic image fragments that are uniquely localized. In the case of the microstructure of vermicular cast iron, local features may include areas containing separated graphite, ferrite areas, or pearlite stripes. Local feature descriptors allow for the extraction of information from these specific fragments, which is crucial in the recognition and analysis of this structure.

Resistance to transformations: Local features are often resistant to various types of image transformations, such as changing the scale, rotation, or translation [3]. The study of the microstructure of cast iron is based on images that can be collected under different conditions. Local feature descriptors allow for consistency in feature extraction, regardless of these transformations. Precise matching: Local feature descriptors allow for precise matching of features between different images. In the case of the project on the microstructure of cast iron, this allows for the identification and comparison of different instances of this structure in the images.

1.2. Selecting Appropriate Descriptors for the Microstructure of Vermicular Cast Iron

Due to the microstructure of vermicular cast iron, where graphite is characterized by vermicular (worm-like) graphite particles, which differ significantly from flake graphite in gray cast iron and spheroidal graphite in ductile iron, this material finds wide industrial applications. It is particularly used in components requiring a combination of high mechanical strength with excellent resistance to thermal and impact fatigue. The unique morphology of graphite with complex shapes presents a challenge in microstructure identification, especially compared to flake and spheroidal graphite analysis. The shape of graphite precipitates contributes to an optimal balance between ductility and strength, making this material highly attractive for modern engineering applications [6,7].

The key step in effectively recognizing the microstructure of cast iron in images is to select appropriate descriptors. Descriptors are the most important part of the image analysis process, because they allow the extraction of important information from images and their representation in the form of numerical data. In the case of cast iron microstructure, there are many features that can be important for recognition and analysis [8]. It is important to select descriptors that are able to capture these features effectively and reliably. In order to find interesting features of the images for vermicular cast iron, local feature descriptors were used. The literature analysis shows that they are insensitive to image changes. Therefore, the size or the way of lighting should not affect the searched characteristic feature. The focus was on two popular descriptors in the field of image processing, such as SIFT and HOG. The decision was made based on the work [9] where it was indicated that SITF (Descriptor Scale Invariant Feature Transform) works well in identifying similar images or their fragments, is insensitive to changes in contrast or brightness. The descriptor analyzes the orientation of the gradient calculated in the extrema of the Laplacian function with Gauss. At the key point for the image, a vector of local features is calculated. The first step is to search for extrema in scale and space. For this purpose, a convolution function is used, which uses the image brightness function and the Gaussian function, taking the scale as an argument. Increasing the scale value leads to a greater blurring of the image. In this way, the images are repeatedly blurred. Their groups form octaves. Between octaves, the image resolution is reduced by a factor of two. The DoG function of the difference for two scale values is used to detect key points. Potential key points, or extrema, are selected by comparing neighboring pixels in successive scales. To confirm whether a given extremum is indeed a key point, the Taylor formula is applied, performing differentiation and transformations, which allows to determine the exact location of the local extremum. Finally, only points with sufficiently high contrast that do not lie on the same line are retained. In this process, the Hessian (H) and the curvature coefficient are used. Rotation resistance is obtained by calculating the gradient and rotation for the neighborhood of the key point. The last part of the SIFT descriptor consists of a vector of 128 values of the orientation histogram, obtained by analyzing the gradient coefficient in a 16x16 neighborhood of the keypoint. The gradients are weighted with a Gaussian filter with a scale equal to half the image scale. The vector is then normalized to ensure robustness to changes in the object's brightness and contrast. Using the SIFT descriptor requires a significant amount of computation to be able to calculate even the keypoints themselves. The pool of existing descriptors is relatively large, because it gives 255128 possible feature values, which comes down to creating complex algorithms that could compare images. Due to its high computational cost, this descriptor is not the best choice for large databases, especially in systems where time is a key factor.

2. SIFT Implementation and Results

The application verifying the assumptions was written in Python. OpenCV libraries were imported to use the SIFT method and Numpy, which allows for manipulation of microstructure images. After reading the microstructure image from the file, a SIFT algorithm object is created, which finds key points and their corresponding descriptors in the microstructure image. The function from the OpenCV library is used for this purpose: cv2.SIFT_create(). The microstructure image is reduced, and then key points are plotted on it using red dots. Then, for each point, the orientation (green line indicating the direction) and the blur scale (blue circles) are marked (Figure 2, subpoints b, d). Orientation helps find the direction of the “greatest uniqueness” of a given area, and the blur scale determines how large a given structure is around the selected feature point. Finally, the image with the changes plotted is displayed.

```
Histogram orientacji:
[  1.   0.   0.   0.   0.   0.   1.   3.   1.   0.   0.   0.   0.   1.
   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.  18.   0.   0.   1.   4.  43.  70. 146.  49.   5.
   0.   0.  21. 129.  30.   9. 152.  22.   0.   0.   1.   4.   1.   8.
  56.   7.   0.   0.   0.   0.   0.   0.  43.  13.  66.  35.  27.  30.
  30. 112.  50.  10.  27.  81. 120.  94.  12.  13. 152.  19.   2.   3.
   6.   6.   1.  36.  92.   6.   0.   0.   0.   0.   0.   8.   7.  21.
 152.  45.   2.   0.   0.   2.  88.  48. 151.  88.   9.   0.   0.  19.
 152.  30.   5.   2.   0.   0.   0.  45.  65.   0.   0.   0.   0.   0.
   1.  17.]
Ramka (bounding box): [  4.7271862  736.30633545   4.93487263   4.93487263]
```

Fig. 1. Orientation Histogram with Frame

Figure 1 shows the orientation histogram for the first keypoint with a bounding box defining the area in the image where this descriptor was calculated. The values of the width and height of the box are around 4.93, which indicates that the analyzed area is small. The center position of the box is (4.73, 736.31), which defines where the SIFT algorithm located the first keypoint.

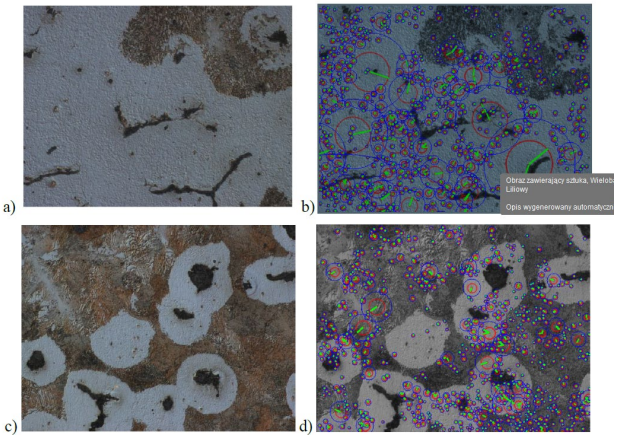

Fig. 2. Microstructure images A (a) and C (c) with plotted changes (b, d): key points (red circles), orientation (green line indicating the direction) and blur scale (blue circles)

Figure 2 shows two images of microstructures with the corresponding result images after passing the microstructures to the SIFT descriptor calculation program. The marked key points are important from the analysis point of view, they have a particular feature that is important for the structure being studied. The orientation indicates the direction of the preferred orientation of the structures in the material, this is important when the material properties change depending on the direction. The blur scale indicates how far from a given key point or structure area the influence of a given phenomenon can be expected.

2.1. HOG algorithm

After enabling libraries (cv2, skimage.feature with HOG function) and loading the microstructure image using the hog function, descriptors are calculated (code fragment 1). The appropriate parameters are selected:

```
# Oblicz deskryptory HOG
features, hog_image = hog(image_microstructure, orientations=8,
                          pixels_per_cell=(16, 16),
                          cells_per_block=(1, 1),
                          visualize=True)
```

Code Fragment 1: Calling the Built-in HOG Function with the Parameters Used

In code snippet 1, the orientations parameter corresponds to the number of gradient directions in one block, pixels_per_cell specifies the number of pixels in one cell, cells_per_block the number of cells in one block, and visualize=True returns the HOG visualization image along with descriptors. How scikit-image HOG works:

Global Normalization (optional):

Reduce the impact of lighting effects with global normalization, for example through gamma compression. Image Gradients (First Row): Calculate image gradients in x and y directions, capturing contour, silhouette, and texture information. Gradient Directions (Second Row): Create local histograms of gradient directions for each cell, resulting in an "orientation histogram". The orientations=8 parameter specifies how many directions to divide the gradient angle range in one cell.

Local Normalization (Third Order):

Normalization of gradients in local groups of cells - blocks which improves robustness to lighting, shadow and contrast variables. The parameters pixels_per_cell=(16, 16) and cells_per_block=(1, 1) define the number of pixels in one cell and the number of cells in one block, respectively. Collecting HOG Descriptors (Final Step): Collecting HOG descriptors from all blocks in a dense, overlapping grid of blocks to create a joint feature vector for window classification.

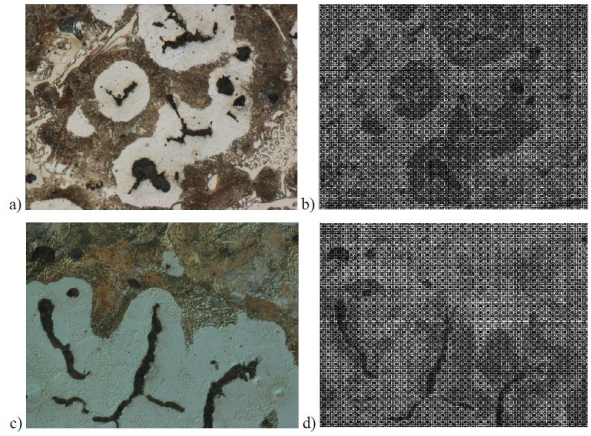

Fig. 3. Images of microstructures (a, c) with HOG descriptors (b, d) calculated for them

Figure 3 shows the results of the program calculating the HOG descriptor for two microstructure images (a, c). The resulting images (b, d) show distinct precipitates of vermicular cast iron. They are darker in color and have a worm-like shape, which makes them clearly visible against the background of other microstructures.

2.2. Application of image processing techniques for edge analysis and detection

The use of gradient proved to be an effective solution for detecting vermicular cast iron, which distinguishes it from other microstructures. For this reason, the image intensity gradient was used in the contour search algorithm.

Method 1

The first program uses image analysis, focusing on low light areas around the largest closed contours. After loading the image, it is converted to grayscale using the built-in OpenCV cvtColor function. Then, the image is smoothed using a Gaussian filter to reduce noise. The next step is edge detection using the Canny operator. After that, a morphological closure operation is applied to connect the separated edge regions. Then, contours are found again and filtered to leave only those that are closed (have more than 4 points). Closed contours are sorted by their area in descending order. Finally, only 10% of the largest contours are selected from the sorted ones. Then, the low light areas inside the

contours are analyzed: masks of low light areas inside the selected contours are created and contours in low light areas are found. The last step is to leave only 0.5% of the largest contours and draw them on the output image.

Convert to Grayscale:

After loading the image, the image is converted to grayscale using the cvtColor function from the OpenCV library [10]. It takes two arguments: the image and cv2.COLOR_BGR2GRAY, a constant defining the grayscale conversion code. The function converts the input image from one color space to another.

The following conversion occurs::

$$\text{RGB[A] do Gray: } Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Formula 1: Transformation occurring when calling the cvtColor function from the OpenCV library to convert an image to shades of gray, source: [11]

where R is red, G is green, and B is blue.

Their values are multiplied by the appropriate coefficients to obtain the color gray (formula 1). Conversion to grayscale can facilitate image analysis, reduce data complexity, increase performance, and enable better compatibility with some image processing algorithms.

Image Smoothing:

The Gaussian filter was used to smooth the image and reduce noise. The GaussianBlur function from the OpenCV library (code fragment 2) was used for this purpose, which uses the Gaussian kernel [12]. The function accepts the following parameters:

```
blurred = cv2.GaussianBlur(gray, (3, 3), 0)
```

Code Fragment 2: Passing arguments to a built-in GaussianBlur function

where: gray is the input image (3, 3) is the kernel size, 0 corresponds to the standard deviation value.

Filter kernel is the area that is moved over the image to perform the operation, in this case it is 3 pixels by 3 pixels (code fragment 2). A larger area size leads to more intensive smoothing, but can also introduce more blur. The standard deviation is responsible for the blurring of the image, increasing its value increases the blurring.

Operator Canny:

The Canny operator is used to detect edges on the previously smoothed image. The algorithm removes noise using a 5x5 Gaussian filter and finds edges using the image intensity gradient. The smoothed image is filtered by formula 2 in both horizontal and vertical directions to obtain the first derivative in the horizontal (Gx) and vertical (Gy) directions. By analyzing the collected information from both these images, the gradient of the edges and their direction can be determined for each pixel using the following formulas:

$$Edge\_Gradient(G) = \sqrt{G_x^2 + G_y^2}$$

$$\text{Angle}(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Formula 2: Calculating the direction and gradient of edges in the Canny edge detection algorithm

The gradient direction is always perpendicular to the edge. It is rounded to one of four angles representing the vertical, horizontal and two diagonal directions. After obtaining the gradient magnitude and direction, a full scan of the image is performed to remove pixels that are not edges by checking if the pixel is a local maximum in its neighborhood in the gradient direction. Then, a decision is made which of them are the most important, creating a binary image with clear edges, using two threshold values: minVal and maxVal (code fragment 3). All edges with intensity gradients above maxVal are definitely edges, and those below minVal are definitely not edges, so they are discarded. Those that are between these two thresholds are classified as edges or non-edges depending on their connectivity. If they are connected to pixels of another edge, they are considered as part of the edge. Otherwise, they are also discarded. This step also removes small pixel noises assuming that the edges are long lines. This way we get clear edges on the image.

The arguments passed to the function in the program are:

```
edges = cv2.Canny(blurred, threshold1=30, threshold2=100)
```

Code Fragment 3 Passing arguments to a built-in Canny function (OpenCV library)

where: blurred is the smoothed image passed to edge detection, thereshold1 is the first thresholding threshold that determines when edges are considered weak, and thereshold2 is the second thresholding threshold that determines when edges are considered strong.

Morphological Closing Operation:

Closing consists of two steps: dilation and expansion. Then we move on to erosion and reduction. The effect of this operation is filling small holes inside the objects [13]. The morphological closing operation was used to connect the separated edge areas. The ones function from the numpy library was used to create a 5x5 kernel, which is an array filled with 1. The morfologyEx function from the OpenCV library is used for closing, in which the image with edges, MORPH_CLOSE - a constant representing the closing operation and the previously created kernel are passed.

Finding Contours:

The contours in the image are found using the findContours function from the OpenCV library (code snippet 4). The parameters of the function include the source image, a method to approximate the contours (in this case, the outer contours), and a method to represent the contour, where the contours will be narrowed and only their significant points will be preserved [14].

Used parameters:

```
findContours(closed_edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Code Fragment 4: Calling a Built-in Function to Find Contours with Passed Arguments (OpenCV Library)

Filter, sort and select 10% of contours:

Only those that are closed (have more than 4 points) will remain. Closed contours are sorted by their area in descending order. Only the 10% largest contours among those sorted will be selected.

Analysis of Areas with Low Light Intensity:

Masks of low-intensity areas inside the selected contours are created. The low_intensity_areas image is initialized as a black image of the same size as the original image. Then, for each contour in largest_contours, a low_intensity_mask is created based on the low-intensity area inside that contour. This mask is then summed with the low_intensity_areas image using a logical AND operation (cv2.bitwise_and), allowing only the lowintensity areas to be included in the resulting image [15]. The whole process is repeated for all contours, ending with a low_intensity_areas image containing the low-intensity areas inside the largest contours.

Finding Contours in Low Light Areas:

Contours in low light areas are found using the previously described findContours function.

Filtering Outlines in Low Light Areas and Drawing:

What remains are the largest 0.5% of contours that are drawn in the resulting image.

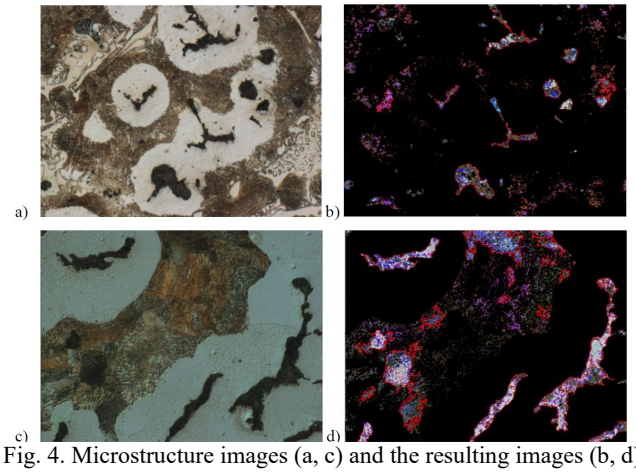

Fig. 4. Microstructure images (a, c) and the resulting images (b, d) generated for them with contours marked

Figure 4 shows the results of the program finding contours for two images of microstructures (a, c). In the resulting images (b, d), the areas of low light intensity were marked with a red contour. Using this method, it was possible to outline several microstructure precipitates, including vermicular cast iron. Elements with a high gradient were marked. In the further part of the paper, two methods will be presented to improve the performance of this program so as to ignore other microstructures.

Method 2 – Pink Squares

The next method focuses on "painting" the areas outside the ferrite precipitates. Modifications have been added to the previous code. The changes appear after drawing the contours on the image. A mask is created for the areas with contours. A thresholding operation is used to be able to adjust the image transformation in such a way that pixels with brightness above a certain threshold are set to the maximum value (255), and the rest to the minimum value (0). The mask has white pixels in the place of the contours, and the remaining pixels are black. Then, using the bitwise_not function from the OpenCV library, a bitwise negation of the mask is performed, which inverts the colors to the opposite. Areas that are not black are painted in pink (Figure 5, subpoints b, c).

Fig. 5. Microstructure images (a) with plotted changes with a threshold value of 20 (b) and 30 (c)

Figure 5 consists of 3 elements: the input image of the microstructure (a) and two output images that differ from each other due to the differently selected thresholds (b, c). Depending on the selected parameter values, the method works more or less precisely.

Method 3 – Searching Inside Closed Contours

Another idea for modifying the first program is to analyze the image for low intensity areas, but only inside closed contours, this way fragments located outside the ferrite areas should be ignored.

The modified program works similarly to its original, with the main changes:

• Thresholding using the Otsu algorithm..

```
_, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
```

Code Fragment 5 Calling the built-in function for image binarization with arguments (OpenCV library)

In code snippet 5, a grayscale image is converted to a binary image using Otsu's algorithm, where pixels are divided into white (255) and black (0) depending on a dynamically determined threshold. Otsu's algorithm analyzes the histogram of the image, looking for a binarization threshold that minimizes the intraclass variance (spread of pixels within classes) while maximizing the interclass variance (differences between classes) [16]. The result is an optimal threshold that effectively separates the background from objects in the image, enabling automatic binarization. This approach eliminates the need for manual threshold selection and is particularly effective in situations where the contrast or illumination of the image is a variable feature.

- Contours are filtered by analyzing the areas with the highest gradient: a list w is created, in which contours are filtered.
- Areas with the highest gradient (threshold above 0.8) surrounded by white pixels are selected and displayed
- The number of final areas is not fixed, but is adjusted depending on the sum of areas.

The areas of all contours are summed up and a threshold is created below which the number of contours will not be adjusted. If the sum of areas is greater than the threshold, the number of contours to draw will be equal to the quotient of the sum of areas and the area threshold. Otherwise, if the sum of areas does not exceed the threshold, the number of contours to draw will be equal to the number of filtered contours. Such a mechanism was introduced in order to flexibly adjust the number of lines drawn depending on the microstructure features. If there are many small areas in the image, the algorithm will be more restrictive and thus

fewer contours will be drawn. In the case of several large areas, a greater number of contours will be drawn.

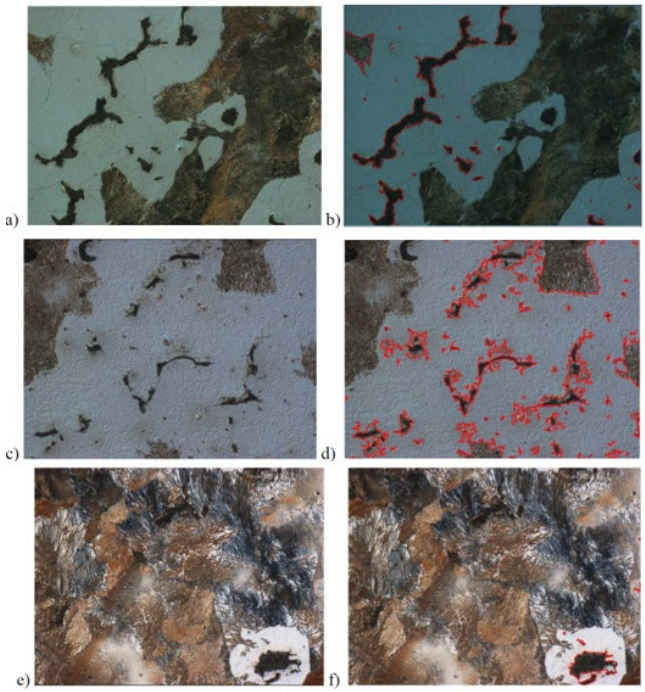

Fig. 6. Images of three microstructures (a, c, e) and their corresponding images generated by method 3 (b, d, f)

Analysis of Figure 6 leads to two conclusions. First: the program results seem to be the most precise for method number 3. Second: the largest number of contours is correctly marked, but not in every case (Figure 6d). Using this approach requires selecting parameters for specific cases, despite attempts to automate their selection. To obtain satisfactory results for all three methods, the parameters must be selected appropriately, depending on the image, which is why a different approach to recognizing vermicular cast iron in images will be used later.

2.3. Creating and training a neural network model

Keras and TensorFlow [17] were used to create the neural network. Keras is a deep learning programming interface written in Python, operating on the TensorFlow machine learning platform. It was created to enable fast analysis. The programs were written in Python using the TensorFlow library and its Keras module to implement a deep neural network. These programs are designed for image classification, specifically for distinguishing between two classes: "is" and "isn't". A database of images was created for the needs of the neural networks.

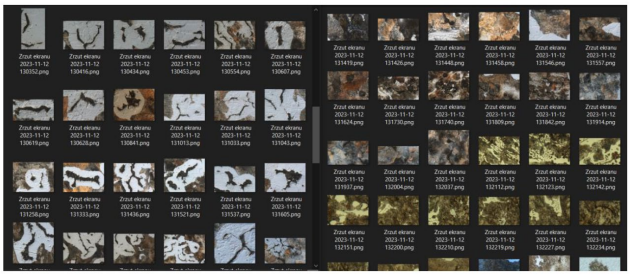

Fig. 7. Database prepared for training and validation of the artificial neural network

Figure 7 shows a database that was specially created for training and validation of neural networks. It consists of 262 elements, which contains individual segments of vermicular cast iron and images of entire melts. The database was divided into training and validation, which contain folders jest and nie_ma. 190 elements were used for training, and 72 elements for validation. Four versions of the program were created to create and train neural network models, the basic structure of which is similar. ImageDataGenerator was used to prepare training and validation data. Batches of tensor data are generated, which contain images, and in some cases, real-time data augmentation is also used [18]. Data augmentation is the process of introducing various random modifications to images to artificially increase the training set. In this way, the diversity of training data is increased and the overall ability of the model to generalize to new data is improved. The Sequential model was used. It is a simple model in Keras that allows building neural networks in a sequential manner, i.e. one layer after another [19].

Layers in models (each model has a different structure): Convolutional layers (Conv2D), Pooling layers (MaxPooling2D), Flatten and Dense.

- The Conv2D layer applies filters to the image, extracting features and creating a new output tensor. These filters are small "windows" that slide over the image [20].
- The MaxPooling2D layer reduces the size of spatial data by selecting the maximum values in a window of a specified size for each input channel. It is often used to reduce the dimensions of the data while preserving the important features [21].
- The Flatten layer in a neural network transforms complex, multidimensional image data into a one-dimensional list so that it can be processed by fully connected layers [22].
- The Dense layer [23] in a neural network is a standard block that transforms the input data by multiplying it by a weight matrix, adding a load vector, and applying an activation function. The units parameter specifies the number of neurons in the layer, and activation allows for setting the activation function, for example ReLU.

The sigmoid activation function was used for the binary classification task. It returns values between 0 and 1, where for small values (<-5) it approaches zero, and for large values (>5) it approaches one. It is often used to transform results into probability intervals [24]. The adam optimizer was used, which is based on adaptive estimation of first and second order moments. It works efficiently and has low memory consumption. The loss function was assumed to be binary_crossentropy used in binary

classification tasks. It measures the difference between the actual and predicted values for a problem where each sample can belong to one of two classes. The fit function is used to train the model on the training and validation data. For a specified number of epochs (for all versions 10), the model is trained on the provided training data (x and y). The arguments of this method allow to adjust the training process, such as the number of epochs, batch size, or use of validation data. The image size is the same and is 150x150 pixels, and accuracy is used as a metric. The differences in the programs include the choice of model architecture, activation functions, data augmentation, and use of existing architecture (in the program 4 VGG16).

Version 1:

ImageDataGenerator from Keras library was used for dynamic real-time image processing during model training. Parameter rescale=1./255 normalizes image pixel values to the range [0, 1].

The construction of the neural network model includes 5 layers.

- Conv2D: Convolutional layer with 32 filters of size (3, 3), ReLU activation function and input shape (img_height, img_width, 3), where 3 is the number of channels (RGB).
- MaxPooling2D: Max pooling layer of size (2, 2).
- Flatten: A layer that flattens the data into a one-dimensional vector before moving on to fully connected layers.
- Dense(64, activation='relu'): Fully connected layer with 64 neurons and ReLU activation function.
- Dense(1, activation='sigmoid'): Output layer with one neuron and sigmoid activation function.

Version 2:

- ImageDataGenerator was used with the same normalization parameter as in version 1.
- Changed the activation function to hyperbolic tangent (tanh) in the convolutional (Conv2D) and fully connected (Dense) layers of size 64. The tanh activation function replaces the standard ReLU function, which may impact the model's ability to capture nonlinear dependencies in the data by using a hyperbolic tangent activation function.

Version 3:

ImageDataGenerator was used to augment the training data through various transformations such as rotation, translation, skewing, zooming, and horizontal flipping. The neural network model contains three convolutional layers with ReLU activation function, MaxPooling layers for dimensionality reduction, Flatten layer for flattening the data, and two fully connected layers with ReLU and sigmoid activation function. Additionally, a Dropout layer was used to regularize the model by randomly excluding neurons during training.

Version 4:

ImageDataGenerator was used with the same normalization parameter as in version 1.

The VGG16 model from Keras library was used, omitting the Dense layers on top of the model (no top). Then, the weights of all layers of the base model were frozen to preserve the learned features. Custom layers were added, including Flatten to flatten the data, a 128-size Dense layer with a ReLU activation function, a Dropout layer for regularization, and an output layer with one neuron and a sigmoid activation function to solve the binary classification problem

3. Results for different versions

Version 1:

Tabela z wynikami Wersja_1:

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	7.5584	55.79%	5.7147	55.56%
2	2.6892	54.74%	0.6942	55.56%
3	0.6791	56.84%	0.7146	48.61%
4	0.6009	70.00%	0.6612	62.50%
5	0.5195	81.05%	0.6600	66.67%
6	0.4319	83.68%	0.7875	63.89%
7	0.4124	84.21%	0.6943	68.06%
8	0.3668	86.32%	0.8253	63.89%
9	0.3315	87.89%	0.6188	68.06%
10	0.2855	90.53%	0.7173	65.28%

Fig. 8. Program results for version 1

The analysis of the results presented in Figure 8 leads to the following conclusions:

The Validation Loss parameter initially decreases, but in some epochs it increases, which indicates possible overfitting of the model, especially if the difference between the Training Loss parameter and the Validation Loss parameter increases. The Validation Accuracy parameter remains at the level of 55-68%, which may indicate some difficulties in generalizing the model on the validation data. The model achieves high efficiency on the training data, but may be less efficient in generalizing to new data, which may indicate overfitting. Augmenting the data or subjecting the model to adjustments (e.g. dropout) can help improve the generalization ability.

Version 2:

Tabela z wynikami Wersja_2:

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	4.0961	49.47%	5.1950	44.44%
2	4.5494	50.53%	4.9863	44.44%
3	4.3595	50.53%	4.7720	44.44%
4	4.1706	50.53%	4.5547	44.44%
5	3.9756	50.53%	4.3376	44.44%
6	3.7898	50.53%	4.1178	44.44%
7	3.5880	50.53%	3.9048	44.44%
8	3.3944	50.53%	3.6923	44.44%
9	3.2085	50.53%	3.4766	44.44%
10	3.0183	50.53%	3.2611	44.44%

Fig. 9. Program results for 2 versions

The analysis of the results presented in Figure 9 leads to the following conclusions:

The Training Loss parameter is relatively constant and does not decrease significantly, suggesting that the model has difficulty adapting to the training data. The Training Accuracy parameter remains at 50.53%, indicating no improvement in classification on the training data. The Validation Loss parameter also remains relatively constant and does not show any significant change. The Validation Accuracy parameter remains at 44.44%, indicating that the model is unable to correctly classify on the validation data. It should be noted that the model in Version_2 does not demonstrate training efficiency or the ability to generalize to new data.

Version 3:

Tabela z wynikami Wersja_3:

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	0.7083	49.47%	0.6818	65.28%
2	0.6063	77.37%	0.7965	58.33%
3	0.5201	78.42%	0.7910	61.11%
4	0.5409	70.53%	0.6236	62.50%
5	0.5744	70.00%	0.6959	59.72%
6	0.5661	70.53%	0.6396	66.67%
7	0.4972	81.05%	0.5823	75.00%
8	0.4612	85.26%	0.7041	69.44%
9	0.4499	84.21%	0.6516	72.22%
10	0.4415	83.16%	0.7083	72.22%

Fig. 10. Results of the program for version 3

Analysis of the results presented in Figure 10 leads to the following conclusions:

The Training Loss parameter systematically decreases, which indicates the effective adaptation of the model to the training data. The Training Accuracy parameter increases, reaching 85.26% in the eighth epoch, which indicates that the model effectively classifies the training data. The Validation Loss and Validation Accuracy parameters show some variability, but generally remain at a satisfactory level. The Validation Accuracy parameter reaches 75.00% in the seventh epoch, which indicates the model's ability to generalize to the validation data. The model in Version_3 seems to cope better with both training and validation data compared to previous versions.

Further adjustments to hyperparameters or model refinements may be considered to further improve the results.

Version 4:

Tabela z wynikami Wersja_4:

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	0.7096	63.68%	0.5289	73.61%
2	0.3624	85.26%	0.5051	79.17%
3	0.2365	91.05%	0.4220	83.33%
4	0.1963	93.68%	0.6110	79.17%
5	0.1266	95.26%	0.4624	81.94%
6	0.1328	96.32%	0.4739	81.94%
7	0.1045	96.84%	0.5546	79.17%
8	0.0928	97.37%	0.5881	79.17%
9	0.0903	97.89%	0.4447	80.56%
10	0.0620	99.47%	0.5738	79.17%

Fig. 11. Program results for 4 versions

Data analysis in Figure 11 leads to the following conclusions:

The Training Loss parameter systematically decreases, which indicates the model is effectively adapting to the training data. The Training Accuracy parameter increases, reaching an impressive 99.47% in the tenth epoch, which indicates that the model is very good at classifying the training data. The Validation Loss parameter also decreases, which indicates the model's ability to generalize. The Validation Accuracy parameter remains at a level above 79%, which indicates the model's effectiveness in classifying the validation data. The model in Version_4 achieves very high effectiveness on the training data, which may suggest that the model may be too well-fitted to the training data (overfitting). The results on the validation data (79.17%) are satisfactory.

In summary, Version 2 is the least effective in both training and generalization. Version 4 may be too well-fitted to the training data, which suggests the possibility of further analysis and optimization..

4. Conclusions

Selected solutions and their modifications were presented. Such a process allowed for drawing conclusions and a deeper understanding of the features of the problem that was undertaken. The use of descriptors did not provide an unambiguous solution to the problem, but helped to understand that calculating the gradient can be an important factor in identifying vermicular cast iron precipitates. It is also possible to collect descriptors from different areas of the image into one feature vector, which is a compact representation of key information. On this basis, it is possible to train the classifier, allowing the model to recognize unique features in a given context, in order to be able to identify objects in the photo. The use of image processing techniques for analysis and edge detection showed that selecting the appropriate parameters for each case leads to correct solutions, but with a database containing more diverse elements it can be burdensome. In the last approach - the use of artificial neural networks, thanks to this, it was also possible to achieve satisfactory results. In this case, the larger the database used for training and validation, the more effective adaptation of the model to training data and the ability to effectively generalize to new data would be possible. In cases where the database contains only a few hundred elements, it is a good practice to use data augmentation, which allows improving the results obtained using the generalization model.

The conducted study analyzed the potential use of selected artificial intelligence algorithms for evaluating the microstructure of vermicular cast iron. The obtained results indicate that image processing techniques and gradient analysis can serve as effective tools for identifying graphite precipitates; however, their effectiveness depends on the parameters of the applied methods and the characteristics of the analyzed dataset. The analysis highlights the potential of the applied models, but expanding the training dataset and further optimizing the network architecture are necessary to enhance their generalization ability and effectiveness in real industrial conditions. Future research should focus on further testing the developed methods on real industrial data and their integration with quality control systems in foundry processes.

References

[1] Kosa, E. (2012). *Research on the structure of nitrided layers produced on EN-GJL 250 cast iron under glow discharge conditions.* Engineering diploma thesis, Warsaw University of Technology, Faculty of Materials Engineering. (in Polish).

[2] Szpak, Ł. (2014). *Image classification based on the author's hypsometric representation*. Master's thesis, Warsaw University of Technology, Faculty of Electronics and Information Technology, Institute of Computer Science. (in Polish).

[3] de Albuquerque, V.H.C., Cortez, P.C., de Alexandria, A.R. & Manuel, J.R.S. (2008). A new solution for automatic microstructures analysis from images based on a backpropagation artificial neural network. *Nondestructive Testing and Evaluation*. 23(94), 273-283. https://doi.org/10.1080/10589750802258986.

[4] Han, Y., Lai, C., Wang, B. & Gu, H. (2019). Segmenting images with complex textures by using hybrid algorithm. *Journal of Electronic Imaging.* 28(1), 013030. DOI: 10.1117/1.JEI.28.1.013030.
[5] Fotos, G., Campbell, A., Murray, P. & Yakushina, E. (2023). Deep learning enhanced Watershed for microstructural analysis using a boundary class semantic segmentation. *Journal of Materials Science*. 58(36), 14390-14410. https://doi.org/10.1007/s10853-023-08901-w.
[6] Azlan Suhaimi, M., Park, K.H., Sharif, S., Kim, D.W. Saladin Mohruni, A. (2017). Evaluation of cutting force and surface roughness in high-speed milling of compacted graphite iron. In MATEC Web of Conferences, 7-9 June 2017 (pp. 1-7). DOI: 10.1051/MATECCONF/201710103016.
[7] Zhu, C. & Zhang, A. (2007). Experimental study on fracture toughness of vermicular cast iron. *Journal of Mechanical Strength*. 29(2), 310-314.
[8] Guzik, P. (2014). *Methods of searching for characteristic points and their features*. Engineering diploma thesis, Warsaw University of Technology Faculty of Electronics and Information Technology Institute of Computer Science. (in Polish).
[9] Nowik, A. (2014). *Local descriptors using reduced binary histogram in similar image retrieval*. Warsaw University of Technology, Faculty of Electronics and Information Technology Institute of Computer Science. (in Polish).
[10] Function Documentation. (2023). *Color Space Conversions*. Retrieved November 25, 2023, from https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab
[11] Color conversions. (2023) Retrieved November 23, 2023, from https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html
[12] Smotking Images: Gaussian Blurring. (2023). Retrieved November 23, 2023, from, https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html
[13] Morphological Transformations: Closing. (2023). Retrieved November 23, 2023, from https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html
[14] Structural Analysis and ShapeDescriptors: findContours. (2023). Retrieved November 23, 2023, from, https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a
[15] Arithmetic Operations on Images: Bitwise Operations. (2023). Retrieved November 23, 2023, from, https://docs.opencv.org/3.4/d0/d86/tutorial_py_image_arithmetics.html
[16] Image Thresholding: Otsu's Binarization. (2023). Retrieved November 23, 2023, from, https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html
[17] About Keras: Keras & TensorFlow 2. (2023). Retrieved November 26, 2023, from https://keras.io/about/tf.keras.preprocessing.image.ImageDataGenerator, Retrieved 2023-11-26, https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator#used-in-the-notebooks
[18] fchollet, The Sequential model. (2023). Retrieved November 26, 2023, from https://keras.io/guides/sequential_model/
[19] Conv2D layer. (2023). Retrieved November 26, 2023, from https://keras.io/api/layers/convolution_layers/convolution2d/
[20] MaxPooling2D layer. (2023). Retrieved November 23, 2023, from https://keras.io/api/layers/pooling_layers/max_pooling2d/
[21] Flatten layer. (2023). Retrieved November 27, 2023, from https://keras.io/api/layers/reshaping_layers/flatten/
[22] Dense layer. (2023). Retrieved November 27, 2023, from https://keras.io/api/layers/core_layers/dense/
[23] Layer activation functions. (2023). Retrieved November 26, 2023, from, https://keras.io/api/layers/activations/#sigmoid-function
[24] Adam. (2023). Retrieved November 26, 2023, from https://keras.io/api/optimizers/adam/