Crack detection in lithium-ion cells using machine learning

Lukas Petrich[a,*], Daniel Westhoff[a], Julian Feinauer[a], Donal P. Finegan[b], Sohrab R. Daemi[b], Paul R. Shearing[b], Volker Schmidt[a]

[a] *Institute of Stochastics, Ulm University, D-89069 Ulm, Germany*
[b] *Electrochemical Innovation Laboratory, Department of Chemical Engineering, University College London, WC1E 6BT, London, UK*

Abstract

It is an open question how the particle microstructure of a lithium-ion electrode influences a potential thermal runaway. In order to investigate this, information on the structural changes, in particular cracked particles, caused by the failure are desirable. For a reliable analysis of these changes a reasonably large amount of data is necessary, which necessitates automatic extraction of particle cracks from tomographic 3D image data. In this paper, a classification model is proposed which is able to decide whether a pair of particles is the result of breakage, of the image segmentation, or neither. The classifier is developed using simulated data based on a 3D stochastic particle model. Its validity is tested by applying the methodology to hand-labelled data from a real electrode. For this dataset, an overall accuracy of 73% is achieved.

Keywords: thermal runaway, lithium-ion battery, crack detection, machine learning, 3D microstructure, stochastic modelling;

1. Introduction and motivation

Lithium-ion batteries combine several beneficial attributes, such as high energy density and low self-discharge. However, one of the biggest drawbacks is their vulnerability to thermal runaway [1]. This is when the temperature of the cell exceeds a certain threshold, e.g. through overcharging, and exothermic decomposition of the electrodes generates additional heat accelerating the process. Such catastrophic failures are rare, but devastating.

*Corresponding author
Email address: lukas.petrich@uni-ulm.de (Lukas Petrich)

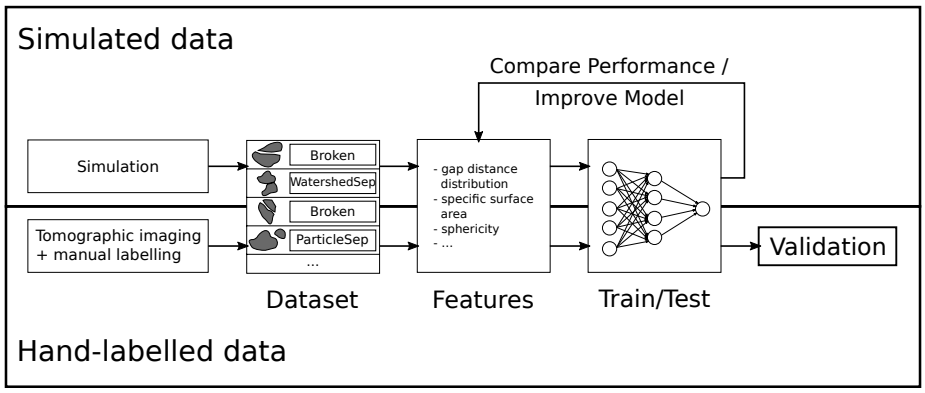


Figure 1: Overview of the model development.

During the thermal runaway particles in the electrode material break, which then affects the reaction [2]. More precisely, it has been shown that smaller particle sizes (and thus higher specific surface area) lead to more intense heat generation with lower onset temperature (see e.g. [3, 4]). However, information on the broken particles is required for a more detailed analysis of which particles are more likely to crack and thus worsen the safety of the cell. The main goal in this paper is to detect these particles and hence to pave the way for further research.

Finding cracked particles, i.e., particles that broke into two parts, has always been challenging. Typically this is done by visual inspection. This method however has several disadvantages. It is time-consuming and processing a large number of particles is infeasible. Moreover, it is tedious even for smaller particle systems and often leads to errors. Other approaches are based on advanced segmentation algorithms [5]. However, such segmentation algorithms often have to be specifically tailored in order to take into account the given features of the considered image data. In the present paper, an algorithm based on machine learning is proposed, where the complex, non-linear nature of our problem makes supervised machine learning [6, 7] an appropriate tool, since the associated techniques possess the ability to make predictions based on previously learned sample data. The classifier, which considers pairs of particles of the post-mortem cell, decides whether they are the result of breakages. In case a particle broke into multiple pieces, every pair of (neighbouring) fragments is detected. To the best of our knowledge, this is the first work of using machine learning for crack detection in lithium-ion batteries.

To develop our classification model, we use simulated data. We generate a particle structure based on a parametric stochastic model (see [8]) and extract pairs of particles with their class labels, which describe the relationship to each other, e.g. if they are the result of breakage. Then, different features and classifiers are investigated. The resulting predictive classifier is retrained and evaluated on hand-labelled data to verify that it is also applicable to real-world

data. This process is illustrated in Figure 1. Note that, in contrast to full hand-labelling, for training of the classifier only a (relatively) small number of labelled particle pairs is necessary, and thus the enormous effort of hand-labelling is strongly reduced.

The algorithm distinguishes three classes. We have a class label for particle pairs which are the result of breakage and one for those which were already separated in the pristine battery cell. But particles which touch each other can also be separated by the watershed algorithm [9] in the course of the image segmentation (cf. Section 2.1). Consequently, we differentiate the following three classes of particle pairs:

- Broken: The particle pair belonged to the same particle before it broke apart during the thermal runaway.
- WatershedSep: The particle pair corresponds to two touching particles in the tomographic image which are split by the watershed transformation.
- ParticleSep: The particle pair consists of unrelated, separate particles, i.e. a pair which is neither Broken nor WatershedSep.

Furthermore, these three classes describe the relationship of one particle to another one. It is hence possible that e.g. a particle which is part of a Broken-pair, can also belong to a (different) WatershedSep-pair. In Figure 2 and Figure 3 several examples of particle pairs are shown.

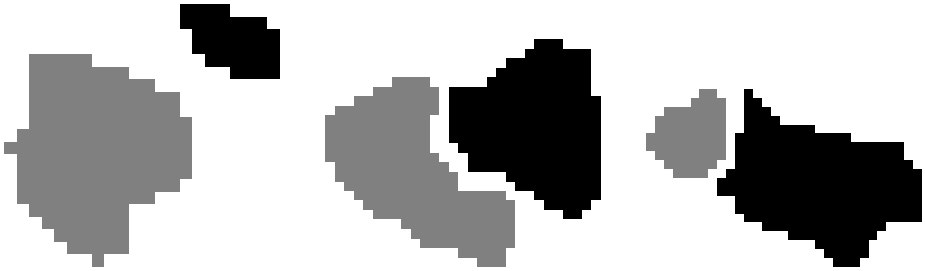
The rest of this paper is structured as follows. In Section 2, we describe tomographic imaging and the acquisition of the validation dataset. After that, in Section 3, we present our algorithm to simulate sample data and the classification model. In Section 4, we follow up with the evaluation results of the classifier on the simulated and on the validation dataset, and discuss its performance. The conclusion in Section 5 provides a summary and further research opportunities.

2. Description of tomographic data

The tomographic dataset used in this work was retrieved from [2] where a commercial $LiCoO_2$ cell underwent thermal runaway via high-rate overcharge electrical abuse. The $LiCoO_2$ sample was extracted post-mortem from an outer layer of the degraded electrode assembly, and exhibits a significantly reduced mean particle diameter, relative to an equivalent sample in its fresh state. This reduction in particle size is described in [2] as being due to the particles cracking when subject to the high temperatures and heating rates associated with thermal runaway.

2.1. Imaging and data pre-processing

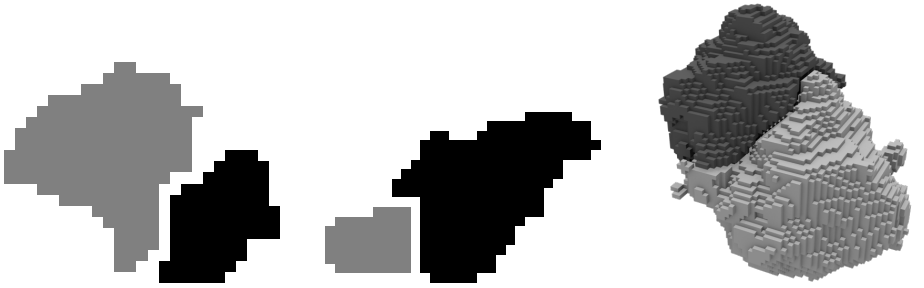
The sample was extracted from the degraded $LiCoO_2$ cell and mounted onto the top of a pin using quick-setting epoxy (Devcon Epoxy Glue). The sample was then imaged in a lab-based X-ray nano-CT system (Zeiss Xradia Ultra 810, Carl



(a) 2D slice of a ParticleSep particle pair.

(b) 2D slice of a Broken particle pair.

(c) 2D slice of a Broken particle pair.

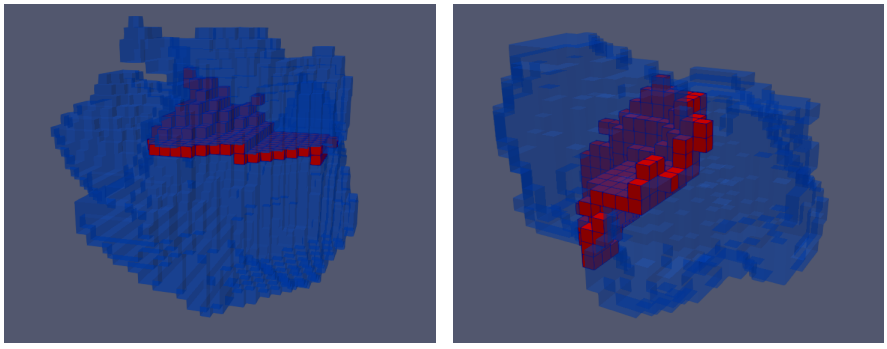


(d) 2D slice of a WatershedSep particle pair.

(e) 2D slice of a WatershedSep particle pair.

(f) 3D rendering of a Broken particle pair.

Figure 2: Examples of particle pairs from the hand-labelled dataset with their class labels.



(a) WatershedSep particle pair.

(b) Broken particle pair.

Figure 3: 3D renderings of the gap (red) between two particles (semi-transparent blue).

Zeiss X-ray Microscopy, USA) in absorption mode with a quasi-monochromatic beam of 5.4 keV. An objective lens giving an optical magnification of 20 was used with 10.3 X-ray magnification with binning 2×2, giving an effective pixel size of 0.1262 µm. The reconstructed tomogram consisted of 901 radiographs taken over a 180° rotation, each captured with an exposure time of 15 s. The radiographs were reconstructed using the Zeiss XMReconstructor software package, which uses a filtered-back-projection algorithm.

We binarized the tomographic image by assigning voxels to the solid (particle) phase or to the void phase. For this, the data has been smoothed with a Gaussian blur filter ($\sigma = 1$) reducing the noise of the image (see [10]). Then a global threshold was applied. The threshold was chosen by visual inspection.

Next, small holes were temporarily removed to keep the watershed transform more stable and to avoid oversegmentation. In order to do this, we detect clusters of the pore space with the Hoshen-Kopelman algorithm [11]. All clusters with less than 10000 voxels are added to the solid phase. This value is chosen to eliminate small perturbations, but it leaves the general shape of the particles intact.

We use a marker-based watershed [9] for image segmentation. For that, the Euclidean distance transformation I_{dist} of the binary Image I is determined, where the distance from every solid voxel to its nearest void voxel is computed. Let $-I_{dist}$ be its negation. Filtering out some local minima of $-I_{dist}$, as described in [12], the remaining local minima are used as markers for the watershed algorithm on $-I_{dist}$. The basins split the solid phase of the binary image I into separate particles, where broken particles are represented as two separate ones.

The last step of the pre-processing is to restore the small holes by adding them to the pore space.

2.2. Labelling of particle pairs

The separate particles were labelled with unique numeric identifiers. Hand-labelling of particle pairs was performed manually by scrolling through the slices of the reconstructed tomogram, and noting the numeric labels of the particle pairs along with their status as Broken, WatershedSep, or ParticleSep, as judged by visual inspection.

3. Classification of particle pairs

We consider two different datasets. One consists of the hand-labelled tomographic image of a post-mortem battery cell described in Section 2 and the other one is computer simulated. The latter dataset gives us an arbitrarily large number of samples which are guaranteed to have the correct labels and thus allows us to do accurate feature engineering.

The steps for the investigation of the simulated data are as follows:

1. Realise a random pristine particle system.

2. Break some particles while remembering the relationship (class label) of all pairs.

3. Build a list of particle pairs with their corresponding class labels.

4. Compute the feature vector for each particle pair.

5. Train and evaluate the classifier.

When we use hand-labelled data we start with step 4. In the next sections these steps are explained in more detail.

3.1. Simulated particle pairs

For the computer generated data we simulate a particle system for a pristine battery cell. Some of its particles are then broken to mimic the effect of overcharging in the real cell. From this artificial "post-mortem" particle system we build a list of particle pairs. For the implementation of this technique, the `GeoStoch` library [13] is used.

3.1.1. Pristine particle system

We start with simulating a system of pristine particles with the methods introduced in [8, 14]. All consecutive steps are illustrated in Figure 4. In the following, we give a short summary of the algorithm, details can be found in [8].

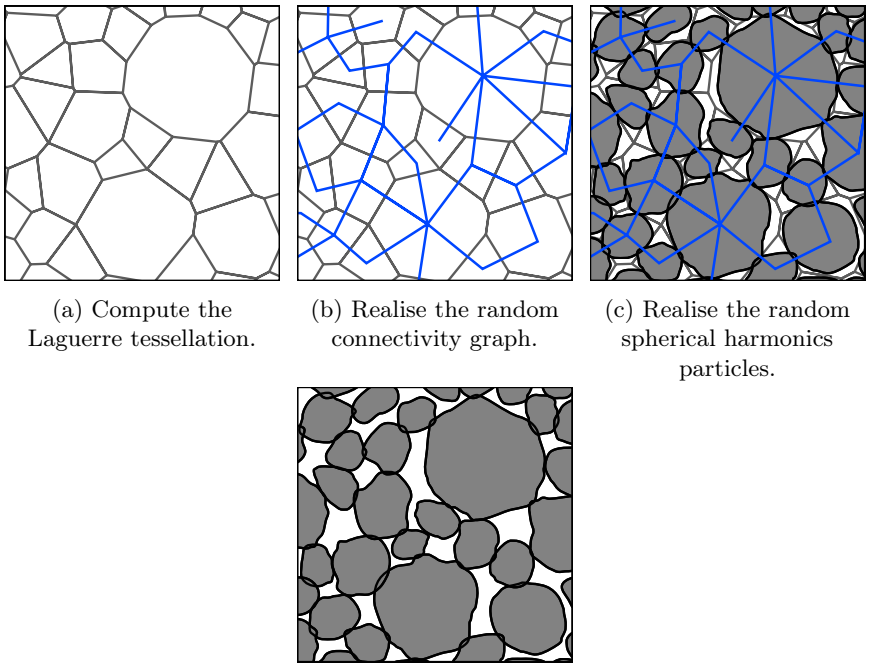
To begin with, a marked point process, namely a random sequential absorption process, is realised, see [15]. Based on that the corresponding Laguerre tessellation (cf. [16]) is calculated which provides convex polyhedrons defining the habitat of the particles.

The connectivity of the particles is roughly described by a random graph. It is based on the connectivity graph of the polyhedrons, which is then reduced to a spanning tree. The tree is determined such that it is maximal with respect to the surface areas of the corresponding polyhedron faces. Additional edges are included with a certain probability that increases with the same surface areas, see [8]. For more details on graph theory we also refer to [17].

The particles are modelled by means of Gaussian random fields on the sphere. For that random spherical harmonics particles (cf. [18]) are generated in the polyhedrons. The particles obey the connectivity graph in such a way that particles are connected where the connectivity graph indicates so. Furthermore, for each particle the ratio of its volume to the volume of the corresponding polyhedron matches the volume fraction of the material.

Then we only keep the system of spherical harmonics particles omitting Laguerre tessellation and connectivity graph.

Since we consider the same type of battery, the same model parameters as in [8] are used.



(a) Compute the Laguerre tessellation.

(b) Realise the random connectivity graph.

(c) Realise the random spherical harmonics particles.

(d) Delete the Laguerre tessellation and the connectivity graph.

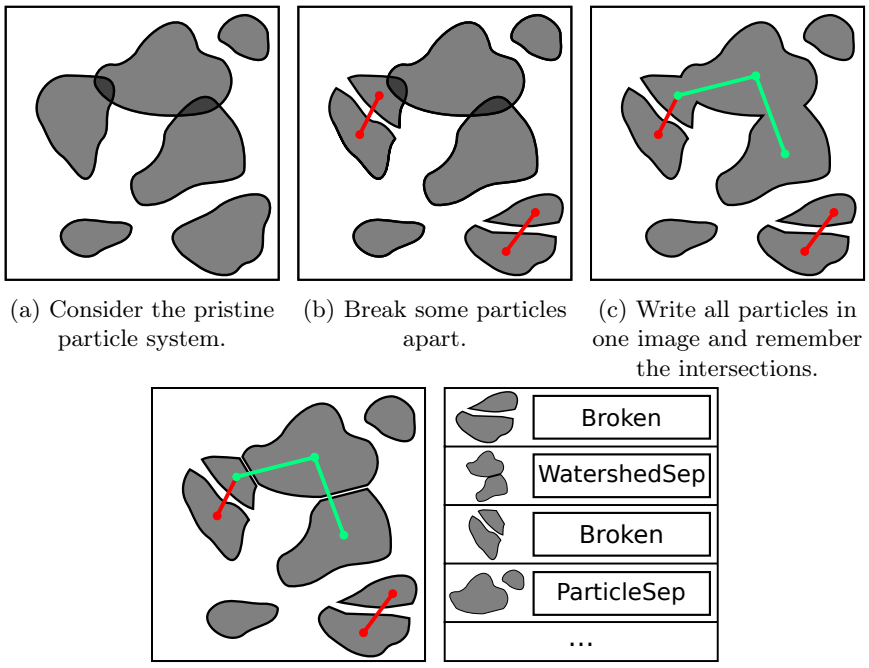
Figure 4: Steps for simulating the pristine particle system.

3.1.2. Extracting labelled particle pairs

Every particle in the pristine system is voxelized and a certain percentage of the particles is additionally broken apart (see Section 3.1.3). Note that the result of this procedure is analogous to the binary image of the experimental data described in Section 2.1.

For the purpose of keeping track of the class labels we define a graph G whose vertices represent the voxelized particles and its edges are marked with the type of the corresponding particle pair. All breakages are added as Broken-edges to the (initially empty) graph G. Furthermore, we consider the union of all voxelized particles as an image I. For every intersecting pair of particles (which we interpret as two particles touching each other) we add an edge to the graph with a WatershedSep label.

In order to separate the particles that touch each other in the voxelized image (which will be the input for the crack detection algorithm), we apply a watershed-based segmentation [9]. However, we slightly deviate from the segmentation procedure for the experimental data to ensure that particles are not wrongly separated, i.e., we choose exactly one watershed marker for each particle. To determine this marker, we compute the Euclidean distance transform



(a) Consider the pristine particle system.

(b) Break some particles apart.

(c) Write all particles in one image and remember the intersections.

(d) Apply the watershed algorithm.

(e) Collect all adjacent particles with their classes.

Figure 5: Steps for extracting the sample particle pairs using a graph to memorise the class labels.

of the binary image I and search for the maximum in each particle.

Finally, we create a list of particle pairs that contains all particle pairs that have a distance smaller than some threshold d_{pair} to each other in the voxelized image. The associated class label is retrieved as the mark of the corresponding edge in the graph, or if no edge is present, the ParticleSep-class is chosen. Note that each particle can occur in several particles pairs. The whole process is depicted in Figure 5.

Summarizing, the procedure for generating the voxelized particle system and the corresponding class labels can be described as follows:

1. Let G be an undirected graph whose vertices correspond to the voxelized particles and whose edges represent potential particle pairs marked with their class labels.

2. For each spherical harmonics particle p_{shp}:

   (a) Voxelize p_{shp} to p_{voxel}.

   (b) With probability q_b, break p_{voxel} and add both fragments as vertices to G connected with a Broken-marked edge. Otherwise add only

p_{voxel} as vertex.

3. Let I be the union of all voxelized particles. If two particles p and p' intersect, add a WatershedSep-marked edge from p to p'.

4. Compute the Euclidean distance transformation I_{dist} of I, where for every solid voxel the distance to its nearest void voxel is determined.

5. Let I_{marker} be the image which represents the regional maximum in each voxelized particle w.r.t. I_{dist}.

6. Separate the particles in I using the watershed transform of $-I_{dist}$ based on the marker image I_{marker}.

7. Arrange all particles in pairs which are within a distance smaller than d_{pair} of each other.

8. List the particle pairs together with the class label of the matching edge in the graph, or ParticleSep if there is no edge.

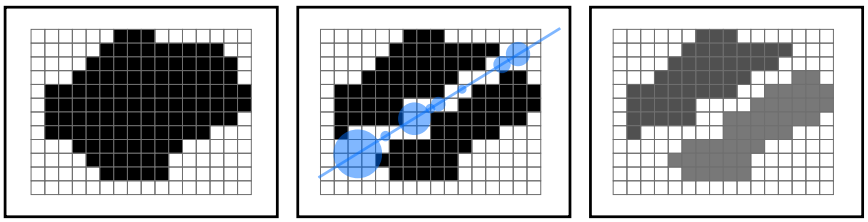
We choose the probability for a particle to break to be 0.3, but since we only use a subset of the particle pairs in order to have the same number of instances for each class (see Section 3.2.2), this parameter has no real impact. For the pairing distance threshold d_{pair}, we use the same value as for the gap distance histogram threshold d_{hist} in Section 3.2.1, namely 15.

3.1.3. Particle breakage

The main focus of the breakage algorithm is to be simple on the one hand but to produce results which are not too easy for a classifier to identify on the other hand. Note that the proposed method does not aim to reproduce the underlying physics of particle breakage.

First, we realise a random plane through the particle's centroid and remove every solid voxel touching it. This guarantees a separation. However, doing only this would be trivial for a classifier to detect since the gap between the two fragments of the particle has constant width (cf. Section 3.2.1). Thus we roughen the breakage surface with a Boolean model [15]. For that, we realise a homogeneous Poisson point process on the separation plane. The points act as centres of balls whose radii are Gamma distributed with constant variance and mean that grows linearly with the distance to the centroid. The broken particle pair consists of all voxels of the original particle which do not touch the plane nor any ball of the Boolean model. But, because the balls can become arbitrarily large, we have to make sure that none of the particle fragments is completely covered and we end up with only one fragment. In this case, we restart the algorithm for the current particle. The entire procedure is illustrated in Figure 6.

In the following, the breakage algorithm for a particle with centroid c_{part} is summarized.



(a) Consider a voxelized particle.

(b) Remove all voxels which touch the plane or are in the Boolean model.

(c) Consider both fragments as new particles.

Figure 6: Steps for breaking particles.

1. Realise a random plane P through the centroid c_{part} whose normal vector is uniformly distributed on the unit sphere.
2. Simulate a homogeneous Poisson process $\{S_n\}$ on P with intensity λ_b.
3. Mark every point S_n with an independent gamma-distributed random variable R_n such that $\mathbf{E}[R_n] = \alpha_b \|S_n - c_{part}\|$ and $\mathrm{Var}[R]_n = \sigma_b^2$.
4. Let the Boolean model B be the union of balls with midpoints S_n and radii R_n.
5. Add every solid voxel in p which touches P or is contained in B to the void phase.
6. If there are not exactly two connecting components, restart with step 1.

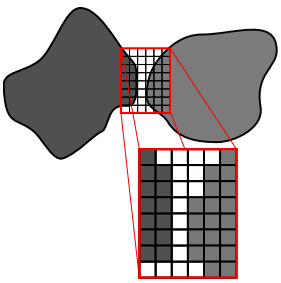
Note that in experimental data particles can also break into more than two fragments. As our classification algorithm works on particle pairs, it is not necessary to represent this property in the simulated data, as cracks would be identified individually for each pair of the parts of the broken particle.

We choose the parameters to be $\lambda_b = 0.4, \alpha_b = 0.075$ and $\sigma_b^2 = 0.2$, which lead to optically reasonable results.

3.2. Classification

Now we assume to have a list of labelled particle pairs. The following procedure does not differ between simulated or hand-labelled data. In both cases we first extract the features from the particle pairs before we train and test the classifier.

The samples are chosen such that every class occurs with the same frequency. If one class has more instances than another one, we use only a random subset of these instances (a procedure called random undersampling). For a dataset with balanced classes the performance metrics will weight every class equally (see Section 4). This is important since we do not want to make assumptions on the frequency of the classes in the tomographic image.



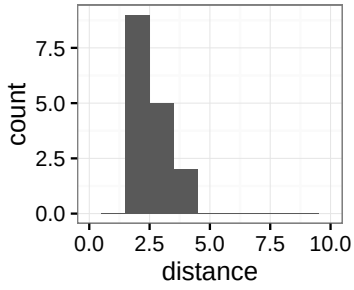
(a) Detailed view on the gap between two voxelized particles.

√17	√10	√5	√2	1
√13	√8	2	1	
√10	√5	√2	1	
3	2	1		
3	2	1		
3	2	1		
3	2	1		
√10	√5	√2	1	

(b) Euclidean distances to the nearest voxel of the first particle (values of boundary voxels in red).

	1	√2	√5	√10	√17
		1	2	3	4
		1	2	3	4
		1	2	3	4
		1	2	3	4
		1	2	3	4
		1	2	3	4
1	1	√2	√5	√10	√17

(c) Euclidean distances to the nearest voxel of the second particle (values of boundary voxels in red).



(d) Distribution of the distance values of all boundary voxels.

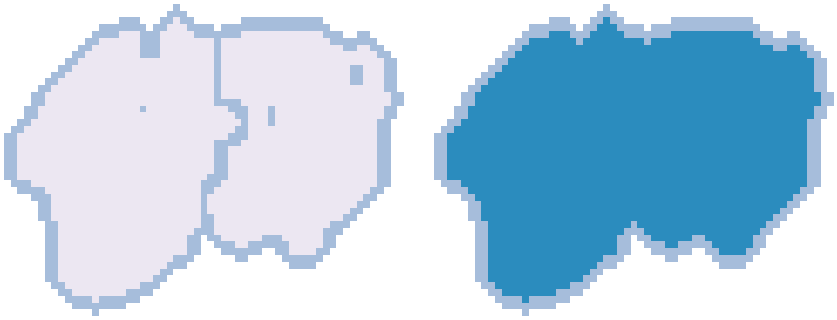
Figure 7: Steps to compute the gap distance distribution.

3.2.1. Features

Good features are probably the most crucial component of every machine learning application [6, 7]. Thus, we considered multiple features, but the one with the most predictive power describes the gap between the particles.

The idea is as follows: A particle can be part of pairs with different class labels. Thus, the geometries of only the individual particles themselves are not sufficient to determine the type of a pair they belong to. The shape of the space between two particles contains much more information.

In order to determine the gap distance distribution we consider voxelized particle pairs. In a first step we collect the (smallest) distances from every boundary voxel of one particle to the other one. Here, a boundary voxel is a voxel that belongs to the solid phase but has a common face with a pore space voxel. This process is illustrated in Figure 7. The distribution of the distance values can be represented in two different ways. First we compute the moments up to the fourth one and second we determine a normalized histogram where each bin has a width of one. Since most classifiers expect feature vectors to have



(a) Original particle pair (grey) and its dilation (light blue).

(b) Dilated particle pair (light blue) and its erosion (cyan blue).

Figure 8: Morphological closing of a particle pair with a ball of radius $r_{comb} = 2$

a common dimension, we truncate the histogram at a certain threshold d_{hist}. Combining both leads to a $(d_{hist} + 1) + 4$ dimensional feature vector to describe the gap of a particle pair. We choose $d_{hist} = 15$.

We also try to combine the two particles and investigate how various shape characteristics change. In order to get a stable result we reorder the particle pairs such that the first particle has a larger volume than the second one. The combination is achieved through a morphological closing with a ball of radius r_{comb}. For that, both particles are first enlarged with radius r_{comb} (dilation) and then shrunken with the same radius (erosion). See Figure 8 and confer [15] for more details. This removes the small openings, in particular narrow cracks. However, if the closing radius is chosen too large, to many details are smoothed out. That is why we settled for $r_{comb} = 2$.

Considering the two particles individually and additionally their combination, we compute their specific surface areas and their sphericities. The latter measures how similar the shape of a particle is to a sphere. Furthermore, let V_1, V_2, V_{comb} be the volumes, and A_1, A_2, A_{comb} the surface areas of the corresponding particles. We also include the following fractions in the feature vector:

$$\frac{V_2}{V_1}, \frac{V_{comb}}{V_1 + V_2}, \frac{A_2}{A_1}, \frac{A_{comb}}{A_1 + A_2}$$

All in all, our feature vector contains 30 values for each pair of particles.

3.2.2. Classifier

For the classification, we use a so-called multilayer perceptron (MLP). We give a brief introduction, but for a more comprehensive discussion on MLPs and machine learning in general, we refer to [6] and [7].

The MLP is a feed-forward neural network with one hidden layer (cf. Figure 9). It tries to solve the problem of mapping a feature vector $(x_1, \ldots, x_d)$ to one of k class labels $(y_1, ..., y_k)$ by imitating biological nerve cells. For that, every unit (neuron) weights its inputs and adds them together. This activation

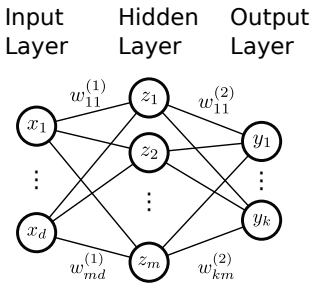


Figure 9: Illustration of a multilayer perceptron for a d-dimensional feature vector and k class labels with m hidden units.

value is then applied to a sigmoid function which acts as a threshold and lets the neuron "fire" if the weighted sum of inputs exceeds the threshold. The result is passed to the next layer as its input. The class label l which corresponds to the output unit y_l with the highest value is chosen as the prediction.

The neural net is trained by finding weights $\mathbf{w} = (\mathbf{w}^{(1)}, \mathbf{w}^{(2)})$ which minimise the cross-entropy (or log) loss function [7] plus the (squared) L_2 norm of the vector $\mathbf{w}$ weighed with $\alpha > 0$ (L_2 regularisation). We solve this optimisation problem using the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm (L-BFGS) which is a variant of the quasi-Newton method BFGS using only a limited amount of computer memory, see [19].

Since the performance of the MLP is sensitive to the scaling of the inputs, we apply standardisation, that is, we centre every component of the feature vector to mean zero and scale it such that its standard deviation is one. However, to ensure that we do not divide by zero we have to remove features which have zero variance (primarily the first two bins of the gap distance histogram). Thus, our classifier consists of removing constant features, standardisation and the actual MLP classification.

The hyperparameters (i.e. the number of hidden units m and the weight α of the L_2 regularisation) are determined by a grid search maximising the accuracy which is estimated through a 5-fold stratified cross-validation. The accuracy is defined as the proportion of correctly classified instances.

In summary, a given dataset is first split into training and test data. Then the following steps are performed.

1. Find the hyperparameters which maximise the accuracy, i.e., for each value:
   (a) Determine the average accuracy using a 5-fold stratified cross-validation on the training data, i.e. for each split:
      i. Train the classifier w.r.t. the specified hyperparameters.
      ii. Determine the accuracy using the part of the training data that has not been exploited so far.
2. Refit the classifier with the entire training set.

3. Evaluate the performance on the test set.

The implementation of the classifier used in this paper is based on the software library `scikit-learn` [20].

4. Results

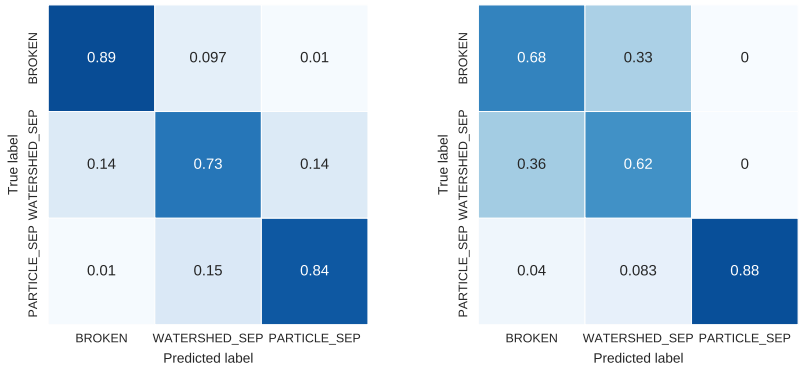
For the performance evaluation of the classifier we randomly choose a quarter of the undersampled dataset for testing, but leaving the ratio of the instances for each class equal. The other 75% are used for training and hyperparameter tuning as described in Section 3.2.2.

4.1. Performance on the simulated data

The simulated dataset consists of 3693 samples and therefore 924 instances for testing. The result of the hyperparameter fitting is given as follows: $\alpha = 0.1$ for the weight of the L_2 regularisation term, and $m = 5$ for the the number of hidden units.

On the test set, the classifier achieves an overall accuracy of 82.1%. The confusion matrix (see Figure 10a) suggests that the Broken instances are classified quite accurately. The result for the ParticleSep samples is also acceptable. However, it has a hard time figuring out if an instance is the result of the watershed transformation. Virtually all false positives and false negatives are related to this class.

This also affects the precision and recall. For a class label l the precision is an estimate for the probability that an instance classified as l, actually belongs to l. On the other hand, a high recall score indicates that a true l-instance is probably classified as such. Good predictions must have both high precision and high recall. That is why the F_1 score, their harmonic mean, is considered.



(a) Simulated data. (b) Hand-labelled data.

Figure 10: Normalized confusion matrices for the simulated and the hand-labelled data.

	precision	recall	F_1	support
Broken	0.859	0.893	0.876	308
WatershedSep	0.749	0.727	0.738	308
ParticleSep	0.852	0.844	0.848	308
average / total	0.820	0.821	0.821	924

Table 1: Performance metrics for the classifier based on the simulated test data.

	precision	recall	F_1	support
Broken	0.630	0.680	0.654	25
WatershedSep	0.600	0.625	0.612	24
ParticleSep	1.000	0.880	0.936	25
average / total	0.745	0.730	0.736	74

Table 2: Performance metrics for the classifier based on the hand-labelled test data.

All three metrics are listed in Table 1 in addition to the number of instances per class. The difficulty of the classifier to distinguish samples with WatershedSep label causes the relatively low F_1 score for this class.

4.2. Validation on the hand-labelled data

The validation of the classification model is based on the hand-labelled data and should answer the question if features designed for the simulated particle pairs are directly applicable to those from a real battery cell.

The entire dataset comprises 294 samples, which leads to a test set of 74 instances. Here, the best hyperparameters turned out to be $\alpha = 0.0001$ and $m = 19$.

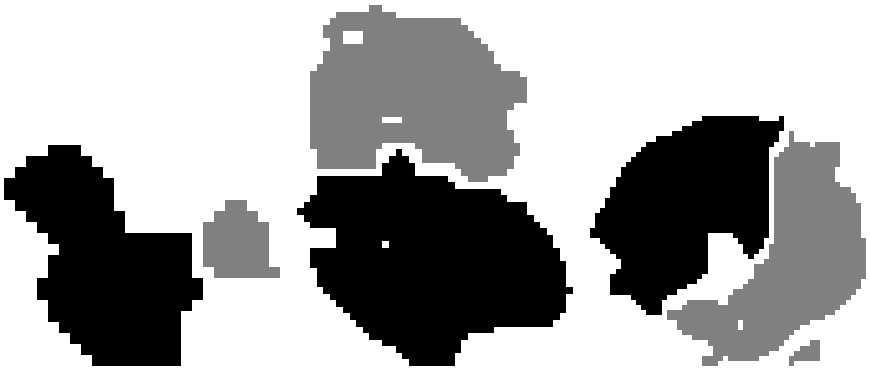
The accuracy over all classes is 73.0%. As can be seen in Figure 10b the classifier is able to clearly detect ParticleSep pairs, but about one third of the Broken-instances are mistaken for WatershedSep-particle pairs and vice versa. This is still roughly twice as good as a random prediction.

Precision and recall are shown in Table 2. As expected, the F_1 score for ParticleSep is higher than that for the other classes.

4.3. Discussion

Of course, it is not surprising that the accuracy of the algorithm is better for simulated data compared to real data, as the simulated data are much simpler and in a sense more regular, which simplifies the learning procedure of the classifier. However, the still comparable accuracies and the similarity of the two confusion matrices shows that the choice of features is reasonable also for real-world data.

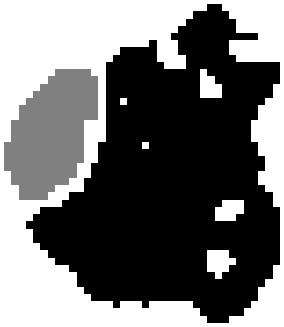
As can be seen in Figure 11, it is even for a human eye quite hard to distinguish WatershedSep from Broken particle pairs only by looking at 2D-slices of the segmented image. The reason that performances for the Broken-instances differ for real and simulated data could be the simplified breakage



(a) true: Broken, predicted: WatershedSep

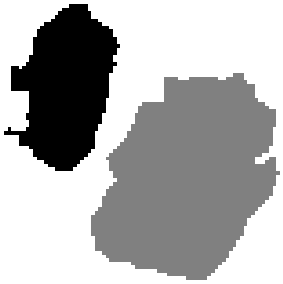
(b) true: Broken, predicted: WatershedSep

(c) true: WatershedSep, predicted: Broken



(d) true: Broken, predicted: Broken

(e) true: WatershedSep, predicted: WatershedSep



(f) true: ParticleSep, predicted: ParticleSep

Figure 11: Six examples of particle pairs with their true and predicted class label.

algorithm which does not capture all properties of real broken particle pairs. Moreover, in many cases, cracks do not separate the particle completely and are then split by the watershed algorithm, which results in a WatershedSep particle pair that exhibits characteristics of both classes, WatershedSep and Broken. Corresponding examples are depicted in Figure 11b and 11c.

5. Conclusion and outlook

In the present paper we used machine learning to detect cracks in the anode of a lithium-ion battery after thermal runaway. The classifier considers pairs of particles and distinguishes three causes for their separation: breakage during the thermal runaway, image segmentation and disjointness in the pristine cell. The decision is mostly based on the shape of the gap between the particles. We

facilitated the classifier development by using simulated anode material which generates an arbitrary amount of correctly labelled sample data.

The validity of the classifier is tested by applying the methodology to hand-labelled data from a real electrode. For this dataset, an overall accuracy of 73% is achieved.

We found that the main difficulty for the classifier is to differentiate between particles that touch each other and were split by the watershed segmentation algorithm, and pairs of two parts of a broken particle. This might be due to the fact that some cracks do not separate the particle completely, but lead to the particle being completely split by the watershed algorithm. Thus, for further developments, it is interesting to enhance the algorithm such that the WatershedSep class is divided in two subclasses, one containing particle pairs that touch each other, and one containing particles that are already slightly cracked, but not completely broken.

In the future, we expect algorithms of this nature to become more commonplace in the analysis of battery tomography data, enhancing the automated segmentation process and contributing to our understanding of microstructure evolution effects as a result of cracking due to calandering or lithiation stress.

Acknowledgement

This work was partially funded by BMBF under grant number 05M13VUA in the programme “Mathematik für Innovationen in Industrie und Dienstleistungen”.

References

[1] D. P. Finegan, M. Scheel, J. B. Robinson, B. Tjaden, I. Hunt, T. J. Mason, J. Millichamp, M. D. Michiel, G. J. Offer, G. Hinds, D. J. Brett, P. R. Shearing, In-operando high-speed tomography of lithium-ion batteries during thermal runaway, Nature communications 6 (2015) 6924. doi:10.1038/ncomms7924.

[2] D. P. Finegan, M. Scheel, J. B. Robinson, B. Tjaden, M. D. Michiel, G. Hinds, D. J. L. Brett, P. R. Shearing, Investigating lithium-ion battery materials during overcharge-induced thermal runaway: an operando and multi-scale X-ray CT study, Physical Chemistry Chemical Physics 18 (2016) 30912–30919. doi:10.1039/c6cp04251a.

[3] J. Jiang, J. Dahn, Effects of particle size and electrolyte salt on the thermal stability of Li0.5CoO2, Electrochimica Acta 49 (16) (2004) 2661–2666. doi:10.1016/j.electacta.2004.02.017.

[4] J. Geder, H. E. Hoster, A. Jossen, J. Garche, D. Y. Yu, Impact of active material surface area on thermal stability of LiCoO2 cathode, Journal of Power Sources 257 (2014) 286–292. doi:10.1016/j.jpowsour.2014.01.116.

[5] L. Gillibert, D. Jeulin, 3D reconstruction and analysis of the fragmented grains in a composite material, Image Analysis & Stereology 32 (2) (2013) 107–115. doi:10.5566/ias.v32.p107-115.

[6] C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[7] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, 2nd Edition, Springer, 2009.

[8] J. Feinauer, T. Brereton, A. Spettl, M. Weber, I. Manke, V. Schmidt, Stochastic 3D modeling of the microstructure of lithium-ion battery anodes via Gaussian random fields on the sphere, Computational Materials Science 109 (2015) 137–146. doi:10.1016/j.commatsci.2015.06.025.

[9] S. Beucher, F. Meyer, The morphological approach to segmentation: The watershed transformation, in: E. Dougherty (Ed.), Mathematical Morphology in Image Processing, CRC Press, 1992, pp. 433 – 481.

[10] R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2011.

[11] J. Hoshen, R. Kopelman, Percolation and cluster distribution. I. cluster multiple labeling technique and critical concentration algorithm, Physical Review B 14 (8) (1976) 3438 – 3445. doi:10.1103/physrevb.14.3438.

[12] A. Spettl, R. Wimmer, T. Werz, H. Heinze, S. Odenbach, C. Krill, V. Schmidt, Stochastic 3D modeling of ostwald ripening at ultra-high volume fractions of the coarsening phase, Modelling and Simulation in

Materials Science and Engineering 23 (6) (2015) 065001. doi:10.1088/0965-0393/23/6/065001.

[13] J. Mayer, V. Schmidt, F. Schweiggert, A unified simulation framework for spatial stochastic models, Simulation Modelling Practice and Theory 12 (2004) 307 – 326. doi:10.1016/j.simpat.2004.02.001.

[14] J. Feinauer, A. Spettl, I. Manke, S. Strege, A. Kwade, A. Pott, V. Schmidt, Structural characterization of particle systems using spherical harmonics, Materials Characterization 106 (2015) 123–133. doi:10.1016/j.matchar.2015.05.023.

[15] S. N. Chiu, D. Stoyan, W. S. Kendall, J. Mecke, Stochastic Geometry and Its Applications, 3rd Edition, Wiley, 2006.

[16] A. Okabe, B. Boots, K. Sugihara, S. N. Chiu, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, 2nd Edition, Wiley, 2000.

[17] R. Diestel, Graph Theory, 4th Edition, Graduate Texts in Mathematics, Springer, 2010.

[18] G. B. Arfken, H.-J. Weber, F. E. Harris, Mathematical Methods for Physicists: A Comprehensive Guide, 7th Edition, Elsevier, 2012.

[19] J. Nocedal, S. J. Wright, Numerical Optimization, 2nd Edition, Springer Series in Operations Research and Financial Engineering, Springer, 2006.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825 – 2830.