Microstructure Classification of Ultra High Steel Carbon through SVM- based Learning Algorithms

A Thesis

submitted by

MADHURIMA MAHAJAN

in partial fulfilment of the requirements
for the award of the degrees of

BACHELOR OF TECHNOLOGY &
MASTER OF TECHNOLOGY

DEPARTMENT OF METALLURGICAL AND MATERIALS ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

JUNE 2021

THESIS CERTIFICATE

This is to certify that the thesis entitled **Microstructure Classification of Ulta High Steel Carbon through SVM-based Learning Algorithms** submitted by **Madhurima Mahajan (MM16B030)**, to the **IIT Madras** for the award of the degree of the **Dual degree in Metallurgical and Materials Engineering** is a bonafide record of research work carried out by her under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Satyesh Kumar Yadav
Research Guide
Assistant Professor
Dept. of Metallurgical and Materials Engineering
IIT Madras, 600 036

Dr. Mansi Sharma
Research Guide
INSPIRE Faculty
Dept. of Electrical Engineering
IIT Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

Firstly, I would like to express my gratitude to my guide, Dr.Satyesh Kumar Yadav, Metallurgical and Materials Engineering Department, IIT Madras, and my co-guide, Dr. Mansi Sharma, Electrical Engineering Department, IIT Madras for giving me an opportunity to work under their guidance. I want to thank them for their patience, motivation and support during the course of the project. I would not have been able to complete this project without their assistance.

I also want to thank Dr. Gandham Phanikumar for motivating and guiding me throughout my undergraduate years. His tremendous support has been vital to my growth.

I take this opportunity to thank all the professors of Metallurgical and Materials department for all the support that they have provided me for the last five years.

ABSTRACT

KEYWORDS: Ultra High Steel Carbon, Microstructure, Convolutional Neural Networks, Support Vector Machine, Classification, Micrograph

One of the goals of Materials Informatics (in general) and Image Driven Machine Learning (in particular) is to extract quantitative data from micrographs to characterize microstructural characteristics efficiently. Towards this goal, we have presented a case study on Ultra High Carbon Steel(UHCS) where we investigate different SVM learning algorithms to classify microstructures from image data.

We have built a microstructure classification model based on the UHCS dataset that classifies a given microstructure into four labels: "Pearlite," "Spheroidite," "Carbide Network," and "Widmanstätten". The dataset comprises images taken at a wide range of magnifications. Feature extraction and dimensionality reduction were implemented before training SVM-based learning algorithms. In the feature extraction step, features from different layers of the pre-trained VGG16 model were evaluated based on their accuracy. In the classification step, we use a two-stage pipeline consisting of binary classification and voting. Twin Support Vector Machine and Least Squares Support Vector Machine are used for binary classification. We then compare the performance of these classifiers with the classical SVM algorithm. Results demonstrate that Twin Support Vector Machine with linear kernel performed the best among all other classifiers considered. Using t-SNE, a visualization technique, we show graphical methods to understand and interpret high-dimensional microstructure representations and observed results.

ABBREVIATIONS

ML	Machine Learning
SVM	Support Vector Machine
TWSVM	Twin Support Vector Machine
LSSVM	Least Square Support Vector Machine
UHCS	Ultra High Steel Carbon
CNN	Convolutional Neural Network

Contents

ACKNOWLEDGEMENTS **1**

ABSTRACT **2**

ABBREVIATIONS **3**

LIST OF TABLES **7**

LIST OF FIGURES **8**

1 INTRODUCTION **8**

2 LITERATURE SURVEY **10**

2.1 Material Informatics 10

2.2 Data-driven Approach for Microstructure based studies 10

3 PROBLEM STATEMENT **12**

3.1 Objective 12

3.2 Approach 12

4 SUPPORT VECTOR MACHINES **14**

4.1 Introduction 14

4.2 Twin Support Vector Machine 15

4.2.1 TWSVM Formulation 16

4.3 Least Squares Support Vector Machines 17

4.3.1 LSSVM Formulation 17

4.4 Kernel Trick 18

5 METHODOLOGY **20**

5.1 Ultra High Carbon Steel Dataset 20

5.1.1 Image Processing 21

5.1.2 Data sampling 22

5.2 Microstructure Representation 22

5.2.1 Convolutional Neural Networks 23

5.2.2 Transfer learning with VGG16 24

5.2.3 Data Visualization 25

5.3 Multiclass Classification 26

5.3.1 One-vs-One technique 27

5.3.2 Voting 27

5.3.3 Evaluation Metrics 28

6 RESULTS AND DISCUSSION **29**

6.1 Classical Support Vector Machine 29

6.2 Twin Support Vector Machine 30

6.3 Least Squares Support Vector Machine 31

6.4 Model Comparison 32

6.5 Binary Misclassification 33

7 CONCLUSION **35**

7.1 Summary 35

7.2 Future Work 36

8 FUNCTIONS FOR MULTI-CLASS CLASSIFICATION **37**

List of Tables

1 Schedule of primary microconstituent labels in the UHCS micrograph dataset 20

2 Data distribution for microstructure classification 22

3 Errors associated with each layer in VGG-16 architecture 25

4 Error scores for pairwise two-label classifiers for SVM 29

5 Error scores for multi-label voting classifier 29

6 Error scores for pairwise two-label classifiers for TWSVM without kernel. . 30

7 Error scores for pairwise two-label classifiers for TWSVM. with linear kernel 30

8 Error scores for multi-label voting classifier for TWSVM 30

9 Error scores for pairwise two-label classifiers for LSSVM with linear kernel 31

10 Error scores for pairwise two-label classifiers for LSSVM with polynomial kernel 31

11 Error scores for multi-label voting classifier for LSSVM 31

12 Final results 35

List of Figures

1 Flowchart of classification model 13

2 Hyperplanes in SVM 14

3 Hyperplanes in TWSVM 15

4 Kernel Trick 18

5 Primary microstructure constituents in the UHCS dataset: (a) spheroidized cementite with red and yellow frames indicating image regions used for feature extraction in the UHCS-600 and UHCS-2400 datasets, (b) carbide network microstructure, (c) pearlite, (d) pearlite containing spheroidized cementite, (e) Widmanstätten cementite, and (f) martensite and/or bainite [4] 21

6 Original micrograph 21

7 Processed micrograph 21

8 Put your caption here 21

9 Operating principle of CNN architecture for surface crack detection [15] . 23

10 VGG-16 architecture [4]; the conv 5-3 layer used for feature extraction is represented by the rightmost blue layer in conv5 24

11 t-SNE visualization for ultra-high steel carbon micrograph dataset 26

12 Voting classifier 27

13 Comparison of classifiers with respect to their voting scores 32

14 t-SNE visualization for Ultra High Steel Carbon micrograph dataset 33

a Training set 33

b Test set 33

CHAPTER 1

INTRODUCTION

Microstructure image data are rich in the morphology and inferred composition of constituent phases. They can provide insights on microstructure formation, structure-property relationships, the impact of processing conditions, and mechanisms that control material behavior and performance. Many challenges exist related to the analysis of microstructure images. These challenges can arise from limited domain knowledge and skill, various forms of image data (e.g., optical and electron microscopy),domain-specific limitations to image analysis techniques, and more.

The Materials Genome Initiative aims to expedite materials discovery and design using computational models and data science approaches. The development of Artificial intelligence methods in Computer Vision has opened the opportunity for computationally guided experiments and repeatable image data analysis. This work demonstrates the application of Computer Vision and SVM-based learning algorithms for microstructure recognition of Ultra High Steel Carbon.

UHCS is low alloyed carbon steel having 1.25-2% carbon approximately. They exhibit remarkable structure properties: have high strength, sharpness, and resilience, and are widely used in automotive and structural applications. Damascus steel and Japanese sword steels are Ultra High Carbon Steels. The strength of UHCS is attributed to fine-grained spheroidized carbides [16].

In this study, we use the UHCS dataset by Carnegie Mellon University, based on the work of Hecht et al. [12; 11]. The dataset contains scanning electron microscopy micrographs of UHCS subjected to different temperatures. It includes several complex microconstituents often seen in UHCS and other similar alloy systems, providing challenging real-world microstructure informatics problem.

In this work, we use the UHCS dataset to build a microstructure classification model using advanced SVM learning algorithms, particularly Twin Support Vector Machine [14] and Least Squares Support Vector Machine for classification [21]. We use a pre-trained VGG16 convolutional neural network to extract features from the image dataset, which acts as an input to the

ML model. We evaluate the features extracted from different layers of the VGG16 model using an SVM classifier for an accurate representation of the image. We then compare the performance of all the classifiers with the classical SVM model. We complement this understanding by visualizing the high-dimensional distributions of each microstructure representation using the unsupervised dimensionality reduction method t-SNE [22] .

In this work, we primarily test the efficacy of SVM-based learning algorithms over classical SVM algorithm for the UHCS microstructure recognition task.

CHAPTER 2

LITERATURE SURVEY

2.1 Material Informatics

Due to advancements in data science and AI, material informatics has grown dramatically over the past several years. Krishna Rajan [19] pioneered the subject of materials informatics and is an early proponent of a data-driven approach in materials research. The wide range of applications of ML in materials science arises because of the large data sets available. The largest existing database based on experimental results from materials has $5x10^5$ data records [25]

Some applications of ML in Materials Science include alloy design, predicting atomistic potentials, structural and thermodynamic properties like thermal expansion coefficient, predicting crack growth, microstructure classification, and more.

2.2 Data-driven Approach for Microstructure based studies

Several Computer Vision and ML tools offer new approaches to encode visual features from microstructure image data. CNNs are well suited for problems that require finding spatial, nonlinear relationships between input and a given response variable of interest. They have been successfully used in applications such as microstructure classification based on scanning electron microscopy (SEM) micrographs and determination of material properties based on microstructure [2; 24]

Pierson, Rahman, and Spear [18] have presented the application of ML in the microstructure-sensitive evolution of a 3D crack surface in a polycrystalline alloy. In this technique, CNN model is used to establish spatial relationships between micromechanical/microstructural features, uncracked microstructure and the 3D crack path. Ml methods have been used to predict the mechanical properties from 3D microstructures using features such as the crystallographic grain orientation as an input [9; 17].

ML combined with image feature extraction techniques are widely used in image classification tasks. Aritra Chowdhury [3] has performed two classification tasks- first, to detect dendritic morphologies and second, to identify different cross-sectional views (longitudinal or transverse) from micrographs identified as depicting dendrites. In this study, various combinations of feature extraction methods- Visual bag of words, texture and shape statistics, pre-trained CNNs and classifiers- Support Vector Machine, Random Forest, Nearest Neighbors used for microstructure classification were used and compared. In another example, two-stage ML pipeline comprising of classification and segmentation steps was executed with Ti-6Al-4V alloy micrographs [1]. Here, CNN is used for microstructure classification and segmentation to estimate the area fraction of equiaxed grains and dominant α variant.

CHAPTER 3

PROBLEM STATEMENT

Material characterization plays an essential role in discovering and designing new materials. Traditionally, human experts have evaluated microstructure images to understand the micrographs and study structure-property relationships. This manual process of interpreting microstructure can lead to bias and error. This bias can be caused by different factors, like an individual's background, education, and experiences [13]

Several methods to classify microstructure images have been tested. However, there is a scope to improve the classification performance using new advanced ML models. In this work, we attempt to solve this problem using Support Vector Machine based classifiers like Twin support vector machine and least-squares Support Vector Machine. Machine learning classification methods like Support Vector Machine, Random Forest, and Nearest Neighbor have already been used in microstructure classification by Aritra, Elizabeth, Bulent, and Daniel (2016), [3] and Brain and Elizabeth (2015), [6].

3.1 Objective

The project aims to implement SVM-based learning algorithms to improve the performance of microstructure recognition over existing methods.

3.2 Approach

The approach for microstructure classification is summarized in Figure 1. Feature extraction is the first step in the process of classification. Feature extraction detects features in an image using computer vision algorithms. However, the classification task doesn't require all the features. Feature selection picks important and distinct features of all features detected and thus, helps in dimensionality reduction. Feature selection also makes the process of training the model faster. The selected features are used in training the model. We use SVM-based classifiers like Twin Support Vector Machine and Least Squares Support Vector Machine. The model then predicts microstructure labels.

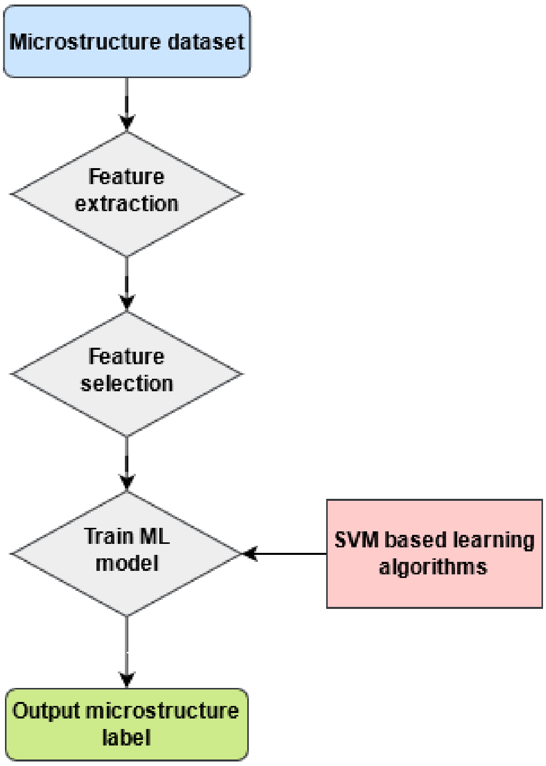


Figure 1: Flowchart of classification model

CHAPTER 4

SUPPORT VECTOR MACHINES

4.1 Introduction

SVM is a supervised machine learning technique used for classification as well as regression. In classification, the SVM identifies a hyperplane in an n-dimensional space (N-the number of features) that optimally splits a dataset into two classes. A hyperplane is a decision boundary which assigns the points on either side to two different class. Although there are numerous possible hyperplanes, SVM finds the hyperplane with the highest margin, i.e., the greatest distance between data points from both classes. This is illustrated in Figure 2.

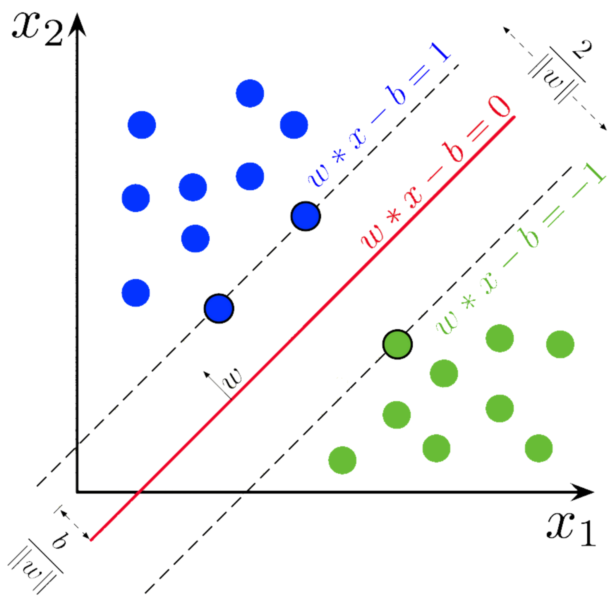


Figure 2: Hyperplanes in SVM

Consider binary classification problem with classes: +1 and -1. with m_1 and m_2 training points in n-dimensional space R^n belonging to +1 and -1 classes respectively. Let matrix $A \in R^{m_1xn}$ and $B \in R^{m_2xn}$ represent training points of class +1 and -1 respectively. For linearly separable data, SVM attempts to find a hyperplane of the form $w^T + b = 0$. According to the principle, we should determine $w \in R^n$, $b \in R$ so that the margin, i.e the distance between the supporting hyperplanes $w^T + b = 1$ and $w^T + b = -1$ is maximum. A directed calculation yields margin equal to $\frac{2}{||w||}$, where $||w||$ is given by $||w||^2 = w^T w$. The maximum mrgin SVM is obtained by solving the following optimization problem.

$$
\begin{aligned}
\mathit{Min} \quad & \frac{||w||}{2} \\
\mathit{subject\ to} \quad & Aw + eb \geq 1 \\
& Bw + eb \leq -1
\end{aligned}
\tag{4.1}
$$

4.2 Twin Support Vector Machine

TWSVM is a new emerging machine learning technique that can be used for both regression and classification problems. TWSVM research is still in its early stages. Still, it has emerged as a hot research subject in machine learning because of its quicker training time and improved classification performance.

[14] proposed TWSVM model that aims to find nonparallel planes that best fits the orientation of clustered data. Unlike SVM that constructs two parallel hyperplanes, TWSVM generates two nonparallel planes- positive hyperplane and negative hyperplane. Each hyperplane represents one of the two classes. Like the concept of maximum margin in SVM, in TWSVM, the hyperplane representing a particular class has to be as far as possible from the other class samples. It should remain close to the representative class's data points. A new data point gets allocated to a class based on its closeness to the two nonparallel hyperplanes. Figure 3 depicts the two nonparallel planes representing two classes.

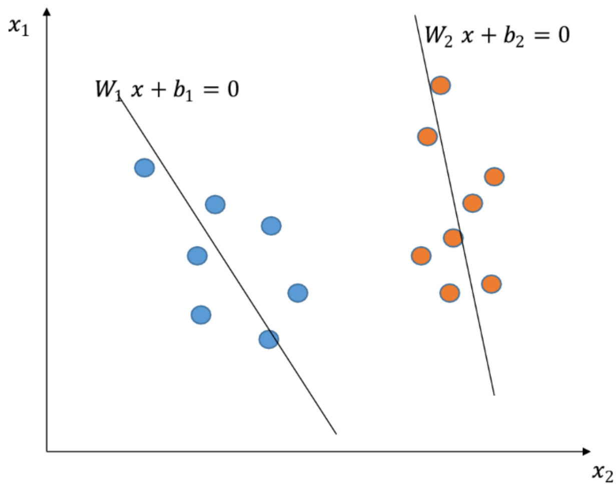


Figure 3: Hyperplanes in TWSVM

These hyperplanes are generated by solving two quadratic programming problems (QPP's). Each QPP corresponds to each class, where it finds the corresponding hyperplane. Unlike SVM, where all data points are included in constraints, in TWSVM, data points are distributed between two QPP's. The patterns of one class act as constraints for other QPP and vice-versa.

4.2.1 TWSVM Formulation

Equation of hyperplane in n-dimensional space is of the form $wx^T + b = 0$. The TWSVM finds two hyperplanes of the form: $w_1x^T + b_1 = 0$ and $w_2x^T + b_2 = 0$. The formulation of TWSVM can be expressed as:

$$\begin{array}{ll} \textit{(TWSVM1)} & \textit{Min}\ \frac{1}{2}(Aw_1+e_1b_1)^T(Aw_1+e_1b_1)+c_1e_2^T\xi_1 \\ \textit{subject to} & -(Bw_1 + e_2b_1) + \xi_1 \geq e_2, \qquad \xi_1 \geq 0 \end{array} \tag{4.2}$$

$$\begin{array}{ll} \textit{(TWSVM2)} & \textit{Min}\ \frac{1}{2}(Bw_2+e_2b_2)^T(Bw_2+e_2b_2)+c_2e_1^T\xi_2 \\ \textit{subject to} & (Aw_2 + e_1b_2) + \xi_2 \geq e_1, \qquad \xi_2 \geq 0 \end{array} \tag{4.3}$$

where c_1, $c_2 \geq 0$ are parameters and e_1 and e_2 are vectors of ones of appropriate dimensions, and $w_1 \in R^n$, $w_2 \in R^n$, $b_1 \in R$ and $b_2 \in R$, and ξ_1 and ξ_2 are the slack variable.

The first term in the objective function of (TWSVM1) given at (4.2) is the sum of squared distances of points belonging to class +1. Thus the minimization of this term will find the hyperplane close to points of class +1. The constraints in (4.2) ensure that the hyperplane $x^Tw_1 + b_1 = 0$ is at a distance of at least one from points belonging to class -1. Only samples of class -1 contribute to the constraints of this problem. ξ_1 represents a vector of error variables that measures the error whenever the hyperplane is closer to points of class -1 than the minimum distance of 1. The second term minimizes error variables to avoid misclassification due to points of class -1. The c_1 parameter acts like trades off between the terms in the objective function of (4.2). A similar explanation can be given for objective function and constraints in (4.3).

Solving these QPPs results in the equation of two hyperplanes. (4.2) attempts to find the hyperplane $x^Tw_1 + b_1 = 0$ where points belonging to class +1 are clustered around it. Similarly in (4.3), points belonging to class -1 get clustered around $x^Tw_2 + b_2 = 0$. The idea is to solve two QPP's of smaller size instead of one large QPP in SVM. This makes TWSVM almost four times faster than SVM [7]. Several extensions to the Twin SVM have been proposed. Shifei Ding has given a detailed review of the variants and applications of Twin SVM in his paper [7].

4.3 Least Squares Support Vector Machines

Suykens and Vandewalle [21] proposed the Least-squares version of SVM. In LSSVM, one finds the solution by solving linear equations instead of quadratic programming problem for classical SVMs. The optimization problem in classical SVM algorithm has inequality constraints, and LSSVM simplifies the problem with equality constraints. In the least-squares method, the optimization problem tries to minimize the sum of squares of error variables, i.e., $\sum e_i^2$ where e_i is the difference between the dependent variable's actual value and predicted value.

4.3.1 LSSVM Formulation

Consider the training data set comprising of N input vectors x_i (i=1 to N) with corresponding labels y_i, where $y_i \in \{-1, +1\}$ and the value of new input vector x is obtained based on the sign of y(x), where

$$y(x) = w^T \phi(x) + b \tag{4.4}$$

$y(x) = 0$ or $w^T \phi(x) + b$ is a decision boundary for binary classification. Thus, two sides separated by decision boundary have values of $y(x)$ positive or negative. All the input points with $y(x) \geq 0$ belong to class +1 and get clustered to one side of the decision boundary. Similarly, input points having $y(x) \leq 0$ belong to class -1 and get clustered on another side of the decision boundary. Therefore, the new input point gets assigned to class +1 when $y(x) \geq 0$ and class -1 when $y(x) \leq 0$. The least-squares version of the SVM method is derived by reformulating the minimization problem as follows-

$$\begin{aligned} \textit{Min J(w,b,e)} = \quad & \tfrac{1}{2} w^T w + \tfrac{1}{2}\gamma \sum_{i=1}^{N} e_i^2 \\ \textit{subject to} \quad & y_i[w^T \phi(x_i) + b] = 1 - e_i, \quad \forall i \end{aligned} \tag{4.5}$$

where γ is a tuning parameter, $b \in R$, and ϕ is a nonlinear function that maps the input space into a higher dimensional space. To solve this optimization problem, we construct the Lagrangian function with Lagrange multipliers α_i as shown in (4.6). Solving equations derived from optimality constraints yields the decision boundary.

Lagrangian:

$$L(w, b, e; \alpha) = J(w, b, e) - \sum_{i=1}^{N} \alpha_i \{y_i[w^T \phi(x_i) + b] - 1 + e_i\} \tag{4.6}$$

Conditions for optimality:

$$\begin{aligned}
\frac{\partial L}{\partial w} &= 0 & \rightarrow \quad & w = \sum_{i=1}^{N} \alpha_i y_i \phi(x_i) \\
\frac{\partial L}{\partial b} &= 0 & \rightarrow \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\
\frac{\partial f}{\partial e_i} &= 0 & \rightarrow \quad & \alpha_i = \gamma e_i, \quad \forall i \\
\frac{\partial f}{\partial \alpha_i} &= 0 & \rightarrow \quad & y_i[w^T \phi(x_i) + b] - 1 + e_i = 0, \quad \forall i
\end{aligned} \tag{4.7}$$

4.4 Kernel Trick

The SVM algorithm finds the hyperplane that linearly separates different classes. In 1D, a hyperplane represents a point. In 2D, it is a line, and in 3D, it is a plane. However, sometimes the data is not linearly separable. In real-world problems, the data is randomly distributed and, thus, is non-linear. The SVM algorithm finds it challenging to classify the non-linear data.

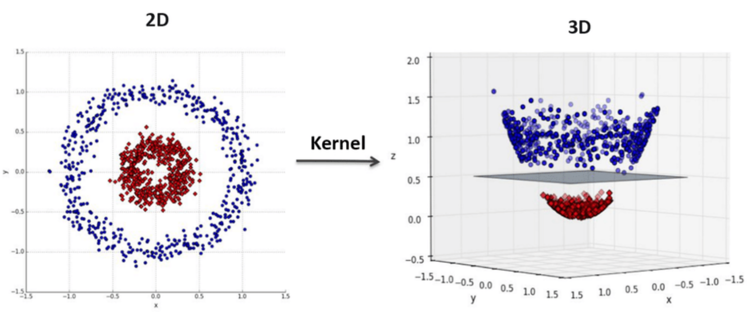


Figure 4: Kernel Trick

The kernel trick projects the non-linear data onto a higher dimensional space where the data can be classified linearly by constructing a hyperplane. This is illustrated in Figure 4. Various kernels can be used like radial basis function, linear kernel, polynomial kernel, etc., which should be chosen depending upon the problem's geometry. In this project, we have used linear and polynomial kernels and compared the results.

$$K(x_i, x_j) = \begin{cases} x_i x_j, & \text{(Linear kernel)} \\ (x_i x_j + 1)^p, & \text{(Polynomial kernel)} \end{cases} \tag{4.9}$$

CHAPTER 5

METHODOLOGY

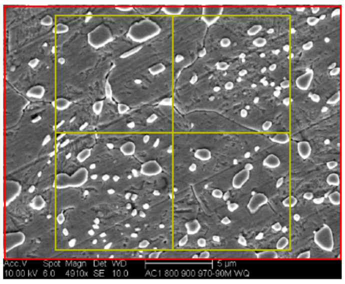
5.1 Ultra High Carbon Steel Dataset

As mentioned, in this project, we use the UHCS dataset by Carnegie Mellon University [5] [10]. The UHCS dataset consists of 961 SEM micrographs of commercial UHCS subjected to a range of heat treatments by Hecht et al. [12] [11]. These micrographs include secondary electron (SE) and back-scattered electron (BSE) images taken at a wide range of magnifications. All 961 images are labeled with their primary microstructure constituents. There are seven distinct primary constituents. Majority of these micrographs have only one primary constituent- spheroidite (Figure 5a), carbide network (Figure 5b), and pearlite (Figure 5c). However, few micrographs contain two primary constituents like pearlite having spheroidite (Figure 5d), pearlite having widmanstätten, widmanstätten cementite (Figure 5e), and martensite (Figure 5f).

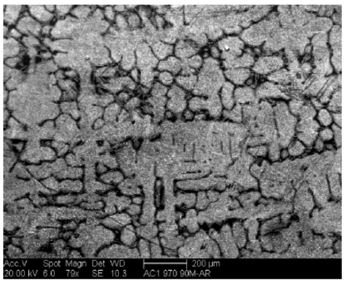
Primary constituents	Number of micrographs
Spheroidite	374
Carbide Network	212
Pearlite	124
Pearlite + Spheroidite	107
Widmanstätten cementite	81
Pearlite + Widmanstätten	27
Martensite/Bainite	36

Table 1: Schedule of primary microconstituent labels in the UHCS micrograph dataset

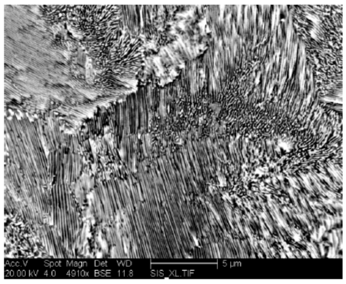
Table 1 shows the distribution of each of these primary microconstituent labels. For the classification experiment, we consider four classes namely: "Spheroidite", "Carbide network", "Pearlite" and "Widmanstätten".



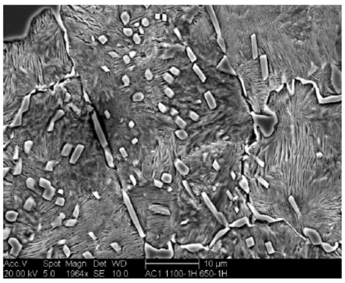
(a)



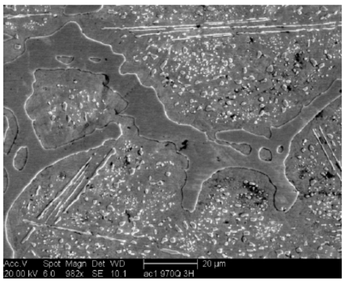
(b)



(c)



(d)



(e)

(f)

Figure 5: Primary microstructure constituents in the UHCS dataset: (a) spheroidized cementite with red and yellow frames indicating image regions used for feature extraction in the UHCS-600 and UHCS-2400 datasets, (b) carbide network microstructure, (c) pearlite, (d) pearlite containing spheroidized cementite, (e) Widmanstätten cementite, and (f) martensite and/or bainite [4]

5.1.1 Image Processing

The black section at the bottom of all 791 micrographs containing experiment-specific information was removed and micrographs of 645 x 522 pixels were cropped to 645 x 484 pixel images as shown in Figure 8.

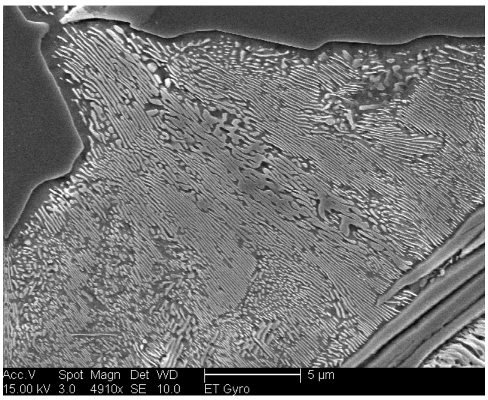


Figure 6: Original micrograph

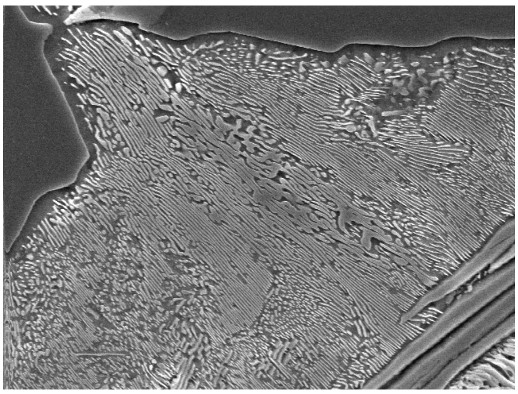


Figure 7: Processed micrograph

Figure 8: Put your caption here

5.1.2 Data sampling

The processed micrographs were further spilt into train and test set as shown in Table 2.

Primary constituents	Train set	Test set	Total
Spheroidite	100	274	374
Carbide Network	100	112	212
Pearlite	100	24	124
Widmanstätten	60	21	81

Table 2: Data distribution for microstructure classification

5.2 Microstructure Representation

Any machine learning problem requires a numerical representation of the data. For image classification tasks, images are transformed into their equivalent numeric representation containing the relevant information necessary to characterize the image. The process of obtaining the numerical representation of images is called feature extraction. The general approach for image classification involves feature extraction and feature selection (dimensionality reduction) to compute feature vectors used for training, validating, and testing various classification models. Computer vision is a field of artificial intelligence that develops computational methods to extract information from images or videos. We use computer vision algorithms to perform feature extraction and feature selection tasks.

Feature extraction finds the 'interesting' part of a microstructure, like edges, globular regions, or corners. Features detected using computer vision algorithms are not necessarily semantically meaningful; however, they are pixel patterns that are mathematically repeatable and recognizable, thereby making the region a good feature. These features are encoded in the form of a vector, known as a feature vector. A feature vector groups the detected feature descriptors together to represent the image.

These feature vectors obtained have high dimensionality and can affect computational efficiency. Thus, feature selection is a dimensionality reduction method that reduces the length of the feature vector while retaining all the image information.

5.2.1 Convolutional Neural Networks

In deep learning, CNN is a class of deep neural networks, most commonly applied to analyze visual imagery [23]. As the name suggests, CNN is a special type of neural network that uses convolution operation for processing input data with 2D shape like images. Images are the 2D matrix of pixels on which CNN operates.

A CNN consists of an input layer, hidden layers, and an output layer. In CNN, hidden layers perform convolution operations; thus, they're called convolutional layers. Each convolutional layer learns specific features from the image dataset and generates a feature map. The convolutional layer output, i.e., feature map, is passed as an input to the next convolutional layer, and this repeats. This is followed by other layers such as pooling layers and fully connected layers. Initial convolutional layers learn simple features like edges, corners, etc., and the complexity of features learned increases with deeper convolutional layers. Figure 9 shows the CNN architecture for surface crack detection example [15]. Pooling layers reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. In Figure 9, max-pooling uses the maximum value of each local cluster of neurons in the feature map.

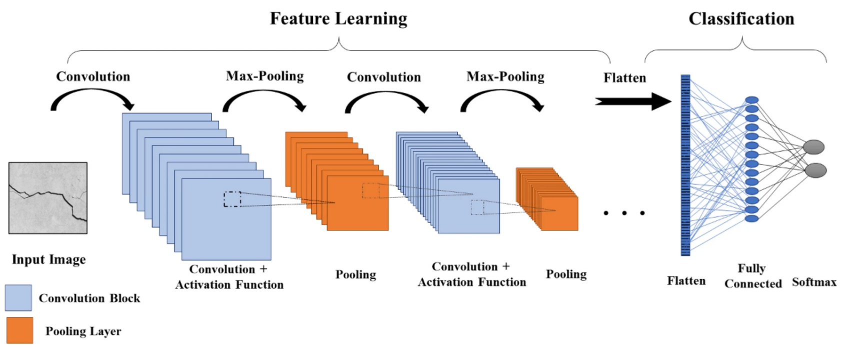


Figure 9: Operating principle of CNN architecture for surface crack detection [15]

CNN's have shown excellent performance at many computer vision tasks. This is because CNNs extract high-level image features by stacking multiple layers of neurons into convolution filters learned from annotated training images. In this work, we have used CNN representations because they are richer, more hierarchical, and effective than other feature extraction methods [3]. Training a CNN model is a notorious task; it requires extreme amounts of data. Thus, for practical purposes concept of 'transfer learning' can be used. Latest studies have shown that deep CNNs can be generalized to new datasets, even when a new task is not related to the original task [8]. Transfer learning is a machine learning approach where a pre-trained model developed for a particular task is reused as the starting point for a model on a second task.

5.2.2 Transfer learning with VGG16

In this study, we have used a pre-trained CNN model trained on the Imagenet database [20]. The high level features from the VGG16 CNN architecture are parameterized for object recognition on the ImageNet ILSVRC-2014 dataset [20]. The ImageNet database contains 1.2 million images with 1000 categories. The output of this architecture acts as input to the classification machine learning model. VGG16 has 14 convolution layers arranged into five blocks delineated by pooling layers, followed by two fully connected layers of 4096 neurons each, and a final 1000-class classification layer. As the VGG16 works on color images, we preprocess each micrograph by replicating the raw grayscale image in each color channel of a new RGB image and subtracting the average intensity of the ImageNet training set for each channel, as recommended by [37]. We used the publicly available parameters provided by the VGG group[37] without any fine-tuning. The VGG16 network architecture is shown in Figure10

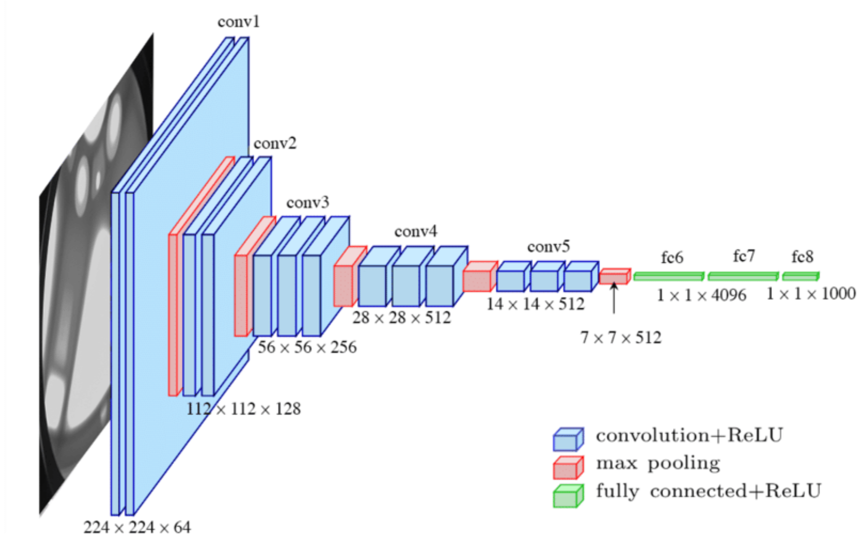


Figure 10: VGG-16 architecture [4]; the conv 5-3 layer used for feature extraction is represented by the rightmost blue layer in conv5

In this experiment, features extracted from conv 1-2, conv 2-2, conv 3-3, conv 4-3, and conv 5-3 are compared by using an SVM classifier to pick the most accurate feature extracting layer. The error was calculated for all binary classification problems for every possible pair of primary microconstituents and model layers. 3 shows classification errors associated with features extracted from each layer. It can be observed that conv 4-3 and conv 5-3 layers both show low errors and thus, yield the most accurate set of features. This can be attributed to the fact that deeper layers in CNN provide more accurate image representations than features computed using layers lower in CNN network hierarchy [3]. However, breaking the tie, conv 5-3 layer was used to extract features.

	Spheroidite vs. Network	**Spheroidite vs. Pearlite**	**Spheroidite vs. Widmanstätten**	**Network vs. Pearlite**	**Network vs. Widmanstätten**	**Pearlite vs. Widmanstätten**
Conv 1-2	0.28	0.185	0.3	0.13	0.29375	0.15625
Conv 2-2	0.1	0.12	0.30625	0.07	0.06875	0.0875
Conv 3-3	0.03	0.03	0.20625	0.04	0.03125	0.05625
Conv 4-3	0.02	0.01	0.1875	0.01	0.00625	0.03125
Conv 5-3	0.025	0.01	0.1875	0	0.00625	0.0375

Table 3: Errors associated with each layer in VGG-16 architecture

5.2.3 Data Visualization

The best way to analyze the data for classification is by visualizing it. It is easy to visualize the data with two or three-dimensional data. The dimensionality gets higher in deep learning. Thus, we may need to visualize the inner feature maps to understand what's inside the network.

We have used the t-SNE (t-distributed Stochastic Neighbor Embedding) algorithm [22], a dimensionality reduction method to visualize the multidimensional data in lower dimensions.Figure 14a and Figure 14b show t-SNE visualization for micrographs in train and test set, respectively.

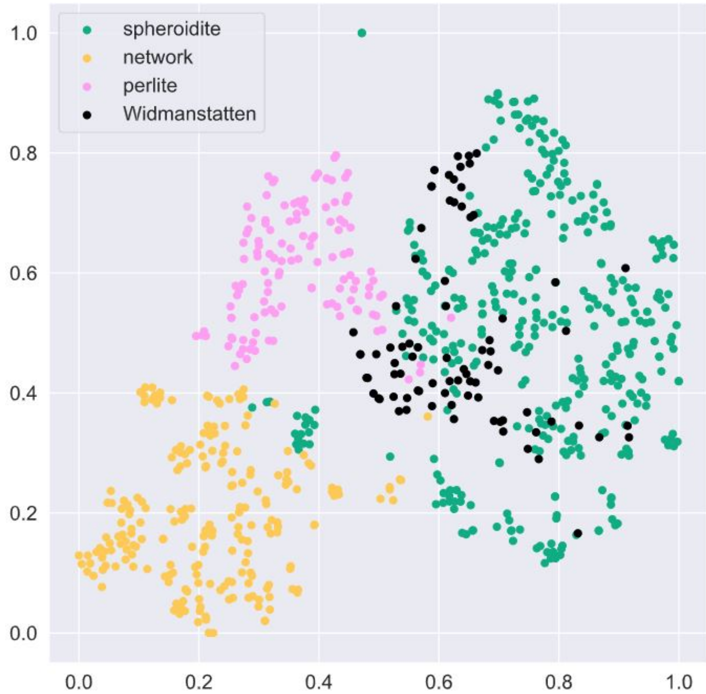


Figure 11: t-SNE visualization for ultra-high steel carbon micrograph dataset

In Figure 14, there are four clusters for each primary micro constituent. It can be observed that there is no overlap between spheroidite, network, and pearlite cluster regions, and thus, features corresponding to these clusters can be distinguished. The widmanstätten region overlaps with the spheroidite region in both train and test set. This is because of the similarity between some features of both these primary constituents. This resemblance affects the performance of binary classification of widmanstätten and spheroidite. We'll discuss this misclassification in detail in the next chapter.

5.3 Multiclass Classification

As the SVM algorithm is a binary classification algorithm and the task is a multi-class classification problem, we use one-vs-one reduction followed by the voting algorithm to transform the problem of multi-class classification into multiple binary problems, which are further trained using SVM models

5.3.1 One-vs-One technique

One-vs-One is a heuristic technique for using binary classification algorithms for multiclass classification problems. Here, we generate N(N-1)/2 binary classification models for each distinct pair of classes for the N-class instances dataset. Thus, the one-vs-one approach splits the dataset into one dataset for each class opposite every other class. Therefore, for the given problem, we generate six binary classification problems, as shown below.

- **Binary Classification Problem 1:** Pearlite vs. Spherodite
- **Binary Classification Problem 2:** Pearlite vs. Widmanstätten
- **Binary Classification Problem 3:** Pearlite vs. Network carbide
- **Binary Classification Problem 4:** Spherodite vs. Widmanstätten
- **Binary Classification Problem 5:** Network carbide vs. Spherodite
- **Binary Classification Problem 6:** Network carbide vs. Widmanstätten

5.3.2 Voting

Voting is a way to combine the predictions from multiple machine learning algorithms and predict an output class based on the highest probability of chosen class or highest number of predictions. The motivation behind this technique is that it exploits the different peculiarities associated with algorithms. Figure 8 clearly illustrates this method. Here, we've used hard voting, which predicts the class that received the highest number of votes, rather than soft voting, which predicts the class based on the highest probability.

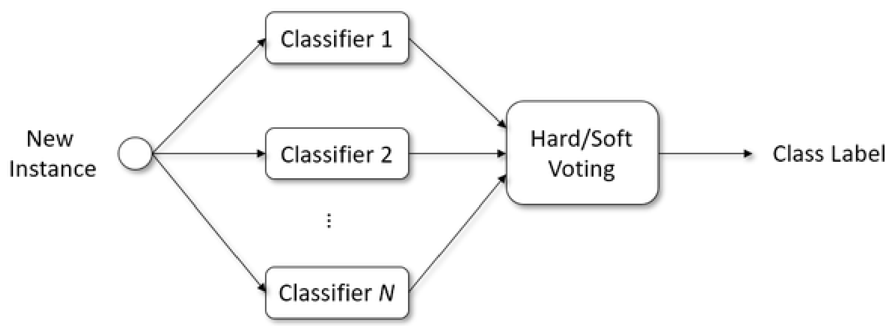


Figure 12: Voting classifier

5.3.3 Evaluation Metrics

After building multiclass classification model, the next step is to check the performance. We make predictions on test dataset and compute error for the voting classifier and corresponding binary classifiers. We calculate the error in following way,

$$\text{Voting score} = \frac{\text{Total no. of correct predictions by voting for given class}}{\text{Total no. of samples of given class}} \tag{5.1}$$

$$\text{Accuracy} = \frac{\text{Total no. of correct predictions}}{\text{Total no. of predictions}} \tag{5.2}$$

$$\text{Error} = \begin{cases} \text{1 - Accuracy,} & \text{(Binary classifiers)} \\ \text{1 - Voting score,} & \text{(Voting classifier)} \end{cases} \tag{5.3}$$

CHAPTER 6

RESULTS AND DISCUSSION

In this experiment, the multi-class classification is implemented in two steps- binary classification followed by voting. We have used different SVM-based learning algorithms in the binary classification step. Each section discusses the performance of each classifier and the parameters used. We then compare the results of each SVM-based classifier with the classical SVM algorithm. In each case, as discussed earlier, the features are extracted from conv 5-3 layer of the VGG16 model, and the same training and test datasets are used. The scores reported in the subsequent sections correspond to the performance of the classifier on the test dataset. The entire code is written in python.

6.1 Classical Support Vector Machine

After using SVM in the binary classification step, the classification error on test data is shown in Table 4. It can be inferred that the model can perfectly classify pearlite from micrographs containing pearlite and either of the other three primary micro constituents with 100 percent accuracy. Similarly, the model can best detect widmanstätten from micrographs containing widmanstätten and carbide network. The highest binary misclassification error is observed for spheroidite and widmanstätten with 0.18613 and 0.14286, respectively. A similar misclassification trend is observed for spheroidite and widmanstätten in the final step of voting in Table 5.

	Spheroidite vs. Network	Spheroidite vs. Pearlite	Spheroidite vs. Widmanstätten	Network vs. Pearlite	Network vs. Widmanstätten	Pearlite vs. Widmanstätten
Spheroidite	0.02920	0.01095	0.18613			
Network	0.01786			0.00893	0.01786	
Pearlite		0		0		0
Widmanstätten			0.14286		0	0.09524

Table 4: Error scores for pairwise two-label classifiers for SVM

Spheroidite	Network	Pearlite	Widmanstätten
0.21168	0.01786	0	0.19048

Table 5: Error scores for multi-label voting classifier

6.2 Twin Support Vector Machine

Here, we have applied two variations of the TWSVM model- one with linear kernel and other without kernel. As discussed earlier in 4.2.1, the TWSVM formulation comprises of c_1, c_2, ξ_1, and ξ_2 parameters. The values of these parameters chosen for this experiment are based on literature survey [14]. We have used 1, 1, 0.1, 0.1 values for c_1, c_2, ξ_1, and ξ_2 respectively. The results for TWSVM model for the both cases are summarised in Table 6, Table 7 and Table 8

	Spheroidite vs. Network	Spheroidite vs. Pearlite	Spheroidite vs. Widmanstätten	Network vs. Pearlite	Network vs. Widmanstätten	Pearlite vs. Widmanstätten
Spheroidite	0.01460	0.01095	0.10584			
Network	0.00893			0	0	
Pearlite		0.04167		0		0
Widmanstätten			0.23809		0	0.09529

Table 6: Error scores for pairwise two-label classifiers for TWSVM without kernel.

	Spheroidite vs. Network	Spheroidite vs. Pearlite	Spheroidite vs. Widmanstätten	Network vs. Pearlite	Network vs. Widmanstätten	Pearlite vs. Widmanstätten
Spheroidite	0.01095	0.01095	0.09124			
Network	0.00893			0	0	
Pearlite		0		0		0
Widmanstätten			0.28571		0	0.04762

Table 7: Error scores for pairwise two-label classifiers for TWSVM. with linear kernel

	Spheroidite	Network	Pearlite	Widmanstätten
Linear kernel	0.10219	0.00893	0	0.28571
Without kernel	0.11314	0.00893	0.04167	0.23809

Table 8: Error scores for multi-label voting classifier for TWSVM

Based on Table 6 and Table 7, we can infer that TWSVM with linear kernel performs better than TWSVM without kernel for most of the binary classification problems except for Widmanstätten. The same trend is observed for multi-label voting classification as per Table 8. Thus, we can infer that TWSVM with linear kernel performs better than classical TWSVM at classifying spheroidite, carbide network, and pearlite.

6.3 Least Squares Support Vector Machine

We tested the LSSVM model with linear and polynomial kernels. In this model, we assumed a γ value of one and a degree of a polynomial of two for the polynomial kernel. The results for both the kernels are summarized in Table 9, Table 10, and Table 11.

	Spheroidite vs. Network	Spheroidite vs. pearlite	Spheroidite vs. Widmanstätten	Network vs. Pearlite	Network vs. Widmanstätten	Pearlite vs. Widmanstätten
Spheroidite	0.05474	0.01460	0.16423			
Network	0.00893			0.01786	0	
Pearlite		0.04167		0		0
Widmanstätten			0.14286		0	0.04762

Table 9: Error scores for pairwise two-label classifiers for LSSVM with linear kernel

	Spheroidite vs. Network	Spheroidite vs. Pearlite	Spheroidite vs. Widmanstätten	Network vs. Pearlite	Network vs. Widmanstätten	Pearlite vs. Widmanstätten
Spheroidite	0.08394	0.02920	.11314			
Network	0.00893			0	0	
Pearlite		0		0.04167		0
Widmanstätten			0.14286		0.04762	0.04762

Table 10: Error scores for pairwise two-label classifiers for LSSVM with polynomial kernel

	Spheroidite	Network	Pearlite	Widmanstätten
Linear kernel	0.20438	0.01786	0.04167	0.09524
Polynomial kernel	0.19343	0.01786	0.04167	0.14286

Table 11: Error scores for multi-label voting classifier for LSSVM

As per the Table 11, no significant difference is observed between the results by LSSVM with linear and polynomial kernels.

6.4 Model Comparison

Based on voting error scores, we compare the multi-class classification performance of SVM-based classifiers to that of the traditional SVM algorithm. It is evident from Figure 13 that TWSVM with linear kernel performs best amongst all the other classifiers for spheroidite, carbide network, and pearlite. However, for widmanstätten, the error scores are significantly higher than other primary microconstituents. This can be attributed to the fact that the training and test set has limited samples of widmanstätten. In machine learning, data plays an important role in the training phase. More the data, the better the model can learn and perform. Due to the smaller dataset of widmanstätten, the VGG16- feature extraction model doesn't effectively learn all the describing features of widmanstatten microstructure. Therefore, the classifiers perform poorly in detecting widmanstätten. We explore the theory of misclassification in the next subsection to understand the observed results better.

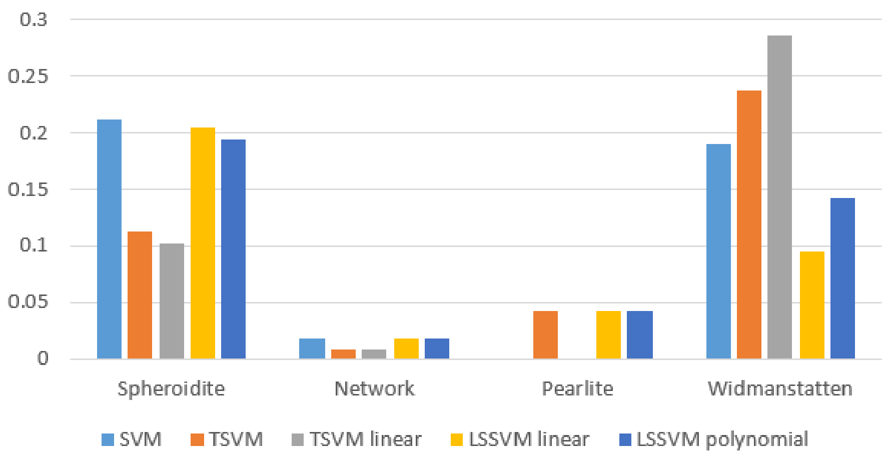
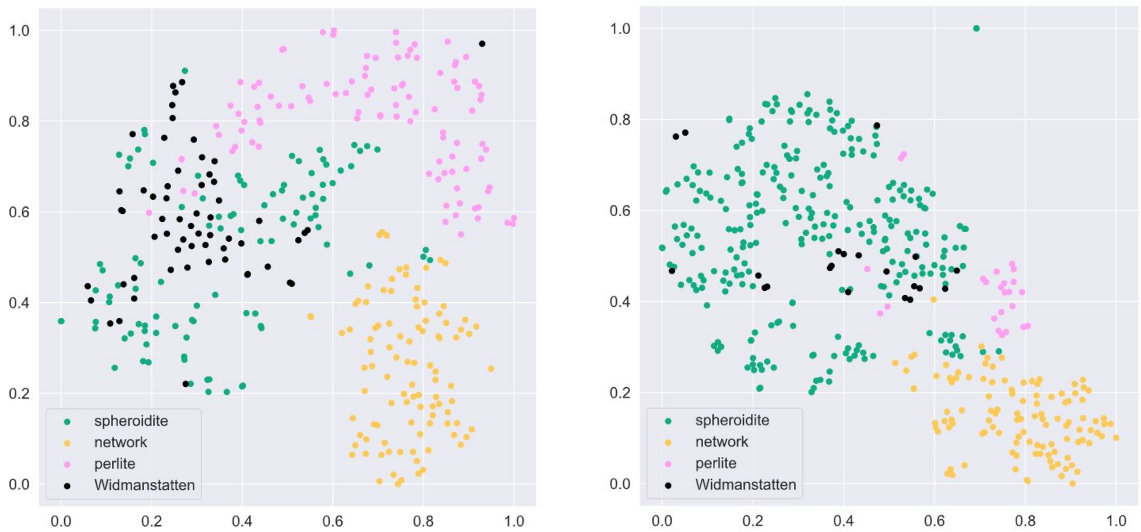


Figure 13: Comparison of classifiers with respect to their voting scores

6.5 Binary Misclassification

In this work, the multi-class classification process involves two steps-binary classifications and voting. And as the output of the voting classifier depends on the output by binary classifiers, the result by binary classifiers can significantly impact the final classification output obtained through voting.



(a) Training set *(b)* Test set

Figure 14: t-SNE visualization for Ultra High Steel Carbon micrograph dataset

Also, the output by an image classifier depends on feature extraction, size of the dataset, and type of classifier. We've already explored different classifiers. We visualize the features obtained by VGG16 to understand binary misclassification and its impact on voting. Figure 14 shows the t-Distributed Stochastic Neighbor Embedding (t-SNE) [22] visualization of micrograph dataset. t-SNE is a non-linear dimensionality reduction method used for visualizing high-dimensional data, which in this case, are the features obtained from conv 5-3 layer of VGG16.

Figure 14 depicts four clusters corresponding to each class. Cluster regions of spheroidite, pearlite and carbide network can be easily distinguished from each other. Similarly, cluster regions of widmanstätten, pearlite, and carbide networks can be visually distinguished. However, there is some overlap between them results in misclassification, as observed before. The overlap of cluster regions implies the similarity between features of their respective classes as determined by the VGG16 model. The cluster regions of spheroidite and widmanstätten coincide in both Figure 14a and Figure 14b. As a result, a huge misclassification error is observed for both microconstituents while classifying one from the other, as observed before.

CHAPTER 7

CONCLUSION

7.1 Summary

In this project, we have successfully applied SVM-based learning algorithms; namely, Twin Support Vector Machine and Least Squares Support Vector Machine on UHCS dataset [5; 10]. The classifier categorizes the input image into one of the following labels- "Pearlite," "Spheroidite," "Carbide Network," and "Widmanstätten."

We have used VGG16, a pre-trained CNN model, to extract features from the image data. We evaluated the features extracted from conv 1-2, conv 2-2, conv 3-3, conv 4-3, conv 5-3 layers of VGG16 architecture using SVM classifier. Features from conv 5-3 layer reported the highest accuracy and, thus, they were picked for feature extraction. In this work, the multi-class classification process comprises two steps- binary classification followed by voting step.

The model comprises of six binary classification problems focusing on the following six pairs of microconstituents: 1. Pearlite vs. Spherodite, 2. Pearlite vs. Widmanstätten, 3. Pearlite vs. Carbide Network, 4. Spherodite vs. Widmanstätten, 5. Carbide Network vs. Spherodite and 6. Carbide Network vs. Widmanstätten. The voting classifier combines the output by these six pairs and predicts the output label that was predicted most number of times.

	SVM	TWSVM	TWSVM (linear kernel)	LSSVM (linear kernel)	LSSVM (polynomial kernel)
Spheroidite	0.21168	0.11314	0.10219	0.20438	0.19343
Network	0.01786	0.00893	0.00893	0.01786	0.01786
Pearlite	0	0.04167	0	0.04167	0.04167
Widmanstätten	0.19048	0.23809	0.28571	0.09524	0.14286

Table 12: Final results

TWSVM with linear kernel outperformed other classifiers in the prediction of pearlite, spheroidite, and carbide network. Overall, all classifiers performed poorly in detecting widmanstätten. This is due to the small size of the dataset. We also explore data visualization approaches (t-SNE) for visualizing features and justifying the observed results. The results are summarized in Table 12

The code for feature extraction and visualization and training the classification models is given in Appendix A.

7.2 Future Work

The microstructure classification depends on the distribution of the dataset, feature extraction, and classification algorithm. In Figure 14, we observed that the cluster region of widmanstätten could not be distinguished from the remaining classes. This results in a huge misclassification error. Thus, the next step in this research could be to evaluate different image representation techniques that can describe a given microstructure uniquely. The model can be generalized by training it using microstructure images of other alloy systems with similar morphologies.

APPENDIX A

FUNCTIONS FOR MULTI-CLASS CLASSIFICATION

A. Function for Twin Support Vector Machine Algorithm

```
import numpy as np
from numpy import linalg
from cvxopt import solvers,matrix
from sklearn import preprocessing
from sklearn.base import BaseEstimator, ClassifierMixin

#Function for 1st hyperplane in TWSVM
def Twin_plane1(R,S,C1,Eps1,reg1):
  SS = np.dot(S.T,S)
  SS = SS + reg1*(np.identity(SS.shape[0]))
  SSR = linalg.solve(SS,R.T)
  RSSR = np.dot(R,SSR)
  RSSR = (RSSR+(RSSR.T))/2
  m2 = R.shape[0]
  e2 = -np.ones((m2,1))
  solvers.options['show_progress'] = False
  vlb = np.zeros((m2,1))
  vub = C1*(np.ones((m2,1)))
  cd = np.vstack((np.identity(m2),-np.identity(m2)))
  vcd = np.vstack((vub,-vlb))
  alpha = solvers.qp(matrix(RSSR,tc='d'),matrix(e2,tc='d'),matrix(cd,tc='d'
    ),matrix(vcd,tc='d'))
  alpha_sol = np.array(alpha['x'])
  z = -np.dot(SSR,alpha_sol)
  w1 = z[:len(z)-1]
  b1 = z[len(z)-1]
  return [w1,b1]

#Function for 2nd hyperplane in TWSVM
def Twin_plane_2(L,N,C2,Eps2,reg2):
  NN = np.dot(N.T,N)
  NN = NN + reg2*(np.identity(NN.shape[0]))
```

```
    NNL = linalg.solve(NN,L.T)
    LNNL = np.dot(L,NNL)
    LNNL = (LNNL+(LNNL.T))/2
    m1 = L.shape[0]
    e1 = -np.ones((m1,1))
    solvers.options['show_progress'] = False
    vlb = np.zeros((m1,1))
    vub = C2*(np.ones((m1,1)))
    cd = np.vstack((np.identity(m1),-np.identity(m1)))
    vcd = np.vstack((vub,-vlb))
    gamma = solvers.qp(matrix(LNNL,tc='d'),matrix(e1,tc='d'),matrix(cd,tc='d'
      ),matrix(vcd,tc='d'))
    gammasol = np.array(gamma['x'])
    z = -np.dot(NNL,gammasol)
    w2 = z[:len(z)-1]
    b2 = z[len(z)-1]
    return [w2,b2]

#Entire TWSVM code to find equation of hyperplanes
class TSVMClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self,Epsilon_1=0.1, Epsilon_2=0.1, C1=1, C2=1,kernel_type
      =0,kernel_parameter=1,reg1=1, reg2=1,fuzzy=0,_estimator_type="classifier
      "):
        self.Epsilon_1=Epsilon_1
        self.Epsilon_2=Epsilon_2
        self.C1=C1
        self.C2=C2
        self.reg1 = reg1
        self.reg2 = reg2
        self.fuzzy = fuzzy
        self.kernel_type=kernel_type
        self.kernel_parameter=kernel_parameter

    def fit(self, X, Y):
        assert (type(self.Epsilon_1) in [float,int])
        assert (type(self.Epsilon_2) in [float,int])
        assert (type(self.C1) in [float,int])
        assert (type(self.C2) in [float,int])
        assert (type(self.reg1) in [float,int])
        assert (type(self.reg2) in [float,int])
```

```
        assert (self.fuzzy in [0,1])
        assert (type(self.kernel_paramete) in [float,int])
        assert (self.kernel_type in [0,1,2,3])

        data = sorted(zip(Y,X), key=lambda pair: pair[0], reverse = True)
        Total = np.array([np.array(x) for y,x in data])
        A=np.array([np.array(x) for y,x in data if (y==1)])
        B=np.array([np.array(x) for y,x in data if (y==0)])

        if(self.fuzzy==1):
            if(self.kernel_type==0):
                rpos=0
                rneg=0
                xpos = np.true_divide(sum(A),len(A))
                for a in A:
                    if(rpos<np.linalg.norm(a-xpos)):
                        rpos = np.linalg.norm(a-xpos)
                xneg = np.true_divide(sum(B),len(B))
                for b in B:
                    if(rneg<np.linalg.norm(b-xneg)):
                        rneg = np.linalg.norm(b-xneg)
                self.xpos_ = xpos
                self.xneg_ = xneg
                self.rpos_ = rpos
                self.rneg_ = rneg
            else:
                rpossq=-np.inf
                termt_1=0
                for i in range(len(A)):
                    t1 = kernelfunction(self.kernel_type,A[i],A[i],self.
kernel_parameter)
                    t2 = 0
                    for j in range(len(A)):
                        t2 += kernelfunction(self.kernel_type,A[j],A[i],
self.kernel_paramter)
                        termt_1 += kernelfunction(self.kernel_type,A[i],A[j
],self.kernel_parameter)
                    t2 = -2*t2/len(A)
                    rpossq = max(rpossq,t1+t2)
                termt_1 = termt_1/(len(A)*len(A))
```

```
                rpossq += termt_1
                rnegsq=-np.inf
                termt_2=0
                for i in range(len(B)):
                    t1 = kernelfunction(self.kernel_type,B[i],B[i],self.kernel_parameter)
                    t2 = 0
                    for j in range(len(B)):
                        t2 += kernelfunction(self.kernel_type,B[j],B[i],self.kernel_parameter)
                        termt_2 += kernelfunction(self.kernel_type,B[i],B[j],self.kernel_parameter)
                    t2 = -2*t2/len(B)
                    rnegsq = max(rnegsq,t1+t2)
                termt_2 = termt_2/(len(B)*len(B))
                rnegsq += termt_2
                self.rpossq_ = rpossq
                self.rnegsq_ = rnegsq
                self.termt_1_ = termt_1
                self.termt_2_ = termt_2
        m1 = A.shape[0]
        m2 = B.shape[0]
        e1 = -np.ones((m1,1))
        e2 = -np.ones((m2,1))
        if(self.kernel_type==0):
            S = np.hstack((A,-e1))
            R = np.hstack((B,-e2))
        else:
            S = np.zeros((A.shape[0],Total.shape[0]))
            for i in range(A.shape[0]):
                for j in range(Total.shape[0]):
                    S[i][j] = kernelfunction(self.kernel_type,A[i],Total[j],self.kernel_parameter)
            S = np.hstack((S,-e1))
            R = np.zeros((B.shape[0],Total.shape[0]))
            for i in range(B.shape[0]):
                for j in range(Total.shape[0]):
                    R[i][j] = kernelfunction(self.kernel_type,B[i],Total[j],self.kernel_parameter)
            R = np.hstack((R,-e2))
```

```
        #Calculation of Parameters for Equation of planes
        [w1,b1] = Twin_plane1(R,S,self.C1,self.Epsilon_1,self.reg1)
        [w2,b2] = Twin_plane_2(S,R,self.C2,self.Epsilon_2,self.reg2)
        self.plane1_coeff_ = w1
        self.plane1_offset_ = b1
        self.plane2_coeff_ = w2
        self.plane2_offset_ = b2
        self.data_ = Total
        self.A_ = A
        self.B_ = B
        return self


    def get_parameters(self, deep=True):
        return {"Epsilon_1": self.Epsilon-1, "Epsilon_2": self.Epsilon_2, "C1": self.C1, "C2": self.C2, "reg1": self.reg1,
                "reg2":self.reg2, "kernel_type": self.kernel_type, "kernel_param": self.kernel_param,"fuzzy": self.fuzzy}

    def set_parameters(self, **parameters):
        for parameter, value in parameters.items():
            self.setattr(parameter, value)
        return self

    def predict(self, X, y=None):
        if(self.kernel_type==0):
            S = X
            w1abs = np.linalg.norm(self.plane1_coeff_)
            w2abs = np.linalg.norm(self.plane2_coeff_)
        else:
            S = np.zeros((self.data_.shape[0],self.data_.shape[0]))
            for i in range(self.data_.shape[0]):
                for j in range(self.data_.shape[0]):
                    S[i][j] = kernelfunction(self.kernel_type,self.data_[i],self.data_[j],self.kernel_parameter)
            w1abs = np.sqrt(np.dot(np.dot(self.plane1_coeff_.T,S),self.plane1_coeff_))
            w2abs = np.sqrt(np.dot(np.dot(self.plane2_coeff_.T,S),self.plane2_coeff_))
```

```
            S = np.zeros((X.shape[0],self.data_.shape[0]))
            for i in range(X.shape[0]):
                for j in range(self.data_.shape[0]):
                    S[i][j] = kernelfunction(self.kernel_type,X[i],self.data_[j],self.kernel_parameter)
        y1 = np.dot(S,self.plane1_coeff_)+ ((self.plane1_offset_)*(np.ones((X.shape[0],1))))
        y2 = np.dot(S,self.plane2_coeff_)+ ((self.plane2_offset_)*(np.ones((X.shape[0],1))))

        #Test data predictions
        yPredict=np.zeros((X.shape[0],1))

        distPlane1 = y1/w1abs
        distPlane2 = y2/w2abs

        for i in range(len(distPlane1)):
            if (distPlane1[i]<distPlane2[i]):
                yPredict[i][0]=0
            else:
                yPredict[i][0]=1

        return yPredict.transpose()[0]

    def decision_function(self,X):

        if(self.fuzzy==1):
            s1=[]
            s2=[]
            if(self.kernel_type==0):
                for i in range(len(X)):
                    s1.append(1-(np.linalg.norm(self.xcenpos_-X[i])/self.rpos_))
                    s2.append(1-(np.linalg.norm(self.xcenneg_-X[i])/self.rneg_))
            else:
                for i in range(len(X)):
                    dsqpos = kernelfunction(self.kernel_type,X[i],X[i],self.kernel_parameter)
                    t1 = 0
```

```
                    for j in range(len(self.A_)):
                        t1 += kernelfunction(self.kernel_type,self.A_[j],X[i],self.kernel_parameter)
                    t1 = -2*t1/len(self.A_)
                    dsqpos += t1
                    dsqpos += self.termt_1_
                    s1.append(1-np.sqrt(dsqpos/self.rpossq_))
                    dsqneg = kernelfunction(self.kernel_type,X[i],X[i],self.kernel_parameter)
                    t1 = 0
                    for j in range(len(self.B_)):
                        t1 += kernelfunction(self.kernel_type,self.B_[j],X[i],self.kernel_parameter)
                    t1 = -2*t1/len(self.B_)
                    dsqneg += t1
                    dsqneg += self.termt_2_
                    s2.append(1-np.sqrt(dsqneg/self.rnegsq_))
            s1 = np.array(s1)
            s2 = np.array(s2)
            return np.true_divide(s1,s1+s2)-0.5
        else:
            if(self.kernel_type==0):
                S = X
                w1abs = np.linalg.norm(self.plane1_coeff_)
                w2abs = np.linalg.norm(self.plane2_coeff_)
            else:
                S = np.zeros((self.data_.shape[0],self.data_.shape[0]))
                for i in range(self.data_.shape[0]):
                    for j in range(self.data_.shape[0]):
                        S[i][j] = kernelfunction(self.kernel_type,self.data_[i],self.data_[j],self.kernel_parameter)
                w1abs = np.sqrt(np.dot(np.dot(self.plane1_coeff_.T,S),self.plane1_coeff_))
                w2abs = np.sqrt(np.dot(np.dot(self.plane2_coeff_.T,S),self.plane2_coeff_))
                S = np.zeros((X.shape[0],self.data_.shape[0]))
                for i in range(X.shape[0]):
                    for j in range(self.data_.shape[0]):
                        S[i][j] = kernelfunction(self.kernel_type,X[i],self.data_[j],self.kernel_parameter)
```

```
            y1 = np.dot(S,self.plane1_coeff_)+ ((self.plane1_offset_)*(np.ones((X.shape[0],1))))
            y2 = np.dot(S,self.plane2_coeff_)+ ((self.plane2_offset_)*(np.ones((X.shape[0],1))))

        #Test data predictions
            yPredict=np.zeros((X.shape[0],1))

            distPlane1 = y1/w1abs
            distPlane2 = y2/w2abs

        #Test data predictions
            for i in range(len(distPlane1)):
                yPredict[i][0] = distPlane2[i]/(distPlane1[i]+distPlane2[i])-0.5
            return yPredict.transpose()[0]
```

B. Functions for feature extraction

```
def features(image_path,Models):
    features=[]
    #layername=['block1_pool','block2_pool','block3_pool','block4_pool','block5_pool']
    for i in range(5):
            model = Models[i]
            img = image.load_img(image_path, target_size=(645, 484))
            x = image.img_to_array(img)
            x = np.expand_dims(x, axis=0)
            x = preprocess_input(x)
            layer_features = model.predict(x)
            layer_features=np.mean(layer_features,axis=(1,2))
            features.append(layer_features)
    Models=[]
    return features

def finalfeatures(image_path,model):
    img = image.load_img(image_path, target_size=(645, 484))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
```

```
    features = model.predict(x)
    return features
```

C. Parameters defined for TWSVM and LSSVM

```
h = .02  # step size in the mesh

#Parameters for TWSVM with linear kernel
params1 = {'Epsilon_1': 0.1, 'Epsilon_2': 0.1, 'C1': 1, 'C2': 1,'
    kernel_type':0,'kernel_parameter': 1,'fuzzy' :0}
#Parameters for TWSVM with polynomiak kernel
params2 = {'Epsilon_1': 0.1, 'Epsilon_2': 0.1, 'C1': 1, 'C2': 1,'
    kernel_type':2,'kernel_parameter': 1,'fuzzy' :0}

#Instantiation for LSSVM with linear kernel
lssvc1 = LSSVC(gamma=1, kernel='linear')
#Instantiation for LSSVM with polynomial kernel
lssvc2 = LSSVC(gamma=1, kernel='poly', d=2)
```

D. Kernel Function for TWSVM

```
def kernelfunction(Type, u, v, p):
    # u, v are array like; p is parameter for kernels; type is:
        # type 1 linear kernel
        # type 2 polynomial kernel
        # type 3 RBF kernels
    if(Type==1):
        return np.dot(u,v)
    if(Type==2):
        return pow(np.dot(u,v)+1,p)
    if(Type==3):
        temp = u-v
        return pow(math.e,(-np.dot(temp,temp)/(p**2)))
```

E. Complete code

1. Loading data

```
filename=r"C:\Users\Madhu\OneDrive\Desktop\DDP\github codes\UHCS-TSVM\Image
    -classification-of-ultrahigh-carbon-steel-microstructures-master\Image-
    classification-of-ultrahigh-carbon-steel-microstructures-master\data\
    metadata.xlsx"
```

```
dataread=pd.read_excel(filename,'metadata',index_col=None, na_values=['NA'
    ])
head=dataread.columns.values.tolist()
prim_constituents=dataread["primary_microconstituent"].values.tolist()
microid=dataread["micrograph_id"].values.tolist()
spheroidite_id=[]
network_id=[]
pearlite_id=[]
Widmanstatten_id=[]
pearandsph_id=[]
pearandwid_id=[]
martensite_id=[]
for i in range(len(dataread)):
    if prim_constituents[i]=='spheroidite':
        spheroidite_id.append(microid[i])
    if prim_constituents[i]=='network':
        network_id.append(microid[i])
    if prim_constituents[i]=='pearlite':
        pearlite_id.append(microid[i])
    if prim_constituents[i]=='spheroidite+widmanstatten':
        Widmanstatten_id.append(microid[i])
    if prim_constituents[i]=='pearlite+spheroidite':
        pearandsph_id.append(microid[i])
    if prim_constituents[i]=='pearlite+widmanstatten':
        pearandwid_id.append(microid[i])
    if prim_constituents[i]=='martensite':
        martensite_id.append(microid[i])
spheroidite_train=spheroidite_id[0:100]
spheroidite_test=spheroidite_id[100:]
network_train=network_id[0:100]
network_test=network_id[100:]
pearlite_train=perlite_id[0:100]
pearlite_test=perlite_id[100:]
Widmanstatten_train=Widmanstatten_id[0:60]
Widmanstatten_test=Widmanstatten_id[60:]
train_set=[spheroidite_train,network_train,pearlite_train,
    Widmanstatten_train]
test_set=[spheroidite_test,network_test,pearlite_test,Widmanstatten_test]
yy=['spheroidite','network','pearlite','Widmanstatten']
```

2. Comparison of features extracted from different layers of VGG16 model

```
#Loading VGG16 model with different intermediate layer outputs
base_model = VGG16(weights='imagenet', include_top=False)
layer_name=['block1_pool','block2_pool','block3_pool','block4_pool','
    block5_pool']
Models=[]
for i in range(5):
    Models.append(Model(inputs=base_model.input, outputs=base_model.
    get_layer(layer_name[i]).output))

#Extracting features from intermediate layers
ex_features=[0,0,0,0]

for ii in range(4):
    label=train_set[ii]
    ex_features[ii]=[]
    for i in range(len(label)):
        image_path=r"C:\Users\Madhu\OneDrive\Desktop\DDP\github codes\UHCS-
    TSVM\Image-classification-of-ultrahigh-carbon-steel-microstructures-
    master\Image-classification-of-ultrahigh-carbon-steel-microstructures-
    master\data\micrographs\micrograph"+str(label[i])+"[cropped].tif"
        result=ex_features(image_path,Models)
        ex_features[ii].append(result)
        gc.collect()
        print('finish '+str(ii)+' '+str(i))
    print('finish '+str(ii))
    with open('objs.pkl', 'wb') as f:
        pickle.dump(ex_features, f)
    f.close()
with open('objs.pkl','rb') as f:
    ex_features = pickle.load(f)
f.close()

#Evaluating the performance of features extracted by different layers of
    VGG16 model and picking the most accurate one
classifier_score=[0,0,0,0,0,0]
classifier_names=[0,0,0,0,0,0]
a=0

for ii in range(4-1):
```

```
    for jj in range(ii+1,4):
        X1=ex_features[ii]
        y1=[yy[ii]]*len(X1)
        X2=ex_features[jj]
        y2=[yy[jj]]*len(X2)
        y=y1[:]
        classifier_score[a]=[]
        classifier_names[a]=yy[ii]+" "+yy[jj]
        for n in y2:
            y.append(n)
        for i in range(5):
            X=[]
            for n in range(len(X1)):
                XX1=X1[n][i][0]
                X.append(XX1.flatten())
            for n in range(len(X2)):
                XX2=X2[n][i][0]
                X.append(XX2.flatten())
            clf = svm.SVC()
            scores = cross_val_score(clf, X, y, scoring='accuracy',cv=10)
            mean_score=scores.mean()
            classifier_score[a].append(1-mean_score)
        a+=1

#write the results to an excel file.
classifierdict={}
for i in range(6):
    classifierdict[classifier_names[i]]=classifier_score[i]
VGG16_results = pd.DataFrame(classifier_dict, index=["C12","C22","C33","C43
    ","C53"])
writer = pd.ExcelWriter('VGG16_results.xlsx')
VGG16_results.to_excel(writer,'Sheet1')
writer.save()
```

3. Extracting features from conv 5-3 layer of VGG16 model

```
model = VGG16(weights='imagenet',include_top=False)
features_final_train=[0,0,0,0]
filename=r"C:\Users\Madhu\OneDrive\Desktop\DDP\github codes\UHCS-TSVM\Image
    -classification-of-ultrahigh-carbon-steel-microstructures-master\Image-
```

```
    classification-of-ultrahigh-carbon-steel-microstructures-master\data\
    micrographs\micrograph"

for i in range(4):
    label=train_set[i]
    features_final_train[i]=[]
    for j in range(len(label)):
        image_path=filename+str(label[j])+"[cropped].tif"
        features_final_train[i].append(final_features(image_path,model))
        gc.collect()
        print('Done '+str(j))
print('finished features training\n')
features_final_test=[0,0,0,0]
for i in range(4):
    label=test_set[i]
    features_final_test[i]=[]
    for j in range(len(label)):
        image_path=filename+str(label[j])+"[cropped].tif"
        features_final_test[i].append(final_features(image_path,model))
        gc.collect()
print('finished feature testing\n')
features_pands=[]
label=pearandsphname
for i in range(len(label)):
    image_path=filename+str(label[i])+"[cropped].tif"
    features_pands.append(final_features(image_path,model))
    gc.collect()

features_pandw=[]
label=pearandwidname
for i in range(len(label)):
    image_path=filename+str(label[i])+"[cropped].tif"
    features_pandw.append(final_features(image_path,model))
    gc.collect()

features_mar=[]
label=martensitename
for i in range(len(label)):
    image_path=filename+str(label[i])+"[cropped].tif"
    features_mar.append(final_features(image_path,model))
```

```
    gc.collect()
print('features extraction done\n')
with open('objsfinal.pkl', 'wb') as f:
    pickle.dump([features_final_train,features_final_test,features_pands,
    features_pandw,features_mar], f)
f.close()
```

4. Feature visualization with t-SNE

```
from sklearn.manifold import TSNE

total_dataset=features_final
all_image_features=[]
all_image_name=[]

for i in range(4):
    for j in range(len(features_final_test[i])):
        total_dataset[i].append(features_final_test[i][j])

for i in range(4):
    X1=total_dataset[i]
    Y1=[yy[i]]*len(X1)
    for n in range(len(X1)):
        XX1=np.mean(X1[n][0],axis=(0,1))
        all_image_features.append(XX1.flatten())
        all_image_name.append(Y1[n])

tsne = TSNE(n_components=2).fit_transform(all_image_features)

def scale_to_range(x):
    val_range = (np.max(x) - np.min(x))
    From_zero = x - np.min(x)
    return From_zero / val_range

# extract x and y coordinates representing the positions of the images on T
    -SNE plot

tsne_x = tsne[:, 0]
tsne_y = tsne[:, 1]
tsne_x = scale_to_range(tsne_x)
```

```
tsne_y = scale_to_range(tsne_y)

#color code for microconstituents
microconstituents={
    'spheroidite' : [16, 172, 132],
    'network' : [254, 202, 87],
    'perlite' : [255, 159, 243],
    'Widmanstatten' : [0, 0, 0],
}

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(aspect='equal')

for label in microconstituents:

    indices = [i for i, l in enumerate(all_image_name) if l == label]
    current_tsne_x = np.take(tsne_x, indices)
    current_tsne_y = np.take(tsne_y, indices)
    color = np.array(microconstituents[label], dtype=np.float) / 255
    ax.scatter(current_tsne_x, current_tsne_y, c=color, label=label)

ax.legend(loc='best')
plt.show()
```

5. Binary classification using TWSVM and LSSVM

```
with open('objsfinal.pkl','rb') as f:  # Python 3: open(..., 'rb')
    features_final,features_final_test,features_pands,features_pandw,
   features_mar = pickle.load(f)
f.close()

classifier_names=[0,0,0,0,0,0]
bi_classifers=[]
c=0

for ii in range(4-1):
    for jj in range(ii+1,4):
        X1=features_final[ii]
        y1=[yy[ii]]*len(X1)
        X2=features_final[jj]
```

```
        y2=[yy[jj]]*len(X2)
        y=y1[:]
        classifier_names[nn]=yy[ii]+" "+yy[jj]
        for n in y2:
            y.append(n)
        X=[]
        for n in range(len(X1)):
            XX1=np.mean(X1[n][0],axis=(0,1))
            X.append(XX1.flatten())
        for n in range(len(X2)):
            XX2=np.mean(X2[n][0],axis=(0,1))
            X.append(XX2.flatten())
        X=np.array(X)
        y=np.array(y)

        #For TWSVM
        bi_classifers.append(OneVsOneClassifier(TWSVMClassifier(**params3)).fit(X,y))
        #if LSSVM
        #biclassifier.append(lssvc1.fit(X, y))
        c+=1
```

6. Voting Classifier

```
def voting(bi_classifers,X):
    y=[]
    for j in range(len(bi_classifers)):
       y.append(bi_classifers[j].predict(X))
    y=np.array(y)
    axis=0
    unique, indices = np.unique(y, return_inverse=True)
    return unique[np.argmax(np.apply_along_axis(np.bincount, axis, indices.reshape(y.shape),
                                None, np.max(indices) + 1), axis=axis)]
```

7. Evaluating multi-class classifier using test set

```
scores=[0,0,0,0]
voting_scores=[]
for i in range(4):
    X1=features_final_test[i]
```

```
    y1=[yy[i]]*len(X1)
    y=y1[:]
    X=[]
    scores[i]=[]
    for n in range(len(X1)):
        XX1=np.mean(X1[n][0],axis=(0,1))
        X.append(XX1.flatten())
    X=np.array(X)
    for j in range(len(bi_classifers)):
        scores[i].append(1-bi_classifers[j].score(X,y))
    y_voting=voting(biclassifers,X)
    voting_score=0
    for j in range(len(y)):
        if y[j]==y_voting[j]:
            voting_score+=1
    voting_score=voting_score/len(y)
    voting_scores.append(1-voting_score)
```

DRIVE LINK OF THE CODE

https://drive.google.com/drive/folders/1EJEA29bFp8Dg0iKJczpOiGDVR9a1-Z8y?usp=sharing

References

[1] **Baskaran, A.**, **G. Kane**, **K. Biggs**, **R. Hull**, and **D. Lewis** (2020). Adaptive characterization of microstructure dataset using a two stage machine learning approach. *Computational Materials Science*, **177**, 109593.

[2] **Cecen, A.**, **H. Dai**, **Y. C. Yabansu**, **S. R. Kalidindi**, and **L. Song** (2018). Material structure-property linkages using three-dimensional convolutional neural networks. *Acta Materialia*, **146**, 76–84.

[3] **Chowdhury, A.**, **E. Kautz**, **B. Yener**, and **D. Lewis** (2016). Image driven machine learning methods for microstructure recognition. *Computational Materials Science*, **123**, 176–187.

[4] **DeCost, B. L.**, **T. Francis**, and **E. A. Holm** (2017). Exploring the microstructure manifold: image texture representations applied to ultrahigh carbon steel microstructures. *Acta Materialia*, **133**, 30–40.

[5] **DeCost, B. L.**, **M. D. Hecht**, **T. Francis**, **B. A. Webler**, **Y. N. Picard**, and **E. A. Holm** (2017). Uhcsdb: Ultrahigh carbon steel micrograph database. *Integrating Materials and Manufacturing Innovation*, **6**(2), 197–205.

[6] **DeCost, B. L.** and **E. A. Holm** (2015). A computer vision approach for automated analysis and classification of microstructural image data. *Computational materials science*, **110**, 126–133.

[7] **Ding, S.**, **J. Yu**, **B. Qi**, and **H. Huang** (2014). An overview on twin support vector machines. *Artificial Intelligence Review*, **42**(2), 245–252.

[8] **Donahue, J.**, **Y. Jia**, **O. Vinyals**, **J. Hoffman**, **N. Zhang**, **E. Tzeng**, and **T. Darrell**, Decaf: A deep convolutional activation feature for generic visual recognition. *In International conference on machine learning*. 2014.

[9] **Frankel, A. L.**, **R. E. Jones**, **C. Alleman**, and **J. A. Templeton** (2019). Predicting the mechanical response of oligocrystals with deep learning. *Computational Materials Science*, **169**, 109099.

[10] **Hecht, M. D.**, **B. L. DeCost**, **T. Francis**, **E. A. Holm**, **Y. N. Picard**, and **B. A. Webler** (). Ultrahigh carbon steel micrographs. `https://hdl.handle.net/11256/940`.

[11] **Hecht, M. D.**, **Y. N. Picard**, and **B. A. Webler** (2017). Coarsening of inter-and intra-granular proeutectoid cementite in an initially pearlitic 2c-4cr ultrahigh carbon steel. *Metallurgical and Materials Transactions A*, **48**(5), 2320–2335.

[12] **Hecht, M. D.**, **B. A. Webler**, and **Y. N. Picard** (2016). Digital image analysis to quantify carbide networks in ultrahigh carbon steels. *Materials Characterization*, **117**, 134–143.

[13] **Kalinin, S. V.**, **B. G. Sumpter**, and **R. K. Archibald** (2015). Big–deep–smart data in imaging for guiding materials design. *Nature materials*, **14**(10), 973–980.

[14] **Khemchandani, R.**, **S. Chandra**, *et al.* (2007). Twin support vector machines for pattern classification. *IEEE Transactions on pattern analysis and machine intelligence*, **29**(5), 905–910.

[15] **Kim, B.**, **N. Yuvaraj**, **K. S. Preethaa**, and **R. A. Pandian** (2021). Surface crack detection using deep learning with shallow cnn architecture for enhanced computation. *Neural Computing and Applications*, 1–17.

[16] **Lesuer, D.**, **C. Syn**, **A. Goldberg**, **J. Wadsworth**, and **O. Sherby** (1993). The case for ultrahigh-carbon steels as structural materials. *JOM*, **45**(8), 40–46.

[17] **Mangal, A.** (2018). *Applied Machine Learning to Predict Stress Hotspots in Materials*. Ph.D. thesis, Carnegie Mellon University.

[18] **Pierson, K.**, **A. Rahman**, and **A. D. Spear** (2019). Predicting microstructure-sensitive fatigue-crack path in 3d using a machine learning framework. *JOM*, **71**(8), 2680–2694.

[19] **Rajan, K.** (2012). Materials informatics. *Materials Today*, **15**, 470.

[20] **Russakovsky, O.**, **J. Deng**, **H. Su**, **J. Krause**, **S. Satheesh**, **S. Ma**, **Z. Huang**, **A. Karpathy**, **A. Khosla**, **M. Bernstein**, *et al.* (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, **115**(3), 211–252.

[21] **Suykens, J. A.** and **J. Vandewalle** (1999). Least squares support vector machine classifiers. *Neural processing letters*, **9**(3), 293–300.

[22] **Van der Maaten, L.** and **G. Hinton** (2008). Visualizing data using t-sne. *Journal of machine learning research*, **9**(11).

[23] **Wiki** (2019). Convolutional neural network.

[24] **Yang, Z.**, **Y. C. Yabansu**, **R. Al-Bahrani**, **W.-k. Liao**, **A. N. Choudhary**, **S. R. Kalidindi**, and **A. Agrawal** (2018). Deep learning approaches for mining structure-property linkages in high contrast composites from simulation datasets. *Computational Materials Science*, **151**, 278–287.

[25] **Yu, X.** (2017). Machine learning application in the life time of materials. *arXiv preprint arXiv:1707.04826*.