Milestone 4

Overview / Objective

Microstructures are 2D images captured in microscopy, which are defined as the structure of a prepared surface of material. Nonetheless, they efficiently result in a structure-property correlation study. However, manual recognition of microscopic data is both human knowledge-intensive and potentially ambiguous, especially in the case of steel. The current study creates a computer vision of important microconstituents of steels such as ferrite, pearlite, austenite, bainite, and martensite, enabling automatic identification of microstructure. Utilizing the Chan-Vese image segmentation methods, the current work extracted the patches of phases from the collected microstructures of steels. These phase patches constituted the database for the development of a deep convolutional neural network VGG-19, ResNet-50 and EfficientnetB3. The article demonstrates the development and prediction framework of in-silico microstructure vision. Utilizing a few typical example microstructure data, the article discussed the automatic labelling knowledge intensiveness and ambiguous behavior of the created microstructure vision.

Dataset Details

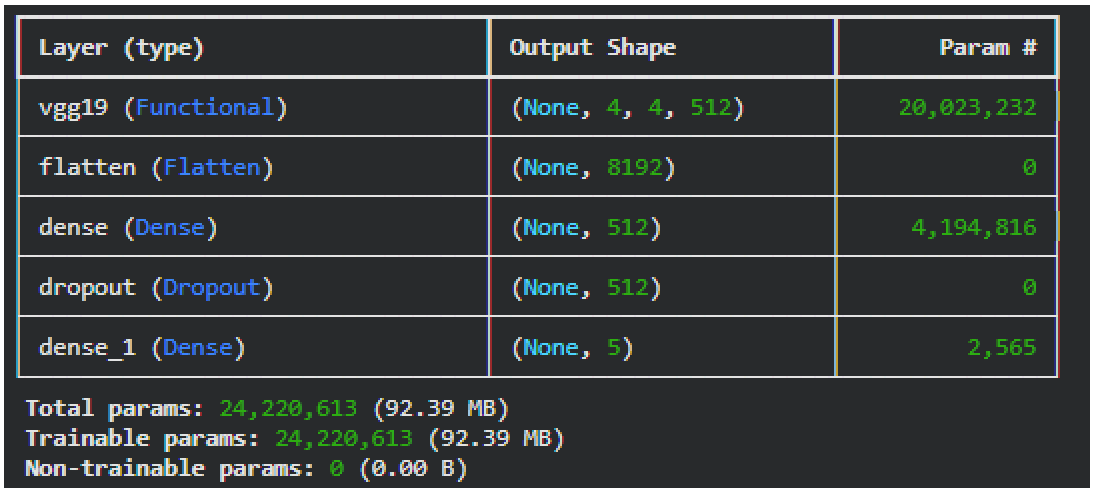
A steel microstructure database was created by an extensive review of published literature. Relevant data were meticulously gathered from various scientific articles, research papers, and industrial reports that detailed the microstructural characteristics and phase compositions of steels belonging to the advanced high-strength steels. The ground reality for labeling micrographs was done as they are reported in the literature. The literature sources were carefully selected based on their relevance, rigor, and the quality of their experimental methodologies.

Model Architecture

The three commonly used CNN-based architectures, like VGG-19, Resnet-50, and EfficientnetB3, are selected to train augmented patches for a comparative analysis of the models. However, some modifications have been made to these original networks to adapt to our work. One additional fully connected layer was added to extract features with high resolution, and one dropout layer was added in the network to avoid overfitting. The last layer of the classification model was SoftMax, and the binary cross-entropy was selected as the loss function in this study. Adam was used as an optimizer for the loss function.

1. VGG – 19

VGG-19 is a convolutional neural network (CNN) based architecture designed for image classification, and it was proposed by the Visual Geometry Group (VGG) from the University of Oxford. VGG-19 is primarily known for its depth and simplicity, making it a popular choice for various image classification and recognition tasks. It consists of a total of 19 layers with learnable weights in which 16 layers are convolutional layers, and 3 layers are fully connected layers followed by 5 max-pooling layers and the last layer is a SoftMax layer as shown in Fig. 1. The main characteristic of VGG-19 is the use of very small convolutional filters of size 3×3 with a stride of 1 and padding of 1.



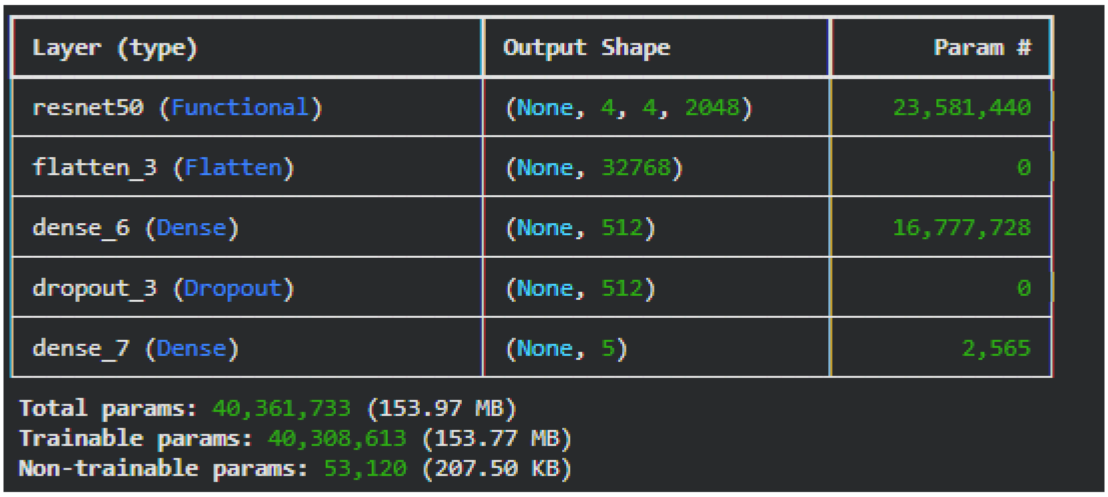
Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 4, 4, 512)	20,023,232
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 512)	4,194,816
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 5)	2,565

```
Total params: 24,220,613 (92.39 MB)
Trainable params: 24,220,613 (92.39 MB)
Non-trainable params: 0 (0.00 B)
```

Fig. 1. The schematic of the VGG-19 model is used for the classification of patches.

2. Resnet50

ResNet-50 is a variant of residual neural network (ResNet) architecture which consists of 50 deep convolutional neural networks to overcome the vanishing gradient problem. In the ResNet model, residual elements with skip connection are primarily incorporated into deep architecture as illustrated in figure 2 (a). Its core building block is the bottleneck residual block, consisting of three convolutional layers: a 1×1 convolution for dimensionality reduction, a 3×3 convolution for feature extraction, and another 1×1 convolution for dimensionality restoration as shown in figure 2 (b). Each block incorporates a skip connection that adds the input directly to the output, enabling stable gradient flow during backpropagation and allowing networks to scale much deeper than before. The architecture begins with convolution and pooling layers, followed by multiple stacked residual blocks, and concludes with global average pooling and a fully connected classification layer. The network learns and chooses how the gradient propagates on its own, therefore even if network depth increases, the error will not increase. The ResNet method primarily learns the residual relationship between input x and output . It finally takes F(x)+x as the output target. The ResNet-50 architecture is splitted into five substrates. Stage 0 in ResNet-50 structure is rather straightforward and may be generally treated as input preprocessing stage. The bottleneck structures that make up the final four phases are comparatively identical. Three bottleneck structures are present in stage 1, and four, six and three bottlenecks are present in the subsequent three phases, respectively.



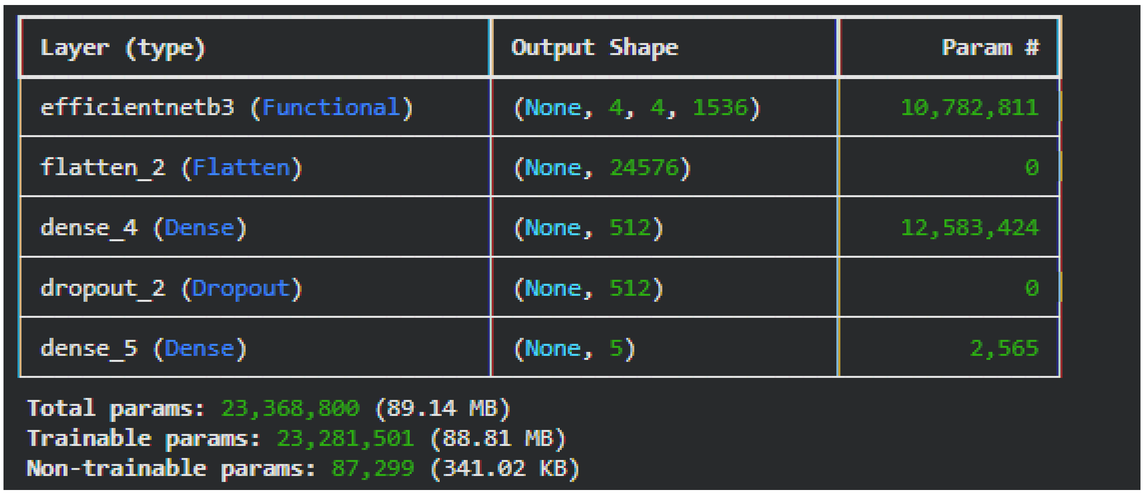
Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 4, 4, 2048)	23,581,440
flatten_3 (Flatten)	(None, 32768)	0
dense_6 (Dense)	(None, 512)	16,777,728
dropout_3 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 5)	2,565

```
Total params: 40,361,733 (153.97 MB)
Trainable params: 40,308,613 (153.77 MB)
Non-trainable params: 53,120 (207.50 KB)
```

Fig. 2. (a) Architecture of ResNet50 containing an initial convolutional layer.

3. EfficientNetB3

The Efficientnetb3 is a variant of the Efficientnet series, is a convolutional neural network that achieves high accuracy while being computationally efficient through a compound scaling method (figure 3). Unlike traditional scaling approaches that only increase network depth or width, EfficientNet uniformly scales depth, width, and input resolution using a fixed compound coefficient. Its architecture is built upon MBConv (Mobile Inverted Bottleneck Convolution) blocks (figure 3(b and c)), which integrate depthwise separable convolutions, expansion and projection layers, squeeze-and-excitation (SE) modules, and inverted residual connections. EfficientNet-B3 represents a scaled-up version of the baseline EfficientNet-B0, increasing both resolution and network size to balance accuracy and efficiency. Compared to other architectures like VGG and ResNet variants, EfficientNet-B3 offers significantly higher accuracy per parameter and FLOP, making it well-suited for real-world applications, particularly in medical imaging (e.g., brain tumor and diabetic retinopathy classification) and mobile deployment where computational resources are limited. Its design demonstrates how carefully balanced scaling can yield superior performance without unnecessary model complexity.



```
┃ Layer (type)                 ┃ Output Shape          ┃       Param # ┃
┃ efficientnetb3 (Functional)  ┃ (None, 4, 4, 1536)    ┃    10,782,811 ┃
┃ flatten_2 (Flatten)          ┃ (None, 24576)         ┃             0 ┃
┃ dense_4 (Dense)              ┃ (None, 512)           ┃    12,583,424 ┃
┃ dropout_2 (Dropout)          ┃ (None, 512)           ┃             0 ┃
┃ dense_5 (Dense)              ┃ (None, 5)             ┃         2,565 ┃
 Total params: 23,368,800 (89.14 MB)
 Trainable params: 23,281,501 (88.81 MB)
 Non-trainable params: 87,299 (341.02 KB)
```

Fig. 3. The schematic representation of EfficientNetB3 architecture consisting of MBConv1 and MBconv6 blocks used to extract feature maps

Training Setup

The last layer of the classification model was SoftMax, and the binary cross-entropy was selected as the loss function in this study. Adam was used as an optimizer for the loss function with a learning rate of 0.01. Accuracy was selected as the evaluation metrics and batch size was 32. The training of the classification model was performed on Intel(R) Xeon(R) Gold 5218R CPU, and Nvidia RTX A5000 GPU in Python-based TensorFlow 2.0 platform.

Initial Training Results

Figure XX represents the accuracy and loss curves on the testing dataset of the phase classification models (i.e., VGG-19, Resnet-50, and EfficientnetB3) respectively. All these classification models are trained from scratch by setting up ‘weights = None’. The accuracy obtained using VGG-19, Resnet-50, and EfficientnetB3 models was 19%, 86% and 19% respectively. Considering the accuracy and time for training the model, Resnet-50 was selected as the classification model in our classification case of microconstituent patches. It was observed that theResnet-50 model’s accuracy does not improve much beyond 5 epochs. The

Resnet-50 model was trained using aforementioned settings which exhibited excellent training and testing performance, indicating that the model did not overfit the training and testing sets.

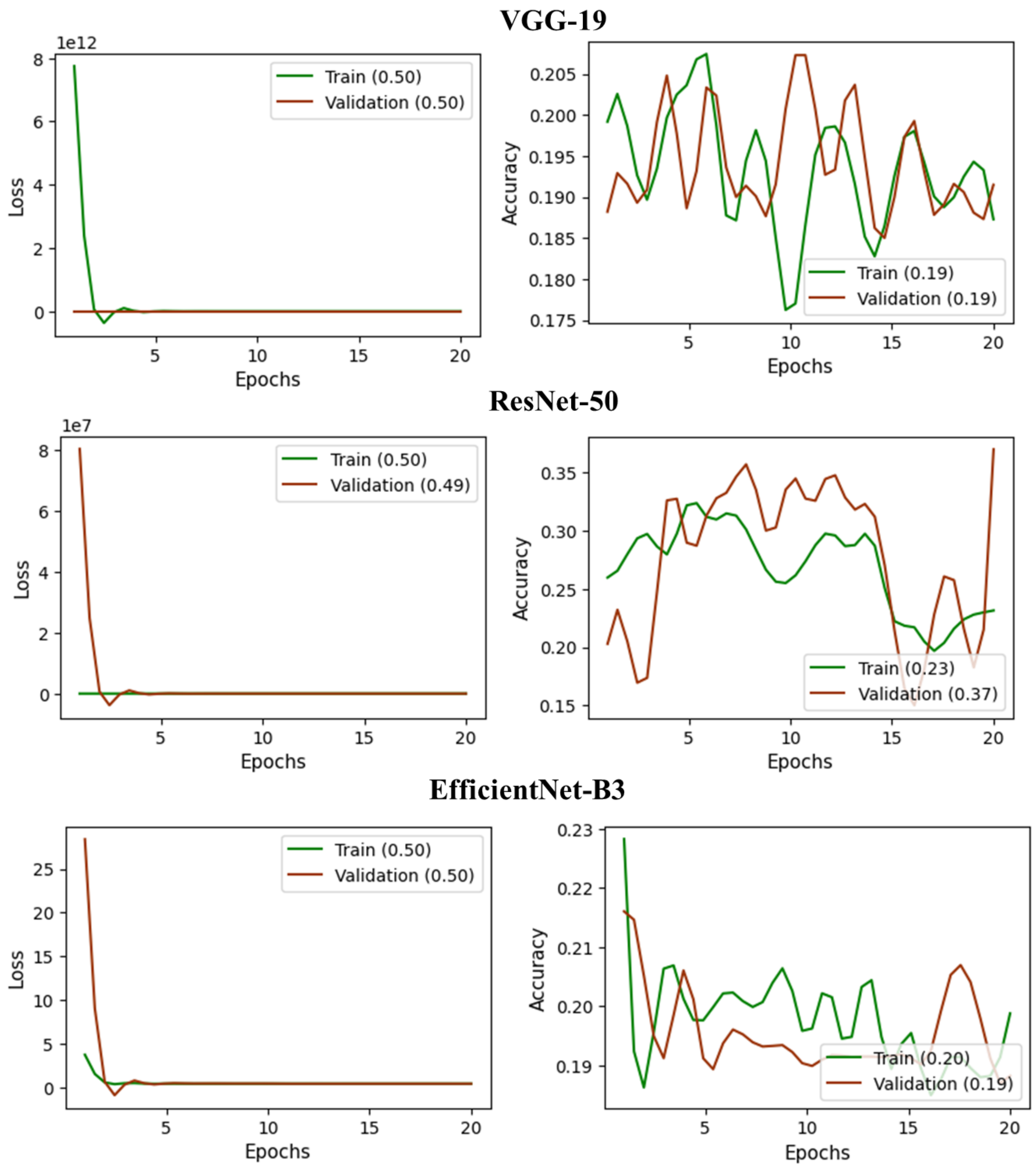


Figure: Training and validation curves (loss and metrics over epochs).

Hyperparameter Experiments

The hyperparameter tuning of Resnet50 explored three key parameters: the number of units in the dense layer, the dropout rate, and the learning rate.

- **Units:** The number of neurons in the dense layer was varied between 256 and 1024 in steps of 256. The tuning process identified 256 units as the optimal configuration, indicating that a smaller, more compact network performed best for this task.
- **Dropout:** The dropout rate was tested across a linear range from 0.0 to 0.7 with increments of 0.1. The optimal dropout rate was found to be 0.1, suggesting that a light amount of regularization helped improve generalization without underfitting.
- **Learning Rate:** Three learning rate options were evaluated — 0.01, 0.001, and 0.0001. The optimal value was 0.001, which provided a good balance between convergence speed and training stability.

Overall, the best-performing model configuration uses 256 units, a 0.1 dropout rate, and a learning rate of 0.001, resulting in an efficient and well-regularized neural network as shown in table XX.

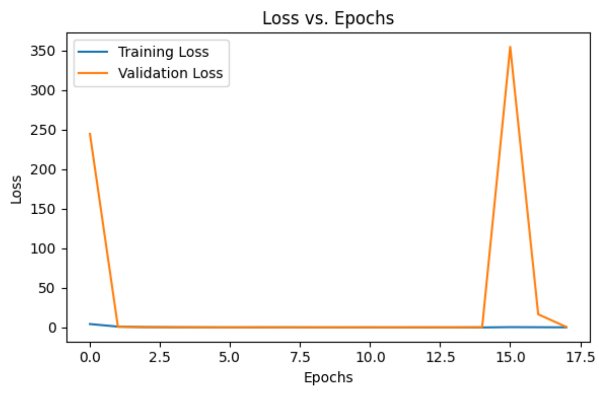
S. No.	learning_rate	units	dropout	score	S. No.	learning_rate	units	dropout	score
1	0.001	256	0.1	0.82	**37**	0.01	1024	0.4	0.28
2	0.01	256	0.1	0.81	**38**	0.01	256	0.5	0.27
3	0.001	512	0	0.78	**39**	0.001	1024	0.5	0.27
4	0.001	512	0.6	0.74	**40**	0.01	1024	0.6	0.26
5	0.001	768	0.4	0.70	**41**	0.01	512	0.5	0.24
6	0.001	512	0.3	0.67	**42**	0.001	768	0.5	0.23
7	0.001	1024	0.1	0.66	**43**	0.01	768	0.6	0.22
8	0.001	256	0.2	0.65	**44**	0.01	768	0.5	0.22
9	0.001	768	0.2	0.64	**45**	0.01	256	0.3	0.21
10	0.01	1024	0.5	0.64	**46**	0.01	768	0.2	0.21
11	0.001	512	0.1	0.62	**47**	0.01	256	0	0.20
12	0.001	256	0.6	0.60	**48**	0.01	256	0.2	0.19
13	0.001	1024	0.4	0.55	**49**	0.0001	1024	0.5	0.19
14	0.01	768	0.3	0.54	**50**	0.0001	256	0.6	0.19
15	0.001	768	0.6	0.54	**51**	0.0001	768	0.1	0.19
16	0.001	256	0.3	0.53	**52**	0.0001	1024	0.6	0.19
17	0.01	512	0.4	0.53	**53**	0.0001	768	0.3	0.19
18	0.01	512	0	0.52	**54**	0.0001	256	0.5	0.19
19	0.001	512	0.4	0.47	**55**	0.0001	1024	0.2	0.19
20	0.001	512	0.2	0.46	**56**	0.0001	768	0.5	0.19
21	0.001	768	0.3	0.44	**57**	0.0001	512	0.5	0.19
22	0.001	512	0.5	0.44	**58**	0.0001	1024	0.3	0.19
23	0.01	512	0.1	0.40	**59**	0.0001	512	0.6	0.19
24	0.01	512	0.6	0.39	**60**	0.0001	768	0.6	0.19

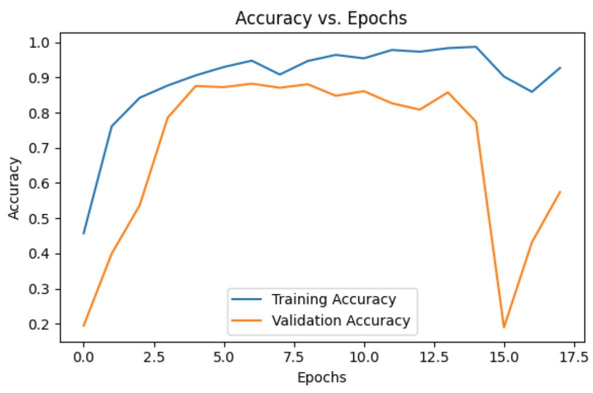
25	0.01	256	0.6	0.39	**61**	0.0001	256	0.3	0.19
26	0.001	1024	0.3	0.39	**62**	0.0001	768	0.2	0.19
27	0.001	768	0.1	0.38	**63**	0.0001	768	0	0.19
28	0.001	256	0.5	0.37	**64**	0.0001	512	0.3	0.19
29	0.01	768	0.4	0.37	**65**	0.0001	512	0.1	0.19
30	0.001	768	0	0.34	**66**	0.0001	1024	0.1	0.19
31	0.01	1024	0	0.34	**67**	0.0001	256	0.4	0.19
32	0.001	256	0	0.33	**68**	0.01	512	0.2	0.19
33	0.01	768	0	0.31	**69**	0.0001	768	0.4	0.19
34	0.01	1024	0.2	0.31	**70**	0.0001	512	0	0.19
35	0.01	768	0.1	0.30	**71**	0.0001	512	0.4	0.19
36	0.01	256	0.4	0.29	**72**	0.0001	1024	0.4	0.19

Regularization and Optimization Techniques

A dense neural network was optimized using hyperparameter tuning, evaluating the number of dense units, dropout rate, and learning rate. The final selected configuration consisted of 256 units in the dense layer, a dropout rate of 0.1, and a learning rate of 0.001. To enhance regularization and mitigate overfitting, L2 kernel regularization was applied on the dense layer. Additionally, early stopping with a patience of 10 epochs was employed to prevent unnecessary training and ensure optimal convergence.

The model achieved a training accuracy of 95% and a validation accuracy of 85%, indicating strong learning performance with some degree of generalization gap. Overall, the chosen hyperparameters and regularization strategies contributed to an efficient and well-balanced model performance. The accuracy and loss plot has been shown in figure XX.





K-fold cross validation

To further evaluate the model's robustness and reduce bias from a single train-validation split, k-fold cross-validation was performed. The final model configuration consisting of 256 dense units, 0.1 dropout, L2 kernel regularization, and with a learning rate of 0.001 was used during the cross-validation process. Across the 5 fold folds, the model demonstrated consistent performance, achieving a best validation accuracy of 95%. This result confirms the model's stability and ability to generalize well across different subsets of the data, supporting the reliability of the chosen. The accuracy across each fold is shown in figure XX. The best performing model is stored into '.h5' for future predictions.

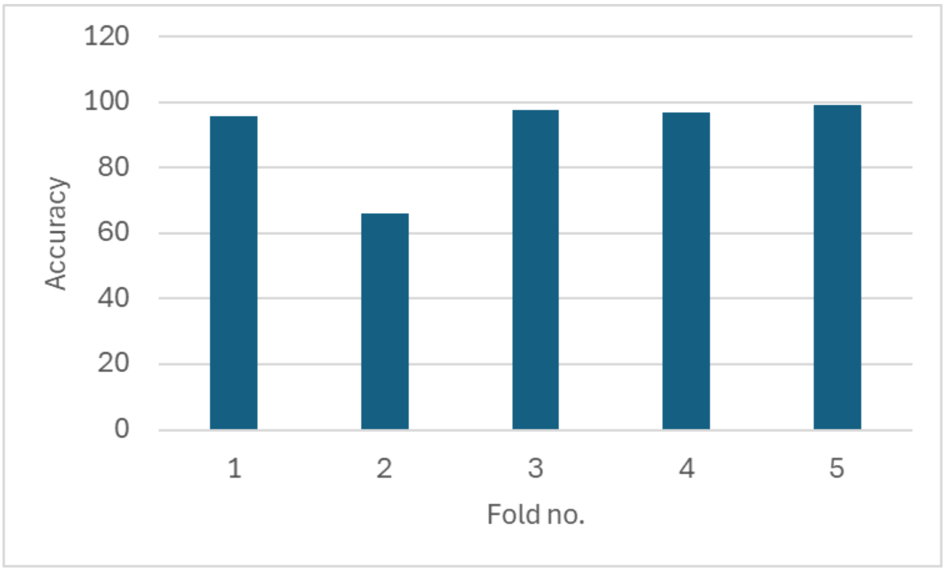


Fig. validation accuracy vs. fold