Machine Learning-Based Detection of Parameter Deviations in FDM Plastic Extrusion Processes

Jiří Tomášek, Jan Bergmann, and Leon Glitt

Abstract. Fused Deposit Modeling (FDM) is a widely used additive manufacturing method prone to failures caused by various machine parameters, making it important to identify the actual cause of errors. Therefore, this project uses a large-scale dataset to develop a machine learning (ML) model for error detection in FDM.
Initial experimentation with a K-Nearest-Neighbor (KNN) algorithm provides a benchmark for the image classification problem. The research then focuses on advanced convolutional neural networks (CNNs) to improve accuracy. Multi-headed neural networks with different backbones, such as ResNet and custom CNNs, are developed to classify four key printing parameters (Hotend Temperature, Z-Offset, Flow Rate, and Lateral Speed), which are assigned to three classes (Low, Good, High).
The KNN approach faces challenges in managing high-dimensional data and struggles to achieve sufficient accuracy. In contrast, CNNs shows promising results with the highest accuracies achieved with a ResNet34. Overfitting represents the biggest challenge of the CNN, but can be tackled by tuning hyperparameters and increasing the number of samples.

Keywords: Aditive Manufacturing · Machine Learning · KNN · Multi-head Models

1 Introduction

1.1 Motivation

Today, FDM is widely used for many applications. However, a significant disadvantage is the need for human intervention when unexpected errors arise. If left undetected, these errors render entire parts unusable, leading to wastage of time and materials. Errors can stem from a variety of process parameters, often influencing each other, making error tracking and parameter adjustment time-consuming. Hence, our aim is to implement a solution by detecting potential errors using image data from the extrusion process. Initially, we'll use a simpler machine learning approach with KNN algorithm, later transitioning to a CNN to leverage its advanced image processing capabilities. The Caxton dataset will be used to train and test the different ML models during this project.

1.2 Research

A non-parametric approach to image classification is introduced by Ramteke et al. in 2012 [3]. Their work presents a K-Nearest Neighbor classifier for medical

brain scans. It uses an image preprocessing step, which includes transforming the images to grayscale and applying a median filter. Furthermore the Haralick texture feature extraction is used to extract: contrast, entropy, homogeneity, correlation and local homogeneity. These features are then used by the KNN model to classify the images into normal or abnormal. Finally, the model achieves an accuracy of 80 % on their dataset.

Wang et al. [2] present an approach to detect surface defects on FDM prints using a custom, lightweight CNN. This was achieved through hyperparameter analysis and the implementation of residual blocks. The main findings were that the number of layers and kernels have a higher impact on the accuracy than the kernel size.

The Paper *Generalisable 3D printing error detection and correction via multi-head neural networks* [1] was our initial inspiration, describing the creation of the Caxton dataset (Chapter 2.1) and an error detection method for FDM. The article focuses on four adjustable parameters - *Hotend Temperature*, *Z-Offset*, *Flow Rate*, and *Lateral Speed* - which significantly influence part quality. The paper describes how these parameters can be estimated using Image data. This is achieved through a CNN, which classifies each parameter as *low*, *good*, or *high*.

The authors achieved an average accuracy of 84.3% over all parameters. Individually, the accuracies were 87.1% for Flow Rate, 86.4% for Lateral Speed, 85.5% for Z-Offset, and 78.3% for Hotend Temperature.

2 Data handling and visualization

2.1 The Caxton dataset

The collaborative autonomous extrusion network (CAXTON) dataset is a large-scale optical in situ process monitoring dataset specifically curated for FDM. It consists of over 1 million sample images capturing material deposition from printer nozzles. Each image in the dataset is labeled with its respective printing parameters expressed as either low, good or high (Fig. 2 appendix B), used during dataset generation.

2.2 Our preprocessing and data visualization

In the preprocessing pipeline, several techniques are employed to extract relevant features of input images. Subsequently, the `crop_image_around_nozzle` function focuses analysis on the region where plastic is extruded, effectively removing extraneous background information. The `resize_image` function resizes images to standard dimensions for easier processing. Both of these functions are employed in both the KNN and Neural Network approaches, which will be elaborated upon later. Conversely, the subsequent functions are exclusively used for the KNN approach, as the Convolutional Neural Network conducts this preprocessing inherently. Conversion to the LAB color space via `rgb_to_lab` provides a more robust representation of the image. Contrast Limited Adaptive Histogram Equalization

(CLAHE) is implemented to adaptively enhance contrast while limiting noise amplification. Conversion to grayscale using `rgb_to_grayscale` simplifies processing tasks by retaining only intensity information. `Unsharp_masking` sharpens images by emphasizing edges and details. Finally, `change_brightness` dynamically adjusts image brightness based on mean brightness values, compensating for lighting variations.

An example of processing a single sample image can be seen in figure 4 appendix B.

3 KNN implementation

We opt to use the non-parametric K-Nearest-Neighbor model as our benchmark for the image classification problem. With the CAXTON dataset, we need to convert the multi-class multilabel classification into a multiclass classification task. This involves one-hot encoding the data, resulting in four final labels represented by a 1D vector of length 3. Each position in the vectors corresponds to a class of each parameter, where the presence of a class is indicated by a value of 1 and the absence by a value of 0. We're predicting each parameter with its own KNN (3 classes: *low*, *good*, *high*).

The feature vector used as input for KNN consists of the pixel values of the flattened and preprocessed image.

3.1 Optimization

There are multiple ways to optimize the performance of the KNN. Firstly, one can apply *normalization* on the data, after splitting the dataset into train, validation, and test sets to avoid data leakage. However the mean and standard deviation values are exceptionally low, resulting in ineffective normalization. For that reason, normalization is not applied on our data. In contrast, we apply *Principal Component Analysis* (PCA) to reduce the dimensionality of our high-dimensional image data. We implement a *grid search algorithm* to find the optimal values for the k-value, the distance metric, and the weight of the distance metric. The possible grid parameters are shown in table 3 in appendix A.

The best parameters are selected based on the accuracy or F1-score on the validation set. Besides that, the size of the resized image also influences the performance and runtime of the KNN. Therefore multiple combinations are used.

3.2 Training

In total, numerous combinations of optimization techniques can be applied. Thus, we conduct three training iterations. Configuration of trainings can be seen in table 4 in appendix A.

In all trainings, the grid search algorithm is used to select the best parameters based on model accuracy or F1-score. Resized images are 16x16 pixels in size

for each training. For each training iteration, 100k samples are randomly drawn from the entire dataset and split with a 70/20/10 split ratio.

3.3 Evaluation

Each model is evaluated on the test set based on accuracy per predicted parameter or F1-score per predicted parameter. Additionally, confusion matrices per predicted parameter are compared qualitatively, and images of correctly and incorrectly classified samples are also compared qualitatively.

4 CNN implementation

Convolutional neural networks are widely used for image classification tasks due to their ability to automatically extract key features from images, which are then processed by the network.

In the following sections, multi-head models with different backbones are introduced.

The input for each model consists of RGB images that have been cropped around the nozzle to a size of 300×300 pixels and subsequently resized to 224×224 pixels.

4.1 Multi-Head model

Predicting all four parameters at once could be done using 4 individual CNNs. However, this approach would not consider possible dependencies between the parameters. This is why we have chosen to build a multi-head model.

The model uses one convolutional backbone and 4 heads. The convolutional backbone extracts important features and feed them into heads. Each head is associated with one parameter and is composed of fully connected layers. The simplified structure of the model can be seen in Fig. 1.

The choice of architecture of the convolutional backbone is crucial. Two primary groups of backbones were implemented. The first group utilizes simple convolution with two (CNN2) and four (CNN4) convolutional layers. The structures of these backbones are described in Tables 5 and 6 in appendix A.

The second group employs ResNet architecture, where only convolutional layers are used. Three different ResNets were used - ResNet18, ResNet34, ResNet50. The ResNet architecture is described in section 4.2 and structures of each ResNet are shown in Fig. 5 in appendix B.

All backbones use $ReLU$ activation functions.

The heads consists of two fully connected layers with $ReLU$ activation function and $Softmax$ in the output nodes. Each backbone outputs a flatten vector of different size, so the number of parameters in heads with different backbones is different. The structures of the heads for CNN2, CNN4, ResNet18 and ResNet34 backbones are shown in Table 7 and for ResNet50 in Table 8 in appendix A.

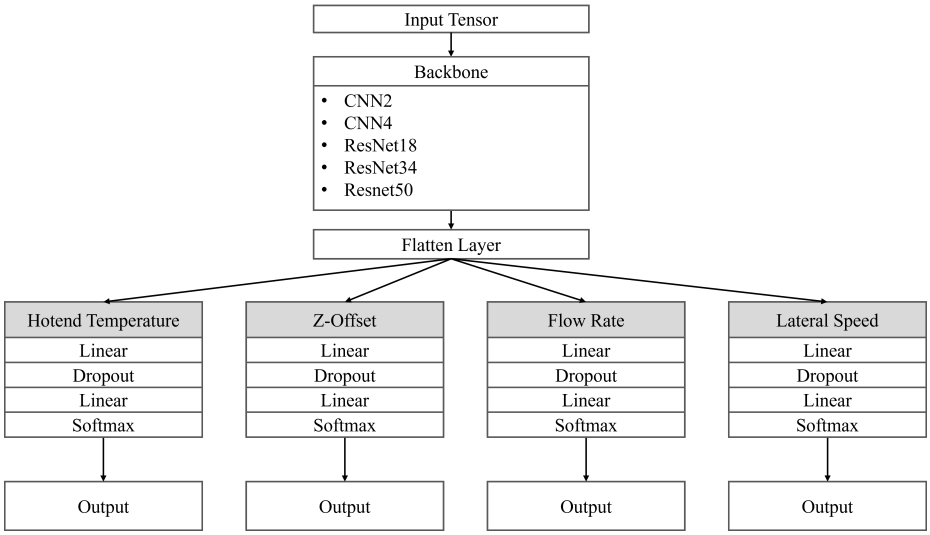


Fig. 1. Network structures used in this work

Each head is updated based on its own loss, while the backbone is updated with the sum of all head losses. This is achieved by summing the losses from each head and backpropagating them through the network. As the gradients between heads are zero, each head is updated only with its own loss.

4.2 ResNet

ResNet (Residual Network) is a deep learning architecture that utilizes skip connections or residual blocks to enable the training of very deep neural networks. These skip connections allow for bypassing of one or more layers to effectively reduce the distance information must travel through the network. This design helps mitigate the vanishing gradient problem, which can hinder training in deep networks by making it difficult for gradients to propagate back to earlier layers.

By using these residual blocks, ResNet improves training stability and efficiency, enabling networks to achieve higher levels of accuracy. Additionally, the architecture allows for deeper networks without sacrificing performance or increasing training time excessively. ResNet's flexibility and robustness have made it a popular choice for various computer vision tasks, including image recognition and classification.

ResNet comes in different depths to suit various application needs. ResNet18 is a shallower version with 18 layers, providing a good trade-off between model complexity and computational efficiency. ResNet-34, with 34 layers, offers improved performance while maintaining a reasonable model size. ResNet50, with 50 layers, utilizes bottleneck blocks to maintain efficiency and high accuracy.

4.3 Optimization

Optimizing the multi-head neural network involves tuning numerous hyperparameters. Some hyperparameters remain fixed while others are optimized during different training iterations, each using distinct sets of hyperparameters. The fixed hyperparameters can be found in Table 9 in appendix A.

The remaining hyperparameters are adjusted through multiple training sessions, where model accuracies, F1-scores, and learning curves are assesed. Refer to Table 10 in appendix A for these parameters. It can be seen that we choose *Dropout* values between 0 and 0.6 and apply either *no regularization*, l_1-*regularization*, or l_2-*regularization*.

In Fig. 3 in appendix B, histograms illustrate distribution of loaded samples for each class for the train set (blue), validation set (orange) and test set (green). It is seen that classes *Z-Offset* and *Hotend Temperature* are not balanced. Weighted loss function is used to counteract the imbalance of the dataset. Hereby the loss for each head takes class weights into account. The class weights are calculated by multiplying the inverse of the number of samples per class with an empirical factor. Furthermore it can be seen, that the distribution is similar for the train, validation and test set. For that reason we don't use a Stratified Sampler.

4.4 Training

The training of the multi-head model with the five different backbones follows a systematic approach. Initially, CNN2 is trained as the backbone to establish a baseline. Various configurations of the hyperparameters listed in Table 10 are tested, while those in Table 9 remain constant. The same configurations are then applied to train the CNN4 backbone. Following this, the more advanced ResNets are used as backbones, beginning with ResNet18. We adjust the hyperparameters to maximize accuracy and apply the same process for ResNet34 and ResNet50. After comparing the results of all models, the best model is selected. Finally, this model is trained on 1M samples for 20 epochs to attain the highest possible accuracy.

In general, a subset of samples is drawn randomly from the entire dataset and split with a 70/20/10 ratio.

An early stopping mechanism is implemented to prevent overfitting and reduce training time. The training process is terminated if the validation loss does not decrease for a specified number of epochs, helping to avoid wasting resources on overfitting the model.

The model with the best average accuracy on the validation set is saved as the best model. This allows for testing and deployment of the most effective model configuration.

Additionally an option to continue training on a saved model is implemented. Using this option, the model and optimizer state are loaded from a previously saved checkpoint, enabling training to resume from a specific point. This makes it easier to handle interruptions and experiment with different configurations.

4.5 Evaluation

The model's performance is assessed on the test set using the accuracy metric and F1-score. The average accuracy and F1-score are calculated across all heads of the model, providing a benchmark to evaluate different backbones and configurations. Additionally, confusion matrices per predicted parameter and learning curves of the training processes are compared qualitatively.

5 Discussion

In this section the results of the conducted experiments are discussed.

5.1 KNN model

Peprocessing	**PCA**	**Accuracy [%] (val/test)**
×	×	**52.03 / 52.42**
✓	×	50.62 / 51.03
✓	✓	51.62 / 51.73

Table 1. Accuracies for training configurations with KNN

Table 1 displays achieved average accuracies with KNN model. Highest test accuracy 52.52% is achieved without preprocessing and PCA.

In general, the proposed KNN struggles to predict parameters accurately, whether collectively or individually. The grid search consistently selects KNN with $k = 1$, indicating potential overfitting, despite similar accuracy or F1-score on the validation and test set. This suggests the problem's dimensionality may exceed KNN's capability. Reducing dimensionality with a smaller image size or PCA doesn't improve accuracy. Preprocessing also has no positive impact on model accuracy, though qualitative analysis suggests correct functionality in classifying images.

Additionaly, due to imbalance in dataset, another evaluation of Training 1 is done based on F1-score. As shown in Table 11 in appendix A, there is a significant variation in the F1-scores among the *Z-Offset* and *Hotend Temperature* parameters. For classes *Hotend Temperature* and *Lateral Speed* the F1-score is almost the same as accuracy.

5.2 Multi-Head model

Table 2 displays the best achieved test accuracies with different backbones and training configurations. As a baseline, the simple CNN2 and CNN4 achieve an accuracy of 59.19% and 55.80%. However it has to be mentioned that the number

of samples is considerably lower with 400k and 200k than 1M for ResNet34 and ResNet50.

Model	Epochs	Samples	Weighted Loss	Dropout	Accuracy [%]
CNN2	20	400k	×	0.4	59.19
CNN4	20	200k	×	0.0	55.80
ResNet18	20	400k	✓	0.6	62.70
ResNet34	20	1M	×	0.5	**76.18**
ResNet50	20	1M	×	0.5	75.04

Table 2. Best test accuracies for each backbone

ResNet34 backbone performs best on the test set with accuracy 76.18%. ResNet34 achieves this accuracy after 15 epochs, which is shown by the learning curve in Fig. 7. It can also be observed, that the model only suffers from slight overffiting. The resulting confusion matrix is displayed in Fig. 8. It is seen, that the predictions for *Flow Rate* and *Lateral Speed* are fairly balanced, while those for *Z-Offset* and *Hotend Temperature* are not. This outcome was expected due to the imbalanced dataset.

The different trainings for each backbone, which results can be seen in Table 12 to Table 16, show that a high number of samples significantly improves the accuracies and reduces overfitting. When training CNNs as opposed to the KNN, we can utilize a larger quantity of samples since training can be executed on GPUs and in batches. Furthermore the influence of different Dropout values and our own implemented weighted loss function can be observed. In general it is noticeable that tuning the Dropout values reduces overfitting and increases the test accuracies. Also using the weighted loss function can prevent overfitting and balance the predictions better. We believe that by tuning the scaling weighted loss factor hyperparameter in future works, the model performance can be further increased.

The biggest problem while training our models seems to be the overfitting. This is illustrated exemplarily by Fig. 6 in appendix B. In comparison, tuning the hyperparameters and increasing the number of samples results in the learning curve in Fig. 7 in appendix B with significantly less overfitting.

Our best model's accuracy of 76.18% is lower than the average accuracy of 84.3%, that was reached by the authors in [1]. However, we use a simpler architecture without attention modules or ensemble learning and introduce our own weighted loss, which leaves room for improvement.

Over all the training results demonstrate a clear trade-off between computational cost and accuracy. Models with lower complexity can still produce reasonable outcomes, while increasing the number of samples and the complexity of the models leads to improved accuracy but also higher computational costs.

5.3 Outlook

In future works, the influence of tuning the hyperparameter of our weighted loss function can be further investigated. Also implementing Attention Modules to analyze the influence on the model's performance could be of interest. Finally using augmented data to further increase the number of training samples and potentially improve the generalization performance can be discovered.

References

1. Brion, D.A.J., Pattinson, S.W. Generalisable "3D printing error detection and correction via multi-head neural networks", *Nature Communications* 13, 4654 (2022).

2. Y. Wang et al., "A CNN-Based Adaptive Surface Monitoring System for Fused Deposition Modeling", *IEEE/ASME Transactions on Mechatronics*, vol. 25, no. 5, pp. 2287-2296, Oct. 2020

3. Z. Fan, J. Xie, Z. Wang, P.-C. Liu, S. Qu, and L. Huo, "Image Classification Method Based on Improved KNN Algorithm", *Journal of Physics: Conference Series*, vol. 1930, no. 1. IOP Publishing, p. 012009, May 01, 2021.

4. K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition." Available: https://arxiv.org/pdf/1512.03385

A Tables

Parameter	Options
k-value	1-10
Distance metric	Euclidean, Manhattan, Chebyshev
Weight	Uniform, Distance

Table 3. Grid Parameters

	Preprocessing	PCA
Training 1	×	×
Training 2	✓	×
Training 3	✓	✓

Table 4. Training Configurations

Layer	Activation Volume Size	Number of parameters
INPUT	$224 \times 224 \times 3$	0
CONV5-16 S=1 P=2	$224 \times 224 \times 16$	1 216
POOL-2 S=2 P=0	$112 \times 112 \times 16$	0
CONV3-32 S=1 P=1	$112 \times 112 \times 32$	4 640
POOL-2 S=2 P=0	$56 \times 56 \times 32$	0
FLATTEN	1×100352	0

Table 5. CNN2 backbone structure. CONVx-N denotes a convolutional layer with N kernels with height and width equal to x. S and P parameters corresponds to stride and padding value. POOL-n denotes a $n \times n$ max-pooling layer.

Layer	Activation Volume Size	Number of parameters
INPUT	$224 \times 224 \times 3$	0
CONV5-16 S=1 P=2	$224 \times 224 \times 16$	1 216
POOL-2 S=2 P=0	$112 \times 112 \times 16$	0
CONV3-32 S=1 P=1	$112 \times 112 \times 32$	4 640
POOL-2 S=2 P=0	$56 \times 56 \times 32$	0
CONV3-64 S=1 P=1	$56 \times 56 \times 64$	18 496
POOL-2 S=2 P=0	$28 \times 28 \times 64$	0
CONV3-128 S=1 P=1	$28 \times 28 \times 128$	73 856
POOL-2 S=2 P=0	$14 \times 14 \times 128$	0
FLATTEN	1×25088	0

Table 6. CNN4 backbone structure.

Layer	Activation Volume Size	Number of parameters		
		CNN2	CNN4	ResNet18 & ResNet34
FC-256	1×256	25 690 368	6 422 784	131 328
DROPOUT	1×256	0	0	0
FC-3	1×3	771	771	771

Table 7. Head structure for CNN2, CNN4, ResNet18, and ResNet34 backbones. FC-x denotes a fully connected layer with x nodes.

Layer	Activation Volume Size	Number of parameters
FC-1024	1×1024	2 098 176
DROPOUT	1×1024	0
FC-3	1×3	3075

Table 8. Head structure for ResNet50 backbone

Hyperparameters	Value
Batch size	32
Learning rate	1e-3
Loss function	CrossEntropy Loss
Optimizer	ADAM Optimizer
Shuffle	True
Train split	0.7
Validation split	0.2
Test split	0.1
Workers	4

Table 9. Fixed hyperparameters for all training iterations

Hyperparameters	Value
Dropout	0.0 - 0.6
Epochs	20 or 40
Regularization	L1, L2 or None
Samples	200k - 400k
Weighted Loss Function	0 - 0.5

Table 10. Adjustable hyperparameters during training iterations

	Flow rate [%]	Lateral speed [%]	Z-offset [%]	Hotend temperature [%]
Low	55.61	54.01	66.56	34.86
Good	42.76	40.81	39.71	60.20
High	56.10	52.14	31.71	43.74

Table 11. F1-scores for each label in each class for training 1.

Epochs	Samples	Weighted Loss	Dropout	Train Accuracy [%]	Test Accuracy [%]
20	200k	×	0.0	69.49	56.89
20	200k	×	0.4	**89.80**	57.16
20	200k	×	0.6	71.03	55.66
20	400k	×	0.4	65.74	**59.19**
20	400k	✓	0.4	66.61	56.02

Table 12. Accuracies for CNN2

Epochs	Samples	Weighted Loss	Dropout	Train Accuracy [%]	Test Accuracy [%]
20	200k	×	0.0	**64.97**	**55.80**
20	200k	×	0.4	62.71	55.41
20	200k	×	0.6	47.03	47.11
20	400k	×	0.4	46.98	46.89
20	400k	✓	0.2	44.07	46.78

Table 13. Accuracies for CNN4

Epochs	Samples	Weighted Loss	Dropout	Train Accuracy [%]	Test Accuracy [%]
20	200k	×	0.0	67.23	60.02
20	200k	×	0.4	75.18	61.41
20	200k	×	0.6	**84.19**	61.68
20	200k	✓	0.6	60.41	52.40
20	400k	✓	0.6	74.05	**62.70**

Table 14. Accuracies for ResNet18

Epochs	Samples	Weighted Loss	Dropout	Train Accuracy [%]	Test Accuracy [%]
20	200k	×	0.5	74.20	61.00
20	200k	✓	0.5	68.52	55.40
20	400k	×	0.5	**81.10**	67.16
20	400k	✓	0.5	73.43	61.25
20	1000k	×	0.5	X	**x**

Table 15. Accuracies for ResNet34

Epochs	Samples	Weighted Loss	Dropout	Train Accuracy [%]	Test Accuracy [%]
20	200k	×	0.0	58.61	56.45
20	400k	×	0.4	74.64	65.94
20	1M	×	0.6	**84.19**	**x**
20	1M	✓	0.6	74.58	72.37

Table 16. Accuracies for ResNet50

B Figures

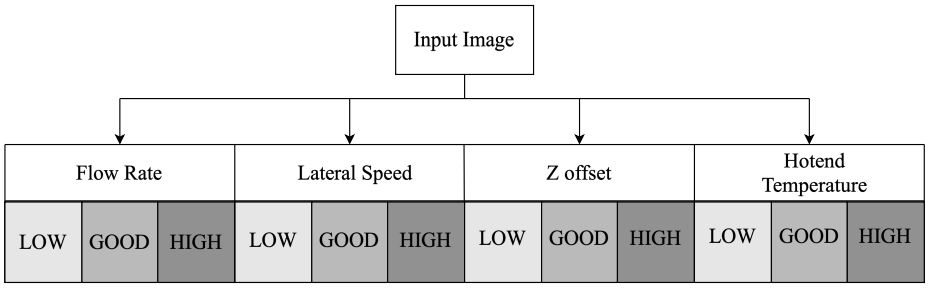


Fig. 2. Labels of each image

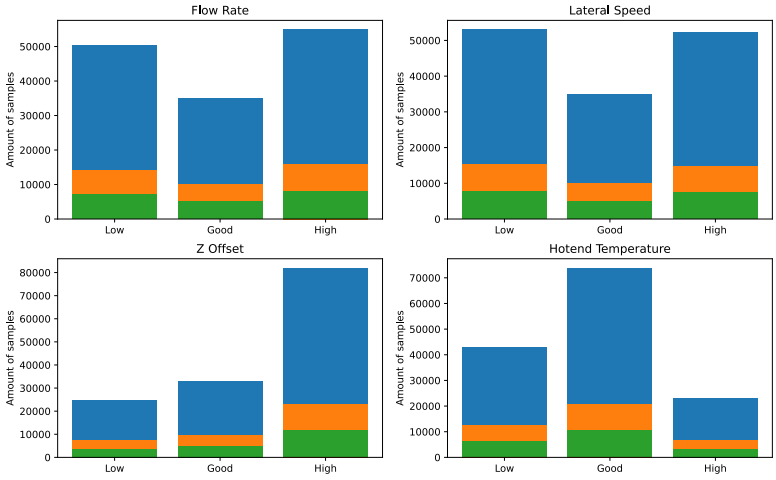
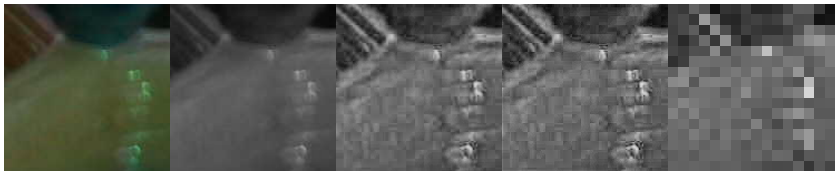


Fig. 3. Distribution of samples per class separately for each parameter, for train set (blue), validation set (orange) and test set (green)



Cropped LAB (only L) CLAHE Unsharp mask Resized 16x16

Fig. 4. Sample image preprocessing example.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig. 5. Structures of ResNet backbones [4]

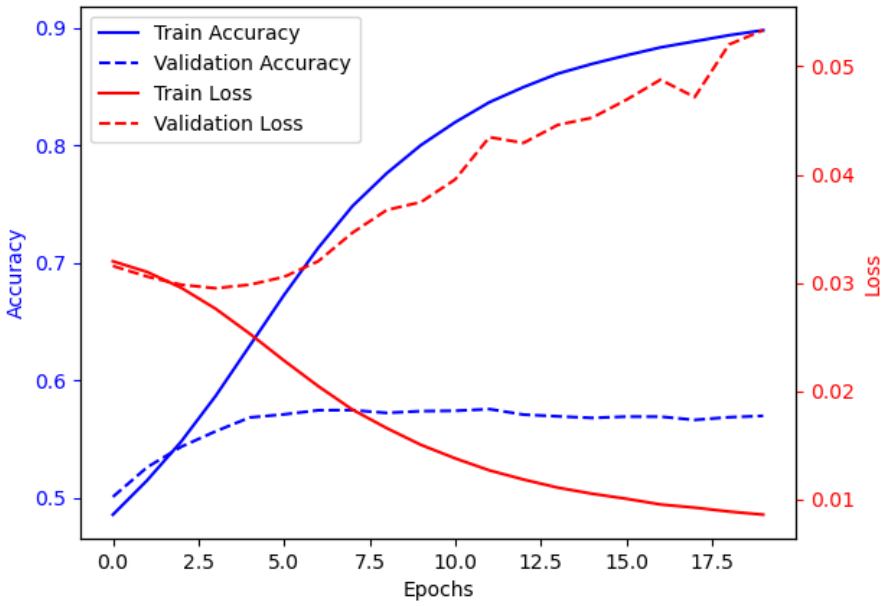


Fig. 6. Learning Curve for CNN2 with strong overfitting

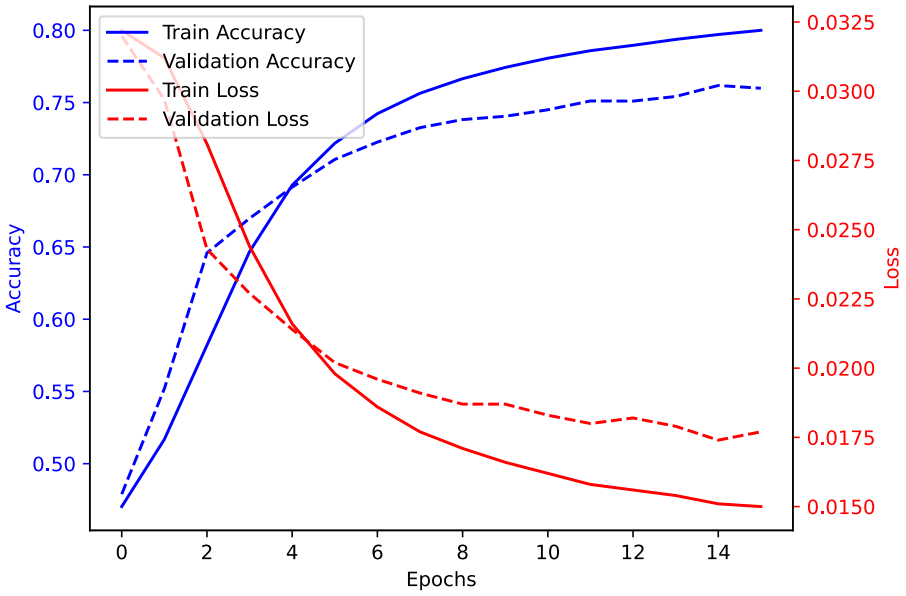


Fig. 7. Learning curve for ResNet34 with highest accuracy

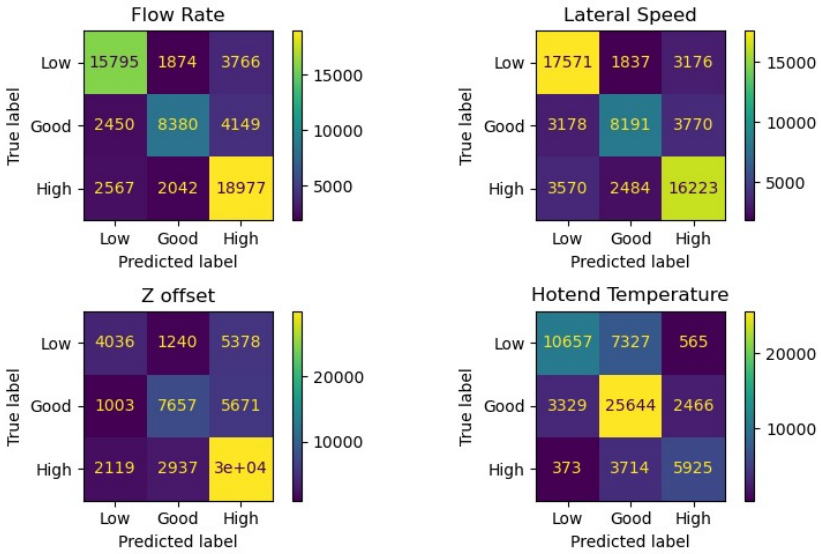


Fig. 8. Confusion matrix of ResNet34 with highest accuracy