EOLVision: Designing a CNN for Optimized End-of-Line Testing in Industrial Applications

Milan Tóth
Department of Computer Science
DHBW Stuttgart
inf22131@lehre.dhbw-stuttgart.de

Marijan Buffi
Department of Computer Science
DHBW Stuttgart
inf22013@lehre.dhbw-stuttgart.de

Repository: https://github.com/milannal1m/EOLVision

Abstract

End-of-line testing is critical for ensuring product quality in industrial automation, yet current systems often rely on inefficient manual or rule-based methods. This study presents EOLVision, a convolutional neural network (CNN) designed for automated quality assessment in modular robotic environments. Using 1,184 labeled images, the model classifies assembled products as "blue", "red", "white" or "fail," achieving a test accuracy of 92.5% and strong performance across all classes. The results demonstrate the model's effectiveness, with potential improvements through larger datasets, better imaging, and enhanced regularization.

1 Introduction

In modern industrial automation, ensuring product quality and detecting defects at the end of a production line is a critical task. End-of-line testing plays a vital role in verifying that manufactured items meet specified quality standards before they reach consumers. Despite its importance, many end-of-line systems rely on manual inspection or traditional rule-based algorithms, which can be slow, error-prone, and inflexible.

Our motivation for addressing this problem comes from the need for a more efficient and reliable solution that integrates seamlessly with modular robotic systems. To explore this, we developed an end-of-line testing CNN model, called EOLVision, which will be used on a Fischertechnik device. This device mimics a real-world production environment, providing an accessible platform to test and validate our approach.

The input to EOLVision is an image of the assembled product, represented by coloured pellets with markings, captured by a camera mounted at the end of the production line. The system uses a convolutional neural network (CNN) to analyse the image and predict the class of the product: either "blue", "red" or "white" (meets quality standards) or "fail" (requires rework or rejection).

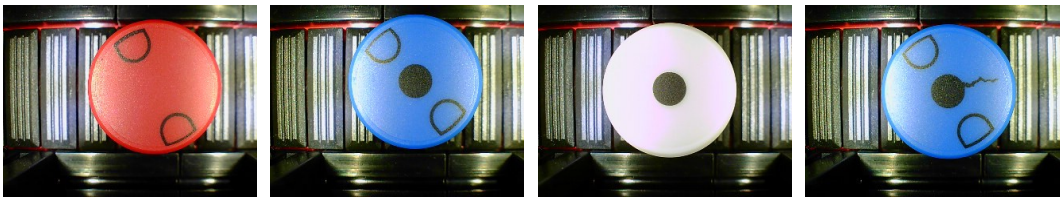
2 Dataset

In this work, we use a dataset provided by our professor, Matthias Drüppel, which is specifically tailored for the Fischertechnik model. The dataset contains 1184 labeled images across 4 classes. Each image is sized 240×320 pixels. It was divided using the Sciki-learn framework (6) as follows:

Training set: 830 images (75% of the data).
Validation set: 177 images (15% of the data).
Test set: 177 images (15% of the data).

2.1 Classes

The four classes are red, blue, white and fail. Red, blue and white refer to pellets of that color that resemble a correctly assembled product, while fail refers to pellets of any color that resemble a faulty product. An example of an image of every class can be seen in Figure 1 below. The respective class is written under the image.



Red Blue White Fail

Figure 1: Example of an image of every class

2.2 Preprocessing

Data preprocessing is a crucial step in the development of a machine learning model, especially for image classification tasks. In this project, several preprocessing techniques were applied to ensure the quality and consistency of the input data. The following steps outline the preprocessing measures implemented:

Image Filtering

During the preprocessing phase, images that were unclear or ambiguous, making it difficult to determine their class, were manually removed from the dataset. This step ensured that only high quality and clearly labeled images were used for training and evaluation, thereby improving the reliability of the model. In the dataset provided, some images were unclear due to overexposure from the camera light. Two examples of such images are shown in Figure 2.

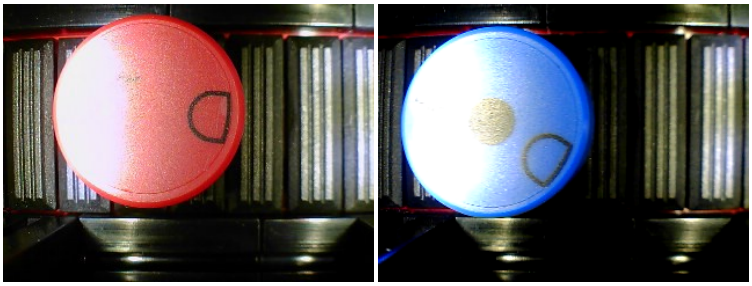


Figure 2: Example of two unclear, images

The left image is labeled red and the right image is labeled blue. However, even to the human eye, it is not possible to tell whether these images should be classified as failures or not.

Image Resizing

To ensure uniformity in the input data, all images were resized to a fixed dimension of 320x240 pixels. The resized images were then converted into NumPy arrays, using the NumPy framework (4), for further processing.

Data Normalization

Normalization is a common preprocessing step that scales the pixel values of the images to a specific range. In this project, the pixel values were normalized to the range [0, 1] by dividing each pixel value by 255.0. This step helps in stabilizing the training process and improving the convergence of the model.

Label Encoding

The class labels were encoded into numerical values, using (6). This encoding converts the class labels into a one-hot encoded format, which is suitable for training the neural network.

3 Methods

The methods used to develop the model are discussed below. The exact values and structure of these methods are given in chapter 4.1. Some methods not discussed in this chapter, such as batch normalization, were tried during the development of the model, but were not used in the final product because they didn't bring any improvement.

3.1 Loss Function

In this project, the Categorical Crossentropy loss function was used to train the Convolutional Neural Network (CNN) model. This function is commonly used for multi-class classification problems. The Categorical Crossentropy Loss function is defined as follows:

$$\text{Loss} = -\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i)$$

- C: the number of classes.
- y_i: the true label for the ith class.
- $\hat{y}_i$: is the probability predicted by the model for the ith class.

This Loss function measures the dissimilarity between the true labels and the predicted probabilities. It is defined as the negative log likelihood of the true labels given the predicted probabilities. The goal of the training process is to minimize this loss, thereby improving the accuracy of the model's predictions.

3.2 Optimizer

In this project, the Adam optimizer of the Keras framework (2) was used to train the CNN model. The Adam optimizer is an adaptive learning rate optimization algorithm that has been widely adopted in the field of deep learning due to its efficiency and effectiveness.

The Adam Optimizer is characterized by several important features. It adjusts the learning rates for each parameter individually, which helps to cope with sparse gradients and improves convergence. In addition, Adam integrates momentum by using moving averages of the gradients, which speeds up convergence and smoothes the optimization process. Finally, Adam includes bias correction terms to compensate for the initial bias in the moment estimates, ensuring more accurate updates.

3.3 Model Architecture

The Convolutional Neural Network (CNN) model used in this project was designed to classify images into four categories: blue, fail, red, and white. The model was built using Keras (2) and TensorFlow (1). The architecture of the model consists of several convolutional layers followed by max-pooling layers, dropout layers for regularization, and dense layers for classification.

3.4 Regularization

In this project, dropout was used as a regularization technique to reduce overfitting. Dropout prevents the co-adaptation of neurons by randomly setting a fraction of neuron activations to zero during training, based on a specified dropout rate. This forces the network to learn more robust features that are not dependent on specific neurons. During inference, dropout is disabled and the full network is used, with weights scaled to account for the units dropped during training. This technique increases the model's ability to generalize and is particularly effective in deep neural networks.

$$Loss_{\text{reg}} = Loss + \lambda \sum_{i=1}^{n} w_i^2$$

- λ: The regularization parameter controlling the strength of the penalty.
- w_i: The weights of the model.
- n: The total number of weights in the model.

Furthermore, L2 regularization, also known as weight decay, was employed as a complementary regularization strategy. L2 regularization works by adding a penalty term, to the loss function. The new formula is shown above. This discourages the network from assigning large weights to any particular connection, thereby promoting simpler models that are less likely to overfit the training data. The combination of dropout and L2 regularization enhances the model's robustness and improves its generalization performance.

3.5 Trainings Techniques

Early stopping and learning rate reduction have been implemented as techniques to optimize training. Early stopping stops training if the validation accuracy hasn't increased in a given number of epochs. After stopping, it restores the weights that achieved the best validation accuracy. The implemented learning rate reduction technique is called learning rate reduction on plateau. It reduces the learning rate when the validation accuracy hasn't increased after a given number of epochs.

4 Results

This section highlights the hyperparameters chosen for the model, as well as its performance and the results obtained.

4.1 Hyperparameters

The selection of appropriate hyperparameters plays a central role in the development of a powerful model. The most important decisions regarding the hyperparameters are explained below:

For this model, a learning rate of 0.0004 was chosen. To make this decision, a grid search was performed first. Then the exact learning rate was determined experimentally. The learning rate was reduced on a plateau as explained in section 3.5. The number of epochs that constitute a plateau was set to 3 and the reduction factor to 0.7, as this gave the best results.

Since this model uses early stopping, the number of epochs was set to a high number and the patience for early stopping was set to 15 epochs, as this gave excellent results while keeping the training time short. The batch size was set to 32, as this reduced overfitting while still providing excellent performance.

The Model Architecture begins with an input layer consisting of a 2D convolutional layer with 32 filters, a kernel size of 3x3, and ReLU activation, accepting the given input shape. This is followed by a max-pooling layer with a pool size of 2x2. The model then includes another 2D convolutional layer with 32 filters and ReLU activation, followed by a dropout layer with a dropout rate of 0.40 and another max-pooling layer. Subsequently, a 2D convolutional layer with 64 filters and ReLU activation is added, followed by another dropout layer with a 0.40 dropout rate and a max-pooling layer. The model then flattens the data to prepare it for dense layers. The dense layers include a layer

with 64 units, ReLU activation, and L2-regularization with a factor of 0.001, followed by a dropout layer with a 0.40 dropout rate. Finally, the model includes a dense layer with units equal to the number of classes and softmax activation for classification. This architecture combines convolutional layers for feature extraction, pooling layers for dimensionality reduction, dropout layers to prevent overfitting, and dense layers for classification. This architecture and parameters were determined experimentally.

4.2 Metrics

The performance of the developed Convolutional Neural Network (CNN) was evaluated using multiple metrics to ensure a comprehensive understanding of its predictive capabilities. The metrics include accuracy, precision, recall, and F1-score, calculated for each class in the dataset. Additionally, the model's training, validation and test accuracies were analyzed to assess its generalization performance. While during development both the confusion matrix and the class-based performance metrics were computed using the validation set, the confusion matrix and metrics shown in the figures below were computed using the test set and together with the test accuracy, represent the performance of the model on previously unseen data.

Accuarcy

In evaluating the performance of the CNN model, the achieved accuracy metrics demonstrate its strong effectiveness in classification tasks. The model reached an impressive training accuracy of 99.9%, indicating that it successfully learned to classify the training data with minimal errors. The validation accuracy of 95.4% suggests that the model generalizes well to unseen data during training, but with some overfitting. Finally, a test accuracy of 92.5% reflects the model's robust performance when deployed on entirely new data, confirming its ability to maintain high accuracy even in real-world scenarios. These results highlight the model's strong predictive capabilities and its potential for practical application.

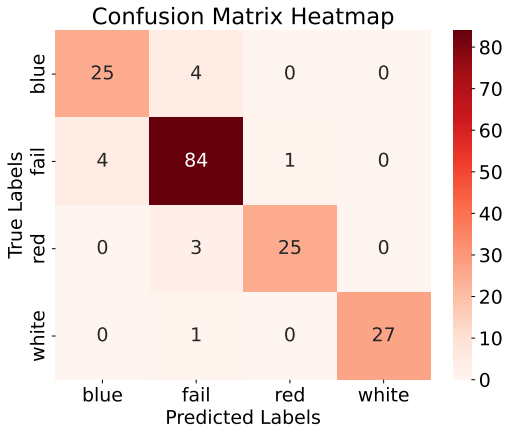


Figure 3: Confusion Matrix

	precision	*recall*	*f1-score*
blue	0.86	0.86	0.86
fail	0.91	0.94	0.93
red	0.96	0.89	0.93
white	1.00	0.96	0.98

Figure 4: Class-based performance metrics

Class-based Performance Metrics

A detailed breakdown of the model's class-based performance is shown in Figure 4.

For the blue class, the model achieved a precision, recall and F1 score of 0.86, indicating a balanced performance in correctly identifying blue class samples. However, the achieved score of 0.86 is by far the worst performance of any class.

The fail class was predicted with high accuracy, as indicated by a precision of 0.91, a recall of 0.94, and an F1 score of 0.93. The high recall suggests that the model successfully captures most of the fail class instances.

The red class had a precision of 0.96 and an F1 score of 0.93, although the recall was significantly lower at 0.89. This indicates that while most of the images that the model predicted to be red were in fact images of the red class, many red images were incorrectly predicted as another class.

The model performed exceptionally well for the white class, achieving perfect precision (1.00) and high recall (0.96), resulting in an F1-score of 0.98.

Confusion Matrix

The confusion matrix heatmap (Figure 3), created using (5) and (7), provides a visual representation of the model's predictions against the ground truth. Most of the predictions align with the true labels, as reflected by the strong diagonal structure. All misclassifications are in the fail class. This means that the model can perfectly distinguish the classes blue, red, and white. The fail class has very similar features to each of the other classes, so the model can't differentiate them as well. As already observed in the previous paragraph, the model performs worst on the blue class, with 8 out of 32 images misclassified.

Discussion of the Metrics

The main takeaway from the metrics is that although the model generally performs well, there is potential to make it even better. The slight overfitting on the training and validation data could be reduced by increased use of reguralization methods or a larger data set. The performance on the blue class could also be improved with a larger data set or methods such as class weights. Also, some of the errors made by the model could be reduced by not using any of the unclear images mentioned in chapter 2.2 for training, validation and testing.

5 Conclusion and future work

This study presented EOLVision, a CNN designed for automated end-of-line testing in industrial environments, which achieved a test accuracy of 92.5%. The model performed exceptionally well for most classes, with challenges mainly in the "blue" class. Dropout regularization and L2 weight decay proved highly effective in improving generalization and reducing overfitting, although slight overfitting was still observed.

The CNN architecture used stood out for its balance between complexity and robustness, outperforming other configurations. For future work, we recommend increasing the dataset size, using higher quality images, using more regularization and experimenting with other architectures. These steps could further improve the classification accuracy and reliability for industrial applications.

References

[1] Abadi, M., et al. (2016) TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv preprint arXiv:1603.04467.

[2] Chollet, F., et al. (2015) Keras. GitHub, `https://github.com/keras-team/keras`.

[3] Clark, A. (2015) Pillow (PIL Fork) Documentation. Read the Docs, `https://pillow.readthedocs.io/`.

[4] Harris, C.R., et al. (2020) Array programming with NumPy. Nature, 585(7825), pp. 357-362.

[5] Hunter, J.D. (2007) Matplotlib: A 2D graphics environment. Computing in science & engineering, 9(3), pp. 90-95.

[6] Pedregosa, F., et al. (2011) Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), pp. 2825-2830.

[7] Waskom, M.L. (2021) seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021.