Segmentation-Based Deep-Learning Approach for Surface-Defect Detection

TENSORFLOW IMPLEMENTATION

AmirHesam Kamalpour

Introduction

In this report, we implement a simplified version of the Segmentation-Based Deep-Learning Approach for Surface-Defect Detection proposed by Domen Tabernik, Samo Sela, Jure Skvarc and Danijel Skocaj. This approach has been shown to achieve state-of-the-art results on the Kolektor dataset. We evaluate the model on different training setups to investigate how different loss functions, optimizers and regularization techniques affect the model's performance.

The main contributions of this report are:

- Implementation of a simplified version of the Segmentation-Based Deep-Learning Approach for Surface-Defect Detection.

- Evaluation of the model on different training setups to determine the impact of different loss functions, optimizers and regularization techniques on model performance.

Dataset

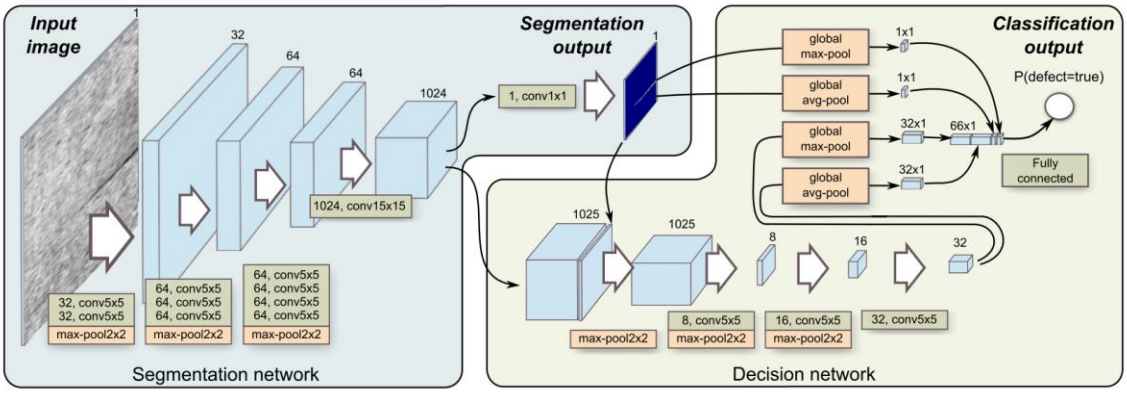
The KolektorSSD dataset is a publicly available dataset specifically designed for surface defect detection on electrical commutators. It consists of high-quality images with a resolution of 1408 × 512 pixels which were captured from 50 defected electrical commutators. Each commutator was captured in eight non-overlapping images, resulting in a total of 399 images.

The defects in the images are primarily microscopic fractures or cracks on the surface of the plastic embedding in electrical commutators. In two of the commutators, the defect is visible in two images, while in the remaining commutators, the defect is only visible in a single image. This means that there are a total of 52 images where the defects are visible.

Implementation

The proposed approach is formulated as a two-stage design; the first stage implements a segmentation network which outputs a mask with 8 times reduced resolution. The second stage, where binary-image classification is performed, includes an additional network that is built on top of the segmentation network and uses both the segmentation output(mask) as well as features of the segmentation net. The first-stage network is referred to as the segmentation network, while the second-stage network, as the decision network.

Firstly, the segmentation network is independently trained. Then, using transfer learning, we freeze the weights of the segmentation network and start training the decision model.



Segmentation network

The proposed network consists of 11 convolutional layers and three max-pooling layers that each reduce the resolution by a factor of two. Each convolutional layer except for the last one is followed by a feature normalization and a non-linear ReLU layer. The first nine convolutional layers use 5×5 kernel sizes, while the last two layers use 15 × 15 and 1 × 1 kernel sizes, respectively. The final output mask is obtained after applying 1 × 1 convolution layer that reduces the number of output channels to a single-channel output map with an 8-times-reduced resolution of the input image.

Decision network

The network takes the output of the last convolutional layer of the segmentation network (1024 channels so-called features) concatenated with a single-channel segmentation output map (mask). This results in 1025-channel volume that represents the input for the remaining layers with a max-pooling layer and a convolutional layer with 5×5 kernel sizes. Combination of both layers is repeated 3 times, with 8, 16 and 32 channels in the first, second and third convolutional layer, respectively. The output of the last convolutional layer results in a 64-times-smaller resolution of the last convolutional layer than that of the original image. Finally, the network performs global maximum and average pooling, resulting in 64 output neurons. Additionally, the result of the global maximum and average

pooling on the segmentation output map are concatenated as two output neurons. This design results in 66 output neurons. This architecture does not require regularization techniques, according to the authors. However, we tested L2 regularization with and without dropout to assess its impact on reducing overfitting. We also tested reducing the learning rate by a factor of 2 when validation accuracy stopped improving.

Data generator

The data generator is responsible for loading and preprocessing the data for training and validation. It takes as input the data directory, a list of positive data files, a list of negative data files, the batch size, the input shape, and the mode (either "Segmentation" or "Decision").

The data generator first shuffles the positive and negative data indices. Then, it iterates over the data in batches of the specified size. For each batch, it loads the images and masks from the specified data directory. The images are preprocessed by resizing them to the specified input shape and converting them to arrays. The masks are preprocessed by resizing them to a smaller size and binarizing them.

The data generator yields the preprocessed images and masks as a tuple. If the mode is "Segmentation", the data generator yields the images and pixel-level masks. If the mode is "Decision", the data generator yields the images and binary decision masks.

The first 30 subfolders are used as training sets, the remaining 20 for testing. The selection process was modified to ensure that the network observed a balanced number of defective and non-defective images. This was achieved by taking images with defects and without defects for every iteration. This mechanism ensures that the system observes defective images at a constant rate; otherwise the learning is unbalanced in favor of the non-defective samples. It should be noted that this leads to training that is not done exclusively by the epochs, as the number of non-defective images is 8-times higher than the number of defective ones and the network receives the same defective image before receiving all the non-defective images.

Learning details

We trained the segmentation network using stochastic gradient descent without momentum and a learning rate of 0.1 for the sigmoid cross-entropy loss. We trained the decision network using two different optimizers and architectures. The best results for decision network were obtained using the following configurations:

Stochastic gradient descent without momentum:

- Learning rate: 0.001
- Momentum = 0.0
- Loss function: Sigmoid cross-entropy loss
- No activation function used for the dense layer

Adam:

- Learning rate: 0.001
- Loss function: Binary cross-entropy loss
- No regularization
- Sigmoid activation function used for the dense layer

When we used the learning rate of 0.1 for SGD as the authors suggested, the model suffered from either vanishing or exploding gradients, resulting in NaN validation loss.

Results

The performance of the model was evaluated using 4 classification metrics:

- Accuracy: The proportion of samples that are correctly classified.
- Precision: The proportion of predicted positives that are actually positive.
- Recall: The proportion of actual positives that are correctly predicted.

- F1-score: The harmonic mean of precision and recall.

The following tables show the results of the Adam and SGD evaluation, respectively:

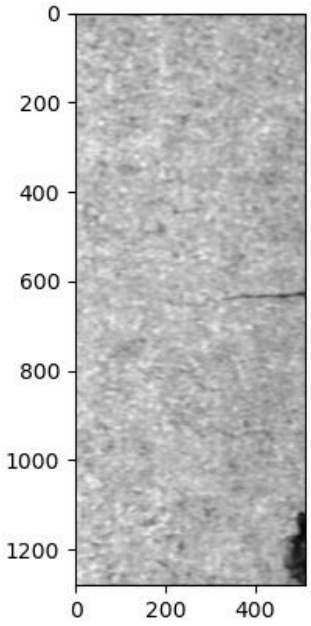
Metric	Value
Accuracy	0.988
Precision	0.986
Recall	1
F1-score	0.993

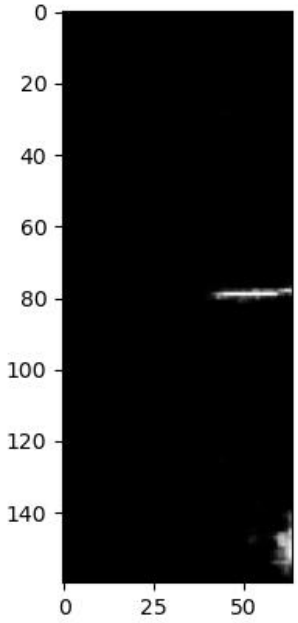
Metric	Value
Accuracy	0.963
Precision	0.972
Recall	0.986
F1-score	0.979

The following table shows the best results achieved during the training process of the models:

	Segmentation Net	Decision Net With Adam	Decision Net With SGD
Epoch	60	8	80
Validation Accuracy	—	1	1
Train Accuracy	—	1	1
Validation Loss	0.0071843	8.8048e-07	0.0055615
Train Loss	0.0010795	0.0001708	0.0065676

Sample Input/output images:





Conclusion

Our study delved into the impact of various optimizers, learning rates, and regularization methods on a deep learning model tailored for defect detection. The investigation revealed that the Adam optimizer, without regularization, surpassed the SGD optimizer with a learning rate of 0.001 in defect classification. Notably, the Adam optimizer demonstrated superior performance metrics, boasting an accuracy of 0.988, precision of 0.986, recall of 1.000, and an F1-score of 0.978, coupled with rapid convergence. In contrast, the SGD optimizer recorded an accuracy of 0.963, precision of 0.972, recall of 0.986, and an F1-score of 0.979, maintaining a consistent learning pace. Further, the inclusion of L2 regularization and dropout not only did not enhance the SGD optimizer's efficacy but also slowed down the training process. These insights endorse the Adam optimizer as the optimal choice for this model's training, given its heightened accuracy, precision, and swift convergence.

References

Domen Tabernik, Samo Sela, ure Skvarc, Danijel Skocaj (2019). Segmentation-Based Deep-Learning Approach for Surface-Defect Detection (https://link.springer.com/article/10.1007/s10845-019-01476-x)

ShuaiLYU (2019). Deep-Learning-Approach-for-Surface-Defect-Detection (https://github.com/ShuaiLYU/Deep-Learning-Approach-for-Surface-Defect-Detection.git)

Domen Tabernik (2019). Surface Defect Detection with Segmentation-Decision Network on KolektorSDD (https://github.com/skokec/segdec-net-jim2019.git)

Kolektor Surface-Defect Dataset (KolektorSDD) (https://www.vicos.si/resources/kolektorsdd/)