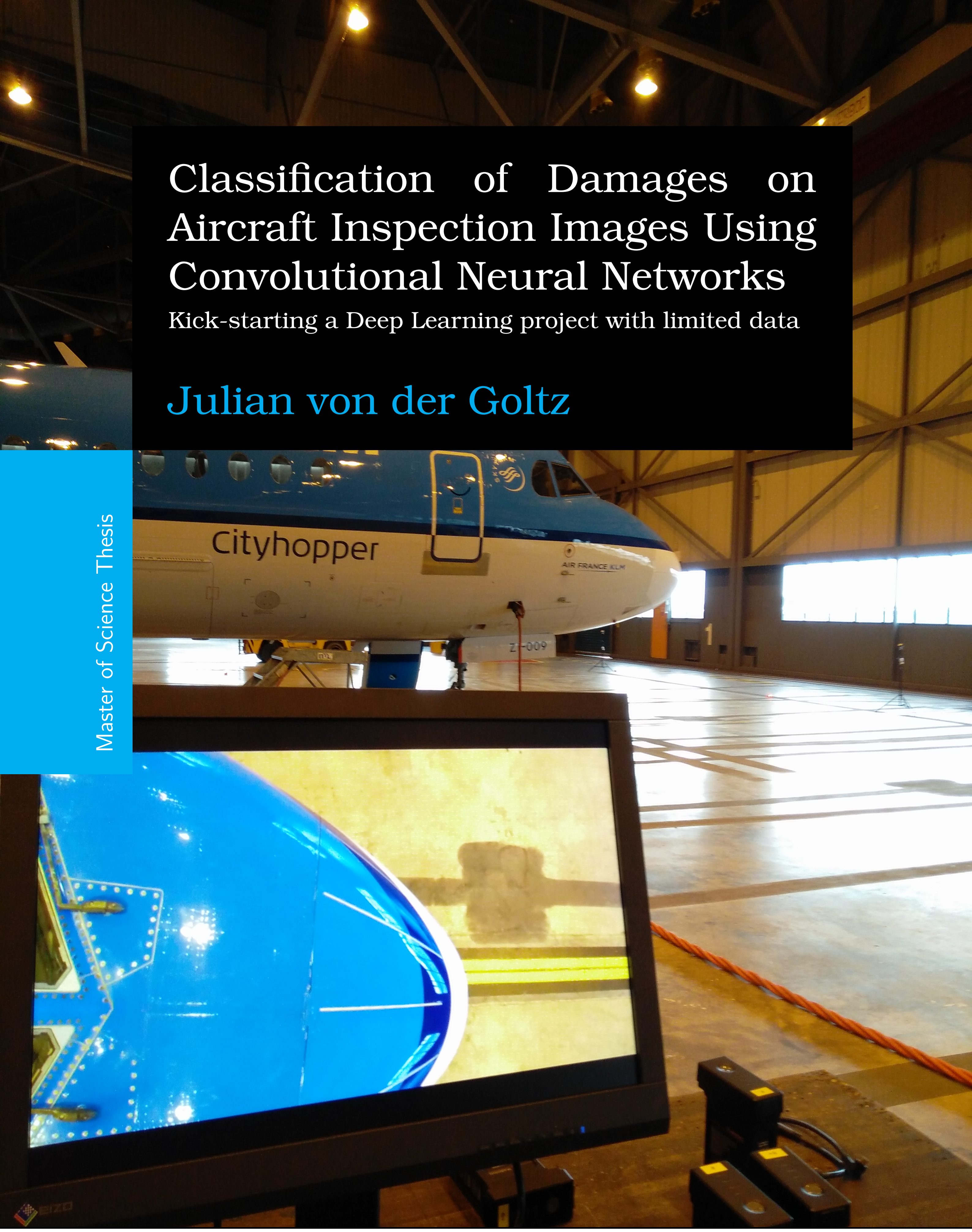


Classification of Damages on Aircraft Inspection Images Using Convolutional Neural Networks

Kick-starting a Deep Learning project with limited data

Julian von der Goltz

Master of Science Thesis

TU Delft Delft University of Technology

Department of Cognitive Robotics (CoR)

MSCCONFIDENTIAL

Classification of Damages on Aircraft Inspection Images Using Convolutional Neural Networks

Kick-starting a Deep Learning project with limited data

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Julian Henry Freiherr von der Goltz

August 14, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was conducted for Mainblades Inspections in The Hague. Their cooperation is hereby gratefully acknowledged.

Abstract

Aircraft inspections after unexpected incidents, like lightning strikes, currently require a time-consuming and costly inspection process, due to the small size of the lightning strike damages. Mainblades Inspections is working on an automated, drone-based solution, that scans the aircraft hull with a high-resolution camera. The objective of this project is to assess the feasibility of using a deep Convolutional Neural Network (CNN) for (semi-)automated damage detection, with the goal of achieving a high recall (low False Negative Rate (FNR)) on the small damages. The problem is framed as a classification problem on limited and imbalanced data. However, it is not pre-defined if single-label or multi-label classification should be used, and both approaches are investigated.

The main contribution of this work is to show experimentally how common deep CNN architectures and Deep Learning practices can be used to train classifiers that recognize damages in a specialized domain, with application-specific metrics. We present methods for synthesizing, pre-processing and re-sampling of the necessary dataset. It is shown that pre-trained, parameter-efficient CNN architectures that implement skip-connections, complemented by global max-pooling before the final layer, are well suited for that dataset. The Xception architecture has been chosen as backbone for the classifier due to its high recall and fast convergence.

To mitigate the detrimental influence of imbalanced training data, training data re-sampling that equalizes the class distribution is implemented. It has a positive effect recall, especially when applied to multi-label classification. When using re-sampling and data augmentation, the performance of multi-label and single-label classification can be brought to the same level. However, the best achieved FNR is 5.4%, with a softmax classifier, combining all regularization methods.

Finally, we investigated how regularization can be used to increase generalization capability with limited training data. Data augmentation is the most effective regularization method, even though its full potential has not been explored yet. Dropout benefits single-label classification but not multi-label classification. L_2-regularization has a moderate positive effect on both. Naively combining the regularization techniques without an exhaustive grid search or automated search on average does not yield any additional gains and shows the limit of manual hyper-parameter tuning.

Table of Contents

List of Figures

List of Tables

Preface

This Master of Science thesis is the result of one year of dedicated work on a very interesting topic and environment. When looking for a thesis topic, I wanted to combine my background and passion for Aerospace with Machine Learning, especially Deep Learning. I also had a preference for conducting the project in a start-up to have a good mix between freedom and a structured environment. I also wanted to see the outcome of my project being deployed in action at least in some way. I found Mainblades Inspections on the website of the DSA Kalman and immediately applied, they seemed to have exactly what I was looking for.

We have come a long way since then: In the beginning there was no data, and the problem was formulated accordingly. However, we managed to get access to more and more images, carefully labeled them and added them to an ever growing database. The first experiments were done with 250 labeled images, the experiments presented here were done on 1400 images, and as of now, we have more than 2200. The focus of this project shifted from the emphasis on the very-low data regime, until it converged to a topic that I feel honored to present in this thesis. On the way, I have gained a lot of knowledge and skills and I am in any circumstance proud of what I have achieved.

All this would not have been possible without the assistance of my supervisors Dr. Wei Pan and Dr.ing. Jens Kober during conducting this project and writing of this thesis. I would like to thank them and my company supervisors Ir. Jochem Verboom and Ir. Dejan Borota from Mainblades Inspections for mentoring and coaching and KLM Engineering & Maintenance for providing valuable data and input. I would also like to acknowledge Google Cloud Platform for providing computing resources as part of their Spark package for start-ups.

Lastly, a special thanks goes to the Deep Learning community that has helped me learn about the topics relevant for this thesis: Universities and companies publishing their lectures and research and open-sourcing their code bases (e.g. TensorFlow); institutions like www.fast.ai that approach Deep Learning from a perspective other than the large-scale, heavy-compute paradigm of the big tech companies; and finally, individuals like Lex Fridman (MIT), Andrej Karpathy (Tesla), Siraj Raval, Andrew Ng and many others that share their valuable experiences and practices.

Delft, University of Technology
August 14, 2019

Julian von der Goltz

Chapter 1

Introduction

Aircraft inspections after unexpected incidents (examples are lightning strikes or bird strikes) currently require a time-consuming process of towing the aircraft to the hangar, setting up the scaffolding and platforms and performing the inspections. For large, wide-body aircraft like the Boeing B-747 series, an inspection can take up to several hours for multiple technicians. Especially lightning strikes currently require an exhaustive manual search on the aircraft hull, because of their small size (in order of millimeters, up to a few centimeters), see Figure 1-1. In most cases, lightning strikes do not require an immediate repair, which results in an unnecessary down-time of the asset due to the inspection.

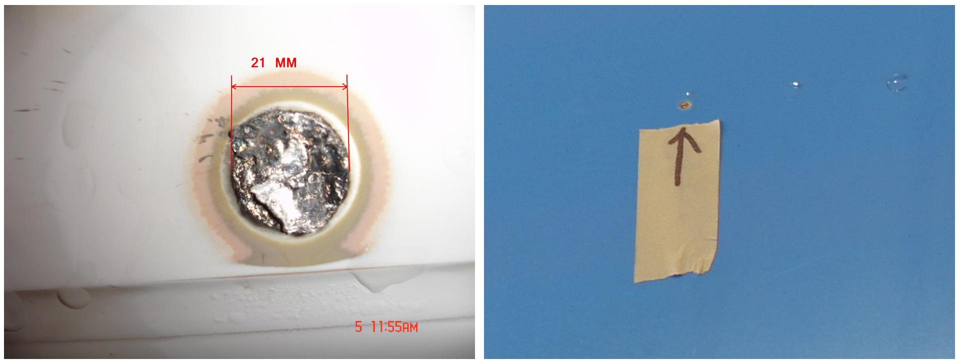


Figure 1-1: Examples of lightning strikes. They are characterized by a discoloration, scorching or melting of a very small area on the aircraft hull.

Mainblades Inspections is working on a drone-based solution, that will autonomously fly around the aircraft and scan the aircraft hull for damages with a high resolution camera. Currently, the company uses a DJI M100 drone and a 16 megapixel DJI X5 camera mounted to the drone on a gimbal, that takes high-resolution photos of the aircraft. This MSc-project is concerned with interpreting the acquired image data and visually detecting the damages. The objective of this project is to assess the feasibility of using a Deep Learning algorithm for (semi-) automated damage detection with the goal of achieving low False Negative Rate (FNR).

The following challenges are identified at the start of the project and govern decisions throughout this work:

- The number and size of damages to be detected is orders of magnitudes smaller than the surface area of an aircraft. Therefore, data points that correspond to the 'background' class are strongly over-represented. This makes it more challenging to achieve a low FNR on the damage classes.
- The available training dataset is small (but growing) compared to datasets typically used for training deep models, which poses a risk of over-fitting and lack of generalization. Currently, the dataset size is in the order of 1400 labeled images, containing approximately 18,600 objects.
- The quality of the available training images is not consistent and does not always match the quality of images taken during deployment with the high resolution drone camera. Many images are taken by low-resolution phone cameras or from a large distance. Some images also contain human made markings, that would not be visible during a drone inspection.

This work focuses on the first two challenges and the third one is left for a future project. For a proof of concept, the mismatch in quality between training and deployment environment can be neglected, and testing will be done within the available dataset. However, it is strongly recommended to address this mismatch, as it will lead to deteriorating performance of the detection model during deployment.

1-1 Background on Deep Learning

In recent years, Deep Learning, a sub-field of Machine Learning and Artificial Intelligence, had a massive increase in popularity due to improvements in computing power and availability of large datasets for training [1]. Deep Neural Networks extract features that are learned directly form data, instead of relying on a programmer to "hard code" those features. Deep Convolutional Neural Networks (CNNs) are on par with, or even exceeding human performance in some visual recognition tasks like the ImageNet challenge [2].

For the application of detecting damages on aircraft, we presume that, if it is possible to recognize damages on a drone photo with the human eye, it must be possible for a sufficiently advanced computer algorithm to recognize damages as well. Damage detection on visual data can be seen as an object detection problem, where objects are localized in bounding boxes and classified into different classes.

Deep Learning for detecting damages or anomalies in images has found applications in Structural Health Monitoring (SHM) for civil engineering to detect cracks in infrastructure [3]–[6], in the medical domain to perform brain tumor segmentation in MRI data [7], and in the financial sector to detect on-line transaction frauds [8], among many other examples. Major automotive manufacturers and start-ups around the world are heavily investing in research into autonomous driving that relies on visual detection of objects that surround the car, like streets and traffic signs, other vehicles, humans, buildings and many more.

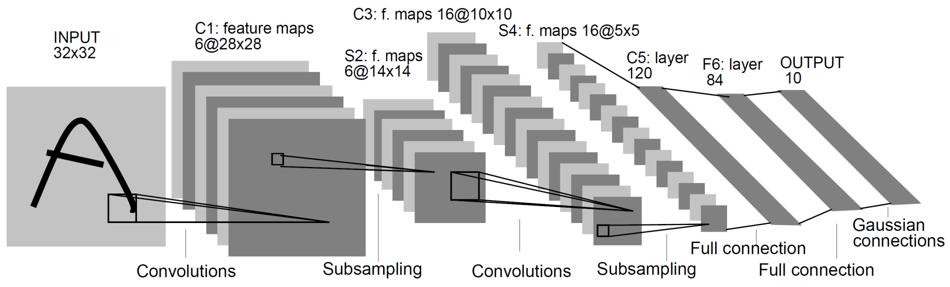


Figure 1-2: LeNet-5 Network Architecture by LeCun, Bottou, Bengio, *et al.* [9].

Convolutional Neural Networks (CNNs)

CNNs have the convenient property of being modular: a network trained for classification can be extended to perform detection. Therefore, classification is seen as the core task throughout this report, but always with the detection application in mind. Figure 1-3 shows example predictions of a CNN for object detection. CNNs are, just as other neural nets, black box models with a large parameter space (in the order of millions to hundreds of millions) that learn the important features directly from data.

CNNs, in contrast to fully-connected (FC) networks, use shared parameters to reduce the number of parameters by utilizing convolutional layers [9]. They exploit the three-dimensional structure of digital images (height, width and color channels), where a fully-connected network would operate on a flattened image vector. A typical CNN consists of a chain of convolutional layers alternating with sub-sampling operations like max- or average-pooling that reduce feature dimensionality and computational demands. Figure 1-2 shows schematically the architecture of the LeNet network proposed by LeCun, Bottou, Bengio, *et al.* [9].

CNNs still need large datasets for training to prevent over-fitting and provide generalization capability. Due to their data-hungry nature, they do not seem to be the best choice for small datasets. However, they exhibit properties that make them suitable and perhaps superior algorithms for this application:

1. CNNs have a large discriminative capacity which helps with dealing with noise and unknown model parameters, where model in this case is a 'visual model' that maps low-level information (image pixels) to high-level abstractions (damage labels). The damage detection algorithm is to be deployed in a wide variety of situations: Changing lighting conditions (i.e. daylight/night time, brightness of lamps or the sun, shading, weather, inside/outside) and changing appearance of the inspection object itself (i.e. aircraft color, damage size, hull materials used, dirt and corrosion). Instead of tuning a feature extractor by hand for all those situations, CNNs can extract the relevant features and abstractions directly from the data.
2. Due to regulatory constraints, the algorithm will be supervised by a human operator, which will provide valuable feedback on the algorithm performance and new ground-truth labels for training. Assuming that there is a continuous inflow of data from ongoing inspections and from situations like mentioned above, the training dataset will grow, and the algorithm can be continuously improved.

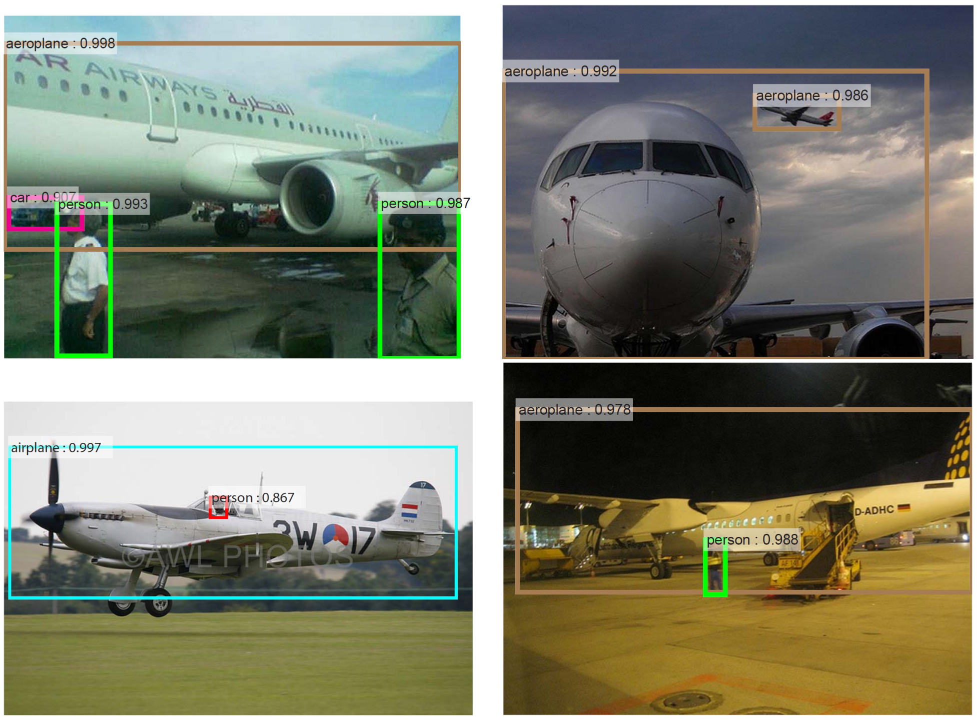


Figure 1-3: Example outputs of a CNN-based object detection model involving aircraft. The images stem from the PASCAL VOC 2007 test set [10]. The object locations are given as bounding boxes along with class labels and confidence scores. Source: Faster R-CNN [11]

3. Techniques like transfer learning, one-shot learning and regularization can help when little data is available. In combination with heuristics and domain knowledge, it is possible to design an initial CNN-algorithm that performs well on the target task.

1-2 Problem Formulation

For the initial proof-of-concept, the problem is framed as a classification problem, where bounding boxes from human annotated inspection images are extracted and classified on a per instance basis. This is to get a better grasp on the eventual classification performance and can be used as a starting point for localization of damages in a later stage.

Due to the safety critical nature of the industry, the goal of the algorithm to be developed is to have a high recall (sensitivity) or low FNR on the damage category. This is the main metric used to evaluate the different experiments. The dataset used for this work has 21 different classes. This includes different types of damages (lightning strikes, abrasion, cracks, dents, scratches, etc.), other objects of interest (text, markings, sensors, repairs) and a generic catch-all background class.

The following aspects make the classification problem of this work unique from image recognition problems presented in research literature, for example the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). The term *application-specific* used throughout the report, refers to these aspects:

- Dataset and its origin; in this case, real-world aircraft inspection images as opposed to publicly-sourced general images.
- The characteristics of the objects of interest; in this case, small sized damages on a high-resolution image, as opposed to objects that fill a significant portion of the image. The term size refers to pixel size and also to the relative size with respect to the original image dimensions.
- Evaluation metrics; in this case, recall on a certain category of classes, as opposed to top-5 accuracy.

Considering the application-specific aspects above and identified challenges, the problem formulation, serving as a guideline throughout this project, reads as follows:

"How can a *visual recognition algorithm* be developed, that reliably classifies *small damages* with *low False Negative Rate (FNR)*, given that there are *large class imbalances* present that bias against the damages, and given *limited training data* in general?"

1-3 Research Hypotheses

The problem formulation can be broken up into five parts. For each part, a set of assumptions are made that are based on previous related work and are therefore not tested. Additionally, each part is assigned a set of hypotheses that are experimentally tested:

I - Visual Recognition Algorithm The goal for this part is to find the basic architecture for an image classifier. It is inferred that a Deep Learning model provides a much better performance than the classical, feature-based Computer Vision (CV) approaches. Furthermore, it is assumed that transfer learning is an important way of initializing a deep CNN, when limited data is available. First, the following hypothesis is tested:

Hyp.A A complete network has to be trained instead of merely training the customized classification on a frozen backbone CNN with pre-trained ImageNet weights, due to the dissimilarity of features that characterize the ImageNet classes and the aircraft damages images dataset.

Using the result, a search over a selection of model architectures is conducted. The selected models are commonly used, the state-of-the art CNNs that also represent the two major CNN design philosophies.

II - Small Damages This part focuses on the application-specific building blocks that are added to the basic architecture. These building blocks are found in the *classification head* of the CNN-classifier. The goal is to find an optimal classification head for small damages, by testing the hypotheses:

Hyp.B Using global max-pooling as a sub-sampling step before the final classification layer leads to better performance than average-pooling on the objects of interest (small damages) due to its emphasis on local features. A combination of max- and average-pooling has at least the same, but possibly better performance, as it combines the advantages of both methods.

Hyp.C Using a convolutional classification layer that outputs a map of class logit scores, has higher performance on small objects and multi-label classification than a fully-connected classification layer fed with globally pooled features, that outputs a vector of class logit scores, due to the local connectivity of a convolutional layer.

III - Low FNR This part is focused on the choice and interpretation of the classifier output. The arguably most fundamental question to be asked is "what kind of classification approach should be used?", as this is not necessarily pre-defined by the data. The goal is to answer this question empirically with experiments.

A low FNR is equivalent to high sensitivity, meaning that the output of the model should be rather conservative and should not return a single, overconfident prediction that could be wrong, but rather a set of less confident predictions, that contains the ground truth in some way. Therefore, the hypothesis reads:

Hyp.D Multi-label classification with independent sigmoid activations is a more appropriate framework for this application than single-label classification with softmax activation. The reasons are that an extracted crop of an inspection image can contain more than one object, the classes (damages in particular) are not mutually exclusive and therefore the score of one class should not be influenced by the score of the other classes.

IV - Low FNR & Large Class Imbalances Increasing the sensitivity to lower the FNR goes hand-in-hand with the class imbalance problem, when the class of interest is under-represented. The goal is to find a simple and effective way of decreasing the FNR on the minority classes. It is assumed that data-based methods (e.g. re-sampling) are more effective for class imbalance problems than algorithm-level methods, when only a small dataset is available. The hypotheses to be tested are then:

Hyp.E Re-sampling to a uniform class distribution mitigates the risk of biasing towards over-represented data. In the process, the under-represented classes are over-sampled, which also increases the sensitivity to them, lowering the FNR.

Hyp.F Combining re-sampling with data augmentation, creates 'synthetic' examples of the minority classes, that mitigate the risk of over-fitting to the over-sampled minority samples.

V - Limited Data The goal of this part is to identify the effects of regularization methods to tackle the common problem of limited data. A naive approach is taken, to find the optimal hyper-parameters for regularization without an exhaustive grid-search: the regularization methods are each tested in isolation from the other methods. The best results are combined afterwards. The hypotheses to test are:

Hyp.G Extensively using regularization techniques (augmentation, dropout and L_2 regularization) increases performance in any case, due to the small size of the training set.

Hyp.H Naively combining the best results of using the regularization techniques in isolation, yields additional gains in performance.

Hyp.I The single-label vs. multi-label test (Hyp.D) is repeated: A multi-label approach with optimal parameters found for Hyp.G has lower FNR than a fully-optimized single-label approach.

1-4 Contributions

The main contribution of this work is to show experimentally how common deep CNN architectures and Deep Learning practices can be used to train classifiers that recognize damages in a specialized domain, and how these practices and design choices influence the performance in terms of application-specific metrics, given the challenges laid out above.

1-5 Thesis Structure

This report is structured in the form of an applied experimental research paper: Related work on damage detection using CNNs in literature is discussed in Chapter 2. Chapter 3 presents the methods used for creating the datasets, choosing networks for the experiments and discusses the technical implementation of re-sampling and regularization.

In Chapter 4, the set-up and results of all experiments to test the hypotheses are presented. Chapter 5 attempts to give explanations and interpretations of the experimental results, especially where a hypothesis is not confirmed by the experiment. Furthermore, Chapter 5 provides a qualitative analysis of some of the trained models and their outputs. Chapter 6 concludes with the findings of this work and answers the problem formulation.

In Appendix A, more detailed information about the dataset and its statistics is provided. Appendix B explains the method used to find optimal learning rates for the training runs. Appendix C contains the tables with the experimental results, and also tables and figures for experiments not discussed in the main matter. Appendix D and E show example predictions and probability maps for each class in the test set, respectively.

Chapter 2

Related Work

This chapter presents related work in research literature, and discusses its relevance for this work. Section 2-1 discusses the relevant Convolutional Neural Network (CNN)-models, Section 2-2 discusses meta-learning and transfer learning as methods to deal with small datasets, Section 2-3 discusses other applications in damage detection, Section 2-4 discusses methods to deal with large class imbalances and Section 2-5 discusses regularization methods to prevent over-fitting and increase the generalization of CNNs.

2-1 Convolutional Neural Networks (CNNs) for Image Recognition

The current generation of CNNs is based on the 'Neocognitron' proposed by Fukushima [12], an Artificial Neural Network (ANN) that is loosely inspired by the visual cortex of animals [13]. They are used to perform recognition tasks like classification and detection of objects on digital images. In contrast to fully-connected (FC) neural networks, CNNs share parameters across the height and width dimensions of the images.

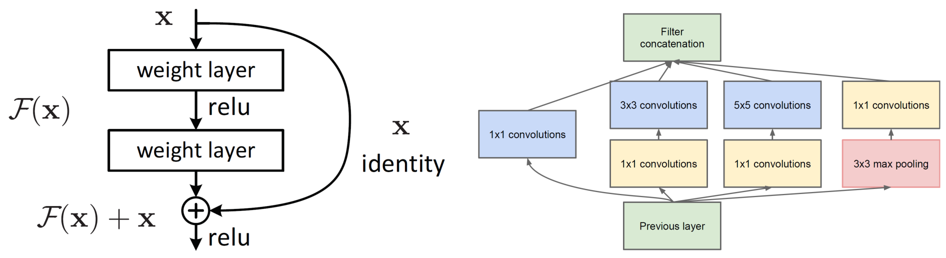
Back-propagation to efficiently calculate gradients of the loss function (e.g. negative log-likelihood, realized in the form of the cross-entropy loss as classification loss) is used in combination with Stochastic Gradient Descent (SGD) to optimize the model parameters [9]. Typically, a CNN-based classifier consists of a convolutional *backbone* and a *head*. The head can consist of one ore more fully-connected (FC) layers to output the final class scores.

Due to recent progress in available data, computing power and algorithms, CNNs have gained popularity in domains previously dominated by classical Computer Vision (CV) techniques. In 2012, AlexNet by Krizhevsky, Sutskever, and Hinton [1] was the first CNN to win the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [2] by a significant margin.

Architectural modifications in the form of residual blocks in ResNets [14], [15] and Inception blocks [16], [17] have been proposed as additional improvements. Residual blocks solve the problem of vanishing gradients by directly connecting lower layer parameters to higher layers and the loss function. This is achieved by using skip-connections, that directly map the input of a layer to its output. The convolutional layers then become residual layers that learn a transformation *added* to the input, rather than a transformation of the input itself. Inception blocks aim to reduce the number of parameters by decomposing a single convolutional operation with a high input depth into multiple parallel convolutions with smaller input depths.

Both these design philosophies have enabled the training of very deep networks with many layers, and form the basis for current state-of-the-art models. For example, Inception-ResNetV2 [18] and ResNeXt [19] combine Inception blocks with skip-connections to further improve the performance on the ImageNet benchmarks.

Depth-wise separable convolutions as implemented in the Xception network [21], reduce the number of parameters, by exploiting the properties of convolutional kernels being separable into point-wise convolutions and depth-wise convolutions. This is an extreme form of an Inception block as proposed in the InceptionV3 architecture [17]. Xception also uses skip-connections in the form of strided 1×1 convolutions to match dimensions between block inputs and outputs [21].



(a) Residual block **(b)** Inception block

Figure 2-1: (a) A residual block for ResNet as proposed by He, Zhang, Ren, *et al.* [14]. (b) An Inception block for GoogLeNet/InceptionV1 as proposed by Szegedy, Liu, Jia, *et al.* [16].

Densely-connected networks (DenseNets) [20] use concatenation of feature maps as a modification to skip-connections. This allows deeper layers to reuse features generated in lower layers and also decreases the number of parameters.

This work heavily relies on Inception and residual blocks, through the use of transfer learning, as discussed in the following Section.

EfficientNets by Tan and Le [22] do not introduce new architectural changes but propose a heuristic for optimally scaling CNNs under certain constraints or requirements. Their work is current state-of-the art and provides an empirical bound in the parameter-accuracy space for ImageNet models. Note that at the time of conducting the experiments, the work on EfficientNets was not yet released and available in Keras, which is why it does not appear in the methods and experiments of this work.

Figure 2-2 shows the ImageNet top-1 validation error against the number of parameters for CNN-models available in the Keras model zoo (`www.keras.io/applications`), complemented by the models of the EfficientNet-architecture, added in this visualization for reference.

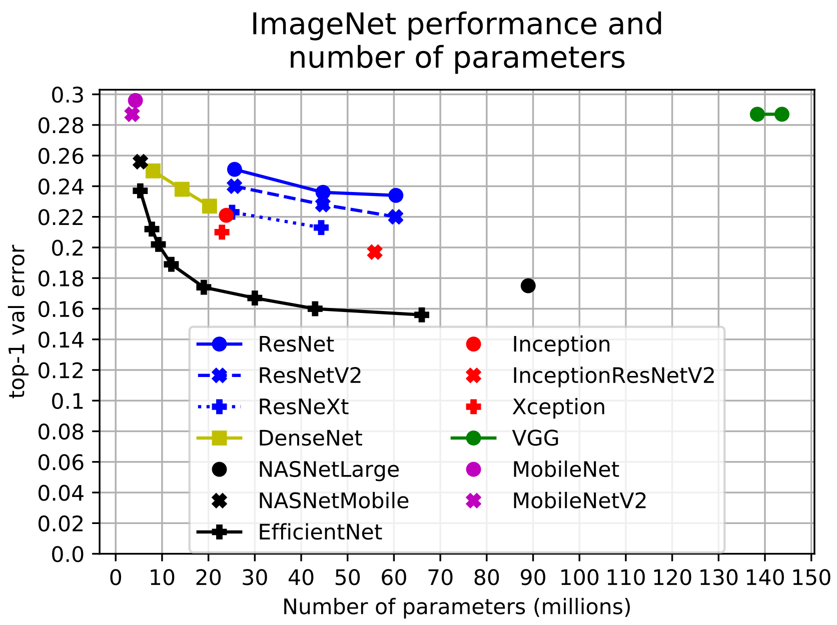


Figure 2-2: Comparison of CNN-architectures evaluated on the ImageNet validation dataset available in the Keras model zoo https://keras.io/applications. Plotted on the x-axis is the number of parameters and on the y-axis is the top-1 validation error. Where possible, members of the same family are connected by lines to visualize the effect of scaling. EfficientNets [22] are not part of the official model zoo and are added for reference.

2-2 Meta-learning and Transfer Learning

Meta-learning is a general term for learning methods that learn higher level properties of learning models, based on prior knowledge about other tasks or architectures. It describes the group of learning algorithms that learn hyper-parameters of an architecture (Automated Machine Learning (AutoML) [23] and Neural Architecture Search (NAS) [24], [25]), or that learn architectures that perform well across tasks (Learning-to-Learn, Low-Shot Learning, Transfer Learning) [26]. The latter is of particular importance for applications with small datasets. Knowledge from a so called *source task* can be used to perform, i.e. generalize, well on a *target task*, that has limited resources, like available data.

Learning-to-learn can be achieved by training a recurrent neural network to estimate the learning parameters [27], [28]. Alternatively, a meta-gradient outer loop can be added to the parameter update rule [29]–[31]. Both approaches are model agnostic. However, the latter does not require an additional network and can generalize better [32]. This makes it appealing for use as a standard tool for learning across tasks. Research in meta-learning across tasks is strongly dominated by low-shot learning, which is why many concepts are often interchanged between the two.

Low-shot learning is an extreme specialization of meta-learning, where a target task has to be learned with only a low number of training data points, usually in the order of 20 examples or less. The two main approaches for low-shot learning methods rely on a support set [33]–[38] or external memories [38]–[40], that represent prior knowledge. Knowledge in the support set is accessed by an attention mechanism and knowledge in the external memory is accessed by a Neural Turing Machine (NTM) [41]. Low-shot methods that do not rely on support sets or memories borrow methods from general meta-learning [42] and data augmentation [37], [42].

The literature shows that low-shot learning is effective at less than 20 novel samples. The available data for this work exceeds this number already by almost two orders of magnitude and it is steadily growing. Additionally, to the best of our knowledge, there is no publicly available dataset of damages that could be used to train a learning-to-learn algorithm to initialize the aircraft damage classifier. Therefore, this work does not make use of the meta-learning or low-shot-learning methods described above and relies on the popular and well studied transfer learning instead.

2-2-1 Transfer Learning

Transfer learning is a special case of meta-learning across tasks, where the model is optimized first on the larger scale and more general source task (*pre-training*) and is then optimized on the smaller, more specialized target task, initialized with the weights from the source task (*fine-tuning*). It has the restriction that the target model has to be structurally identical to the source model.

It is perhaps the most popular and well proven type of meta-learning: Off-the-shelf CNNs pre-trained on ImageNet are used by Razavian, Azizpour, Sullivan, *et al.* [43] to perform visual recognition tasks on other datasets. The CNN architecture called OverFeat [44] is augmented with a Support Vector Machine (SVM) that classifies the 4096-dimensional output of the CNN. They report on-par performance with other state-of-the art models in the respective tasks, without fine-tuning the CNN. Features learned on large datasets have a sufficient representational power and generalization ability to outperform recognition models using hand-engineered features [45].

According to Yosinski, Clune, Bengio, *et al.* [46], lower layers produce more general and transferable features, while higher layers are more specialized to a specific task. When transferring without fine-tuning the backbone, performance degrades the less similar the target task is to the source task. However, even for dissimilar tasks, initialization with pre-trained weights yields higher performance after fine-tuning than initialization with random weights.

Furthermore, ImageNet source performance is related to the performance on the target task, as found by Kornblith, Shlens, and Le [47]. In other words, a higher performing model on ImageNet can be expected to also perform well on classifying aircraft damages, for example.

Mahajan and Girshick [48] explore an extreme approach to pre-training, where ImageNet is the target task and the source task is a weakly-supervised, multi-label classification problem for Instagram hashtags on billions of images. They show that, even though the source training labels are noisy, the enormous scale of the source dataset results in state-of-the-art performance on the ImageNet target task. This is of course not relevant for our application, as there is no large scale, even weakly-supervised damage dataset of any sort. Nevertheless, it could be an interesting project to pursue across research institutions and industries in the future.

The current convention of always starting with pre-trained models is challenged by He, Girshick, and Dollár [49]. The authors train a Mask R-CNN [50] for an object detection and segmentation task. The model is trained from scratch and with pre-training on the ImageNet dataset. They find that pre-training does not necessarily improve final accuracy or help reduce over-fitting when compared to training from scratch. However, pre-training does accelerate convergence and helps reduce over-fitting in small data regimes.

The literature suggest that initializing an aircraft damage classifier that has a small training set with pre-trained weights will provide a good starting point in any case. The small size of the dataset speaks for freezing the backbone and training a classifier on top of the CNN-layers. However, the target task is quite specific and different from the ImageNet classification problem, so it is questionable if features are transferable and freezing lower layers is useful. Therefore, both freezing and training the backbone CNN are investigated in this work (Hypothesis Hyp.A).

2-3 Applications of CNNs in Damage Detection

Automated visual inspections for damages has been done mainly in the area of Structural Health Monitoring (SHM), where cracks are of high importance. This section discusses related applications of CNNs in crack damage detection. Early crack detection models are based on edge-detectors [51]. Abdel-Qader, Abudayyeh, and Kelly [52] analyze four edge-detection techniques (Sobel, Canny [53], Fast Fourier Transform (FFT) [54] and Fast Haar Transform (FHT) [55]) for crack identification in bridges and find that the FHT has the best performance in terms of accuracy.

Zhang, Yang, Zhang, *et al.* [3] train a six-layer CNN as image patch classifier and use its class probability maps to detect cracks. They compare their proposed method to a SVM and a Boosting method that perform classification on hand-engineered feature descriptions based on color and texture of the patches. The CNN outperforms the other two methods in both precision and recall. However, the feature descriptors might have been poorly chosen for classification on cracks.

Dorafshan, Thomas, and Maguire [6] compare an AlexNet-based CNN to a selection of edge-detectors for image-based concrete crack detection. The detection is done in the form of classifying cropped sub-images from the source image. The CNN crack detector is implemented in three different modes: Full training from scratch, freezing the pre-trained convolutional backbone and classifying on the CNN features (classifier mode) and fine-tuning the pre-trained weights of the complete network (transfer mode). The best performance is achieved in transfer mode, which also outperforms the best edge-detection method. Their findings are confirmed by Gao and Mosalam [56], that also report better performance for a fine-tuned model than for a frozen model in classifier mode.

Cha, Choi, and Büyüköztürk [4] train a CNN-classifier on crops of images of buildings to detect cracks in the concrete. To perform the localization of the cracks, they use a sliding-window technique. In a continuation of their work, Cha, Choi, Suh, *et al.* [5] use a Faster Region-based Convolutional Neural Network (R-CNN) [11] for detection of multiple damage types. The Faster R-CNN framework not only allows end-to-end training on the images without cropping them into sub-patches, it also provides detections in the from of bounding boxes.

The literature on damage detection using CNNs stresses the importance of transfer learning, in particular fine-tuning pre-trained weights on the damage detection task. Although ImageNet weights already provide good network initializations for most applications, the need for a public, large-scale structural damage image dataset can be identified. This could provide an even better starting point specifically for the training of CNNs in this application.

2-4 Class Imbalance

Class imbalance refers to the problem in machine learning, where the frequency of examples of one class in the data is not equal to the frequencies of examples of other classes. Class imbalance can result in a degradation of performance on the *minority* or *under-represented* classes. Reasons for degradation can be the choice of cost function and performance metric. The conventional loss functions like cross-entropy loss and metrics like accuracy bias towards the majority classes, because they have a higher influence on the gradient calculations than the minority classes.

Another reason for degradation is that minority classes are hard to learn, because they are not separable from other classes. When feature dimensionality is high, or the minority classes are indistinguishable from noise, the degradation is amplified. Also, the influence of imbalance is more profound with more complex data [57].

Generally speaking, two categories for imbalanced learning can be identified: Methods that operate on the data-level, i.e. perform sampling and possibly additional operations on the training data to equalize class frequencies and methods that operate on the algorithm-level, i.e. perform different learning or prediction operations depending on class frequency.

Re-sampling is a data-level methods that aims at balancing the class distributions in the training data. The options are over-sampling and under-sampling. Over-sampling creates additional examples of the minority class and under-sampling rejects examples of the majority class. A combination of these are referred to as hybrid sampling.

Haixiang, Yijing, Shang, *et al.* [58] have found that re-sampling in general and over-sampling in particular are the most popular choices for class imbalance techniques. The Synthetic Minority Over-Sampling Technique (SMOTE) [59] is a technique used when only few minority examples are available. Synthetic examples are created by interpolating between the extracted features of a data point and one of its neighbors, both belonging to the minority class. Pure over-sampling introduces the risk of memorizing and over-fitting to the minority class. By interpolating and creating ‘synthetic’ data, SMOTE aims to overcome that risk. However, Wong, Gatt, Stamatescu, *et al.* [60] find that performing known transformations in data-space (if available) outperforms transformations in feature-space.

Under-sampling randomly removes samples of the majority class for training to balance the class distribution for training. The disadvantage is that data will be discarded that can be useful in other ways for training. Modifications of under-sampling like one-sided selection [61], aim to mitigate this by removing samples on the classification boundary and samples that are noisy. This leaves only the samples of the majority class that are useful for classification.

For learning algorithms that can be trained with SGD and mini-batches, class-aware sampling [62] is another candidate sampling technique. It samples a uniform class distribution for every *mini-batch* when calculating the gradients, basically combining over- and under-sampling.

An example of an algorithm-level method that is applied during training time is **cost-sensitive learning**: mis-classification of a minority class sample is assigned a higher cost than misclassification of a majority class sample. In its simplest form, the cost assigned to a class is proportional to the inverse of the sample frequency of that class.

More complex cost assignments are possible; for example *focal loss* dynamically scales the cross-entropy loss with a factor that gives confident predictions less weight [63]. It is specifically designed to mitigate the effect of foreground/background class imbalances in dense single-stage object detectors, where background is not filtered by region proposals.

Another method of cost assignment is a cost matrix C that assigns cost $C_{i,j}$ to mis-classifying class i into class j. These cost matrices are more difficult to set up and may require expert knowledge. Khan, Hayat, Bennamoun, *et al.* [64] solve this problem by introducing a novel cost function that is jointly learned with the neural network parameters from data.

Ensemble methods like AdaBoost (adaptive boost) [65] and Bagging (bootstrap aggregating) [66] are algorithm-level methods that increase the performance of single classifiers. RUSBoost [67] and UnderBagging [68] are simple ensemble methods combined with under-sampling, specifically designed for imbalanced data, that stand out in experiments conducted by Galar, Fernandez, Barrenechea, *et al.* [69], against more complex contenders.

Ensemble methods have the disadvantage that multiple network models have to be trained, which is computationally expensive for deep neural nets, that usually require days to weeks to train. However, ensembles can also be be built from different training checkpoints or initializations that are created as by-products of Deep Learning.

Although there are many more methods for cost assignment, and ensembling can provide a boost on imbalanced data, the approach pursued in this work will be data-based. Under-sampling (rejection sampling) is not practical on the available dataset because it would mean to discard more than 90% of the training data due to the strong imbalances (see Appendix A).

Over-sampling would result in the reverse, increasing the training dataset by a factor of 10. This would make it difficult to compare the training losses and metrics with runs that are not re-sampled due to the different number of steps per epoch.

Instead, a combination of under-sampling and over-sampling (hybrid sampling) is implemented that is comparable to class aware-sampling [62], that over-samples minority classes and rejects majority samples and keeps the overall number of samples per epoch constant, to restrict the computational cost of pure over-sampling.

Rejection of majority classes happens during the course of a complete training run, rather than within the course of one epoch, to make sure no data discarded. The advantage of this approach above algorithm-level methods is that we can combine it with data augmentation to create a variation of synthetic samples of the minority classes that helps against over-fitting.

2-5 Regularization

Regularization techniques are techniques that aim to increase the generalization capability of a machine learning model, i.e. increase the performance on data that the model has not seen during training. This is usually done at the cost of decreased performance on the training data.

Regularization reduces the variance of a machine learning model, but increases the bias [70]. This Section discusses four of the most commonly used regularization techniques: Data augmentation, dropout regularization, L_2 weight regularization and early stopping.

2-5-1 Data Augmentation

Data augmentation is the process of adding transformed samples of the original training data during the training process. These transformations preserve the class label of the input, but have a different appearance than the original image, which helps with generalization.

In their landmark paper on CNNs, Krizhevsky, Sutskever, and Hinton [1] have already emphasized the importance of data augmentation to improve the generalization capability. Two forms of augmentation are used for their AlexNet model: The first is generating random image translations and horizontal flips or reflections to make the model invariant against geometrical changes. The second is randomly changing the intensities of the RGB-channel along the first principal component axis to make the model invariant against color and illumination changes.

The possible transformations that can be applied to perform data augmentation are only restricted by the creativity of the designer, many types of augmentations are conceivable. Howard [71] use a cropping and resizing technique that uses all of the given information in one image and apply additional contrast, brightness and color transformations. They also perform training and predicting at varying image resolutions, which introduces scale invariance to the model.

Wong, Gatt, Stamatescu, *et al.* [60] show that augmentation is most effective in data-space, and not in feature-space (as used in SMOTE [59]). Improvements made by augmentations are bound by adding real data to the training process. In other words, obtaining real images beats creating synthetic images.

Cubuk, Zoph, Mane, *et al.* [72] propose a method to automate the design of transformations for augmentation by learning it from data. The approach called AutoAugment aims to find optimal augmentation policies and sub-policies by performing a rigorous search. The policies consist of operations (like translating, rotating, colour transformations, etc.), the probabilities of performing the operation and their magnitudes. Searching the policies is inspired by Reinforcement Learning and other forms of architecture search [24].

Due to the small dataset size available in this work, AutoAugment is inapplicable. Instead, hand-engineered augmentation is employed in this project. To keep the number of hyperparameters low, the focus lies on geometrical (shift, rotational, scale and flip) transformations, but the findings will be extensible to color space augmentations as well.

2-5-2 Dropout

Dropout regularization is another commonly used regularization technique for CNNs proposed by Hinton, Srivastava, Krizhevsky, *et al.* [73] and Srivastava, Hinton, Krizhevsky, *et al.* [74].

The output of a node of the layer before the dropout layer is randomly dropped with a drop rate p_{drop} in each training iteration. During inference, all outputs are combined again. To ensure the same magnitude of activations at training and inference time, the output of the retained units during training is scaled with the inverse of the dropout rate.

Dropout can be applied at every location in the network but is often put right before the final FC layer. Dropout encourages neurons to not 'co-adapt' to the outputs of other neurons by only including random subsets of neurons in every iteration. It can also be seen as a way of training randomly sampled sub-networks at the same time. At inference time, the sub-networks are combined in an ensemble for increased performance.

2-5-3 L_2 Weight Regularization

L_2 weight regularization is a more classical way of performing regularization on machine learning models, but is also an effective method to improve generalization in neural networks [75]. It adds the squared L_2 norm of the network weights θ to the loss function with a coefficient λ (Equation 2-1). This encourages the network to learn small weights and use more features.

$$L = L(Y, \hat{Y}) + L(\theta) = L(Y, \hat{Y}) + \lambda|\theta|_2^2 \tag{2-1}$$

From an optimization perspective, it puts a constraint on the parameters and adds a convex term that simplifies gradient descent and provides a global optimum. From a machine learning and statistics perspective, it reduces the models complexity or capacity by introducing prior knowledge about the weights being normally distributed around zero [70].

When using vanilla SGD, L_2 regularization is also called *weight decay*, because a subtraction term $-\frac{1}{2}W$ is added to the parameter update step. However, when using adaptive or momentum-based optimizers like Adam [76], the L_2 term also influences the momentum term, such that weight decay and L_2 regularization are not equivalent anymore, which has been exposed by Loshchilov and Hutter [77]. By decoupling weight decay from the loss function optimization, one can tune the learning rate and λ independently, which leads to improved generalization.

2-5-4 Early Stopping

Early stopping refers to terminating the training of a neural network when a certain *termination criterion* is reached on the validation data, rather than the training data.

As deep neural networks are heavily over-parameterized, they could theoretically train on the training data until they reach perfect training loss. However, after a certain amount of epochs, the validation loss starts to increase and the validation accuracy decreases. When the validation metric stops to improve, training is terminated and the best checkpoint in terms of validation metric is returned as a final result.

Early stopping is essentially a method of determining the optimal number of epochs as a training hyper-parameter and is a regularization technique on the model that restricts the final model parameters to be within a certain bound from the parameters at training begin

[70]. It is a simple technique at almost no additional cost and is used for all experiments in this work.

Chapter 3

Method

In this chapter, the method used to set-up a damage classification algorithm is discussed: First, a dataset is compiled, then a model is designed consisting of a backbone feature extractor transferred from ImageNet and a customized classification head. The model is trained on the Mainblades dataset, applying re-sampling and regularization in different magnitudes to find an optimal training configuration.

Section 3-1 describes the creation of the classification dataset. Section 3-2 discusses the available feature extractors and Section 3-3 the classification head on top of the feature extractor. Section 3-4 presents the re-sampling technique used and Section 3-5 describes the implemented regularization methods.

3-1 Dataset

The source data consists of 1418 inspection images, the majority provided by KLM Engineering & Maintenance, but a minor part was taken by the drone itself. The official Structural Repair Manual (SRM) of aircraft manufacturer Boeing is used as a basis for the labeling. It contains a description of 27 damages that can occur during aircraft operation. Additionally, other objects like text, markings are labeled as well.

Each inspection image is labeled such that it is compatible to the format of most modern object detection frameworks (e.g. TensorFlow Object Detection API [78]). The label consists of a set of bounding boxes, expressed as pixel coordinates, normalized by the image height and width: (y_{min}/h, x_{min}/w, y_{max}/h, x_{max}/w). Each bounding box is assigned a class label encoded as an integer.

20,000 bounding boxes are obtained after labeling. After filtering irrelevant classes like the aircraft itself, around 16,000 bounding boxes are extracted for training and testing. As a reference for the 'negative' class, the same amount of background patches with random sizes is randomly sampled from all images.

An output image is obtained by extracting a bounding box by keeping its original pixel size and padding it with the per-channel-means of the source image to the output size where necessary. Over-sized boxes are down-scaled with bi-linear interpolation and fixed aspect ratio and then also padded to the output size.

The target labels y for each instance are created in two ways:

1. For single-label classification, i.e. the classes are mutually exclusive and thus their probabilities sum up to one, the class name of the bounding box is encoded into an integer. During training, the class label is encoded into a one-hot label-vector for the neural network target. The resulting dataset is denoted **DatasetV1** and contains only the class text and integer label for each image.
2. For multi-label classification, i.e. the class probabilities are independent of each other and more than one class can be present, the bounding box is checked for overlap from other bounding boxes with a different class label. Instead of setting the label-vector entries to 1 for the classes that overlap, a 'smooth label' approach was taken to avoid over-confident predictions of the neural net. For each class, the value in the label vector is proportional to the total Intersection-over-Area (IoA) of all boxes of that class with respect to the current box. The resulting dataset is called **DatasetV2** and can still be used to train a single-label classifier, by using only the label of the original bounding box and neglecting the overlapping boxes.

DatasetV1 and DatasetV2 contain different splits of the data and also different background samples due to the use of different random seeds. Networks that are trained on different datasets should not be compared to each other. DatasetV1 is only used for the experiments in Section 4-2, to find the best Convolutional Neural Network (CNN) model as feature extractor.

After extraction and adding background examples, the classification dataset consists of about 30,000 images in 21 classes. The classes are categorized, where all damages classes are categorized into one generic damage category. Finally, the data is shuffled, and 10% of the samples of each class are assigned to the test set (for final model comparison), 20% are assigned to the validation set and 70% are used for training.

A detailed description of the dataset with statistics can be found in Appendix A.

3-2 Backbone Feature Extractor

The backbone feature extractor generates features h from the input image x. The classifier (or detector) then can use the features h to classify x.

A CNN is used as a feature extractor as it can learn the relevant features from the training data. The CNN can be seen as a function f, parameterized by its weights and biases θ, that maps the input image x into feature vector h (Equation 3-1).

$$h = f_\theta(x) \tag{3-1}$$

The choice of backbone architecture is done on the basis of computational constraints. ResNet50 [14] with its original input resolution of 224×224 and batch size 24 fits into one Tesla P100

Table 3-1: Comparison of CNN-architectures evaluated on the ImageNet validation dataset.

Architecture	**Input shape**	**Feature map shape**	**Parameters**	**ImageNet val. error**
DenseNet121 [20]	$224\times224\times3$	$7\times7\times1024$	**8,062,504**	0.25
Inception V3 [17]	$299\times299\times3$	$8\times8\times2048$	23,851,784	0.221
ResNet50 [14]	$224\times224\times3$	$7\times7\times2048$	25,636,712	0.251
ResNeXt50 [19]	$224\times224\times3$	$7\times7\times2048$	25,097,128	0.223
Xception [21]	$299\times299\times3$	$10\times10\times2048$	22,910,480	**0.21**

GPU with 16GB memory. Four other models that have similar memory footprint and performance on ImageNet and are available in the Keras model zoo (www.keras.io/applications) are chosen: ResNeXt50 [19], DenseNet121 [20], InceptionV3 [17] and Xception [21].

This selection of models represent the two main design approaches of CNNs: ResNet, ResNeXt and DenseNet are representatives for residual networks that make use of skip-connections and InceptionV3 and Xception represent the design philosophy around inception blocks. Those approaches are not necessarily mutually exclusive; the Xception network also uses skip-connections.

Table 3-1 gives an overview over the network characteristics. The DenseNet121 is a special case in this collection: it has significantly less parameters than the other models (8 million vs. 23-26 million). In terms of parameter count, DenseNet169 and DenseNet201, with 16 million and 20 million parameters, respectively, are more comparable. However, they have a significantly increased GPU memory usage due to the concatenation of feature maps in between convolutional layers. This makes them impractically slow to train on the available hardware and therefore, the smaller DenseNet121 model was chosen as a representative for that family.

The selected models are customized to the Mainblades target task as follows: The pre-trained weights are downloaded, the last classification layer for the ImageNet classes is removed, a customized classification head (Section 3-3) is added instead, and all weights are re-trained on the target task.

As all networks are initialized with their respective ImageNet weights, the processing (reshaping and standardizing) of the input images is also performed according to their ImageNet implementation. This means that for some networks, the input is centered and scaled with the ImageNet dataset means and standard deviation.

3-3 Classification Head

On top of the convolutional backbone, a classification head is added. It converts the extracted features into class probabilities, by means of a global pooling, a classification layer and a final activation function.

In contrast to the feature extractor, which can be universally used on different tasks, the classification head is application-specific and depends heavily on the problem formulation, for example type of classification, number of classes, etc.

3-3-1 Global Pooling Layer

The pooling layer reduces the three-dimensional feature map of the last convolutional layer to a one-dimensional feature vector. The size of the vector depends on the number of features n_{feat} from the conv-layer. It is implemented as:

- Max-pooling, i.e. taking the maximum of all activations along the height and width dimensions. This emphasizes local features.
- Average-pooling, i.e. taking the average of all activations along the height and width dimensions. This emphasizes features that span over a wide area in the feature map.

A third option is concatenating the max-pooling and average-pooling, which combines the advantages of both and lets the dense layer learn the optimal combination of max-pooling and average-pooling.

3-3-2 Classification Layer

The classification layer has weights W_{cls} and biases b_{cls} and converts the feature activations h into a logit or raw score s_c for each class c. Its input shape depends on the size of the previous layer. The output shape is equal to the number of classes K.

Usually, the classification also includes an activation function. In this work, the activation function is considered separately, because the problem type (single-label or multi-label classification) is not pre-defined.

Therefore, the classification layer is simply a linear layer that can be expressed as:

$$s = hW_{cls} + b_{cls}. \tag{3-2}$$

3-3-3 Final Activation Function

The final activation function is the last element of a neural network that converts logits into class probabilities. There are two options, dependent on what problem type is used: For single-label classification, i.e. the classes are mutually exclusive and thus their probabilities sum up to one, the softmax activation is used. It estimates a conditional discrete probability distribution $\hat{p}(c|x)$ across the classes, where c is the class encoded as integer and x is the input image.

We define the vector of logit score as s, the one-hot-encoded ground truth label-vector as y and the number of classes as K. The subscripts i and j denote the ith and jth entries of the corresponding vector, respectively. Then the softmax activation can be written as in Equation 3-3.

$$\hat{y}_i = \hat{p}(c = i|x) = \frac{\exp(s_i)}{\sum_{j=1}^{K} \exp(s_j)}. \tag{3-3}$$

The output of the softmax layer is $\hat{y}$, the predicted label-vector and the predicted class is $c = \arg\max_i \hat{y}_i$.

The per-sample loss function that corresponds to single-label classification activated by the softmax function is the categorical cross-entropy, given in Equation 3-4.

$$\ell(y, \hat{y}) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{3-4}$$

Alternatively, the problem can be seen as a multi-label classification task, where each class probability is independent of the other classes. In this case, more than one class can be present and therefore, there are K different binary classification problems. For each class, an independent probability $\hat{p}(c|h)$ is estimated with the logistic sigmoid activation (short sigmoid), given in Equation 3-5.

$$\hat{y}_i = \hat{p}(c = i|x) = \frac{1}{1 + \exp(-s_i)}. \tag{3-5}$$

The predicted class for image x are all classes c where $\hat{y}_c \geq t$, with t being the classification threshold. In this work, the threshold is set to $t = 0.5$ without further tuning or experiments to determine an optimal value.

In `tf.keras`, the mean of K binary cross-entropy terms is used for the multi-label classification loss, given in Equation 3-6.

$$\ell(y, \hat{y}) = -\frac{1}{K}\sum_{i=1}^{K} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i). \tag{3-6}$$

The batch loss is obtained by averaging the per-sample losses over one mini-batch with batch-size N:

$$L(Y, \hat{Y}) = \frac{1}{N}\sum_{i=1}^{N} \ell(y^i, \hat{y}^i), \tag{3-7}$$

where y^i and $\hat{y}^i$ are the true label and predicted label of sample i, respectively. L is minimized by optimizing the parameters θ with Stochastic Gradient Descent (SGD).

3-3-4 Order of Classification and Pooling

For the overall layout of the classification head, it is possible to build two different configurations: the conventional option, dubbed here as **fully-connected (FC) classification** looks as follows:

Backbone CNN $\rightarrow$ global pooling layer $\rightarrow$ $n_{feat} \times K$ FC classification layer $\rightarrow$ activation function $\rightarrow$ loss function.

This configurations collects features that support a class probability *globally*. They are combined into a *global* logit score for each class, which is then finally converted into a probability. The reasoning behind this is that 'evidence' for a certain class can be spread out across the image, and with this method, one can leverage the combination of global features.

However, there is a disadvantage to this approach: if more than one small object is present in an image, the features of the different instances that are present locally are combined globally, which results in wrong scores and possibly wrong predictions, especially when training is done on only one instance per image.

An alternative approach, where one is interested in the probability that a certain small object (or several objects) is present somewhere, is to switch the order of classification and pooling and convert the FC-classification into a point-wise (1×1) convolution (**convolutional classification**):

Backbone CNN $\rightarrow 1 \times 1 \times n_{feat} \times K$ conv. classification layer $\rightarrow$ global pooling layer $\rightarrow$ activation function $\rightarrow$ loss function.

This configuration collects features that support a class *locally.* The logit score for the class is calculated *at each location* in the feature map. Then, for each class, the logit score is collected globally and converted to a probability. The hypothesis is, that this method works well for multi-label classification and small objects, but could give raise to poor performance on larger objects, because their features more spread out.

3-4 Re-sampling

To counter the detrimental effect of class imbalances in the training set, a simple re-sampling technique is implemented to equalize the class distribution, and therefore improve performance on the minority classes. A combination of under-sampling and over sampling is used that ensures that the number of total samples in one epoch is still the same:

First, samples of the minority class are repeated, such that their number is equal to the desired number of samples per class in a uniform distribution. The number of times a sample of class i is repeated is:

$$n_{\text{repeat},i} = \text{ceil}\left(\frac{1}{f_{\text{initial},i} \cdot K}\right), \tag{3-8}$$

where $f_{\text{initial},i}$ is the initial normalized frequency of class i in the training set, and K is the number of classes. The ceil() operator makes sure that samples of the majority classes ($f_{\text{initial},i} > 1/K$) are repeated exactly once and not discarded entirely. The new frequencies f_{repeated} of the repeated dataset are then:

$$f_{\text{repeated},i} = \frac{n_{\text{repeat},i} \cdot f_{\text{initial},i}}{\sum_j^K n_{\text{repeat},j} \cdot f_{\text{initial},j}} \tag{3-9}$$

Second, the complete dataset is repeated n_{epochs} times (which is an arbitrarily high number as the training is terminated based on convergence in the validation loss) and then the samples of the majority class(es) are rejected with a probability $p_{\text{reject},i}$:

$$p_{\text{reject},i} = 1 - \frac{\min(f_{\text{repeated}})}{f_{\text{repeated},i}} \tag{3-10}$$

With this particular order of repeating and rejecting, it can be ensured that all samples in the majority class have a chance of appearing in a training epoch, and when training long enough, no data is discarded. It also introduces randomness in the data that can help with regularization.

3-5 Regularization

The following regularization methods are applied in this work:

- **Data Augmentation**: Augmentation is implemented by sequentially applying the following geometrical transformations, each with a probability p_i:

  1. Translating the image in x and y directions by a random amount, such that at least 50% of the original image remains.
  2. Rotating the image by a random angle, drawn uniformly from $[-180°,\ 180°)$
  3. Scaling the image by a random factor chosen as follows: a number a is drawn uniformly from $[-1.0,\ 1.0)$, the image is then scaled by factor 10^a.
  4. Flipping the image on the vertical edge (left-right).
  5. Flipping the image on the horizontal edge (up-down).

  If necessary, the image is padded with the per-channel mean of the original image, which emulates the background color.

  The probability of each transformation is a function of the hyper-parameter p_t, the probability that at least one transformation is applied on a training sample. The relationship between the probability of an individual transformation p_i and the overall transformation probability p_t is given in Equation 3-11.

  $$p_t = p_1 + (1 - p_1)p_2 + (1 - p_1)(1 - p_2)p_3 + ... \tag{3-11}$$

  By setting $p = p_1 = p_2 = p_3 = ...$, and restricting the augmentation policy to the five transformations above, Equation 3-11 becomes:

  $$p_t = \sum_{i=1}^{5}(1 - p)^{i-1}p \tag{3-12}$$

  When setting p_t as a hyper-parameter, p is obtained by solving the polynomial in Equation 3-12. See Table 3-2 for example values.

  When combining data augmentation with re-sampling, the sampling is applied *before* the transformations, to ensure that transformed examples are not duplicated and variation is introduced in the minority classes.

  Table 3-2: Probability p, that an individual transformation is applied to a training example as a function of overall probability p_t, that at least one transformation is applied.

Overall p_t	0.1	0.25	0.5	0.75	0.9
Individual p	0.021	0.056	0.129	0.242	0.369

- **Dropout Regularization**: Dropout regularization is applied after the global pooling layer and before the fully-connected classification layer with p_{drop} as hyper-parameter.

- L_2 **Weight Regularization**: Weight regularization is implemented by adding $\lambda|\theta|_2^2$ to the loss function, where $|\theta|_2$ is the L_2-norm of the model weights and biases and λ is a variable hyper-parameter.

- **Early Stopping**: All runs are terminated after no improvement in sample weighted validation loss is recorded for 10 epochs.

Chapter 4

Experiments

In this chapter the experiments and the respective results are presented. The goal of the experiments is to confirm or disprove the hypotheses set up in the introduction. The secondary goal is to investigate the influence of certain network design choices, training configurations and hyper-parameters on the performance of the Convolutional Neural Networks (CNNs) in our application.

The chapter starts with the general set-up of all experiments in Section 4-1, showing the general training configuration and metrics used during training and evaluation. Section 4-2 describes the experiments done to evaluate different CNN models as backbone feature extractor for the classifier. Section 4-3 describes the experiments for the design of a classification head on top of the feature extractor.

This includes the type of global pooling method, order of pooling and calculating the classification score and choice of loss and output activation functions.

Section 4-4 shows the experiments and results on data re-sampling and Section 4-5 describes the experiments and results for the regularization techniques of data augmentation, dropout, and L_2 regularization. Section 4-6 describes the experiments where the findings of the previous experiments are combined.

This chapter follows a descriptive approach and does not provide an interpretation or explanation of the experimental results, which will be given in Chapter 5.

Detailed tables with all the results and additional experimental results not deemed relevant for the main matter can be found in Appendix C.

4-1 Set-Up

4-1-1 Training Hyper-Parameters

The following list is an overview of the computation environment and the training and optimizer parameters that are set for each training run.

- **Batch size**: 24, which is the maximum allowable for the available hardware and the default input resolutions.
- **Termination criterion**: All runs are terminated after no improvement in sample-weighted validation loss is recorded for 10 epochs.
- **Optimizer**: Adam with values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$ from [76] and the TensorFlow default for $\epsilon = 10^{-7}$.
- **Learning rate**: Obtained by performing a learning rate range test [79] for one epoch. Triangular cyclical learning rate schedule described in [79], where one cycle corresponds to four epochs, without further decay. See Appendix B for implementation details.
- **Weight initialization**: All backbone weights are initialized with weights pre-trained on ImageNet. The customized fully-connected (FC)-layers are initialized with the default `tf.keras` Glorot-uniform initialization.
- **Input processing**: Cropping to the original input shape of each backbone (see Table 3-1), centering and scaling according to the original backbone implementation.
- **Hardware**: Virtual machine on Google Cloud Platform with two Tesla P100 GPUs with 16GB memory each.
- **Framework & version**: TensorFlow v1.13.1 and v1.14.0 with `tf.keras` API.

4-1-2 Calculation of the Damage Output Score

The output of the neural network is a probability for each class. There is no separate classification for ‘damage’/‘no damage’. Instead, the output probability for damage is calculated from the class probabilities:

For single-label classification, the predicted damage probability is the **sum** of the probabilities for all classes in the damage category.

For multi-label classification, the predicted damage probability is the **max** of the probabilities for all classes in the damage category. The classification threshold is 0.5, and is not fine-tuned on the validation set to further increase sensitivity.

4-1-3 Evaluation and Testing Metrics

The following set of metrics is used for evaluation (i.e. model selection, monitoring the training process) and for final testing and comparing the different models and training configurations.

- **Validation loss**: For monitoring the termination criterion, and choosing the best checkpoint, the *sample-weighted* validation loss is used. The sample weight is proportional to the inverse of the frequency of the class corresponding to the sample.
- **Accuracy**: Fraction of correct predictions. A prediction is defined as the $\arg\max$. This is also the case for the multi-label approach, to be able to compare single-and multi-label classification. This is a standard metric for reference.
- **Balanced accuracy**: Accuracy adjusted for imbalanced data, punishes ‘guessing’ towards the majority class. Equivalent to the average recall on all classes.

- **Recall on damages**: Fraction of true damages that are predicted correctly as damages, main performance metric. It is a less strict metric than looking at the average of recall on all damage classes, because it does not punish the classification of one damage class instance into a different damage class.

For all plots, **error** (1 − accuracy), **balanced error** (1 − balanced accuracy) and **False Negative Rate (FNR) on damages** (1 − recall) are used throughout this and future reports to better illustrate scale.

4-2 Backbone Feature Extractor

In this Section, the experiments that will test the hypotheses around finding a suitable backbone feature extractor for the CNN-classifier are described.

4-2-1 Freezing the Convolutional Backbone

The first set of experiments tests the hypothesis that a complete network has to be re-trained for this task and merely training the top-layer is not sufficient to get high performance (Hyp.A).

In the first run, the ResNet50 CNN [14], a popular candidate for transfer learning, is frozen and only the top FC-classification layer is trained. In the second run, the top FC-classification together with the last batch-normalization and convolutional layer is re-trained (three layers in total). In the third run, all layers from the top up to and including *conv5-block3* are re-trained. In the final run of this experiments, the complete network is re-trained after being initialized on ImageNet weights.

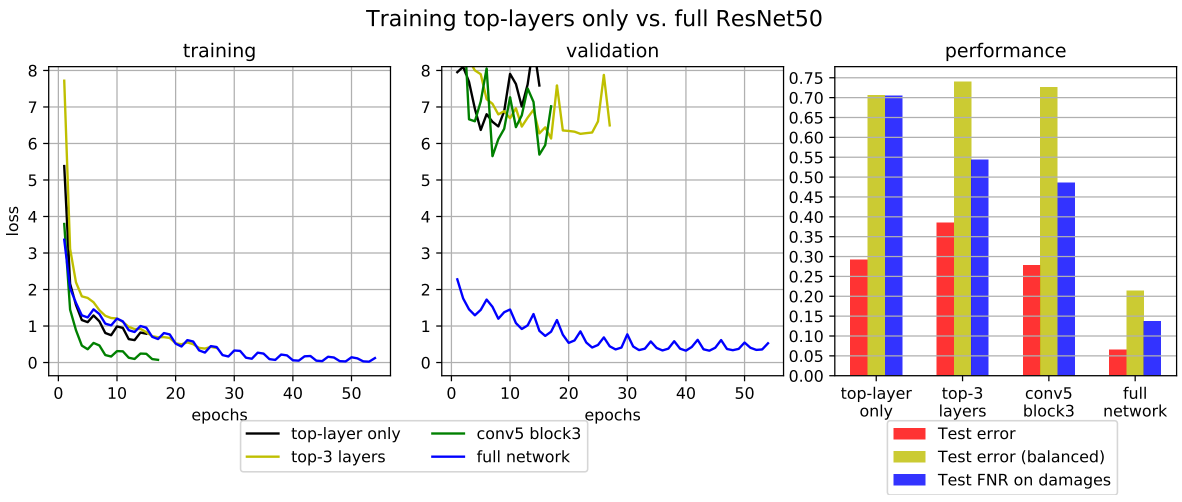


Figure 4-1: Results for experiments with different numbers of layers of ResNet50 unfrozen for training. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints. All four runs are trained and tested on DatasetV1 (single-label).

Figure 4-1 shows the results for the training runs with different number of trainable layers. Note that the loss shows a cyclical behavior. This is caused by the cyclical learning rate schedule (Appendix B).

The loss plots show that with frozen convolutional weights, the training loss can be low, but the validation loss is high and unstable. Only when training the full network, the validation loss follows the training loss until it plateaus. This comes at the cost of longer training times and increased computational demand.

Freezing all layers except the top-layer gives reasonable results in terms of error (29.2%). However, the balanced error and FNR values are high at both 70.6%. When training the top-3 layers, error and balanced error increase to 38.5% and 74.1%, respectively, while FNR goes down to 54.5%.

Further increasing the number of trainable layers to include all layers up to and including *conv5-block3* of the ResNet50 model, leads to minor improvements in the metrics. When finally training the full network, the error drops to 6.7%, the balanced error to 21.4% and the FNR on damages to 13.7%. This is approximately a factor five improvement from training only the top-layer, implying that Hyp.A can be confirmed with a large degree of certainty.

4-2-2 Choosing a CNN Architecture

To improve on the baseline set by ResNet50, the search is extended to other pre-trained CNN architectures to answer the question of which model architecture has the highest performance in terms of FNR?

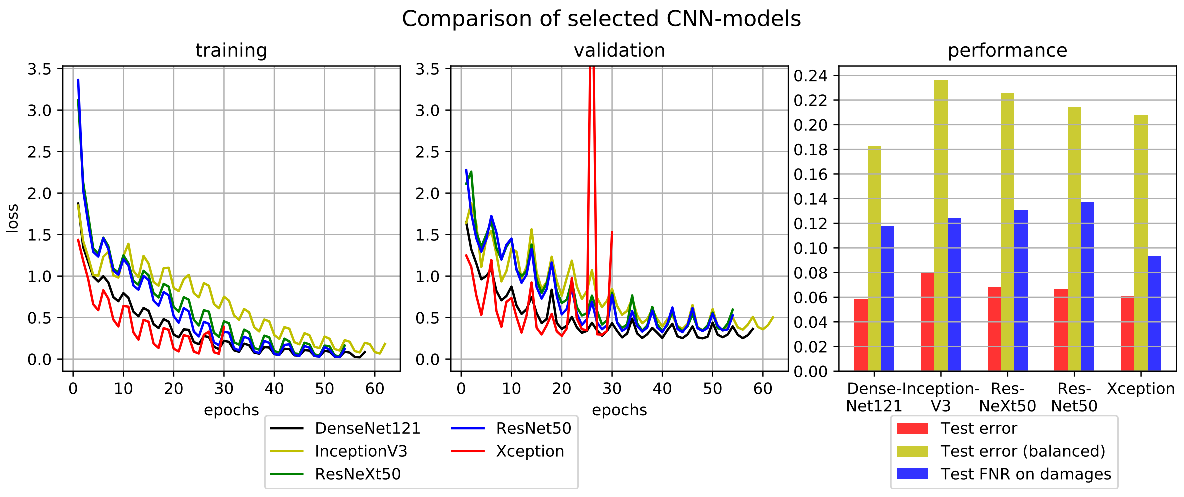


Figure 4-2: Five selected CNN architectures trained on DatasetV1 (single-label), initialized with ImageNet pre-trained weights. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints. No regularization and re-sampling are applied, and softmax activation is used for classification.

In Section 3-2, DenseNet121 [20], InceptionV3 [17], ResNet50 [14], ResNeXt50 [19] and Xception [21] are identified as candidates. Each of the five models is trained on our dataset once until convergence. Max-pooling followed by the linear classification layer and softmax activation with categorical cross-entropy make up the classification head. No regularization and no re-sampling are applied to get the bare-bones performance of each model.

Figure 4-2 shows the basic results of the training runs. DenseNet achieves the lowest validation loss, but Xception shows the fastest convergence; its training is terminated after 30 epochs while the other models take 54-62 epochs until they reach the termination criterion. ResNet50 and ResNeXt50 show similar behavior. The InceptionV3 model converges slowly compared to the other networks.

In terms if final metrics, DenseNet121 reaches the highest test accuracy and balanced test accuracy, but the Xception network scores lowest on FNR. The FNR is improved from 13.7% with ResNet50 to 9.4% with the Xception network. Due to its low FNR and fast training behavior, it is used for all other experiments.

4-3 Classification Head

This section describes the experiments testing the hypotheses around designing the classification head.

4-3-1 Global Pooling Layer

It is hypothesized, that global max-pooling is beneficial for small objects, and combining max- and average-pooling by concatenation can give further performance increases (Hyp.B).

Three training runs are performed for this experiment, the first one using average-pooling, the second using using the combination of max- and average-pooling and the third using max-pooling on top of the last feature map of the Xception backbone CNN.

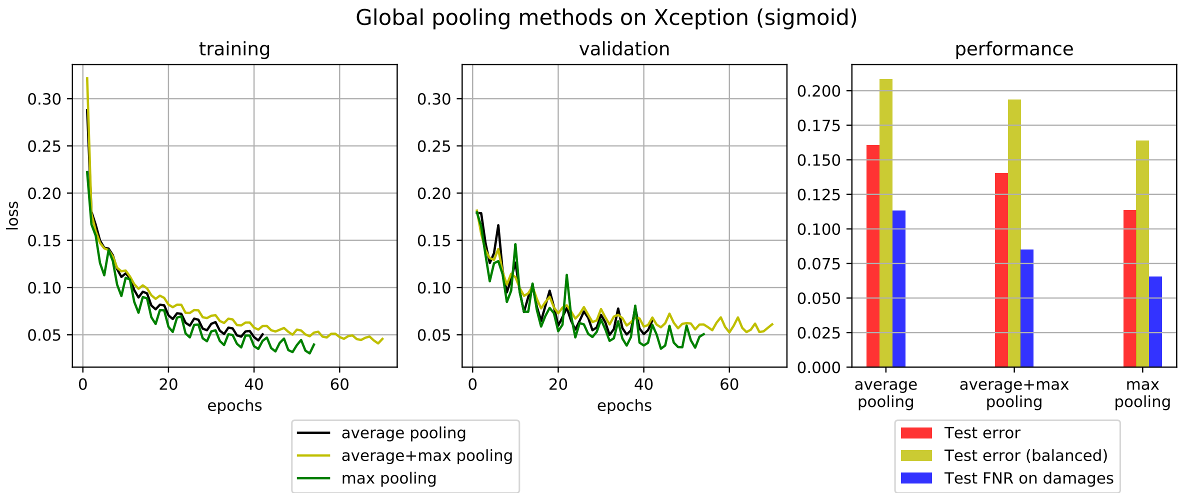


Figure 4-3: Training with different methods of global pooling on the last convolutional feature map of Xception with sigmoid activation. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints. The runs are trained and tested on DatasetV2. The training set is re-sampled and augmented with $p_t = 0.5$.

Figure 4-3 shows the results for the pooling-layer experiments: For all three metrics, max-pooling has the best performance with 11.4% error, 16.4% balanced error and 6.5% FNR. The combination of max- and average-pooling does not increase the performance of max-pooling alone and achieves 14.0% error, 19.4% balanced error and 8.5% FNR. Average-pooling has the worst performance with 16.1% error, 20.8% balanced error and 11.3% FNR. This confirms Hyp.B.

4-3-2 Final Activation Function - Single-label vs. Multi-label Classification

The hypothesis to test is that using multiple positive predictions, i.e. a multi-label approach in one image, increases the sensitivity (recall) of the classifier (Hyp.D).

Two training runs are conducted as a baseline for the classification head, one using softmax and one using sigmoid activation with FC-classification. Figure 4-4 shows that the results.

Note how for both softmax and sigmoid activation, the validation loss is greater than the training loss and plateaus, while the training loss keeps decreasing towards zero.

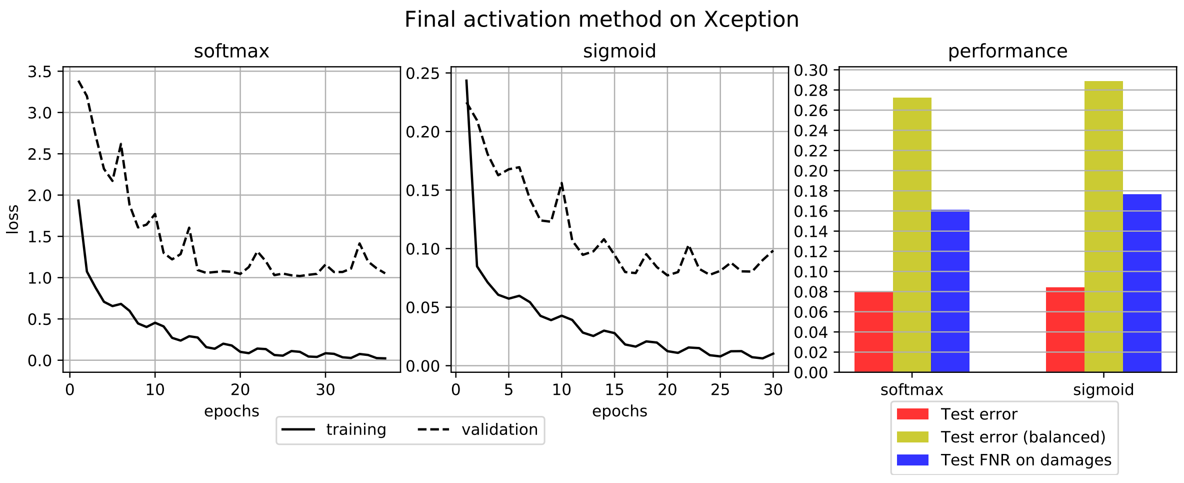


Figure 4-4: Effect of final activation and loss function. Left: categorical cross-entropy loss for softmax activation. Middle: binary cross-entropy loss for sigmoid activation. Right: performance metrics on the best checkpoints for both softmax and sigmoid activation. The runs are trained and tested on DatasetV2, without re-sampling or augmentation enabled.

The metrics for single-label (softmax) classification are lower than the ones for multi-label (sigmoid) classification, e.g. for FNR, the softmax activation with 16.1% FNR performs slightly better than the sigmoid activation with 17.6% FNR.

This does not Hyp.D, that the multi-label configurations produce higher recall. However, multi-label classification is not immediately rejected, because the difference is not that significant. All the following experiments will also consider multi-label classification to investigate, if there exists a configuration, that can bring the performance of multi-label classification to the same level as single-label classification.

4-3-3 Order of Classification and Pooling - Fully-connected Classification vs. Convolutional Classification

It is hypothesized, that convolutional classification results in a lower FNR on the damages due to its emphasis on local features (Hyp.C).

To test this, the runs in Figure 4-4 are repeated with a convolutional layer to calculate the class logit scores. The results are shown in Figure 4-5. The convolutional classification method does not decrease the difference between the training and validation losses, so it does not have a regularizing effect.

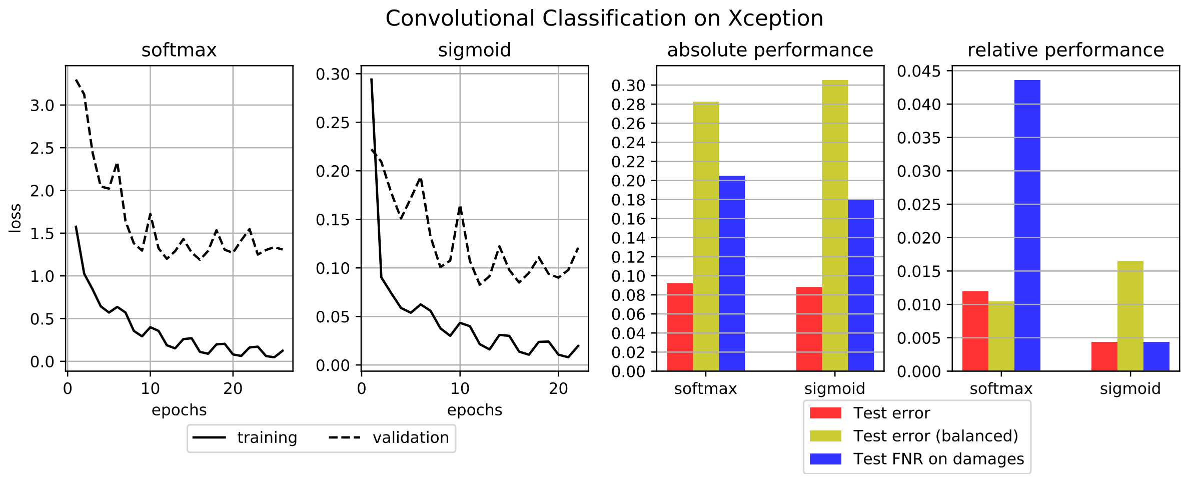


Figure 4-5: Effect of classification method (order of pooling and class logit layer). The two plots on the left show the cross-entropy losses for softmax and sigmoid classification after each epoch, respectively. The two plots on the right show the absolute metrics and metrics relative difference to the baseline experiments in Figure 4-4. The runs are trained and tested on DatasetV2, without re-sampling or augmentation enabled.

For both softmax and sigmoid activation the metrics are higher than when using FC classification. The softmax FNR climbs from 16.1% to 20.5%. For sigmoid activated classification, the FNR increases from 17.6% to 18.1% when using convolutional classification. Hyp.C, that the convolutional classification as implemented here, lowers the FNR, therefore has to be rejected.

4-4 Re-sampling

To test if re-sampling the training data to a uniform distribution decreases the FNR on damages (Hyp.E), and what the effect is on the different classification types, the experiments in Figure 4-4 are repeated with re-sampling enabled. The results are shown in Figure 4-6. In the loss plots, it can be seen that re-sampling increases the variance of the softmax validation loss but decreases the variance of the sigmoid loss.[1]

[1]Variance in this context is the difference between the peaks and bottoms in the loss caused by the cyclical learning rate schedule.

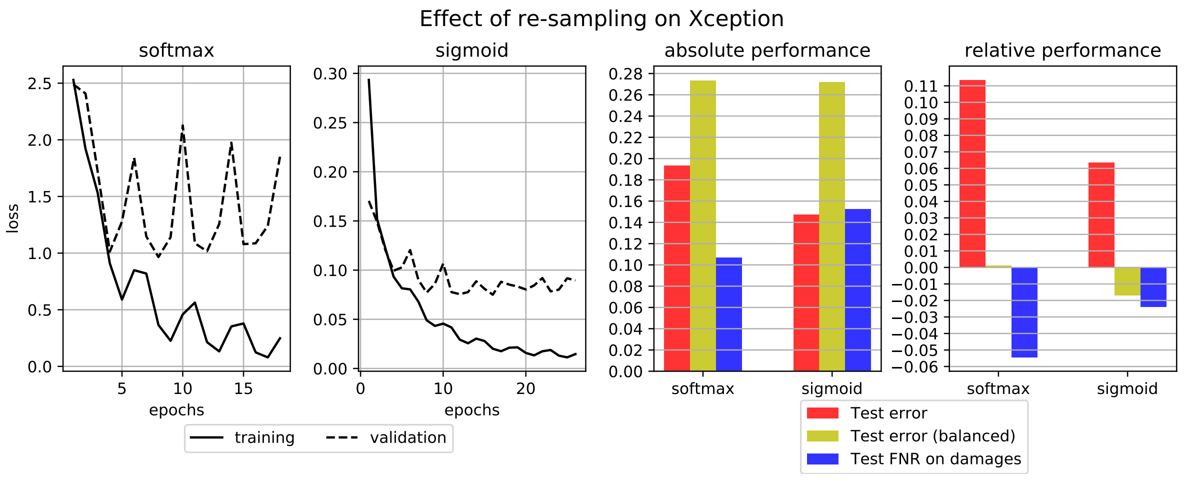


Figure 4-6: Effect of re-sampling on the two activation functions. The two plots on the left show the cross-entropy losses for softmax and sigmoid activation after each epoch, respectively. The two plots on the right show the absolute metrics and metrics relative difference to the baseline experiments in Figure 4-4. The runs are trained and tested on DatasetV2, without augmentation enabled.

For softmax activation, re-sampling increases the test error from 8.0% to 19.4%, slightly increases the balanced error from 27.2% to 27.3% and decreases the FNR from 16.1% to 10.7%. For sigmoid activation, the test error again increases from 8.4% to 14.7%. The balanced error also sees only a minor change from 28.9% to 27.2%. The FNR is lowered from 17.6% to 15.3%. After training with re-sampling, softmax activation still has lower metrics than sigmoid activation.

Re-sampling benefits softmax activation more than sigmoid activation in terms of FNR, it increases the gap between softmax FNR and sigmoid FNR. The increase in error due to re-sampling is expected, as mis-classifications on the majority class have less impact on the optimization with re-sampling enabled.

Although this confirms Hyp.E, the inconsistent, barely positive effect on the balanced accuracy (=average recall) is unexpected, as re-sampling should increase the recall on all under-represented classes. This could be an indicator for the necessity for data augmentation when over-sampling under-represented data (Hyp.F).

4-5 Regularization

This section describes the experiments to investigate on one hand, the influence of regularization techniques on the classification performance, and on the other hand, to find optimal regularization parameters for each technique, respectively.

Due to the small data set size, the hypothesis is that extensive use of regularization techniques (augmentation, dropout and L_2 regularization) increases performance in any case (Hyp.G).

4-5-1 Data Augmentation

For data augmentation in particular, the hypothesis is that it is necessary to make re-sampling work properly (Hyp.F). To test this, augmentation is applied with different transformation probabilities p_t to a re-sampled training dataset. In the training runs, p_t ranges from 0.0 (no transformation/augmentation) to 0.9 (90% of the training samples are transformed in some way).

Single-label Classification (Softmax)

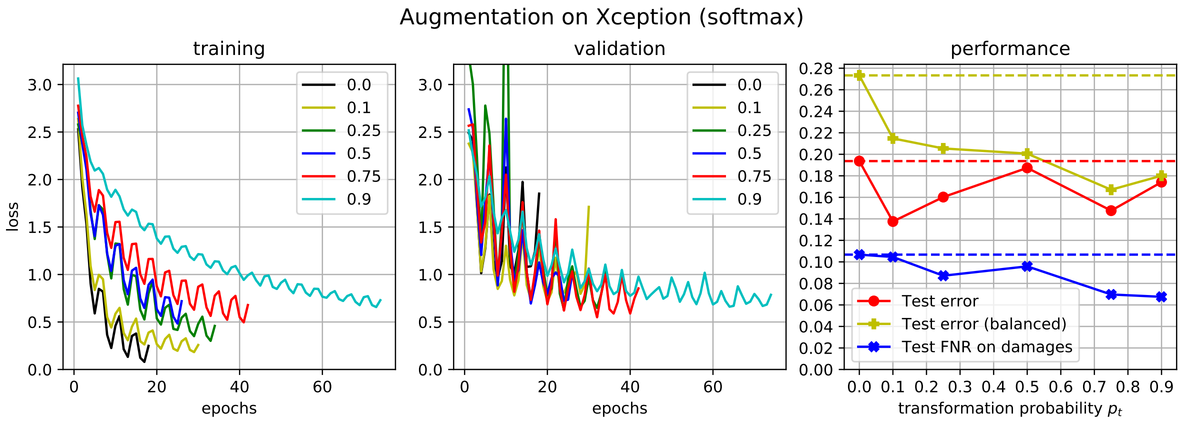


Figure 4-7: Training with varying transformation rates on Xception with softmax activation on DatasetV2. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints as a function of p_t. The dashed lines denote the baseline without augmentation. The training set is re-sampled to a uniform distribution.

Figure 4-7 shows the results for training Xception with FC single-label classification head. Increasing p_t increases the training loss and slows down convergence, leading to longer training times. The effect on validation loss is not obvious, but the validation loss sees more extreme spikes when compared to the run without augmentation. It can be seen that augmentation increases the number of epochs until the termination criterion is triggered.

For some values of p_t, the error decreases. The balanced error and the FNR see more consistent improvements with higher p_t. At $p_t = 0.9$, the FNR is at 6.8%, down from 10.7% without augmentation, confirming the augmentation component of Hyp.G for softmax activation.

Multi-label Classification (Sigmoid)

Figure 4-8 shows the results for training Xception with a multi-label head and data augmentation enabled. Just as with softmax activation, adding augmentation increases the training loss and slows down convergence. The validation loss does not show the large spikes that are observable in the softmax validation loss. The lowest validation loss is clearly at $p_t = 0.5$.

There is an increase in error and balanced error with approximately constant FNR until $p_t = 0.25$ but a sharp drop for all three metrics at $p_t = 0.5$. The FNR at $p_t = 0.5$ is down to 6.5% from 15.3% without augmentation. This also confirms the augmentation component of Hyp.G for sigmoid activation.

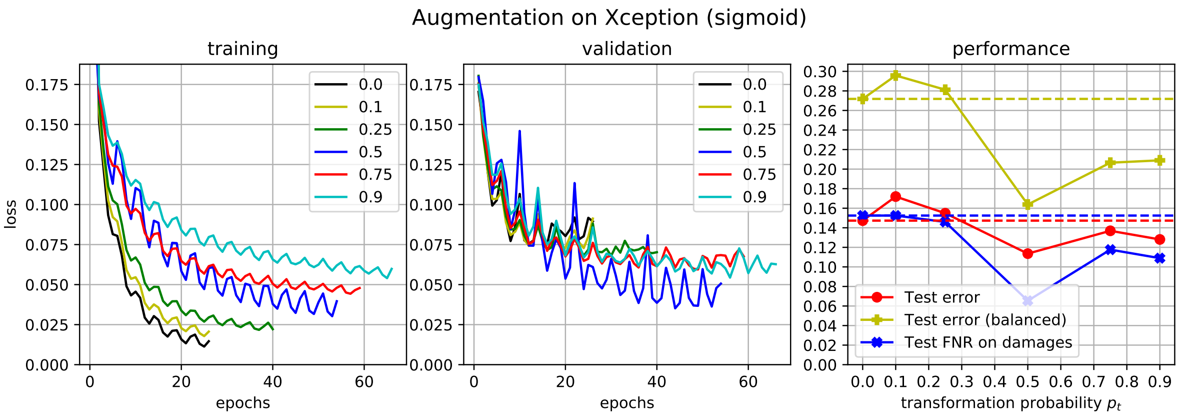


Figure 4-8: Training with varying transformation rates on Xception with sigmoid activation on DatasetV2. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints as a function of p_t. The dashed lines denote the baseline without augmentation. The training set is re-sampled to a uniform distribution.

Combining Augmentation with Re-sampling

The effect of combining re-sampling and augmentation on the configurations as presented in Figure 4-4 is visualized in Figure 4-9. For both activation functions, the validation loss is lower (softmax) or equal to the training loss (sigmoid), but more noisy.

The FNR improves across both configurations, the stronger improvement is found for the sigmoid configuration. With combined re-sampling and augmentation, the FNR for sigmoid activation is now on par with the softmax configuration.

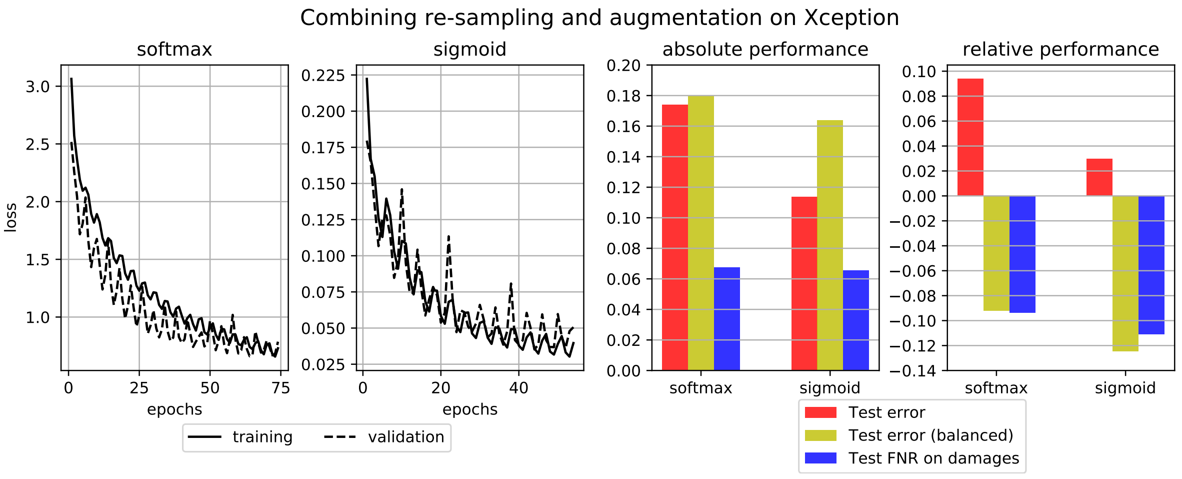


Figure 4-9: Effect of combining re-sampling *and* augmentation on the two activation functions. The two plots on the left show the cross-entropy losses for softmax and sigmoid activation after each epoch, respectively. The two plots on the right show the absolute metrics and metrics relative difference to the baseline experiments in Figure 4-4. The runs are trained and tested on DatasetV2 and re-sampling enabled. $p_t = 0.9$ for the softmax configuration and $p_t = 0.5$ for the sigmoid configuration.

This set of experiments confirms Hyp.F, that augmentation is necessary for re-sampling to work properly, as additional performance gains can be observed when augmentation is applied.

4-5-2 Dropout

To test the influence of dropout regularization, two experiments are performed with different dropout rates in each run, one for the FC-single-label case, one for the FC-multi-label case. Five runs are done with the dropout rate set to 0.1, 0.25, 0.5, 0.75 and 0.9, respectively.

Single-label Classification (Softmax)

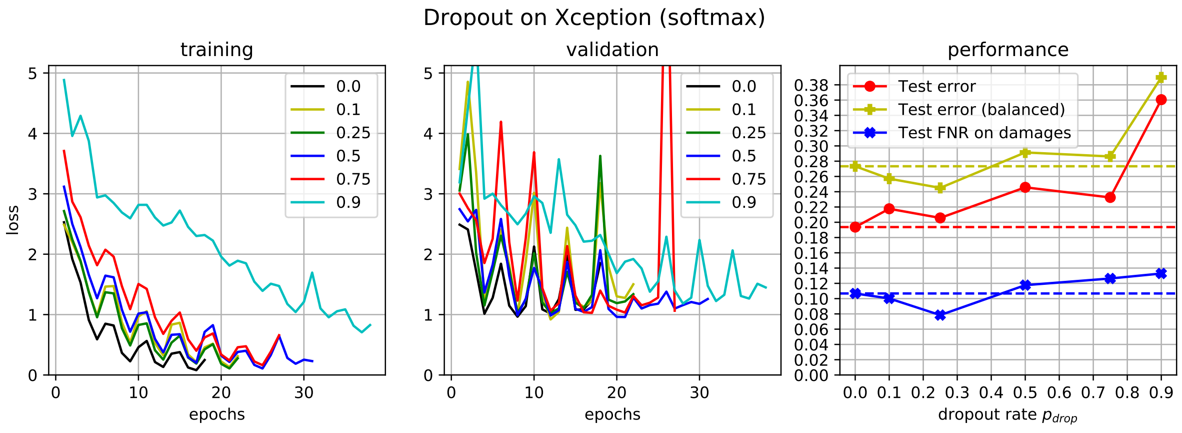


Figure 4-10: Training with varying dropout rates on Xception with softmax activation on DatasetV2. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints as a function of p_{drop}. The dashed lines denote the baseline without augmentation. The training set is re-sampled to a uniform distribution.

Figure 4-10 shows the results of using dropout on the single-label configuration with softmax. As expected, dropout increases the training loss and slows down convergence. There does not seem to be a decrease in the lowest validation loss, but dropout amplifies the spikes in the validation loss curves that are caused by the cyclical learning rate.

Error and balanced error increase with higher dropout rate, but the FNR improves until $p_{\text{drop}} = 0.25$ and increases again from $p_{\text{drop}} = 0.5$. At the minimum of $p_{\text{drop}} = 0.25$ the FNR amounts to 7.8%, down from 10.7% without dropout. For single-label classification, a moderate dropout rate seems to be beneficial for the testing FNR on the damages, but not for the balanced error. This confirms the dropout component of Hyp.G for softmax activation.

Multi-label Classification (Sigmoid)

The results of using dropout on the multi-label configuration are presented in Figure 4-11. The training loss increases with higher dropout rate and the loss converges slower. The validation loss is less noisy and doesn't show the spikes that can be seen in the single-label validation loss curves in Figure 4-10. The validation loss does not show any clear improvements with respect to the configuration when no dropout is applied.

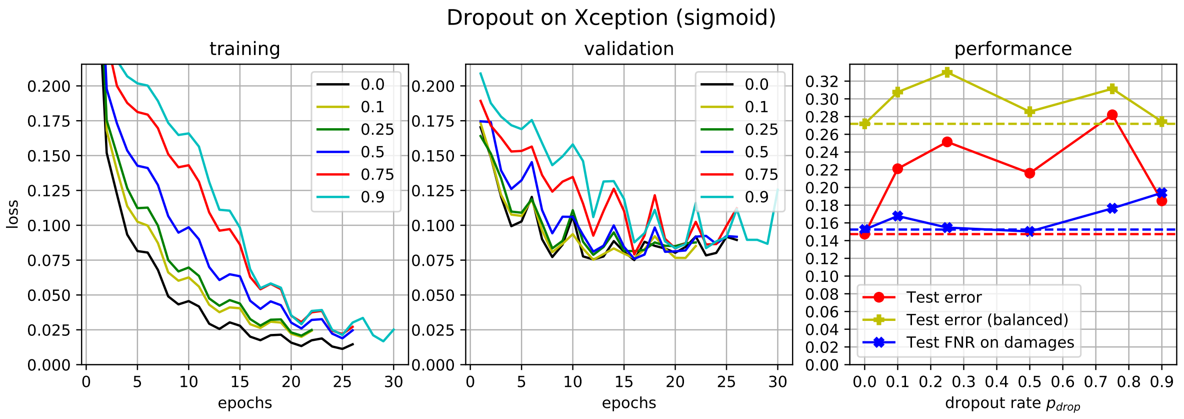


Figure 4-11: Training with varying dropout rates on Xception with sigmoid activation on DatasetV2. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints as a function of p_{drop}. The dashed lines denote the baseline without augmentation. The training set is re-sampled to a uniform distribution.

Error and balanced error increase when dropout is applied, but there is no clear relationship observable between dropout rate and the decrease in performance. The influence on the FNR is similar, but less pronounced. Only at a dropout rate of 0.5, there is an improvement in FNR from 15.3% to 15.0%. This improvement is too small to conclude it did not happen by chance. Therefore, the experiment shows that for the sigmoid case, the dropout component of Hyp.G can not be confirmed.

4-5-3 L_2 Weight Regularization

For weight regularization, similar experimental set-ups are used as before. For both single-label and multi-label configurations, a series of networks are trained with varying parameter λ for the $L2$ regularization cost term. For this experiment, a logarithmic range of values for λ from 10^{-7} to 10^{-3} is chosen, which is a range around the $\lambda = 10^{-5}$ used in the original Xception implementation [21].

Single-label Classification (Softmax)

Figure 4-12 shows the results of using L_2-regularization on the single-label Xception model. The training loss is higher, which is partly due to the added $\lambda|\theta|_2^2$-term in the training loss. The validation loss shows spikes, that seem to stabilize with $\lambda = 10^{-3}$. The lowest validation loss is observed for $\lambda = 10^{-6}$ at 8 epochs. Note that for $\lambda = 10^{-5}$, the best validation loss already occurs after 4 epochs, leading to termination after already 14 epochs. The consequences of this are further discussed in Section 5-8.

The metrics for $\lambda = 10^{-7}$ start above their baseline values for the training run without regularization. Without more experiments, it cannot be concluded if that is a random occurrence, or if a range of lower λ values provides better results.

For the available runs, the error and balanced error see an increase when L_2-regularization is applied until $\lambda = 10^{-5}$, where they decrease again, but do not reach the level when no regularization is applied. The FNR however, decreases with increasing λ and sees the minimum

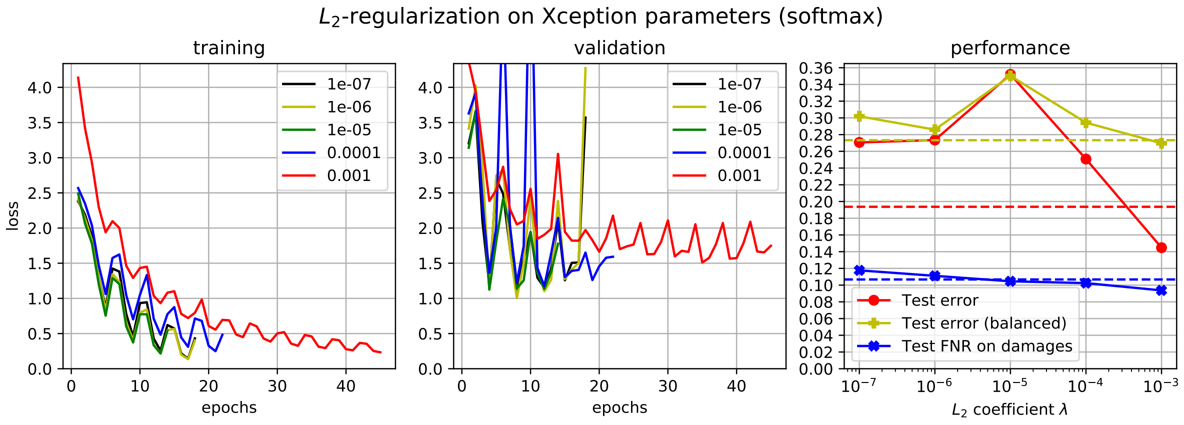


Figure 4-12: Training with varying strengths of L_2 regularization on all Xception model parameters (weights and biases) with softmax activation on DatasetV2. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints as a function of λ. The dashed lines denote the baseline without augmentation. The training set is re-sampled to a uniform distribution.

value at $\lambda = 10^{-3}$ of 9.4%, down from 10.7%. Increasing λ beyond 10^{-3} leads to unstable training loss with no convergence and is therefore not shown in this figure. There is a weak but consistently positive trend visible for FNR, confirming the L_2 component of Hyp.G for softmax activation.

Multi-label Classification (Sigmoid)

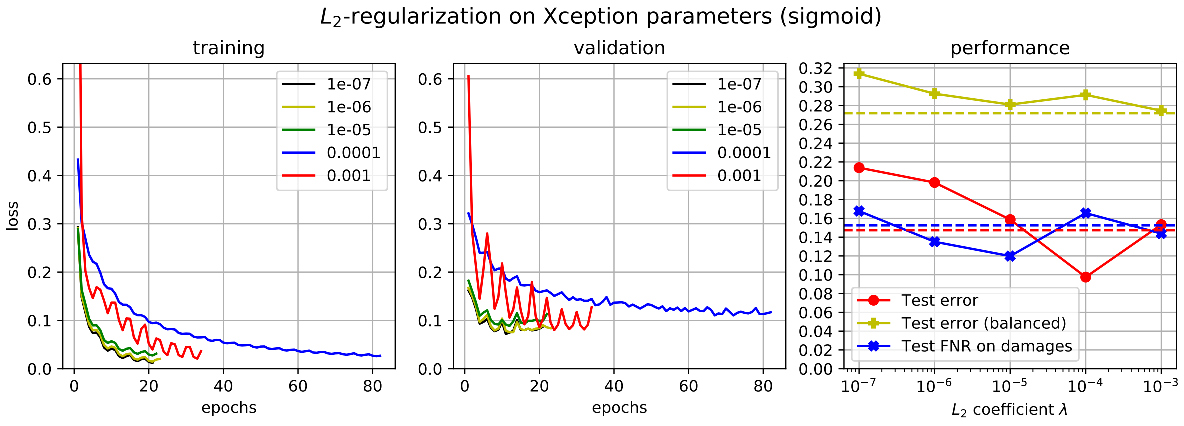


Figure 4-13: Training with varying strengths of L_2 regularization on all Xception model parameters (weights and biases) with sigmoid activation on DatasetV2. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints as a function of λ. The dashed lines denote the baseline without augmentation. The training set is re-sampled to a uniform distribution.

Figure 4-13 shows the results for the experiment on L_2 regularization for the multi-label Xception model. The training loss increases and converges slower, consistent with expectations and the other experiments. The validation loss does not seem to improve.

Interestingly, at $\lambda = 10^{-4}$, the training loss and validation losses become very smooth and training proceeds for more than twice as long than for the other runs. This run also shows the best overall error, which is not the metric of interest, and therefore without effect.

The metrics decrease until $\lambda = 10^{-5}$, where balanced error and FNR start to increase again. Error and balanced error still see a minor increase with respect to the baseline, but the FNR decreases to 12.0% from 15.3%, confirming the L_2 component of Hyp.G for sigmoid activation.

4-6 Combining the Regularization Techniques

In the final set of experiments, the findings of the previous experiments are naively combined. The regularization hyper-parameters that lead to the lowest FNR are selected. This Section has two goals:

First, checking for each activation function, if combining the regularization methods with the parameters found in Section 4-5, leads to an improvement with respect to the cases where one regularization methods is applied in isolation (Hyp.H).

Second, testing the hypothesis that an optimized sigmoid activated network with multi-label classification has better FNR than a softmax activated network. The training runs are repeated several times with different random seeds, to get a statistically significant estimate of which network actually performs better.

For the repeated experiments, the means of the metrics are taken as the reference values. To visualize the uncertainty, the 95% confidence intervals around the means are displayed as well. The standard deviation is not shown, as it does not take into account the small sample size.

The approach taken to to test if combined regularization provides any gains above isolated regularization, is as follows: For the isolated regularization runs, there is only one training run available, respectively. Therefore, it is not possible to perform a t-test with two samples. Instead, it is checked for the relevant metric, if the result of the isolated regularization model is above the upper bound of the confidence interval around the mean of the combined runs.

Single-label Classification (Softmax)

The softmax network uses $p_t = 0.9$, $\lambda = 10^{-3}$ and $p_{\text{drop}} = 0.25$, as found in Section 4-5. The experiment uses re-sampled training data from DatasetV2. The results for the set of four repeated runs are shown in Figure 4-14.

It can be seen, that combining the regularization methods stabilizes the validation loss, there are no large spikes. Training proceeds for a long time with slow convergence (98-159 epochs), due to the large values for p_t and λ.

The different random seeds lead to different realizations of the random mini-batches, transformations and dropped neurons. The resulting variation in training and validation loss can be observed accordingly. The detailed results for each run can be found in Appendix C, Table C-16.

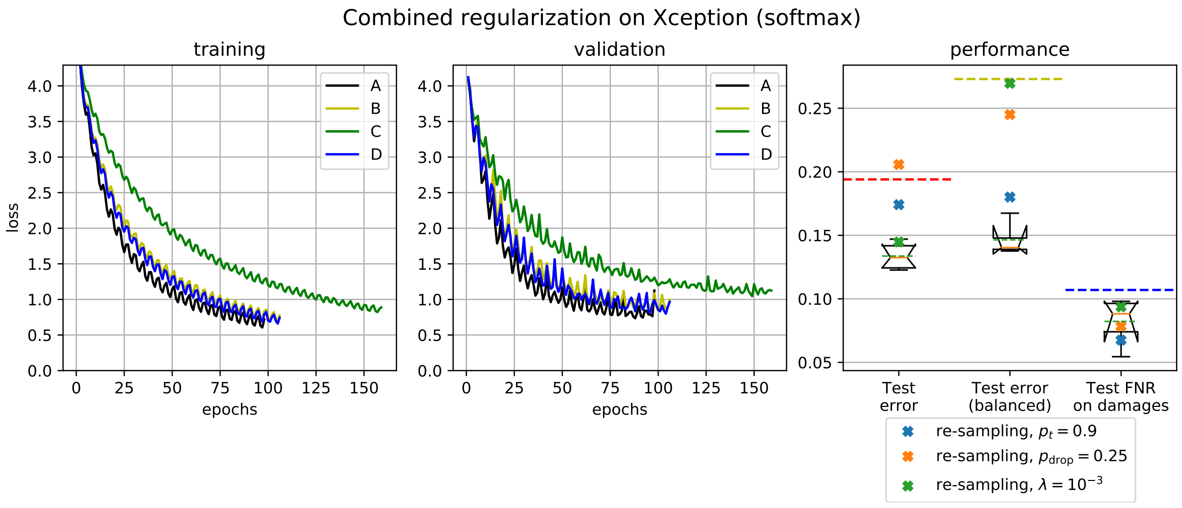


Figure 4-14: Training with combined regularization with softmax activation on DatasetV2. $p_t = 0.9$, $\lambda = 10^{-3}$ and $p_{\text{drop}} = 0.25$. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints as box-plots and compared to runs with isolated regularization. The dashed lines denote the baseline without augmentation. The boxes range from the 1st quartile to the 3rd quartile of the sample. The median of the respective metric across all runs is the orange solid line, the mean the green dashed line. The 95% confidence interval is marked by the notch in the box, which can extend beyond the box itself, due to the small sample size. The whiskers extend to the full range of the sample.

Table 4-1: Comparing isolated regularization with combined regularization (sigmoid). For the isolated regularization, the runs with the best FNRs are chosen. For the combined regularization the mean and 95% confidence interval on the mean is displayed.

Name	**Test Error**	**Balanced Test Error**	**Test FNR**
re-sampling only	0.194	0.273	0.107
re-sampling & $p_t = 0.9$	0.174	0.180	0.068
re-sampling & $p_{\text{drop}} = 0.25$	0.206	0.245	0.078
re-sampling & $\lambda = 10^{-3}$	0.145	0.270	0.094
re-sampling & all combined	0.134 ± 0.009	0.146 ± 0.011	0.082 ± 0.016

The balanced error for the combined runs is significantly better than the balanced errors for the runs with isolated regularization: The mean and confidence interval are lower than all other runs.

The mean FNR of the repeated runs is 8.2%, but the best run (D) achieves a FNR of 5.4%. It can be observed, that the other data points with isolated regularization fall within the 95% confidence interval of the repeated runs with combined regularization. Therefore, combined regularization does not yield an improvement *on average*, which rejects Hyp.H for softmax activation.

This also implies that variation in the FNR can not be attributed to the regularization methods with certainty, but are possibly purely caused by chance.

However, we can still use Run D for further use, as this is the best result in terms of FNR achieved in this work.

Multi-label Classification (Sigmoid)

The sigmoid configuration uses $p_t = 0.5$, $\lambda = 10^{-5}$ and $p_{\text{drop}} = 0.0$, as found in Section 4-5. The experiment uses re-sampled training data from DatasetV2. The results for the sigmoid configuration are shown in Figure 4-15, with details in Appendix C, Table C-17.

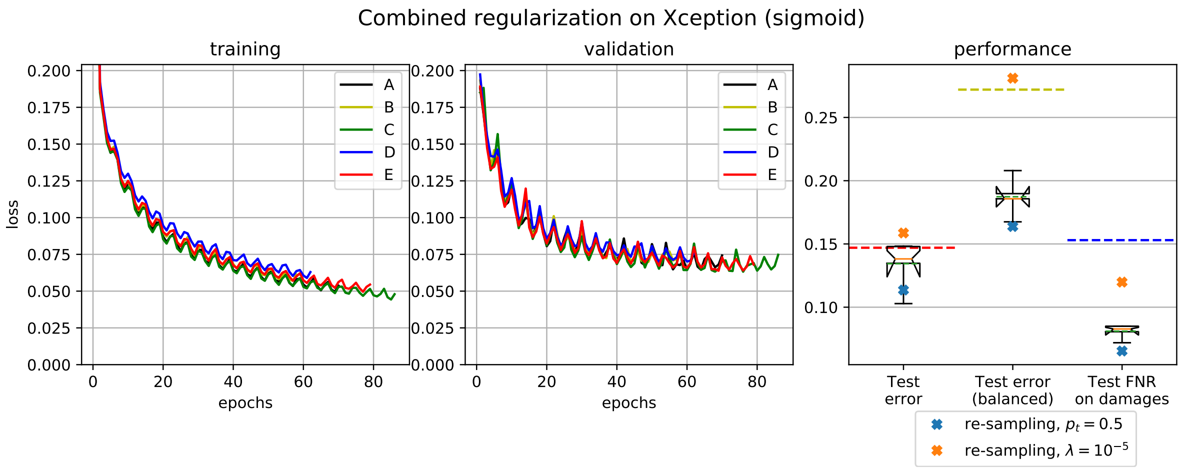


Figure 4-15: Training with combined regularization with sigmoid activation on DatasetV2. $p_t = 0.5$, $\lambda = 10^{-5}$ and $p_{\text{drop}} = 0.0$. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints as box-plots and compared to runs with isolated regularization. The dashed lines denote the baseline without augmentation. The boxes range from the 1st quartile to the 3rd quartile of the sample. The median of the respective metric across all runs is the orange solid line, the mean the green dashed line. The 95% confidence interval is marked by the notch in the box, which can extend beyond the box itself, due to the small sample size. The whiskers extend to the full range of the sample.

Training proceeds for 58 to 86 epochs, shorter than the softmax runs, due to the smaller values of p_t and λ. There is less variation in the loss magnitude for both training and validation loss, which can be attributed to the lack of a dropout layer, and the lower p_t value.

Table 4-2: Comparing isolated regularization with combined regularization (sigmoid). For the isolated regularization, the runs with the best FNRs are chosen. For the combined regularization the mean and 95% confidence interval on the mean is displayed.

Name	**Test Error**	**Balanced Test Error**	**Test FNR**
re-sampling only	0.147	0.272	0.153
re-sampling & $p_t = 0.5$	0.114	0.164	0.065
re-sampling & $\lambda = 10^{-5}$	0.159	0.281	0.120
re-sampling & all combined	0.134 ± 0.010	0.187 ± 0.008	0.081 ± 0.003

When comparing combined regularization to the individual regularization runs, a significant improvement is only observable for the run with only L_2-regularization applied with $\lambda = 10^{-5}$. The run with $p_t = 0.5$ has lower balanced error and FNR than the run with combined regularization and it is also outside the lower bound of the confidence interval.

Therefore, for sigmoid-activated multi-label classification, combining the regularization methods naively, does not yield any significant gains, especially when compared to applying data augmentation alone, which rejects Hyp.H for sigmoid activation as well.

Single-label vs. Multi-label Classification

It can be seen in Table 4-3 that the mean FNR for the sigmoid configuration with combined regularization is lower than for the softmax configuration with combined regularization indicating that Hyp.I could be true. To confirm that this is the case with some degree of certainty, a statistical hypothesis test is formulated:

Hyp.I-0 $\text{FNR}_{\text{sigmoid}} \geq \text{FNR}_{\text{softmax}}$

Hyp.I-1 $\text{FNR}_{\text{sigmoid}} < \text{FNR}_{\text{softmax}}$

A one-sided, two-sample t-test with unequal variance (Welch's t-test) is used to test this hypothesis. The desired confidence is 95%, therefore $\alpha = 0.05$. Hyp.I-0 is rejected in favor of Hyp.I-1 if $t > 0$ and $p/2 < \alpha$, where t is the t-statistic and p is the p-value of the *two-sided* t-test. Table 4-3 shows the t-statistics and p-values for the two-sided t-test for all metrics on the test set.

The value of t in a two-sided test indicates if the two sample means come from the same distribution, which is the case for $t = 0$. This is the null-hypothesis of the two sided test.

The p-value is the probability of the two-sided test that null-hypothesis is true, given the combined degree of freedom for both samples. In the one-sided case, $p/2$ gives the probability that H0 is true, i.e. $t < 0$, which is why we require $p/2 < \alpha$.

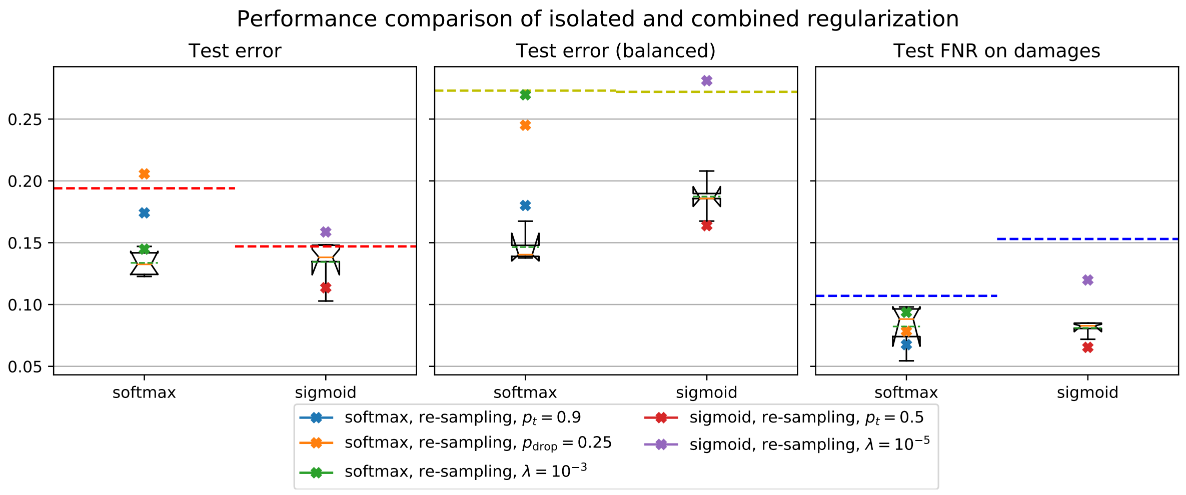


Figure 4-16: Comparing softmax and sigmoid with combined regularization after repeated training runs. The dashed lines denote the baseline without augmentation. The boxes range from the 1st quartile to the 3rd quartile of the sample. The median of the respective metric across all runs is the orange solid line, the mean the green dashed line. The 95% confidence interval is marked by the notch in the box, which can extend beyond the box itself, due to the small sample size. The whiskers extend to the full range of the sample.

For the FNR, we have $t = 0.116$ and $p = 0.914$, which does not meet the condition $p/2 < \alpha$ for the required confidence level. Therefore, we can not reject Hyp.I-0 in favor of Hyp.I-1, and can not confirm Hyp.I with 95% confidence.

However, when looking a the balanced error with $t = -4.286$ and $p = 0.004$ it is possible to conclude with a $1 - p/2 = 99.8\%$ confidence that the softmax single-label configuration has a lower balanced test error than the sigmoid configuration with combined regularization.

Table 4-3: Means and 95% confidence intervals for the training runs with combined regularization. Also shown are the t-statistic and p-value of the two-sided, two-sample t-test with unequal variance (Welch's t-test) from `scipy.stats.ttest_ind`.

Name	**Runs**	**Test Error**	**Balanced Test Error**	**Test FNR**
softmax	4	0.134 ± 0.009	0.146 ± 0.011	0.082 ± 0.016
sigmoid	5	0.134 ± 0.010	0.187 ± 0.008	0.081 ± 0.003
t-statistic	-	-0.070	-4.286	0.116
p-value	-	0.947	0.004	0.914

Chapter 5

Discussion

This chapter discusses the experimental results, attempts to provide explanations where the Hypotheses have to be rejected, analyses selected Convolutional Neural Networks (CNNs) from the experiments and also gives a discussion on the limitations of this work.

The results for the backbone feature extractor are discussed in Section 5-1. Section 5-2 discusses the results for classification head experiments. Section 5-3 discusses the results of the data re-sampling experiments and Section 5-4 discusses the results for the regularization techniques of data augmentation, dropout, and L_2-regularization. Section 5-5 discusses the combined regularization experiment results.

After the discussion of the experimental results, to better understand the differences between single-label and multi-label classification approaches, Section 5-6 discusses the classification performance on the confusion matrices and ROC-curves and Section 5-7 gives a qualitative analysis of the CNNs outputs. Section 5-8 concludes this chapter with the general limitations of this work.

5-1 Backbone Feature Extractor

5-1-1 Freezing the Convolutional Backbone

To test Hyp.A, a classifier is trained on the features of a pre-trained ResNet50 backbone with frozen weights and then with the complete network trainable. The experiments show that freezing the backbone is not sufficient to get a high performance. Fine-tuning the complete network gives a significant boost for all metrics. This confirms the findings in related literature where fine-tuned networks outperform networks with frozen weights.

However, the experiments in this work do not investigate what would happen if the complete network is trained from scratch. According to literature, this would lead to heavy over-fitting as long as the training dataset is small, but could lead to improved performance when more data is available [49]. It is not exactly clear how large the dataset has to be, so this remains a topic for future investigation.

5-1-2 Choosing a CNN Architecture

To find a good CNN-architecture in terms of False Negative Rate (FNR) as backbone feature extractor, an experiment is conducted by training five networks (ResNet50, ResNeXt50, DenseNet121, InceptionV3 and Xception) until convergence. The Xception architecture has the lowest FNR with 9.4%, while DenseNet121 has the lowest balanced error of 18.2%.

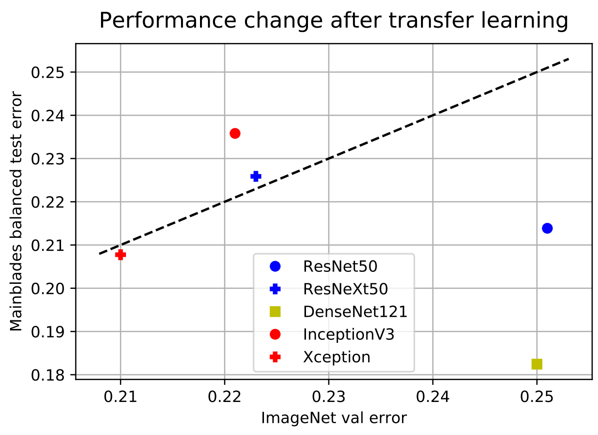


Figure 5-1: Effect of transferring from the general ImageNet source task to a more specialized target task: The balanced error on the inspection images test set against the ImageNet validation error. Models below the dashed lines have lower error and models above the dashed line have a higher error on the target task. No re-sampling is performed, and softmax activation is used for classification.

To visualize the actual influence of transfer learning, the performance on the target task (Mainblades damage classification) is plotted against the source task (ImageNet classification) in Figure 5-1. The balanced error on the Mainblades test data is used for a fair comparison with the ImageNet validation error.

InceptionV3 has clearly a higher error on our data than on ImageNet. The original Inception implementation uses so-called auxiliary towers. These are additional outputs at lower layers of the CNN that do the same classification as the final layer. This ensures that the training loss is propagated to the lower layers as well. This has not been implemented for the experiments in this work, which explains the slow convergence and poor performance of InceptionV3. The necessity of adding auxiliary towers can also be seen as an inherent flaw of that architecture.

ResNeXt50 and Xception each have error rates on our data that is comparable to their ImageNet error rate. ResNet50 and DenseNet121 perform significantly better on our data than on ImageNet.

To provide an explanation we look at the architectural building blocks of the models: InceptionV3 and ResNeXt50 have in common that they use parallel convolutional pathways in their architectures and perform worse on the Mainblades data. Xception, ResNet50 and DenseNet121 use only one convolutional pathway combined with a skip-connection and perform better on our data.

Although the importance of skip-connections is well known for training deep neural networks, it seems to be an important factor in this application. Convolutional blocks with skip-connections that consist of at most a 1×1 convolution, preserve high frequency information in an image better than convolutional blocks with no skip-connection. Therefore, they are better suited to forward-propagate information about smaller objects, especially when used in deep networks. Further investigation is necessary to confirm this, but it is definitely a promising finding.

Another possible explanation for the success of the Xception network in particular on damage FNR is the output size of its final feature map: The height and width of the Xception feature map is 10 pixels, while the height and width of InceptionV3 is 8 pixels and 7 pixels for all other models (See Table 3-1). The final neurons of Xception have therefore a smaller receptive field and are focused on a smaller region in the input image, which could also be beneficial for smaller features.

Both of these findings imply that a shallow model, using skip-connections and small convolutional kernels is best suited for this application. The surprising performance of the parameter efficient DenseNet121 in terms of balanced accuracy suggests that other light-weight CNN-architectures (e.g. MobileNet [80], MobileNetV2 [81], NASNetMobile [25], or the new EfficientNet [22]) should be included in the search for future work.

5-2 Classification Head

5-2-1 Global Pooling

Average-pooling, max-pooling and a combination of the two are tried to test Hyp.B: Max-pooling is better suited for small-object classification. Max-pooling has indeed the lowest metrics, confirming the hypothesis. Reasons for this are that average-pooling emphasizes features that span over a wide area in the feature map, while max-pooling is invariant to the size of a feature but only looks at the location in the feature map with the most dominant activation.

It is interesting that the combination of both also performs worse than max-pooling alone. A possible explanation is over-fitting due to the introduction of more neurons in the final dense layer, but more investigation has to be performed to draw a definite conclusion.

5-2-2 Final Activation Function - Single-label vs. Multi-label Classification

To find an optimal approach to the classification problem, a set of training runs is conducted to test whether multi-label classification with sigmoid activation and binary cross-entropy loss leads to a better FNR than single-label classification with softmax and categorical cross-entropy loss (Hyp.D).

The results for multi-label classification (with sigmoid activation and binary cross-entropy) are slightly poorer, when no regularization or re-sampling is applied, rejecting the hypothesis that multi-label classification leads to lower FNR.

However, all following experiments with re-sampling and regularization are done on both single-label and multi-label heads, to test if there exists a configuration where multi-label classification does outperform single-label classification.

5-2-3 Order of Classification and Pooling - Fully-connected Classification vs. Convolutional Classification

Another set of training runs is conducted to test whether a convolutional classification layer performs better than a fully-connected (FC) classification layer (Hyp.C).

The experimental results show that convolutional classification leads to poorer performance than FC-classification, especially when using softmax activation. Hyp.C is therefore rejected.

With convolutional classification, the dense connection between logit score calculation and activation function is broken by switching the order of the pooling operation and logit score calculation. As all outputs of the softmax layer depend on all inputs, this could be detrimental for the back-flow of gradients from the softmax layer to the parameters of the convolutional logit score layer during training.

An alternative, that rejoins the activation function with the logit layer, could be to switch the order of pooling and activation, such that we have:

$$\text{Backbone CNN} \rightarrow 1 \times 1 \times n_{feat} \times K \text{ conv. classification layer} \rightarrow \text{activation function} \rightarrow \\ \text{global pooling layer} \rightarrow \text{loss function.}$$

However, in this case, the dense connection between the activation function and the loss function is broken, which could have detrimental consequences as well.

Another explanation for the poor performance of convolutional classification is that perhaps some degree of non-local connectivity is required, which could be solved by increasing the kernel size of the convolutional logit layer from 1×1.

The most extreme case would be a kernel of size 10×10, the feature map size of the last Xception layer. This is equivalent to flattening the feature map and performing FC-classification without the need for pooling, at the cost of a significant increase (100-fold) in number of parameters.

5-3 Re-sampling

It is hypothesized, that re-sampling helps with the testing performance on the minority classes (Hyp.E). For both activation functions, the FNR decreases with pure re-sampling, the improvement is more profound for the softmax case. The balanced error should decrease as well, but this is not the case for the softmax configuration.

An explanation is that there the damages are actually not the most under-represented classes. There are other classes with very few examples. The pure re-sampling probably leads to over-fitting on those classes. This stresses the importance of using re-sampling with data augmentation (Hyp.F).

More sophisticated re-sampling techniques, possibly in combination with model ensembling, are conceivable that specifically optimize for performance on certain critical damage classes.

5-4 Regularization

The hypothesis to test for regularization is that it will help improve the testing performance in any case, due to the small size of the training data set (Hyp.G).

5-4-1 Data Augmentation

The experiments show that augmentation is indeed a very reliable form of regularization. Now, the benefits are more profound for the sigmoid activation. In combination with resampling, it can be seen that the performance of the bare-bones sigmoid classifier can be improved to the level of the softmax classifier.

Both now have similar FNR (6.8% for softmax and 6.5% for sigmoid). The sigmoid classifier also has a lower error and balanced error. With approximately the same number of false negatives on the damage classes, multi-label classification now has less overall false predictions, making it superior to single-label classification.

Data augmentation is identified as the single most effective regularization technique. This confirms Hyp.G for data augmentation. Except for longer training times, it also shows the least amount of negative side effects like increased variance in the validation loss.

5-4-2 Dropout Regularization

For softmax classification, dropout gives a minor improvement in FNR when used moderately up to $p_{drop} = 0.25$, but the error metrics increase. For the sigmoid case, dropout does not have a significant positive effect at all.

The experiments accept the dropout component of Hyp.G for softmax activation and reject it for sigmoid activation.

These observations are noticeable, because dropout is considered a good practice to prevent over-fitting throughout literature. It is possible, that dropout regularization interacts negatively with other parameters of the network and training (for example over-sampling or max-pooling).

Furthermore, it is recommended by Srivastava, Hinton, Krizhevsky, *et al.* [74] to increase the number of nodes before the dropout layer by a factor that corresponds to the dropout rate, and use different learning rates. Both are not investigated here.

A possibility worth investigating is combining the concatenation of max- and average pooling (Section 3-3) to double the number of inputs to the dropout layer and use a dropout rate of $p_{\text{drop}} = 0.5$. Additionally, the optimizer configuration can be specifically tuned for the use of dropout.

5-4-3 L_2 Weight Regularization

L_2-regularization in the softmax classifier has a positive effect on FNR, with $\lambda = 10^{-3}$ being optimal. The effect on the other two metrics is negative.

For the sigmoid head, the effect is positive until $\lambda = 10^{-5}$ for the FNR. The balanced error also improves with a higher λ but stays higher than the balanced error without augmentation.

For both softmax and sigmoid activations, Hyp.G is confirmed.

Although improvements in FNR can be achieved with L_2-regularization, there is barely any improvement on the other two metrics, which does not confirm Hyp.G, that L_2-regularization improves testing performance in any case.

A reason could be that the L_2 implementation of this work, together with the Adam optimizer and the automated learning rate finder, are not optimal, and that decoupled weight decay as proposed by Loshchilov and Hutter [77] should be used instead. In that case, the optimal learning rate is obtained independently of λ and vice-versa. As a result, there should be more consistent performance gains across all metrics on the testing data.

5-5 Combining the Regularization Techniques

The experiments that combine the regularization parameters that are found earlier in isolated experiments do not yield any improvements in FNR on average, rejecting Hyp.H. However, one of the runs results in a FNR of 5.4% which is lower than any value obtained before.

The balanced error of the combined regularization softmax run is with a high confidence better on average than wit isolated regularization.

The experiments show that there are (complex and so far unknown) interactions and stochastic effects between the different regularization techniques. They also show, by extension, that the limits of manual hyper-parameter search are quickly reached, when several hyper-parameters have to be tuned at once.

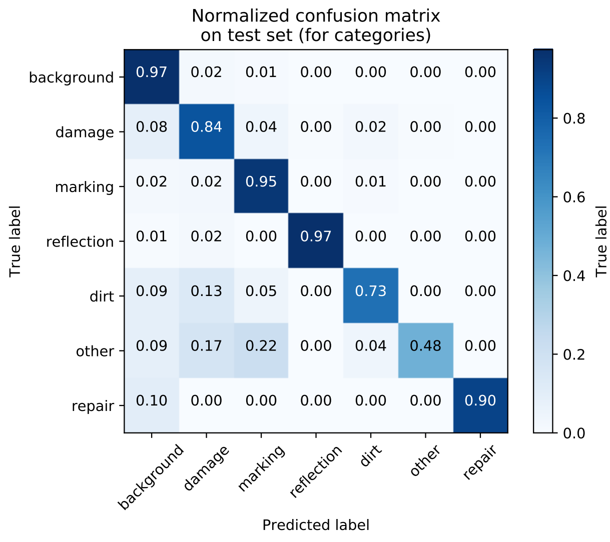
When setting up the experiment, it was assumed that the combining the best hyper-parameter values leads to the best possible configuration of each learning method. Therefore, it would have been possible to finally make a conclusive statement about single-label and multi-label classification. However, as there is reason to believe that the optimal configuration for both classification methods has not been found, the result of the final comparison with the Welch t-test has to be taken with care.

5-6 Analysis of Classification Performance

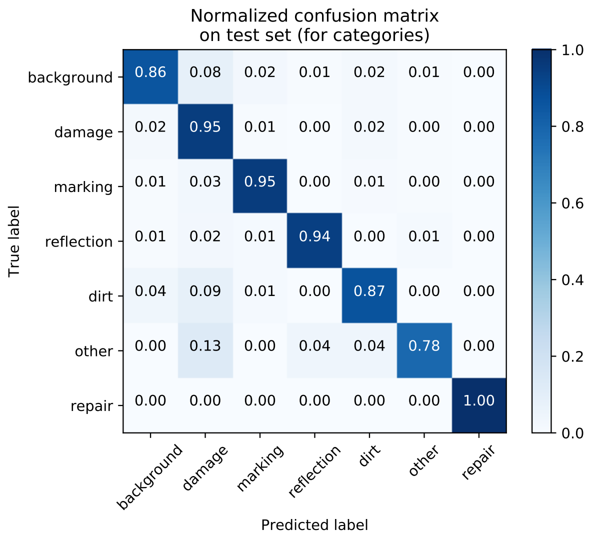
5-6-1 Confusion Matrices

Figure 5-2 shows the confusion matrices on the categories for the bare-bones single-label (softmax) classifier (a), the single-label classifier trained on re-sampled data and $p_t = 0.9$, $p_{\text{drop}} = 0.25$ and $\lambda = 10^{-3}$ (b), the bare-bones multi-label (sigmoid) classifier (c) and the multi-label classifier trained on re-sampled data and $p_t = 0.5$ (d).

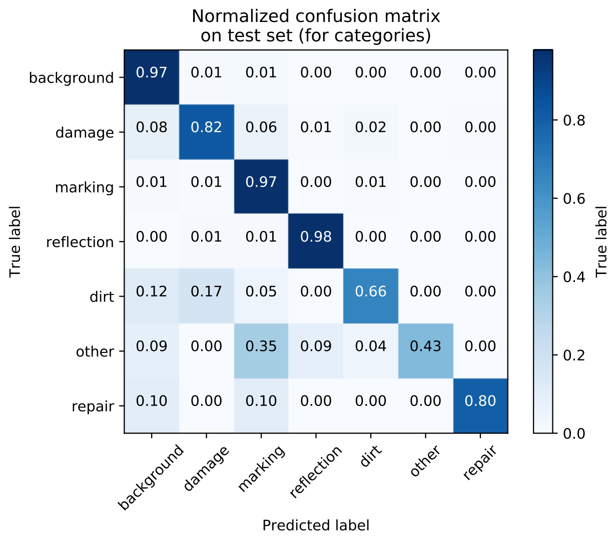
For the confusion matrices, a prediction is defined as the $\arg\max$ of the classifier's output probability vector for both single-label and multi-label classification. The entries on the diagonal of the confusion matrix correspond to the recall or sensitivity for each class.



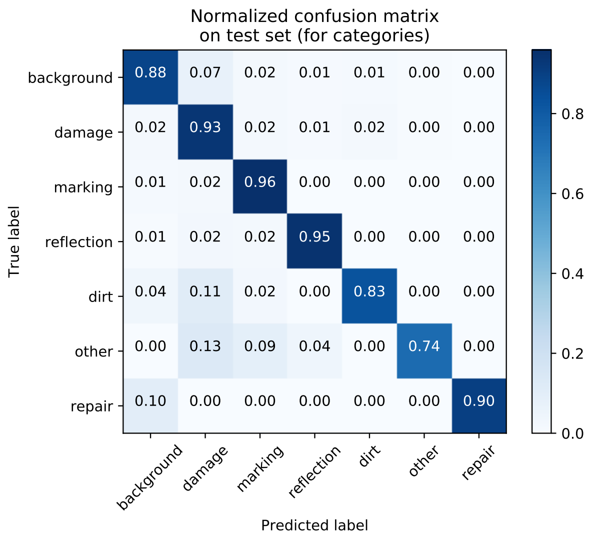
(a) Softmax, bare-bones



(b) Softmax, re-sampling, $p_t = 0.9$, $p_{\text{drop}} = 0.25$, $\lambda = 10^{-3}$



(c) Sigmoid, bare-bones



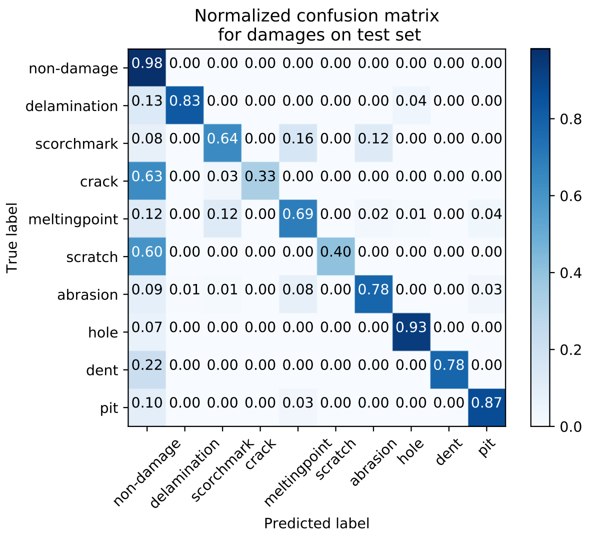
(d) Sigmoid, re-sampling, $p_t = 0.5$

Figure 5-2: Normalized confusion matrices for categories on a selection of models.

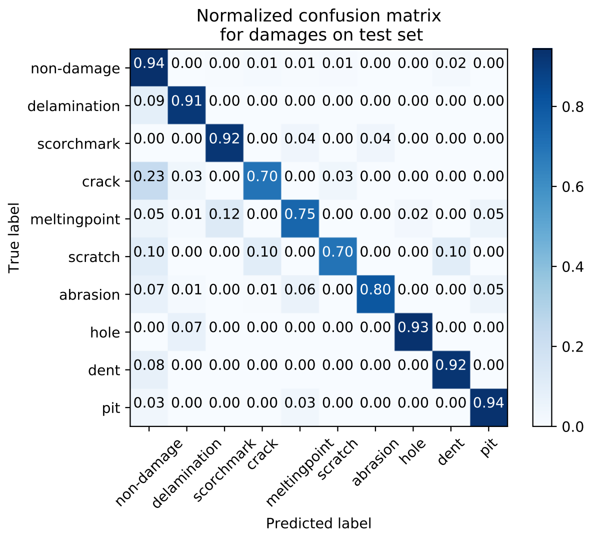
The training runs (b) and (d) are the best performing configurations that are found during this work in terms of FNR.

The bare-bones classifiers (a,c) perform well on the 'background', 'marking' and 'reflection' categories with 96%-98% recall. These are the over-represented categories in the dataset and the resulting overall accuracy score is accordingly high. For the under-represented categories, the recall scores are significantly lower. Note that 'damage' as a category is not under-represented, but some of the classes within that category are.

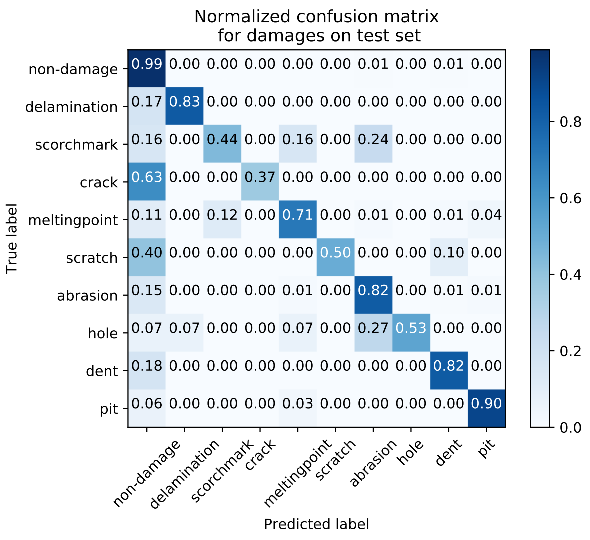
With the application of class-dependent re-sampling and data augmentation, it can be seen that more predictions fall correctly on the diagonal of the confusion matrix. Re-sampling and augmentation increase the recall for 'damages', 'dirt', 'other' and 'repairs' as expected. The average of all recall scores (equivalent to the balanced accuracy metric used throughout this work) is higher and the differences in recall between the categories are smaller.



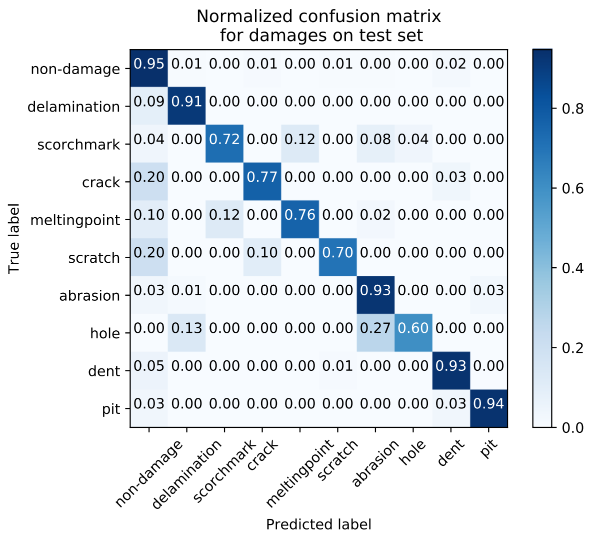
(a) Softmax, bare-bones



(b) Softmax, re-sampling, $p_t = 0.9$, $p_{\text{drop}} = 0.25$, $\lambda = 10^{-3}$



(c) Sigmoid, bare-bones



(d) Sigmoid, re-sampling, $p_t = 0.5$

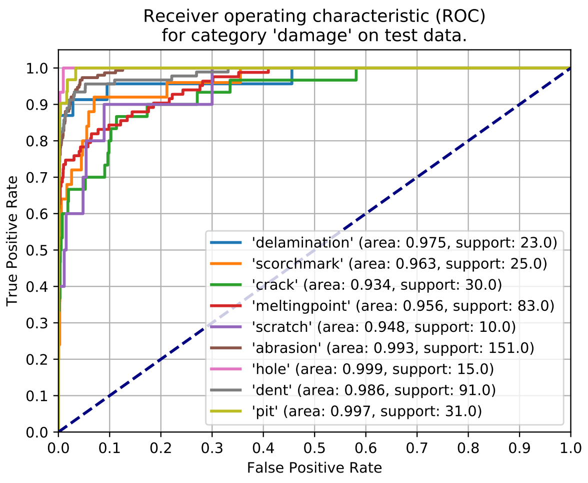
Figure 5-3: Normalized confusion matrices for classes in category 'damage' on a selection of models.

Figure 5-3 shows the confusion matrices for all classes in the 'damage' category. All other classes are combined under the 'non-damage' category. Again, for the bare-bones configurations (a,c), there are significant off-diagonal entries and false negative predictions of 'damages' into 'non-damages', for example for 'crack' and 'scratch'. Re-sampling and augmentation also decreases the rate of mis-classifications within the 'damage' category. but mainly mis-classifications into 'non-damages'.

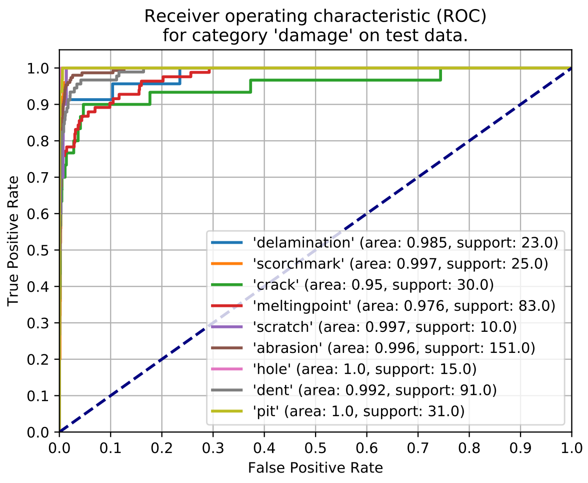
5-6-2 Receiver Operating Characteristic (ROC)

After training the CNN-classifiers, additional tuning of the classification thresholds can increase the True Positive Rate (TPR) (decrease FNR) at the cost of increased False Positive Rate (FPR). This trade-off is reflected in a ROC. The ROC-curves for all classes in category ‘damage’ for the four CNNs are shown in Figure 5-4.

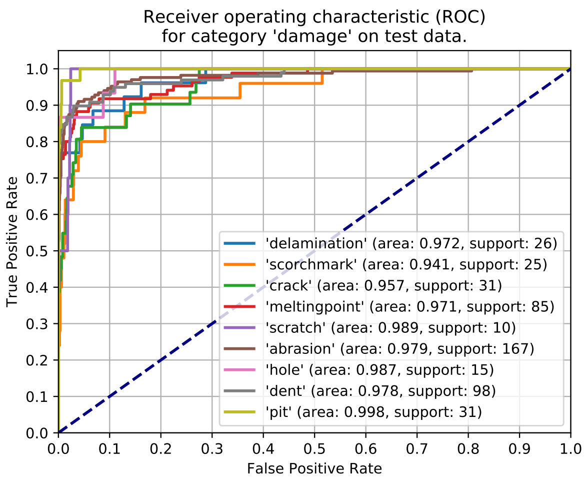
Note that for some classes, the support in the sigmoid case is higher. Although the images in both single-label and multi-label test sets are identical, the images in the multi-label test set are allowed to have more than one label, explaining the higher support.



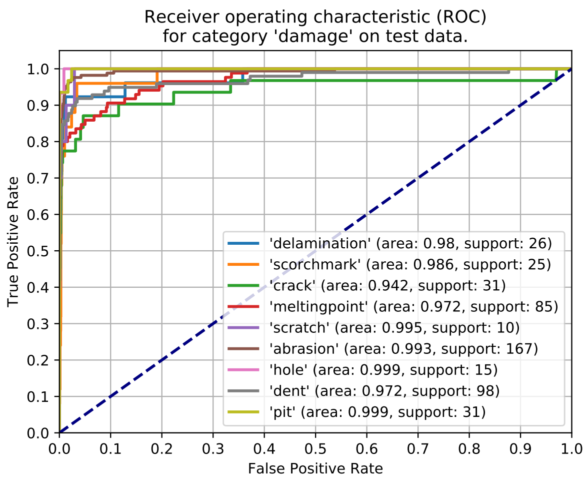
(a) Softmax, bare-bones



(b) Softmax, re-sampling, $p_t = 0.9$, $p_{\text{drop}} = 0.25$, $\lambda = 10^{-3}$



(c) Sigmoid, bare-bones



(d) Sigmoid, re-sampling, $p_t = 0.5$

Figure 5-4: Receiver Operating Characteristic (ROC) for for classes in category ‘damage’ on a selection of models.

Generally, augmentation and re-sampling increase the Area Under ROC (AUC) for most classes, except for ‘abrasion’ for the single-label and ‘crack’ and ‘dent’ for the multi-label configuration. With the current set of networks, at 100% TPR on all damages, the FPR is around 75% for single-label and almost 100% for multi-label classification, due to the under-represented 'crack' class and the notoriously difficult 'dent' class.[1] We acknowledge that more data and training is needed to get practical, acceptable FNR and FPR numbers.

5-7 Analysis of Network Outputs

5-7-1 Class Probability Scores

Figure 5-5 and Figure 5-6 show the output probabilities for some damage examples from the test set. The softmax and sigmoid networks that have generated the probabilities are the single-label classifier trained on re-sampled data and $p_t = 0.9$, $p_{\text{drop}} = 0.25$ and $\lambda = 10^{-3}$ and the multi-label classifier trained on re-sampled data and $p_t = 0.5$.

The examples are cherry-picked such that the strengths and weaknesses of softmax and sigmoid classification can be illustrated. More examples can be viewed in Appendix D.

The softmax classifier has the advantage that its output probabilities always sum up to one and that there is always an $\arg\max$ that defines a classification. The disadvantage is that it neglects the fact that an image could be one or more things.

For example in Figure 5-5a, the softmax classifier falsely predicts 'pit' with 77% and distributes the probabilities among all classes, resulting in a low score for the ground truth of 23%. The sigmoid classifier also falsely classifies the input as a 'pit' with 74% but also recognizes the correct class 'abrasion' with 67%, which is above the default threshold of 50%. In practice, both detections would be returned to avoid a false negative, especially when both detections fall under the 'damage' category.

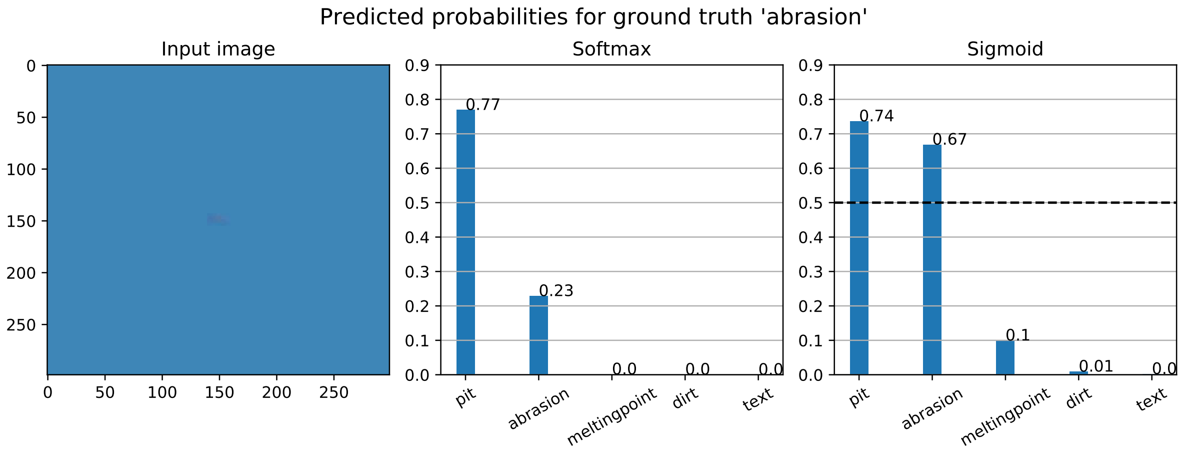
A similar observation can be made in Figure 5-6a, where the softmax classifier correctly predicts 'crack' with a high score of 97%. The sigmoid classifier also correctly predicts 'crack' with a score of 66% but also assigns a significant score to the semantically similar 'scratch' class. The softmax classifier suppresses the score for 'scratch' by means of the softmax activation function, but the sigmoid classifier is able to return probability scores for each class independently of the other classes.

A disadvantage of the sigmoid classifier is that it is possible that no class is triggered and nothing is detected, possibly resulting in a false negative, which could be caught by the $\arg\max$ or a top-k rule for the softmax classifier. This can be seen in the 'scorch mark' example in Figure 5-6b. A top-3 rule would have resulted in a correct prediction of the softmax classifier.

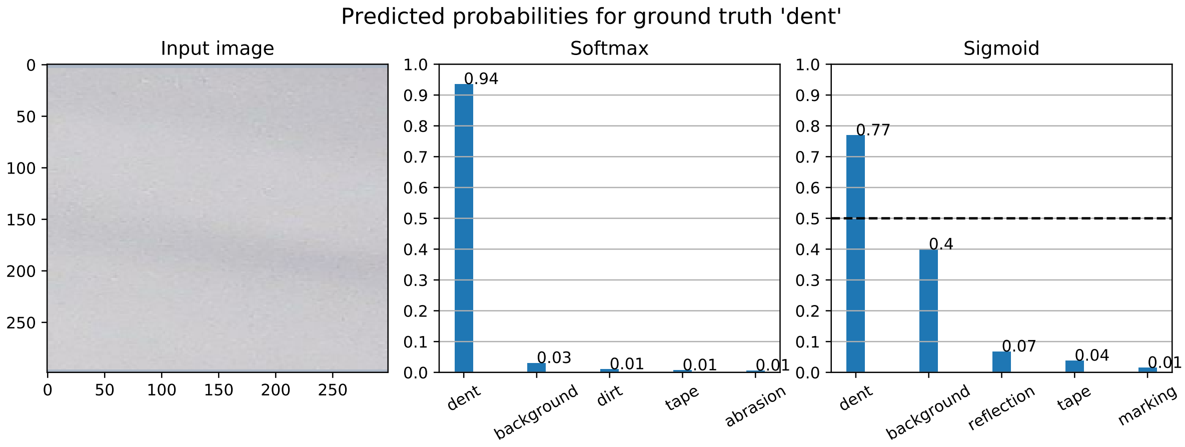
The pros and cons for the single-label and multi-label classification approaches are not trivial to trade-off. A middle way is conceivable where an ensemble of a single- and a multi-label classifier combine their predictions by means of a voting rule, which poses an interesting research topic for future work.

Another possibility is to use multiple classification heads, possibly in a hierarchical manner, that have different tasks and therefore, different activation functions.

[1] Dents are even for humans difficult to recognize when looking orthogonally at the surface. They become more visible at an angle, which is not the default operating mode of the drone camera. Nevertheless, soft gradients in the image make them recognizable in some cases. See Figure 5-5b for an example of a dent. Dent detection is not the main objective of this work. Other methods are investigated for this purpose.



(a)



(b)

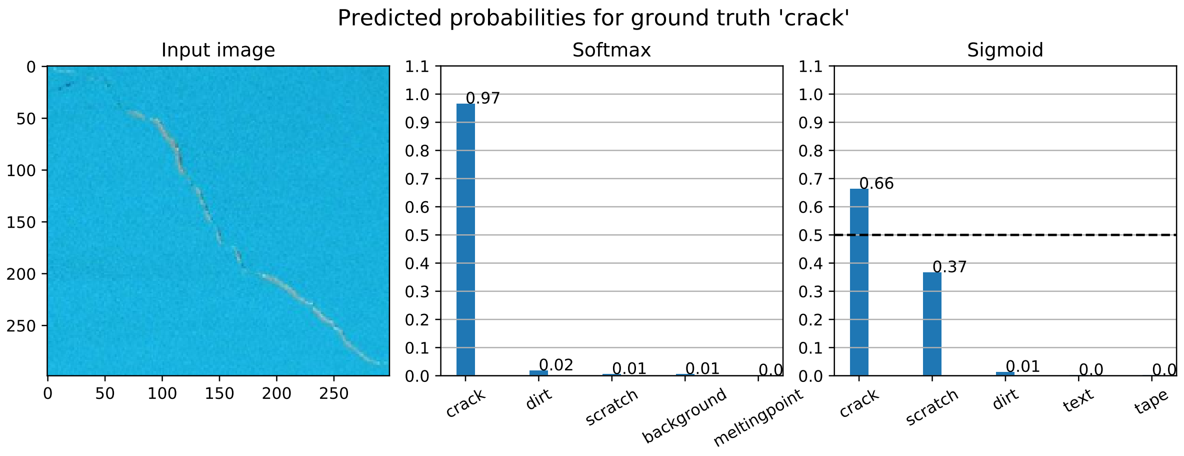
Figure 5-5: Predicted probabilities on 299×299 pixel crops of examples of 'abrasion' and 'dent'. The dashed horizontal line depicts the default classification threshold of 0.5 for the sigmoid classifier. A single-label detection is defined as the $\arg\max$ of the probability vector.

5-7-2 Class Probability Maps

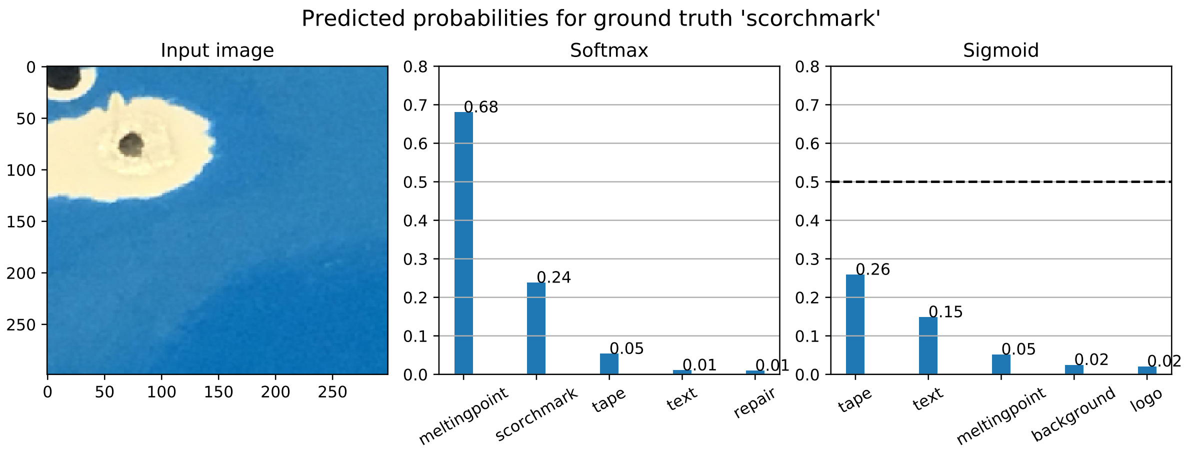
The trained networks will eventually not be used as classifier on 299×299 pixel crops. Instead, they will be used on larger images to detect damages, for example in a sliding-window manner similar to [3], [4]. After training on bounding box locations, the networks can also be used in a detection framework like Faster Region-based Convolutional Neural Network (R-CNN) [11] or Single Shot Detection (SSD) [82]. The advantage of the sliding-window technique is that no additional re-training is necessary at the cost of less precise localization.

To convert the existing networks into a sliding-window detector, the global max-pooling layer that reduces the last $10 \times 10 \times 2048$-shaped feature map of the Xception network is replaced by a *local* max-pooling layer with stride one. Additionally, the FC classification layer is converted into a 1×1 convolution, keeping the weights of the FC-layer.

It is important, that the stride of the pooling layer is set to one and not equal to the window size, as would be typical for a local pooling layer. This ensures that the network slides



(a)



(b)

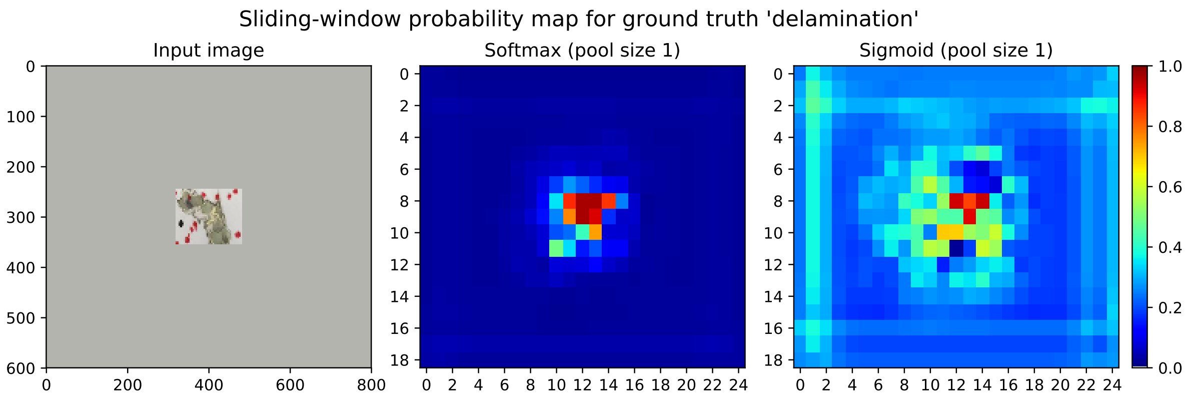
Figure 5-6: Predicted probabilities on 299×299 pixel crops of examples of 'crack' and 'scorch mark'. The dashed horizontal line depicts the default classification threshold of 0.5 for the sigmoid classifier. A single-label detection is defined as the $\arg\max$ of the probability vector.

over the input image and produces an output map instead of just classifying adjacent but non-overlapping patches independently.

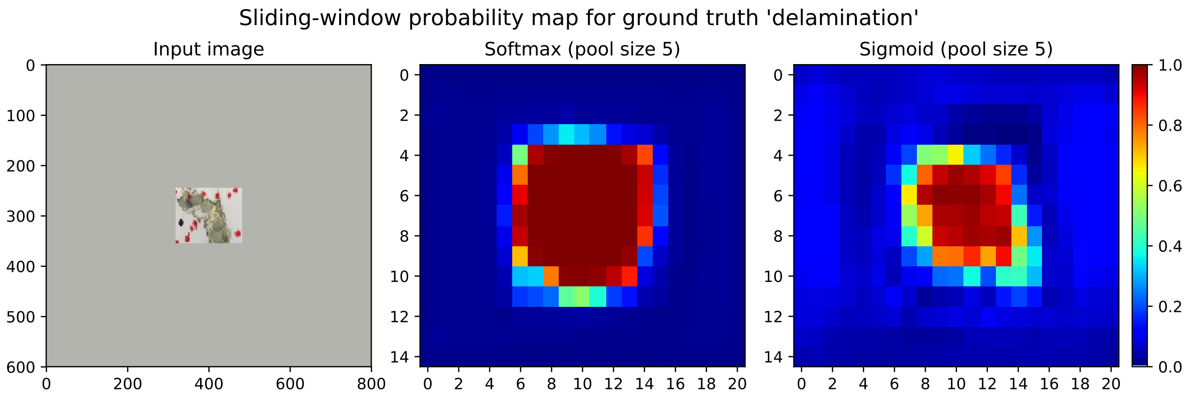
The window size of the local max pooling is now a new parameter that has to be set. A window size of 1×1 (i.e. no pooling) is equivalent to the convolutional classification idea in Section 3-3. A window size of 10×10 is equivalent to the original classifier: For an input of 299×299 pixels, the sliding-window classifier with 10×10 pooling window and the original classifier produce identical outputs. For larger images, a probability map is obtained with the sliding-window classifier, where the original classifier still outputs a probability vector over all classes.

Figure 5-7 shows the probability maps for two sliding-window classifiers with varying window size for the max-pooling layer to visualize their suitability as 'weak' location predictors in a sliding-window manner. The same network is used as in the previous section. More examples (for a fixed pooling window size) can be viewed in Appendix E.

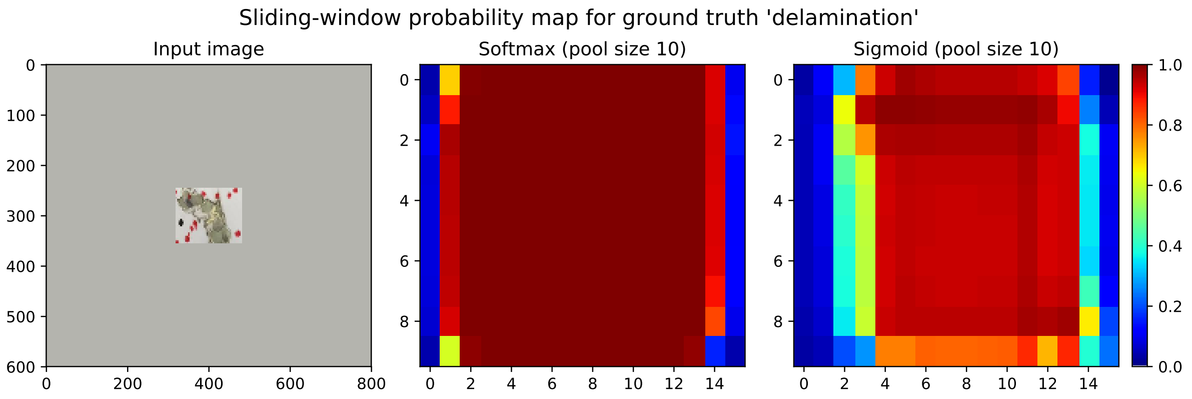
It can be seen that the fully convolutional sigmoid classifier is more sensitive to local features. However, the it produces more artifacts and noise, that could lead to false predictions. The softmax classifier profits from increased pooling window size and seems to produce a smoother probability map. For both, an increased pooling window size results in higher scores at the cost of lower spatial resolution and therefore less precise localization of a damage.



(a) 1×1 local pooling window size.



(b) 5×5 local pooling window size.



(c) 10×10 local pooling window size.

Figure 5-7: Class probability maps for a 600×800 pixel 'delamination' example and different pooling window sizes of a sliding window classifier.

5-8 General Limitations

This Section briefly discusses the general limitations of this work, that may or may not require a re-evaluation of the experiments and the conclusions drawn from them.

Dataset

The strongest limitations of the method presented here are rooted in the dataset: The difference in training and validation/testing performance is not high across all experiments, which is usually an indication that the network has a good generalization capacity.

However, because of the small dataset size, this is a dangerous conclusion. For both single-label and multi-label approaches, it is problematic that one object instance can occur in multiple bounding boxes. This complicates exact, class-dependent sampling and leads to data leakage from the training into the test set.

Also, the method of extracting bounding boxes leads to isolation of the objects from its source context, which simplifies training and gives good results during testing, if the test objects are also extracted from their source context in exactly the same way. However, when deployed to the target task, the added context will result in performance losses.

Optimization Parameters

The process of training is not optimized in any way; e.g. there is no reasoning behind the choice for the optimizer other than that it is commonly used. Optimizer parameters and the cyclical learning rate parameters are kept fixed at their originally proposed default values. For training a classifier for actual deployment, these parameters have to be included in the search process to get additional gains in testing performance and generalization.

However, the arguably most important training hyper-parameter, the learning rate, is found in an automated way specifically for each configuration (sampling & augmentation & network & loss), see Appendix B.

Generally, it has to be acknowledged that proper hyper-parameter tuning plays an important role when training deep neural networks; a bad network with good hyper-parameters can outperform a good network with bad hyper-parameters.

Termination Criterion

The termination criterion is not optimal. Due to the lack of off-the-shelf availability of balanced metrics (class-specific recall, balanced accuracy, etc.) in the `tf.keras` framework at the start of experimentation, the sample-weighted validation loss is used as a metric where the termination criterion is evaluated on. It is observed, that the actual metrics of interest keep improving after the validation loss has converged.

This turns out to be a problem for training runs with regularization on softmax activation, where training is terminated very early due to the noisy validation loss. This contributes to the poor performance of those runs, as running longer (with more parameter updates) seems to improve generalization [83]. However, for consistency with the older experiments, the termination criterion has not been changed, and it is recommended for future work to repeat the experiments with more appropriate monitoring metrics and termination criteria.

This poses the question, if a high degree of regularization improves generalization as intended due to constraining the model capacity (dropout and L_2) or synthetically increasing the dataset size (augmentation), or if the improvements are caused by the longer training times that are induced by increased regularization. This question is left open for future work.

Statistical Significance

The experimental set-up, with the exception of the set-up in Section 4-6, lacks statistical significance: All of the training runs for each configuration are performed only once, instead of performing multiple runs and averaging the results. This does not reflect the variance that is present in the models. Therefore, small changes between different configurations can not necessarily explained by the changed parameter, but could be attributed to chance.

However, the experiments in this work show qualitatively the influence of the parameters and design choices where trends or patterns are visible. The discussion aims to abstain from over-interpreting the results where differences are small.

Chapter 6

Conclusion

This chapter concludes the findings of this work by answering the question posed in the Problem Formulation in Section 6-1. For each part of the Problem Formulation, the results of testing the hypotheses are summarized. Lastly, Section 6-2 proposes topics for future work.

6-1 Problem Formulation and Hypotheses

In the Introduction the following question is asked:

"How can a *visual recognition algorithm* be developed, that reliably classifies *small damages* with *low False Negative Rate (FNR)*, given that there are *large class imbalances* present that bias against the damages, and given *limited training data* in general?"

The problem formulation was broken up into five parts. For each part, a set of goals, assumptions and hypotheses were formulated based on previous related work.

I - Visual Recognition Algorithm The goal for this part was to find the basic architecture for an image classifier. Testing the hypotheses lead to the following result:

Hyp.A It is found, that transferring a network with weights from ImageNet, freezing its weights and training only the top-layer is not sufficient. Fine-tuning the weights provides better performance in terms of all metrics, by a factor of five on ResNet50.

Using that result, a search over a selection of pre-trained model architectures was conducted. The search yielded the following results, that provide useful insights for future work:

- The Xception [21] network performed best in terms of recall and convergence time. DenseNet121 [20] had the highest balanced accuracy.

- The number of network parameters by itself is not indicative for eventual performance. In our experiments, the architectures that make efficient use of their parameters (DenseNet121 and Xception) performed better than the ones that have more parameters.
- ImageNet performance is not indicative for the performance on a specialized application and even less so when different metrics are employed: Single pathway residual networks with skip-connections (ResNet50, DenseNet121 and Xception) exhibit advantageous properties and have higher performance on our data relative to their ImageNet performance. Networks with multiple parallel pathways (InceptionV3 and ResNeXt) do not perform as well on this task.
- The resulting network can be converted into a sliding-window classifier that provides 'weak' (i.e. imprecise, suggestive) predictions of damage location, without additional data and training. Over time, additional, more precise outputs like bounding box predictions can be added.

II - Small Damages This goal of this part was tp find an optimal classification head for small damages. Testing the hypotheses found:

Hyp.B A global max-pooling layer that emphasizes small features before the dense layer works well in this application. The commonly used average-pooling performs considerably worse. The combination of average- and max-pooling performs also worse than max-pooling alone.

Hyp.C Training a classifier that performs class logit calculation in a convolutional manner from scratch does not provide any gains with respect to the conventional fully-connected classification method. Apparently, some degree of non-local connectivity is required for good performance. This is confirmed when converting a fully-connected (FC) classifier to output probability maps in a convolutional manner.

III - Low FNR This part focused on the choice and interpretation of the classifier output. The goal was to find the best classification approach: The following has been found:

Hyp.D Performance with a single-label classification head (softmax activation) is higher than with a multi-label head (sigmoid activation) when using a bare-bones configuration. When re-sampling and augmentation are applied, the performances are similar.

The single-label and multi-label approaches proved difficult to trade off. It remains to be investigated, what type of activation function in the output layer truly provides better performance, when considering how the algorithm will be used in practice and how it interacts with the user.

As a proof-of-concept, it was shown that damages can be correctly be classified with a recall of up to 94.6% (5.4% FNR) with the current dataset. This was done without exploring the interactions between hyper-parameters and without designing an architecture specifically for this problem.

As the dataset is growing, it will be possible to obtain more statistically relevant results, that also quantify the reliability of the model.

IV - Low FNR & Large Class Imbalances The goal of this part was to find a simple, but effective data-based class imbalance method that decreases FNR specifically on the minority classes. The hypothesis test yielded the following results:

Hyp.E Hybrid re-sampling in its simple implementation presented here, has a positive effect on FNR, as expected.

Hyp.F For the softmax activated classifier, re-sampling has to be combined with data augmentation to also yield gains in balanced accuracy. The combination of re-sampling and augmentation provides consistent overall gains in the balanced metrics that are larger than the sum of the individual gains.

V - Limited Data The focus of this part was to identify the effects of regularization methods to tackle the common problem of limited data. The regularization techniques were applied in isolation, independently of each other and are then combined: This lead to the following results:

Hyp.G **Data augmentation** as a standard regularization method in related work is the most effective regularization technique in this work. All three metrics for both softmax and sigmoid activation see consistent improvements. The performance gains for sigmoid activation are more profound. The full potential of data augmentation has not been explored yet, as the experiments only used geometrical and no color transformations.

Dropout only provides a small gain in recall for softmax activation when used moderately and no gains for balanced accuracy. For sigmoid activation, no clear positive effect is observable in the experiments. This is not consistent with the common practice of applying dropout for regularization.

L_2-regularization provides small gains in recall but not in balanced accuracy. The optimal magnitude of the L_2-coefficient λ is different for softmax ($\lambda = 10^{-3}$) and sigmoid activation ($\lambda = 10^{-5}$) due to the different scales of the categorical and binary losses.

Hyp.H Naively combining the best results of using the regularization techniques in isolation, does not yield additional gains in FNR performance on average. The softmax classifier with combined regularization shows significantly improved balanced error on averagen and produces the best run found in this work with 5.4% FNR.

Hyp.I Although multi-label classification with combined regularization shows slightly lower FNR than single-label classification on average, this doesn't allow drawing conclusions about the superiority of multi-label classification, because the previous hypothesis had to be rejected, and the statistical confidence is low.

In general, running isolated experiments on the regularization techniques and naively combining the results, shows the limits of manual search for the hyper-parameters.

By implementing more advanced hyper-parameter tuning, augmentation and other regularization techniques, further performance gains should be possible. Automated search techniques, that directly optimize on the metric of interest, can discover interactions between the hyper-parameters, that would be hard to find with manual search.

6-2 Future work

The following immediate implementation changes are proposed to address the limitations of this work:

- Re-generate the dataset such that objects are not extracted and isolated from their source context. In this work bounding boxes are individually extracted by cropping, centering and padding from the source image. The target image thus contains one 'main' object of the bounding box and possibly other overlapping objects. Instead, the source image could be evenly cropped into patches similar to [3], [6]. This would then result in a more appropriate multi-label dataset, as several objects can now be equally present in one patch. It would be also easier to estimate the true distribution between foreground and background.
- Widen the search to more architectures, e.g. MobileNet [80], MobileNetV2 [81], NASNetMobile [25], EfficientNet [22].
- Include skew and color transformations for data augmentation.
- Implement the AdamW optimizer [77] to decouple weight regularization from loss optimization for better generalization and independent tuning of learning rate and λ.
- Decay the maximum learning rate of the cyclical learning rate scheduler instead of using a constant learning rate amplitude to get more performance close to convergence.
- Use the actual metrics of interest when monitoring and terminating the training runs.

Now that the (isolated) influence of the different design and training configurations on the performance of the classifier is understood, the first step after concluding this project would be to automate the search of all other hyper-parameters in an efficient manner, also exploring the interaction between the hyper-parameters, for example with Bayesian techniques [23].

The resulting network, can be frozen or fine-tuned as a backbone feature extractor in a detector like Faster Region-based Convolutional Neural Network (R-CNN) and Single Shot Detection (SSD) to also predict bounding box locations. A sufficiently large dataset with bounding box labels is required for this.

Possibly, additional modifications have to be made to the detection framework to deal with the application-specific aspects (e.g. detection of small bounding boxes on high-resolution images make it hard to use sub-sampling for efficient computation), that are somewhat different to the main-stream research on object detection.

Because the classifier/detector will be deployed in a safety critical application, it is strongly recommended to implement algorithms that estimate the uncertainty of a prediction. Attention should be paid to verify human annotations before feeding them as training data. On the other hand human feedback can be used in a fail-safe mechanism and for continuous improvement.

For the long run, the application-specific findings make it worthwhile to investigate into alternative architectures that are tailored to the target task. When more data is available, this process can be automated by employing Automated Machine Learning (AutoML) and Neural Architecture Search (NAS) techniques [23].

Appendix A

Dataset

In this chapter, the dataset and process of collecting the data is described.

A-1 Data Collection

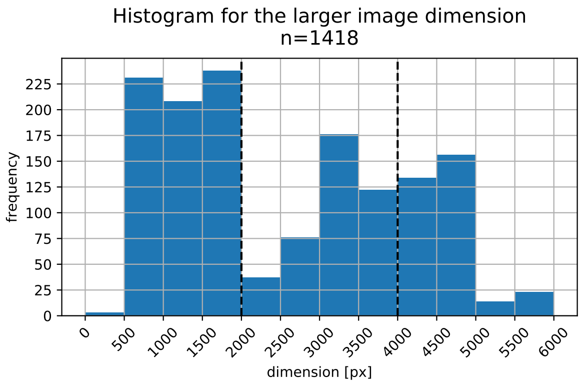
The image data for this projected is collected from various resources. The majority of damage images is provided by KLM Engineering and Maintenance, without structured labels. A smaller part of the image dataset is taken by the DJI X5 drone camera.

The content and quality of those images varies, some photos are taken before a repair and do not have any manually added annotations on them (which is the desired form). However, many images are heavily annotated and some show the damages after they have been repaired. Images that are taken from the inside of a plane or are not related in any other way to the inspection context, are removed from the dataset.

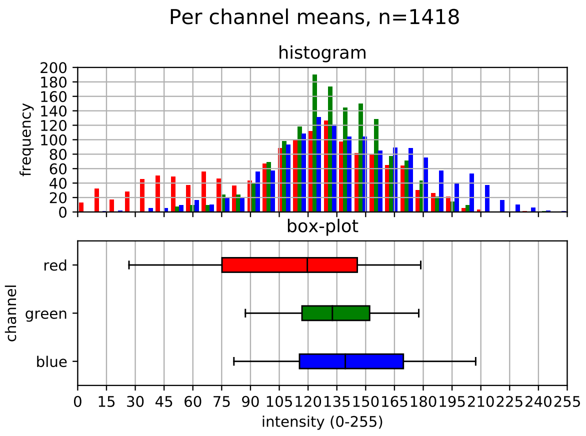
The result is a dataset of 1418 inspection images. Figure A-1 shows the general statistics of the inspection images. It is clear from Figure A-1a, that a majority of images has a smaller resolution than the X5 drone images, which have a 4k resolution (at least one image dimension has 4000 pixels or more). We generally consider all images above 2k desirable. The 2k and 4k limits are marked with dashed lines in Figure A-1a.

Figure A-1b shows the color statistics of the inspection images. The box-plots confirm the suspicion that the mostly KLM-sourced dataset has a blue bias. What the effect of that is, and how to mitigate possible risks due to this color bias, is not investigated in this work. Methods like converting the images to gray-scale and extensive color transformations for training data augmentation are conceivable as temporary solutions. More importantly, data from airplanes with different colors needs to be acquired.

As the availability of training images is limited, all of inspection images are used for training, validation or testing, despite their poor properties. For a proof-of-concept, this is acceptable. However, when training a model for deployment, more data has to be acquired with the desired properties, and the selection of training examples has to be more strict.



(a) Distribution of image dimensions. Only the larger image dimension is used, i.e. width for landscape photos. The dashed vertical lines denote the 2k and 4k marks.



(b) Color distribution in the dataset. Each data point is the per-channel mean of one image. Whiskers of the box-plots extend to the 5th and 95th percentile, respectively.

Figure A-1: Image statistics of the inspection images, partly provided by KLM Engineering & Maintenance and partly collected with the DJI X5.

The inspection images that are subsequently labeled as follows:

- At least one polygon outlining the aircraft section(s)
- Bounding boxes outlining damages and other objects of interest, annotated with categories that are then subdivided into different classes.
- Attributes like image quality, lighting conditions and if the photo was taken inside a building (hangar) or outside.

The categories are subdivided into the following classes:

- Background: generic, catch-all background class. Can be aircraft hull or other non-aircraft background (Figure A-5a).
- Damage:
  - Delamination: Removed surface layer of composite structures (Figure A-5b).
  - Scorch mark (lightning strike): scorched, blackened area caused by lightning strike (Figure A-5c).
  - Crack: Damaged part with significant structural damage (Figure A-6a).
  - Melting point (lightning strike): Molten area due caused by lightning strike (Figure A-6b).
  - Scratch: Damaged surface, or paint. No structural damage (Figure A-6c).
  - Abrasion: (Figure A-7a)
  - Hole: (Figure A-7b)
  - Dent: Deformation (mostly metal) of an area, e.g. due to a collision with a bird (Figure A-7c).
  - Pit (lightning strike): very small puncture in the hull caused by lightning strike (Figure A-8a).

- Markings:
  - Text: Printed or handwritten text, can also be added by image manipulation software (Figure A-8b).
  - Tape: Temporarily applied tape used to mark damages or repair tape (Figure A-8c).
  - General marking: For example emergency exit marking on the wings, arrows for opening doors etc. (Figure A-9a).
  - Logo: Airline logo, flag, engine manufacturer logo (Figure A-9b).
  - Sticker: Specifically so called ‘Registered Damage’ stickers used by KLM to mark damages that are already known (Figure A-9c).
- Reflections: Reflections caused by lamps or the sun. Later actual lamps and the sun are included as well because they are indistinguishable from reflections (Figure A-10a).
- Dirt: all sorts of non-nominal appearances due to corrosion, oil or other fluid spills, bird droppings, etc. (Figure A-10b).
- Repair: A damage that has been repaired, but is still distinguishable from the nominal appearance of the repaired part. For example a damaged rivet that has been drilled out and replaced by a new, larger rivet (Figure A-10c).
- Other:
  - Angle-of-attack sensor: Sensor to measure angle-of-attack (Figure A-11a).
  - Pitot tube: Sensor to measure airspeed (Figure A-11b).
  - Static discharger: Long object used to discharge static electricity to the surrounding air, often point of entry or exit of a lightning strike (Figure A-11c).

Of course, there are many more objects to be found on an aircraft, but their low frequency in the available data does not justify using them for any machine learning purposes. Figure A-2 shows the general statistics of the dataset.

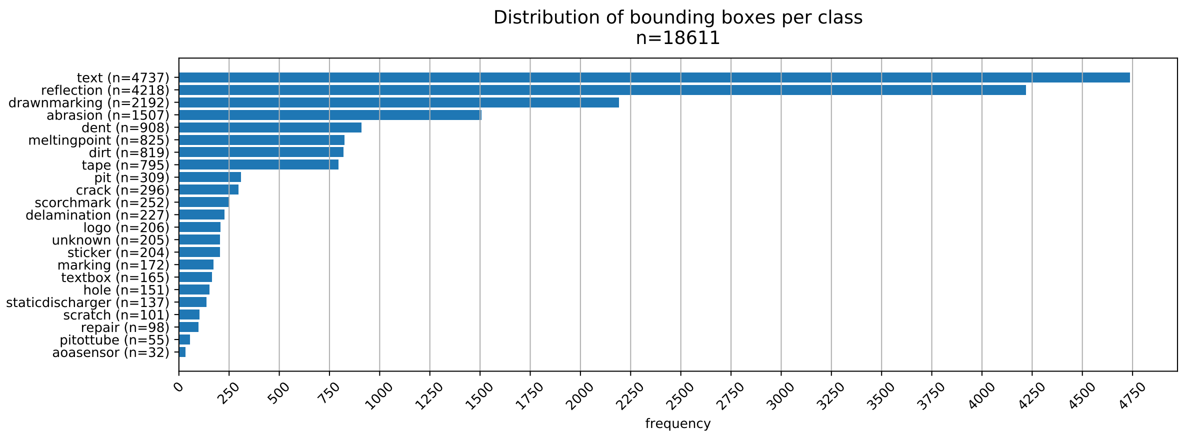


Figure A-2: Distribution of classes in the dataset.

A-2 Detection Data

The data for the detection task consists of the images and bounding box coordinates and class labels. The aircraft polygon is initially also converted into a rectangular bounding box. All images are rotated to landscape format, but are not reshaped to a fixed shape by default.

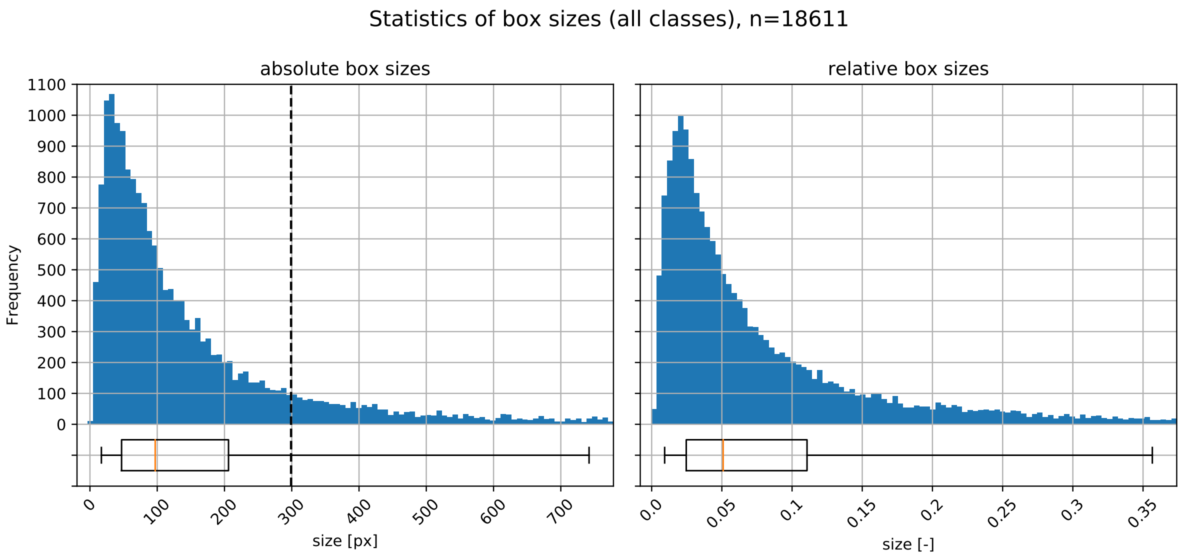


Figure A-3: Distribution of bounding box sizes for all classes. Whiskers of the box-plots extend to the 5th and 95th percentile, respectively.
Left: absolute bounding box size $\max(\text{box height}, \text{box width})$ in pixels. The dashed vertical line denotes 299 pixels, which is the input size of Xception [21].
Right: relative bounding box size $\max(\text{box height}/\text{image height}, \text{box width}/\text{image width})$.

A-3 Classification Data

A-3-1 Single-label classification

To create a dataset for the multi-class, single-label classification task, the following procedure was applied to the detection data:

1. A target output image size of 600×800 pixels is specified.
2. Random bounding boxes with a minimum size of 75×100 pixels are sampled that do not overlap with the existing object bounding boxes to extract background examples.[1]
3. For each bounding box, the intersection (IoU) with other object bounding boxes is checked, the box is discarded when the intersection exceeds a certain threshold.
4. The content of the bounding boxes are extracted *without changing their pixel size.*

[1]The search for background patches is brute-forced: 1. Sample random bounding box. 2. Check overlap with other boxes. 3. If no overlap: add box to background class. Else: discard box. 4. Repeat 1.-3. until maximum iterations or number of required background patches is reached.

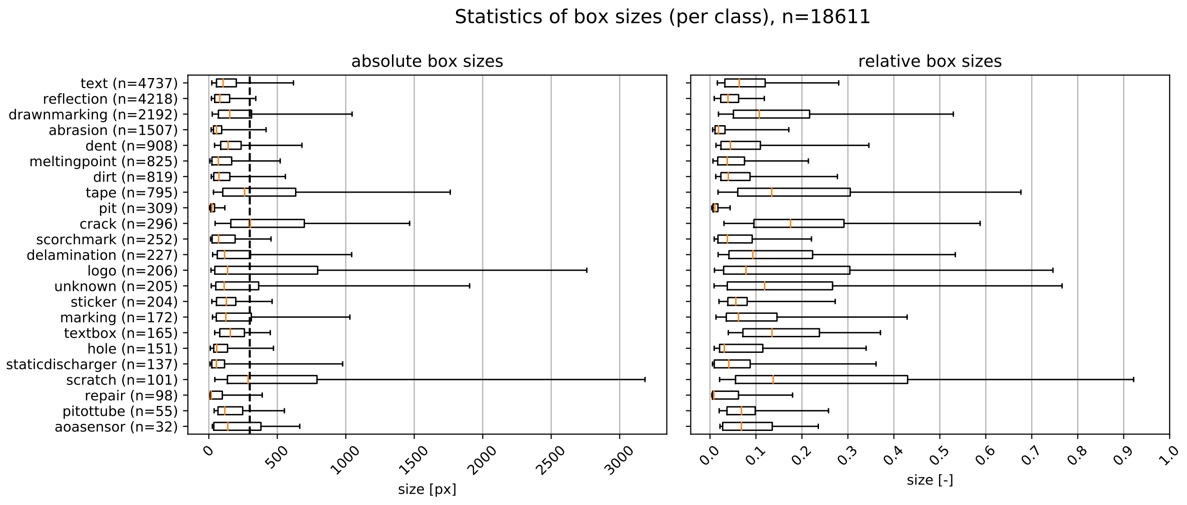


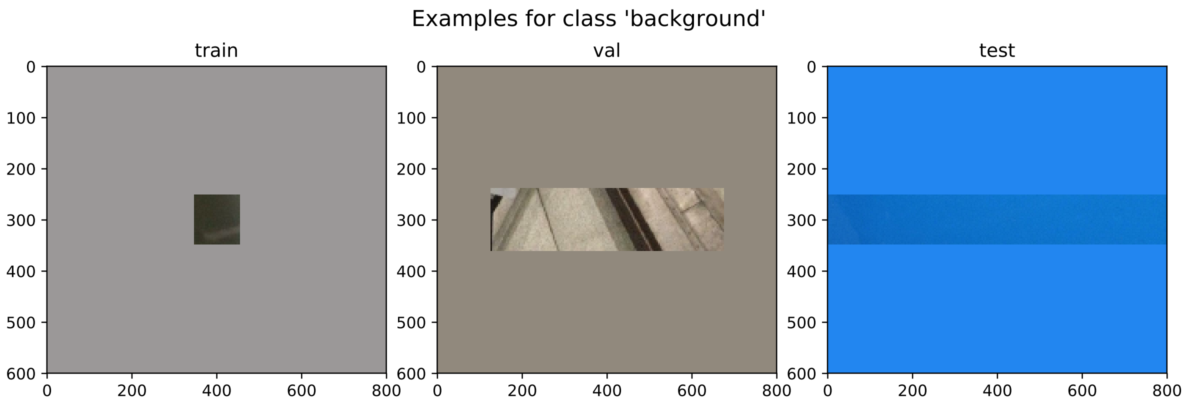
Figure A-4: Distribution of bounding box sizes per class. Whiskers of the box-plots extend to the 5th and 95th percentile, respectively.
Left: absolute bounding box size $\max(\text{box height}, \text{box width})$ in pixels. The dashed vertical line denotes 299 pixels, which is the input size of Xception [21].
Right: relative bounding box size $\max(\text{box height}/\text{image height}, \text{box width}/\text{image width})$.

5. if the bounding box is larger than the target size, the patch is resized and padded to the target size, keeping the bounding box aspect ratio.
6. If necessary, the resulting patch is rotated to landscape format.
7. Finally, the patch is padded with the channels means of the source image to the target output size.

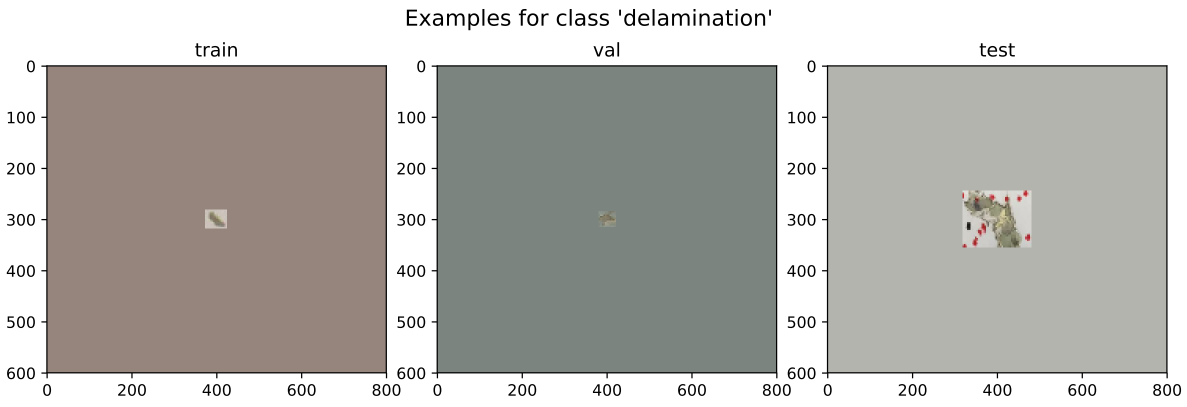
A-3-2 Multi-label classification

For the multi-class, multi-label classification task is created by slightly changing the previously described procedure:

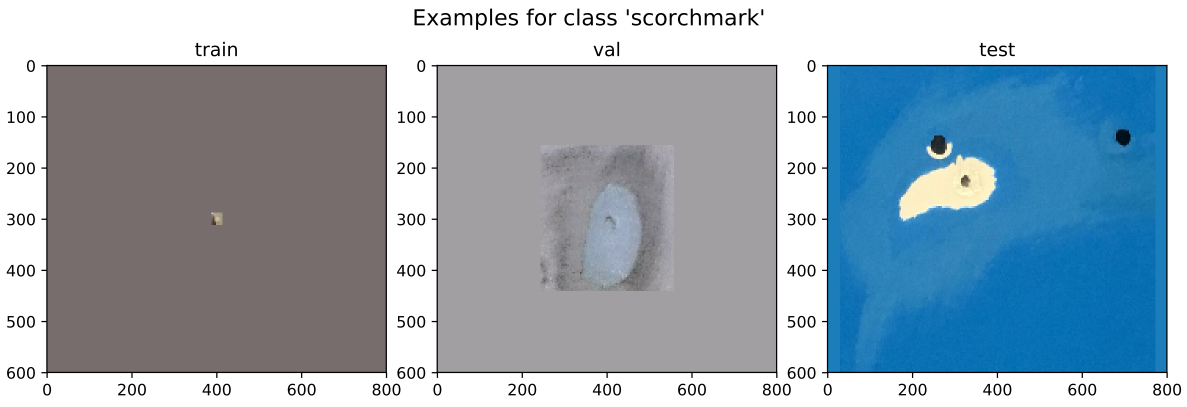
1. A target output image size of 600×800 pixels is specified.
2. Random bounding boxes with minimum size 75×100 pixels are sampled that do not overlap with the existing object bounding boxes to extract background examples.
3. For each bounding box, the overlap with other bounding boxes is checked. For each class, the total union of bounding boxes that falls within the current bounding box is calculated. The union is divided by the area of the current bounding box to get a ground truth score for each class, resulting in a maximum possible score of 1 for each class.
4. See single-label procedure for steps 4-7.



(a)

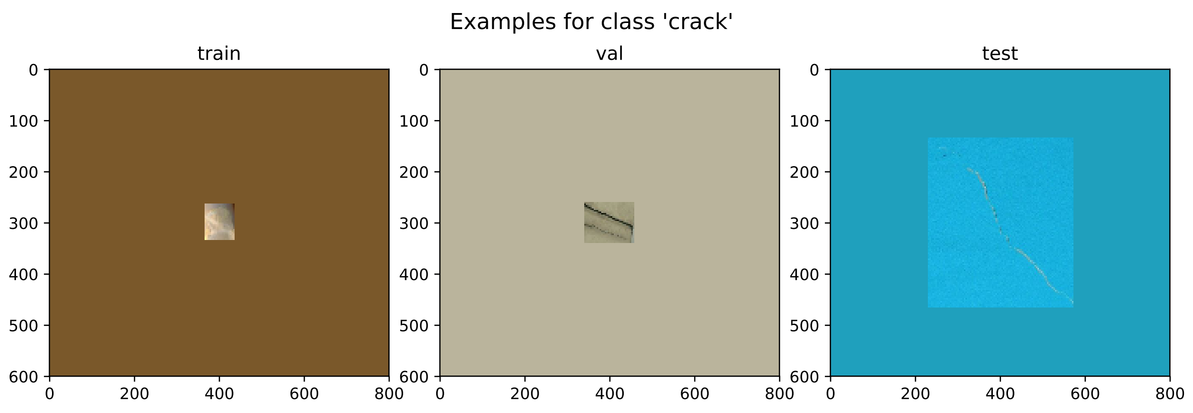


(b)

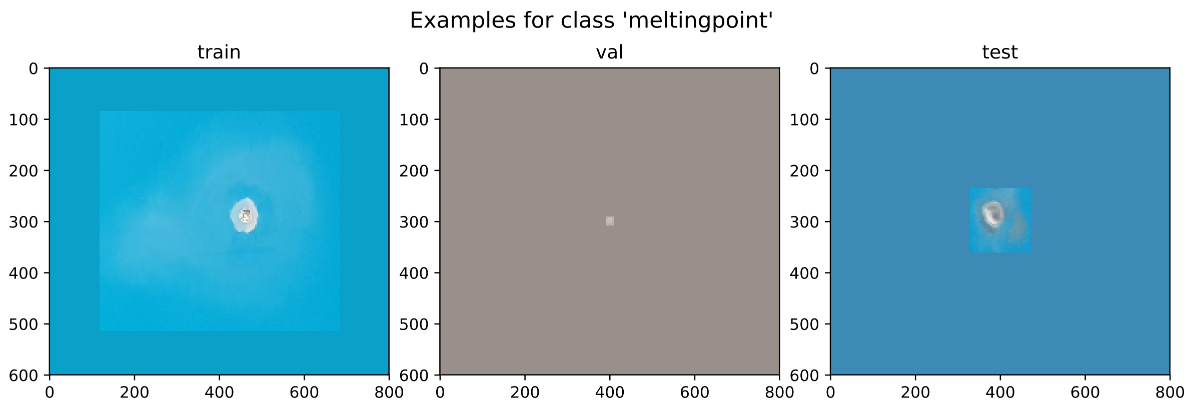


(c)

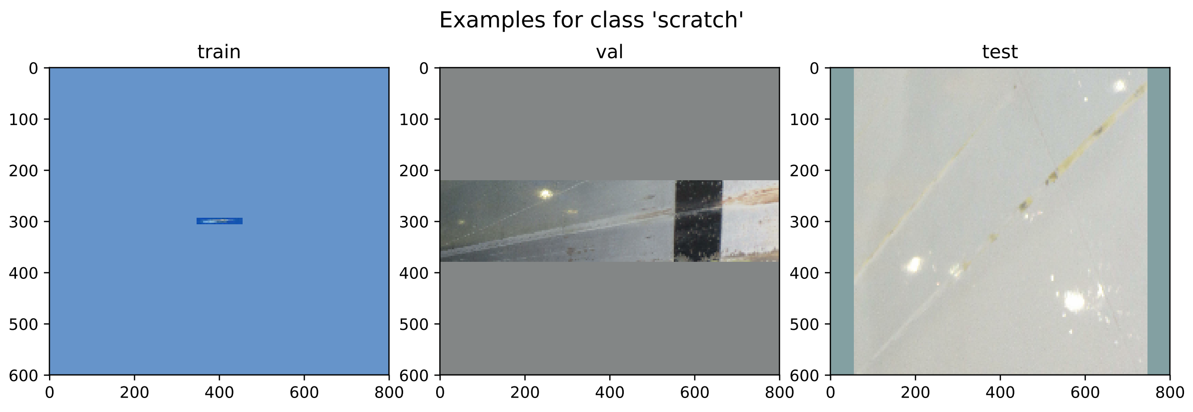
Figure A-5: 600×800 pixel examples of 'background', 'delamination' and 'scorch mark'.



(a)

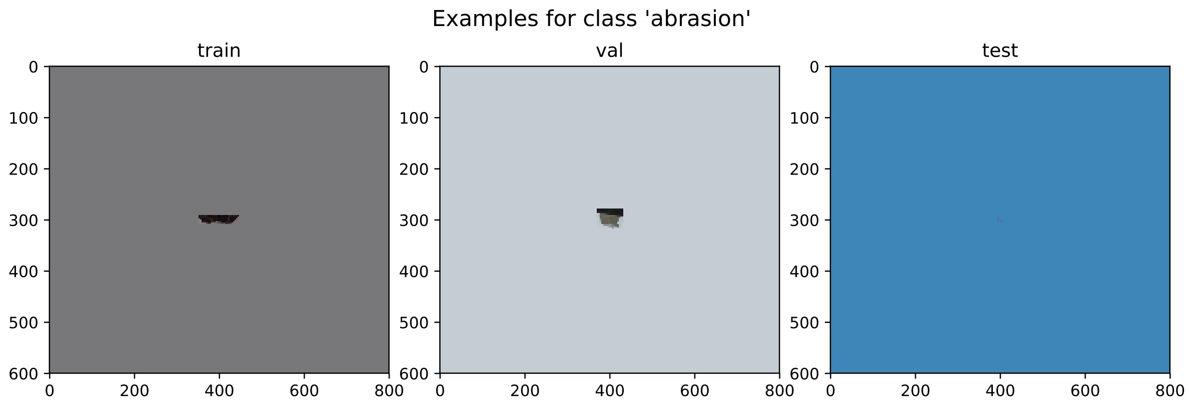


(b)

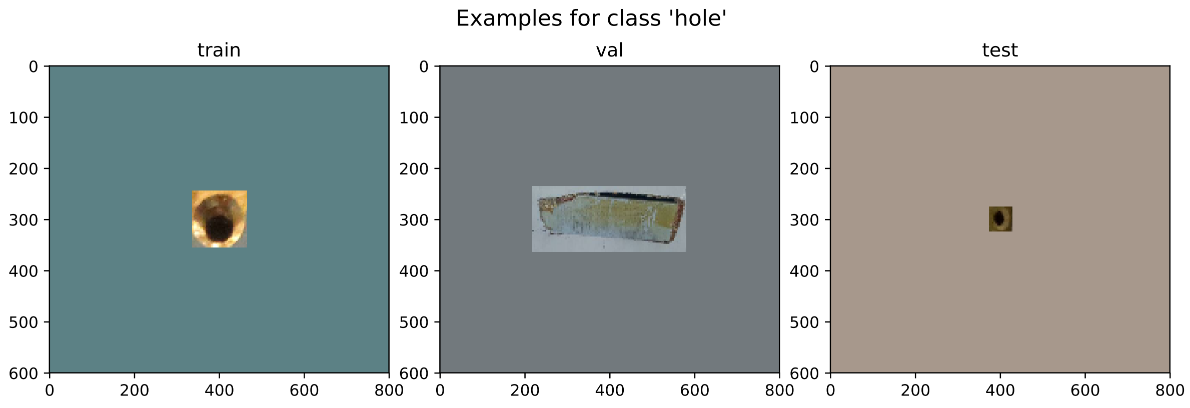


(c)

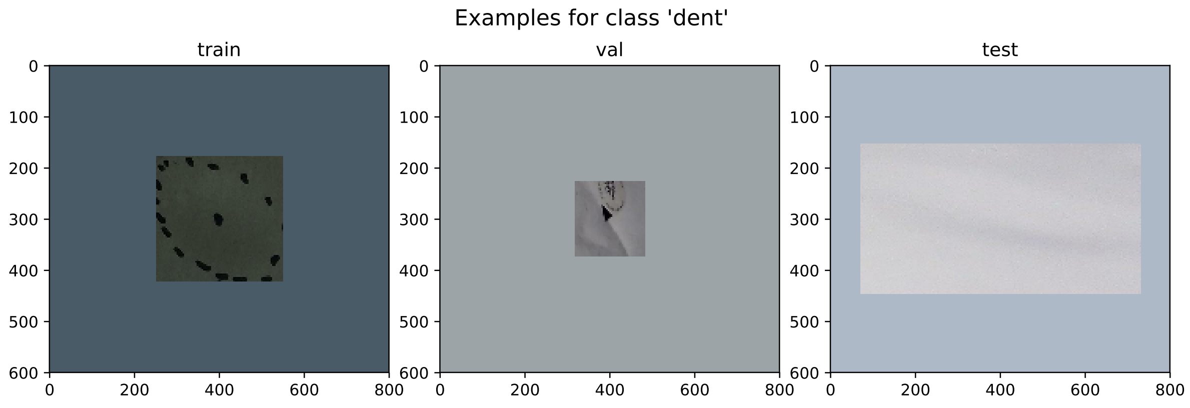
Figure A-6: 600×800 pixel examples of 'crack', 'melting point' and 'scratch'.



(a)

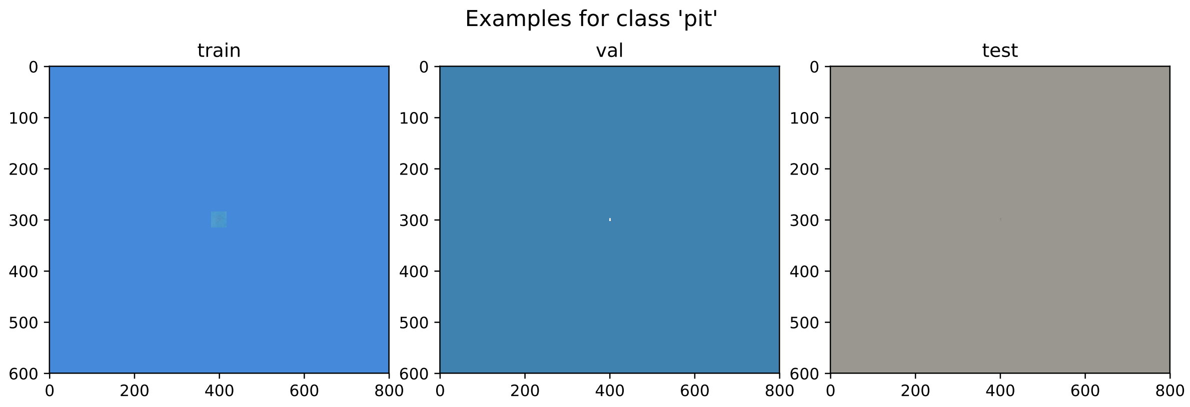


(b)

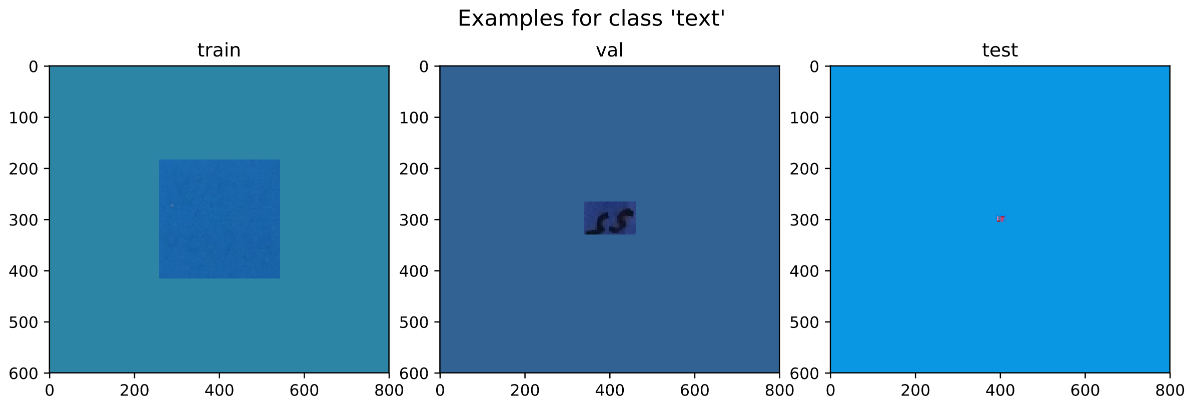


(c)

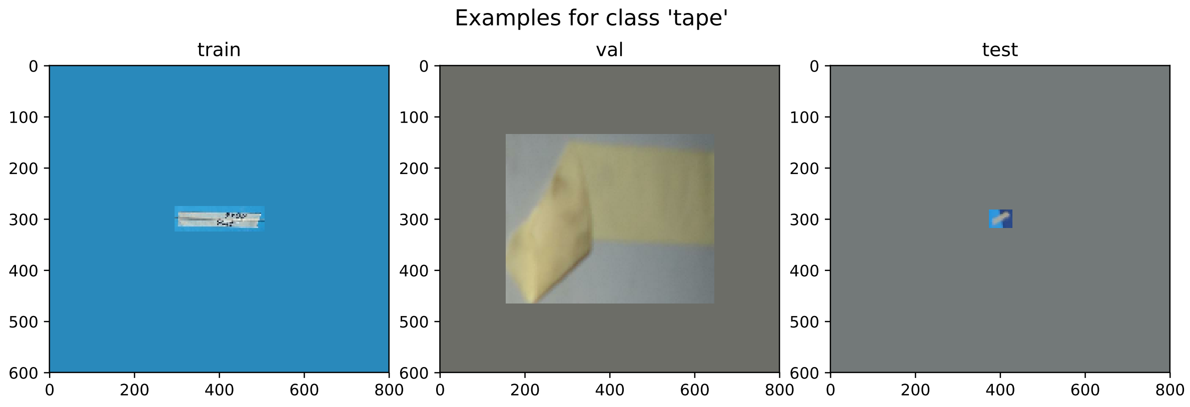
Figure A-7: 600×800 pixel examples of 'abrasion', 'hole' and 'dent'.



(a)

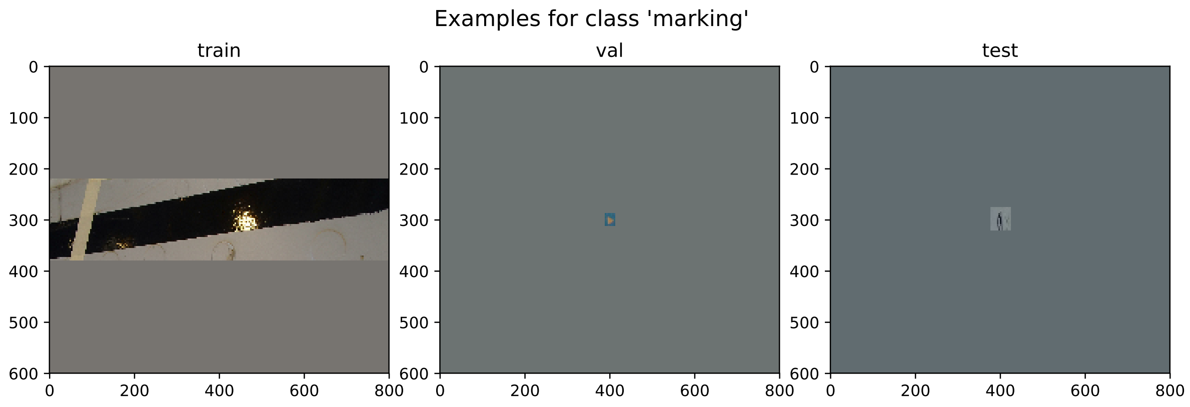


(b)

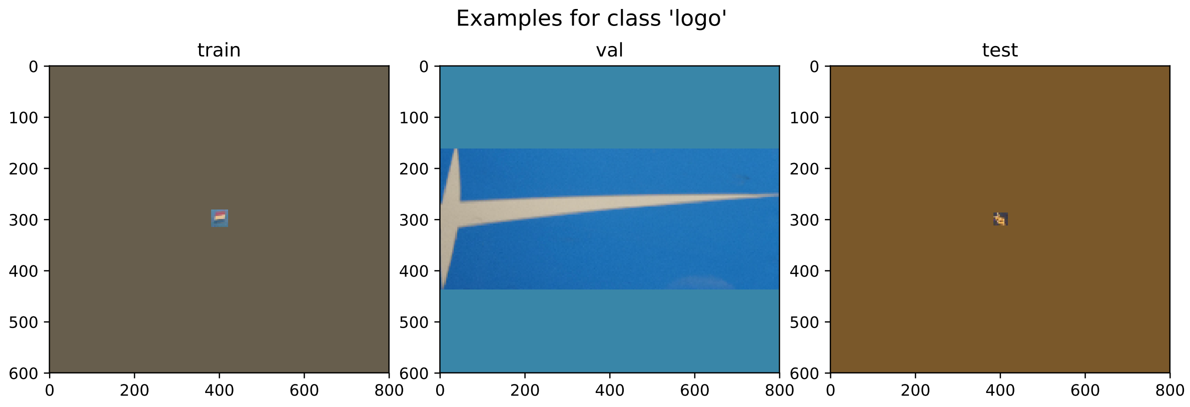


(c)

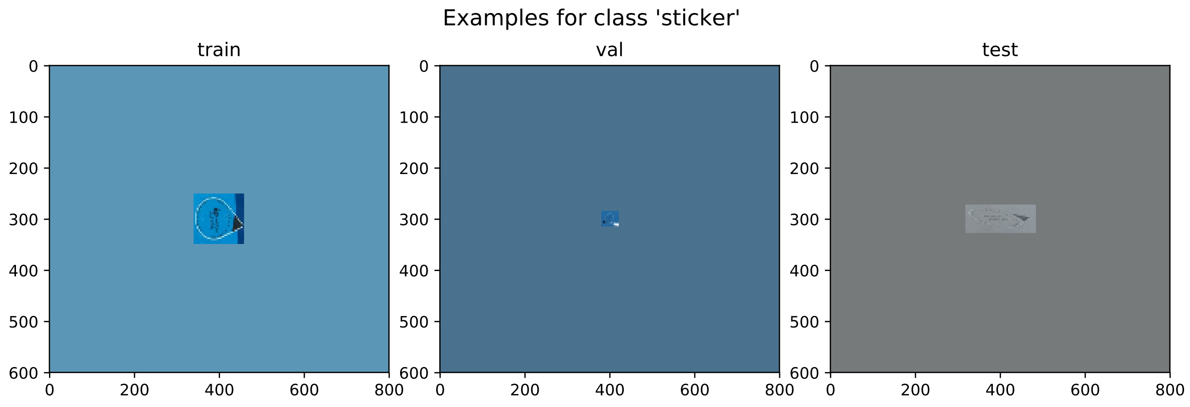
Figure A-8: 600×800 pixel examples of ‘pit’, ‘text’ and ‘tape’.



(a)

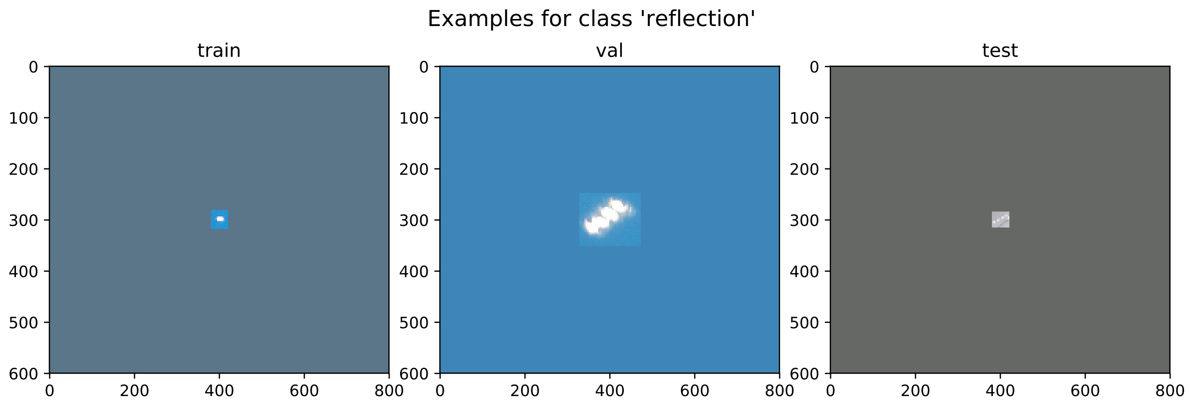


(b)

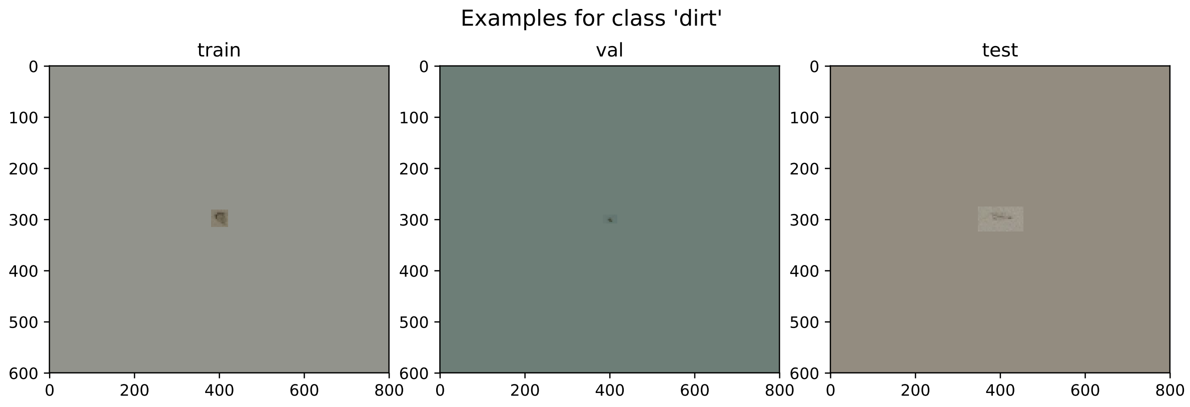


(c)

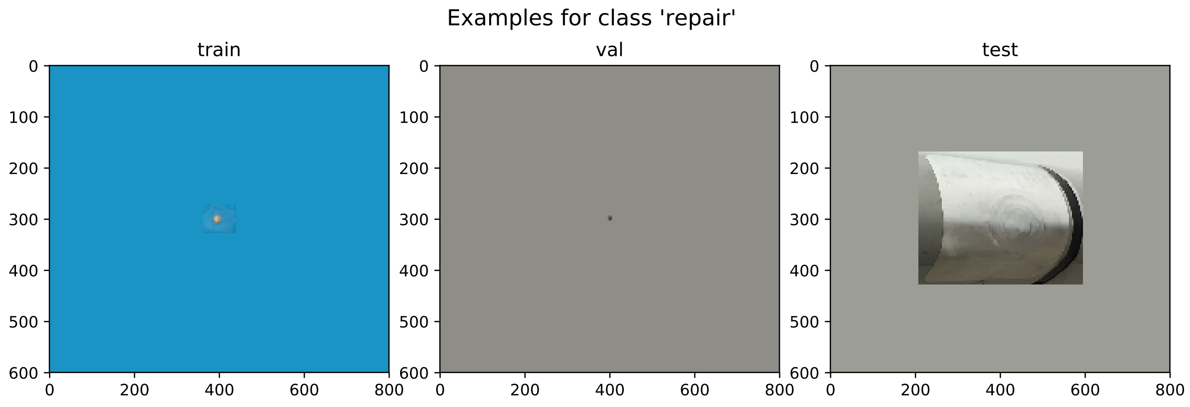
Figure A-9: 600×800 pixel examples of 'marking', 'logo' and 'sticker'.



(a)

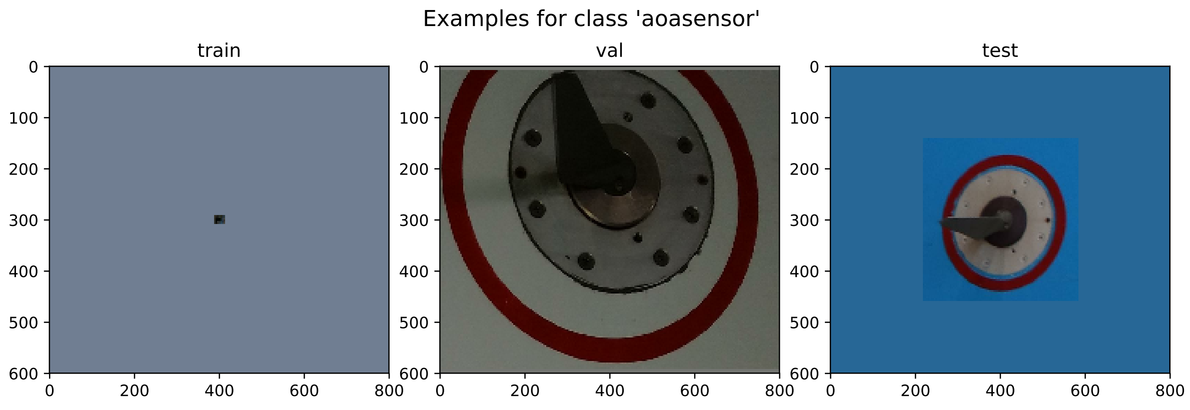


(b)

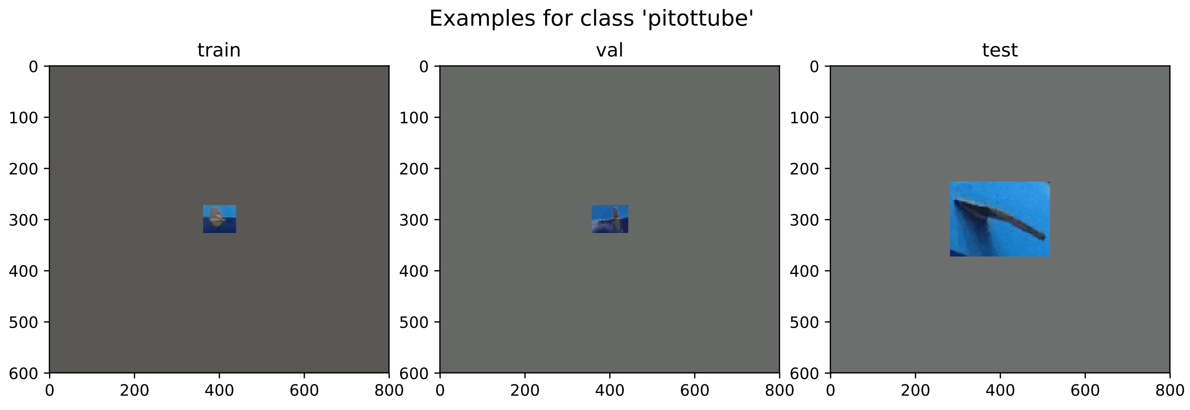


(c)

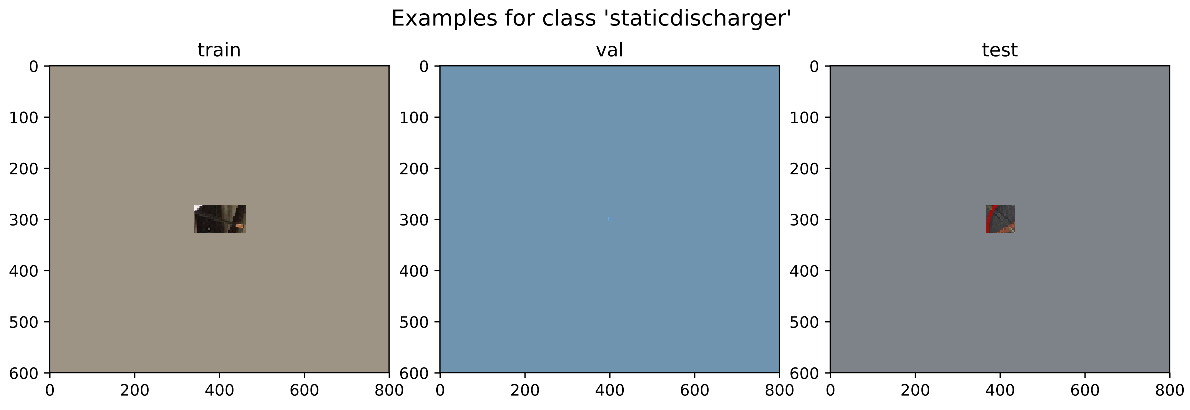
Figure A-10: 600×800 pixel examples of 'reflection', 'dirt' and 'repair'.



(a)



(b)



(c)

Figure A-11: 600×800 pixel examples of 'AoA sensor', 'pitot tube', and 'static discharger'.

Appendix B

Learning Rate Range Test

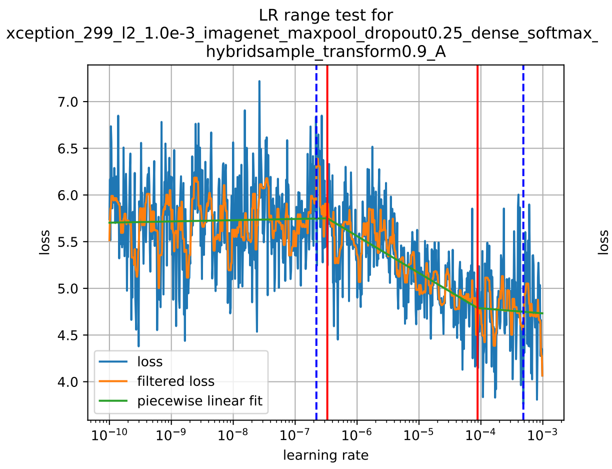
This chapter describes the method used for the learning rate range test as proposed by [79], [84] to avoid manual search for optimal learning rates.

[79] propose a cyclical learning rate schedule, where the learning rate cycles between a *base learning rate* and *max learning rate* in a triangular pattern. Instead of finding the exact value of an optimal learning rate, one can now specify a range that contains the optimal value and let the learning rate cycle around that value. The periodic ramping up and down of the learning rate additionally has a regularizing effect as it helps overcoming local minima. This policy - and an extreme version of it, the one-cycle-policy - has lead to fast training times with competitive results on the ImageNet benchmark [85], [86].

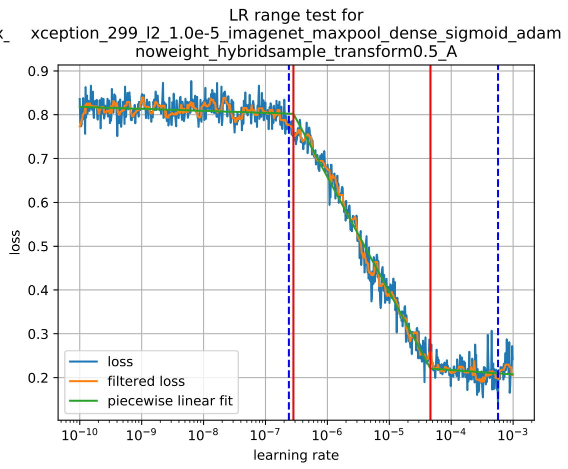
To find the base and max rates, [79] conduct a learning rate range test (LR-range test) by increasing the learning rate over one epoch from a sufficiently small initial learning rate to a high learning rate. They record the resulting loss (or accuracy). Initially the loss will not improve as the learning rate is too small. At a certain point, the loss will start to drop until the learning rate gets too high, the loss plateaus and then becomes unstable. The point where the loss is at its minimum is selected as the *max learning rate*, the point where the loss starts to decrease is the *base learning rate*. In between those points the rate-of-change for the loss is the highest, and therefore, the optimal learning rate lies somewhere in that interval.

In this work, finding the base and max rates is automatized by performing a LR-range test by exponentially increasing the learning rate over one epoch and fitting a piece-wise linear curve on the LR-loss plot. As the loss can be noisy, the loss is filtered before fitting the curve with a simple median filter with filter size 9 (i.e. the median loss of 9 batch updates is taken). The piece-wise linear curve has three segments. Figure B-1 shows an example of learning rate range test.

The parameters are found by performing a non-linear least squares optimization with soft L_1-loss to mitigate the effect of outliers in the loss function. The initial guesses for the parameters are as follows: The first segment is initialized with slope of 0 at a value of the first filtered loss, the first breaking point is initialized at half of the (exponential) learning rate of the minimum loss and the second breaking point is initialized at the minimum loss. The last segment again has slope 0.



(a) Softmax activation and categorical cross-entropy loss



(b) Sigmoid activation and binary cross-entropy loss

Figure B-1: Examples of learning rate range tests. First, the learning rate is increased exponentially for one epoch. The recorded loss is then filtered and curve-fitted with a piece-wise linear function. The blue dashed vertical lines indicate the initial guess, the red solid lines indicate the final guess for the base and maximum learning rate for the cyclical learning rate schedule used during training.

Appendix C

Detailed Experimental Results

C-1 Backbone Feature Extractor

C-1-1 Freezing the Convolutional Backbone

Table C-1: Training runs on frozen ResNet50 layers on DatasetV1.

Name	Epochs	Trainable params	Train Error	Val Error	Test Error	Balanced Test Error	Test FNR
top-layer only	15	43029	0.292	0.295	0.292	0.706	0.706
top-3 layers	27	1101845	0.368	0.374	0.385	0.741	0.545
conv5 block3	17	4514837	0.272	0.264	0.279	0.727	0.486
full network	54	23630741	0.006	0.067	0.067	0.214	0.137

C-1-2 Choosing a Convolutional Neural Network (CNN) Architecture

Table C-2: Training runs on different CNN architectures on DatasetV1.

Architecture	Epochs	Trainable params	Train Error	Val Error	Test Error	Balanced Test Error	Test FNR
DenseNet121	58	7059029	0.003	0.055	0.058	0.182	0.118
InceptionV3	62	21845813	0.017	0.078	0.079	0.236	0.124
ResNeXt50	54	23091157	0.009	0.071	0.068	0.226	0.131
ResNet50	54	23630741	0.006	0.067	0.067	0.214	0.137
Xception	30	20904509	0.014	0.060	0.061	0.208	0.094

C-2 Classification Head

C-2-1 Global Pooling Layer

Table C-3: Training runs on Xception with different global pooling methods on DatasetV2, re-sampling and $p_t = 0.5$.

Pooling Method	**Epochs**	**Train Error**	**Val Error**	**Test Error**	**Balanced Test Error**	**Test FNR**
average pooling	42	0.117	0.162	0.161	0.208	0.113
average+max pooling	70	0.089	0.141	0.140	0.194	0.085
max pooling	54	0.068	0.120	0.114	0.164	0.065

C-2-2 Classification Layer

Table C-4: Training runs on Xception with different activation functions on fully-connected (FC) classification layer on DatasetV2.

Activation	**Epochs**	**Train Error**	**Val Error**	**Test Error**	**Balanced Test Error**	**Test FNR**
softmax	37	0.006	0.067	0.080	0.272	0.161
sigmoid	30	0.020	0.079	0.084	0.289	0.176

Table C-5: Training runs Xception with different activation functions on convolutional classification layer.

Activation	**Epochs**	**Train Error**	**Val Error**	**Test Error**	**Balanced Test Error**	**Test FNR**
softmax	26	0.008	0.078	0.092	0.283	0.205
sigmoid	22	0.030	0.081	0.088	0.305	0.181

C-3 Re-sampling

Table C-6: Training runs Xception with re-sampling enabled.

Activation	**Epochs**	**Train Error**	**Val Error**	**Test Error**	**Balanced Test Error**	**Test FNR**
softmax	18	0.126	0.182	0.194	0.273	0.107
sigmoid	26	0.084	0.147	0.147	0.272	0.153

C-4 Regularization

C-4-1 Data Augmentation

Four experiments with data augmentation are done:

1. Xception & max-pooling & FC classification, softmax and sigmoid activation, $p_t = 0.5$, no re-sampling, on DatasetV2 (Table C-7, Figure C-1).
2. Xception & max-pooling & FC classification & softmax & categorical cross-entropy, varying p_t, re-sampled training data, on DatasetV1 (Table C-8, Figure C-2).
3. Xception & max-pooling & FC classification & softmax & categorical cross-entropy, varying p_t, re-sampled training data, on DatasetV2 (Table C-9).
4. Xception & max-pooling & FC classification & sigmoid & binary cross-entropy, varying p_t, re-sampled training data, on DatasetV2 (Table C-10).

Table C-7: Training runs with $p_t = 0.5$ (no re-sampling) on DatasetV2.

Activation	**Epochs**	**Train Error**	**Val Error**	**Test Error**	**Balanced Test Error**	**Test FNR**
softmax	54	0.009	0.056	0.068	0.243	0.126
sigmoid	82	0.020	0.065	0.062	0.212	0.129

To get a better understanding of how re-sampling and augmentation interact with each other, the experiments of Figure 4-4 are repeated with data augmentation enabled, with a fixed $p_t = 0.5$. The results are shown in Table C-7 and Figure C-1. For both single-label and multi-label configurations and all metrics, augmentation has a positive effect, but the more profound drop can be seen for multi-label classification. The difference in training and validation loss for the sigmoid activation has disappeared, which speaks for augmentation as regularizer.

Table C-8: Training runs on softmax with **re-sampling and augmentation** enabled on DatasetV1.

p_t	**Epochs**	**Train Error**	**Val Error**	**Test Error**	**Balanced Test Error**	**Test FNR**
0.0	74	0.007	0.070	0.073	0.192	0.126
0.1	51	0.034	0.084	0.090	0.178	0.109
0.25	51	0.047	0.091	0.097	0.176	0.111
0.5	38	0.100	0.134	0.143	0.167	0.063
0.75	46	0.108	0.137	0.143	0.168	0.070
0.9	50	0.122	0.144	0.152	0.151	0.078

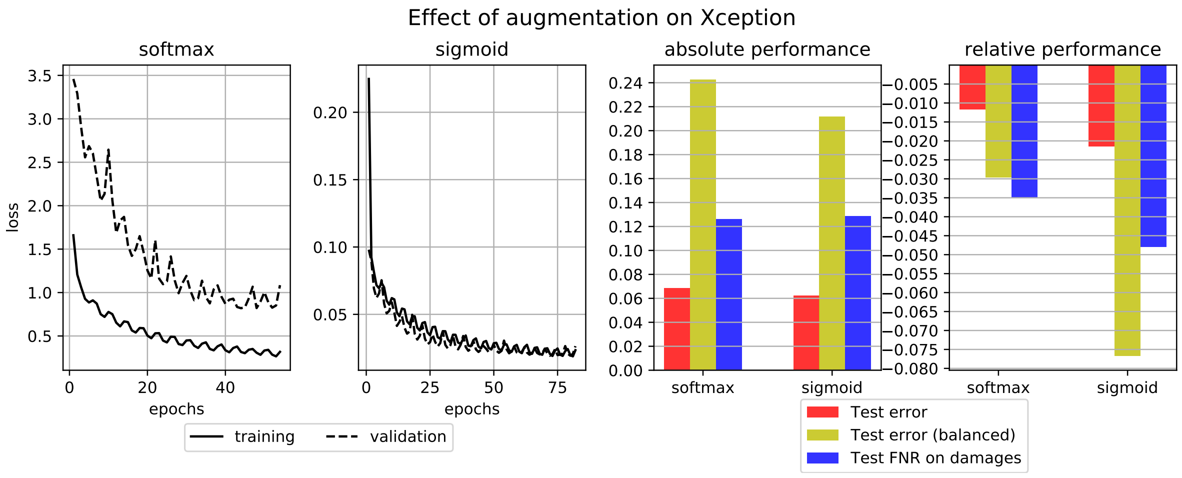


Figure C-1: Effect of augmentation on the two activation functions. The two plots on the left show the cross-entropy losses for softmax and sigmoid activation after each epoch, respectively. The two plots on the right show the absolute metrics and relative difference to the baseline experiments in Figure 4-4. The runs are trained and tested on DatasetV2, with $p_t = 0.5$ and no re-sampling.

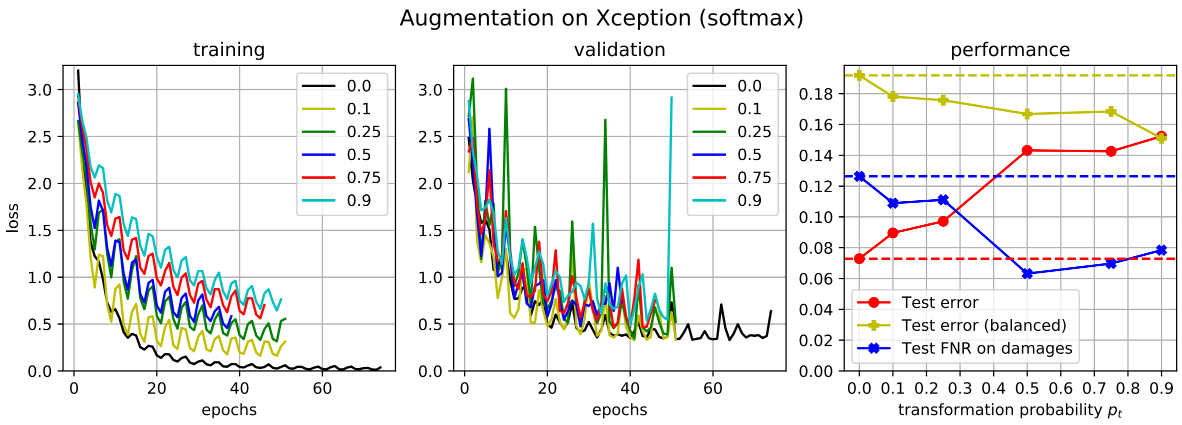


Figure C-2: Training with different transformation rates on Xception with softmax head on DatasetV1. Experiments are run with transformation rates of 0.1, 0.25, 0.5, 0.75 and 0.9, respectively. The training set re-sampled to a uniform distribution.

Table C-9: Training runs on Xception with softmax with re-sampling and augmentation enabled on DatasetV2.

p_t	Epochs	Train Error	Val Error	Test Error	Balanced Test Error	Test FNR
0.0	18	0.126	0.182	0.194	0.273	0.107
0.1	30	0.064	0.132	0.138	0.215	0.105
0.25	34	0.100	0.146	0.160	0.205	0.087
0.5	26	0.143	0.182	0.187	0.201	0.096
0.75	42	0.110	0.150	0.148	0.167	0.070
0.9	74	0.135	0.182	0.174	0.180	0.068

Table C-10: Training runs on Xception with sigmoid with re-sampling and augmentation enabled on DatasetV2.

p_t	**Epochs**	**Train Error**	**Val Error**	**Test Error**	**Balanced Test Error**	**Test FNR**
0.0	26	0.084	0.147	0.147	0.272	0.153
0.1	26	0.101	0.169	0.172	0.296	0.153
0.25	40	0.079	0.150	0.155	0.281	0.146
0.5	54	0.068	0.120	0.114	0.164	0.065
0.75	59	0.070	0.132	0.137	0.207	0.118
0.9	66	0.068	0.131	0.128	0.209	0.109

C-4-2 Dropout

Two experiments with dropout are done:

1. Xception & max-pooling & FC classification & softmax & categorical cross-entropy, varying p_{drop}, re-sampled training data, on DatasetV2 (Table C-11).
2. Xception & max-pooling & FC classification & sigmoid & binary cross-entropy, varying p_{drop}, re-sampled training data, on DatasetV2 (Table C-12).

Table C-11: Training runs on Xception with softmax with re-sampling and dropout enabled on DatasetV2.

p_{drop}	**Epochs**	**Train Error**	**Val Error**	**Test Error**	**Balanced Test Error**	**Test FNR**
0.0	18	0.126	0.182	0.194	0.273	0.107
0.1	22	0.175	0.212	0.218	0.257	0.100
0.25	22	0.152	0.199	0.206	0.245	0.078
0.5	31	0.201	0.248	0.246	0.291	0.118
0.75	27	0.194	0.230	0.232	0.286	0.126
0.9	38	0.332	0.343	0.361	0.390	0.133

Table C-12: Training runs on Xception with sigmoid with re-sampling and dropout enabled on DatasetV2.

p_{drop}	**Epochs**	**Train Error**	**Val Error**	**Test Error**	**Balanced Test Error**	**Test FNR**
0.0	26	0.084	0.147	0.147	0.272	0.153
0.1	22	0.155	0.210	0.221	0.308	0.168
0.25	22	0.188	0.246	0.251	0.330	0.155
0.5	26	0.161	0.212	0.216	0.285	0.150
0.75	26	0.236	0.267	0.282	0.311	0.176
0.9	30	0.134	0.183	0.185	0.274	0.194

C-4-3 L_2 Weight Regularization

Three experiments with L_2 weight regularization are done:

1. Xception & max-pooling & FC classification & softmax & categorical cross-entropy, varying λ, re-sampled training data, on DatasetV2 (Table C-13).
2. Xception & max-pooling & FC classification & sigmoid & binary cross-entropy, varying λ, re-sampled training data, on DatasetV2 (Table C-14).
3. Xception & max-pooling & FC classification & sigmoid & binary cross-entropy, L_2-regularization only applied to final layer parameters, varying λ, re-sampled training data, on DatasetV2 (Table C-15, Figure C-3).

Table C-13: Training runs on Xception with softmax with re-sampling and L_2-regularization enabled on DatasetV2.

λ	Epochs	Train Error	Val Error	Test Error	Balanced Test Error	Test FNR
10^{-7}	18	0.222	0.269	0.271	0.302	0.118
10^{-6}	18	0.212	0.260	0.274	0.286	0.111
10^{-5}	14	0.316	0.352	0.352	0.350	0.105
10^{-4}	22	0.200	0.249	0.251	0.294	0.102
10^{-3}	45	0.063	0.137	0.145	0.270	0.094

Table C-14: Training runs on Xception with sigmoid with re-sampling and L_2-regularization enabled on DatasetV2.

λ	Epochs	Train Error	Val Error	Test Error	Balanced Test Error	Test FNR
10^{-7}	21	0.147	0.205	0.214	0.314	0.168
10^{-6}	23	0.127	0.192	0.198	0.293	0.135
10^{-5}	22	0.094	0.156	0.159	0.281	0.120
10^{-4}	82	0.024	0.087	0.097	0.291	0.166
10^{-3}	34	0.086	0.132	0.153	0.275	0.144

Table C-15: Training runs on Xception with sigmoid with re-sampling and L_2-regularization on the last classification layer enabled on DatasetV2.

λ	Epochs	Train Error	Val Error	Test Error	Balanced Test Error	Test FNR
10^{-4}	23	0.123	0.181	0.187	0.308	0.153
10^{-3}	42	0.033	0.102	0.103	0.294	0.187
10^{-2}	43	0.087	0.155	0.163	0.288	0.205
10^{-1}	74	0.027	0.087	0.090	0.253	0.159
10^{0}	74	0.028	0.085	0.093	0.263	0.181
10^{1}	90	0.031	0.092	0.098	0.264	0.174

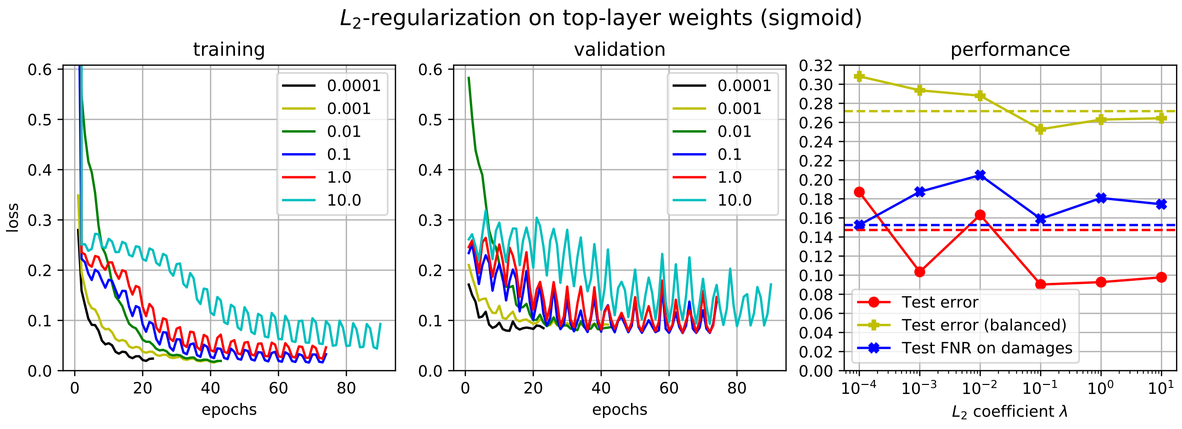


Figure C-3: Training with different strengths of L_2-regularization on the classification layer parameters (weights and biases) with sigmoid activation on DatasetV2. Left: training loss after each epoch. Middle: validation loss after each epoch. Right: performance metrics on the best checkpoints as a function of λ. The training set is re-sampled to a uniform distribution. Experiments were run with values for λ of 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0} and 10^{1}, respectively.

C-5 Combining the Regularization Techniques

Finally, two experiments are done combining all the previous findings:

1. Xception & max-pooling & FC classification & softmax & categorical cross-entropy, $\lambda = 10^{-3}$, $p_t = 0.9$ and $p_{\text{drop}} = 0.25$, re-sampled training data, on DatasetV2, repeated three times (Table C-16).
2. Xception & max-pooling & FC classification & sigmoid & binary cross-entropy, $\lambda = 10^{-5}$, $p_t = 0.5$ and $p_{\text{drop}} = 0.0$, re-sampled training data, on DatasetV2, repeated five times (Table C-17).

Table C-16: Training runs on Xception with softmax with combined regularization enabled on DatasetV2.

Name	Epochs	Train Error	Val Error	Test Error	Balanced Test Error	Test FNR
A	98	0.078	0.118	0.140	0.141	0.096
B	106	0.076	0.119	0.123	0.138	0.081
C	159	0.095	0.147	0.147	0.167	0.098
D	106	0.078	0.121	0.125	0.139	0.054

Table C-17: Training runs on Xception with sigmoid with combined regularization enabled on DatasetV2.

Name	Epochs	Train Error	Val Error	Test Error	Balanced Test Error	Test FNR
A	70	0.076	0.138	0.135	0.186	0.085
B	58	0.096	0.152	0.148	0.208	0.081
C	86	0.051	0.114	0.103	0.167	0.072
D	62	0.097	0.156	0.148	0.186	0.085
E	79	0.077	0.139	0.138	0.190	0.083

Table C-18: Means and 95% confidence intervals for the training runs with combined regularization on DatasetV2 (train and val).

Name	Runs	Train Error	Val Error
softmax	4	0.082 ± 0.007	0.126 ± 0.011
sigmoid	5	0.079 ± 0.010	0.140 ± 0.009
t-statistic	-	0.281	-1.320
p-value	-	0.788	0.229

Table C-19: Means and 95% confidence intervals for the training runs with combined regularization on DatasetV2 (test).

Name	Runs	Test Error	Balanced Test Error	Test FNR
softmax	4	0.134 ± 0.009	0.146 ± 0.011	0.082 ± 0.016
sigmoid	5	0.134 ± 0.010	0.187 ± 0.008	0.081 ± 0.003
t-statistic	-	-0.070	-4.286	0.116
p-value	-	0.947	0.004	0.914

Appendix D

Predicted Probabilities

Figures D-1 to D-7 show the output probabilities for damage examples from the test set.

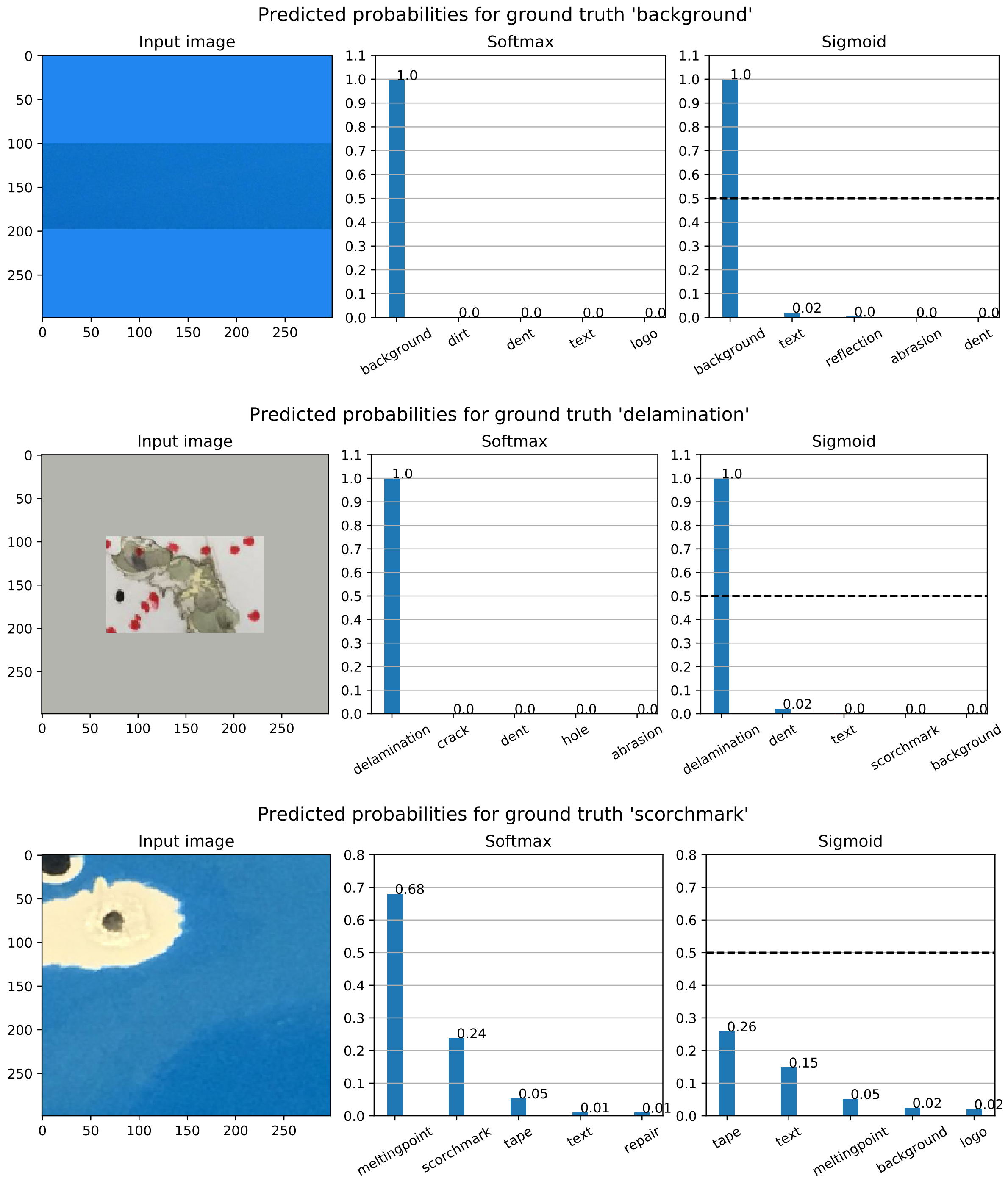


Figure D-1: Predicted probabilities on 299×299 pixel crops of ‘background’, ‘delamination’ and ‘scorch mark’.

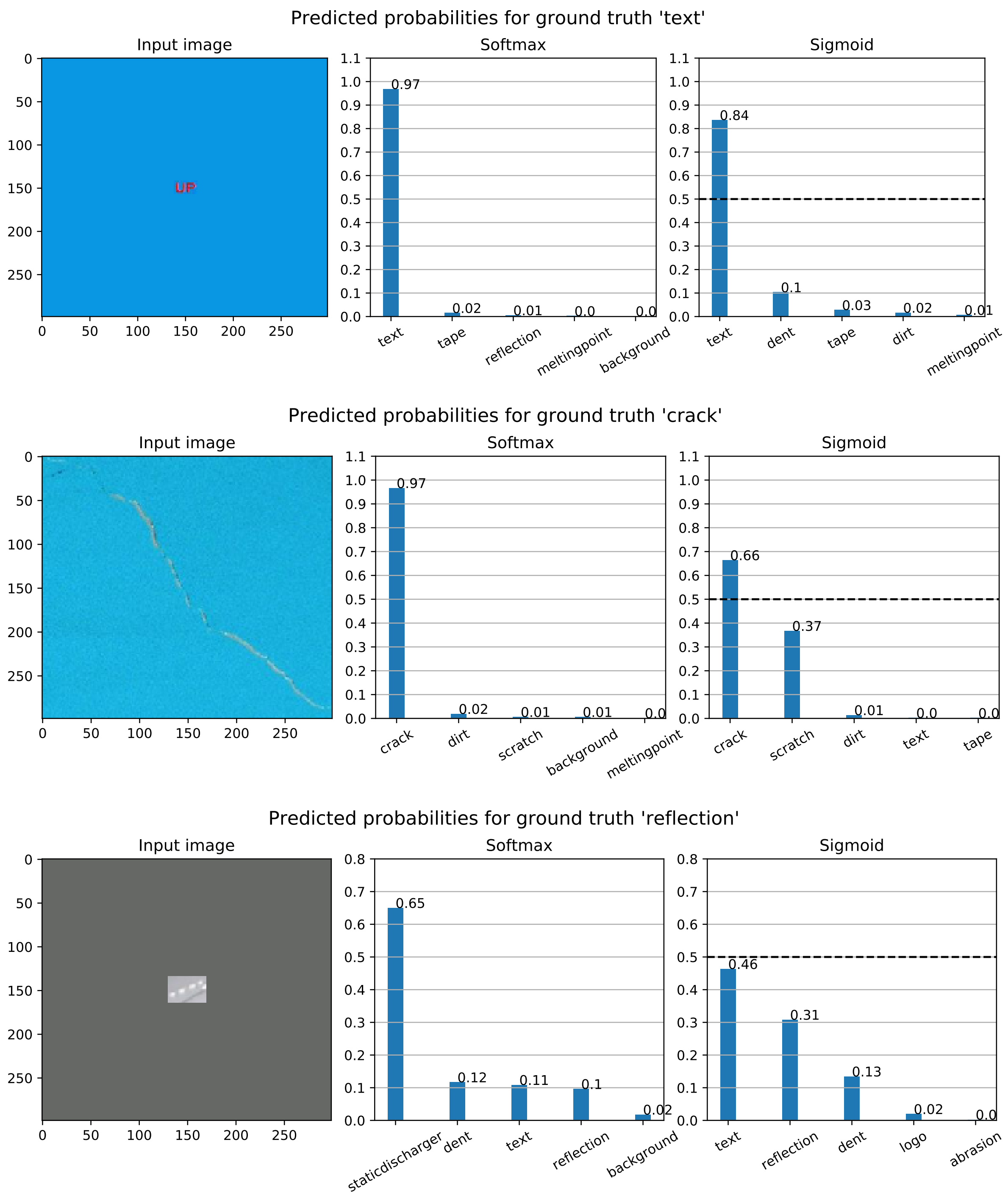


Figure D-2: Predicted probabilities on 299×299 pixel crops of 'text', 'crack' and 'reflection'.

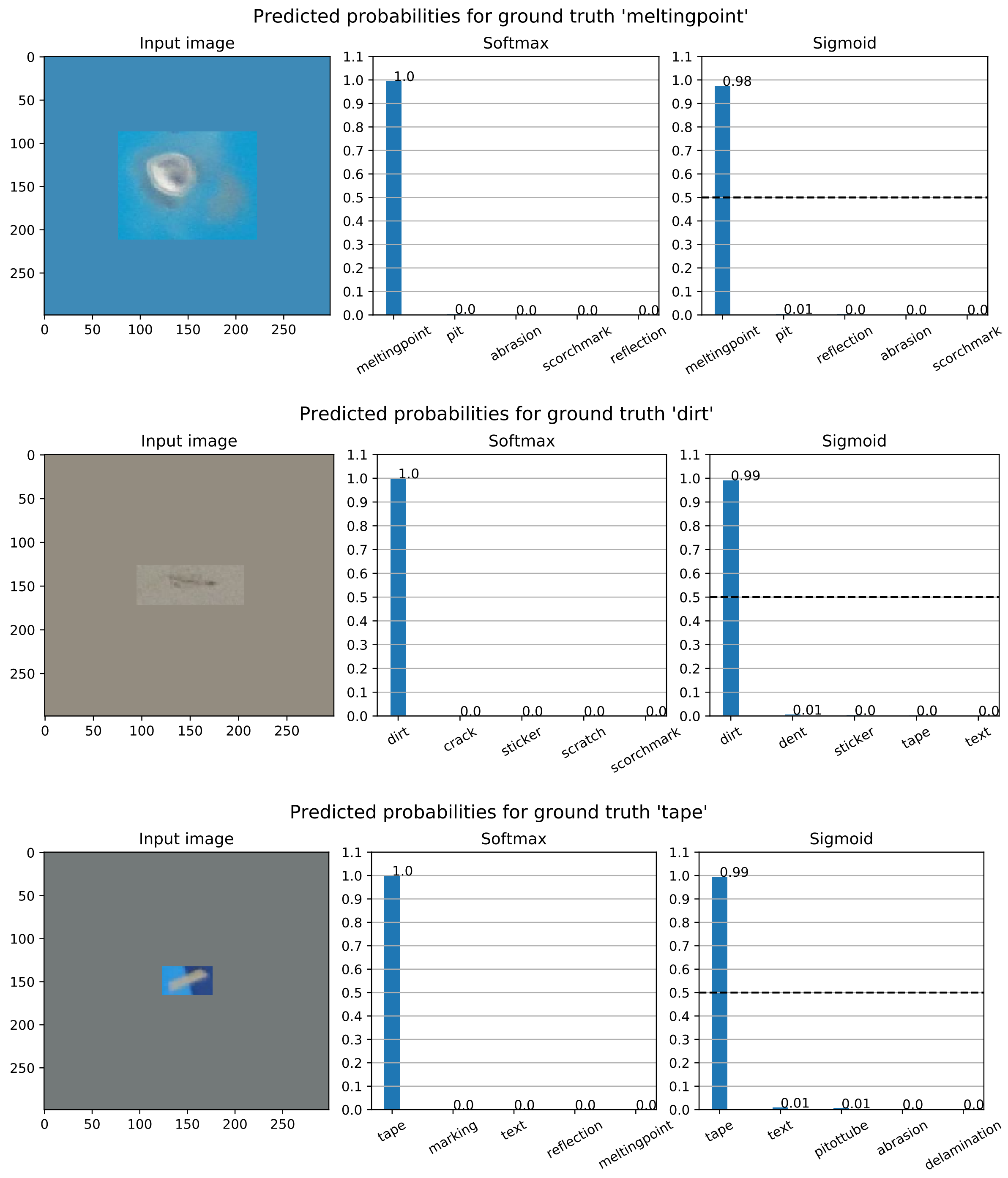


Figure D-3: Predicted probabilities on 299×299 pixel crops of 'melting point', 'dirt' and 'tape'.

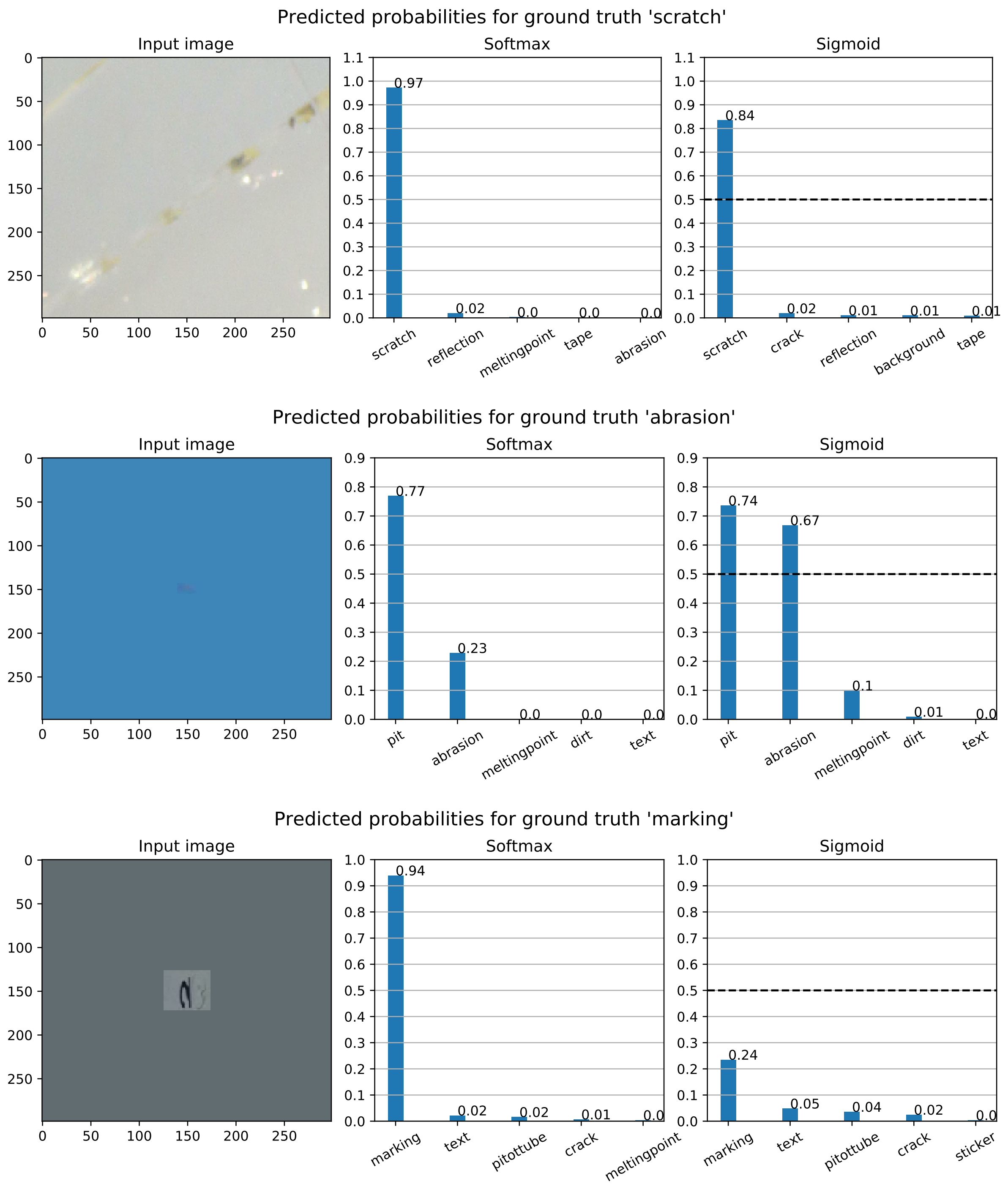


Figure D-4: Predicted probabilities on 299×299 pixel crops of 'scratch', 'abrasion' and 'marking'.

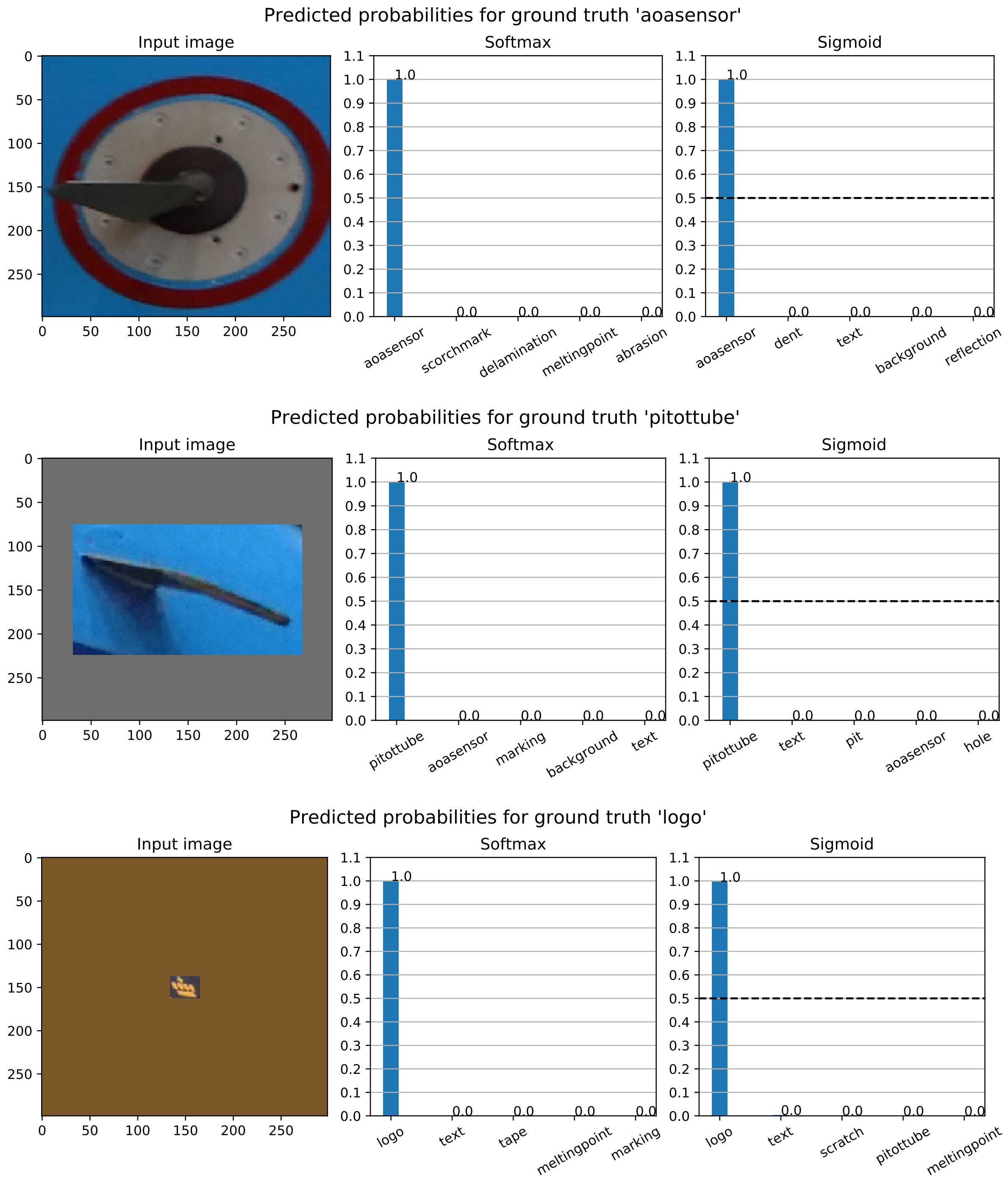


Figure D-5: Predicted probabilities on 299×299 pixel crops of 'AoA sensor', 'pitot tube' and 'logo'.

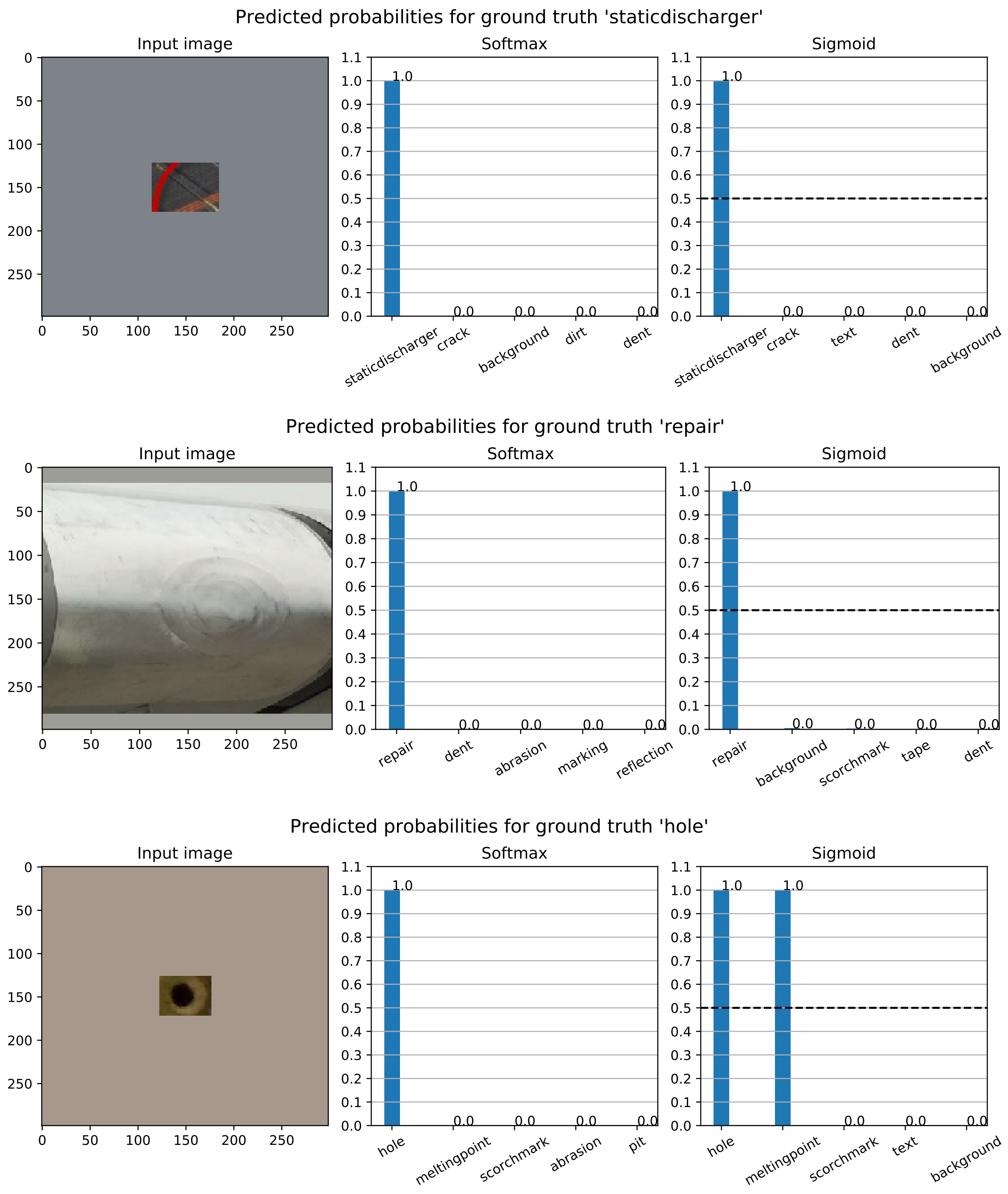
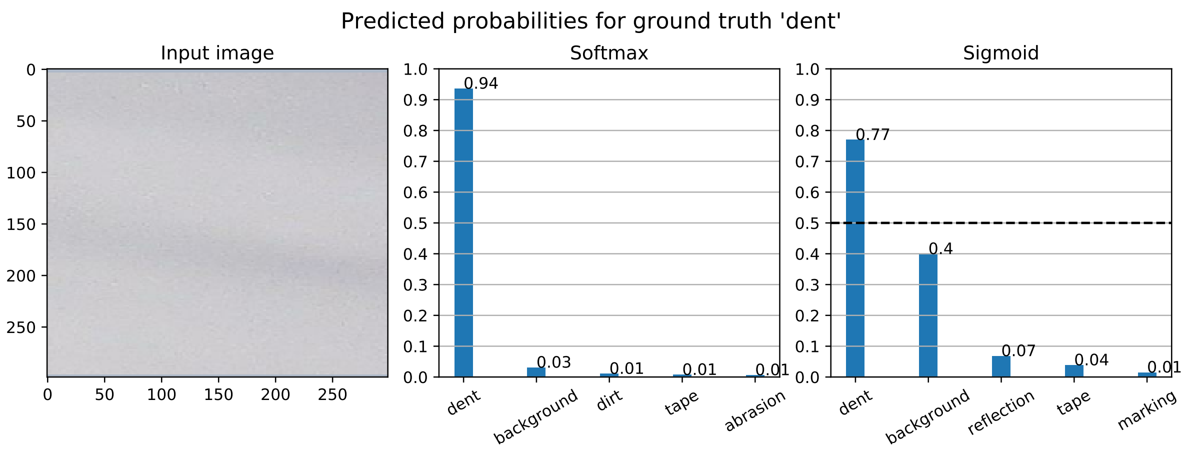
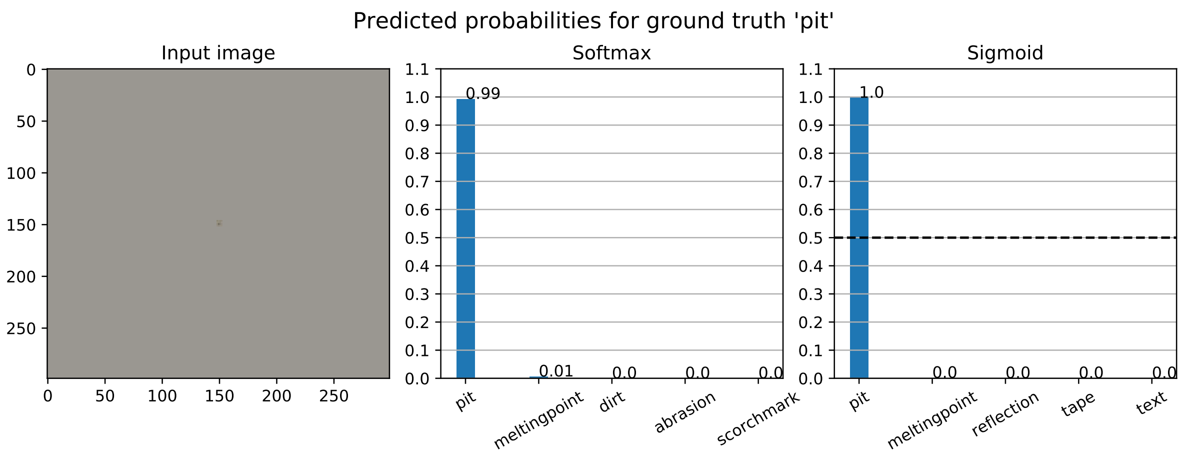


Figure D-6: Predicted probabilities on 299×299 pixel crops of 'static discharger', 'repair' and 'hole'.





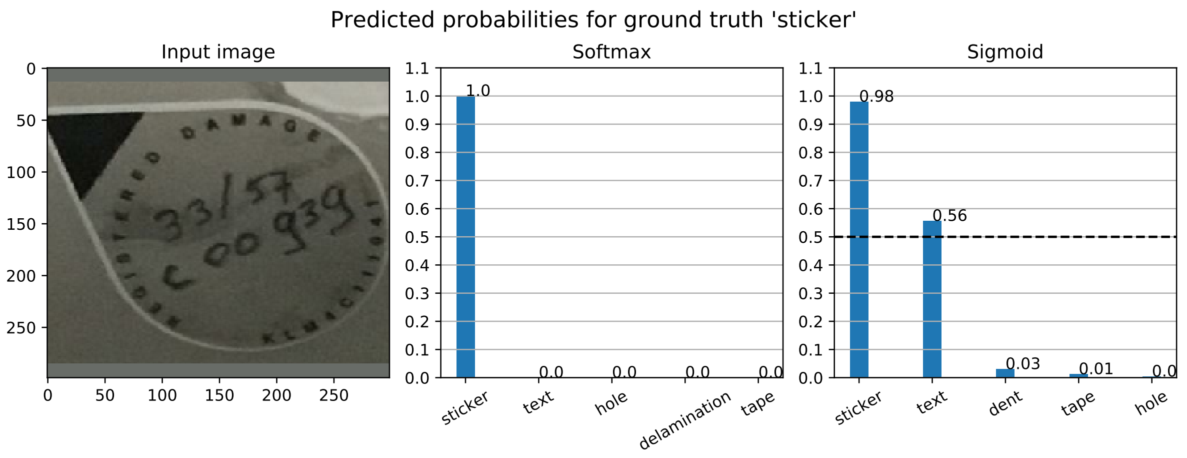


Figure D-7: Predicted probabilities on 299×299 pixel crops of 'dent', 'pit' and 'sticker'.

Appendix E

Class Probability Maps

Figures E-1 to E-3 show the probability maps for damage examples from the test set as obtained by converting an Xception classifier to a sliding-window classifier with a 3×3 max-pooling layer. The images are kept at their 600×800 pixel size and are randomly rotated and translated to visualize the outputs of the sliding-window classifier.

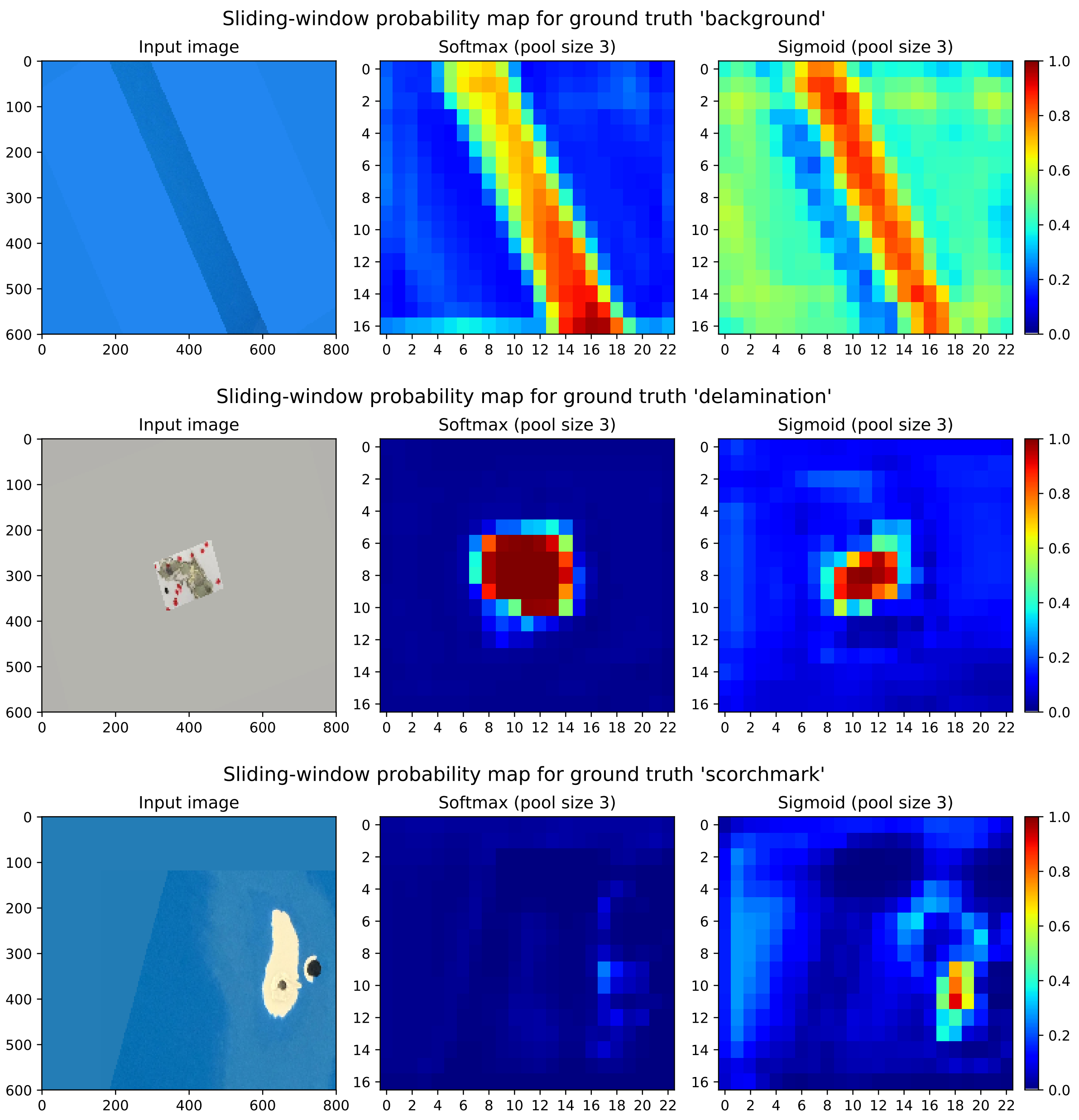


Figure E-1: Class probability maps on 600×800 pixel examples of 'background', 'delamination' and 'scorch mark' using a sliding-window classifier with 3×3 max-pooling window. Images are randomly rotated and translated.

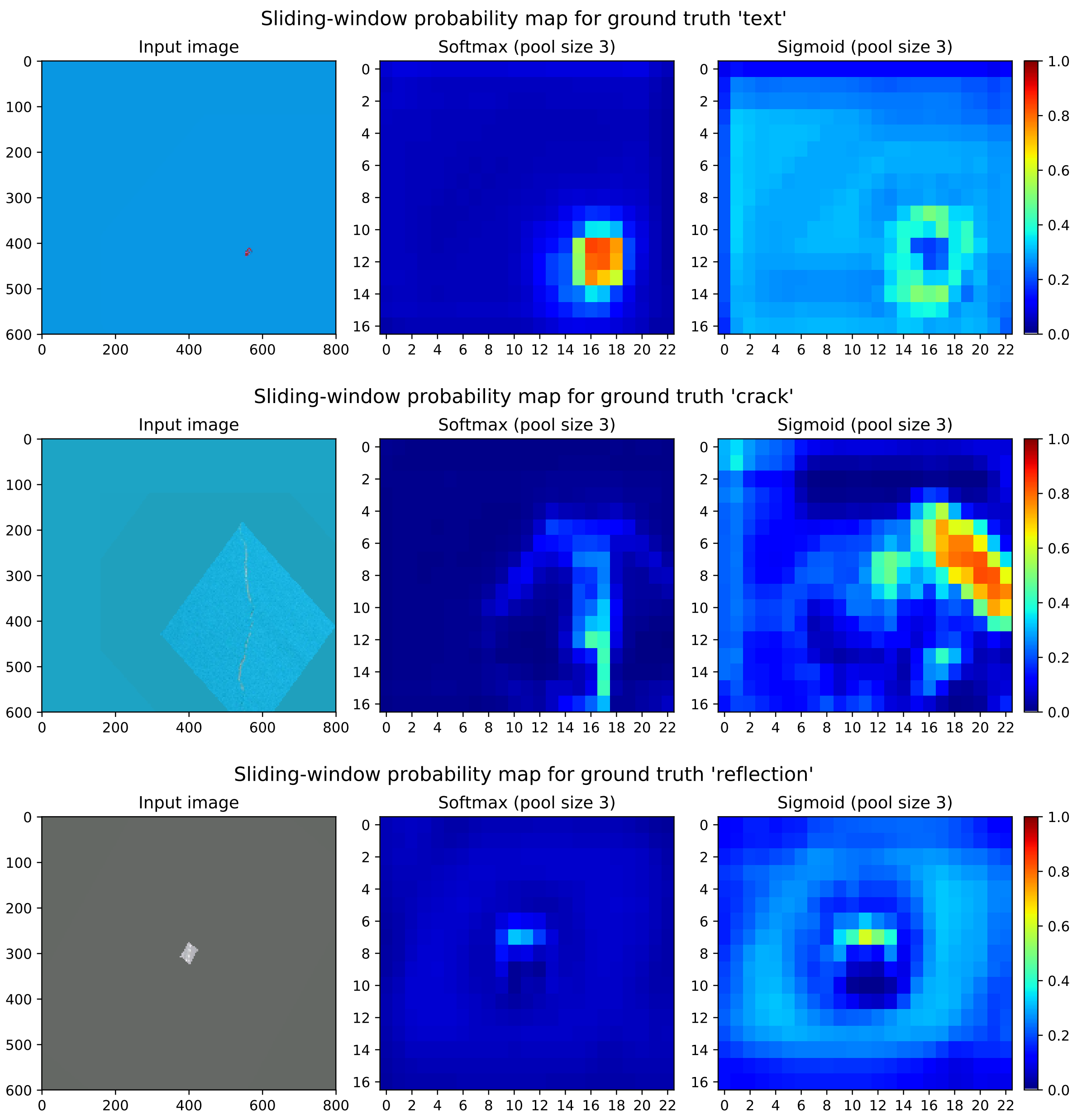


Figure E-2: Class probability maps on 600×800 pixel examples of 'text', 'crack' and 'reflection' using a sliding-window classifier with 3×3 max-pooling window. Images are randomly rotated and translated.

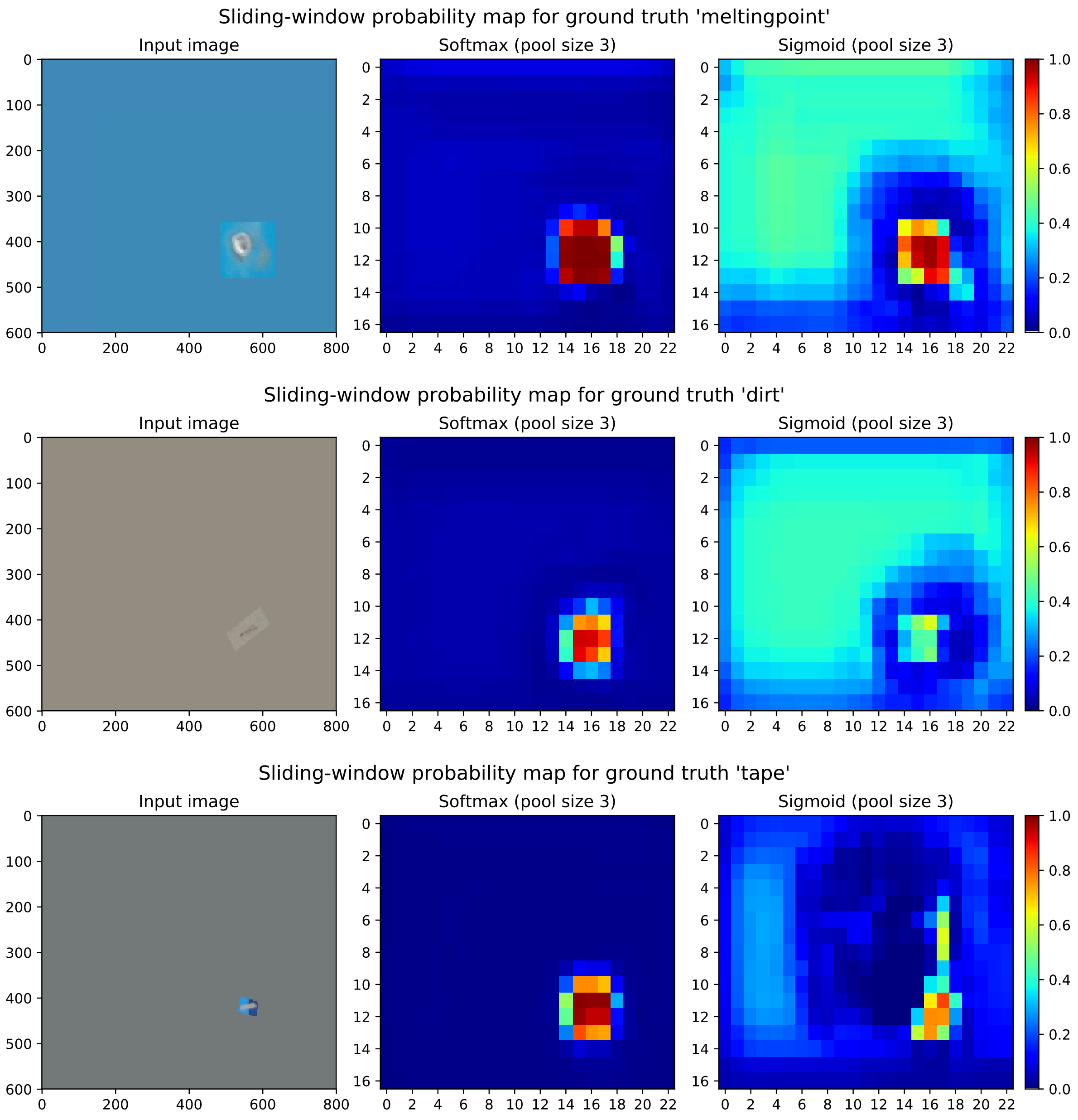


Figure E-3: Class probability maps on 600×800 pixel examples of 'melting point', 'dirt', and 'tape' using a sliding-window classifier with 3×3 max-pooling window. Images are randomly rotated and translated.

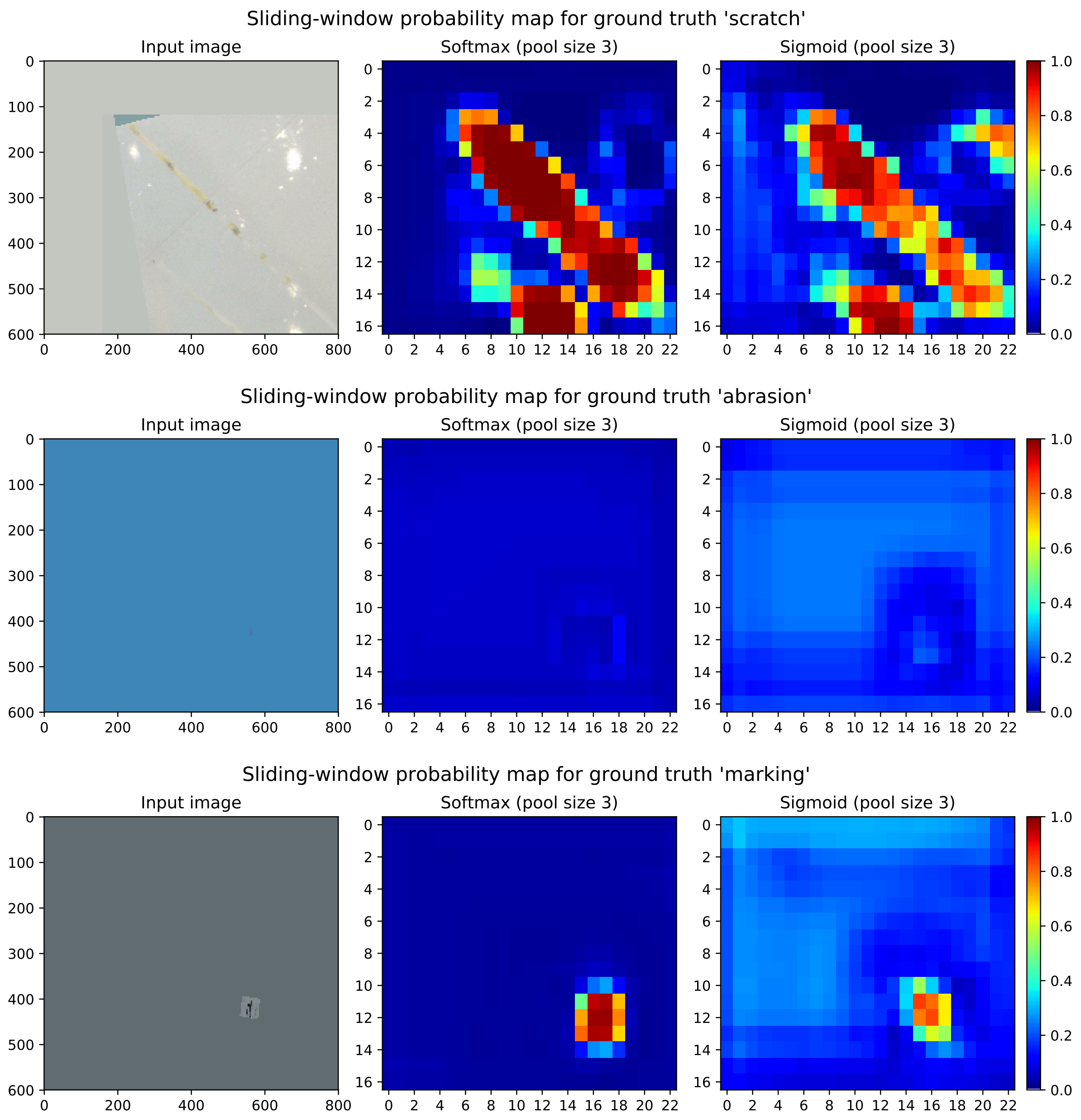


Figure E-4: Class probability maps on 600×800 pixel examples of 'melting point', 'pit', and 'scorch mark' using a sliding-window classifier with 3×3 max-pooling window. Images are randomly rotated and translated.

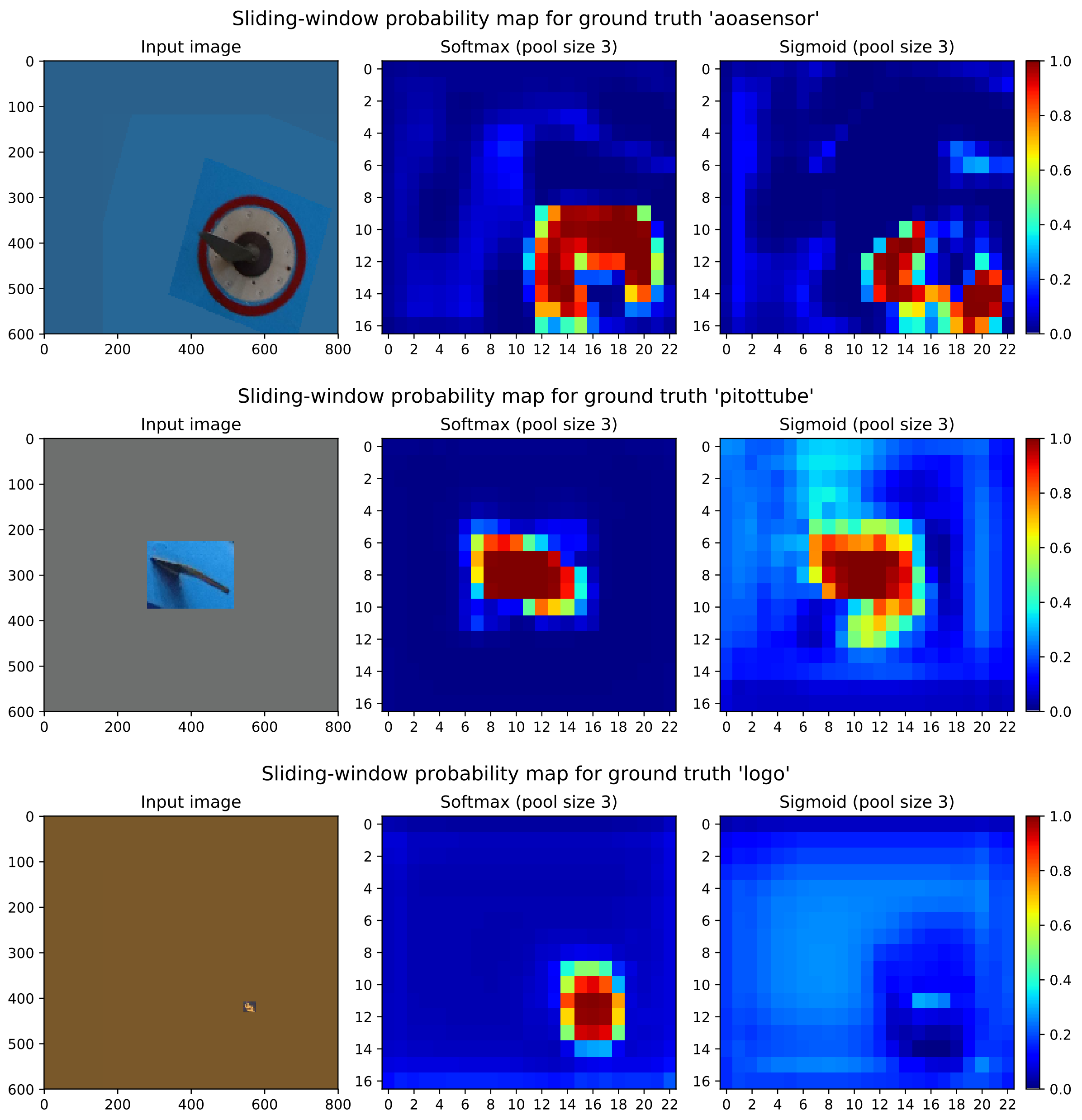


Figure E-5: Class probability maps on 600×800 pixel examples of ‘AoA sensor’, ‘pitot tube’, and ‘logo’ using a sliding-window classifier with 3×3 max-pooling window. Images are randomly rotated and translated.

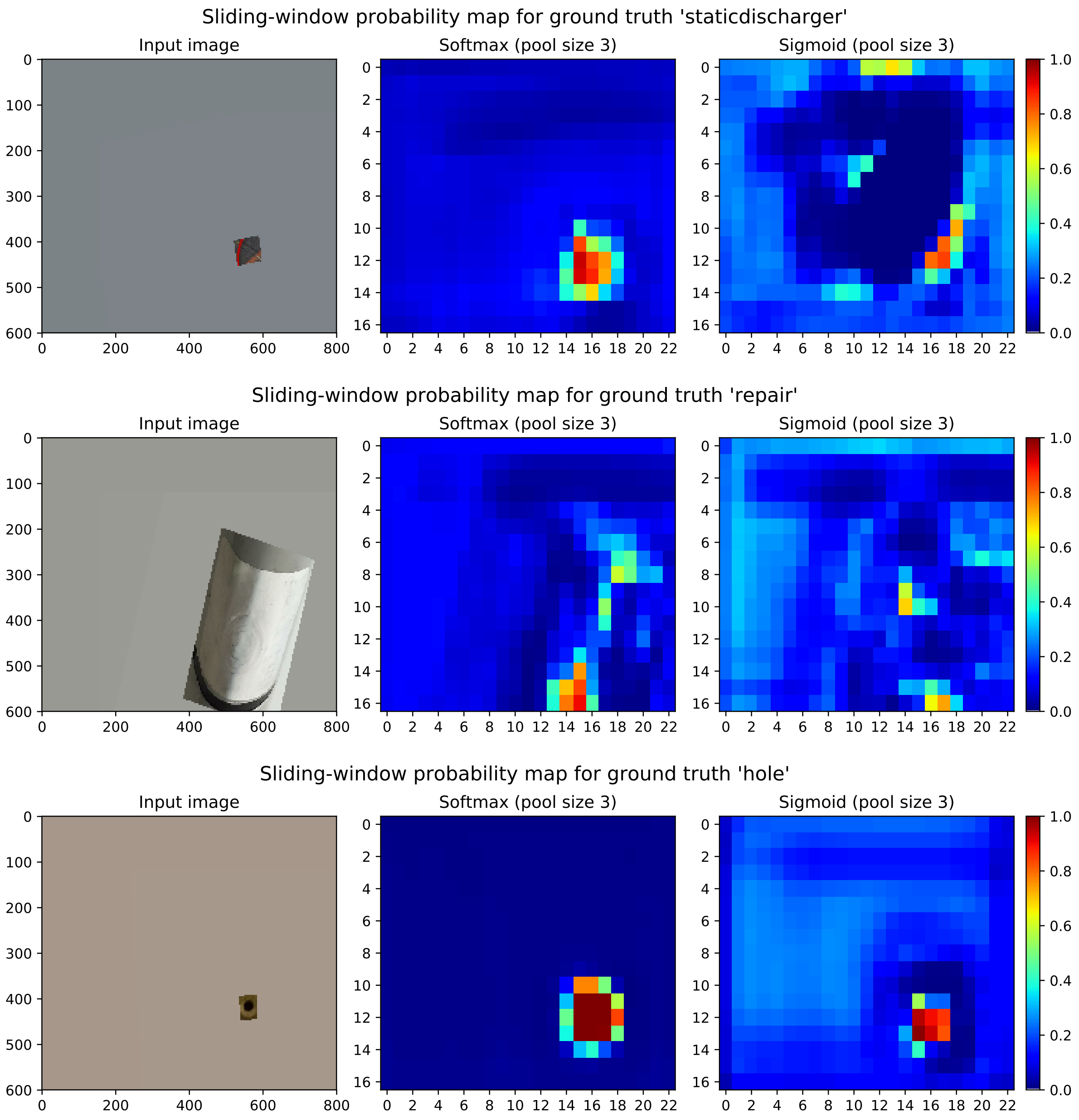


Figure E-6: Class probability maps on 600×800 pixel examples of 'static discharger', 'repair', and 'hole' using a sliding-window classifier with 3×3 max-pooling window. Images are randomly rotated and translated.

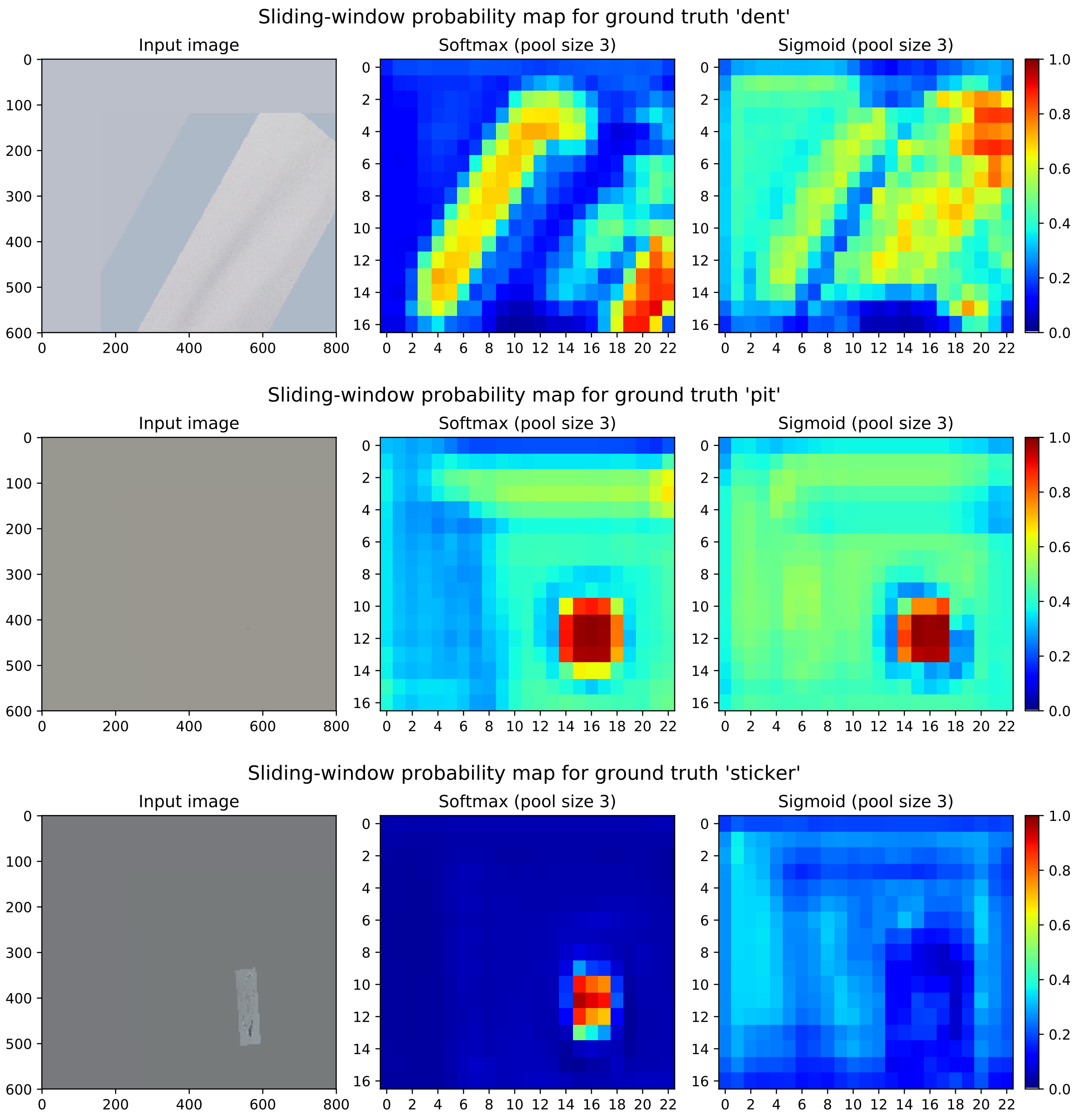


Figure E-7: Class probability maps on 600×800 pixel examples of 'dent', 'pit', and 'sticker' using a sliding-window classifier with 3×3 max-pooling window. Images are randomly rotated and translated.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *NIPS*, vol. 25, pp. 1097–1105, 2012.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

[3] L. Zhang, F. Yang, Y. D. Zhang, and Y. J. Zhu, “Road Crack Detection Using Deep Convolutional Neural Network,” *ICIP*, 2016.

[4] Y.-J. Cha, W. Choi, and O. Büyüköztürk, “Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 32, no. 5, pp. 361–378, 2017.

[5] Y.-J. Cha, W. Choi, G. Suh, S. Mahmoudkhani, and O. Büyüköztürk, “Autonomous Structural Visual Inspection Using Region-Based Deep Learning for Detecting Multiple Damage Types,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 9, pp. 731–747, 2018.

[6] S. Dorafshan, R. J. Thomas, and M. Maguire, “Comparison of deep convolutional neural networks and edge detectors for image-based crack detection in concrete,” *Construction and Building Materials*, vol. 186, pp. 1031–1045, 2018. [Online]. Available: `https://doi.org/10.1016/j.conbuildmat.2018.08.011`.

[7] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P. M. Jodoin, and H. Larochelle, “Brain tumor segmentation with Deep Neural Networks,” *Medical Image Analysis*, vol. 35, pp. 18–31, 2017. arXiv: `1505.03540`.

[8] Z. Zhang, X. Zhou, X. Zhang, L. Wang, and P. Wang, “A Model Based on Convolutional Neural Network for Online Transaction Fraud Detection,” *Security and Communication Networks*, vol. 2018, 2018.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] M Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," Microsoft Research, Tech. Rep., 2015. arXiv: 1506.01497v3.

[12] K. Fukushima, "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," *Biol. Cybernetics*, pp. 193–202, 1980. [Online]. Available: https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf.

[13] D. H. Hubel and T. N. Wiesel, "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex," *J. Physiol*, vol. 160, pp. 106–154, 1962.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016. arXiv: 1512.03385.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," in *ECCV*, 2016. arXiv: 1603.05027v3.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, C. Hill, and A. Arbor, "Going deeper with convolutions," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 1–9, 2015. arXiv: 1409.4842.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *CVPR*, 2016. arXiv: 1512.00567.

[18] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *CoRR*, 2016. arXiv: 1602.07261.

[19] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," in *CVPR*, 2017. arXiv: 1611.05431v2.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017. arXiv: 1608.06993.

[21] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in *CVPR*, 2017, pp. 1251–1258. arXiv: 1610.02357.

[22] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *ICML*, 2019. arXiv: 1905.11946.

[23] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018.

[24] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *ICLR*, 2017. arXiv: 1611.01578.

[25] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *CVPR*, 2018, pp. 8697–8710. arXiv: 1707.07012.

[26] J. Vanschoren, "Meta-Learning: A Survey," *CoRR*, 2018. arXiv: 1810.03548v1.

[27] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to Learn Using Gradient Descent," *ICANN*, pp. 87–94, 2001. [Online]. Available: http://link.springer.com/10.1007/3-540-44668-0_13.

[28] S. Ravi and H. Larochelle, "Optimization as a Model for Few-Shot Learning," in *ICLR*, 2017.

[29] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," in *ICML*, 2017. arXiv: 1703.03400v3.

[30] C. Finn, K. Xu, and S. Levine, "Probabilistic Model-Agnostic Meta-Learning," *CoRR*, 2018. arXiv: 1806.02817.

[31] T. Kim, J. Yoon, O. Dia, S. Kim, Y. Bengio, and S. Ahn, "Bayesian Model-Agnostic Meta-Learning," pp. 1–16, 2018. arXiv: 1806.03836.

[32] C. Finn and S. Levine, "Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm," in *ICLR*, 201. arXiv: 1710.11622.

[33] G. Koch, R. S. Zemel, and R. R. Salakhutdinov, "Siamese Neural Networks for One-Shot Image Recognition," in *ICML*, vol. 37, 2015.

[34] L. Bertinetto, J. F. Henriques, J. Valmadre, P. H. S. Torr, and A. Vedaldi, "Learning feed-forward one-shot learners," in *NIPS*, 2016. arXiv: 1606.05233.

[35] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching Networks for One Shot Learning," in *NIPS*, 2016. arXiv: 1606.04080.

[36] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical Networks for Few-shot Learning," *CoRR*, 2017. arXiv: 1703.05175v2.

[37] Y.-X. Wang, R. Girshick, M. Hebert, and B. Hariharan, "Low-Shot Learning from Imaginary Data," *CoRR*, 2018. arXiv: 1801.05401v2.

[38] Q. Cai, Y. Pan, T. Yao, C. Yan, and T. Mei, "Memory Matching Networks for One-Shot Image Recognition," *CoRR*, 2018. arXiv: 1804.08281.

[39] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "One-shot Learning with Memory-Augmented Neural Networks," *CoRR*, 2016. arXiv: 1605.06065.

[40] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap, "Meta-Learning with Memory-Augmented Neural Networks," in *ICML*, vol. 48, 2016. arXiv: 1605.06065.

[41] A. Graves, G. Wayne, and I. Danihelka, "Neural Turing Machines," *CoRR*, vol. abs/1410.5, 2014. arXiv: 1410.5401.

[42] B. Hariharan and R. Girshick, "Low-Shot Visual Recognition by Shrinking and Hallucinating Features," *ICCV*, 2017. arXiv: 1606.02819.

[43] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," in *CVPR*, 2014, pp. 512–519. arXiv: arXiv:1403.6382v3.

[44] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," in *ICLR*, 2013. arXiv: 1312.6229.

[45] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition," *CoRR*, 2013. arXiv: 1310.1531.

[46] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” In *NIPS*, vol. 27, 2014. arXiv: 1411.1792v1.

[47] S. Kornblith, J. Shlens, and Q. V. Le, “Do Better ImageNet Models Transfer Better?,” 2018. arXiv: 1805.08974.

[48] D. Mahajan and R. Girshick, “Exploring the Limits of Weakly Supervised Pretraining – Facebook Research,” *CoRR*, 2018. arXiv: 1805.00932. [Online]. Available: https://research.fb.com/publications/exploring-the-limits-of-weakly-supervised-pretraining/.

[49] K. He, R. Girshick, and P. Dollár, “Rethinking ImageNet Pre-training,” Facebook AI Research, Tech. Rep., 2018. arXiv: 1811.08883v1.

[50] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2980–2988, 2017. arXiv: 1703.06870.

[51] D. Ziou and S. Tabbone, “Edge detection techniques - An overview,” *International Journal of Pattern Recognition and Image Analysis*, vol. 8, no. 4, 1998. [Online]. Available: https://pdfs.semanticscholar.org/587a/acc01a4c33f0fe7fb172f5db785f40522b57.pdf.

[52] I. Abdel-Qader, O. Abudayyeh, and M. E. Kelly, “Analysis of Edge-Detection Techniques for Crack Identification in Bridges,” *Journal of Computing in Civil Engineering*, vol. 17, no. 4, pp. 255–263, 2003.

[53] J. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[54] J. W. Cooley and J. W. Tukey, “An Algorithm for the Machine Calculation of Complex Fourier Series,” *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[55] G Kaiser, “The Fast Haar Transform,” *IEEE Potentials*, vol. 17, no. 2, pp. 34–37, 1998.

[56] Y. Gao and K. M. Mosalam, “Deep Transfer Learning for Image-Based Structural Damage Recognition,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 9, pp. 748–768, 2018.

[57] M. Buda, A. Maki, and M. A. Mazurowski, “A systematic study of the class imbalance problem in convolutional neural networks,” *Neural Networks*, vol. 106, pp. 249–259, 2018. arXiv: 1710.05381.

[58] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, “Learning from class-imbalanced data: Review of methods and applications,” *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017.

[59] N Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-Sampling Technique,” *JAIR*, vol. 16, pp. 321–357, 2002. arXiv: 1106.1813.

[60] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, “Understanding Data Augmentation for Classification: When to Warp?” In *DICTA*, 2016. arXiv: 1609.08764.

[61] M. Kubat and S. Matwin, “Addressing the Curse of Imbalanced Training Sets: One-Sided Selection,” *ICML*, vol. 4, no. 1, pp. 179–186, 1997. arXiv: 1011.1669v3.

[62] L. Shen, Z. Lin, and Q. Huang, "Relay backpropagation for effective learning of deep convolutional neural networks," *LNCS*, vol. 9911, pp. 467–482, 2016. arXiv: 1512.05830v2.

[63] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in *ICCV*, 2017. arXiv: 1708.02002.

[64] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, "Cost Sensitive Learning of Deep Feature Representations from Imbalanced Data," *CoRR*, 2015. arXiv: 1508.03422.

[65] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[66] L. Breiman, "Bagging Predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[67] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "RUSBoost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, vol. 40, no. 1, pp. 185–197, 2010. arXiv: 1011.1669v3.

[68] R. Barandela, J. S. Sánchez, and R. M. Valdovinos, "New Applications of Ensembles of Classifier," *Pattern Analysis and Applications*, vol. 6, pp. 245–256, 2003.

[69] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 463–484, 2012.

[70] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[71] A. G. Howard, "Some Improvements on Deep Convolutional Neural Network Based Image Classification," 2013. arXiv: 1312.5402.

[72] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning Augmentation Policies from Data," 2018. arXiv: 1805.09501.

[73] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, pp. 1–18, 2012. arXiv: 1207.0580.

[74] N. Srivastava, G. E. Hinton, A. Krizhevsky, Ilya Sutskever, and R. R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *JMLR*, 2014.

[75] A. Krogh and J. A. Hertz, "A Simple Weight Decay Can Improve Generalization," in *NIPS*, vol. 4, 1992, pp. 950–957. [Online]. Available: http://0-citeseerx.ist.psu.edu.innopac.up.ac.za/viewdoc/summary?doi=10.1.1.41.2305.

[76] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR*, 2015. arXiv: 1412.6980.

[77] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *ICLR*, 2019. arXiv: 1711.05101.

[78] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *CVPR*, 2017. arXiv: 1611.10012.

[79] L. N. Smith, “Cyclical learning rates for training neural networks,” *WACV*, pp. 464–472, 2017.

[80] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” *CoRR*, vol. abs/1704.0, 2017. arXiv: 1704.04861.

[81] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in *CVPR*, 2018, pp. 4510–4520. arXiv: arXiv:1801.04381v4.

[82] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “SSD: Single Shot MultiBox Detector,” in *ECCV*, 2016. arXiv: 1512.02325v5.

[83] E. Hoffer, I. Hubara, and D. Soudry, “Train longer, generalize better: closing the generalization gap in large batch training of neural networks,” in *NIPS*, 2017. arXiv: 1705.08741. [Online]. Available: http://arxiv.org/abs/1705.08741.

[84] L. N. Smith, “No More Pesky Learning Rate Guessing Games,” U.S. Naval Research Laboratory, Tech. Rep., 2018. arXiv: 1506.01186v2.

[85] L. N. Smith and N. Topin, “Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates,” 2017.

[86] L. N. Smith, “A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay,” US Naval Research Laboratory, Tech. Rep., 2018. arXiv: 1803.09820.

Glossary

List of Acronyms

ANN	Artificial Neural Network
AUC	Area Under Receiver Operating Characteristic (ROC)
AutoML	Automated Machine Learning
CNN	Convolutional Neural Network
CV	Computer Vision
FC	fully-connected
FFT	Fast Fourier Transform
FHT	Fast Haar Transform
FNR	False Negative Rate
FPR	False Positive Rate
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
IoA	Intersection-over-Area
IoU	Intersection-over-Union
NAS	Neural Architecture Search
NTM	Neural Turing Machine
R-CNN	Region-based Convolutional Neural Network
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SHM	Structural Health Monitoring

SMOTE	Synthetic Minority Over-Sampling Technique
SRM	Structural Repair Manual
SSD	Single Shot Detection
SVM	Support Vector Machine
TPR	True Positive Rate