Visual damage detection on aircraft using deep learning

How across-the-board machine learning techniques can boost accurate detection with limited data

Julian von der Goltz

Literature Survey

TU Delft Delft University of Technology

Department of Cognitive Robotics (CoR)

Visual damage detection on aircraft using deep learning

How across-the-board machine learning techniques can boost accurate detection with limited data

Literature Survey

Julian Henry Freiherr von der Goltz

February 7, 2019

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

The work in this project is carried out at Mainblades Inspections and KLM Engineering and Maintenance. Their cooperation is hereby gratefully acknowledged.

Abstract

Aircraft inspections after unexpected incidents such as lightning strikes, bird strikes or hail strikes cause an unwanted and expensive inspection process. Performing a quick, drone-based damage inspection on an aircraft autonomously can save a vast amount of money due to the time-consuming process of manually preparing and inspecting the aircraft and the unplanned down-time of the asset. Recent advances in Deep Learning and applications in fields such as civil engineering, medical image analysis, financial fraud detection and autonomous driving suggest that an accurate and reliable damage detection algorithm based on Convolutional Neural Networks (CNNs) is feasible. This literature survey aims at identifying the building blocks, challenges and possible solutions associated with designing a damage detection algorithm. The first part of this report is concerned with the basic building blocks of a potential detection model. These building blocks are the backbone CNN, pre-trained as damage classifier and the overall object detection architecture that features additional components for localization and scale invariance. The second part discusses the challenges that arise due to the project constraints: The amount of training data from this very specific domain is limited. The data is highly imbalanced; containing a high number of non-damage or background samples and a low number of actual damages. The image quality also ranges from poor (taken by cell phones with additional compression) to very high (taken by cameras with large sensors and optical zoom). Lastly, the overall number of training samples is low when compared to datasets like ImageNet that are available for research. The methods presented here specifically address the class imbalance problem and low-data problem. The techniques that emerged from this study are generative deep models that synthesize new training data and meta-learning that aims at optimizing across tasks. Generative data augmentation conditioned on labels or attributes can compensate for the class imbalance, artificially improve image quality and increase training set size. However, meta-learning, and more specifically low-shot learning, is best applicable to a very-low-data regime. We propose an approach that combines a DenseNet implementation as baseline classifier with a Conditional Variational Autoencoder (CVAE) to boost the classification accuracy with limited available data. The obtained classification model can then be extended to be used in a damage detection model for a later stage of this project.

Table of Contents

1 Introduction **1**

1-1 Background on Computer Vision and Deep Learning 2

1-2 Goals and Problem Formulation 4

1-3 Research Questions 5

1-4 Report Structure 5

2 Basic Principles of Deep Learning **7**

2-1 Deep Neural Networks 7

2-2 Convolutional Neural Networks (CNNs) 10

3 Image Recognition **13**

3-1 Classical Image Recognition 13

3-2 Image Recognition using Convolutional Neural Networks 14

3-3 Summary 19

4 Object Detection Architectures **21**

4-1 Staged Detection Architectures 22

4-2 Single-Stage Detection Architectures 24

4-3 Summary 26

5 Class Imbalance Methods **29**

5-1 Performance Metrics 29

5-2 Data-Level Methods 30

5-3 Algorithm-Level Methods 32

5-3-1 Cost-Sensitive Methods 32

5-3-2 Thresholding 32

5-3-3 Ensemble Methods 32

5-4 Summary 33

6 Data Augmentation 35
6-1 Classical Augmentation Techniques . . . 35
6-2 Augmentation Techniques based on Generative Models . . . 36
6-2-1 Variational Autoencoders (VAEs) . . . 37
6-2-2 Generative Adversarial Networks (GANs) . . . 40
6-3 Summary . . . 44

7 Meta-Learning 47
7-1 Meta-Learning Across Tasks . . . 48
7-2 Transfer Learning . . . 49
7-3 Low-Shot Learning . . . 50
7-4 Summary . . . 57

8 Summary 59
8-1 Building Blocks for Damage Detection . . . 59
8-2 Challenges and Possible Solutions . . . 60

9 Conclusion 63
9-1 Future Work . . . 65
9-2 Other Considerations . . . 66

A Requirements 67

Glossary 77
List of Acronyms . . . 77
List of Symbols . . . 79

List of Figures

1-1 Example of a lightning strike entry and exit 1
1-2 Example outputs of a CNN-based object detection model 3
1-3 Structure of this literature survey. 6

2-1 A simple neural network 8
2-2 Visualization of a convolutional layer 11
2-3 LeNet-5 architecture 12

3-1 Inception Module 16
3-2 Residual Unit 17
3-3 Modified Residual Unit 17
3-4 DenseNet architecture 18

4-1 R-CNN 22
4-2 Fast R-CNN 22
4-3 Faster R-CNN 23
4-4 R-FCN 23
4-5 RetinaNet 25
4-6 mAP vs. inference time for various detection models 27

5-1 Receiver Operating Characteristic (ROC) curve 31

6-1 Variational Autoencoder 37
6-2 Conditional Variational Autoencoder 39
6-3 Overview of a selection of GAN architectures 44

7-1 Model-Agnostic Meta-Learning (MAML) 49
7-2 Siamese Net, Siamese Learnet and Learnet 51
7-3 Memory Augmented Neural Network (MANN) strategy 52

7-4 Matching Network (MN) Architecture 53
7-5 Two-phase training for low-shot learning 54
7-6 Memory Matching Network (MM-Net) Architecture 57

List of Tables

3-1 Overview of a selection of CNN models 20

4-1 Overview of a selection of object detection models 28

5-1 Confusion Matrix 30

7-1 General guidelines for transfer learning 50
7-2 Comparison table for meta-learning and low-shot methods 58

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. In most of the cases, lightning strikes do no require an immediate repair, which results in an unnecessary down-time of the asset due to the inspection. Figure 1-1 shows an example of a lightning strike entry and exit.

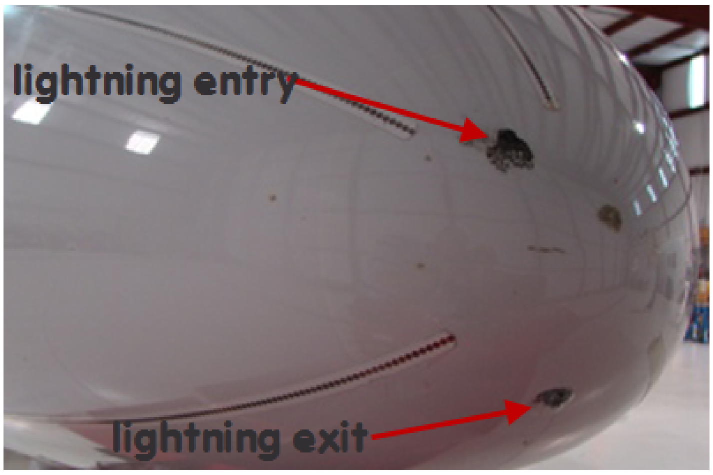


Figure 1-1: Example of a lightning strike entry and exit. Source: Mainblades Inspections

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. This MSc-project is concerned with interpreting the acquired image data and visually detecting the damages. It is clearly an application-focused problem setting with real-life constraints. The objective of this literature survey is to identify the building blocks of a damage detection model and identify challenges and possible solutions for this specific problem setting, in the form of existing algorithms, methods and best practices from Computer Vision and Deep Learning.

Additionally, due to constraints in availability and quality of training data, this survey will also present techniques from the latest research in the field and identify open areas of research. Eventually, the outcome of this MSc-project will be an assessment of the feasibility of using a Deep Learning algorithm for damage detection, where the literature survey will serve as a starting point for the proposed implementation and experiments.

1-1 Background on Computer Vision and Deep Learning

The premise that started this project is as follows: It is possible for humans to recognize damages on aircraft by looking at high-resolution pictures taken from a drone. Considering recent technological advancements and applications for example in autonomous driving, we presume that 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 application of object detection from the field of Computer Vision. Classically, object detection is done by hand-engineering features that identify objects, locating the features and assigning them to a category. Deep neural networks are promising candidates to replace hand-crafted algorithms as they are theoretically able to approximate any function, given a high enough capacity [1], which can be created using deep neural networks with many layers.

Applications of 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 [2]. Especially Deep Convolutional Neural Networks (CNNs) are on par with, or even exceeding human performance in some visual recognition tasks [3]. Deep learning for detecting damages or anomalies in images has found applications in civil engineering to detect cracks in infrastructure [4], [5], in the medical domain to perform brain tumour segmentation in MRI data [6] and in the financial sector to detect on-line transaction frauds [7]. 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.

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 1-2 shows an example of CNNs 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 using convolutional layers, but still need

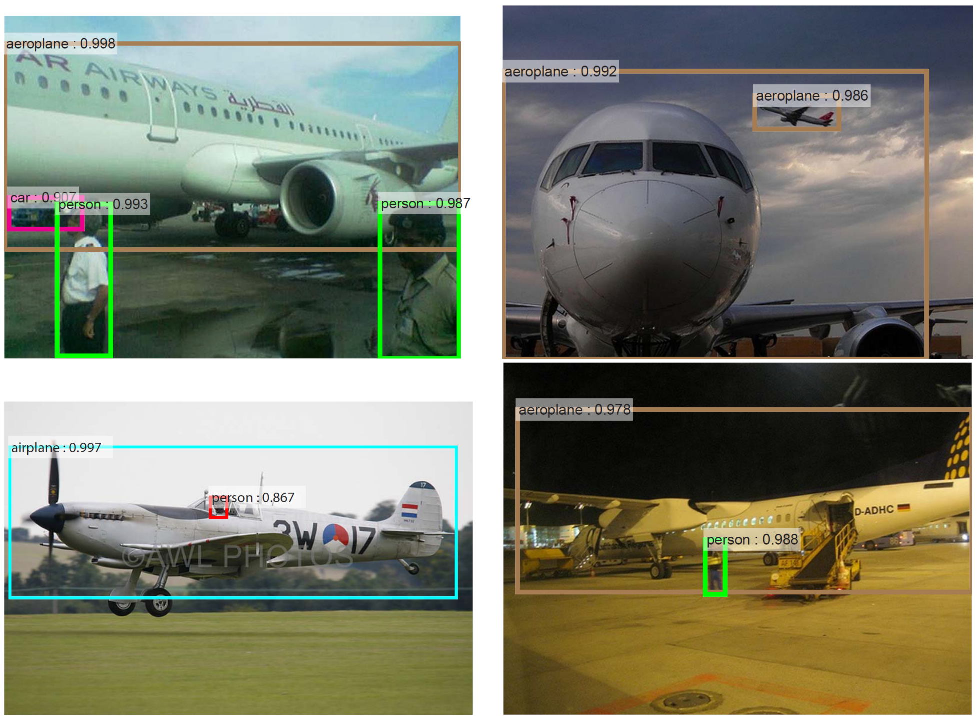


Figure 1-2: Example outputs of a CNN-based object detection model involving aircraft. The images stem from the PASCAL VOC 2007 test set [8]. The object locations are given as bounding boxes along with class labels and confidence scores. Source: Faster R-CNN [9]

large datasets for training to prevent overfitting and provide generalization capability. The training data is therefore a crucial component in setting up a CNN-based recognition pipeline.

Data Augmentation

Within Machine Learning, there is ongoing research to explore methods to minimize the dependency on big datasets. One approach is to work on the data level and artificially increase the size of the dataset and remove class imbalances present in the training data [10], that can be detrimental to the performance on the minority (i.e. damage) classes. This can be done by specifying some randomized transformations (e.g. translations, rotations or colour transformations) by hand and applying them to the dataset before training. This process is called data augmentation and is currently considered a best practice in Deep Learning.

A new area of research are generative methods like Variational Autoencoders (VAEs) [11] or Generative Adversarial Networks (GANs) [12], that approximate the data distribution and generate new data. This could be applied to dataset augmentations as well to increase dataset size and improve the generalization capacity of the main classifier.

Meta-Learning

Meta-learning describes methods for learning to learn on the algorithm level. Meta-learning is concerned with learning meta-architectures of a learning algorithm (i.e. number of layers, neurons in a neural net) or learning how to perform well *across* different tasks rather than learning a *specific* task like a certain classification mapping [13]. Transfer learning is a meta-learning approach where models trained on large datasets can be reused for slightly different tasks or smaller datasets. Low-shot learning, which is another meta-learning approach, goes even further and explores algorithms that can learn new tasks from only a low number of data points (in the order of zero to 20 examples vs. several thousand examples for conventional deep learning).

1-2 Goals and Problem Formulation

The goal of this project is to detect damages on aircraft surfaces based on drone inspection images. The output of the algorithm shall be the class, confidence and locations of the damages (similar to Figure 1-2). It is not strictly necessary to achieve close to 100% accuracy from the start, but due to the safety aspect in aviation, the False Negative Rate (FNR) of damages that have to be repaired immediately should be near zero. A the time of writing, the number of classes that have to be distinguished is around 20-30. This includes different types of damages (lightning strikes, abrasion, cracks, dents, scratches, etc.), other objects of interest (text, markings) and a generic catch-all background class. The intermediate goal, the guideline throughout this text, is to assess the feasibility of a CNN-based damage recognition algorithm given the following challenges:

- 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 overrepresented. 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 overfitting and lack of generalization. Currently, the dataset size is in the order of 250 labelled images, containing approximately 3,200 objects. Around 5,000 - 9,000 images are in the pipeline to be labelled.
- 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 damage images are taken by low-resolution phone cameras or from a large distance. Some images of damages also contain alterations like handwritten markings and repair instructions, that would not be visible during a drone inspection.

Therefore, the problem formulation for this project reads as follows:

How can a visual detection algorithm be developed, that reliably detects damages with low False Negative Rate (FNR), given that the available data for training is limited in both quantity and quality, and given that there are large class imbalances present, that bias against the damages?

1-3 Research Questions

In this literature survey, there are two main objectives: the first is to identify the building blocks for a damage detection model and the second is to identify possible solutions to the challenges given in the problem formulation above. To solve the challenges, the problem is divided into sub-problems with research questions which will serve as guidelines throughout the report:

- How can we help a damage detection model deal with large class imbalances, such that a low FNR on the minority classes is achieved?
- How can we make sure that a damage detection model achieves high generalization capability when only a small training dataset is available?
- How can we standardize the quality of the training images and match it to the testing images?
- Is there a relationship between the previously mentioned? In how far are they equivalent or different?

For these research questions, it is assumed that Object Detection by itself is a solved problem. However, this report still dedicates a large part to Deep Learning, Image Recognition and Object Detection in general to provide a bottom-up basis for answering the research questions.

1-4 Report Structure

Figure 1-3 shows the structure of this survey. It relates the high-level goal of the MSc-project to the sub-objectives of this literature survey and the associated research topics.

Chapter 2 discusses the basics of Deep Learning and CNNs and introduces the relevant terminology. Chapter 3 and 4 treat Image Recognition and Object Detection, respectively. All chapters from Chapter 3 on present multiple methods, where certain decisions have to be made, therefore, they contain each a short summary to compare the methods. Accurate classification with limited data poses the core challenge, therefore the techniques presented in Chapters 5-7 are focused on the recognition/classification problem and not on object detection. They treat relevant research on the Class Imbalance Problem, Data Augmentation and Meta-Learning, respectively.

The techniques discussed in this report represent a wide range of sub-topics in Deep Learning, with the exception of Reinforcement Learning and Semi-Supervised Learning. Chapter 8 summarizes the findings of the report and Chapter 9 concludes by selecting the most promising ones for future work.

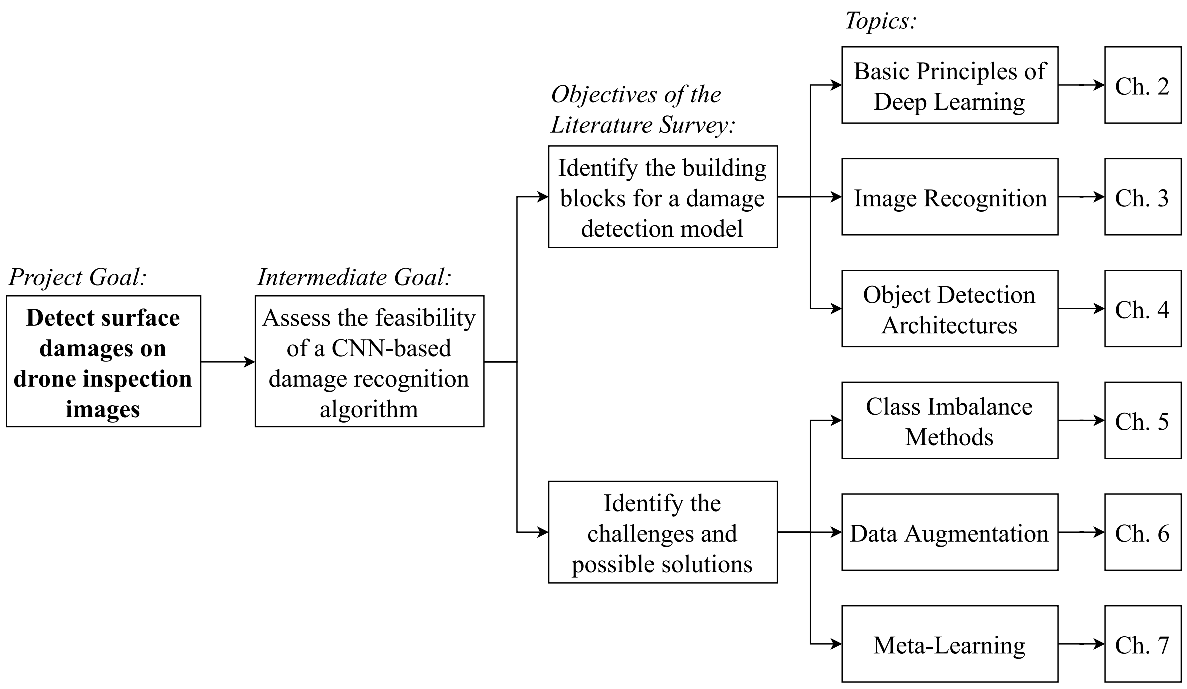

Project Goal:
Detect surface damages on drone inspection images
Intermediate Goal:
Assess the feasibility of a CNN-based damage recognition algorithm
Objectives of the Literature Survey:
Identify the building blocks for a damage detection model
Identify the challenges and possible solutions
Topics:
Basic Principles of Deep Learning
Ch. 2
Image Recognition
Ch. 3
Object Detection Architectures
Ch. 4
Class Imbalance Methods
Ch. 5
Data Augmentation
Ch. 6
Meta-Learning
Ch. 7

Figure 1-3: Structure of this literature survey.

Chapter 2

Basic Principles of Deep Learning

This chapter discusses the necessary basics of deep learning as introduced in the two books by I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* The MIT Press, 2016 [14] and by T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning.* Springer, 2009 [15]. Important symbols and terminology (in *italics*) will be introduced in this chapter as well. The other chapters will frequently refer to the concepts introduced here. Readers familiar with Deep Learning, or specifically CNNs, training by back-propagation, choice of hyperparameters, generalization error and cross validation may skip this chapter. Section 2-1 discusses Deep Neural Networks in general, while Section 2-2 goes into more detail about Convolutional Neural Networks (CNNs).

2-1 Deep Neural Networks

Deep Learning is the class of machine learning models that uses Deep Neural Networks as underlying architecture for classification and regression. When using the term learning, this text refers to *supervised learning*, if not mentioned otherwise. That is, learning from a ground truth in a training dataset that has been acquired (in other words, gathered, processed and labelled with ground truths) a priori. Simple *feed-forward* artificial neural networks, also called *multi-layer perceptrons* are function approximators that map an input x to an output y, where the function parameters are obtained by optimizing an objective function on the training dataset.

Components of Neural Networks

Neural Networks consist of one or more *layers* of *neurons* that are loosely related to biological neurons in the sense that they process several input signals and *activate*, or “fire”, on a single output, depending on a (linear) combination of the inputs. It should be noted, that the goal of deep learning research is not necessarily to replicate a biological brain, though many analogies exist. The *activation function* is a non-linear function that can also be seen as *thresholding* (e.g. Rectified Linear Unit (ReLU) [16]) or as *squashing* (e.g. Sigmoid, Hyperbolic Tangent, Softmax). The non-linearity is necessary to increase the expressive *capacity* of the neural net. Without them, they would just be linear regressors or classifiers. Each layer l takes the output vector h_{l-1} of the neurons of the previous layer, applies an affine transformation and passes it element-wise through the activation function $\phi_l(u_l)$:

$$h_l = \phi_l(u_l) = \phi_l(W_l h_{l-1} + b_l) \tag{2-1}$$

The first layer takes x as input and the last layer outputs y. Figure 2-1 shows a simple neural net, consisting of an input layer, a *hidden layer* and an output layer. Hidden layers are the layers between input and output layers. Their output is called *hidden variable* or *latent variable*.

By the Universal Approximation Theorem [1], neural networks are in theory capable of approximating any non-linear function f, given a high enough number of neurons (it is not known, however, what the value of that number is). This is where the motivation for "deep" networks comes from: Using more and more layers to increase the capacity of the neural net. Recent architectural modifications like residual blocks [17] have made "going deeper" more feasible. However, deeper networks introduce more parameters, increasing the danger of *overfitting*. Methods to prevent overfitting are discussed below.

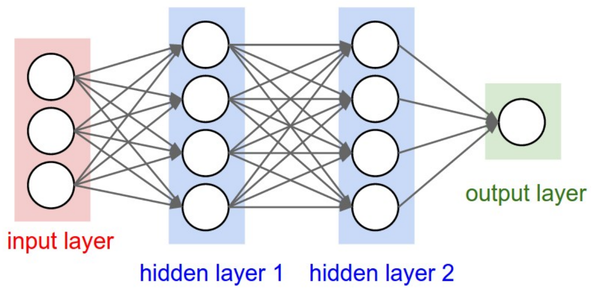


Figure 2-1: Visualization of a simple neural network in a graph. The graph nodes correspond to neurons and their activations, the edges of the graph correspond to linear transformations with weights W and biases b. This neural network has an input vector with dimension 3, two hidden layers with 4 *hidden units* each and an output of dimension 1. Source: CS231n [18]

Training a Neural Network

A deep neural network is characterized by the number of layers, number of neurons in each layer, the weight matrix W_l, bias vector b_l and activation function ϕ_l in each layer L. The parameters W and b (grouped together as θ) are *trained* by minimizing a loss function. This loss function is usually the negative log-likelihood of the training data $-\log p(x, y; \theta)$. Neural Nets therefore *learn* from the data and do not require any other domain knowledge. The log-likelihood reduces to a certain performance metric dependent on the model type $p(y|x)$, for example squared error for regression or cross entropy for classification. Due to the non-linear activation functions, the loss is non-convex in θ and there is no way to obtain a global optimum in closed form. Neural nets are thus optimized using iterative methods such as gradient descent until a satisfying point is found:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}(f_{\theta_{t-1}}, \mathcal{D}) \tag{2-2}$$

where t is the iteration index, α is the learning rate, ∇_θ is the gradient with respect to the parameter θ and $\mathcal{L}(f_\theta, \mathcal{D})$ is the loss of the network output f_θ evaluated on the dataset $\mathcal{D}$. In contrast to vanilla gradient descent, Stochastic Gradient Descent (SGD) uses only a randomly sampled subset $\mathcal{D}_B$ of the training data in each parameter update step to calculate the gradients. These subsets are called *mini-batches* and are characterized by the *batch-size*, i.e. the number of training samples they contain. The optimizer will thus iterate through the training dataset across mini-batches. One complete pass through the dataset is called an *epoch*. Other iterative optimizers have been developed for Deep Learning that make use of momentum terms and other modifications like higher moments, for example Adam [19].

Calculating the gradients of the loss (or error) with respect to the parameters is called *back-propagation* [20]. Defining f as the network output, the gradient of the loss $\mathcal{L}$ is:

$$\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial f} \frac{\partial f}{\partial \theta} \tag{2-3}$$

Using Equation 2-1, and absorbing the biases b into the the weight matrix W, the gradients of the network with respect to its own parameters in the last layer L are:

$$\frac{\partial f}{\partial W_L} = \frac{\partial h_L}{\partial W_L} = \frac{\partial \phi_L(u_L)}{\partial u_L} h_{L-1} \tag{2-4}$$

The intermediate variables h_{L-1} and u_L are the outputs of lower layers and calculated in the *forward pass*. They can be reused to determine the gradients for the *backward-pass*. The gradients of the activations $\frac{\partial \phi_L(u_L)}{\partial u_L}$ can be obtained analytically, therefore, no numerical gradient approximation by perturbations is necessary. Going back layer by layer, for each layer l, the gradients $\frac{\partial h_l}{\partial W_l}$ are obtained by multiplication of gradients from later layers with output variables of previous layers. This can be implemented very efficiently by performing matrix multiplications in Graphical Processing Units (GPUs). A downside of this is the phenomenon of vanishing gradients in deep networks, which is addressed by skip connections and residual blocks [17] (see Chapter 3).

The parameters are randomly initialized and iteratively updated until a convergence criterion is satisfied. For training the network, certain *hyperparameters* have to be chosen, such as learning rate, batch size, momentum and regularization parameters. Hyperparameters are determined using *cross-validation*, meaning that for each hyperparameter, a range of values is selected. Each value is used for several training runs on different random splits of the training data into *training* and *validation sets*. The average performance on the validation sets is reported for that hyperparameter value and the value with the highest performance will be used for the testing phase.

Generalization Capability and Overfitting

Generalization capability of the final neural network model is tested with a *test set* that contains data not used during training. In contrast to finding the global optimum in conventional optimization, it is not preferable for neural networks to find a global optimum or even a local optimum. This is because neural networks are prone to *overfitting*. A neural network that is trained to a global optimum on the training set and achieving close to perfect training accuracy, would have a poor performance on validation and test sets. The network is severely overfitted to the training set, meaning it can perfectly memorize every training example, but is not able to process new examples.

Overfitting can be avoided by using large training sets and applying *regularization* terms on the parameters (from a statistical point of view, parameter regularization is a way of representing prior knowledge $p(\theta)$ in the cost function. From an optimization point of view, it reduces the model complexity by keeping the parameters small and it prevents ill-posed solutions). Another way of regularization is *dropout* regularization [21], where during training, some connections are randomly left out. The reasoning behind dropping connections is that each neuron is encouraged to learn features that are useful "on its own", instead of relying on other neurons. Additionally, carefully choosing hyperparameters and using heuristics during training, like the early stopping rule, are standard practices to prevent overfitting. The early stopping rule makes sure that the network stops training when no improvement on a separately held validation set is seen.

Other Types of Neural Networks

Next to feed-forward neural nets, there are other neural net architectures like *Recurrent Neural Networks (RNNs)* and *CNNs*. RNNs take as input a data point x_t at time instance t and update some internal state h_t as a function of the input and previous state h_{t-1}. An RNN is able to capture temporal relations between data points. The output y_t is a function of h_t and x_t. They are good candidates to operate on time-series data. There is an analogy to dynamical systems in control theory, but the two are different in the sense that RNNs are learned from data and not modelled from physical laws and that they are non-linear by choice, whereas it is often preferred to treat dynamical systems as linear approximations. A widely used example of recurrent nets is the Long Short-Term Memory (LSTM), which is able to learn both long and short term relations in sequences of data.

CNNs capture spatial relations between data points by *convoluting* over data arranged in 2-dimensional grids. This makes them ideal for image analysis which is the topic of interest in this project. The following section will elaborate on CNNs.

2-2 Convolutional Neural Networks (CNNs)

A CNN is a special form of deep neural networks used for image recognition. The Stanford University lecture by F.-F. Li, J. Johnson, and S. Yeung, *CS231n Convolutional Neural Networks for Visual Recognition*, 2018. [Online]. Available: http://cs231n.stanford.edu (visited on 10/16/2018) [18] provides a very good overview of CNNs, including the most recent research in the field.

Components of CNNs

In a CNN, the first layers are not fully connected as in normal neural nets. Instead, each neuron has a *kernel* or *filter* with weights that convolute across the image (our output of the previous layer), corresponding to the idea that relationships between pixels are stronger, if they are in the same neighbourhood. The filter weights and biases are multiplied and

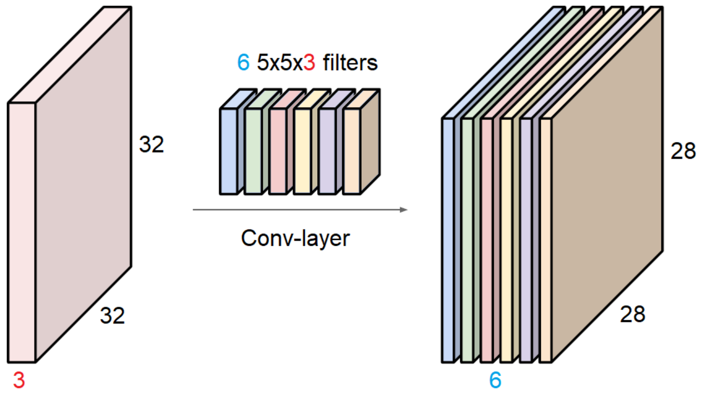


Figure 2-2: Visualization of a convolutional layer. Every filter convolutes over the input volume and produces one channel in the output volume. Best viewed in colour. Source: CS231n [18]

added with image patches at different locations and passed through an activation function. A *convolutional layer (conv-layer)* is characterized by the number of filters, the filter size (something like 3×3, 5×5, etc.) the *stride* (step size that the filter moves on the image), *padding* (additional values added outside of the input image to preserve height and width in the output) and activation function. By using this kind of shared weights architecture, the number of parameters is greatly reduced, compared to a fully connected architecture. Each neuron has a *perceptive field*. It is the set of pixel locations in the input and the feature maps up until the layer where the neuron is located, that influence the neuron's output. The receptive field of a neuron with 5×5 kernel is larger than the receptive field with of a 3×3 kernel. Also, a neuron in a higher layer has a larger receptive field than a neuron in a low layer.

Preserving the structure of images throughout the network is visualized in Figure 2-2 using 3D-volumes (called tensors) instead of concatenated vectors. The height and width of the volume are the height and the width of the image. The depth is the number of *channels* (three for RGB-images or the number of feature maps). If the input to a conv-layer has size $h \times w \times d$ and the layer has c filters, a kernel size of k, stride of s and padding of p, the output of will be of size $(h + 2 \cdot p - k)/s + 1 \ \times \ (w + 2 \cdot p - k)/s + 1 \ \times \ c$. The filters will be of size $k \times k \times d$ and every depth channel in the output corresponds to the input convoluted with one filter of the conv-layer. The number of parameters in a conv-layer is $c \cdot (k \cdot k \cdot d + 1)$, where the $+1$ term stems from the addition of a bias in each neuron. All these parameters can be learned through back-propagation, given that the activation functions are differentiable. Each filter learns its parameters from data. An output channel of a conv-layer is called a *feature or activation map*, as it corresponds to the activations caused by a certain feature in the input.

Historically, convolutional layers are alternated with *pooling layers*. Pooling layers perform a down-sampling operation by dividing the image into patches (similar to convolutions, characterized by size 3×3, 5×5, etc. and stride) and selecting a representative value for every patch. *Max-pooling* selects the maximum value in the patch, *average-pooling* selects the average of the patch. This operation serves to reduce computing requirements in the subsequent layers while preserving important information. Pooling layers do not contain any learnable parameters, i.e. weights or biases.

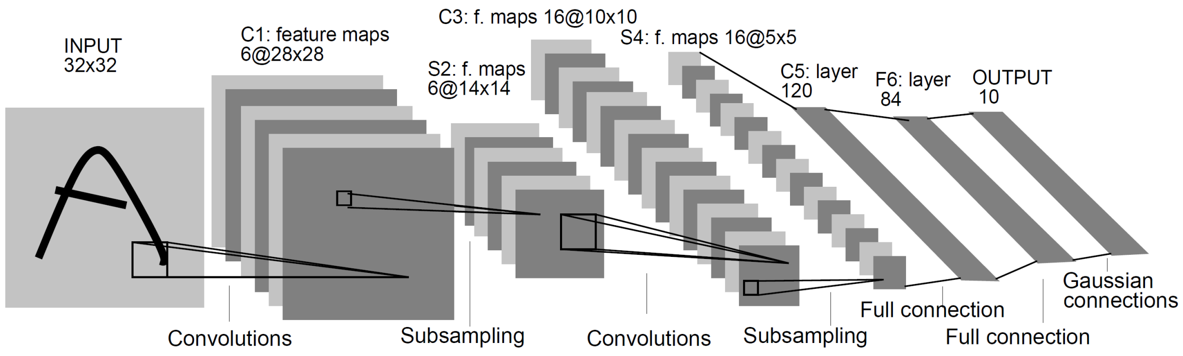


Figure 2-3: Architecture of the LeNet-5 CNN by LeCun, Bottou, Bengio, *et al.* [22].

Classification and Regression with CNNs

A pure CNN by itself does not strictly do classification or regression, it can be seen as a non-linear *feature extractor* learned on data, that outputs a certain embedding of the input image into a lower dimensional space. The extracted features can be used by a subsequent linear machine learning method to make predictions. Therefore, after the convolutional and pooling layers, *fully-connected (FC) layers* are added. Note that until here, the network is invariant to input vector size. The fully-connected final layers output scalars for regression, probabilities over classes (softmax-classifier) or the class score in case of a multi-class Support Vector Machine (SVM). The expression CNN usually refers to the combination of convolutional net, also called feature extractor, and fully connected classifier or regressor, called *head*. Any type of classifier or regressor is possible on top of the CNN. This modularity makes it very versatile for a wide range of tasks that use visual data.

History of CNNs

LeCun, Bottou, Bengio, *et al.* [22] pioneered CNNs with the application in handwritten character recognition. The architecture of his LeNet-5 is shown in Figure 2-3. It consists of two conv-layers followed by subsampling layers and two fully-connected layers. The first conv-layer C1 has six 5×5 kernels without padding. Subsampling is done by locally averaging in 2×2 patches, multiplying the result with a learnable coefficient, adding a learnable bias and passing the result through a sigmoid function. The second conv-layer C3 has 16 filters. The spatial resolution is thereby further decreased and the depth further increased. C5 is also a 5×5 conv-layer with 120 filters but gets 5×5 feature maps as input, which makes it equivalent to a fully connected layer. The last hidden layer F6 is fully connected with 84 units and the output computes the Euclidean distance to parameter vectors that correspond to digit classes and are fixed beforehand. The predicted class is thus the one with the smallest distance to the embedding created by the network.

Development in CNNs took off in 2012, when AlexNet [2] won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [3], [23]. The specific CNN architectures that have been developed since then for classification and other visual recognition tasks will be presented in more detail in Section 3-2.

Chapter 3

Image Recognition

The first step to object detection is object recognitions. Object classification naturally follows recognition. First, important features have to be recognized, then these features are assigned to an object class. Once recognitions and object classification is solved, moving on to detection is a rather straightforward step. Image Recognition is treated in this Chapter and its integration into a detection architecture is treated in Chapter 4. The first part of this chapter (Section 3-1) gives a brief overview of classical image recognition models and the second part (Section 3-2) gives a more detailed description of Deep Learning based models using Convolutional Neural Networks (CNNs).

3-1 Classical Image Recognition

Object recognition and subsequently classification is a standard task in Computer Vision. Until the advent of Deep Neural Networks, classification was done by looking for hand-engineered features in the input image to represent the objects of interest. Throughout this report, the term *hand-engineering* refers to meticulously defining and programming properties (local and global features, spatial relationships) of objects that uniquely identify them and possibly programming of complex heuristics to make the algorithms more robust.
Features could be eyes in faces, wheels in vehicles, or more low-level features like edges or blobs. These classical object representation models can be divided into different model classes:

- Bag of Word (BoW) models extract local features around an interest point but do not use spatial relationships.
- Global Objective Models extract characteristic global features, examples are the Histogram of Oriented Gradients (HOG) [24] and Scale-Invariant Feature Transform (SIFT) [25].
- Part-Based Models like the Constellation Model [26], that include features and mutual locations between them in the representation.

The extracted features are fed into a classical machine learning algorithm (meaning non-deep neural network) such as Support Vector Machine (SVM), k-Nearest Neighbours (kNN), Bayesian Classifier, etc. to determine if the object is present (recognition) or to which class the object belongs to (classification).

The examples above are representative but do not mirror the complete diverse space of models that have been developed over the years. Their advantage is that the features often are comprehensible i.e. they correspond to object parts that humans can recognize and understand as well. Additionally, they do not require large data to train as the number of parameters is usually small. On the other hand, they require the fine-tuning of many hyperparameters, which can get very complex when different feature extractors are combined. Other disadvantages are that hand-engineering can be very cumbersome and expensive for a big number of object classes. Hand-engineering also requires domain knowledge if the objects in question are from a specific field. It is also possible that the object class is represented in a form that may not be obvious to a human programmer and that the hand-engineered representation will not generalize well or fail to capture edge cases.

3-2 Image Recognition using Convolutional Neural Networks

A deep Convolutional Neural Network (CNN) has the property that it learns the features that are important from the data itself (see Section 2-2 for basic principles). No hand-engineering and domain knowledge is required, only a sufficient amount of representative training data must be available. It is robust against variations in translations, rotations, scale etc. of the input, if the variation is also present in the training data. Due to the deep connectivity, it catches global and local relationships between features. CNNs trained on large datasets are outperforming hand-engineered feature extractors even on novel datasets or tasks [27], [28] (See also Section 7-2).

As the parameter space is very large, the required training sets are also very large. Two developments in the 2010s lead to the advent of CNNs and deep learning in general: The available computing power in highly parallelized Graphical Processing Units (GPUs) was sufficient to train deep CNNs on large data sets. At the same time, vast amounts of data in all forms became available and accessible through the internet. As a consequence, CNNs had their breakthrough in 2012 when AlexNet [2] won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [3], [23] with a large margin before the second place.

The ILSVRC serves as a benchmark for visual recognition tasks. It is a competition on the ImageNet dataset, that contains over 14 million labelled images. The competition itself is done on a subset of ImageNet with 1,000 non-overlapping classes, which is much larger but also more complex than the data for the problem at hand [1]. The winners of this competition set state-of-the-art benchmarks for image recognition. A selection of their models as well as some relevant evolutions of them is presented below. Not all existing CNN model architectures can be discussed in this text. The ones that are discussed, are representative of the recent trends in deep learning.

[1]To put this into perspective, at the time of writing, there are less than 10,000 images available for the damage detection, with an average of 13 objects of interest per image, which puts an upper bound for the total number of objects at 130,000 for approximately 20-30 object classes.

AlexNet

Krizhevsky, Sutskever, and Hinton [2] attribute the success of their AlexNet architecture to the following features, with the most important first:

- The use of non-saturating Rectified Linear Units (ReLUs) [16] as non-linearities:

  $$\phi(x) = \max(0, x) \tag{3-1}$$

  It prevents gradients from vanishing as opposed to saturating non-linearities like hyperbolic tangents or sigmoids. This leads to faster training. It effectively makes the neural network a large piecewise linear approximator.
- Training on multiple GPUs. As mentioned, highly parallelized matrix operations in Graphical Processing Units (GPUs) accelerated training of large scale neural networks.
- Local Response Normalization. Normalizes a kernel activation by the sum of adjacent kernel maps, increases generalization performance.
- Overlapping pooling. Setting the pooling layers such that the patches to be pooled from overlap, further reduces error rates.

The first two features have prevailed and been used by all models that follow. The other two were deemed less effective or important by subsequent research. AlexNet consists of eight layers with a total of 60 million parameters: five conv-layers and three fully-connected (FC) layers. The first conv-layer uses 96 filters of size $11 \times 11 \times 3$ with stride 4, the second layer uses 256 filters of size $5 \times 5 \times 48$, the third conv-layer uses 384 filters of size $3 \times 3 \times 256$, the fourth layer uses 384 filters of size $3 \times 3 \times 192$ and the fifth layer has 384 filters of size $3 \times 3 \times 192$. The two following fully-connected layers have 4069 neurons each and the output layer is a softmax across 1000 outputs, each corresponding to the probability for one of the ImageNet classes.

The network is split after the first conv-layer, parallelized across two GPUs and rejoined after in the output layer. They tackle overfitting by using extensive data augmentation (Section 6-1) and dropout regularization [21], both of which became standard practices in deep learning.

ZF-Net and VGG Net

In 2013, ZF-Net [29], a slightly modified and fine-tuned version of AlexNet, won the ILSVRC. Their main contribution is the introduction of visualization techniques to show the activities of the different layers in a CNN. They use their findings to debug problems with AlexNet and obtain better results, showing the importance of hyperparameter tuning in neural nets.

In 2014, the second place went to VGG Net by Simonyan and Zisserman [30]. It is worth a mention because of its homogeneous network architecture. VGG Net consistently uses small filters of size 3×3 with stride 1, and 2×2 max-pooling with stride 2 (non-overlapping pooling). They argue that multiple layers of small filters have the same receptive field as a less layers with large filters. This increases the number of non-linearities and reduces the number of parameters. The authors also step away from Local Response Normalization as it increases computation resources with limited increase in performance. The three fully connected layers have again size 4096, 4096 and 1000, respectively. The deepest variant of their VGG Net has 19 weight layers with about 144 million parameters. It also has the best performance compared to less deep variants.

Inception Networks

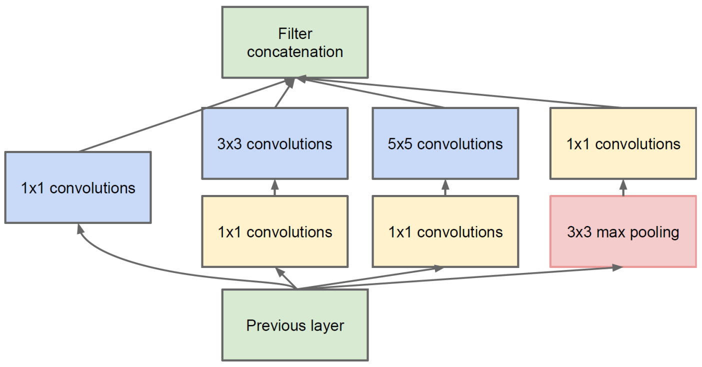


Figure 3-1: Inception module with dimension reductions. Source: GoogLeNet/Inception-v1 [31]

The actual winner of the 2014 ILSVRC was GoogLeNet by Szegedy, Liu, Jia, *et al.* [31], based on the Inception architecture. Their approach was the first one that deviated from simply stacking more and more conv- and pooling layers serially: It introduces *inception layers* that perform different operations (conv, pooling) in parallel (see Figure 3-1), with the goal of increasing the depth and width of the network while keeping the computational cost constant. A key component that enables different convolutional operations in parallel is that 1×1 convolutions are applied before the large convolutions to reduce the dimensionality of the feature maps. This reduces the number of parameters by orders of magnitudes. GoogLeNet has 22 layers with only a total of 5 million parameters.

For training, auxiliary convolutional classifies are added after some intermediate layers to provide strong gradient signals to the lower layers. These classifiers are discarded during inference. Further improvements like batch-normalization [32] and the introduction of design principles for scaling up convolutional networks efficiently [33], lead to the Inception-v2 and Inception-v3 architectures, respectively. Xception (EXtreme Inception) byChollet [34] uses depth-wise separable convolution, i.e. a depth-wise convolution performed independently on each input channel, followed by a point-wise (1×1) convolution. This boosts the accuracy again with respect to Inception-v3, in a direct comparison by [34], Inception-v3 achieves a 5.9% single-crop, single-model top-5 accuracy and Xception achieves 5.9% single-crop, single-model top-5 accuracy while having even less parameters.

Residual Networks

Residual Networks, or ResNets by He, Zhang, Ren, *et al.* [17] set a new benchmark by winning all five categories in the 2015 ILSVRC. They represent another breakthrough in deep learning. They address a fundamental problem in deep networks: performance degrades when adding more layers, which can not be attributed to overfitting but is rather caused by

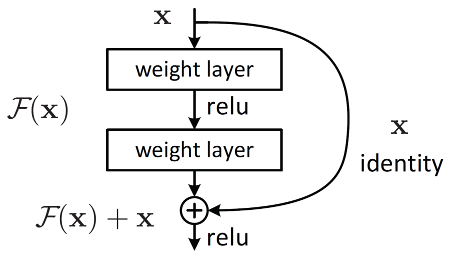


Figure 3-2: A Residual Unit, building blocks for deep residual networks. Source: ResNet [17]

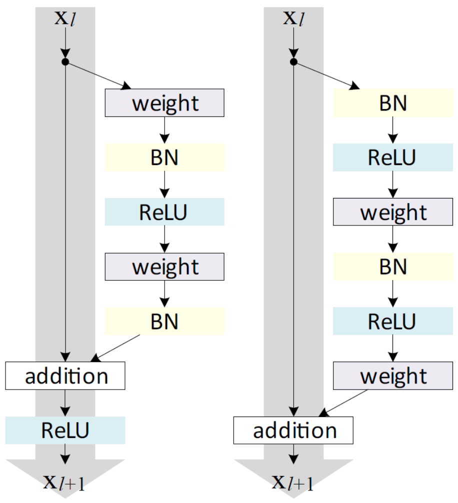


Figure 3-3: Left: Original Residual Unit [17]. Right: Modified Residual Unit. Source: He, Zhang, Ren, *et al.* [35]

vanishing gradients. The loss signal back-propagated through the layers gets weaker towards the bottom layers because of multiplication with the gradients of each layer. This problem can be solved by *residual units*, that map the input of a layer to its output by a *skip connection* or *short-cut* (see Figure 3-2). The gradients can then bypass a layer during back-propagation. This allows increasing the accuracy by adding more layers to the network. Their 152-layer ResNet ensemble has a top-5 error rate of 3.57% on the ImageNet test set. The general form of a residual unit can be expressed as:

$$\begin{aligned} y_l &= h(x_l) + \mathcal{F}(x_l, W_l) \\ x_{l+1} &= f(y_l) \end{aligned} \tag{3-2}$$

where $h(x_l) = x_l$ is an identity mapping. $\mathcal{F}$ is a learned residual function (one or more conv-layers) and f is the ReLU activation. He, Zhang, Ren, *et al.* [35] find that the identity mapping is indeed the most optimal choice for $h(x_l)$ in terms of training loss and error. They also modify the residual unit design so that it performs activation before weight multiplication and completely inside the residual path (Figure 3-3). The modified expression for the residual unit is:

$$x_{l+1} = x_l + \mathcal{F}(x_l, W_l) \tag{3-3}$$

By recursion of Equation 3-3, one can obtain, for any deeper (higher) layer L and any shallower (lower) layer l:

$$x_L = x_l + \sum_{i=l}^{L-1} \mathcal{F}(x_i, W_i) \tag{3-4}$$

In this residual architecture, the conv-layers in $\mathcal{F}$ do not transform the features x directly, but return modifications that are added to x. This makes it even easier for gradients to flow through the layers to lower layers. Equation 3-5 shows how the gradient of loss $\mathcal{E}$ back-propagates to layer l:

$$\frac{\partial \mathcal{E}}{\partial x_l} = \frac{\partial \mathcal{E}}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial \mathcal{E}}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=l}^{L-1} \mathcal{F}(x_i, W_i)\right) \tag{3-5}$$

It can be seen that the gradient $\frac{\partial \mathcal{E}}{\partial x_L}$ directly affects layer l without being multiplied with any weight layers in between layer L and l, so that gradients will never vanish due to small weights. Note that this also makes the use of auxiliary classifiers, like in GoogLeNet, obsolete. Both the Inception and ResNet architectures do not use any fully-connected layers before the final output and softmax layer. This greatly reduces the number of parameters. The output layer operates directly on the last feature map of the convolutional part, possibly with a pooling layer in between.

The creators of GoogLeNet add residual units to their Inception architecture in Inception-ResNet-v1 and Inception-ResNet-v2 for an even higher performance [36]. At the same time, the creators of ResNet propose ResNeXt [37], a ResNet evolution with elements inspired by the Inception architecture. Inception-ResNet-v2 and ResNeXt-101 further decrease the ImageNet validation error to around 3% and are on par with the Inception-v4 architecture. The main advantage of the residual connections in Inception-ResNet-v2 compared to Inception-v4 is the faster training speed, i.e. faster convergence.

Dense Convolutional Networks

Another evolution of skip connections is implemented in the DenseNet architecture by Huang, Liu, Van Der Maaten, *et al.* [38]. Within a so called *dense block* all convolutional layers produce feature maps of the same size. Each layer takes the feature maps of *all* preceding layers concatenated with each other as input. The difference to a skip connection in a residual unit is, that the feature maps are not merged with later feature maps by addition but by concatenation, therefore preserving much more of their information. An example of a DenseNet is visualized in Figure 3-4. The dense architecture allows layers to directly reuse features from

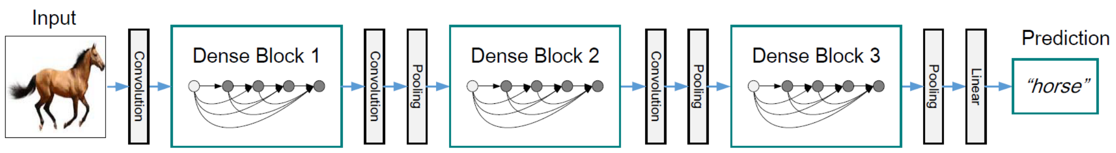


Figure 3-4: Example of a DenseNet with three dense blocks. Between the dense blocks, so called *transition layers* (conv and pooling) are placed that down-sample the resolution of the feature maps. Within a dense block, the resolution is constant such that each layer can take the feature maps of all preceding layers as input. Source: DenseNets [38]

previous layers. As a consequence, the model becomes more compact and fewer parameters and thus fewer computational resources are needed for the same performance compared to the ResNet architecture. As more layers are added, accuracy consistently improves without overfitting. The authors compare the ImageNet performance of DenseNets with ResNets, both trained on the same data preprocessing and with the same hyperparameters optimized for ResNet. They show that a DenseNet-201 with 20 million parameters achieves the same validation single-crop top-1 accuracy of 22.5% as a ResNet-101 with 40 million parameters. Further improvements can be made by optimizing the hyperparameters for the DenseNet models.

Residual units, skip connections and inception modules form the basic building blocks of state-of-the-art recognition models. They have not only been used for classification, but also for other visual recognition tasks like object detection (Chapter 4) or segmentation [39], [40].

3-3 Summary

Image Recognition is a fast moving field in Computer Vision. Challenges like the ILSVRC [3], [23] steadily put forth new state-of-the-art recognition models. A subset of these models is discussed here, starting with AlexNet [2] and ending with DenseNets [38]. The trend not only has been going towards more layers, but also towards applying tricks to overcome problems in training, overfitting and improving gradient flow and speed. Table 3-1 shows an overview of CNN models presented in this chapter. The error is the top-5 test error on the ImageNet test set. The top-5 error rate is the rate of predictions where the correct class is not in the five highest scoring classes of the network output. The reported performance on the very large ImageNet dataset, that is also more complex than our own data in terms of numbers of classes and diversity, indicate that these CNN-models could achieve high performance on the problem at hand.

Designing and training a completely new network architecture is not the goal of this project. Therefore, one of the above network architectures will be chosen as a baseline and will be extended with methods discussed in the subsequent chapters. Many models, even with pre-trained weights on large datasets like ImageNet, are available on-line for the various Machine Learning frameworks like `keras` [41], `tensorflow` [42] and others.

As the dataset size is limited and deep networks with many parameters are prone to overfitting, it is desirable to chose a shallow network, that can grow with the size of the dataset and has a high parameter efficiency. The DenseNet architecture focuses attention on design choices that specifically exploit feature reuse and skip connections to decrease the number of parameters and will be chosen as a classification baseline for this project. Additionally, ResNet-50, a network often used as a baseline for research, will be used as a second baseline. Once an initial classification baseline is established, it is possible to scale up by add more layers or performing architectural tweaks.

Ultimately, a baseline convolutional model performing well as classifier can be reused as a backbone for Object Detection, which is discussed in the next Chapter. This can be achieved by adding additional heads on top that produce the relevant outputs, e.g. scalars to regress the location and size of bounding boxes containing instances of objects.

Table 3-1: Overview of a selection of CNN models. The error is the best reported top-5 error in single-models and ensembles on the ImageNet-1k test set. Single-model results should not be compared one-to-one because they are obtained on different numbers of image crops.

Model	**Main Characteristic**	**Layers**	**Params**	**ImageNet Single-Model Error**	**ImageNet Ensemble Error**
AlexNet [2]	ReLU activation, large scale GPU training	8	60 million	18.2%	15.3%
ZF-Net [29]	Hyperparameter tuning	8	60 million	16.0%	14.8%
VGG-19 [30]	Homogeneous architecture	19	144 million	7.0%	6.8%
GoogLeNet [31]	Inception modules reducing parameters	22	5 million	7.89%	6.67%
Inception-v3 [33]	Inception modules reducing parameters	42	24 million	4.2%	3.58%
Xception [34]	Depth-wise separable convolutions reducing parameters	36	23 million	5.5%[2]	-
ResNet-152 [17]	Residual units preventing vanishing gradients	152	60 million	4.49%	3.57%
Inception-ResNet-v2 [36]	Combining residual and inception units	-	54 million	3.7%	3.08%
ResNeXt-101 [37]	Combining residual and inception units	101	44 million	3.7%	3.03%
DenseNet-201 [38]	Dense blocks reducing parameters and preventing vanishing gradients	201	20 million	5.54%[2]	-

[2]These models did not directly participate in the ILSVRC and are tested only with single crops of the test images, in a single-model configuration, while the other models are tested with multiple crops in both single-models and ensembles.

Chapter 4

Object Detection Architectures

This chapter gives an overview of object detection algorithms, their historical context and their components. Object detection is defined as locating and classifying certain object instances in an image, such as faces, cars, animals, etc. or in this case, damages on an aircraft. An object detection algorithm therefore consists of a recognition part and a locating part. Image recognition using Convolutional Neural Networks (CNNs) is discussed in Chapter 3, while this chapter treats the integration into an object detection model where a locating part is added to the architecture. All of the following approaches are built around CNNs, which are called the *backbone networks* of the detection architecture. Object detection with classical (non-deep learning) Computer Vision models using hand-engineered features is not discussed in this report. A short overview of the most popular algorithms before Deep Learning can be found in Agarwal and Ogier [43].

Agarwal and Ogier [43] review the advances in object detection with neural networks. The early batch of models developed for detection used a staged approach, where the first stage selects Regions of Interest (RoIs) and the second stage does further classification and location regression. Staged detection architectures are discussed in Section 4-1. More recent models use a unified, single-stage approach, discussed in Section 4-2.

Training and evaluation of object detection models is usually done on the Microsoft COCO dataset [44] or the Pascal VOC dataset [45], [46]. MS COCO has 2.5 million labelled instances of 91 object types in 328,000 images. The Pascal VOC 2012 dataset has 20 classes and 11,530 training/validation images containing 27,450 RoI annotated objects and 6,929 segmentations. The most common evaluation metric for object detection is the mean average precision (mAP) at a predefined Intersection-over-Union (IoU). IoU is the percentage of overlap for a predicted bounding box and a ground-truth bounding box. For Pascal VOC, the mAP is the mean of the precision averaged over all classes at IoU > 0.5. For MS COCO, the mAP is again the mean of the precision averaged over all classes and additionally averaged over IoU-values ranging from 0.5 to 0.95.

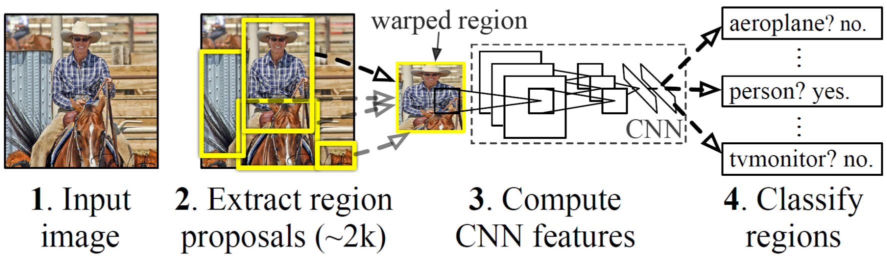


Figure 4-1: Region-based Convolutional Neural Network (R-CNN) Architecture. Region proposals are generated using selective search. Extracted regions are resized and fed to a CNN classifier. Source: Girshick, Donahue, Darrell, *et al.* [47]

4-1 Staged Detection Architectures

A very naive approach to detection of object instances in an image is to slide a (CNN-) classifier across the image and return the regions with a high confidence for certain classes according to some threshold. This method is quite expensive, as the CNN has to be evaluated at every pixel location in the query image.

Instead, it makes more sense to first select Regions of Interest (RoIs) and then classify on them. R-CNNs by Girshick, Donahue, Darrell, *et al.* [47] employ exactly such a staged approach: First, a selective search algorithm [48] proposes RoIs in an image. The image is then cropped to that RoI and resized, which is called *RoI-pooling*. Next, the resized RoI proposal is fed to a CNN-classifier which returns the class of the potential object in the RoI. This has to be repeated for every proposed RoI. In addition to the K classes, the classifier also has to detect falsely proposed regions, which are rejected by assigning the RoI to a catch-all or background class. The total number of classification outputs then becomes $K + 1$. Figure 4-1 shows a schematic of R-CNN.

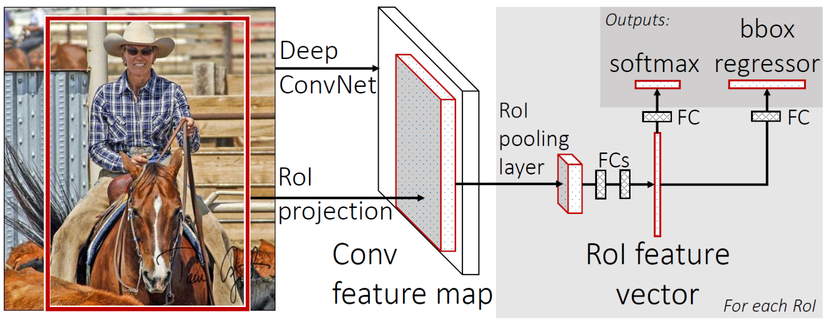


Figure 4-2: Fast R-CNN Architecture. The image is passed through a deep CNN. Region proposals are projected on a conv-feature map and extracted. The RoI feature vector is fed to a fully connected classification head and a bounding box regression head. Source: Girshick [49]

Although already better than naive detection, the CNN still has to perform a complete forward pass at every RoI. The successor, Fast R-CNN by Girshick [49], improves speed by feeding the complete query image to the CNN and then extracting the RoIs from a feature map of a higher CNN-layer. This follows the idea that low-level learned features extracted for classification are generalizable to other tasks (see Section 7-2). The classification and bounding box regression can then be performed directly on that patch of the feature map (Figure 4-2).

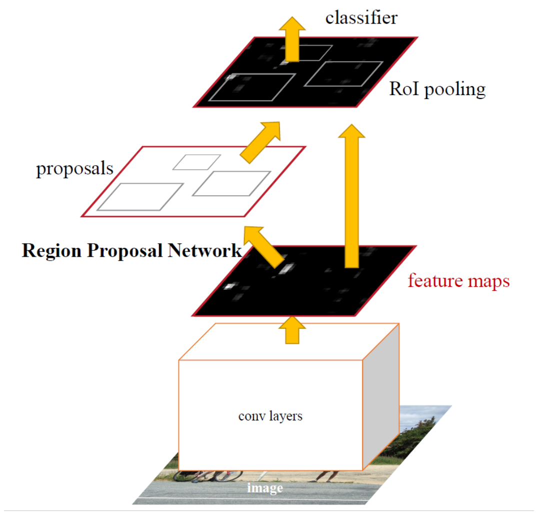


Figure 4-3: Faster R-CNN Architecture. The image is passed through a deep CNN. A Region Proposal Network (RPN) creates region proposals on a conv-feature map. The regions are extracted and fed to the classifier. Source: Ren, He, Girshick, *et al.* [9]

Faster R-CNN by Ren, He, Girshick, *et al.* [9] further improves speed by replacing the selective search algorithm by a neural network based method, called Region Proposal Network (RPN). The RPN slides across the feature map of a CNN to propose RoIs (Figure 4-3). It is a small network consisting of an intermediate, fully-connected layer and a two parallel, fully connected layers, one for box-regression (*reg*) and one for objectness classification (*cls*). At each position, the RPN predicts bounding boxes and objectness for A possible anchors. The *reg*-layer therefore has $4A$ outputs and the *cls*-layer has $2A$ outputs. Training of the faster R-CNN architecture is done by alternating between training the RPN and fine-tuning the backbone network. The backbone is initialized with a model pre-trained on ImageNet.

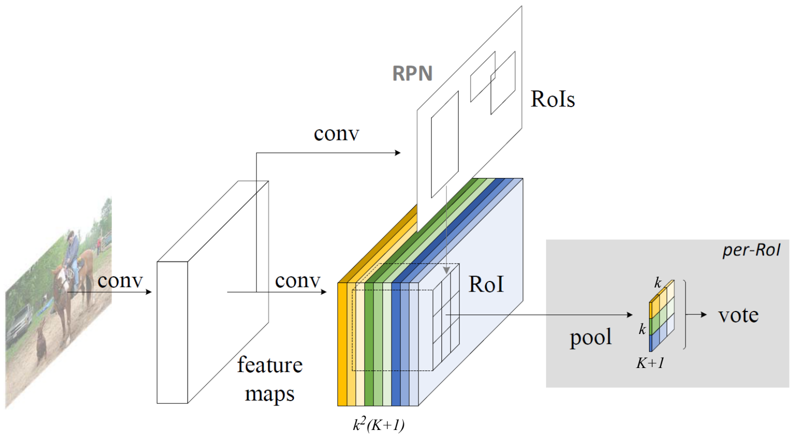


Figure 4-4: Region-based Fully-Convolutional Network (R-FCN) Architecture. An RPN proposes RoIs, which are then pooled from the k^2 score maps. RoI-pooling is position sensitive; the RoI of each score map is pooled into one grid cell. The final classification output is a global average vote on the grid. Source: R-FCN [50]

Dai, Li, He, *et al.* [50] propose the R-FCN. It is a two-staged approach that removes all fully-connected layers to reduce parameters and increase speed. The idea is that a RoI can be divided into a $k \times k$ grid. Each cell in that grid contains the location dependent scores of the K objects plus background. A convolutional layer that produces a feature map of the complete image with $k^2(K+1)$ channels is appended to the backbone CNN. A *score map* is part of that feature map and consists of $K+1$ channels. Each score map corresponds to one cell in the $k \times k$ RoI. An RPN proposes regions, which is followed by *position-sensitive* RoIs-pooling. This means that pooling is performed on each score map independently, and each score map is (average-)pooled into a cell of the $k \times k$ grid for each RoI. The output of that process is a $k \times k \times (K+1)$ tensor. Next, the average score across the grid for each class is taken which results in a $(K+1)$-dimensional vector that has the final class scores for the RoI. Figure 4-4 shows the R-FCN architecture.

Similarly, bounding box regression for each RoI is performed: a $4k^2$-dimensional convolutional layer is appended to the backbone in parallel to the score conv-layer. Position-sensitive RoI-pooling then gives a $k \times k \times 4$ tensor. A 4-dimensional vector that contains the bounding box coordinates for each RoI, is obtained by averaging across the grid.

The reason why training the RPN and the classifier simultaneously is difficult to do is because differentiating the RoI-pooling operation is non-trivial. For unified, end-to-end training, it is beneficial to remove the RoI-pooling layer. In this case, the object detection model becomes a single-stage model, i.e. a model that predicts bounding box locations and classes in one forward-pass simultaneously. This model family is discussed in the next section.

4-2 Single-Stage Detection Architectures

A staged approach for object detection is very intuitive, but the strength in convolutional architectures lies in their capability of performing many calculations in parallel instead of sequentially. Furthermore, a neural network with enough capacity should be trainable for box regression and classification simultaneously in an end-to-end manner, which could speed up inference and make training less complicated. This motivates the creation of single-stage detection architectures.

The You-Only-Look-Once (YOLO) Network proposed by Redmon, Divvala, Girshick, *et al.* [51] is specifically designed for high-speed and can run in real-time on videos, for example. It divides the input image in an $S \times S$ grid and predicts A bounding boxes, confidence for these boxes (objectness score) and K class probabilities for each grid cell. The authors naturally use a GoogLeNet inspired backbone network (see Section 3-3) for high speed and put two fully-connected layers on top of it. The final output is a $S \times S \times (K + 5A)$ tensor. The detection on an input image can thus be performed by one single feed-forward pass through the network.

When compared to a detection model that uses RoI-pooling before classification and box regression, e.g. an R-CNN model, YOLO completely discards region proposals and pooling and can use global information for the direct prediction of classes and locations. The disadvantage is that it can only detect one object per grid cell.

Redmon and Farhadi [52] increase the performance and create YOLO-v2 and YOLO-9000. The latter is noteworthy because it trains on the ImageNet dataset for *classification* in addition to the COCO dataset for detection. It is able to detect more than 9000 different object categories. Redmon and Farhadi [53] propose another incremental speed and accuracy improvement with the YOLO-v3 model.

Single Shot Detection (SSD) by Liu, Anguelov, Erhan, *et al.* [54] adds additional conv-layers that progressively decrease in size and produce feature maps at multiple scales. At each feature map with size $W \times H \times p$, small $3 \times 3 \times p$ kernels are applied to output the class scores and box locations. So a total of $(K + 4)A$ filters are applied to each feature map, where A is the number of anchor boxes. The output is then a $W \times H \times (K + 4)A$ tensor *for each feature map*, compared to the *single* and very compact $S \times S \times (K + 5A)$ output tensor of YOLO.

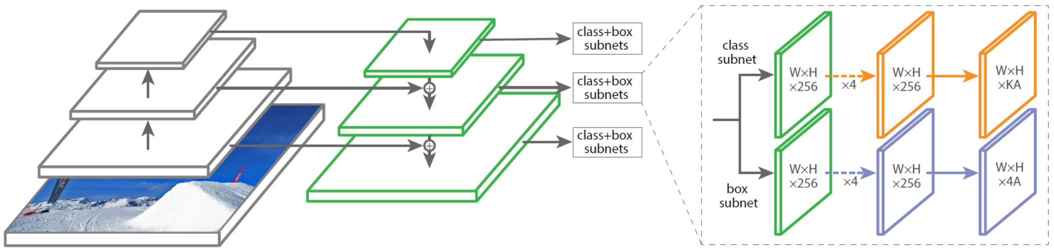


Figure 4-5: RetinaNet Architecture. Left: Backbone feed-forward network (ResNet). Middle: Feature Pyramid Network. Right: fully-convolutional class and box subnets. Source: RetinaNet [55]

While the YOLO models are strongly focused on high speed in real-time detection, a single-stage model that aims to match the accuracy of staged-models is RetinaNet by Lin, Goyal, Girshick, *et al.* [55]. They attribute the inferior accuracy of single-stage models to the imbalance of background to foreground in the input image, which is caused by the sheer number of *dense proposals* at every location in the feature map. The overrepresented background class dominates the loss and therefore the gradients, even though it is easy to estimate. Staged models like the R-CNN family avoid background/foreground imbalance by only proposing certain regions to the classifier, narrowing down the number of candidate object locations and filtering out background, i.e. they use *sparse proposals*. RetinaNet addresses the imbalance by proposing a new loss function called *focal loss*. The (binary) focal loss is a dynamically scaled cross-entropy loss, shown in Equation 4-1.

$$\begin{aligned} p_t &= \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise} \end{cases} \\ \text{FL}(p_t) &= -\alpha_t (1 - p_t)^\gamma \log(p_t) \end{aligned} \tag{4-1}$$

where p is the network output, y is the ground-truth label and α and γ are scaling parameters. The scaling factor $(1 - p_t)^\gamma$ gives classes with a high confidence less weight.

RetinaNet uses a ResNet backbone, together with a Feature Pyramid Network (FPN) [56]. An FPN is a network structure that creates multi-scale feature pyramids. It rejoins lower resolution feature maps of higher layers in the backbone network to higher resolution feature maps of lower layers by up-sampling. This induces scale invariance in the detection model and makes it particularly strong for small object detection.

Figure 4-5 shows the RetinaNet architecture. The FPN extracts feature maps from different layers of the backbone, up-samples them and adds them to the feature map of a previous layer.

On every feature pyramid layer, two subnets perform prediction of bounding box locations and class labels, respectively. The subnets are fully convolutional and therefore invariant to the input size. Therefore, the subnets can share weights across the pyramid layers. The subnets predict boxes and classes on different scales. RetinaNet also works with anchors. The number of anchors is A, each with a different aspect ratio and scale. For each anchor, K class probabilities and 4 bounding box regression values are predicted at each spatial location. The class subnet then outputs a tensor of $W \times H \times KA$ and the box subnet outputs $W \times H \times 4A$ for each feature pyramid layer. Non-max suppression removes all overlapping bounding box proposals with the same predicted class that do not have the highest confidence for that class.

4-3 Summary

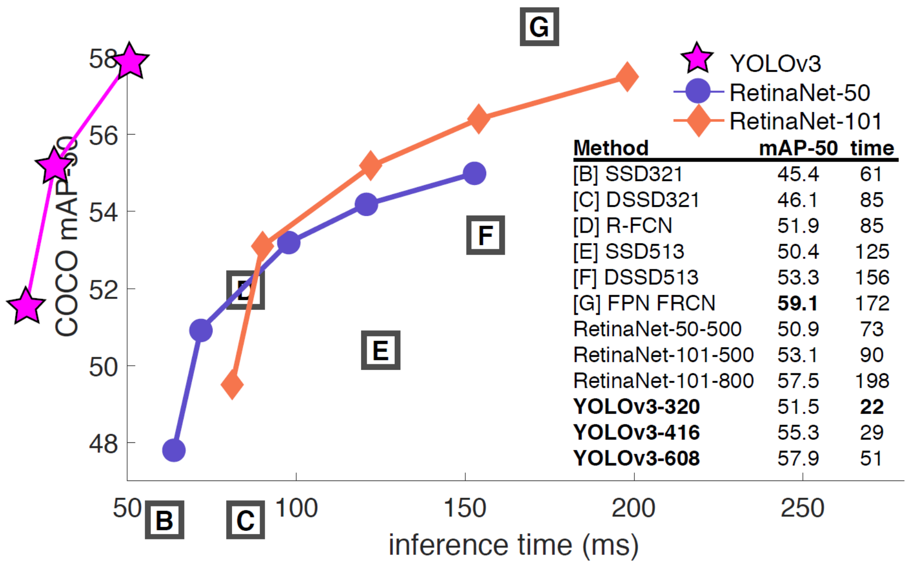
Object detection models can be divided into two categories: Staged models and single-stage models. Staged models, often also called region-based models, perform region proposals first and subsequently classification and bounding box regression. The R-CNN [9], [47], [49] family is the most well-known family of staged models. Its latest version, the Faster R-CNN architecture [9] performs region proposals with a Region Proposal Network (RPN), extracts and resizes the RoI to be fed to a classifier and bounding box regressor. The R-FCN architecture [50] is another staged approach, which replaces the fully-connected classification and regression heads by fully convolutional heads and position-sensitive RoI-pooling for increased speed.

Staged models have the disadvantage of being slow and complicated to train. Single-stage models are unified models that perform classification and localization end-to-end simultaneously and require only a single forward pass. The YOLO family [51]–[53] of models is designed for maximum speed and can be used for real-time applications. It predicts the anchor box location and class for each cell in a fixed grid on the input image.

SSD [54] is not restricted to a fixed grid. Instead, it predicts the anchor box locations and classes on multiple feature maps of varying scale at each pixel location, yielding a higher mAP than YOLO-v1 and YOLO-v2 but at a reduced speed. YOLO-v3 also uses multi-scale feature maps amongst other improvements, which gives it an advantage over the original SSD. RetinaNet [55] uses a more advanced version of multi-scale features called Feature Pyramid Network (FPN) and a loss function that is specifically designed to increase accuracy for dense box proposals. This brings it on the same accuracy level as staged-methods while still being faster. Though the different name, RetinaNet is essentially an evolution of the SSD architecture.

Table 4-1 presents an overview of the detection models discussed in this chapter. Figure 4-6 shows the speed and accuracy for various models. For the application in damage detection, there is currently no strict requirement for high speed or low computational cost. Important are high accuracy, especially for small objects due to the nature of the damages. Although YOLO-v3 is on par with RetinaNet in terms of mAP @ 0.5, as can be seen in Figure 4-6, RetinaNet performs much better on the mAP for small objects [53]. This makes the RetinaNet architecture the most promising candidate.

It has to be noted, that a convenient property of detection models in general is that they are highly modular. The consequences are that concepts can be interchanged between the



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 4-6: mAP vs. inference time for various detection models. mAP is evaluated on the COCO test set with mAP at IoU=0.5. Source: RetinaNet [55], YOLO-v3 [53]

different architectures, architectures can be easily augmented for more advanced tasks (for example Mask R-CNN [40]), scaled up for improved performance and components (like the backbone) can be updated when better versions become available. This enables a programmer to design for specific applications and requirements.

Table 4-1: Overview of a selection of object detection models. Backbones and mAP are not necessarily consistent with the original papers, but rather represent the best implementation to date for fair comparison. mAPs are for the Pascal VOC 2012 test set (mAP @ 0.5) and the MS COCO test set (mAP @ [0.5:0.95]).

Model	**Backbone**	**Architecture**	**VOC mAP**	**COCO mAP**
R-CNN [47]	AlexNet [2]	selective search + Region of Interest (RoI)-pooling on input image	62.4%	-
Fast R-CNN [49]	VGG-16 [30]	selective search + RoI-pooling on feature map	68.4%	-
Faster R-CNN [17]	ResNet-101 [17]	Region Proposal Network (RPN) + RoI-pooling on feature map	83.8%	34.9%
R-FCN [50]	ResNet-101	score maps + RPN + position sensitive RoI-pooling on score maps	82.0%	-
SSD512 [54]	VGG-16	multiscale feature maps + multiple convolutional regression and classification heads	80.0%	26.8%
YOLO-v1 [51]	GoogLeNet [31]	fixed-grid + fully-connected regression and classification head	57.9%	-
YOLO-v2 [52]	Darknet-19 [57]	fixed-grid + convolutional regression and classification head	73.4%	21.6%
YOLO-v3 [53]	Darknet-53	fixed-grid + multiscale feature maps + multiple convolutional regression and classification heads	-	33.0%
RetinaNet [55]	ResNeXt-101 [37]	Feature Pyramid Network (FPN) [56] + multiple convolutional regression and classification heads	-	40.8%

Chapter 5

Class Imbalance Methods

When training models for classification, the samples are rarely perfectly distributed across classes and can strongly over-represent some classes (*majority classes*) above other classes (*minority classes*). This chapter gives an overview of methods dealing with class imbalances. Haixiang, Yijing, Shang, *et al.* [58] provide an extensive review of the latest research concerning the class imbalance problem, while Buda, Maki, and Mazurowski [10] choose representative methods from each category applied to neural networks and compare them against each other.

There is a rich variety of methods in the field, but this chapter assesses the methods for use in neural network training. Before presenting them, performance metrics for imbalanced classifications are established in Section 5-1. Then, the two categories for imbalanced learning are discussed: firstly, methods that operate on the data-level (Section 5-2), i.e. perform sampling and possibly additional operations on the training data to equalize class probabilities and secondly, methods that operate on the algorithm level (Section 5-3), i.e. perform different learning or prediction operations depending on class probability. The findings are summarized in Section 5-4.

Especially in applications where rare, but catastrophic anomalies have to be detected, it is critical that detection performance is not degraded by class imbalances in the training data. Reasons for degradation can be the choice of cost function and performance metric. For example, during model parameter optimization, the cost function gradients will be strongly influenced by the majority class. Also, overall accuracy as metric biases towards the majority class. Other reasons are that minority classes are difficult to learn because they are not easily separable from other classes especially when feature dimensionality is high, or the minority classes are indistinguishable from noise. Also, the influence of imbalance is more profound with more complex data [10].

5-1 Performance Metrics

Before going into detail, a performance metric for imbalanced classification has to be established. Overall prediction accuracy will not give satisfying results, which can be shown with a

Table 5-1: Confusion Matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

simple example: If we have a binary classification problem with class 0 having 990 examples and class 1 having 10 examples, the classifier can easily achieve a 99% accuracy on the training set by always guessing class 0, and therefore always predicting the wrong label for class 1. A better metric is to look at the confusion matrix (Table 5-1) and defining *precision* or *Positive Predictive Value (PPV)*, which is the proportion of positive predictions for a certain class that are actual positives:

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5-1}$$

The *recall* or *True Positive Rate (TPR)*, is the proportion of actual positives that are also predicted as positives:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5-2}$$

In the above example, the precision and recall for class 1 would be 0%, and the precision and recall for class 0 would be 99% and 100%, respectively. Additionally, the *False Negative Rate (FNR)* is the proportion of actual positives that is predicted as negatives:

$$\text{FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}} = 1 - \text{TPR} \tag{5-3}$$

In safety critical applications, it is desired to have a high recall for the critical (minority) class, in other words, minimize the FNR of the anomalies or damages. The *fall-out* or *False Positive Rate (FPR)* is the proportion of actual negatives that is predicted as positives:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{5-4}$$

The Receiver Operating Characteristic (ROC) curve (Figure 5-1) plots the recall against the fall-out, calculated as a function of a classifier parameter, for example a decision threshold. It allows to trade-off recall and fall-out, i.e. minimizing the misclassified anomalies at the cost of a higher number of false positives. The ideal point of the ROC is (0,100). Moving to that point is equivalent to maximizing the area under the ROC curve, or AUC.

5-2 Data-Level Methods

The data-level methods are methods that are applied prior to learning, and perform some pre-processing on the training data. The dominant technique for pre-processing is resampling. Resampling aims at balancing the class distributions in the training data. The options are oversampling and undersampling. Oversampling creates artificial examples of the minority class and undersampling leaves out examples of the majority class. A combination of these are referred to as hybrid methods.

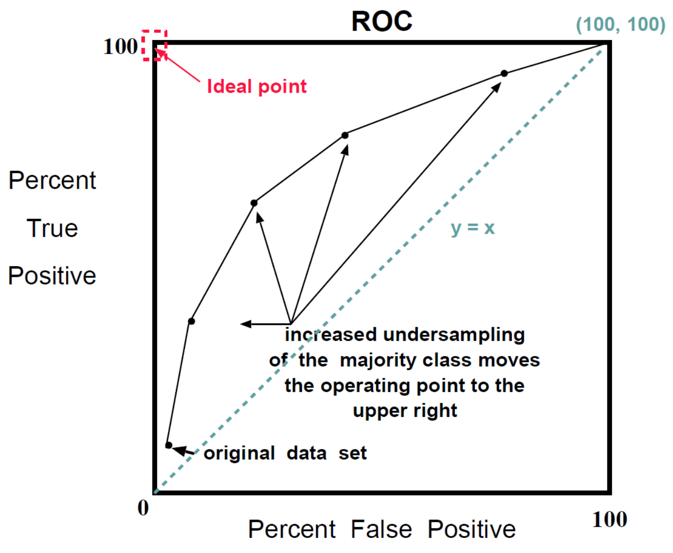


Figure 5-1: ROC curve. Source: SMOTE [59]

Haixiang, Yijing, Shang, *et al.* [58] have found that resampling in general and oversampling in particular are the most popular choices for class imbalance techniques, Synthetic Minority Over-Sampling Technique (SMOTE) [59] is commonly used when only few minority examples are available. SMOTE operates in feature space, not in data space. Synthetic examples $x^s_{i,j}$ are created by interpolating between a data point of the minority class x_i and one of its k neighbours x_j in the minority class:

$$x^s_{i,j} = \lambda x_i + (1-\lambda)x_j, \quad \forall j = 1, ..., k, \quad \lambda \sim \mathcal{U}(0,1) \tag{5-5}$$

By interpolating between examples, SMOTE aims to overcome the issue of overfitting that is present when only replicating samples of the minority class. Figure 5-1 qualitatively shows the effect SMOTE has on the ROC curve. Many other variations of SMOTE and oversampling exist that will not be discussed in detail here. One worthy mention, specifically for neural networks and other learning algorithms that can be trained with Stochastic Gradient Descent (SGD), is class-aware-sampling [60], that samples a uniform class distribution for every mini-batch when calculating the gradients.

The other resampling technique, undersampling, randomly removes samples of the majority class for training to balance the class distribution for training. However, the disadvantage is that data will be discarded that can be useful in other ways for training. Modifications of undersampling 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.

The work of Havaei, Davy, Warde-Farley, *et al.* [6] is an interesting real-world application of training a Convolutional Neural Network (CNN) on imbalanced data: The dataset for their brain tumour segmentation network consists to 98% of healthy voxels. They address the imbalance by dividing the training phase into two steps. The first training step samples patches from the brain images such that each label is equally represented in the training data. The second step freezes all layers except for the output layer and retrains the output layer on the natural distribution of labels.

5-3 Algorithm-Level Methods

Algorithm-level methods are applied during training and testing time and change the classifier behaviour dependent on the class.

5-3-1 Cost-Sensitive Methods

An example of an algorithm-level method that is applied during training time is cost-sensitive learning: misclassification of a minority class sample is assigned a higher cost than misclassification of a majority class sample.

Focal loss as used for RetinaNet [55], dynamically scales the cross-entropy loss with a factor that gives confident predictions less weight. It is specifically developed for object detection (see also Section 4-2) and mitigates the effect of foreground/background class imbalances in dense single-stage detectors, where background is not filtered by region proposals.

Machine Learning frameworks like `keras` [41] and `scikit-learn` [62] have interfaces to assign scalar weights to each class in the estimator, which makes basic cost-sensitive training straightforward to implement. Note that assigning different weights on the cost for different classes is equivalent to resampling; using more samples of a class means it has a higher weight in the total cost.

More complex cost assignments are possible, for example cost matrices C with that assign cost $C_{i,j}$ to misclassifying class i into class j. These cost matrices are more difficult to set up and may require expert knowledge. Khan, Hayat, Bennamoun, *et al.* [63] solve this problem by introducing a novel cost matrix ξ on the network output. The cost function parameters and neural network parameters are jointly learned from data.

5-3-2 Thresholding

Thresholding is an algorithm-level method that is applied during testing time. In general, the threshold can be set to optimize for any criterion, but it can also be used to compensate for prior class probabilities. It assigns different thresholds to classes with different prior probability $p(y)$. If the training data has different class probabilities than actual class probabilities, dividing the network output by the training data class probability and multiplying by correct class probabilities (if available) as the network outputs posterior probabilities $p(y|x)$ [64]. This is particularly useful after training a classifier with a re-sampled dataset.

5-3-3 Ensemble Methods

Ensemble methods use multiple classifiers in parallel to increase classification accuracy. The idea is that an ensemble of weak classifiers combined creates a strong classifier that outperforms the weak classifiers. Each classifier is trained on a different re-sampled set of the training data to create diversity in the classifiers that will improve overall performance. AdaBoost (adaptive boost) [65] and Bagging (bootstrap aggregating) [66] are widely used algorithms for ensemble learning.

Bagging is a simple form of ensemble learning where for each classifier, a new dataset is sampled with replacement from the original dataset, possibly keeping the number of samples in each dataset the same. This introduces diversity in each classifier. Then the classifiers are trained on their respective datasets and during testing, some form of majority voting is performed to reach a final decision.

AdaBoost trains each classifier on the whole dataset, serially instead of in parallel. Initially, the classifier assigns equal weights to the training instances but increases the weights for misclassified instances and decreases them for correctly classified instances after each training round. The next following classifier is then trained with the new instance weights. Each classifier itself also gets a weight according to its overall accuracy. During testing, a weighted vote is given by each classifier.

Galar, Fernandez, Barrenechea, *et al.* [67] review ensemble methods that are specifically designed for imbalanced training by combining them with techniques presented above. They find that ensembles clearly improve the performance when compared to a single classifier trained with data pre-processing on imbalanced data. Ensemble learning in combination with random undersampling, more specifically RUSBoost [68] and UnderBagging [69] stand out in their experimental evaluation: although being simple implementations, they perform better than more complex contenders.

The literature research on image recognition in Chapter 3 has revealed that all winning models of the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [3] use ensembles. These ensembles are trained (and initialized) independently of each other, which offers some percentage points increase in validation performance. In applications where every bit of performance counts, this is a viable option to be considered.

5-4 Summary

In this section, the suitability of both data-level and algorithm level methods for neural networks is summarized:

In Deep Learning, it is desirable to use as much data as possible to avoid overfitting. In this setting oversampling is preferred above undersampling. Oversampling for classification is a rather simple, proven heuristic and does not necessarily lead to overfitting when used with neural networks and can be used best with thresholding to compensate for the re-sampled training data class probabilities [10]. Oversampling is a member of a wider family of methods that aim to increase dataset size. These data augmentation methods are discussed in Chapter 6.

Simple cost-sensitive training can be implemented with class weights obtained from empirical class distributions in the training data. It is more computationally efficient than resampling. More sophisticated cost methods exist with dynamic cost assignment [63] and can be combined with ensemble methods, for example [65].

Ensemble methods increase the performance of single classifiers on imbalanced data but have the disadvantage that multiple networks 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 check-points or initializations that have to be created anyway. Dropout regularization can be seen as a weak form of combining randomly sampled networks for increased performance [21].

The takeaway of this chapter is that some form of oversampling, compared against or combined with cost-sensitive methods can work well with the problem at hand. Synthetic samples of the minority class could be created using a SMOTE variation or other forms of data augmentation (see Chapter 6). To gain some additional performance, multiple classifiers can be combined into an ensemble.

Chapter 6

Data Augmentation

Throughout this report, the importance of large datasets for training a Deep Learning model has been stressed multiple times. A large, diverse dataset reduces the risk of overfitting, increases generalization capability and reduces the variance in the classifier performance (i.e. different training runs with different initializations are more likely to converge to the same performance). This chapter discusses methods that aim at increasing the size of the dataset by augmenting it with artificially generated new samples. For example, when "natural" data, from the domain of interest, is expensive or impossible to obtain. The focus lies on augmentation techniques for visual data, i.e. images.

Section 6-1 discusses the classical data augmentation techniques that use hand-engineered rules or heuristics, where Section 6-2 discusses the augmentation techniques that are based on (deep) generative models. The generative models try to approximate the distribution of the natural data generating process, such that the augmented dataset resembles the actual distribution as close as possible. Section 6-3 summarizes the findings on this chapter.

6-1 Classical Augmentation Techniques

In this report, the term *classical augmentation* refers to augmentation techniques that are based on hand-engineered, pre-defined rules or heuristics that perform label-preserving transformations to the raw data. Krizhevsky, Sutskever, and Hinton [2] use two forms of augmentation for their AlexNet model: The first is generating random image translations and horizontal flips or reflections to make the model invariant against translational changes. The second is randomly changing the intensities of the RGB-channel along the first principal component axis to make the model invariant against colour and illumination changes. By these transformations, they increase their dataset size by three orders of magnitude.

Howard [70] investigates additional augmentation techniques to improve Convolutional Neural Networks (CNNs). They use a different cropping and resizing technique to use all of the given information in one image and apply additional contrast, brightness and colour transformations. They also perform training and predicting at varying image resolutions, which introduces scale invariance to the model.

Another way of augmenting data for classifiers that have a separate feature extractor and classifier, is to apply transformations in feature space, similarly to Synthetic Minority Over-Sampling Technique (SMOTE) [59], where synthetic samples are created by linear interpolations in feature space. However, Wong, Gatt, Stamatescu, *et al.* [71] find that performing known transformations in data space (if available) outperforms transformations in feature space. On a side note, their experiments also show that improvements made by augmentations are bound by adding real data to the training process. In other words, obtaining real images beats creating synthetic images.

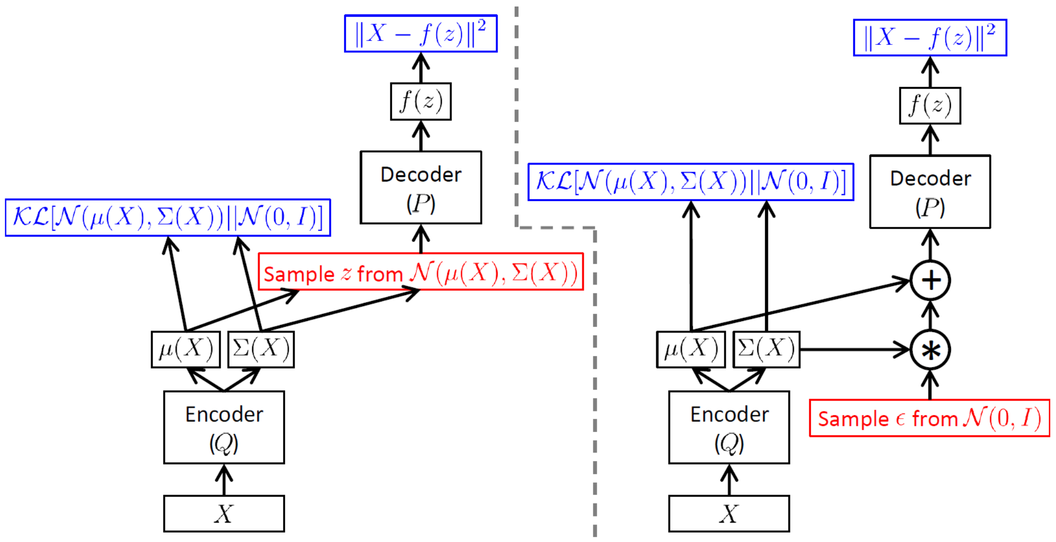
It is interesting that in the Deep Learning era, classifiers based on hand-engineered features are categorically rejected as models that learn features directly from data perform better (Chapter 3), but that the data used to train these models is heavily processed and augmented by hand-engineered transformations. Cubuk, Zoph, Mane, *et al.* [72] very recently propose a method to indeed learn the augmentations automatically from data called AutoAugment. They aim 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 [73]. The policies are selected by a Recurrent Neural Network (RNN) that is trained on a training set with a Proximal Policy Optimization Algorithm [74]. The reward signal returned to the policy learner is the performance improvement of a child model (the model that does the actual classification or regression) on the validation set that is kept separate from the training set.

`keras` [41] and `tensorflow` [42] and other Machine Learning frameworks have built-in functions for classical augmentation techniques that are ready to use.

6-2 Augmentation Techniques based on Generative Models

Although classical augmentation techniques as described in Section 6-1 work well in practice to reduce overfitting, there are some shortcomings: The transformations performed in practice are merely heuristics and resemble an arbitrary selection of rules. Also, it seems that the chosen transformation somewhat depend on the creativity of the programmer, if (s)he misses or overlooks a possible transformation that could be present in the data generating distribution, the learning model will not be invariant to that transformation. Preventing overfitting, or increasing generalization capability, which is the ultimate goal here, means capturing the true distribution of data that the model will encounter during training and testing, including any transformations that could occur. The premise of this section is that there possibly exists an algorithm that can approximate the true data generating distribution $p(x)$. *Generative models* do not only attempt to estimate the conditional distribution $p(y|x)$ but the joint distribution $p(y, x) = p(y|x)p(x)$.

Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) are such models parametrized as (deep) neural networks that were proposed to create artificial images



$\|X - f(z)\|^2$
$f(z)$
Decoder (P)
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)]$
Sample z from $\mathcal{N}(\mu(X), \Sigma(X))$
$\mu(X)$
$\Sigma(X)$
Encoder (Q)
X
$\|X - f(z)\|^2$
$f(z)$
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)]$
Decoder (P)
$+$
$\mu(X)$
$\Sigma(X)$
$*$
Encoder (Q)
Sample ϵ from $\mathcal{N}(0, I)$
X

Figure 6-1: A Variational Autoencoder. Left: Encoding - sampling - decoding. This structure is not fully differentiable, as the gradients can not flow through the sampling operation. Right: VAE with "reparametrization trick" enables back-propagation of gradients through the sampling operation. Source: Doersch [75]

with varying degrees of realism, amongst others. The following two subsections will discuss these models.

6-2-1 Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs), introduced by Kingma and Welling [11], assume that data is generated by a random process. They approximate the data generating distribution by conditioning it on an unobserved latent variable z, where z is sampled from a prior distribution $p_\theta(z)$. So there are two steps in the process: first sampling z from $p_\theta(z)$ and then sampling x from the conditional likelihood $p_\theta(x|z)$. The distributions are parametrized by θ and it is assumed that the PDFs of both prior and likelihood are differentiable with respect to θ and z. This is implemented as the decoding part of the autoencoder in the form of a Multi-Layer Perceptron (MLP).

The encoding part of the VAE, encodes the input x to a distribution of latent variables $z \sim q_\Phi(z|x)$, which approximates the intractable true posterior $p_\theta(z|x)$. $q_\Phi(z|x)$ is also called the *recognition model*. The encoding MLP gives the mean $\mu(x)$ and the variance $\sigma^2(x)$ for the Gaussian distribution $q_\Phi(z|x) = \mathcal{N}(z; \mu, \sigma^2 I)$ as a function of x. So given a data point x, the encoder outputs a distribution over latent codes z, from which x could have been generated.

In turn, given a code vector z, the decoder outputs the distribution $p_\theta(x|z)$ over possible values of x that correspond to z. It is generally also assumed to be a Gaussian.

The cost function to be maximized is the marginal log-likelihood of the data $\log p_\theta(x) = \sum_{i=1}^{N} \log p_\theta(x^{(i)})$, which can be rewritten as:

$$\log p_\theta(x^{(i)}) = \mathcal{D}_{KL}\left(q_\Phi(z|x^{(i)})||p_\theta(z|x^{(i)})\right) + \mathcal{L}\left(\theta, \Phi; x^{(i)}\right) \tag{6-1}$$

the first term is the (always non-negative) Kullback-Leibler divergence [76] between the approximate and true posterior and $\mathcal{L}\left(\theta, \Phi; x^{(i)}\right)$ is the *variational lower bound* on the marginal likelihood. The first term is not known but we can aim to maximize the second term, or lower bound, which is given as:

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}\left(\theta, \Phi; x^{(i)}\right) = -\mathcal{D}_{KL}\left(q_\Phi(z|x^{(i)})||p_\theta(z)\right) + \mathbb{E}_{q_\Phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)}|z)\right] \tag{6-2}$$

The first term of the lower bound is the KL-divergence that regularizes Φ to pull the approximate posterior towards the prior of z and the second term reduces to the negative expected reconstruction error. In the original paper, $p(z)$ is assumed to be the standard normal distribution $\mathcal{N}(z; 0, I)$ for numerical convenience, but it could be replaced by other types of distributions. The terms in Equation 6-2 can be sampled as mini-batches and differentiated using the *re-parametrization trick*. It enables back-propagation through the sampling layer to calculate the gradients by replacing $z \sim \mathcal{N}(z; \mu, \sigma^2 I)$ with $z = \mu(x) + \sigma(x)\epsilon$ and $\epsilon \sim \mathcal{N}(0, I)$, which is now differentiable in μ, σ and x and can be optimized using Stochastic Gradient Descent (SGD).

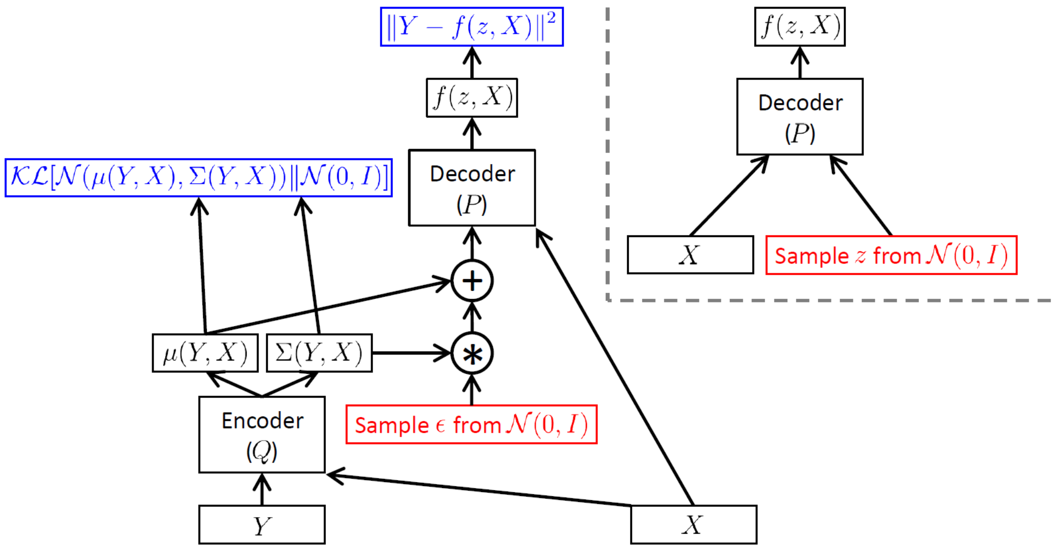
Jorge, Paredes, Sanchez, *et al.* [77] use a VAE to augment training datasets to increase classifier performance. They used the output of the encoder, the latent variable z corresponding to input x, to obtain a distortion $\hat{z}$ in feature space. z is distorted by adding random noise, interpolating and extrapolating. Two approaches are tested:

1. Performing classification in the latent space on $\hat{z}$ (similar to SMOTE)
2. Reconstructing $\hat{x}$ from $\hat{z}$ and subsequently classifying $\hat{x}$, that is, in the data space.

It was found in experiments on the MNIST and UJIPEN datasets using a convolutional autoencoder, that classifying in the data space gives better results while classifying in the latent space gives poor results (consistent with the results of [71]). For an MLP classifier, interpolation gives a slightly better improvement above the baseline (no data augmentation) and for an SVM classifier, extrapolation performs better on MNIST, and interpolation better on UJIPEN. Not surprisingly, for the kNN-classifier, interpolation gives the best results. Adding random noise gave the worst improvement across all the classifiers. The absolute accuracy was best for the MLP, but the other classifiers gained more performance by augmentation. However, the authors do not fully leverage the generative nature of VAEs: They do not generate new data by sampling from a distribution. Instead, they use the encodings of a given data point x and perform "hand-engineered" distortions (interpolation, extrapolation with fixed parameters and adding noise). The same could have been done with a conventional autoencoder.

Conditional Variational Autoencoders (CVAEs)

The problem with a pure VAE as data generator is that the created samples have no labels. Jorge, Paredes, Sanchez, *et al.* [77] ensure label-preservation by conditioning a new distorted example on a true data point, for which the label is known. Creating a VAE that generates conditioned samples could be beneficial for label-preserving data augmentation or generation. Sohn, Lee, and Yan [78] do exactly that by introducing CVAEs for learning structured output representations. A prediction y is conditioned on the input x. The conditional generative



$\|Y - f(z,X)\|^2$
$f(z,X)$
$\mathcal{KL}[\mathcal{N}(\mu(Y,X),\Sigma(Y,X))\|\mathcal{N}(0,I)]$
Decoder (P)
+
$\mu(Y,X)$
$\Sigma(Y,X)$
*
Encoder (Q)
Sample ϵ from $\mathcal{N}(0,I)$
Y
X
$f(z,X)$
Decoder (P)
X
Sample z from $\mathcal{N}(0,I)$

Figure 6-2: A Conditional Variational Autoencoder. Left: A CVAE during training time. The condition x is fed to both encoder and decoder. Right: A CVAE during test time, where conditioned samples are generated. Source: Doersch [75]

process is again modelled by sampling a latent variable z for a given observation x: $z \sim p_\theta(z|x)$. Then the output is generated from $p_\theta(y|x,z)$. The variational lower bound of the likelihood is:

$$\begin{aligned}\log p_\theta(y^{(i)}|x^{(i)}) \geq \mathcal{L}\left(\theta, \Phi; x^{(i)}, y^{(i)}\right) = &- \mathcal{D}_{KL}\left(q_\Phi(z|x^{(i)}, y^{(i)})||p_\theta(z|x^{(i)})\right) \\ &+ \mathbb{E}_{q_\Phi(z|x^{(i)},y^{(i)})}\left[\log p_\theta(y^{(i)}|x^{(i)}, z)\right]\end{aligned} \tag{6-3}$$

Note that x is defined differently than in [11]: Here x is a given condition and independent of z and y takes the role of x in [11]. The authors use CVAEs in experiments to do one-to-many prediction tasks, for example reconstructing MNIST digits from only parts of the image. However, the conditioning is very flexible, it could also be used for classification with the decoding part of the CVAE (many-to-one: predict label y from a given image x and a sampled z), but there is also a use case for conditional data generation by turning the conditioning around, i.e. create an image x, conditioned on label (or attribute) y and a sampled z.

Disentangled Conditional Variational Autoencoders

Yan, Yang, Sohn, *et al.* [79] propose a conditional variational autoencoder that generates samples based on visual attributes of the data. Additionally, they “disentangle” the generation process for the foreground an background of the image. The image is then reconstructed by adding the foreground to the background multiplied element-wise with a gating matrix. The foreground and the gating matrix are conditioned on the attributes y and the latent z, while the background image is only conditioned on z. Experiments with the Labelled Faces in the Wild (LFW) dataset and Caltech-UCSD Birds-200-2011 (CUB) dataset show that this disentangled CVAE (disCVAE) is able to reconstruct and generate more realistic looking images based on attributes than CVAEs and conventional VAEs. The attributes represent additional information that the disCVAE can use for learning, therefore generating better examples.

Beyond simply generating new data, this method has some interesting use cases:

- given an image x, attributes y and its encoding z, generating a $\hat{x}$ with different desired *attributes* y', e.g. augmenting existing data with distortions to increase diversity in the data. Attributes that can be generated are for example image resolution, or filling in missing or occluded parts of the image.
- generating new samples $\hat{x}$ by sampling a z, with certain desired *labels* y'. This enables the generation of examples from a minority class to reduce class imbalances in the dataset.

So there is definitely potential for VAEs to be used as data generators to improve the classifier performance by increasing dataset size. An open area of research is how we can specifically train a VAE to achieve this objective, similar to the approach taken by AutoAugment [72], where augmentation policies are learned on the training data to maximize the performance of a child classifier. The possibilities are for example jointly training the classifier and VAE by back-propagating the classification loss through the classifier input into the VAE and optimizing its parameters. The second possibility is researching distribution types for the prior $p(z)$ different from the standard normal distribution that could improve the classifier generalization capability. Nalisnick and Smyth [80] present a method to learn priors of model parameters for invariance against certain transformations. Although this model is not for learning priors of latent variables, the methods could be possibly transferred to the use in VAEs. Research into different prior types for VAEs has been done by Nalisnick and Smyth [81], Casale, Dalca, Saglietti, *et al.* [82], and Dilokthanakul, Mediano, Garnelo, *et al.* [83].

Another positive side-effect of the encoding-sampling-decoding procedure of VAEs in the context of anomaly (or in our case damage) detection, is that it is possible during prediction to estimate the likelihood of a new sample x. First, x is encoded to z and then its likelihood is estimated with $p(z)$. So for example, even if a discriminative classifier $p(y|x)$ gives a high confidence score for a label y given x, the generative model $p(x|z)$ might return a low likelihood indicating that this sample has not been encountered before and is very unlikely to happen, i.e. it is an outlier. The model then can take special measures to deal with data point x, for example consulting a human expert, which is beneficial in safety critical applications.

6-2-2 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) by Goodfellow, Pouget-Abadie, Mirza, *et al.* [12] were proposed at the same time as VAEs but have a very different architecture: Two networks are competing against each other: The generator network G creates images x from a noise input z. The discriminator network D evaluates if an input x is real or generated by G. Note that although z is called noise here, it serves a similar function as the latent variable z in a VAE, but can not be obtained from encoding x!
G approximates the data generating distribution p_{data} with its own distribution p_g and D

estimates the probability that x is from p_{data} or not. The objective function for training a GAN is given by Equation 6-4:

$$\min_G \max_D V(D, G) = \mathbb{E}_{p_{data}(x)} [\log D(x)] + \mathbb{E}_{p_z(z)} [\log (1 - D(G(z)))] \tag{6-4}$$

Equation 6-4 consists of the following two terms:

- $\mathbb{E}_{p_{data}(x)} [\log D(x)]$: The expected log-likelihood that a real example is predicted real. D maximizes this term.
- $\mathbb{E}_{p_z(z)} [\log (1 - D(G(z)))]$: The expected log-likelihood that a generated example $G(z)$ is predicted fake. D maximizes this term and G minimizes.

Given G and D have enough capacity, this two-player minimax game has an equilibrium solution at $p_g = p_{data}$ and $D(x) = \frac{1}{2}$, meaning that the generator creates images that the discriminator can not distinguish from real images. Training a GAN is done by alternating between training D and G. There are however issues with stability and convergence, and training requires careful selection of hyperparameters. Different methods to address these issues have been proposed [84], [85].

Research in GANs moves towards scaling up with more data and deeper models. BigGANs by Brock, Donahue, and Simonyan [86] is a significantly scaled up GAN that improve both fidelity and variety of the generated examples at a higher resolution than has been attempted so far. For scaling up, they increase the batch size and the width in every layer. They also try out different choices for the latent prior $p(z)$ and show that a truncated normal distribution provides a way to trade off between sample variety and fidelity. The authors also empirically analyse methods to solve training instabilities for large scale GANs, but they conclude that complete stability comes at a significant cost to performance.

Conditional Generative Adversarial Networks (CGANs)

Mirza and Osindero [87] present a proof-of-concept for conditional image generation using CGANs. They extend the conventional GAN architecture by feeding the generator a condition y along with the noise z and also feed y to the discriminator. They show that it is possible to create MNIST digits conditioned on its labels. CGANs have found applications in for example colourization of black-and-white Manga images [88], [89].

Auxiliary Classifier GANs (AC-GANs)

Odena, Olah, and Shlens [90] add an auxiliary classifier to the discriminator that gives class probabilities, and additionally feed the class label to the generator input to produce images conditioned on the label. They call their method AC-GAN. Both generator and discriminator are optimized to predict the correct class labels. They test the discriminability with a pretrained CNN-classifier (InceptionNet [31]). The experiments show that class conditioning improves the image generation quality. Another contribution is that they demonstrate that high-resolution images provide more information about classes than low resolution images. AC-GANs are used by Zhang, Ji, and Lin [91] for style transfer in Anime sketches. The work by Odena, Olah, and Shlens [90] suggests that combining a generative model such as a GAN with a discriminative classifier can at least improve the performance of one of the two. However, for the problem at hand, using a GAN to aid the classifier is of more use than the opposite case.

Bayesian Data Augmentation GANs (BDA-GANs)

Tran, Pham, Carneiro, *et al.* [92] and Antoniou, Storkey, and Edwards [93] directly address data augmentation using GANs. Tran, Pham, Carneiro, *et al.* [92] formulate the data augmentation problem as a statistical learning problem. The synthetic data samples for augmentation are treated as missing (latent) variables $z = (y^a, x^a)$, where x^a is a generated image and y^a is its corresponding label. They propose an algorithm called Generalized Monte Carlo Expectation Maximization (EM), that gives a Maximum A-Posteriori (MAP) estimate for the model parameters $\theta^* = \arg\max_\theta \log p(\theta|x, y)$, where the likelihood is given by:

$$\log p(\theta|x, y) \approx \log p(\theta) + \frac{1}{N} \sum_{n=1}^{N} (\log p(y_n|x_n, \theta) + \log p(x_n|\theta)) \tag{6-5}$$

EM is an iterative optimization method for problems with latent variables, where θ^i is the estimate for the parameters of the model $p(\theta|x, y)$ at iteration i. The E-step takes the expectation of $\log p(\theta|x, y, z)$ with respect to the conditional predictive distribution $p(z|\theta^i, x, y)$:

$$Q(\theta, \theta^i) = \mathbb{E}_{p(z|\theta^i,x,y)} [\log p(\theta|x, y, z)] = \int_z \log p(\theta|x, y, z) p(z|\theta^i, x, y) dz \tag{6-6}$$

During training, the expectation is calculated by averaging over mini-batches. The M-step then maximizes this expression:

$$\theta^{i+1} = \arg\max_\theta Q(\theta, \theta^i) \tag{6-7}$$

The individual terms of Q can be linked to terms in the extended GAN loss function for a generator G, authenticator A and an additional classifier C (A and C are discriminators in standard GAN-parlance), that are jointly trained. G and A are conditioned on class label y. This implementation called BDA-GAN, is architecturally similar to the AC-GAN, with the difference that the classification is done in a separate network. The function Q can be optimized using SGD, where sampling mini-batches and calculating the cost function Q correspond to the E-step and performing a parameter update corresponds to the M-step. Advantages of this method are that new data is sampled from the distribution which is learned from the annotated training set and the generator is jointly trained with the classifier to improve the classification performance. Any type of classifier and generator architecture can be used with this approach. Experiments show improvements of performance compared to conventional data augmentation and the classifier without data augmentation. Unfortunately, the authors do not address the typical training instability problems of GANs and also experiment only on the very simple datasets for MNIST, CIFAR-10 and CIFAR-100. It would be interesting to see how this fares with a dataset like ImageNet or similar.

Data Augmentation GANs (DA-GANs)

Antoniou, Storkey, and Edwards [93] propose a GAN architecture that uses an image conditioned GAN for data augmentation, called DA-GAN. More precisely, the generator *and* discriminator are conditioned on a true image x_i. The generator then produces $\hat{x}$, a transformed version of x_i and the discriminator decides if $\hat{x}$ is from the data generating distribution (given x_i) or not. The reference for the discriminator from the real distribution is an image x_j, different from x_i, but of the same class. The DA-GAN therefore learns to generate class-preserving transformations but is not directly optimized to help a classifier. Experiments on Omniglot, EMNIST and VGG-Faces datasets with ResNet-derived classifiers show clear improvements of the classifiers compared to training without DA-GANs. Also here, the DAGAN is agnostic to the type of classifier that is chosen.

Comparing the DA-GAN and BDA-GAN [92] architectures, only small differences can be found: DA-GAN uses a GAN conditioned on *true images* x. This makes DA-GANs suitable for augmentation tasks on previously unseen classes or on unlabelled data. BDA-GAN uses a GAN conditioned on *image class label* y, to be used in a supervised setting, which makes it suitable for example for class imbalance tasks. Both can be used to augment certain image attributes like image resolution. In the DA-GAN, x is first projected onto a lower dimensional manifold before given as a condition to the generator network. From a scientific perspective, BDA-GANs are theoretically more sound, as their optimization technique is derived from methods in statistical learning, while DA-GANs are the outcome of empirically trying a model architecture. Furthermore, BDA-GANs are trained to optimize some classification loss of a classifier.

Deep Adversarial Data Augmentations (DADAs), and Tanda

In a current preprint, Zhang, Wang, Liu, *et al.* [94] propose DADA for extremely low data regimes. They start by formulating the data augmentation problem from scratch and framing it as an adversarial problem. They define a classifier C that takes an image x and predicts a class label y. Additionally, it predicts if the image is real, which makes C equivalent to D in a GAN. The output of C has thus dimension $2k$, where k is the number of classes. Secondly, they define the augmenter A, that synthesizes new data conditioned on label y, similar to the generator G in a conditional GAN. To really distinguish DADA from GAN, the authors introduce a novel loss function on the $2k$-output with the goal to synthesize more diverse samples than GANs. It is a quite interesting approach but less relevant for the problem setting of this report, as we do not want to combine the classifier and discriminator.

Ratner, Ehrenberg, Hussain, *et al.* [95] use a generative sequence model called *Tanda* trained on unlabelled data with Reinforcement Learning to learn transformation functions for data augmentation. This approach is also not applicable to our problem: We do have unlabelled data (wide-shots of the aircraft), but it is not in the form to be directly used in classification as the objects have to be extracted first, which is done by labelling them.

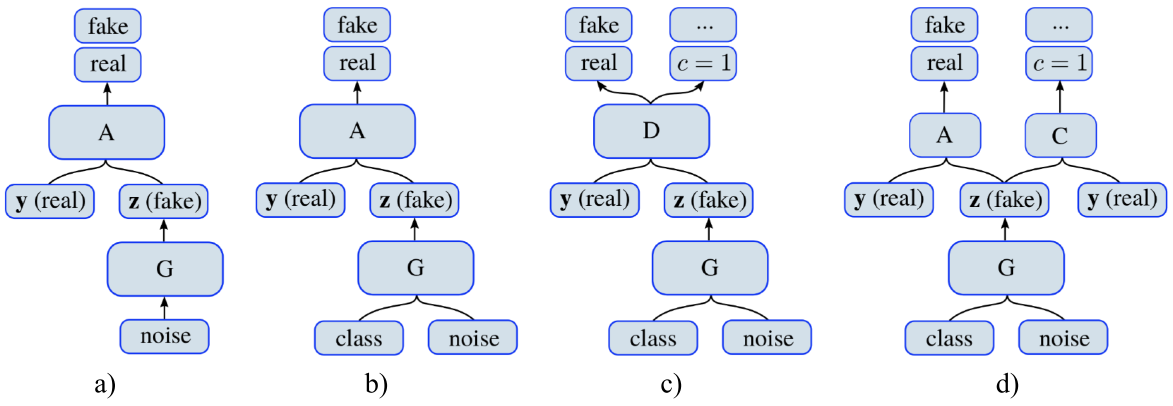


Figure 6-3: Overview of a selection of GAN architectures. a) GAN [12]; b) CGAN [87]; c) AC-GAN [90]; d) BDA-GAN [92]. A: Authenticator, C: Classifier, D: Discriminator, G: Generator. Source: Tran, Pham, Carneiro, *et al.* [92]

6-3 Summary

This section briefly summarizes the findings in data augmentation for visual data. Classical augmentation techniques based on hand-engineered transformation rules have been proven very useful in increasing generalization capability of classifiers in image recognition tasks [2], [70]. These rules are merely heuristics, can require expensive selection procedures and their effectiveness is somewhat dependent on the creativity of the programmer. Research in that field has not been progressing, with the exception of AutoAugment [72], that aims at learning the transformation policies from data with Reinforcement Learning.

Generative models are machine learning models that try to approximate the data generating distribution $p(x)$ of a training set and are therefore able to create diverse new examples for an augmented training set. Deep generative models can be divided into models derived from Variational Autoencoders (VAEs) [11] and Generative Adversarial Networks (GANs) [12].

VAEs use an encoding-sampling-decoding scheme parametrized by neural networks. Encoding a data point results in a conditional distribution $p_\theta(z|x)$ of latent variables z. Sampling z from that distribution and decoding leads to a reconstruction of the original data point. Alternatively, z can be sampled from a prior distribution $p_\theta(z)$ to generate a completely new data point. Conditioning the encoder and decoder on labels y or attributes gives the user additional control over labels and attributes of newly synthesized data [78], [79], which can be used in the imbalanced class setting. Open areas of research are how to train an autoencoder specifically for classification performance maximization of a separate classifier, either by joint training and classification loss back-propagation into the VAE or by learning a latent prior to improve performance.

GANs consist of two networks that compete against each other during training. The first network is a generator G which creates data form noise, and the second network, the discriminator D, judges if its input is fake or real. At the equilibrium, G will produce data samples from the data distribution $p_{data}(x)$ that D can not distinguish from real samples. GANs can also be conditioned, which enables generation of samples given incomplete images or image class labels [87], [90]. This is subsequently used by [92]–[94] to create data augmenting GANs, where [92], [94] train their data augmenting models to also improve classification performance. GANs suffer from instability during training [84], [85] and do not necessarily create diverse images, which is partly addressed by [86]. Figure 6-3 gives an overview of the GAN architectures discussed above.

Reinforcement Learning approaches to learn transformations [72], [95] actually form a different family of techniques for data augmentation but are out of the scope of this project.

Recent research on generative models, especially for data augmentation, has mostly been focused on GANs and has achieved staggering results in realism when scaled up [86]. However, we do not necessarily need access to that kind of realism as our damage examples are relatively simple (they do not have complex shapes like legs, heads, faces, etc.) and we also do not have the data for it. There are two other decisive advantages of VAE-based models over GANs:

- They offer encoding capability to the latent space z. This enables two things: estimating the likelihood of a query image x and having the choice of generating new samples by conditioning on a query image x or by conditioning it on a label y.
- They do not suffer from the training instabilities that GANs have.

Furthermore, VAEs offer some potential starting points for new research, as they have not enjoyed a lot of attention so far. For example, choice of latent priors or joint training with classifiers could be promising ways of using VAEs for data augmentation.

Chapter 7

Meta-Learning

While the previous chapter discussed methods to be applied on the training data, this chapter discusses methods on the learning model to increase classification performance, when only small datasets are available. Meta-learning is a general term for learning higher level properties of learning models. It can be seen as adding an additional optimization loop "on top" of the usual optimization process for learning from data. It is also called *learning to learn* but there is no clear definition for meta-learning yet. Essentially, there are two directions of meta-learning:

- Learning the best model architectures and (hyper-) parameters for a certain task.
- Learning a model to perform well across different tasks.

This chapter focuses on the second type of meta-learning: Deep learning has demonstrated promising results in certain domains and we want to find ways of levering performance from other domains for our purposes. Section 7-1 presents methods for meta-learning across tasks. There are two sub-types of meta-learning: transfer learning, discussed in Section 7-2, and low-shot learning, discussed in Section 7-2. Section 7-4 summarizes this chapter.

In this text, a task is specified by its output type (classification or regression), number of outputs (i.e. number of classes), trained weights, the dataset it is trained on and performance metrics or cost functions it is evaluated on. Vanschoren [13] additionally includes hyperparameter settings, pipeline compositions, and network architectures to define the *meta-data*. However, we assume that these are fixed for a given model architecture.

We define a task for initial training (or *pre-training*) as *base task* and similarly object classes for initial training *base classes*, the new task (class) that the model is transferred to is called *target task (class)*. In the context of low-shot learning, the target task is also called *novel* task.

7-1 Meta-Learning Across Tasks

Meta-learning across tasks is strongly dominated by transfer learning (Section 7-2) and low-shot learning (Section 7-3), but there are some publications that discuss more general meta-learning across tasks.

Meta-Learning with Recurrent Models

Hochreiter, Younger, and Conwell [96] apply gradient descent methods for meta-learning by using a Long Short-Term Memory (LSTM) [97] as meta-learner. Ravi and Larochelle [98] go a step further and use an LSTM as *meta-learner* to learn the gradient descent update of a *learner*. The conventional gradient upgrade rule is given by:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t \tag{7-1}$$

The update rule of an LSTM is given by:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{7-2}$$

The authors observe the similarity between Equation 7-1 and Equation 7-2 and set $c_t = \theta_t$, and $\tilde{c}_t = \nabla_{\theta_{t-1}} \mathcal{L}_t$. f_t and i_t are parametrized as well so that the meta-learner can determine the optimal values itself. The training procedure consists of an inner loop that updates the parameters θ_t of learner M with Equation 7-2 and an outer loop that updates the parameters Φ_d of meta-learner R using $\nabla_{\Phi_{d-1}} \mathcal{L}_{test}$. Experiments on miniImageNet show a close performance to Matching Networks (MNs) for the one-shot task and exceed performance of MNs on the five-shot task, the state-of-the-art few-shot method at that time.

Meta-Learning with Gradient-Based Models

Finn, Abbeel, and Levine [99] propose Model-Agnostic Meta-Learning (MAML) for fast adaptation of deep networks. It is an algorithm that performs meta-learning on any model f with any loss function, provided it is trainable by gradient descent. The goal is to train a model on a variety of tasks such that it will be able learn new tasks with only a small new training data set. In other words, it trains models to be easy to fine-tune by making the parameters θ sensitive to changes in the tasks. The model then trains "general-purpose" representations from the data. These are representations that are useful across tasks and help the model to adapt quickly to new tasks. The proposed algorithm first calculates the updated parameters θ_i' for a randomly sampled task $\mathcal{T}_i$ and its loss $\mathcal{L}_{\mathcal{T}_i}(f_\theta)$ by doing an "inner" (task) gradient step:

$$\theta_i' = \theta_{t-1} - \alpha \nabla_{\theta_{t-1}} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_{t-1}}) \tag{7-3}$$

The "outer" (meta) gradient step is performed by summing the gradients of task losses evaluated at θ_i' and updating θ accordingly:

$$\theta_t = \theta_{t-1} - \beta_t \nabla_{\theta_{t-1}} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) \tag{7-4}$$

t denotes the iteration number and i denotes the task index. Basically, this method adds an outer gradient update loop to a conventional gradient descent training procedure. It requires the calculation of the gradient through the gradient, which can be computationally expensive. However, the authors show that a first order approximation does not degrade accuracy significantly while being 33% faster. After meta-training with this outer and inner gradient update procedure has converged, the trained weights can be used to initialize training on a target task. The algorithm is able to converge on the target task within few iterations The authors demonstrate the effectiveness of this algorithm on a simple regression example

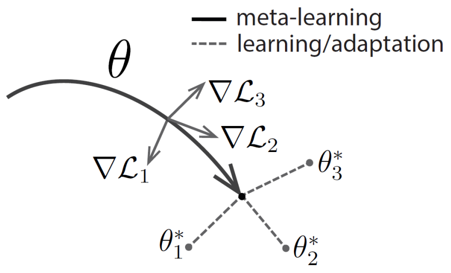


Figure 7-1: Diagram visualizing the gradients of the Model-Agnostic Meta-Learning (MAML) training procedure. Source: MAML [99]

to approximate a sinusoidal function and on the Omniglot and miniImageNet datasets. The algorithm can effectively be used for training a low-shot classifier, regressor or reinforcement learning agent. It outperforms Memory Augmented Neural Networks (MANNs) [100], [101], Siamese Nets [102] and MNs [103] for both Omniglot and miniImageNet one-shot and five-shot tasks. Modifications to MAML have been proposed by [104], [105]. Finn and Levine [106] show that gradient-based meta-learning can approximate any learning algorithm and that it is able learn strategies that generalize more widely than strategies learned by recurrent models.

It can be seen that meta-learning across tasks is characterized by learning in two phases, called *meta-training* (training on tasks) and *meta-testing* (training and evaluating on data). It becomes more eminent in the following two sections about transfer learning and low-shot learning.

7-2 Transfer Learning

Transfer learning is a special case of meta-learning across tasks, where the model is optimized first on one task and is then optimized on a second task. It works with the additional restriction that the novel model has to be structurally similar to the base model. For neural networks in particular, transfer learning takes the parameters (weights and biases) of a base model and initializes the target model with said parameters. The target model is *pre-trained* on a broad dataset and can then be *fine-tuned* for a target task on a new dataset. The two-phase training set-up is obvious.

It is perhaps the most popular and well proven type of meta-learning: Sharif Razavian, Azizpour, Sullivan, *et al.* [107] use an off-the-shelf Convolutional Neural Network (CNN) pre-trained on ImageNet to perform visual recognitions tasks on other datasets. The CNN architecture called OverFeat [108] 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. Donahue,

Table 7-1: General guidelines for transfer learning, dependent on target task and available dataset for target task. Source: CS231n [18]

	Similar Task	**Different Task**
Small Dataset	linear classifier on top layer	linear classifier on top layer + additional tricks
Large Dataset	fine-tune some layers from top	fine-tune larger number of layers from top

Jia, Vinyals, *et al.* [27] demonstrate that features learned on large datasets have a sufficient representational power and generalization ability to outperform recognition models using hand-engineered features. Yosinski, Clune, Bengio, *et al.* [28] perform a more rigorous study on transferability of CNN features. They observe that lower layers produce more general and transferable features, while higher layers are more specialized to a specific task. When transferring without fine-tuning, performance degrades the further the target task is away from the base task. However, even for unrelated tasks, initialization with pre-trained weights yields higher performance after fine-tuning than initialization with random weights.

Pre-trained weights for various model architectures and frameworks are freely available online. The Stanford lecture CS231n [18] gives a general guideline for transfer learning which is depicted in Table 7-1. In the special case where the target task is very different from the base task and the available dataset is small (as in our case), the advantages of transfer learning break down and other techniques might be necessary, for example more specialized meta-learning methods like low-shot learning for learning from very small datasets. These are discussed in the following section.

The current convention of always starting with pre-trained models is challenged by He, Girshick, and Dollár [109]. The authors train a Mask R-CNN [40] for an object detection and segmentation task. The models 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 overfitting when compared to training from scratch. However, pre-training does accelerate convergence and helps reduce overfitting in very small data regimes. Interestingly, they also conclude that pre-training a detection model on a classifications task can be omitted when obtaining the classification data is expensive. Instead, they suggest to collect data in the target domain.

7-3 Low-Shot Learning

Low-shot learning is a learning method, where a model is optimized to learn a novel task very quickly with only a low number of training data points for that task. It is also called few-shot learning, with zero-shot learning and one-shot learning as the most extreme cases. Unfortunately, there is no method that could do this without extensive pre-training or other kinds of prior knowledge from one or more base tasks, which is why low-shot learning currently falls under the category of meta-learning.

One-shot learning was first mentioned in literature by Li, Fergus, and Perona [110], [111]. They present a Bayesian approach to detect object categories. Objects are represented by the constellation model [26]. Prior knowledge is given as a probability density function on the model parameters, obtained from pre-training on other object categories. The posterior on the parameters is obtained by updating the prior given observations of the novel class.

Recent research on Bayesian low-shot methods has been done by Lake, Salakhutdinov, and Tenenbaum [112], [113]. However, their method called Bayesian Program Learning (BPL) is focused on imitating human learning behaviour. For their experiments they hand-engineered parts, sub-parts, primitives and sequences thereof for the representation of the visual objects (handwritten characters from the Omniglot dataset). Due to the very domain specific, hand-crafted implementation and the focus on human learning, BPL is not a suitable candidate for this project but has been included for the sake of completeness.

Siamese Networks

Low-shot learning with neural networks has been an active area of research: Koch, Zemel, and Salakhutdinov [102] use a so called *Siamese Network* for one-shot image recognition. A Siamese Network consists of two identical networks that are connected at some higher layer, borrowing from the idea of Siamese Twins (Figure 7-2). They train the Siamese Network first for the verification task, i.e. predicting if two images are form the same class or not. Each of the two "twins" takes an image as input. The networks have a convolutional architecture and are joined in the penultimate layer, where the weighted L_1 distance between the feature maps of the CNNs is calculated. The final layer uses a sigmoid activation and outputs the probability that the two images are from the same class. After training for the verification task, a query image can be classified by comparing it to one example of each class and assigning it to the class with the lowest distance score. For the one-shot classification task, they compare the query image to examples of classes that were not present during training. We will call the set of these comparison examples *support set* S_{sup}. The support set has to be carried along during testing time. Experiments were done on the Omniglot data set, where an accuracy of 92.0% was achieved (Humans: 95.5%, HBPL [112]: 95.2%).

Learnets

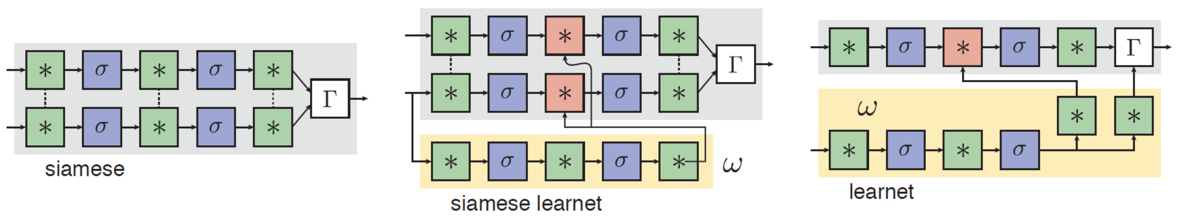


Figure 7-2: Siamese Net, Siamese Learnet and Learnet. The Learnet architectures predict the parameters of a network from a single example, replacing static convolutions (green) with dynamic convolutions (red). The siamese learnet predicts the parameters of an embedding function that is applied to both inputs, whereas the single-stream learnet predicts the parameters of a function that is applied to the other input. Linear layers are denoted by $*$ and nonlinear layers by σ. Dashed connections represent parameter sharing. Source: Bertinetto, Henriques, Valmadre, *et al.* [114]

Bertinetto, Henriques, Valmadre, *et al.* [114] use a feed forward network architecture called *Learnet* to predict the parameters of a so called *pupil network* from a single instance (Figure

7-2). The pupil network predicts the class of the input image. Its predicted parameters are the convolution weights. Training is set-up similarly to the Siamese Net [102]: The network is fed image pairs and a label indicating if the pair is from the same class or not. However, instead of learning embeddings that are compared for similarity, the feed-forward learner dynamically predicts the convolution weights of a classifier as opposed to the the Siamese Net architecture. To reduce the number of parameters of the Learnet, the weights to be predicted are factorized so that each feature channel of the convolutional layer is convoluted independently. The number of parameters then scales linearly with the number of channels instead of quadratically. Experiments on a character recognition task were done with a Siamese Net with shared and unshared weights, a Siamese Net with unshared and factorized weights, a Siamese *Learnet* with shared weights and a single stream Learnet. Each were tested with different similarity metrics in the final layer, such as inner product, Euclidean distance, and weighted ℓ_1 distance. The single stream Learnet with ℓ_1 final layer has the best error rate with 28.6%, compared to a Siamese Learnet with ℓ_1 final layer and a 31.4% error rate. The original Siamese Net implementation achieves an error rate of 37.3% with Euclidean distance final layer. Similarly to the Siamese Nets, the Learnet has to carry a support to initialize the classifier parameters.

Memory Augmented Neural Networks (MANNs)

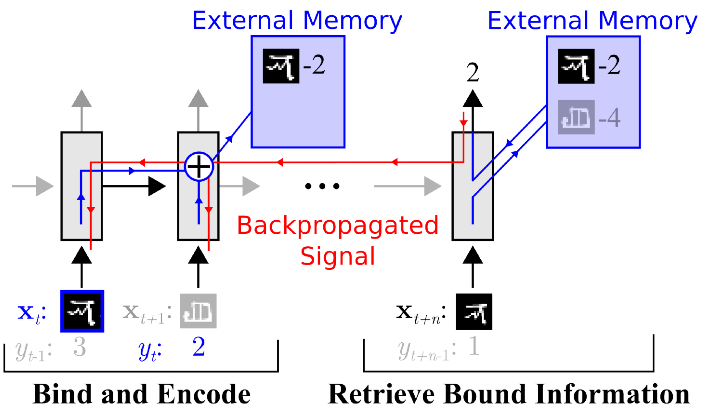


Figure 7-3: MANN training and prediction strategy. Sample data x_t from a time t is bound to the appropriate class label y_t, which is presented at $t+1$. Later, when a sample from this same class is seen, it retrieves this bound information from the external memory to make a prediction. Backpropagated error signals from this prediction step will then shape the weight updates from the earlier steps in order to promote this binding strategy. Source: Santoro, Bartunov, Botvinick, *et al.* [100]

Santoro, Bartunov, Botvinick, *et al.* [100], [101] use Memory Augmented Neural Networks (MANNs), that is neural networks that store an embedding of the input in an external memory M_t (Figure 7-3). Given input x_t, the output prediction is done on a vector o_t that is composed of some features h_t extracted from the input, concatenated with a memory r_t that corresponds to the input. The read-and-write operations to the memory is controlled by a Recurrent Neural Network (RNN), a variant of a Neural Turing Machine (NTM) [115]. An NTM is a completely differentiable architecture to access a memory and can thus be trained from data. h_t is the hidden state of the controller RNN. r_t is a weighted average of rows in

M_t where memories with a high similarity to the key k_t (cell state of the controller RNN) receive a higher weight. The writing access scheme updates M_t with a weighted k_t, where the weights are based on the least recently used memory location (LRU access scheme). o_t is then fed to a single layer classifier with softmax output for one-hot classification. Episode loss is the cross entropy loss. During training, the MANN is fed samples in a sequence and has to predict their labels. The loss is then back-propagated through time, and the controller learns to save and access relevant data in the memory. During (one-shot) testing time, the network is not trained on the novel class, but simply saves it for later reference.
Experiments were done with an LSTM controller and LRU access scheme and 15 object classes after 100,000 training episodes on the Omniglot dataset. This yielded 36.4% accuracy after training with one instance of an unknown class, 82.8% after two instances, 91.0% after three, 92.6% after four, 94.9% after five and 98.1% after ten instances trained on five classes.

Matching Networks (MNs)

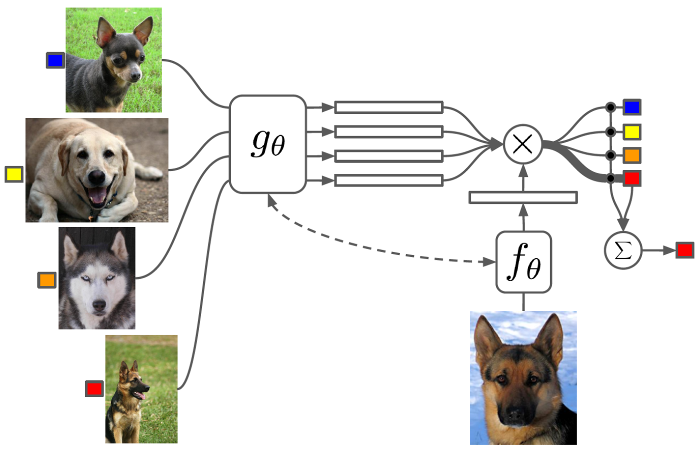


Figure 7-4: MN architecture. The support set of a few examples is encoded by a neural network g and the query image is encoded by a neural network f. The class score is a softmax over a similarity/distance between encoded query image and support set images. Source: Vinyals, Blundell, Lillicrap, *et al.* [103]

Vinyals, Blundell, Lillicrap, *et al.* [103] introduce Matching Networks (MNs) for one-shot learning (Figure 7-4). This could be seen as a generalization of the work of [100] and [102]. They use a support set S_{sup} of previously unseen labelled images and use an attention mechanism to assign a label $y = \arg\max_k \hat{p}_k(x)$ to a query image x. The attention mechanism is the softmax over the cosine distance d between the embeddings $f(x, S_{sup})$ and $g(S_{sup})$:

$$\begin{aligned} h(x, S_{sup}; \theta) &= \hat{p}(x) \\ \hat{p}_k(x) &= \frac{\sum_{(x_i,y_i)\in S_{sup}} e^{-d(f(x),g(x_i))}\mathbb{I}[y_i = k]}{\sum_{(x_i,y_i)\in S_{sup}} e^{-d(f(x),g(x_i))}} \\ f(x, S_{sup}) &= \text{AttLSTM}\left(\phi(x;\theta_\phi), \{g(x_i)\}_{i=1}^N; \theta_f\right) \\ \{g(x_i)\}_{i=1}^N &= \text{BiLSTM}\left(\{\phi(x_i;\theta_\phi)\}_{i=1}^N; \theta_g\right) \end{aligned} \tag{7-5}$$

The nomenclature for Equations 7-5 to 7-7 are slightly modified from their original papers [103], [116], [117] to make them consistent within this report. h is the low-shot classifier, that classifies a query image x. It carries a support set S_{supp} that has elements (x_i, y_i). $\hat{p}$ is the probability vector across the classes. ϕ is a (convolutional) neural network that extracts features of x and x_i. θ_ϕ, θ_f and θ_g are parameters of ϕ, f and g, respectively. d is a distance metric, in this case the cosine similarity, but it could be any distance.

Instead of using an encoded external memory as in MANNs [100], the memory is simply the support set S_{sup}, but it is in fact also encoded by the function g. Similarly, one can view one "twin" of the Siamese Net architecture as the encoding function g_i for the i_{th} support image, and the other "twin" as the function f to encode the query image.
Matching Nets improve the accuracy of MANN to 98.1% accuracy in a one-shot task and to 98.8% in a five-shot task with each five classes! The high accuracy of this method set a new benchmark at that time, but has beaten more recently [98], [99], [117], [118]. Still, in the $n = 1, 2$ regime, Matching Nets stay unbeaten. However, they have some shortfalls: In theory, it is enough to chose one example of each class for S_{sup}, but in practice, to be sure, we probably want to use the complete training set as support, because it is not known exactly what is a "good" or "bad" support. Also, if there is a class imbalance in S_{sup}, the attention mechanism biases towards classes with high occurrences due to the summation of class frequencies in the numerator. There are solutions to these issues addressed in [116], [117].

Shrinking and Hallucinating Features

Hariharan and Girshick [118] try to solve visual recognition by shrinking and hallucinating features. They denote classes which are known have many training examples available as base class and the classes for the low-shot task as novel class. *Shrinking* denotes a novel loss function that forces the weights of a classifier trained on a large data set to minimize the loss of training on a small data set. It is called the Squared Gradient Magnitude (SGM) loss and can be seen as a weighted L_2 regularization on the feature activations. *Hallucinating* is a novel way of generating new samples of novel classes by looking at variations in the base classes: A multilayer perceptron takes a uniformly sampled image pair from a base class and a sampled image from a novel class and applies the transformation between the base images to the novel image.

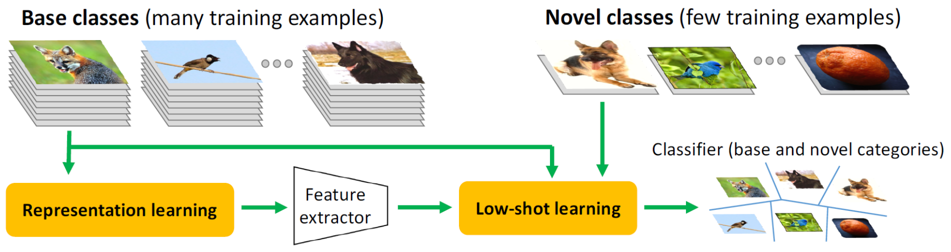

Base classes (many training examples)
Novel classes (few training examples)
Classifier (base and novel categories)
Representation learning
Feature extractor
Low-shot learning

Figure 7-5: Two-phase training for low-shot learning: representation learning and low-shot learning. Source: Hariharan and Girshick [118]

Training is split into two phases (Figure 7-5): First, training a feature representation on a large data set of base classes (with SGM regularization). Second, the low-shot phase (with hallucinating) where a linear classifier head is trained on the feature representation. Experiments show that these methods outperform Matching Networks [103] for $n > 2$ where Matching Networks stay strong for one-shot and two-shot problems. Their experimental set-up is also more relevant for low-shot learning situations that arise in practice. The method does not require any form of memory or support set with embedding to be carried on during testing and is thus more comparable to the methods of Section 7-1.

Prototypical Networks (PNs)

PNs by Snell, Swersky, and Zemel [116], use a fairly simple architecture for the low-shot classifier h that computes the softmax over distances to the class means μ_k (*prototypes*) in S_{Sup}:

$$\begin{aligned} h(x, S_{sup}; \theta) &= \hat{p}(x) \\ \hat{p}_k(x) &= \frac{e^{-d(\phi(x;\theta_\phi),\mu_k)}}{\sum_j e^{-d(\phi(x;\theta_\phi),\mu_j)}} \\ \mu_k &= \frac{\sum_{(x_i,y_i)\in S_{sup}} \phi(x_i;\theta_\phi)\mathbb{I}[y_i = k]}{\sum_{(x_i,y_i)\in S_{sup}} \mathbb{I}[y_i = k]} \end{aligned} \tag{7-6}$$

It resembles the k-means algorithm [119], except that the means are calculated in a learned feature space that is extracted by ϕ, and that S_{sup} does not necessarily consist of all the available data. The advantage of collapsing the support into the class means μ_k is that overrepresented classes do not overwhelm minority classes. Ren, Triantafillou, Ravi, *et al.* [120] use Prototypical Networks in combination with soft k-means clustering to perform semi-supervised few-shot classification.

Prototype Matching Networks (PMNs)

Wang, Girshick, Hebert, *et al.* [117] continue on the concepts of hallucinating in [118]. They propose a new method that can hallucinate examples with features that are useful for the classification task (instead of learning features that are useful for just generating data). The hallucinator (a neural network G) is combined with a meta-learner based on Matching Nets [100] and Prototypical Nets [116]. The new model called Prototype Matching Network (PMN) improves on the weakness of Matching Nets not being robust against class imbalances in the support set by using contextually embedded class means as reference for the attention mechanism instead of a embedded support set:

$$\begin{aligned} h(x, S_{sup}; \theta) &= \hat{p}(x) \\ \hat{p}_k(x) &= \frac{e^{-d(\phi(x;\theta_\phi),\nu_k)}}{\sum_j e^{-d(\phi(x;\theta_\phi),\nu_j)}} \\ f(x, S_{sup}) &= \text{AttLSTM}\left(\phi(x;\theta_\phi), \{\nu_k\}_{i=1}^{N}; \theta_f\right) \\ \nu_k(x) &= \frac{\sum_{(x_i,y_i)\in S_{sup}} g(x_i)\mathbb{I}[y_i = k]}{\sum_{(x_i,y_i)\in S_{sup}} \mathbb{I}[y_i = k]} \\ \{g(x_i)\}_{i=1}^{N} &= \text{BiLSTM}\left(\{\phi(x_i;\theta_\phi)\}_{i=1}^{N}; \theta_g\right) \end{aligned} \tag{7-7}$$

Training is done in a two-phase manner. In both phases, $G(x, z; \theta_G)$ is used to generate new data S^G from a sample x in S and a noise vector z. Firstly, in the meta-training phase, the hallucinator G is trained jointly with the classifier $h(x, S_{sup}; \theta)$ (containing the PMN) and therefore will help improving the classifying performance. Meta-training means it is optimized to generalize across tasks, therefore, only a subset of the training set with m out of C base classes is used, creating a different classification task for each meta-training iteration. h will not be biased towards a specific classification task. From that subset, a support set S_{sup} and a training set S_{train} is sampled. S_{train} is used to back-propagate the weight-gradients of h and G. The subset S_{sup} is used to calculate the embedded class means.

Secondly, in the meta-testing phase, the hallucinator is kept fixed and the classifier h is fine-tuned on novel-classes. Experimentation was done on the ImageNet dataset. A CNN based on ResNet-10 and ResNet-50 was trained on the base classes first to extract the features of the data. These pre-computed features served as the input to h and G. For G, a three layer Multi-Layer Perceptron (MLP) with Rectified Linear Unit (ReLU) activations and ReLU output was used. They compare PMN to PMN and hallucinating (PMN+G), Prototypical Network (PN), PN and hallucinating (PN+G) and to MN, where all shared design parameters are equal between the PMN and PM. As baselines, they use a logistic regression linear classifier and a logistic regression linear classifier with the hallucinating procedure of [118].

In terms of absolute accuracy, PMN+G beats all baselines except for the $n = 20$ case, all classes are evaluated and a prior probability on the novel class is included. Their experiments also show that the advantage of all the low-shot methods with respect to the linear classifiers seems to vanish with $n > 20$! The experimental results for one-shot and five-shot classification of the ResNet-50 based models on novel classes is shown the third block of Table 7-2.

The work of Wang, Girshick, Hebert, *et al.* [117] is in so far interesting because it discusses many issues that have been addressed in this report as well: It builds a connection between low-shot learning, meta-learning and data augmentation. They also propose solutions to previously mentioned problems: By combining the Matching Nets with Prototype Nets and using contextually embedded class means, there is no need to carry a support set or memory during testing time like [100], [102], [103], [114]. Additionally, it is more resilient against class imbalances. They also show that low-shot learning methods do not add any performance when the number of novel samples is higher than around 20 samples.

Memory Matching Networks (MM-Nets)

A recent paper proposes to combine Memory Augmented Neural Networks and Matching Networks, dubbed Memory Matching Networks (MM-Nets). A novel image is embedded by a CNN, compared to the embedded support images and classified by evaluating a similarity over the embeddings. The memory module provides the embedded support images. At the same time, a contextual learner in the form of a bidirectional LSTM uses the memory module to predict the kernel weights of the CNN. This rather complicated and nested architecture outperforms the previous low-shot methods and also MAML (Table 7-2).

Using an architecture that is specifically designed for the low-shot classification task, like a Siamese Net, Learnet, MANN, MN, PMN and MM-Net as a feature extractor in an object detection architecture seems complicated, because the input in an object detection framework contains an unknown number of classes instead of just one. In that case, the methods

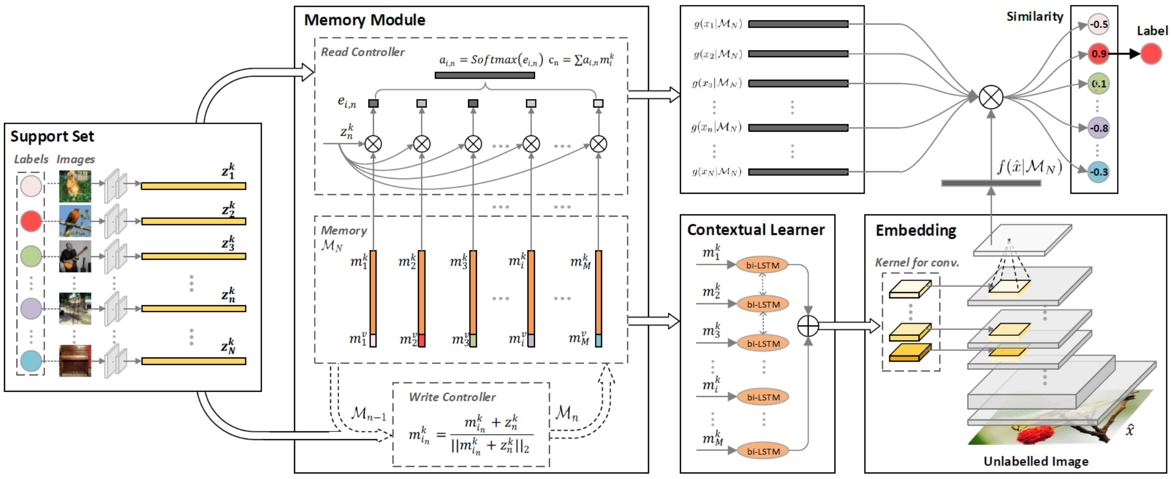


Figure 7-6: MN architecture. Source: MM-Net [121]

mentioned above could possibly be used in the second step of a staged detection procedure: Predicting the class after RoI-pooling.

7-4 Summary

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. In this chapter, the focus was on meta-learning to improve learner's performance across tasks.

This can be achieved by training a recurrent neural network like LSTM to estimate the learning parameters [96], [98] or by adding a meta-gradient outer loop to the gradient update rule [99], [104], [105]. Both are model agnostic, however, the latter does not require an additional meta-learner network (with complex hyperparameter tuning etc.) and can generalize better [106], which 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.

Transfer learning is a specialization of meta-learning across tasks where the learner's model architecture is fixed and weights are pre-trained on a base model and a large dataset. The pre-trained weights can be used to initialize the target model, which considerably increases performance on the target task with accelerated convergence [27], [107]. Depending on the distance between base and target task and size of the target training dataset, different levels of fine-tuning are required [28]. Transfer learning is a standard practice in deep learning and many pre-trained models are available on-line.

Low-shot learning is a more extreme specialization of meta-learning, where a target task or novel task has to be learned with only a low number of training data points. Low-shot learning methods rely on a support set [102], [103], [114], [116], [117], [121] or external memories [100], [101], [121] that represent prior knowledge. This knowledge can be accessed by different access schemes, for example attention mechanisms or NTM. Low-shot methods that do not rely on support or memory borrow from general meta-learning [118] and data augmentation [117], [118].

Table 7-2: 5-way (5 classes) top-5 accuracy on novel classes of some of the meta-learning and low-shot learning methods. The first and second block of the table, provided by [99] and [121], compares the different models trained on the Omniglot and *mini*ImageNet datasets, respectively. The third block of experiments was performed by [117] and compares methods trained on the more complex ImageNet dataset.

	1-shot	**5-shot**
1. Omniglot as provided by [99], [121]		
MANNs no conv [100]	82.80%	94.90%
MAML no conv [99]	89.70±1.10%	97.50±0.60%
Siamese Nets [102]	97.30%	98.40%
MNs [103]	98.10%	98.90%
MAML [99]	98.07±0.40%	99.90±0.10%
MM-Nets [121]	99.28±0.08%	99.77±0.04%
2. *mini*ImageNet as provided by [99], [121]		
MNs [103]	43.56±0.84%	55.31±0.73%
Meta-learner LSTM [98]	43.44±0.77%	60.60±0.71%
MAMLs [99]	48.70±1.84%	63.11±0.92%
MM-Nets [121]	53.37±0.48%	66.97±0.35%
3. ImageNet w. ResNet-50 as provided by [117]		
PMNs + G [117]	54.70%	77.40%
PMNs [117]	53.30%	75.90%
PNs + G [117]	53.90%	75.70%
PNs [116]	49.60%	74.40%
MNs [103]	53.50%	72.70%

This chapter is heavily biased towards low-shot learning, but it gradually became clear from the literature research and from the problem setting, that extreme low-shot learning was not necessary: The literature shows that low-shot learning is effective at less than 20 novel samples. The available data for the problem at hand at the time of writing exceeds this number already by one order of magnitude and it is steadily growing. Adding to that the incompatibility of low-shot methods for object detection, the conclusion of this chapter is that the simple, less restrictive, more flexible, but equally performing meta-learning method of MAML [99] and the widely used transfer learning are more promising ways to move forward.

Chapter 8

Summary

The objective of this literature survey was to identify the building blocks of a damage detection model and identify the corresponding challenges and possible solutions from recent literature. Chapters 2, 3 and 4 discussed the damage detection building blocks, these are summarized in Section 8-1.

Chapter 5 discussed the possible solutions to mitigate the risks of class imbalances, Chapter 6 presented the latest research in data augmentation techniques to increase data availability and quality. Chapter 7 discussed methods on the model-level to perform well across tasks, also to reduce dependency on large datasets. They are are summarized in Section 8-2.

8-1 Building Blocks for Damage Detection

The building blocks of a state-of-the-art damage detection model evolve around Convolutional Neural Networks (CNNs). While Chapter 2 briefly discussed necessary basics for Deep Learning and consequently, deep CNNs, Chapter 3 went into more detail about the architectural choices for recent CNN-models for Image Recognition. The latest models are characterized by many layers and design choices, such as Inception modules [31] and residual units [17], that improve generalization, speed up training and make the architectures more efficient overall. The current state-of-the-art CNNs, like Inception-v4, Inception-ResNet-v2 [36], Xception [34], ResNeXT [37] and DenseNets [38] are capable of solving complex, large scale recognition tasks as given by the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). This confirms the assumption that CNN-based models are suitable candidates to perform damage detection. DenseNets are especially suitable because of their parameter efficiency and low risk of overfitting.

Going from simple Image Recognition (classification of a single object instance) to Object Detection (classification and localization of multiple object instances) is a question of scaling up an existing CNN and augmenting it with the relevant components, such as region proposal algorithms and bounding box regression heads. Chapter 4 discussed the implementation of CNNs into object detection architectures. Staged detection architectures first perform region proposals. The proposals are extracted, resized, and then one-for-one classified and put into bounding boxes. The Region-based Convolutional Neural Network (R-CNN) family [9], [47], [49] and the Region-based Fully-Convolutional Network (R-FCN) model [50] are representative for this approach. Until recently, complicated training caused by the staged nature of these models is offset by their high mean average precision (mAP). However, the latest trend is going towards unified single-stage models, which require only a single forward pass through the detection network. These single-stage models are represented by You-Only-Look-Once (YOLO) [51]–[53], Single Shot Detection (SSD) [54] and RetinaNet [55]. YOLO-v3 [53] and RetinaNet (which is actually and extension of the SSD architecture) are the current state-of-the-art, where YOLO-v3 is highly optimized for real-time speed and RetinaNet performs better on small objects.

8-2 Challenges and Possible Solutions

The second objective of identifying solutions to the challenges for damage detection evolve around the availability of training data. This area also provides the most potential for new research. For training deep neural networks, a vast amount of training data is necessary to achieve generalization and not overfit to the training set. The methods discussed here address the training of a CNN-based classification model. Once the classifier can classify damages with a satisfying accuracy, it is trivial to transfer the model to the object detection task.

One aspect of data availability is imbalanced data, which is discussed in Chapter 5. Imbalanced data can lead to a heavy bias towards the majority class in the training set and can be detrimental to the performance on the minority class. The class imbalance is a widely studied problem in Machine Learning in general. Methods to deal with imbalanced datasets can be split into data-level methods and algorithm-level methods. Data-level methods aim at reducing the imbalance directly on the training data by oversampling or undersampling or some combination of them. Within the context of Deep Learning, oversampling is preferred because of the data-hungry nature of deep nets and does not necessarily lead to overfitting [10]. Oversampling also includes the generation of synthetic data, i.e. data that does not stem from the original data source but is generated by some hand-engineered or automated process like Synthetic Minority Over-Sampling Technique (SMOTE) [59]. On the algorithm-level, methods compensate for imbalanced data by assigning class-specific cost, assigning different thresholds to different classes and combining multiple learning models to create a stronger model (ensembling).

To increase the number of training examples for neural networks, data augmentation is used. It naturally follows from the idea of oversampling. Chapter 6 describes the classical and generative data augmentation techniques. It is common practice in Deep Learning to apply certain translational, rotational and colour transformations to the samples in the original dataset to improve generalization and invariance of the learning model, this is called classical data augmentation.

Recent research in generative models promises to automatically generate data from the original distribution. Variational Autoencoders (VAEs) [11] and Generative Adversarial Networks (GANs) [12] are the prominent representatives of deep generative models and especially GANs have recently enjoyed a lot of attention from the research community. With

Conditional Variational Autoencoders (CVAEs) [78] and Conditional Generative Adversarial Networks (CGANs) [87], it is possible to generate data according to some condition, e.g. a class label, attribute or an existing image. For example: the model takes a class label as condition and generates an image of that class to be fed as training example to the classifier. Alternatively, the model could take a low-resolution image as condition and produce a high-resolution output and by that increasing or equalizing the quality of the images in the classifier training set. The possibilities are almost endless.

Currently, VAEs are more suited for the damage detection application at hand because they are more flexible with respect to the conditional data generation, they can decode and estimate the likelihood of an existing data point and they do not have the stability and convergence issues that are typical for training GANs. For both VAEs and GANs, the question remains how well these models can help improve a classifier and replace real data collection to some extent.

Meta-Learning is next to generative data augmentation the other major research topic of this survey. It is discussed in Chapter 7. The focus lies on meta-learning to optimize a model across tasks. This requires training in two phases: The first phase is called meta-training or pre-training, which is training on one or multiple base or source tasks. The second phase is meta-testing or training on the target task. General methods for meta-learning across tasks were proposed that use recurrent models [96], [98] and gradient-based models like Model-Agnostic Meta-Learning (MAML) [99], [104], [105]. The latter is able to generalize better than the former [106]. Furthermore, MAML does not require an additional meta-learner neural network but only some additional hyperparameters and can be used for any differentiable learning model.

Sub-topics of meta-learning are transfer learning and low-shot learning. Transfer learning studies how models can be trained on a general task and then be reused for a different class, which is possibly more specialized. It is a widely used and reliable practice in deep learning and is probably also beneficial to this damage detection task. Low-shot learning is another direction of meta-learning with the extreme setting that the target task only has a very low number of training examples. Low-shot methods rely on prior knowledge, which is implemented as support set [102], [103], [114], [116]–[118] or external memory [100], [101]. Low-shot methods can have an impressive accuracy in the one- or two-shot regime (like Matching Nets [103], [117]) but their advantage over conventional learning models diminishes with a higher number of training samples (usually above 20 samples). Therefore, general meta-learning methods not restricted to the low-shot regime are more applicable to this project.

Chapter 9

Conclusion

The main goal of this project is to create an image recognition model to detect, i.e. classify and localize, damages on images of aircraft taken by a drone. This chapter concludes the report, by listing the building blocks of a detection model, answering the research questions of Section 1-3 and proposes a new research question for future work on this project in Section 9-1. Additionally, Section 9-2 gives a short overview of considerations not discussed in this literature review.

Building Blocks

The most optimal choices for the building blocks discussed in Chapters 3 and 4 for the problem at hand are:

- A Convolutional Neural Network (CNN) as feature extractor: Due to the small dataset size and therefore the risk of overfitting, a model that is shallow and has a high parameter efficiency is preferred. The DenseNet architecture [38] will serve as a baseline model for feature extraction and classification.
- Detection architecture: On top of the feature extractor (backbone), more output heads are added to also include localization of objects. Single-stage models perform efficient detection in one-forward pass through the resulting network and can be trained in an end-to-end manner. You-Only-Look-Once (YOLO) [51] is optimized for speed, Single Shot Detection (SSD)-based models [54] are optimized for accuracy.
- Other detection components: The SSD-derived RetinaNet architecture [55] uses Feature Pyramid Networks (FPNs) [56] and focal loss to achieve current state-of the-art results on detection tasks and is especially useful for identifying small objects due to the FPN.

Research Questions

The second part of this report, Chapters 5-7, discussed solutions to the challenges presented in Section 1-2. The research questions in Section 1-3 derived from the challenges can now be answered as follows:

- *How can we help a damage detection model deal with large class imbalances, such that a low False Negative Rate (FNR) on the minority classes is achieved?*
  First, we have to make a distinction between class imbalances in classification and detection. To mitigate performance degradation in classification on the important classes due to class imbalance towards generic background examples, oversampling of the minority classes will be performed. This results in an unbiased training distribution that can be used to train an unbiased feature extractor and classifier.
  For detection, this feature extractor can be frozen and only the weights of the detection heads will be trained in a second training phase. Focal loss can be used to give more importance to examples difficult to classify [55].

- *How can we make sure that a damage detection model achieves high generalization capability when only a small training dataset is available?*
  Classical data augmentation, a standard practice in Deep Learning, will be used as a baseline method to increase dataset diversity and size to prevent overfitting. A Conditional Variational Autoencoder (CVAE) [78] implementation can be used to experimentally investigate the suitability of generative models for automated data augmentation. However, the generative model can be only used in training for classification. Transfer learning, another standard practice, is applicable to this problem and trivial to implement. More sophisticated meta-learning methods like low-shot learning, are not applicable as they are specifically designed for the very-low-data regime.

- *How can we standardize the quality of the training images and match it to the testing images?*
  A CVAE can be used to take a low-resolution image and output a high-resolution image, therefore increasing the quality of the data fed to the classifier. Other kinds of conditional generation, e.g. based on image attributes are possible. This can be applied to both classification and detection data.

- *Is there a relationship between the previously mentioned? In how far are they equivalent or different?*
  Obviously, oversampling against class imbalance and data augmentation are related methods that operate on the *data-level.* One can use dynamic augmentation to create more augmented examples of the minority class and less augmented examples of the majority classes. The combination of oversampling and augmentation results in added randomness that has a regularizing effect, which is preferred to oversampling alone, that exactly replicates minority samples.
  A generative model can take the augmentation task *and* the task of ensuring image quality at the same time. However, using models for augmentation is experimental and an area of research and is not guaranteed to give better results than established methods.
  Transfer learning, and meta-learning in general, is different in the regard that it operates on *model-level* and provides an initialization method for training in the target domain. It's most important characteristic, training in multiple phases (e.g. pre-training and fine-tuning) is a recurrent theme throughout this report and a consistent best practice

for virtually all aspects discussed above. It will therefore also play an important role in the development of a the damage detection system.

9-1 Future Work

There are two aspects to the continuation of the project: The experimental research aspect and a more results-oriented, technical aspect. The research approach would focus on investigating and experimenting with generative data augmentation methods. The results-oriented approach would focus on gathering the data and implementing and tuning a detection model with the goal of achieving low FNR and high mean average precision (mAP).

There is significant overlap between those approaches, but the difference lies in the fact that the experimental methods presented in meta-learning and generative models are designed for the classification task and can currently not be transferred directly to the detection problem.

As a compromise, a classifier and feature extractor will be developed by experimenting with generative augmentation. The outcome of this experimental research will be directly beneficial to the creation of a damage detection model for deployment: the resulting classifier can be re-purposed as backbone feature extractor for the damage detection model.

Therefore, the new modified research question that will guide the project after concluding the literature research now reads:

How can a Convolutional Neural Network (CNN) be extended with generative data augmentation, such that it reliably classify damages with low False Negative Rate (FNR), given that the available data for training is limited in both quantity and quality, and given that there are large class imbalances present, that bias against the damages?

The following action plan has been compiled to address the research question. The first part consists of training a feature extractor and classifier:

1. Collect relevant data by extracting the surface objects from the aircraft inspection images and create a pre-processing pipeline to prepare the experiments
2. Analyse the data and investigate general statistics, attributes, etc.
3. Use ResNet and DenseNet as classifiers and initialize weights pre-trained on ImageNet
4. Train the classifiers in multiple phases with oversampling and classical augmentation and establish it as a baseline given the relevant metrics
5. Train a data-generating neural net (e.g. convolutional Conditional Variational Autoencoder (CVAE)) to perform generative augmentation of the damage data
6. Perform experiments to compare baseline and generative methods

Once the classifier gives satisfying performance, the object detection model can be trained in the second part:

7. Take the first layers of the classifiers with fine-tuned weights as feature extractor for the object detector

8. Add a Feature Pyramid Network (FPN) to recreate the RetinaNet architecture and ensure good performance on small objects
9. Add bounding box regression and classification heads on top of the FPN
10. Train the complete object detection network on the inspection images

All of these steps include sub-steps that are not listed, like cross-validation and hyperparameter tuning.

9-2 Other Considerations

There are some aspects to damage detection, its building blocks and challenges that are not treated in this report. On the engineering side, it is assumed that real-time detection on embedded devices is not needed, so no further research into detection models that are optimized for this has been done.

Segmentation and pixel-wise classification for more precise detection of damages is another step on top of the basic detection that can be performed once detection has been solved. Interestingly this will also help improve detection performance [40] at the cost of more expensive labelling.

On the solutions and research side, Bayesian Neural Networks for meta-learning and uncertainty estimates are not included into this survey. Although briefly touched upon in Chapter 7, they are too immature to be included into this application and their inclusion would go beyond the scope of this project.

Finally, active learning and semi-supervised learning could be a field of research that is interesting for deployment and continuous improvement of the damage detection pipeline. As new detection requests come in, the model could keep improving itself with human feed-back by selecting instances to be labelled or verified by humans and by taking into account unlabelled images.

Appendix A

Requirements

Damage Detection

Preliminary requirements for the damage detection algorithm:
Functional Requirements:

1. The damage detection algorithm shall detect damages in inspections images of aircraft taken by a UAV
2. The algorithm shall take 2D images from the high resolution DJI Zenmuse X5 as input
3. The algorithm shall output the classes of the detected damages
4. The algorithm shall output the location of the detected damages within the input images
5. The algorithm shall classify damages with a false negative rate of at most **tbd** %

Non-Functional Requirements:

6. The algorithm shall be able to deal with images taken under various lighting conditions
7. The algorithm shall be able to deal with images taken of different aircraft models and colours
8. The algorithm shall be trained on an available inspection dataset
9. The algorithm shall be able to adapt to changes in the training data set

Inspection Image Dataset

For the dataset, the following preliminary requirements hold:

A The dataset shall contain inspection images of aircraft for training the damage detection algorithm specified above

B Every image shall be annotated with the (bounding box) location and class of the present damages

C Every image shall be annotated with the (bounding box) location and class of other **tbd** detectable objects

D Optionally, every image shall be annotated by segmenting it into **tbd** background classes and **tbd** (damage) object classes. The requirements B and C can then be indirectly fulfilled from the segmentation

E Adding and annotating images shall be possible for outside parties (e.g. maintenance engineers)

Bibliography

[1] G Cybenko, “Approximation by superpositions of a sigmoidal function,” *Math. Control. Signals Syst.*, vol. 2, no. 4, pp. 303–314, 1989. [Online]. Available: https://doi.org/10.1007/BF02551274.

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

[3] 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.

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

[5] Y.-J. Cha, W. Choi, and O. Büyüköztürk, “Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks,” *Comput. Civ. Infrastruct. Eng.*, vol. 32, no. 5, pp. 361–378, 2017.

[6] 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,” *Med. Image Anal.*, vol. 35, pp. 18–31, 2017. arXiv: 1505.03540.

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

[8] 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.

[9] 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.

[10] M. Buda, A. Maki, and M. A. Mazurowski, “A systematic study of the class imbalance problem in convolutional neural networks,” *Neural Networks*, vol. 106, 2017. arXiv: 1710.05381v1.

[11] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” 2014. arXiv: 1312.6114v10.

[12] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Nets,” in *NIPS*, 2014, pp. 2672–2680. arXiv: 1406.2661v1.

[13] J. Vanschoren, “Meta-Learning: A Survey,” *CoRR*, 2018. arXiv: 1810.03548v1.

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

[15] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2009.

[16] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” in *ICML*, 2010.

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *CoRR*, 2015. arXiv: 1512.03385.

[18] F.-F. Li, J. Johnson, and S. Yeung, *CS231n Convolutional Neural Networks for Visual Recognition*, 2018. [Online]. Available: http://cs231n.stanford.edu (visited on 10/16/2018).

[19] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *CoRR*, pp. 1–15, 2014. arXiv: 1412.6980.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, 1986.

[21] G. E. Hinton, N Srivastava, A Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *CoRR*, 2012. arXiv: 1207.0580v1.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” in *IEEE*, vol. 86, 1998, pp. 2278–2324. [Online]. Available: http://ieeexplore.ieee.org/document/726791/{\#}full-text-section.

[23] J Deng, W Dong, R Socher, L Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *CVPR*, 2009, pp. 248–255.

[24] R. K. McConnel, *Method of and apparatus for pattern recognition*, 1982.

[25] D. G. Lowe, “Object Recognition from Local Scale-Invariant Features,” in *ICCV*, 1999.

[26] M. Weber, M. Welling, and P. Perona, “Unsupervised Learning of Models for Recognition,” *ECCV*, vol. 1842, pp. 18–32, 2000.

[27] 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.

[28] 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.

[29] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” in *ECCV*, 2014, pp. 818–833. arXiv: 1311.2901v3.

[30] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *ICLR*, 2015. arXiv: 1409.1556v6.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, and A. Arbor, "Going deeper with convolutions," *CVPR*, 2015. arXiv: 1409.4842.

[32] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015. arXiv: 1502.03167.

[33] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *CoRR*, 2015. arXiv: 1512.00567v3.

[34] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017, pp. 1800–1807. arXiv: 1610.02357.

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

[36] 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.

[37] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," *CoRR*, 2016. arXiv: 1611.05431v2.

[38] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017, pp. 2261–2269. arXiv: 1608.06993.

[39] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous Detection and Segmentation," *CoRR*, 2014. arXiv: 1407.1808v1.

[40] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 2980–2988, 2017. arXiv: 1703.06870.

[41] F. Chollet and Others, *Keras*, 2015. [Online]. Available: https://keras.io.

[42] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-scale Machine Learning," in *Proc. 12th USENIX Conf. Oper. Syst. Des. Implement.*, ser. OSDI'16, Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026899.

[43] S. Agarwal and J. Ogier, "Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks," *CoRR*, 2018. arXiv: 1809.03193v1.

[44] T. Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," *CoRR*, 2014. arXiv: 1405.0312v3.

[45] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *IJCV*, vol. 88, no. 2, pp. 303–338, 2010.

[46] M. Everingham, S. M. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective," *IJCV*, vol. 111, no. 1, pp. 98–136, 2014.

[47] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014. arXiv: 1311.2524v5.

[48] J. R. R. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, "Selective Search for Object Recognition," *IJCV*, 2012.

[49] R. Girshick, "Fast R-CNN," in *ICCV*, 2015. arXiv: 1504.08083v2.

[50] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object Detection via Region-based Fully Convolutional Networks," *CoRR*, 2016. arXiv: 1605.06409v2.

[51] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *CoRR*, 2015. arXiv: 1506.02640v5.

[52] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *CVPR*, 2017, pp. 6517–6525. arXiv: 1612.08242.

[53] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *CoRR*, 2018. arXiv: 1804.02767.

[54] 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.

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

[56] T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *CVPR*, 2017, pp. 936–944. arXiv: 1612.03144.

[57] J. Redmon, *Darknet: Open source neural networks in C.* [Online]. Available: https://pjreddie.com/darknet/.

[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 Syst. Appl.*, 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] L. Shen, Z. Lin, and Q. Huang, "Relay backpropagation for effective learning of deep convolutional neural networks," in *ECCV*, 2016. arXiv: 1512.05830.

[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. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Work. Lang. Data Min. Mach. Learn.*, 2013, pp. 108–122. [Online]. Available: http://scikit-learn.org.

[63] 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.

[64] M. D. Richard and R. P. Lippmann, "Neural Network Classifiers Estimate Bayesian a posteriori Probabilities," *Neural Comput.*, vol. 3, no. 4, pp. 461–483, 1991.

[65] Y. Freund and R. E. Schapire, “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting,” *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.

[66] L. Breiman, “Bagging Predictors,” *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[67] 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 Trans. Syst. Man, Cybern. Part C (Applications Rev.*, vol. 42, no. 4, pp. 463–484, 2012.

[68] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, “RUSBoost: A hybrid approach to alleviating class imbalance,” *IEEE Trans. Syst. Man, Cybern. Part ASystems Humans*, vol. 40, no. 1, pp. 185–197, 2010. arXiv: 1011.1669v3.

[69] R. Barandela, J. S. Sánchez, and R. M. Valdovinos, “New Applications of Ensembles of Classifier,” *Pattern Anal. Appl.*, vol. 6, pp. 245–256, 2003.

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

[71] 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.

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

[74] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” pp. 1–12, 2017. arXiv: 1707.06347.

[75] C. Doersch, *Tutorial on Variational Autoencoders*, 2016. arXiv: 1606.05908v2.

[76] S. Kullback and R. A. Leibler, “On Information and Sufficiency,” *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, 1951. [Online]. Available: https://www.jstor.org/stable/2236703.

[77] J. Jorge, R. Paredes, J. A. Sanchez, and M. Bened, “Empirical Evaluation of Variational Autoencoders for Data Augmentation,” *VISAPP*, vol. 5, pp. 96–104, 2018.

[78] K. Sohn, H. Lee, and X. Yan, “Learning Structured Output Representation using Deep Conditional Generative Models,” *NIPS*, pp. 3483–3491, 2015. [Online]. Available: http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.

[79] X. Yan, J. Yang, K. Sohn, and H. Lee, “Attribute2Image: Conditional image generation from visual attributes,” in *ECCV*, 2016. arXiv: 1512.00570.

[80] E. Nalisnick and P. Smyth, “Learning Priors for Invariance,” in *AISTATS*, vol. 84, 2018.

[81] E. Nalisnick and P. Smyth, “Variational Reference Priors,” in *ICLR*, 2017.

[82] F. P. Casale, A. V. Dalca, L. Saglietti, J. Listgarten, and N. Fusi, “Gaussian Process Prior Variational Autoencoders,” in *NIPS*, 2018.

[83] N. Dilokthanakul, P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran, and Murray Shanahan, "Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders," in *ICLR*, 2017. arXiv: 1611.02648v2.

[84] N. Kodali, J. Abernethy, J. Hays, and Z. Kira, "On Convergence and Stability of GANs," 2017. arXiv: 1705.07215.

[85] L. Mescheder, A. Geiger, and S. Nowozin, "Which Training Methods for GANs do actually Converge?," 2018. arXiv: 1801.04406.

[86] A. Brock, J. Donahue, and K. Simonyan, "Large Scale GAN Training for High Fidelity Natural Image Synthesis," 2018. arXiv: 1809.11096v1.

[87] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *CoRR*, 2014. arXiv: 1411.1784v1.

[88] Y. Ci, X. Ma, Z. Wang, H. Li, and Z. Luo, "User-Guided Deep Anime Line Art Colorization with Conditional Adversarial Networks," *ACM Multimed. Conf.*, 2018. arXiv: 1808.03240.

[89] P. Hensman and K. Aizawa, "CGAN-Based Manga Colorization Using a Single Training Image," in *ICDAR*, vol. 3, 2018, pp. 72–77. arXiv: 1706.06918.

[90] A. Odena, C. Olah, and J. Shlens, "Conditional Image Synthesis with Auxiliary Classifier GANs," in *ICML*, 2017. arXiv: 1610.09585v4.

[91] L. Zhang, Y. Ji, and X. Lin, "Style Transfer for Anime Sketches with Enhanced Residual U-net and Auxiliary Classifier GAN," 2017. arXiv: 1706.03319.

[92] T. Tran, T. Pham, G. Carneiro, L. Palmer, and I. Reid, "A Bayesian Data Augmentation Approach for Learning Deep Models," in *NIPS*, 2017. arXiv: 1710.10564.

[93] A. Antoniou, A. Storkey, and H. Edwards, "Data Augmentation Generative Adversarial Networks," *CoRR*, 2017. arXiv: 1711.04340.

[94] X. Zhang, Z. Wang, D. Liu, and Q. Ling, "DADA: Deep Adversarial Data Augmentation for Extremely Low Data Regime Classification," 2018.

[95] A. J. Ratner, H. R. Ehrenberg, Z. Hussain, J. Dunnmon, and C. Ré, "Learning to Compose Domain-Specific Transformations for Data Augmentation," *NIPS*, 2017. arXiv: 1709.01643v3.

[96] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to Learn Using Gradient Descent," *ICANN*, pp. 87–94, 2001.

[97] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

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

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

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

[101] 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.

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

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

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

[105] T. Kim, J. Yoon, O. Dia, S. Kim, Y. Bengio, and S. Ahn, “Bayesian Model-Agnostic Meta-Learning,” *CoRR*, 2018. arXiv: 1806.03836.

[106] C. Finn and S. Levine, “Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm,” in *ICLR*, 2018. arXiv: 1710.11622.

[107] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, “CNN Features off-the-shelf: an Astounding Baseline for Recognition,” in *CVPR*, 2014, pp. 806–813. arXiv: 1403.6382v3.

[108] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks,” *CoRR*, 2013. arXiv: 1312.6229.

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

[110] F.-F. Li, R. Fergus, and P. Perona, “A Bayesian Approach to Unsupervised One-Shot Learning of Object Categories,” in *ICCV*, 2003.

[111] F.-F. Li, R. Fergus, and P. Perona, “One-shot learning of object categories,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 594–611, 2006.

[112] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “One-shot learning by inverting a compositional causal process,” *NIPS*, pp. 1–6, 2013.

[113] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “Human-level concept learning through probabilistic program induction,” *Science (80-.).*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[114] 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.

[115] A. Graves, G. Wayne, and I. Danihelka, “Neural Turing Machines,” *CoRR*, pp. 1–26, 2014. arXiv: 1410.5401.

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

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

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

[119] S. P. Lloyd, “Least squares quantization in PCM,” *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, 1982. [Online]. Available: http://mlsp.cs.cmu.edu/courses/fall2010/class14/lloyd.pdf.

[120] M. Ren, E. Triantafillou, S. Ravi, K. Swersky, J. Snell, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel, "Meta-Learning for Semi-Supervised Few-Shot Classification," in *ICLR*, 2018. arXiv: 1803.00676v1.

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

Glossary

List of Acronyms

AC-GAN	Auxiliary Classifier Generative Adversarial Network (GAN)
AUC	Area Under ROC
BoW	Bag of Word
BDA-GAN	Bayesian Data Augmentation GAN
BPL	Bayesian Program Learning
CNN	Convolutional Neural Network
CGAN	Conditional Generative Adversarial Network
CVAE	Conditional Variational Autoencoder
DA-GAN	Data Augmentation GAN
DADA	Deep Adversarial Data Augmentation
EM	Expectation Maximization
FC	fully-connected
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPN	Feature Pyramid Network
FPR	False Positive Rate
GAN	Generative Adversarial Network
GPU	Graphical Processing Unit

HOG	Histogram of Oriented Gradients
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
IoU	Intersection-over-Union
kNN	k-Nearest Neighbours
LSTM	Long Short-Term Memory
MAML	Model-Agnostic Meta-Learning
MANN	Memory Augmented Neural Network
mAP	mean average precision
MLP	Multi-Layer Perceptron
MM-Net	Memory Matching Network
MN	Matching Network
NTM	Neural Turing Machine
PDF	Probability Density Function
PMN	Prototype Matching Network
PPV	Positive Predictive Value
PN	Prototypical Network
R-CNN	Region-based Convolutional Neural Network
R-FCN	Region-based Fully-Convolutional Network
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
RoI	Region of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SGM	Squared Gradient Magnitude
SIFT	Scale-Invariant Feature Transform
SMOTE	Synthetic Minority Over-Sampling Technique
SSD	Single Shot Detection
SVM	Support Vector Machine

TN	True Negative
TP	True Positive
TPR	True Positive Rate
VAE	Variational Autoencoder
YOLO	You-Only-Look-Once