UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID Official Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Department of Computer Systems and Computation
Universitat Politècnica de València

Wind Turbine Blade Damage Identification using Deep Learning Algorithms

MASTER THESIS

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Author: Juan Pizarro Muñoz

Tutor: Roberto Paredes Palacios

Course 2018-2019

Resumen

El buen estado de las palas en los aerogeneradores es primordial para el correcto funcionamiento y óptima generación de energía. Por ello el monitoreo y reparación de las palas son tareas habituales en el ciclo de vida de un parque eólico.

Las tareas de adquisición de imágenes, revisión y clasificación de la gravedad de los de daños son demandantes en tiempo y requieren personal cualificado, además de ser muy riesgosas ya que algunas se realizan en altura. También es necesario detener los aerogeneradores durante la inspección, lo que disminuye la capacidad de producción energética.

En este TFM el objetivo es evaluar distintos algoritmos de redes de aprendizaje profundo, deep learning networks, para apoyar en la tarea de detección y clasificación de los de daños a partir de fotografías de las palas de los aerogeneradores. Para ello se dispone de imágenes con daños, ej. grietas , agujeros, entre otros, que han sido capturadas en distintos parques eólicos y en distintos modelos de palas.

Palabras clave: Redes de Aprendizaje Profundo, Redes Convolucionales, Detección, Identificación, Clasificación, Daños, Turbina Eólica, Drone, UAV

Abstract

The good condition of wind turbine blades is essential for correct operation and optimum generation of energy. Therefore, monitoring and repairing the blades are common tasks in the life cycle of a wind farm.

The tasks of image acquisition, inspection and damage classification are demanding in time and require expert knowledge, in addition to being very risky since some are performed at height. It is also necessary to stop the wind turbines during the inspection, which decreases the energy production capacity.

This Master Thesis objective is to evaluate algorithms of deep learning networks, to support the task of detection and classification of damages in photographs of the wind turbine blades. To do this, a dataset of images with damages is available, e.g. cracks, holes, among others, images of different blade models captured in different wind farms.

Key words: Deep Learning Networks, Convolutional Networks, Detection, Identification, Classification, Damage, Wind Turbine Blade, Drone, UAV

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

The evolution of machine learning in recent years allows us to address different types of problems obtaining acceptable results, good enough to solve industry problems. It helps us avoid some repeatable task allowing to increase the safety and reliability in the workplace. Thus, Nowadays, it might be possible to find proposals for solutions based on machine learning and deep learning for different computer vision tasks, such as image classification and object detection.

Operation and Maintenance (O&M) in Wind Power Operations costs typically account for 20–25% of the total generation costs for both onshore and offshore wind [51, 2]. Therefore, the energy industry could benefit from the implementation of machine learning projects that help to reduce the cost of operations and maintenance.

Most investigated approaches to the problem of damage detection or condition monitoring in wind turbines use SCADA as a primary data source, in conjunction with classical machine learning (not deep learning) approaches. Also, Wind turbine blades should be periodically inspected looking for cracks, holes and other types of damage to ensure proper and safe operation. Therefore, blade images are taken and commonly organised and classified manually.

This master's thesis explores the use of deep learning for computer vision to find damages on the images of wind turbine blade inspections reviewed and annotated by experts. This project was carried out in conjunction with ROBUR Wind[1] in order to collaborate in the development of a solution proposal for a real wind industry problem and also gave us access to annotated datasets of wind turbine inspections made with drones.

1.1 Motivation

Helping to reduce the time and cost of O&M in the wind industry could make a significant impact on improving people's lives through the use use of more clean energy. As noted by Bhattacharya et al. [5], there is an increasing deployment of renewable energy. That helps in addressing climate change and in creating wider energy access to the billions of people who are still in the poverty trap.

Also, the tasks of image acquisition, inspection and damage classification are demanding in time and require expert knowledge. Some of them are very risky because they are performed at height. It is also necessary to stop the wind turbines during the revision, which decreases the energy production capacity.

[1] https://www.robur-industry-service.com/industriesegmente/wind/

Therefore, the application of deep learning algorithms in this area is an exciting field that could have a huge social impact and great professional development in the future.

1.2 Objectives

The objective of this master's thesis is to evaluate algorithms of deep learning networks, to support the task of detecting and classifying the damages in the photographs of the wind turbine blades.

The proposed objectives are:

- Build a solution proposal for an actual industry problem with real data.
- Study the state of the art of deep networks for image classification and object detection.
- Build an annotated dataset, selecting and cropping images from a set of blade inspection images.
- Build detection and classification systems based on deep learning networks to detect and classify damages on wind turbine blade images.
- Choose the best trained models comparing the obtained results.

1.3 Thesis Structure

This thesis is structured as follows:

- Chapter 1 shows the introduction, motivation and thesis objectives.
- Chapter 2 presents the literature review which support our further development.
- Chapter 3 exposes the datasets and preprocessing techniques.
- Chapter 4 explains models, architectures and parameters available.
- Chapter 5 describes the obtained results with different model configurations.
- Chapter 6 draws conclusions and propose some future work.

CHAPTER 2
Literature Review

This chapter aims to review the relevant literature and research related to image classification and object detection applied to damage detection on wind turbine blades. Image classification or recognition is the task of classifying an image to a particular class out of possible predefined classes. Object detection is the task of determining the location and labels of the objects present in an image.

This chapter is structured as follows:

- Section 2.1 presents the most commonly used metrics in image classification and object detection.
- Section 2.2 presents an overview related to Traditional Object Detection.
- Section 2.3 describes Neural Networks and its most relevant characteristics.
- Section 2.4 presents Applications of Convolutional Neural Network, mostly related to Image classification.
- Section 2.5 describes relevant object detection papers.
- Section 2.6 explains the Single Shot MultiBox Detector model.
- Section 2.7 presents a historical literature review of Computer Vision for Wind Turbine Condition Monitoring.

2.1 Performance Measure

This section presents the performance measures used to evaluate the trained models for image classification and object detection.

2.1.1. Precision and Recall

Precision is the answer to *How many selected items are relevant?*. It is the ratio between the number of true positives and the total of selected items, as shown in Equation 2.1. The Recall is the answer to *How many relevant items are selected?*. Which corresponds to the ratio between the number of true positives and the total of items, as shown in Equation 2.2. A graphical representation of precision and recall is shown in Figure 2.1.

$$P = \frac{TP}{TP + FP} \tag{2.1}$$

$$R = \frac{TP}{TP + FN} \tag{2.2}$$

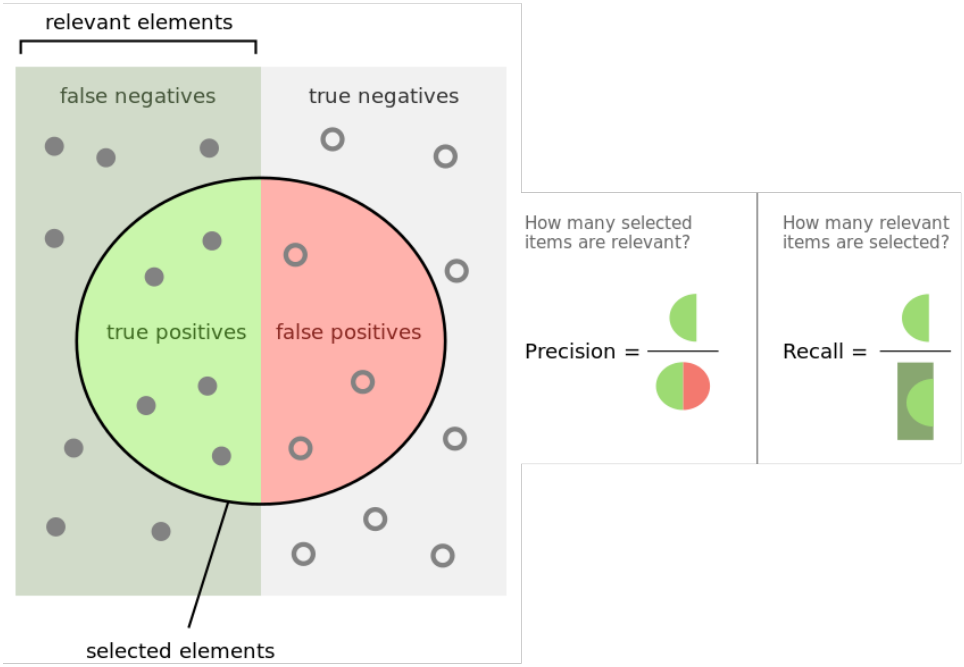


Figure 2.1: Precision and Recall.
Image Source https://commons.wikimedia.org/wiki/File:Precisionrecall.svg.

2.1.2. IoU

When the object localisation is predicted, a commonly used metric is the jaccard index or intersection-over-union (IoU). It corresponds to the ratio between the overlap of the predicted location and a ground truth location, and the area of the union of both locations as shown in Figure 2.2.

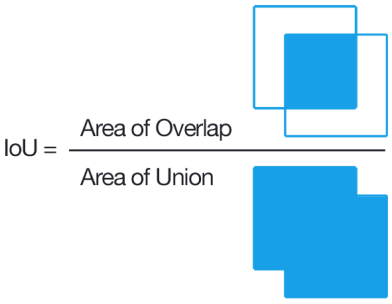


Figure 2.2: IoU. Image Source https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

2.1.3. Mean Average Precision

The mean average precision (mAP) is defined as the average of the maximum precisions at different recall values. In the case of object detection systems, this measure is calculated for specific IoU thresholds.

2.2 Traditional Object Detection

Traditional object detection methods are built on handcrafted features and shallow trainable architectures [57]. Most of them can be divided into three main steps: candidate region proposal, feature extraction and classification. For feature extraction, hand-designed shallow features such as Scale Invariant Feature Transform (SIFT) [29] and HOG (Histogram of Oriented Gradients) [9] were commonly used. In 2001, Viola et al. [52] proposed the use of haar-like features and a cascading classifier named AdaBoost [14] to classify faces. Then in 2004, Lowe [29] presented the SIFT (Scale Invariant Feature Transform) algorithm, a method for extracting distinctive invariant features from images using difference-of-Gaussian. The classification could be done with SVMs [7], Random Forests [6] or AdaBoost [14].

On the other hand, Deng and Yu [10] noted that features like SIFT and HOG only capture low-level edge information. It is more challenging to design features that effectively capture mid-level information such as edge intersections or high-level representation such as object parts.

2.3 Neural Networks

According to Goodfellow et al. [18] there have been three waves of development of deep learning. The first wave started with cybernetics in the 1940s–1960s, with the development of theories of biological learning [30, 21] and implementations of the first models such as the perceptron [38] allowing the training of a single neuron. The second wave started with the connectionist approach of the 1980–1995 period, with back-propagation [40] to train a neural network with one or two hidden layers. The current and third wave, deep learning, started around 2006 [22, 4, 34].

First, in Subsection 2.3.1 The Perceptron is presented. Later, in Subsection 2.3.2 the Feedforward Neural Networks are described. After that, in Subsection 2.3.3 the Parameter Learning is explained. Then, in Subsection 2.3.4 some common Activation Functions are shown. Finally, in Subsection 2.3.5 the Convolutional Neural Networks are presented.

2.3.1. The Perceptron

In the context of neural networks, a perceptron is an artificial neuron, illustrated in Figure 2.3, that can be seen as a weighted linear combination of the inputs followed by an activation function. It takes D real numbers as inputs $(x_1, x_2, \ldots, x_{D-1}, x_D)$, each input is weighted and summed together, and a bias term w_0 is added, as shown in Equation 2.3.

In Equation 2.4, the activation function f is applied to $z(x)$ in order to obtain the output $s(x)$. It is important to note that this function could be non-linear, e.g. a step function with 0 as threshold is shown in Equation 2.5.

$$z(\mathbf{x}) = \sum_{i=1}^{D} \omega_i x_i + w_0 = w^T x + w_0 \tag{2.3}$$

$$s(\mathbf{x}) = f(z(x)) \tag{2.4}$$

$$f(z) = \left\{ \begin{array}{ll} 1(+1) & \textit{if } z > 0 \\ 2(-1) & \textit{if } z < 0 \end{array} \right\} \tag{2.5}$$

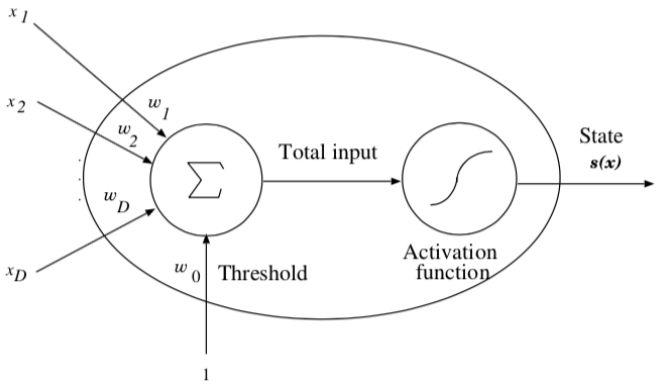


Figure 2.3: Single layer perceptron.

The weights w_d and the bias w_0 are parameters that should be determined or learned. If the problem is linearly separable, a perceptron and the Perceptron algorithm [38] could be used.

2.3.2. Feedforward Neural Network

Deep feedforward networks, also often called feedforward neural networks, or multi-layer perceptrons (MLPs), have the goal to approximate some function f^* [18]. It defines a mapping $y = f(x;\theta)$, whose parameters θ should be learnt to obtain the best function approximation to the unknown function f^*.

As shown in Figure 2.4, the model could be seen as a composition of multiple Perceptrons. It is associated with a directed acyclic graph describing how the functions are composed together. For example, we might have three functions f^1, f^2, and f^3 connected in a chain, to form the output $f(x) = f^3(f^2(f^1(x)))$. In this case, f^1 is called the first layer of the network, f^2 is called the second layer, and so on.

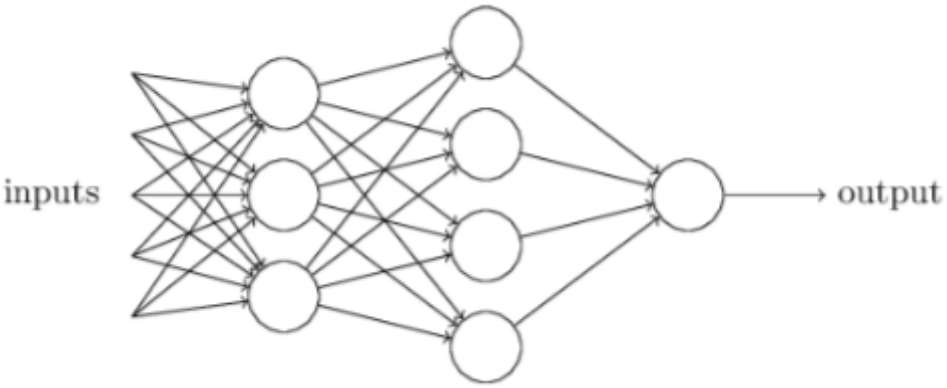


Figure 2.4: Multi-layer perceptron. Source [31].

In addition, a feedforward operation or propagation is called to the evaluation of the function f being from the input layer x, through the intermediate layers, and finally to the output.

2.3.3. Parameter Learning

As noted previously, the parameters θ should be determined. These parameters could be estimated as the parameters whose produce the smallest error on a set of annotated

samples. Therefore, a cost function $J(\theta)$, also called objective function, must be defined to measure how well the model is predicting the values compared with the ground truth labels using the function parameters θ.

Given a cost function $J(\theta)$ defined by the parameters θ, the problem to learn θ can be formulated as:

$$\hat{\theta} = \arg\min_{\theta} J(\theta) \tag{2.6}$$

2.3.3.1. Gradient Descent

Gradient descent, shown in Equation 2.7, compute a local minimum of $J(\theta)$ with respect to parameters θ. It is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \in R^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t. to the parameters [39].

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.7}$$

2.3.3.2. Back-propagation

The back-propagation, presented by Rumelhart et al. [40], is an algorithm that computes the chain rule of calculus, with a specific order of operations that is highly efficient.

During training, forward propagation can continue onward until it produces a scalar cost $J(\theta)$. Then a back-propagation algorithm allows the information from the cost to then flow backwards through the network to compute the gradient at each node or neuron.

According to Goodfellow et al. [18], the term back-propagation is often misunderstood as meaning the whole learning algorithm for multi-layer neural networks. Back-propagation refers only to the method for computing the gradient, while another algorithm, such as stochastic gradient descent, should be used to perform learning using this gradient, that means to update the θ parameters or network weights.

2.3.3.3. Cost Function

The chosen cost function is closely related to the prediction task. Indeed, it needs to measure how good, or bad the model is predicting comparing with the ground truth label.

Usually, the inputs for a loss function defined on a single data point are the ground truth label y and the predicted value $\hat{y}$. In general, a mean quadratic error function could be used for regression problems, and a cross-entropy function could be used for classification problems.

2.3.4. Activation Functions

To be able to solve more complex or nontrivial problems, nonlinearities should be are added to the neural network. This could be done using nonlinear activation functions.

The most common activation functions are (some of them are shown in Figure 2.5):

Linear

$$f_L\left(z_j\right) = z_j \tag{2.8}$$

Step

$$f_E(z_j) = \left\{ \begin{array}{ll} 1 & if\, z_j > 0 \\ 0 & if\, z_j < 0 \end{array} \right\} \tag{2.9}$$

ReLU (rectified linear unit)

$$f_R(z_j) = \max(0, z_j) \tag{2.10}$$

PReLU (parametric rectified linear unit)

$$f_{PR}(z_j) = \left\{ \begin{array}{ll} z_j & if\, z_j > 0 \\ az_j & if\, z_j \leq 0 \end{array} \right\} \tag{2.11}$$

Sigmoid

$$f_S(z_j) = \frac{1}{1 + \exp(-z_j)} \tag{2.12}$$

Hyperbolic tangent

$$f_T(z_j) = \frac{\exp(z_j) - \exp(-z_j)}{\exp(z_j) + \exp(-z_j)} \tag{2.13}$$

Softmax

$$f_{SM}(z_j) = \frac{\exp(z_j)}{\sum_{j'} \exp(z_{j'})} \tag{2.14}$$

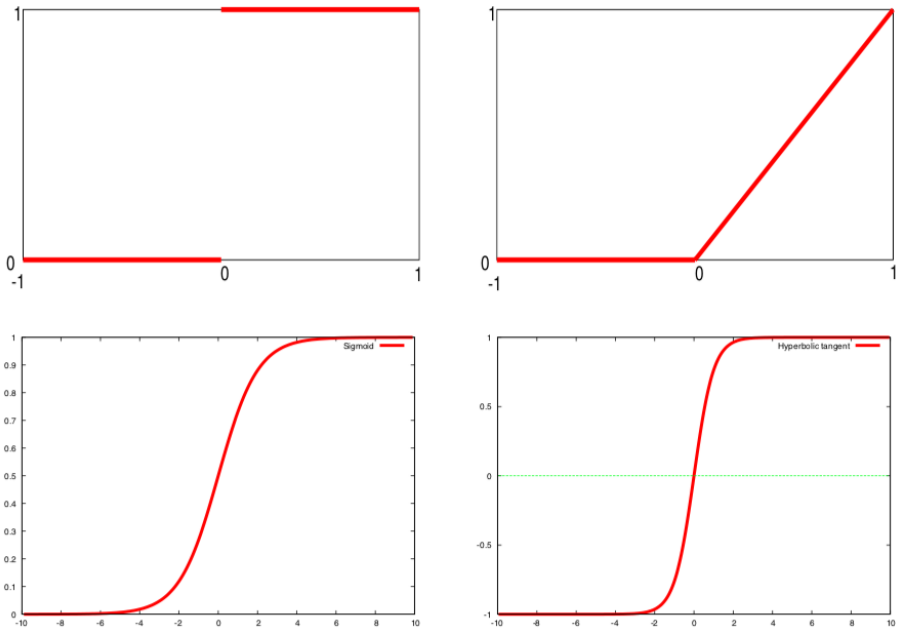


Figure 2.5: Activation functions. The step, the ReLU, the Sigmoid and the Hyperbolic tangent activa-tion functions.

2.3.5. Convolutional Neural Networks

Convolutional neural networks, also called CNN or ConvNets, are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [18].

Traditional neural network layers compute for each neuron in a layer a weighted linear combination of the inputs coming from the previous layer. Also, it can be seen as a matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit, as illustrated in Equation 2.3. That means every output unit could interact with every input unit.

Convolutional networks, however, typically have sparse interactions, as they interact with a subsection of the previous layer neurons. They can retain some spatial information, which is essential in computer vision tasks.

2.3.5.1. Convolutional Layer

The convolution can be seen as an element-wise multiplication between a matrix of parameters and a subsection of an input matrix or previous layer's outputs. In Figure 2.6, it can be seen how each output value or neuron interacts with only a subsection of the input values or previous layer neurons, that means every output value is a kind of local representation of the subsection with which it is interacting.

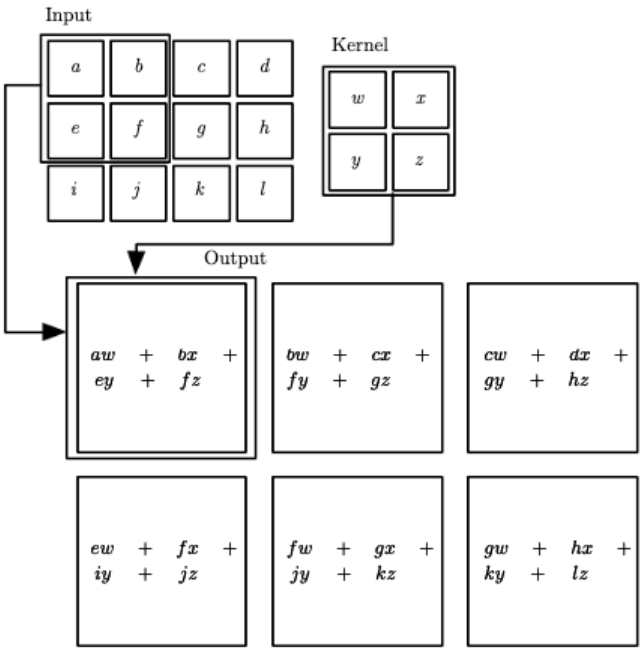


Figure 2.6: An example of 2D convolution. Source [18, Figure 9.1].

2.3.5.2. Pooling Layer

To reduce the computational cost, a pooling operator could be used, which also deal with multi-scale and capture higher-level features. The Figure 2.7 shows that the output values could be the average (left) or the max value (right) of the neurons with which they interact.

For example, the value 15 of the green box in the left square corresponds to the average of the values 21,8,12,19. The value 21 of the green box in the right square corresponds to the maximum of the values 21,8,12,19.

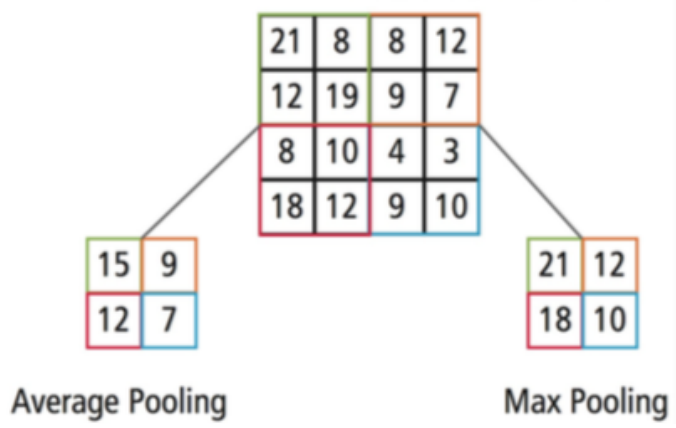


Figure 2.7: Pooling Layer kernel size of 2, and stride of 2. Image source https://ip.cadence.com/uploads/901/cnn_wp-pdf

2.3.6. Regularization

A trained model can make very few errors on the sample data on which it is trained but performs poorly on new data. This situation, typically referred to as overfitting, is exacerbated by complex models as the neural networks, and small samples [12].

This section shows the most common techniques to avoid overfitting in neural networks.

2.3.6.1. Data Augmentation

Bengio et al. [3] proposed to generate additional training data by applying local affine-transformations to enhance the robustness to variations in position, size, orientation, and other distortions.

Following these ideas, nowadays, it is very common to use transformations such as flipping, rotation, scaling, adding salt and pepper noise, varying lighting condition and perspective transform. It is so common that most of them are already implemented by the deep learning framework to be used out of the box.

2.3.6.2. Gaussian Noise

Zur et al. [58] investigated the effect of a noise injection method on the overfitting problem of artificial neural networks (ANNs) in two-class classification tasks. They found that training artificial neural networks with noise injection can reduce overfitting, it could be greater than using early stopping and similar as using weight decay.

This technique is commonly used together with Batch Norm because the weight range would be known that makes it easier to find how much noise should be added.

2.3.6.3. Dropout

Dropout presented by Srivastava et al. [45] is a simple way to prevent neural networks from overfitting. Its key idea is to randomly drop units (along with their connections) from the neural network during training, as shown in Figure 2.8.

Essentially random variables control the activation of the neurons, and at test time, outgoing weights are multiplied by the probability that the neuron has been retained.

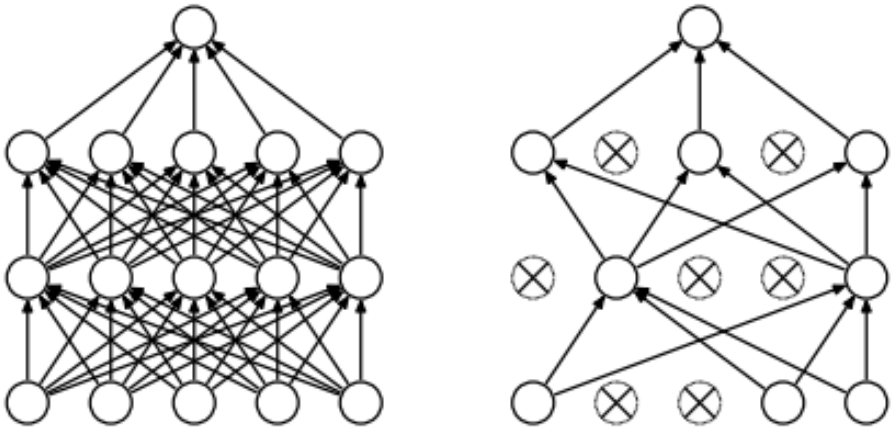


Figure 2.8: Dropout. Image Source [45].

2.3.6.4. MixUp

Zhang et al. [56] proposed to train neural networks on convex combinations of pairs of examples and their labels to alleviate undesirable behaviors such as memorization and sensitivity to adversarial examples.

It is a simple and data-agnostic data augmentation routine, which constructs virtual training examples as follows:

$\tilde{x} = \lambda x_i + (1 - \lambda) x_j$, where x_i, x_j are raw input vectors

$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$, where y_i, y_j are one-hot label encodings

Typical values of λ are in the range of $[0.2 - 0.4]$

2.3.7. Transfer Learning

Neural networks rely on the availability of a large amount of labeled data to train a model. However, labeled data are often scarce and expensive to obtain [8]. When there is no such a large amount of labeled data, it is possible to use transfer learning to improve the performance on a target learning task using the network weights of a trained model for similar learning tasks on a large labeled dataset [35].

According to Pan and Yang [32], transfer learning aims to extract the knowledge from one or more source tasks and applies that knowledge to a target task, as shown in Figure 2.9. They also proposed the following formal definition:

Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(\dots)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.

In the context of neural networks, we could identified two general approaches to transfer learning: feature extraction and finetuning.

Feature extraction It is done by using the vector produced by a forward pass of a trained neural network, and then training a classifier or a regressor using that feature vector dataset.

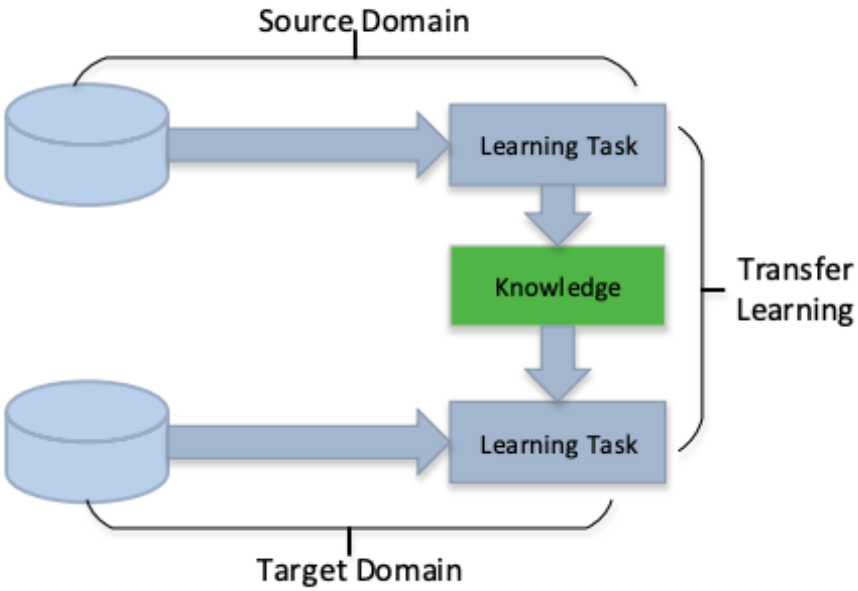


Figure 2.9: Transfer Learning. Image Source [49].

Finetuning It is done by initializing a neural network with the weights of a pretrained network optimised for a large labeled dataset like ImageNet, and then updating the network's weights using a different, commonly smaller dataset [27].

2.4 Applications of Convolutional Neural Network

The concept of convolutional neural network (CNN) was originally introduced by Fukushima [15] in 1980, but just in 1998 LeNet [26] was the first successful deep CNN.

This section presents the most relevant architecture designs to carry out mostly Image Classification tasks: LeNet-5 in Subsection 2.4.1, AlexNet in Subsection 2.4.2 Overfeat in Subsection 2.4.3, VGG in Subsection 2.4.4, GoogLeNet in Subsection 2.4.5 and ResNet in Subsection 2.4.6.

2.4.1. LeNet-5

In 1998, Lecun et al. [26] proposed a network for Handwritten Digit Recognition trained on the MNIST dataset. This network could be considered as the first successful deep CNN.

The architecture, shown in Figure 2.10, has two convolutional layers, each followed by an average pooling layer, followed by two consecutive fully connected layers. The activation functions are sigmoid and tanh.

2.4.2. AlexNet

AlexNet, illustrated in Figure 2.11, proposed by Krizhevsky et al. [25] won the ImageNet ILSVRC challenge 2012. The architecture contains eight layers with weights; the first five are convolutional, and the remaining three are fully connected. The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels.

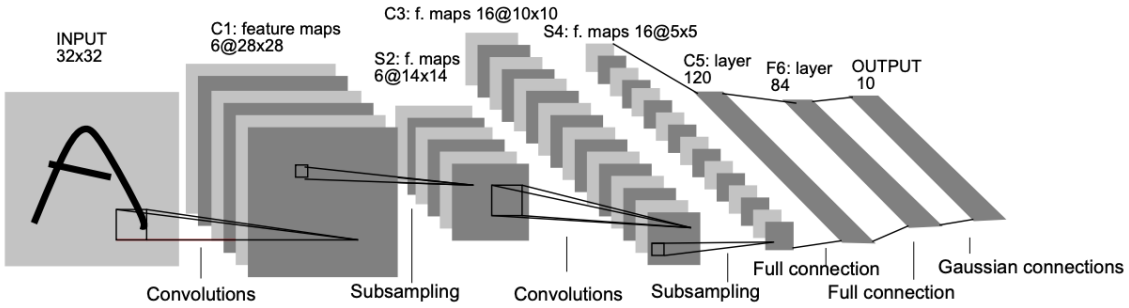


Figure 2.10: Architecture of LeNet-5, a convolutional neural network used for digits recognition. Each plane is a feature map, i.e., a set of units whose weights are constrained to be identical [26].

The architecture is similar to LeNet, but use ReLU as activations and Dropout for regularization. It also increases the input size to 224x224 pixels.

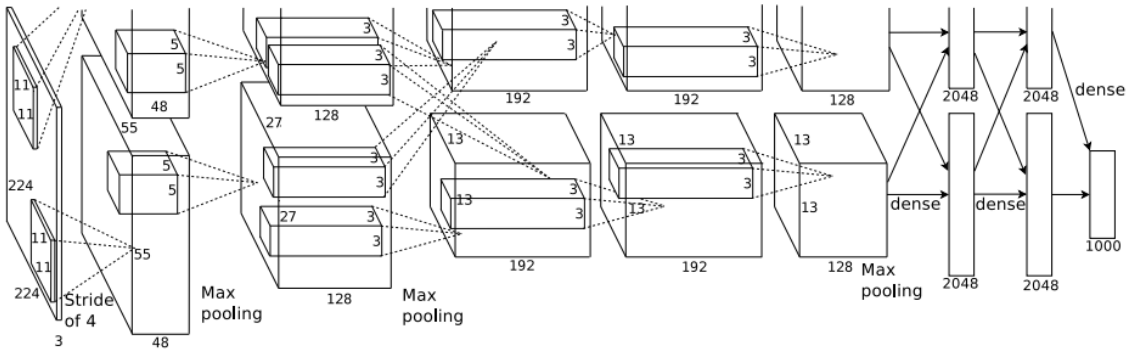


Figure 2.11: Architecture of AlexNet [25].

2.4.3. Overfeat

In 2013, Overfeat presented by Sermanet et al. [42] proposed the idea to efficiently use of CNN for detection using a sliding windows approach.

During training, a CNN produces only a single spatial output (Figure 2.12 top). But when applied at test time over a larger image, it produces a spatial output map, e.g. 2x2 (Figure 2.12 bottom). Since all layers are applied convolutionally, the extra computation required for the larger image is limited to the yellow regions.

2.4.4. VGG

In 2014, VGG team [44] won the localisation challenge of ILSVRC 2014 [41]. It is also the 1st runner-up in the task of object detection in the ILSVRC 2014.[1]

As noted by Khan et al. [24], VGG replaced the 11x11 and 5x5 filters with a stack of 3x3 filters layer and experimentally demonstrated that concurrent placement of 3x3 filters can induce the effect of the large size filter, reducing the computational cost.

2.4.5. GoogLeNet

GoogLeNet presented by Szegedy et al. [48] won the ILSVLC 2014 competition and is also known as Inception-V1. Its main contribution was the inception block, which per-

[1] http://image-net.org/challenges/LSVRC/2014/results

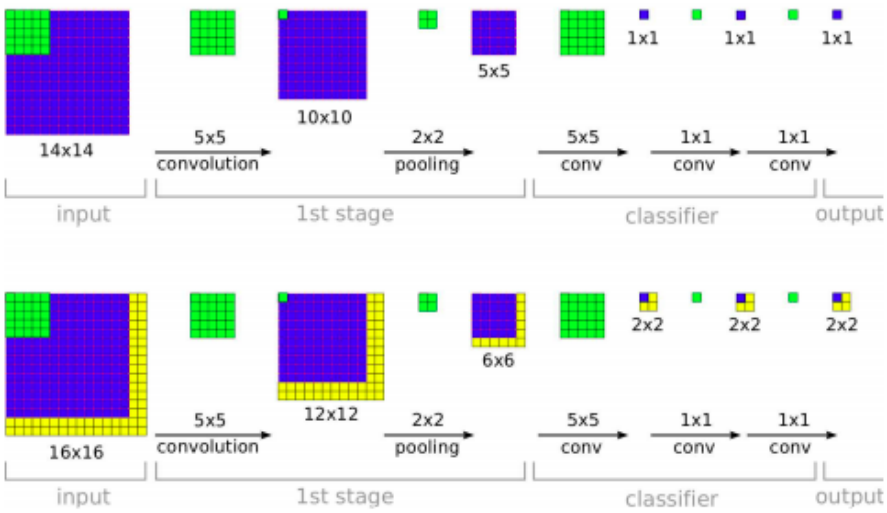


Figure 2.12: The efficiency of ConvNets for detection [42, Figure 5].

forms multiple convolutions with different kernel sizes in parallel and concatenates their outputs, reducing the number of parameters in the network.

2.4.6. ResNet

Presented by He et al. [20] in 2015, ResNet with a depth of over 150 layers won the image classification, detection, and localization in ILSVRC 2015. [2]

The new key idea is to fit residual mappings instead of mappings, as illustrated in Figure 2.13. ResNet introduced shortcut connections within layers to enable cross-layer connectivity. However, those shortcuts or gates are data-independent and parameter-free in comparison to Highway Networks [46] presented in 2015 too.

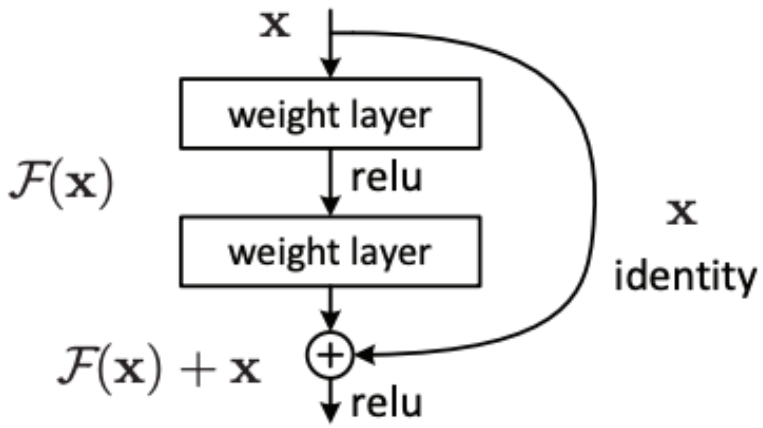


Figure 2.13: Residual learning: a building block [20].

2.5 Object Detection

Object detection is the task of identifying all objects from a specific closed-set of pre-defined classes by putting a bounding box around each object present in an image [1].

[2] http://image-net.org/challenges/LSVRC/2015/results

Overfeat, presented in Subsection 2.4.3, was the first publication related to efficient object detection with CNN. However, the most impressive advancements to use a convolutional neural network to tackle the task of object detection were based on the classical three-step machine learning approach: selective search, feature extraction and classification. This family of algorithms called region convolutional neural networks is presented in Subsection 2.5.1. Then, In Subsection 2.5.2 recent architectures designs based on the idea of single-stage object detector are presented.

2.5.1. Region CNN Networks

2.5.1.1. R-CNN

In 2013, R-CNN was presented by Girshick et al. [17]. As shown in Figure 2.14, they applied the three-step strategy: selective search (e.g. Uijlings et al. [50]), feature extraction (e.g. SIFT [29]) and classification (e.g. SVM [7]); but a CNN was used to extract the features representation from each region proposal instead of a commonly used handcreafted feature extraction algorithms.

They resize the selected image proposal region (yellow boxes in Figure 2.14) to fit in a CNN. Each region is represented with a fixed-size vector extracted from the Fully-Connected (FC) layer from Alexnet.

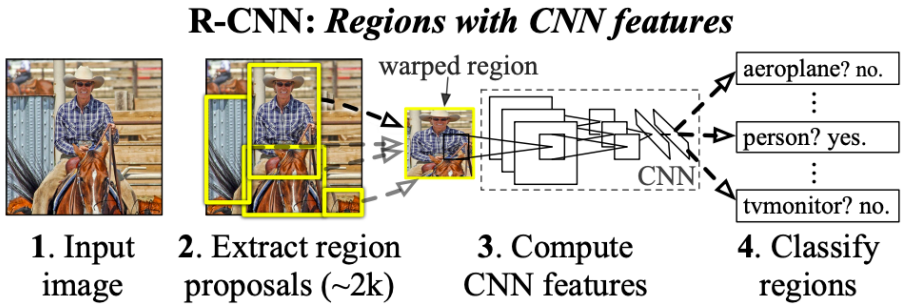


Figure 2.14: R-CNN. Image source [17].

2.5.1.2. Fast R-CNN

In 2015, Girshick [16] presented Fast R-CNN improving the speed and accuracy of the R-CNN. As R-CNN is slow because it performs a CNN forward pass for each object proposal without sharing computation, Fast R-CNN perform only one forward pass. Therefore the regions re-use this same forward. As shown in Figure 2.15 the bounding boxes (red) are projected to the feature map coordinates. They also used a more accurate backbone network, VGG16 and data augmentation to achieve scale-invariant object detection.

2.5.1.3. Faster R-CNN

In 2015, Ren et al. [37] introduced a Region Proposal Network (RPN) that shares full-image convolutional features with the detection network. Thus enabling nearly cost-free region proposals, reducing the infer time drastically.

An RPN, shown in Figure 2.16, is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position, which is trained end-to-end to generate high-quality region proposals. After that, Fast R-CNN for detection is used.

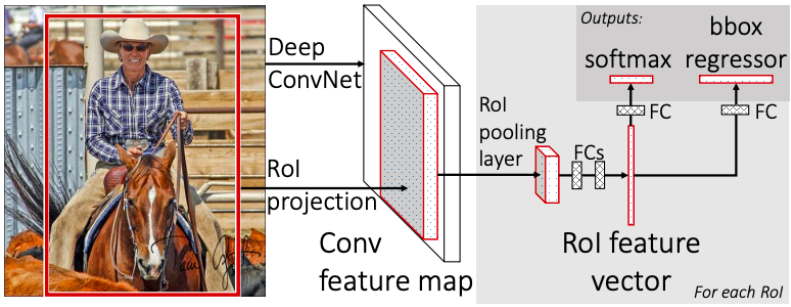


Figure 2.15: Fast R-CNN. Image source [16].

In addition, RPN and Fast R-CNN are merged into a single network by sharing their convolutional features.

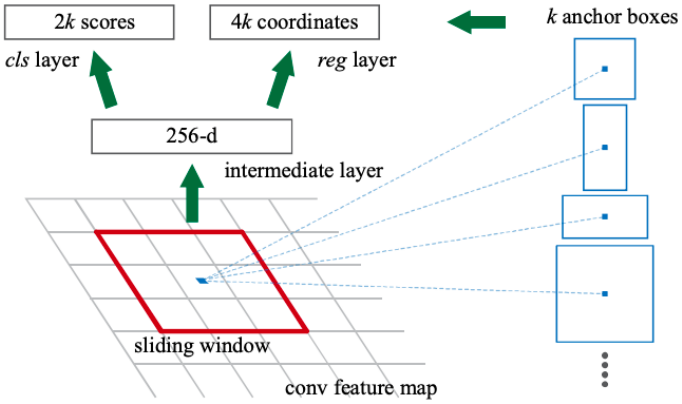


Figure 2.16: Faster R-CNN. Image source [37].

2.5.2. Single-Stage Object Detector

Another approach to using CNN for object detection is the so called single shot or single stage detectors, since both object localisation and classification are done within a single feed-forward through the network.

In 2015, Redmon et al. [36] presented YOLO, were they frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Also In 2015, Liu et al. [28] presented a single deep neural network, called Single Shot MultiBox Detector (SSD), which discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. This neural network is described in Section 2.6.

2.6 Single Shot MultiBox Detector (SSD)

This section presents the most relevant concepts of the SSD model presented by Liu et al. [28].

SSD framework is shown in Figure 2.17. It takes an image and the ground truth boxes (blue and red boxes in (a)) for each object during training as input. Then, in a convolutional fashion, a small set (e.g. 4) of default boxes of different aspect ratios at each

location in several feature maps from multiple layers for prediction at different scale are evaluated (e.g. $8x8$ and $4x4$ in (b) and (c)).

Finally, for each default box, both the shape offsets and the confidences for all object categories ($(c_1, c_2, \ldots, c_p)$) are predicted.

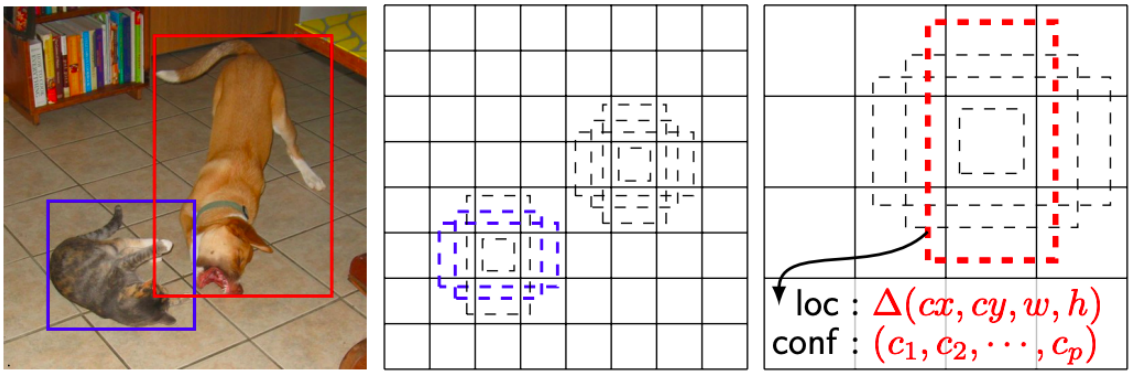


Figure 2.17: bbox of SSD [28].

2.6.1. Architecture

The Architecture is based on a truncated basic CNN, trained on image recognition. The use of a VGG16 is proposed, but other CNN could also be used.

As seen in Figure 2.18, the SSD model uses some feature layers directly from the base network: *conv4_3* and *conv7/FC7*; and also adds several feature layers to the end of a base network that decreases in size progressively: *conv8_2*, *conv9_2*, *conv10_2* and *conv11_2*. Each feature layer predicts the offsets to default boxes and their associated class confidences. Using feature maps from different layers allows predicting at different scales. This base network is called the backbone network.

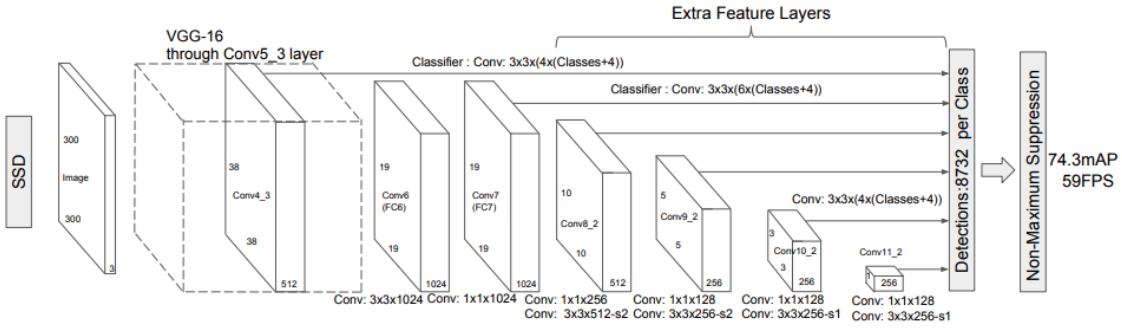


Figure 2.18: Architecture of SSD [28].

In fact, SSD with an $300x300$ input size significantly outperforms its $448x448$ YOLO [36] counterpart in accuracy on VOC2007 test while also improving the speed.

2.6.2. Training

The SSD training objective or cost function is derived from the MultiBox objective [13] but is extended to handle multiple object categories. Let $x_{ij}^p = \{1, 0\}$ be an indicator for matching the i-th default box to the j-th ground truth box of category p. The matching

variable x_{ij}^{p} is 1 when the jaccard overlap or IoU between the ground truth and the default bounding box is higher than a threshold (e.g. 0.5). The overall objective loss function, shown in Equation 2.15, is a weighted sum of the localization loss (loc) and the confidence loss (conf), weighted by a parameter α.

$$L(x,c,l,g) = \frac{1}{N}\left(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)\right) \tag{2.15}$$

Then, the confidence loss function is illustrated in Equation 2.16. It is a softmax loss over multiple classes confidences (c).

$$L_{conf}(x,c) = -\sum_{i\in Pos}^{N} x_{ij}^{p}\log\left(\hat{c}_i^{p}\right) - \sum_{i\in Neg}\log\left(\hat{c}_i^{0}\right) \tag{2.16}$$

where $\hat{c}_i^{p} = \frac{\exp\left(c_i^{p}\right)}{\sum_p \exp\left(c_i^{p}\right)}$

Next, the localisation loss function is shown in Equation 2.17. It is a Smooth L1 loss [17] between the predicted box (l) and the ground truth box (g).

$$L_{loc}(x,l,g) = \sum_{i\in Pos}^{N}\sum_{m\in\{cx,cy,w,h\}} x_{ij}^{k}\text{smooth}_{\text{L1}}\left(l_i^{m} - \hat{g}_j^{m}\right) \tag{2.17}$$

Finally, the ground truth information needs to be assigned to specific outputs in the fixed set of detector outputs. Each ground truth box is matched to the default box with the best jaccard overlap higher than a threshold (e.g. 0.5). This define which is the positive and negative training samples. Because there could be a significant imbalance between the positive and negative training examples, the use of a subset of the negative samples with the highest confidence loss is proposed. It is so called hard negative mining ratio.

2.6.3. Data Augmentation

Data augmentation is crucial, according to Redmon et al. [36] there is a gain of 8% mAP when data augmentation is used. The following subsections will describe the strategies that could be called Zoom-In and Zoom-Out.

2.6.3.1. Zoom-In Strategy

A zoom-in operation to make the model more robust to various input object sizes and shapes is proposed. Many larger training examples are randomly generated by one of the following options:

- Use the entire original input image.
- Sample a patch so that the minimum jaccard overlap (IoU) with the objects is 0.1, 0.3, 0.5, 0.7, or 0.9.
- Randomly sample a patch.

2.6.3.2. Zoom-Out Strategy

Because the classification task for small objects is relatively hard for SSD, the use of zoom-out data augmentation to create more small training examples with the idea of improving

the accuracy is proposed. This operation is so called expansion. First, the images are randomly placed on a canvas of 16x of the original image size filled with mean values. Then, a random crop is extracted.

2.7 Computer Vision for Damage Detection on Wind Turbines

Most models applied to wind turbine condition monitoring use SCADA or simulated data, with almost two-thirds of methods using classification and the rest relying on regression, only a few approaches utilised images [47].

However, in recent years, some research has been conducted to detect damage to the wind turbine blade using images as a data source. In 2017, Wang and Zhang [53] used Unmanned Aerial Vehicles (UAVs) taken images to detect surface cracks in blades using the Viola-Jones framework [52]. The same authors in 2019 proposed a two-stage approach for detecting surface cracks on the wind turbine blades via analyzing blade UAVs taken images. The first stage consists of a locating cracks method based on extracting Haar-like features, and the second stage consists of obtaining crack contours based on boundaries of crack segments [54].

It is important to keep in mind that recently, in 2019, it is possible to find some research using deep learning. Shihavuddin et al. [43] developed a deep learning-based automated damage suggestion system by analysis of drone inspection images. They used faster R-CNN with an advanced augmentation step called the "Multi-scale pyramid and patching scheme" that enables the network to achieve better learning. In the 52nd CIRP Conference on Manufacturing Systems placed on June 2019[3], Denhof et al. [11] propose to automate the visual surface inspection of Wind Turbine Rotor Blades with convolutional neural networks (CNN) using models such as DenseNet, VGG and ResNet.

[3] https://www.cirp-cms2019.org/

CHAPTER 3
Datasets

This chapter presents the datasets and preprocessing techniques.

First, in Section 3.1 the dataset creation and protocols are presented. Later, in Section 3.2 the prediction tasks are described. Then, in Section 3.3 the preprocessing techniques are explained. Finally, in Section 3.4 the preprocessed datasets are shown.

3.1 Dataset Creation and Protocols

Datasets of an ongoing wind turbine inspection were used. More than 1000 wind towers would be reviewed, and their damage classified.

First, a small dataset with image samples was reviewed to understand the type of images and annotations they planned to produce in the coming months. It was also important to know about the data format and image quality.

Later, the first real dataset called ds-00, which contained drone images from 12 inspections, was accessed. One *inspection* corresponds to the revision of one tower, which contains one *mission* folder for each of its wind turbine blades. In general, a tower has three blades, so there must be three missions for each tower. After that, a dataset called ds-01 was accessed, which consisted of information on 154 valid inspections according to our selection protocol. The selection protocol is presented in Subsection 3.1.1. Next, another dataset was accessed, called ds-02, which contains information about 261 valid inspections.

In this thesis, most of our experiments were performed using the dataset ds-00 and a subset of the dataset ds-01. It is estimated that it corresponds to 1/8 of all available data.

However, the evaluation of multiple deep learning models configurations on the dataset ds-02 is beyond the scope of this thesis, due to the size of the data set, the limited hardware capacity available to do our experiments and the time available for this master's thesis.

The images were captured in a resolution of 5472x3648 pixels. Every damage has been annotated with a bounding box, severity and type. Five levels of severity and 15 damage types were used. The damage types are: bubbles, cast, chipped coat, coat fault, crack around lightning bolt, crack around spl bolt, cracks, damaged laminate, erosion, lightning damage, lightning hit- receptor, noise, paint erosion on smt, rub mark, and scratch.

Image samples of bubbles and cast are shown in Figure 3.1 and Figure 3.2 respectively. In addition, in Appendix B on page 52 more image samples are presented.

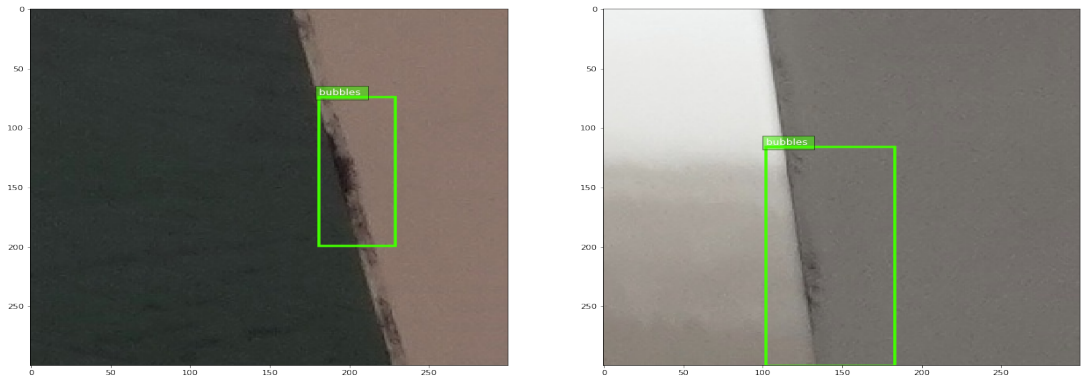


Figure 3.1: Bubbles

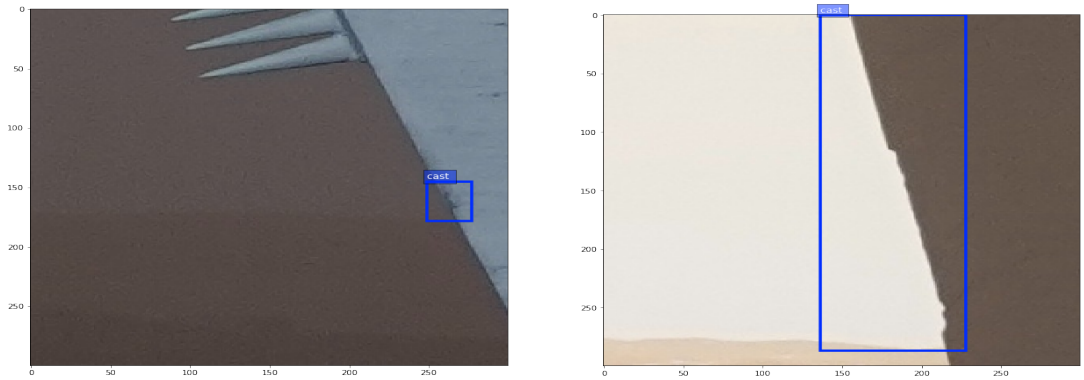


Figure 3.2: Cast

3.1.1. Inspection Selection Protocol

The inspections were carried out by two teams, one of them has more experience than the other. The first revision was carried out by the less experienced team whose main outcome was to try to locate all possible damages. Then, the most experienced team reviewed the proposed locations to verify if there were actual damages. They also were in charge of classifying the severity and type. A graphical representation of this inspection process is shown in Figure 3.3.

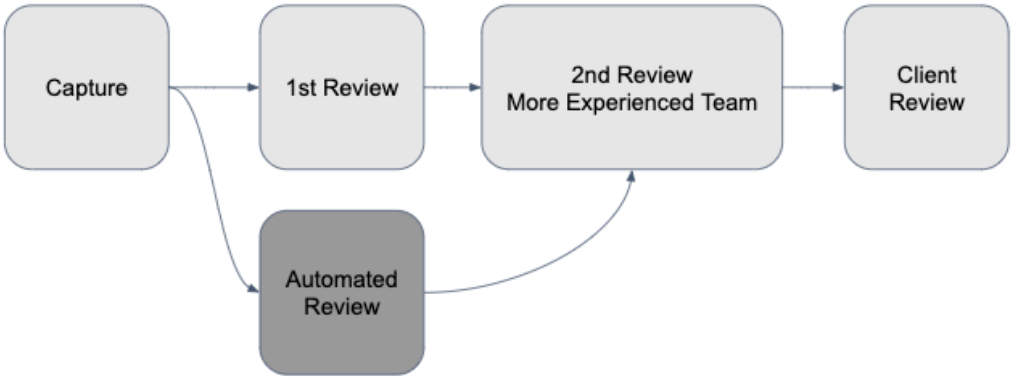


Figure 3.3: Damage inspection process

Because it was obtained access to all revisions, but not all of them were reviewed by the most experienced team at that time, a selection protocol was defined. A valid inspection is defined as those that were reviewed by the most experienced team. In addition,

some of them were reviewed by more than one team member of the more experienced team. For that reason, the data of the last revision was selected.

3.1.2. Datasets

In Table 3.1 the number of inspections, missions, images and damages of all datasets are shown. Also, it shows if a dataset contains only valid inspections according to the defined selection protocol of Subsection 3.1.1.

It can also be seen that the number of valid revisions of the dataset ds-01 is almost half of the dataset ds-02, 154 and 261 respectively, but the total number does not increase too much, from 602 to 689. In addition, it can be seen that in ds-01, there are 21906 damages, but in ds-01-val-01 there are only 4385 verified.

Table 3.1: Inspection datasets

Dataset	Inspections	Missions	Images	Damages	Valid only
ds-00	12				
ds-01	602	1797	16928	21906	n
ds-01-val-01	154	461	3352	4385	y
ds-02	689	2056	19197	24823	n
ds-02-val-01	261	780	6168	7986	y

Each data set has been divided into 85% for train and 15% for dev (validation), ensuring that all images and annotations of a mission are in train or dev exclusively.

3.1.2.1. ds-01-val-01 dataset

The ds-01-val-01 dataset is imbalanced as can be seen in Table 3.2. There are some damage types with only one sample, and there is one class with almost 50% of all samples. Also, there are damages of type *n/a* that were annotated during the 1st review but were not reviewed by the most experienced team. The damage inspection process is described in Subsection 3.1.1.

In addition, in Table 3.3, it can be appreciated that more than 50% of the boxes are less than 50 pixels wide, which is very small compared to the full resolution size of the images. It can be seen the same behaviour with the box heights. Then, in Table 3.4 can be seen that the mean width is 63.36 pixels, and the mean height is 95.5 pixels. It also shows that 75% of the boxes are less than 62 pixels wide or less than 70 pixels high.

3.1.2.2. ds-02-val-01 dataset

When the ds-02-val-01 dataset was analysed, similar patterns to those of the ds-01-val-01 dataset presented in Subsection 3.1.2.1 were found.

The damage type is imbalanced as shown in Table 3.5. The damage type *coat fault* has more than 50% of all annotated damages. Then, in Table 3.6 can be seen that more than 60% of the bounding boxes are very small, they are less than 50 pixels wide. Finally, the Table 3.7 shows that the average width and height are less than 65 pixels.

Table 3.2: Damage type samples of the ds-01-val-01 dataset

Damage Type	Samples
coat fault	2411
chipped coat	789
n/a	301
erosion	248
noise	174
bubbles	157
lightning hit- receptor	87
rub mark	52
paint erosion on smt	33
scratch	23
*** undefined ***	23
cracks	23
damaged laminate	22
crack around lightning bolt	18
lightning damage	15
cast	10
crack around spl bolt	1
tip break damage	1

Table 3.3: Percentage of bounding box damages grouped by height or width of the ds-01-val-01 dataset

Pixel Range	Height (%)	Width (%)
(0, 10]	0.00	0.00
(10, 20]	0.16	0.15
(20, 30]	0.26	0.28
(30, 50]	0.24	0.25
(50, 100]	0.16	0.20
(100, 200]	0.09	0.08
(200, 300]	0.03	0.02
(300, 10000]	0.07	0.03

3.2 Prediction Tasks

It is possible to define different use cases using one single dataset. For that reason, different prediction tasks are defined based on the related use case and available data.

A prediction task is defined by its target labels. In some cases, the labels (damage type) in the dataset may differ from the target labels of the prediction task. For example, a target label called *damage* could represent multiple damage types. As shown in Table 3.8, the labels could be the same as the source labels found in the dataset or artificial labels like *damage* or *other*. When an artificial label is used, an index map to do a mapping from the source labels to the target labels should be applied. In Table 3.9, the index maps are shown.

Table 3.4: Bounding box size statistics of the ds-01-val-01 dataset

Statistic	Width	Height	Area
count	4388.00	4388.00	4388.00
mean	63.37	95.50	21767.42
std	127.17	218.54	206072.28
min	10.00	11.00	143.00
25%	24.00	23.00	576.00
50%	34.00	35.00	1292.00
75%	62.00	70.00	4219.50
max	3993.00	3602.00	10094304.00

Table 3.5: Damage type samples of the ds-02-val-01 dataset

Damage Type	Samples
coat fault	4879
chipped coat	1155
erosion	352
n/a	305
bubbles	261
crack around lightning bolt	235
noise	197
paint erosion on smt	186
lightning hit- receptor	156
*** undefined ***	60
rub mark	56
lightning damage	43
cracks	29
scratch	26
damaged laminate	24
cast	17
crack around spl bolt	6
missing seal between smt & shell	3
tip break damage	1

3.3 Preprocessing

The drone inspection images were captured in a high resolution of $5472x3648$ pixels. For that reason, it was opted to extract crops of the images to decrease hardware requirements. They were also scaled to a lower resolution looking to reduce the training memory requirements. These strategies are based on the ideas of Shihavuddin et al. [43] and Xia et al. [55].

For object detection tasks, a sliding window strategy was used to crop only sections of the image that contain annotated damages. Fixed windows size and fixed stride were used. Then, when the IoU between the cropped damage and the original damage is below a threshold, the damage is marked as difficult.

In the case of image classification tasks, the damages were cropped using the ground truth bounding boxes. Then, each cropped damage is re-scaled to a fixed width and height.

Table 3.6: Percentage of bounding box damages grouped by height or width of the ds-02-val-01 dataset

Pixel Range	Height (%)	Width (%)
(0, 10]	0.00	0.00
(10, 20]	0.14	0.12
(20, 30]	0.26	0.27
(30, 50]	0.26	0.27
(50, 100]	0.18	0.20
(100, 200]	0.09	0.09
(200, 300]	0.02	0.02
(300, 10000]	0.05	0.02

Table 3.7: Bounding box size statistics of the ds-02-val-01 dataset

Statistic	Width	Height	Area
count	7991.00	7991.00	7991.00
mean	62.75	80.66	15411.55
std	105.89	175.79	154783.07
min	10.00	11.00	143.00
25%	25.00	24.00	638.00
50%	36.00	36.00	1400.00
75%	63.00	64.00	3958.00
max	3993.00	3602.00	10094304.00

To differentiate each preprocessed dataset, a nomenclature was defined to name the transformed dataset according to the transformation or preprocessing strategy that was applied.

The possible suffixes for object detection tasks are:

- $sWxH$ when a scale of width W and height H was applied.
- cS when square crops of size S were extracted.
- vN to specify the dataset version N, in case of changes in the software or the extraction strategy.

The possible suffixes for image classification tasks are:

- rsN when damages are extracted and re-scaled to a fixed size of NxN pixels.

For example, *ds-01-val-01-c300-v01* is based on the *ds-01-val-01* dataset and square crops of size 300 pixels were extracted, it was also identified as the first version of the preprocessed dataset.

These preprocessing strategies were implemented as python scripts:

- reducing_big_images.py: Used to extract patches of specific offset or size. It was implemented using impy[1].

[1] https://github.com/lozuwa/impy

Table 3.8: Target labels

Name	Target labels
labels	All the labels
labels_v1	*lightning hit-receptor, chipped coat, coat fault, erosion, bubbles*
labels_v2	*coat fault, other*
labels_v3	*coat fault*
labels_v4	*coat fault, chipped coat*
labels_v5	*chipped coat*
labels_v6	*damage*
labels_v7	All the labels, but not including *N/A* nor *undefined* type
5c	Predict five label (image classification)

Table 3.9: Index maps

Name	Index map
index_map_15c2d	all type are mapped to 0
index_map_2c2d	*coat fault* and *chipped coat* are mapped to 0
index_map_3c2d	*coat fault, chipped coat* and *erosion coat* are mapped to 0
index_map_v2	*coat fault* is mapped to 0, other types are mapped to 1

- reducing_big_images_aug.py: Used to apply scale transformations. It was implemented using impy.
- voc_crop.py: Used to extract patches using a window of a fixed size and step size or stride. It was implemented using MXNet[2] and GluonCV[3] [19].
- voc2cls_crop.py: Used to extract crop of damages used in image classification tasks. It is base on voc_crop.py.

3.4 Preprocessed Datasets

The datasets used in the first series of experiments were preprocessed using the python scripts based on impy which are shown in Table 3.10. However, most of our experiments use the datasets listed in Table 3.11, which were preprocessed using the Python scripts based on MXNet and GluonCV. In the other hand, for image classification tasks, the preprocessed datasets shown in Table 3.12 were created.

[2] https://mxnet.apache.org
[3] https://gluon-cv.mxnet.io

Table 3.10: Preprocessed datasets using impy

Name	Description
ds-00-reduced1000	patches with size of 1000
ds-00-reduced400	patches with size of 400
ds-00-reduced600	patches with size of 600
ds-01-s5472x3648-c1000	scaled to 5472x3648 patches with size of 1000
ds-01-s5472x3648-c1000-s300x300	scaled to 5472x3648 patches with size of 1000 scaled to 300x300
ds-01-s5472x3648-c1000-s512x512	scaled to 5472x3648 patches with size of 1000 scaled to 512x512
ds-01-val-01-c1000-v01	patches with size of 1000

Table 3.11: Preprocessed datasets for object detection tasks

Name	Description
ds-01-val-01-c1000-v02	Crop of 1000, stride 500, IoU=1.
ds-01-val-01-c1000-v03	Crop of 1000, stride 900, difficult<.7
ds-01-val-01-c1000-v04	Crop of 1000, stride 500, IoU>0, difficult<.7
ds-01-val-01-c300-v01	Crop of 300, stride 150, IoU>0, difficult<.7
ds-01-val-01-c512-v01	Crop of 512, stride 256, IoU>0, difficult<.7

Table 3.12: Preprocessed datasets for image classification tasks

Name	Description
ds-01-val-01-rs224v01	re-scaled to 224

CHAPTER 4
Methods

This chapter presents the models and parameters available.

First, in Section 4.1 the Models and Architectures are presented. Then, the Section 4.2 shows the training and validation parameters.

4.1 Models and Architectures

For object detection tasks, MXNet[1] and GluonCV[2] [19] were used to implements the models and architectures. It was opted to use those libraries because they already have model implementations to base on to modify and build the specific models and features evaluated in this project. In addition, they obtained good performance in evaluations carried out to compare different SSD implementations and different frameworks.

For image classification tasks the implementations and experiments have been carried out using PyTorch[3] [33] and the FastAI library[4] [23].

4.1.1. SSD Backbones for Object Detection

The SSD architecture presented in Section 2.6 is based on extending a CNN, called backbone network. In our experiments, VGG and ResNet were used. The Table 4.1 shows the backbones based on VGG and the Table 4.2 shows a sample list of the backbones based on ResNet. In addition, it shows the feature maps from which the features are extracted and how many additional layers are added. A complete list of the used SSD-ResNet backbone networks can be found in Table A.1 on page 51.

Table 4.1: SSD backbones based on VGG

Backbone network name	Description
vgg16_atrous	vgg16 atrous without batch norm
vgg16_atrous_02	vgg16 atrous with batch norm

[1] https://mxnet.apache.org
[2] https://gluon-cv.mxnet.io
[3] https://pytorch.org/
[4] https://github.com/fastai/fastai

Table 4.2: Compact list of SSD backbones based on ResNet

Backbone network name	Description
resnet18_v1	Last activation of stage 3 y 4 + 4 extra
resnet18_v1_02	Last activation of stage3 + 5 extra
resnet18_v1_02_gn	Last activation of stage3 + 5 extra + gaussian noise
resnet18_v1_03_02_01_numfilters01	Last activation of stage3 + 2 extra of 256
resnet18_v1_03_02_01_numfilters02	Last activation of stage3 + 2 extra of 128
resnet18_v1_03_02_ratios01	Last activation of stage3 + 1 extra + $[1, 2, 0.5, 3, 1.0/3]$
resnet50_v1	Last activation of stage 3 y 4 + 4 extra

4.1.2. Image Classification

The available datasets were relative small. The ds-01-val-01 dataset consists of less than 5000 images with damages. For that reason the use of Transfer Learning (Subsection 2.3.7) was proposed.

Pre-trained models could be utilised, replacing their last layers to match the shape of our prediction tasks. Then the training could be done in two steps. First, only the newly added layers are optimised for some epochs. Then, all layer could be trained. It may also be useful to add intermediate layers between the pre-trained network activations and the last fully-connected layer that has as many units a number of classes or labels.

For example, for a ResNet trained in ImageNet, we should remove the last fully connected layer of 1000 neurons or units, replacing it with a new layer, e.g. a layer with five neurons for the prediction task *5c* because that task consists of predicting five labels.

4.2 Training and Validation Parameters

In Subsection 4.2.1, the parameter settings available for object detection tasks are presented. Then, the parameter settings for image classification tasks are described in Subsection 4.2.2.

It is important to note that the training and validation parameters presented in this section could produce more combinations than those evaluated in the scope of this thesis.

4.2.1. Object Detection Parameters

The parameters could be divided into architectural, data, training and validation parameters. The Table 4.3 shows the architectural parameters, like the network architecture and the backbone network. Then, the Table 4.4 shows the data parameters, like the train and validation dataset. Depending on the prediction task, the target labels and index map must be chosen. The general training parameters like the initial learning rate and batch size are shown in Table 4.5. Also, It is possible to use pre-trained model weights for the backbone network offered by the MXNet and GluonCV projects.

In addition, model specific training parameters are shown in Table 4.6. Some of them configure how the data augmentation is performed or how the loss function is calculated. The Table 4.7 shows the validation parameters. These parameters could also be changed during the testing time to measure the performance of a trained model with different metric parameters.

Table 4.3: Architectural parameters

Parameter name	Description
network architecture	SSD Section 2.6
backbone network	Subsection 4.1.1
data-shape	Pixel size of the input images

Table 4.4: Data parameters

Parameter name	Description
train dataset	Dataset used for training Section 3.4
validation dataset	Dataset used for validation Section 3.4
label	Target labels Section 3.2
index-map	Section 3.2
train-skip-difficult	If the difficult damages of the training dataset should be skipped
val-skip-difficult	If the difficult damages of the validation dataset should be skipped
mixup	If mixup data augmentation should be used

4.2.2. Image Classification Parameters

The image classification training could be done in two steps. First, the newly added layers are set as trainable, and the remaining layers are frozen. This training step can be done for a maximum number of epochs *learn_fit_1*. Then, a finetune step where all layers are set as trainable can be done for a maximum number of epochs *learn_fit_2*. A fixed learning rate could be used for each step, or when the learning rate value is less than zero the learning rate finder function[5] is used. The parameters are shown in Table 4.8.

[5] https://docs.fast.ai/basic_train.html#lr_find

Table 4.5: General training parameters

Parameter name	Description
pretrained-base	If pre-trained weights for the backbone network should be used
lr	Learning rate
batch-size	Batch size
epochs	Epochs

Table 4.6: Training parameters

Parameter name	Description
train-transform-random-crop-with-constraints	Random crop constraints
max-ratio	Max ratio
loss-negative-mining-ratio	Ratio of negative vs. positive samples
loss-min-hard-negatives	Minimum number of negatives samples
loss-lambd	Relative weight between classification and box regression loss
train-transform-iou-thresh	IOU overlap threshold for maximum matching
val-transform-iou-thresh	IOU overlap threshold for maximum matching

Table 4.7: Validation parameters

Parameter name	Description
val-metric-iou-thresh	IOU overlap threshold for TP for mAP Metric

Table 4.8: Image classification parameters

Parameter name	Description
path_ds	Path where the train and val datasets are located
classes_sel	Classes or damage types to predict
bs	Batch size
img_size	Image size
ps	Drop out rate added to the newly added layers
base_arch_name	Name of architecture
oversample	Oversample strategy
mixup	Mixup data augmentation
lr_fr	Learning rate for fine tuning
lr_un	Learning rate for training
learn_fit_1	Epochs for fine tuning
learn_fit_2	Epochs for training
np_random_seed	
lin_ftrs	Intermediate hidden sizes. An array specifying the number of units per layer added between the pre-trained network activations and the last fully-connected layer or labels (Default 512)

CHAPTER 5
Results

Because the images were taken in high resolution, it is proposed first to try to locate the damages in the images, then try to classify each damage as its type.

The Section 5.1 describes the results of several experiments performed to evaluate different object detection model configurations to locate damages on wind turbine blade images. Then, in Section 5.2 the results of the experiments with image classification models to classify each damage as its damage type are shown.

5.1 Object Detection

This section describes the results of several experiments performed to evaluate different SSD model configurations to locate the damages.

Our first obtained results using a preprocessing strategy applied by the python scripts based on impy are presented in Subsection 5.1.1. Later, the results of predicting two labels: *coat fault* or *other* are shown in Subsection 5.1.2. Then, the results of predicting one label: *coat fault* are shown in Subsection 5.1.3. Next, in Subsection 5.1.4 the results of predicting two labels: *coat fault* or *chipped coat* are presented. After that, the Subsection 5.1.5 presents the results of predicting the label *damage*.

5.1.1. Results of Predicting Each Damage Type Separately

In this section, our first experimental results are presented.

Learning Rate

One of the first steps to start training a neural network is trying to find a good learning rate. Therefore, vgg16_atrous models were trained with the learning rate range $[1.0, 0.00001]$. The Table 5.1 shows good learning rate values, values greater than 0.0005 are not shown because cause the loss to explode or the mAP to jump between lower values.

Should Damage Types be Grouped?

When the exploratory analysis of the dataset was performed, it was found that some damage type looks very similar. It seems that the variation between some classes is small. Therefore, models to predict the five damage types with more samples were trained. Also, models to predict the damage type with more samples and the rest types as *other*

Table 5.1: VGG16 models trained to predict labels v1 with different learning rates

lr	validation_map
0.0001	0.324
0.0005	0.308
5e-05	0.297
1e-05	0.233
1e-06	0.133

label were trained. This was done to find out if grouping some damages significantly improves the mAP.

As shown in Table 3.8 on page 23, labels *v1* corresponds to the target labels of the five damage types with more samples and labels *v2* corresponds to the damage type with more samples and a second label *other*. An index map to map the damages that are different than *coat fault* to the label *other* should be used.

The Table 5.2 shows that the model trained with labels *v2* got the highest mAP.

Table 5.2: VGG16 models trained to predict labels v1 and v2

labels	validation_map
v2	0.374
v1	0.324

Crop Constraints Data Augmentation

There are a bunch of very small damages on the dataset, so it was decided to evaluate the use of smaller damage image sections in the data augmentation phase. The original crop constraints from Liu et al. [28] were compared with two alternatives to try to obtain bigger damage image resolution.

The Table 5.3 shows that the original constraints obtained the highest mAP.

Table 5.3: VGG16 models trained to predict labels v2 with different crop constraints

crop_constraints	validation_map
def	0.354
new	0.345
new	0.255

Backbone Networks: VGG16, VGG13, VGG11

The Table 5.4 shows that better mAP is obtained when a more powerful model is used as the backbone network. The VGG16 got the highest mAP.

Summary

Table 5.5 presents a summary of the obtained results.

Table 5.4: Backbone networks: vgg16, vgg13, vgg11

network	validation_map
vgg16_atrous	0.344
vgg13_atrous	0.303
vgg11_atrous	0.287

Table 5.5: Summary of predicting each damage type separately

	validation_map
vgg16_atrous, labels=v1, lr=0.0005	0.308
+ lr=0.0001	0.324
+ labels=v2	0.374

5.1.2. Results of Predicting the *coat fault* Type and the Rest Types as *other* Label

In this section, the obtained results of trained models to predict a damage as *coat fault* or *other* are presented. It corresponds to the labels *v2* of Table 3.8 on page 26. It is important to note that *coat fault* is the damage type with more samples in the dataset.

Crop Size Pre-processing

Because the input images are re-scaled to the model input size and there are very small damages, models with a crop of 300, 500 and 1000 pixels were trained to evaluate how the crop size affects the performance. A stride of the half of the crop size was used, as shown in Table 3.11 on page 27.

In Table 5.6, it can be seen that when the crop size is reduced, a higher mAP is obtained.

Table 5.6: VGG16 models trained to predict labels v2 with different crop sizes

train_dataset	validation_map
ds-01-val-01-c300v01	0.376
ds-01-val-01-c512v01	0.335
ds-01-val-01-c1000v04	0.289

Batch Norm

SSD models were trained with the vgg16_atrous as the backbone, with and without using batch norm. The models with batch norm were trained from scratch and the models without batch norm used a pre-trained vgg16_atrous.

The model with batch norm obtained the lowest mAP as can be seen in Table 5.7. This could be because the batch size was small compared to the one used for training the VGG16 backbone model without batch norm. A batch size of 16 was used due to the available hardware resources for training.

Table 5.7: VGG16 models trained to predict labels v2 with and without batch norm

network	validation_map
vgg16_atrous	0.380
vgg16_atrous_02	0.282

Summary

Table 5.8 presents a summary of the results of trained models with vgg16_atrous, labels=v2, lr=0.0001.

Table 5.8: Summary of predicting the *coat fault* type and the rest types as *other* label

	validation_map
crop_size=1000	0.289
+ crop_size=512	0.335
+ crop_size=300	0.376
+ vgg16_atrous with batch norm	0.282

5.1.3. Results of Predicting the *coat fault* Label

In this section, the obtained results of trained models to predict the damage type *coat fault* are presented. It is the type with more samples in the dataset. This prediction task corresponds to the labels *v3* of Table 3.8 on page 26. To evaluate the SSD300 model, image crops of 300 pixels in size were extracted.

Backbone Networks

The original SSD publication used VGG as a backbone network for its generalisation capabilities, but other convolutional networks could also be used. Later, an SSD with ResNet101 backbone network was published on the paper's git repo[1]. For that reason, it was decided to evaluate the SSD with ResNet backbone.

The SSD-ResNet34 version implemented in GluonCV was modified, following the paper's git repo[2] implementation. Some differences were found between the version of the authors of SSD and the version of GluonCV.

The Table 5.9 shows the results of using different pre-trained models as backbone: VGG16, ResNet18 or our modified ResNet34. It can be seen that when the VGG16 or our modified ResNet34 is used, a similar mAP is obtained.

Table 5.9: Backbone networks: vgg16_atrous, resnet18_v1 and resnet34_v1_02

network	validation_map
resnet34_v1_02	0.681
vgg16_atrous	0.679
resnet18_v1	0.444

[1] https://github.com/weiliu89/caffe/tree/ssd
[2] https://github.com/weiliu89/caffe/blob/ssd/examples/ssd/ssd_pascal_resnet.py

ResNet as Backbone with 1 Feature Map

Different feature map layers from the backbone network could be used, so models were trained with only one feature map: the last activation of stage2 or the last activation of stage3 of a ResNet34.

In Table 5.10, it can be seen that the model with the last activation of stage3 as feature map obtained the best results, both models ending in *_02* obtained higher mAP than the ones ended on suffix *_01*.

Table 5.10: ResNet as backbone with 1 feature map

network	validation_map
resnet34_v1_02_02	0.674
resnet34_v1_04_02	0.666
resnet34_v1_02_01	0.654
resnet34_v1_04_01	0.630

ResNet as Backbone with 1 or 2 Feature Maps

The SSD300 network could use layers of the backbone network directly as feature maps, or the ones that are added on top of the backbone network.

Models were trained using the last activation of two layers directly from the backbone and adding four extra feature map layers, this networks ending in *_00*. Also, models were trained using the last activation of only one layer directly from the backbone and adding five extra feature map layers, this networks ending in *_02* or *_03*.

The Table 5.11 shows that using the last activation of stage3 and adding five extra layers obtained higher mAP. Then using the last activation of stage2 and adding five extra layers obtained the second higher mAP. Finally, using the last activation of stage3, the last activation of stage4, and adding four extra layers obtained the lowest mAP.

It is important to note that the *resnet34_v1_00* network is heavily based on the *resnet18_v1* already implemented in GluonCV. The network ending in *_02* or *_03* corresponds to our modified RestNet implementations.

Table 5.11: ResNet as backbone with 1 or 2 feature maps

network	crop_constraints	validation_map
resnet34_v1_02	new	0.681
resnet34_v1_02	def	0.679
resnet34_v1_03	def	0.660
resnet34_v1_03	new	0.643
resnet34_v1_00	new	0.532

Summary

Table 5.12 presents a summary of the results of trained models with labels=v2, lr=0.0001 and crop_size=300.

Table 5.12: Summary of predicting the *coat fault* label

	validation_map
resnet18_v1	0.444
+ resnet34_v1_00 (GluonCV resnet34_v1)	0.532
+ resnet34_v1_03 (2 feature from backbone + 4 extra)	0.643
+ resnet34_v1_02 (1 feature from backbone + 5 extra)	0.681

5.1.4. Results of Predicting Two Labels: *coat fault* or *chipped coat*

In this section, the results of trained models to predict *coat fault* or *chipped coat* as damage type are presented. Those are the two damage types with more samples in the dataset. To evaluate the SSD300 model, image crops of 300 pixels in size were extracted. This prediction task corresponds to the labels *v4* of Table 3.8 on page 26.

Zoom-Out Data Augmentation

Liu et al. [28] proposed a strategy to improve the performance on small object detection. They proposed to use a *zoom out* operation to create more small training examples, as presented in Subsection 2.6.3 on page 18. Therefore, it was evaluated whether using a zoom out of 4x or not using it will make significant changes in the performance.

The Table 5.13 shows that when the max ratio is increased, a lower mAP is obtained. This could happen because there are too many small damages in our studied dataset.

Table 5.13: Zoom-Out Data Augmentation

crop_constraints	max_ratio	validation_map
new	1.0	0.465
def	4.0	0.434
new	4.0	0.425

Hard Negative Mining

Because a significant imbalance between the positive and negative training examples occurs, Liu et al. [28] proposed to sort them using the highest confidence loss for each default box and pick the top ones, specifically a ratio of 3:1 was good for the evaluated dataset.

In our case, the ratios 3:1, 5:1 and 15:1 were evaluated. It can be seen in Table 5.14 that when the loss negative mining ratio is increased, a higher mAP is obtained.

Table 5.14: Hard negative mining ratios: 3:1, 5:1 and 15:1

negative_mining_ratio	network	validation_map
15.0	resnet34_v1_02_gn	0.497
15.0	resnet34_v1_02	0.495
5.0	resnet34_v1_02	0.480
3.0	resnet34_v1_02	0.478

Zoom-In Data Augmentation

Because the studied *ds-01* dataset has a bunch of very small damages, it was evaluated to extract patches with a smaller minimum jaccard overlap threshold than the used by Liu et al. [28]. It corresponds to the zoom-in strategy presented in Subsection 2.6.3 on page 18.

The Table 5.15 shows when the original (def) crop with constraints thresholds was used, a lower mAP is obtained.

Table 5.15: ResNet34 with random crop with constraints thresholds

crop_constraints	network	validation_map
new	resnet34_v1_02	0.491
new	resnet34_v1_02_gn	0.488
def	resnet34_v1_02	0.480

Jaccard Overlap Threshold in Matching Strategy

During training, and also during validation, default boxes are matched to any ground truth with jaccard overlap higher than a threshold. Liu et al. [28] used a threshold of 0.5.

In Table 5.16 can be observed a huge improvement in the mAP when the val_iou_thr value is reduced to 0.3.

Table 5.16: Jaccard Overlap Threshold

tr_iou_thr	val_iou_thr	val_met_iou_thr	network	validation_map
0.3	0.3	0.3	resnet34_v1_02	0.571
0.5	0.5	0.5	resnet34_v1_02_gn	0.497
0.5	0.5	0.5	resnet34_v1_02	0.495
0.3	0.3	0.5	resnet34_v1_02	0.477
0.3	0.5	0.5	resnet34_v1_02	0.468

Each threshold column is described as follows:

- *tr_iou_thr* is the threshold used for the matching strategy during training.
- *val_iou_thr* is the threshold used for the matching strategy during validation.
- *val_met_iou_thr* is the IoU threshold for accepting a box as true positive (TP) when calculating the mAP metric.

Summary

Table 5.17 presents a summary of the results of trained models with resnet34_v1_02, labels=v2, lr=0.0001 and crop_size=300.

5.1.5. Results of Predicting All Types as *damage* Label

In this section, the results of trained models to predict all damage types as *damage* label are presented. An index map to map all source damage type labels to the *damage* label

Table 5.17: Summary of predicting two labels: *coat fault* or *chipped coat*

	validation_map
negative mining ratio=3, max_ratio=4	0.425
+ max_ratio=1	0.465
+ negative mining ratio=5	0.480
+ negative mining ratio=15	0.495
+ iou_thr=0.3	0.571

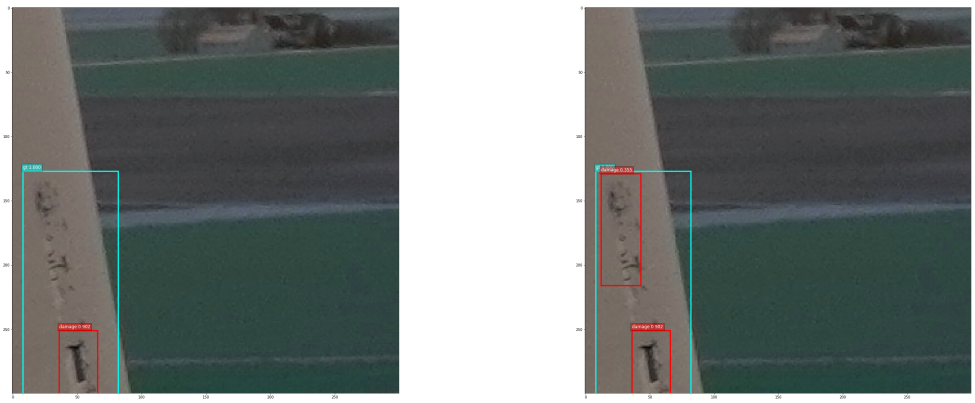
was used. To evaluate the SSD300 model, image crops of 300 pixels in size were extracted. This prediction task corresponds to the labels *v6* and the index map *index_map_15c2* described in Table 3.8 on page 26.

Similar performance can be obtained when using a ResNet34 as a backbone network with or without using mixup data augmentation, as shown in Table 5.18. However, when Gaussian noise is used, it seems that using mixup produce better results. It can also be seen that using ResNet with Gaussian noise and mixup obtains comparable results as using ResNet without Gaussian noise.

Table 5.18: Mixup Data Augmentation

mixup	network	validation_map
True	resnet34_v1_02	0.646
False	resnet34_v1_02	0.643
True	resnet34_v1_02_gn	0.642
False	resnet34_v1_02_gn	0.630

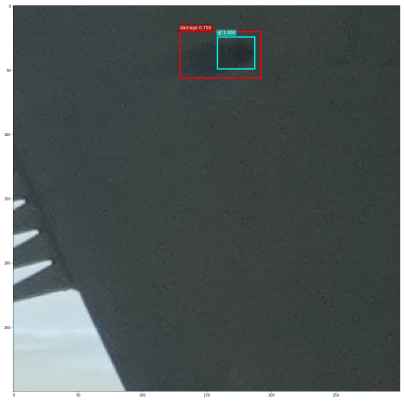
Figure 5.1 shows prediction samples, where the ground truth boxes are in light blue and the predictions are in red. Figure 5.2 shows prediction samples with different IoUs where it is possible to find out how the IoU and the prediction boxes are related.



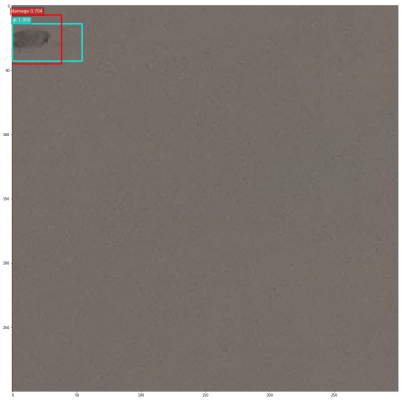
(a) Confidence 0.5 **(b)** Confidence 0.3

Figure 5.1: Prediction samples

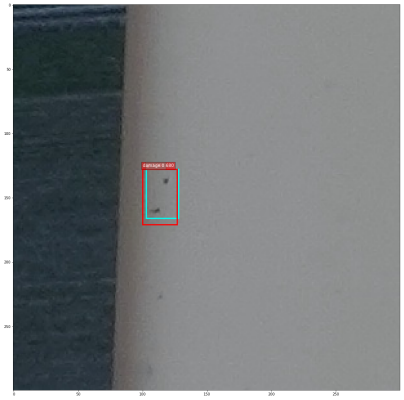
Because more experiments have been performed than those presented in this section, graphs of all the results can be found in Appendix C on page 56.



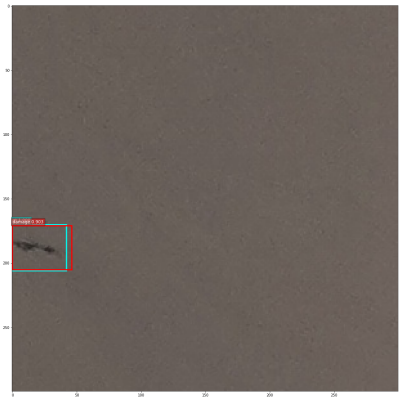
(a) Confidence 0.5 and IoU 0.318



(b) Confidence 0.5 and IoU 0.567



(c) Confidence 0.5 and IoU 0.765



(d) Confidence 0.5 and IoU 0.856

Figure 5.2: Prediction samples with different IoU

5.1.6. Results of Predicting All Types as *damage* Label on the Test Set

We evaluated the performance of the best obtained model of Subsection 5.1.5 on the test set. This test set is composed of inspections we got access after the overall revision process finished. It means that the new data set is supposed to contains only inspections reviewed by both teams, as explained in Subsection 3.1.1.

Because all previously accessed data was split only in train and dev due the number of images and annotations, we decided to split the new dataset into six parts, one for testing and the rest five for training using 5-fold cross-validation, ensuring that the inspections of *ds-01-val-01* are contained in one fold. Table 5.19 shows the obtained results of the best model on the test set of ds-11-c300-v01 dataset.

5.2 Image Classification

When the damage is located, it is possible to take a crop of them and use a dedicated neural network to predict only the type of that damage. In some cases, it also makes sense to group the damage types, for example, when there are only a few samples per

Table 5.19: Results of predicting all types as damage label on the test set

network	train set	test set	IoU thresh	mAP
resnet34_v1_02	ds-01-val-01-c300-v01	ds-11-c300-v01-test	0.5	0.542
resnet34_v1_02	ds-01-val-01-c300-v01	ds-11-c300-v01-test	0.4	0.672
resnet34_v1_02	ds-01-val-01-c300-v01	ds-11-c300-v01-test	0.3	0.751
resnet34_v1_02	ds-11-c300-v01-train-0	ds-11-c300-v01-test	0.5	0.615
resnet34_v1_02	ds-11-c300-v01-train-0	ds-11-c300-v01-test	0.4	0.739
resnet34_v1_02	ds-11-c300-v01-train-0	ds-11-c300-v01-test	0.3	0.805

damage type or when they look similar. To do this, each damage is cropped from the dataset and then resized to the input size of the neural network. The five damage types with more samples are used to train the models presented in this section. Therefore, in this section, the results of trained models for image classification to predict the damage type are presented.

First, in Subsection 5.2.1, experiments using pre-trained networks as feature extractor are shown. Then, in Subsection 5.2.2, the results of finetuning multiple architectures are shown.

5.2.1. Feature Extraction

The models were trained with a feature extraction strategy consisting of one step. The newly added layers were set as trainable, and the remaining layers were frozen. This training step was carried out for a maximum of 50 epochs. In this section, the results of resnet34 are presented because it was the best trained model.

It is important first to find good learning rates for training. In Table 5.20, it can be seen that learning rates between 0.1 and 0.001 should obtain higher accuracy. Table 5.21 shows that it is better to use dropout. In Table 5.22 can be observed that the highest accuracy was obtained when over-sampling was not used. Finally, Table 5.23 shows that the highest accuracy was obtained using mixup.

Table 5.20: Learning Rate for 5c

lr	acc
0.1	0.726
0.001	0.701
0.01	0.696
0.0001	0.686

Table 5.21: Dropout for 5c

model	ps	acc	std	n
resnet34	0.300	0.752	0.000	1
resnet34	0.500	0.739	0.000	1
resnet34	0.000	0.732	0.000	1

Because more experiments have been performed than those presented in this section, graphs of all the results can be found in Appendix D on page 57.

Table 5.22: over-sampling Data Augmentation for 5c

model	os	acc	n
resnet34	no	0.732	1
resnet34	in callbacks	0.724	1

Table 5.23: mixup Data Augmentation for 5c

model	mixup	acc	n
resnet34	True	0.764	1
resnet34	False	0.739	1

Summary

Table 5.24 presents a summary of the results of trained models with resnet34, lr=0.01.

Table 5.24: Summary of predicting using feature extraction

	validation_map
resnet34	0.696
+ fully connected layer of 512	0.732
+ dropout 0.500	0.739
+ dropout 0.300	0.752
+ os and no dropout	0.724
+ no os, mixup and dropout 0.5	0.764

5.2.2. Fine Tune

The models were also trained with a finetune strategy consisting of two steps. First, the newly added layers were set as trainable, and the remaining layers were frozen. This training step was carried out for a maximum of 5 epochs. Then, a finetune step was performed where all layers were set as trainable for a maximum of 40 epochs. A grid of hyperparameters is shown in Table 5.25 which was defined based on the findings of Subsection 5.2.1.

In Table 5.26, it can be seen that the highest accuracy was obtained using a pre-trained ResNet50, and image transformation (tfms) as data augmentation without using dropout or mixup. It could also be seen that the majority of the best results were obtained with resnet50, and only one model different than resnet, the VGG11, appear in the top 20. Also, almost all model were not using over-sampling. The best results were obtained with a learning rate for the finetune step of 1e-4 or 1e-5. Learning rate of 0.01 obtained the best results when dropout was not in use.

Table 5.25: Finetune hyperparameter grid

Parameter	Values
ps	0.5, 0.3, 0.0
mixup_enable	True, False
oversample	in callbacks, no
base_arch_name	resnet10, resnet12, resnet18, resnet34, resnet50, vgg11_bn, vgg16_bn, vgg19_bn
lr_fr	0.1, 0.01, 0.001
learn_fit_1	10, 5, 3
lr_un	0.001, 0.0001, 1e-05
learn_fit_2	40
np_random_seed	42
bn_final	True
pretrained	True
wd	0.01
tfms_enable	True
lin_ftrs	512
loss_func	FocalLoss

Table 5.26: Top 20 finetuned models for Image Classification

model	lr	lr2	ps	mixup	os	tfms	acc
resnet50	0.01	1e-05	0.0	False	no	True	0.773
resnet50	0.001	0.0001	0.0	True	in callbacks	True	0.769
resnet50	0.001	1e-05	0.3	False	no	True	0.769
resnet50	0.01	0.0001	0.0	False	no	True	0.766
resnet50	0.01	0.0001	0.0	False	no	True	0.766
resnet34	0.01	0.0	0.5	True	no	True	0.764
resnet50	0.01	1e-05	0.3	True	no	True	0.762
resnet50	0.001	0.0001	0.0	False	no	True	0.762
resnet18	0.001	1e-05	0.5	False	no	True	0.762
resnet50	0.001	0.0001	0.5	True	no	True	0.762
resnet18	0.001	1e-05	0.5	True	no	True	0.760
resnet50	0.01	1e-05	0.5	True	no	True	0.760
resnet34	0.01	0.0001	0.3	True	no	True	0.760
resnet18	0.001	1e-05	0.3	True	no	True	0.760
resnet34	0.01	0.0001	0.0	True	no	True	0.760
resnet50	0.001	0.0001	0.0	False	no	True	0.760
resnet34	0.001	0.0001	0.0	True	no	True	0.760
resnet34	0.01	1e-05	0.3	False	no	True	0.758
resnet18	0.01	1e-05	0.5	True	no	True	0.758
vgg11_bn	0.01	1e-05	0.3	False	no	True	0.758

CHAPTER 6
Conclusions and Future Work

In this thesis, algorithms of deep learning networks have been evaluated to support the task of detecting and then classifying the damages in photographs taken by drones as part of wind turbine blade inspections. To do that, we first did a literature review of deep networks for image classification and object detection.

Some damages were too small compared to the full-size images. Therefore we extracted crop of the full-size images to build datasets for image classification and object detection. With that datasets, we managed to train models to find damages of different sizes and shapes on high-resolution images.

After evaluating multiple use cases with various model configurations, and comparing their results, we were able to improve the performance obtained by our first experiments gradually and to formulate a solution proposal for a real industry problem with real data. We proposed to use first an object detection model to find where the damages are located and to use an image classification model to classify each damage by its type.

Our main conclusions related to the studied use cases, and the available datasets are:

- It is better to group damages to find out where the damage is located and then classify its type.
- It is better to take small crops of the full-size images when the damages are small.
- It is better to use higher hard negative mining ratio.
- It is better to generate smaller samples using the zoom-in data augmentation strategy.
- It possible to increase the mAP using smaller IoU thresholds.
- It is useful to use the original model publication and its code repository as a reference, and thinking about the other available resources on the internet as complimentary material, even if big companies sponsor them. Following that idea, we were able to improve the results of an existing implementation following the original model publications.

In this thesis, we trained models to evaluate the performance of different model configurations for different use cases. With that in mind, a possible extension to our work is to choose a specific use case and focus on that use case to do more exhaustive evaluations. Also, we used a small part of all images and damage annotations available. Therefore, a possible next step is to train models using a more significant portion of that data. A more deep analysis could also be made to understand the performance of the models by grouping damages by its identifiable characteristics such as shape and size.

Bibliography

[1] M. Acharya, K. Jariwala, and C. Kanan. Vqd: Visual query detection in natural scenes. *arXiv preprint arXiv:1904.02794*, 2019.

[2] E. W. E. Association et al. Wind energy - the facts, 2009. URL https://www.wind-energy-the-facts.org/operation-and-maintenance-costs-of-wind-generated-power.html.

[3] Y. Bengio, Y. LeCun, C. Nohl, and C. Burges. Lerec: A nn/hmm hybrid for on-line handwriting recognition. *Neural computation*, 7(6):1289–1303, 1995.

[4] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[5] M. Bhattacharya, S. R. Paramati, I. Ozturk, and S. Bhattacharya. The effect of renewable energy consumption on economic growth: Evidence from top 38 countries. *Applied Energy*, 162:733 – 741, 2016. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2015.10.104. URL http://www.sciencedirect.com/science/article/pii/S0306261915013318.

[6] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[7] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995. ISSN 1573-0565. doi: 10.1007/BF00994018. URL https://doi.org/10.1007/BF00994018.

[8] W. Dai, Y. Chen, G.-R. Xue, Q. Yang, and Y. Yu. Translated learning: Transfer learning across different feature spaces. In *Advances in neural information processing systems*, pages 353–360, 2009.

[9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005. doi: 10.1109/CVPR.2005.177.

[10] L. Deng and D. Yu. Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014. ISSN 1932-8346. doi: 10.1561/2000000039. URL http://dx.doi.org/10.1561/2000000039.

[11] D. Denhof, B. Staar, M. Lütjen, and M. Freitag. Automatic optical surface inspection of wind turbine rotor blades using convolutional neural networks. *Procedia CIRP*, 81:1166 – 1170, 2019. ISSN 2212-8271. doi: https://doi.org/10.1016/j.procir.2019.03.286. URL http://www.sciencedirect.com/science/article/pii/S2212827119305918. 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019.

[12] E. R. Dougherty and U. Braga-Neto. Epistemology of computational biology: mathematical models and experimental prediction as the basis of their validity. *Journal of Biological Systems*, 14(01):65–90, 2006.

[13] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[14] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997. ISSN 0022-0000. doi: https://doi.org/10.1006/jcss.1997.1504. URL http://www.sciencedirect.com/science/article/pii/S002200009791504X.

[15] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980. ISSN 1432-0770. doi: 10.1007/BF00344251. URL https://doi.org/10.1007/BF00344251.

[16] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL http://arxiv.org/abs/1504.08083.

[17] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL http://arxiv.org/abs/1311.2524.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[19] J. Guo, H. He, T. He, L. Lausen, M. Li, H. Lin, X. Shi, C. Wang, J. Xie, S. Zha, A. Zhang, H. Zhang, Z. Zhang, Z. Zhang, and S. Zheng. Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. *CoRR*, abs/1907.04433, 2019. URL http://arxiv.org/abs/1907.04433.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[21] D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.

[22] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. doi: 10.1162/neco.2006.18.7.1527. URL https://doi.org/10.1162/neco.2006.18.7.1527. PMID: 16764513.

[23] J. Howard et al. fastai. https://github.com/fastai/fastai, 2018.

[24] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *CoRR*, abs/1901.06032, 2019. URL http://arxiv.org/abs/1901.06032.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[27] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen. Deep learning for generic object detection: A survey. *arXiv preprint arXiv:1809.02165*, 2018.

[28] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. URL http://arxiv.org/abs/1512.02325.

[29] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000029664.99615.94. URL https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[30] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. doi: 10.1007/BF02478259. URL https://doi.org/10.1007/BF02478259.

[31] M. A. Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

[32] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[33] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[34] C. Poultney, S. Chopra, Y. L. Cun, et al. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2007.

[35] R. Raina, A. Y. Ng, and D. Koller. Constructing informative priors using transfer learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 713–720. ACM, 2006.

[36] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL http://arxiv.org/abs/1506.02640.

[37] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL http://arxiv.org/abs/1506.01497.

[38] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[39] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL http://arxiv.org/abs/1609.04747.

[40] D. Rumelhart, G. Hinton, and R. Williams. Learning representations byback-propagating errors. *Nature*, pages 533–536, 1986.

[41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. URL http://arxiv.org/abs/1409.0575.

[42] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[43] A. Shihavuddin, X. Chen, V. Fedorov, A. Nymark Christensen, N. Andre Brogaard Riis, K. Branner, A. Bjorholm Dahl, and R. Reinhold Paulsen. Wind turbine surface damage detection by deep learning aided drone inspection analysis. *Energies*, 12(4):676, 2019.

[44] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[46] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL http://arxiv.org/abs/1505.00387.

[47] A. Stetco, F. Dinmohammadi, X. Zhao, V. Robu, D. Flynn, M. Barnes, J. Keane, and G. Nenadic. Machine learning methods for wind turbine condition monitoring: A review. *Renewable Energy*, 133:620 – 635, 2019. ISSN 0960-1481. doi: https://doi.org/10.1016/j.renene.2018.10.047. URL http://www.sciencedirect.com/science/article/pii/S096014811831231X.

[48] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

[49] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning. *CoRR*, abs/1808.01974, 2018. URL http://arxiv.org/abs/1808.01974.

[50] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[51] S. Verbruggen. Onshore wind power operations and maintenance to 2018, 2018. URL http://www.endsintelligence.com/report/onshore-wind-power-operations-and-maintenance-to-2018/.

[52] P. Viola, M. Jones, et al. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1:511–518, 2001.

[53] L. Wang and Z. Zhang. Automatic detection of wind turbine blade surface cracks based on uav-taken images. *IEEE Transactions on Industrial Electronics*, 64(9):7293–7303, Sep. 2017. ISSN 0278-0046. doi: 10.1109/TIE.2017.2682037.

[54] L. Wang, Z. Zhang, and X. Luo. A two-stage data-driven approach for image-based wind turbine blade crack inspections. *IEEE/ASME Transactions on Mechatronics*, 24 (3):1271–1281, June 2019. ISSN 1083-4435. doi: 10.1109/TMECH.2019.2908233.

[55] G. Xia, X. Bai, J. Ding, Z. Zhu, S. J. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang. DOTA: A large-scale dataset for object detection in aerial images. *CoRR*, abs/1711.10398, 2017. URL http://arxiv.org/abs/1711.10398.

[56] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017. URL http://arxiv.org/abs/1710.09412.

[57] Z. Zhao, P. Zheng, S. Xu, and X. Wu. Object detection with deep learning: A review. *CoRR*, abs/1807.05511, 2018. URL http://arxiv.org/abs/1807.05511.

[58] R. M. Zur, Y. Jiang, L. L. Pesce, and K. Drukker. Noise injection for training artificial neural networks: A comparison with weight decay and early stopping. *Medical Physics*, 36(10):4810–4818, 2009. doi: 10.1118/1.3213517. URL https://aapm.onlinelibrary.wiley.com/doi/abs/10.1118/1.3213517.

APPENDIX A
SSD ResNet Backbones

The Table A.1 shows a complete list of the used SSD-ResNet backbones. A more compact list is shown in Table 4.2 on page 29.

Table A.1: SSD-ResNet backbones

net	info
resnet18_v1	Last activation of stage 3 y 4 + 4 extra
resnet18_v1_02	Last activation of stage3 + 5 extra
resnet18_v1_02_gn	Last activation of stage3 + 5 extra + gaussian noise
resnet18_v1_03_02	Last activation of stage3 + 1 extra
resnet18_v1_03_02_01	Last activation of stage3 + 2 extra
resnet18_v1_03_02_01_01	Last activation of stage3 + 3 extra
resnet18_v1_03_02_01_01_01	Last activation of stage3 + 4 extra
resnet18_v1_03_02_01_numfilters01	Last activation of stage3 + 2 extra of 256
resnet18_v1_03_02_01_numfilters02	Last activation of stage3 + 2 extra of 128
resnet18_v1_03_02_numfilters01	Last activation of stage3 + 1 extra of 256
resnet18_v1_03_02_numfilters02	Last activation of stage3 + 1 extra of 128
resnet18_v1_03_02_ratios01	Last activation of stage3 + 1 extra + $[1, 2, 0.5, 3, 1.0/3]$
resnet18_v1_04_01	Last activation of stage2
resnet18_v1_04_02	Last activation of stage 3
resnet34_v1_00	Last activation of stage 3 y 4 + 4 extra
resnet34_v1_00_01	Last activation of stage 2 y 4
resnet34_v1_00_02	Last activation of stage 3 y 4
resnet34_v1_00_03	Last activation of stage 2 y 3
resnet34_v1_01	Last activation of stage 3 + 5 extra
resnet34_v1_02	Last activation of stage 3 + 5 extra
resnet34_v1_02_01	Same as resnet34_v1_04_01
resnet34_v1_02_02	Same as resnet34_v1_04_02
resnet34_v1_02_gn	Last activation of stage3 + 5 extra + gaussian noise
resnet34_v1_03	Last activation of stage2 + 5 extra
resnet34_v1_03_02	Last activation of stage3 + 1 extra
resnet34_v1_04_01	Last activation of stage2
resnet34_v1_04_02	Last activation of stage3
resnet50_v1	Last activation of stage 3 y 4 + 4 extra

APPENDIX B
Damage Type Samples

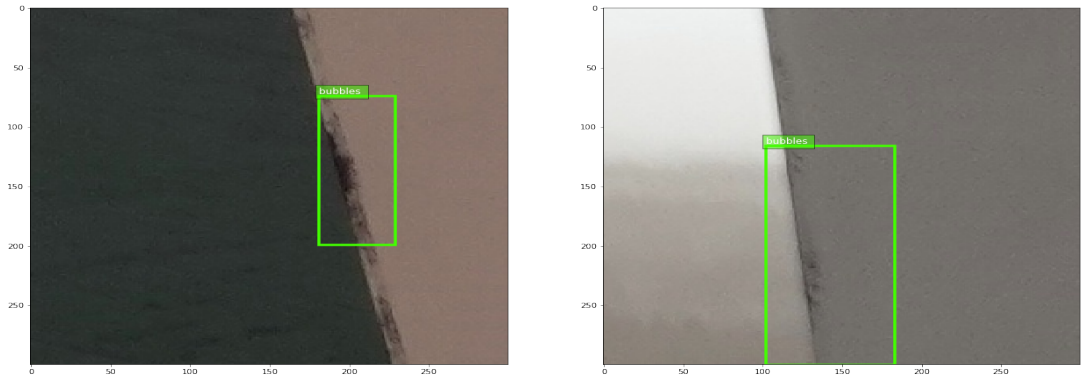


Figure B.1: Bubbles

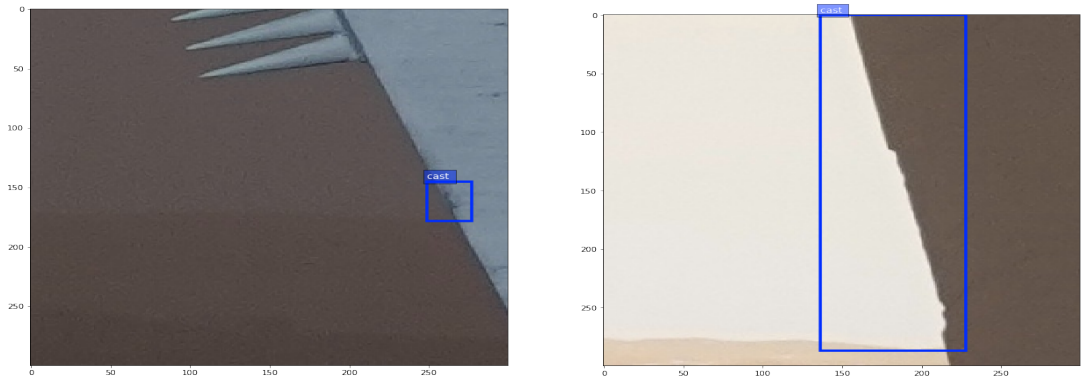


Figure B.2: Cast

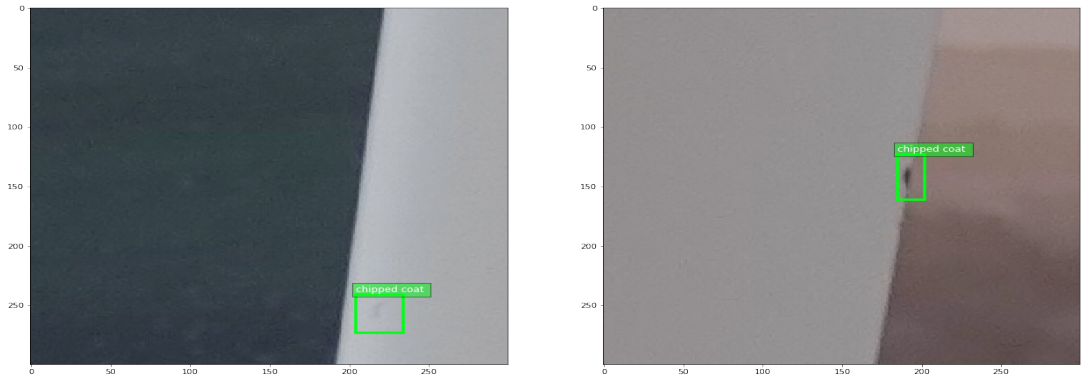


Figure B.3: Chipped coat

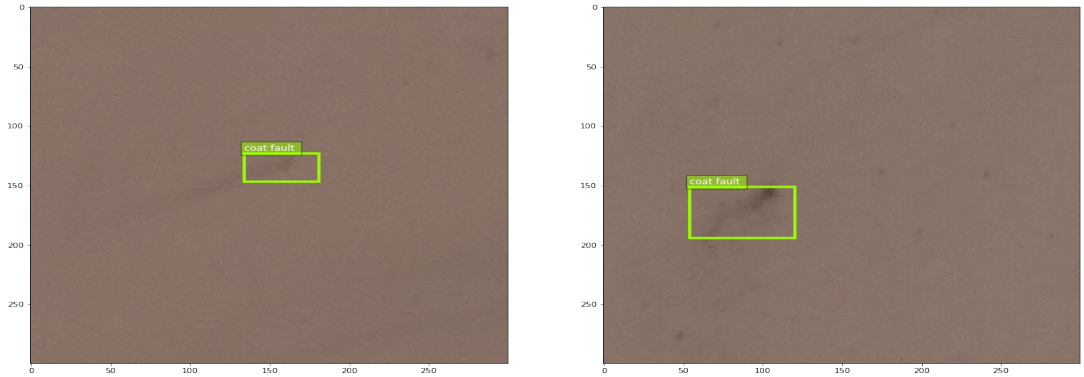


Figure B.4: Coat fault

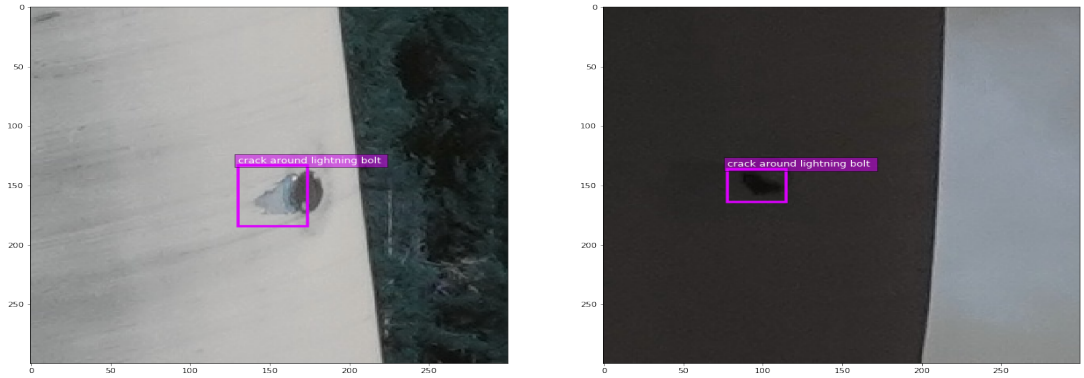


Figure B.5: Crack around lightning bolt

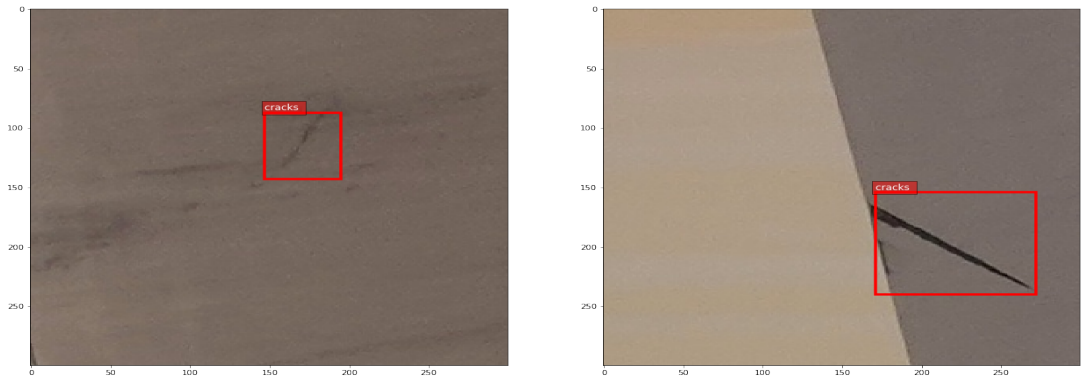


Figure B.6: Cracks

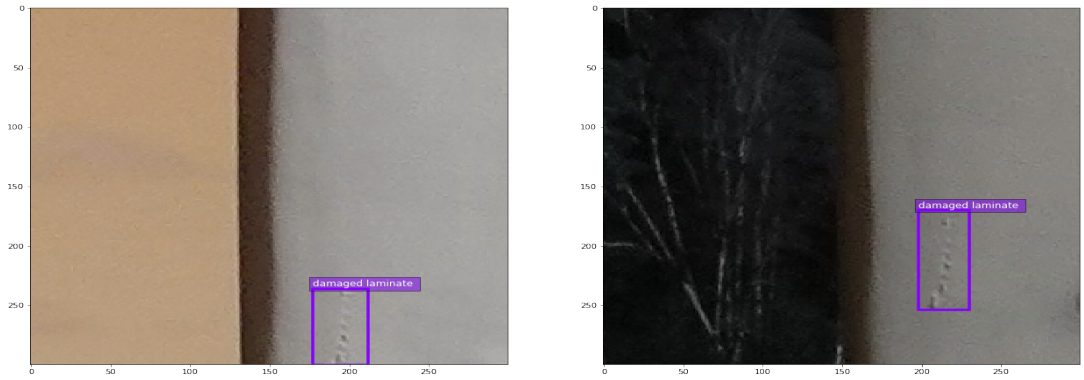


Figure B.7: Damaged laminate

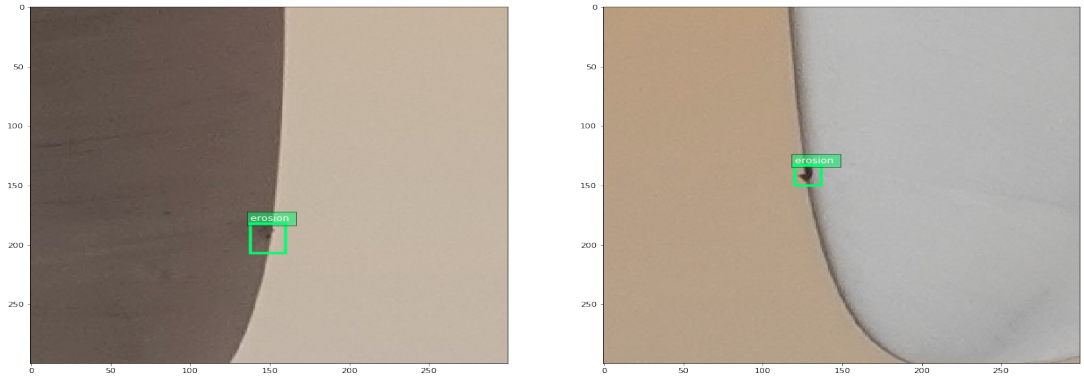


Figure B.8: Erosion

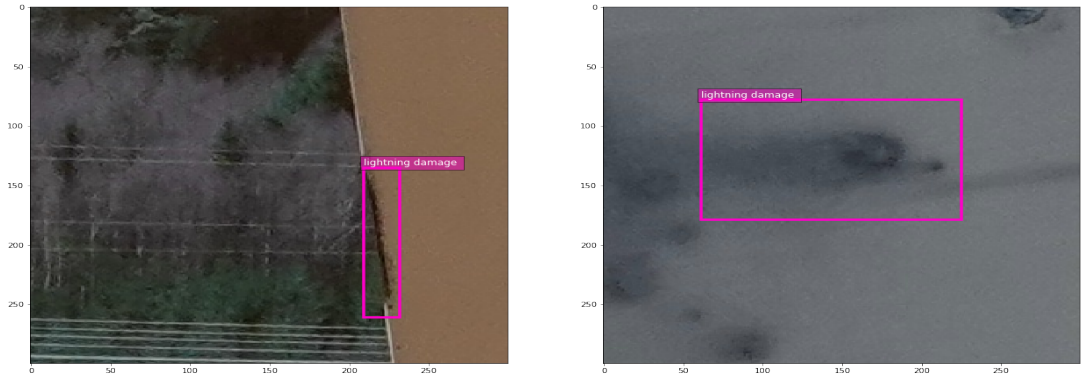


Figure B.9: Lightning damage

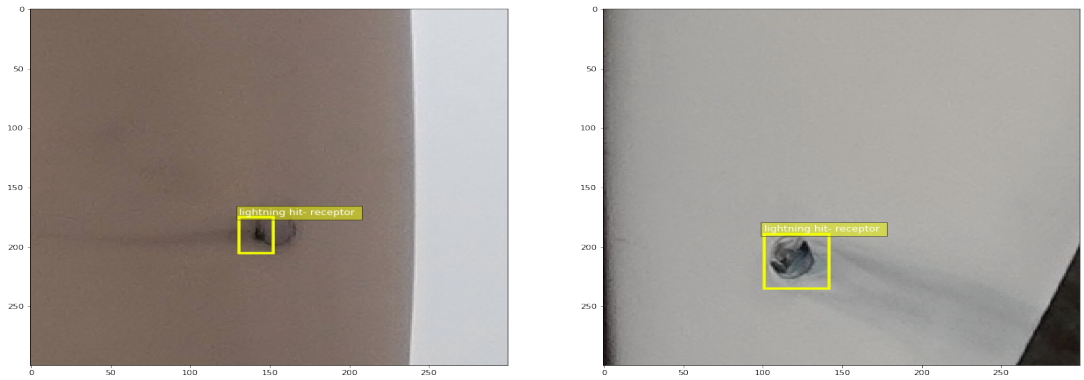


Figure B.10: Lightning hit receptor

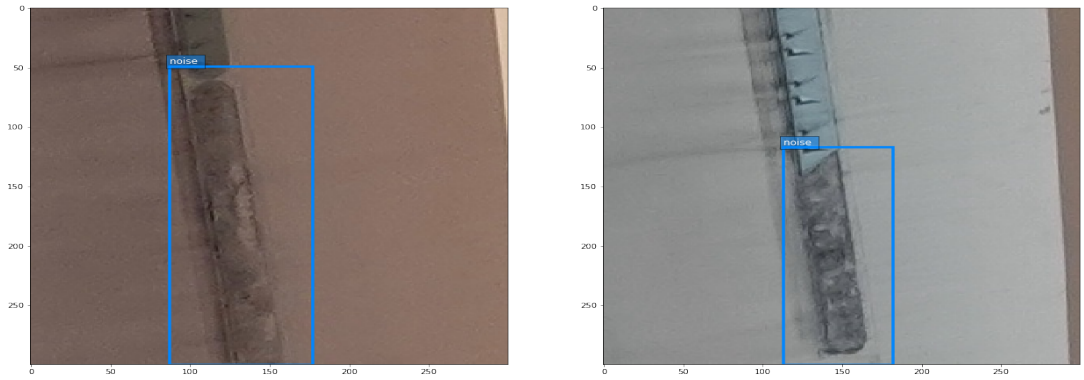


Figure B.11: Noise

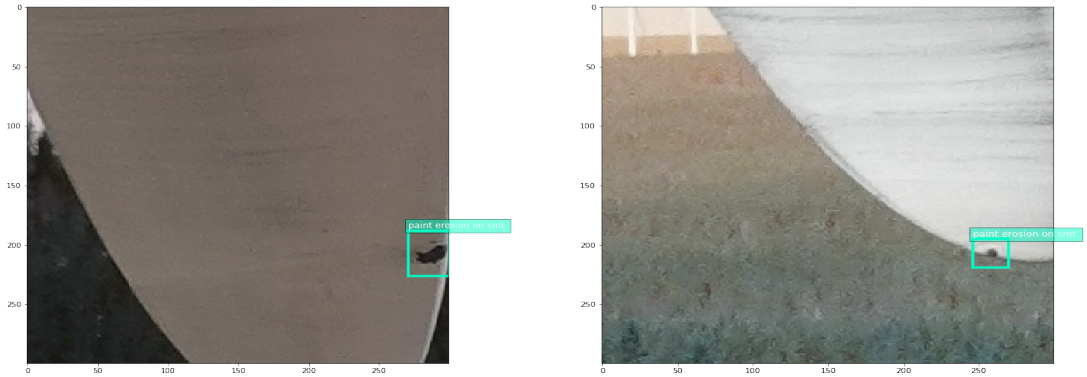


Figure B.12: Paint erosion on smt

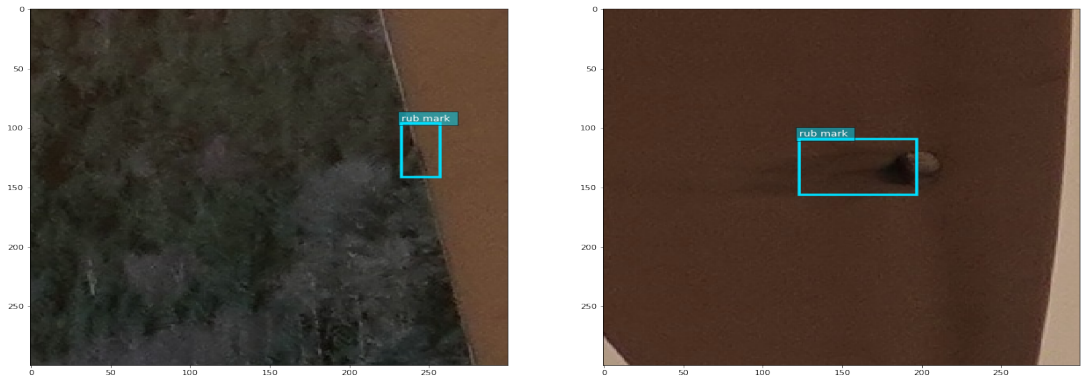


Figure B.13: Rub mark

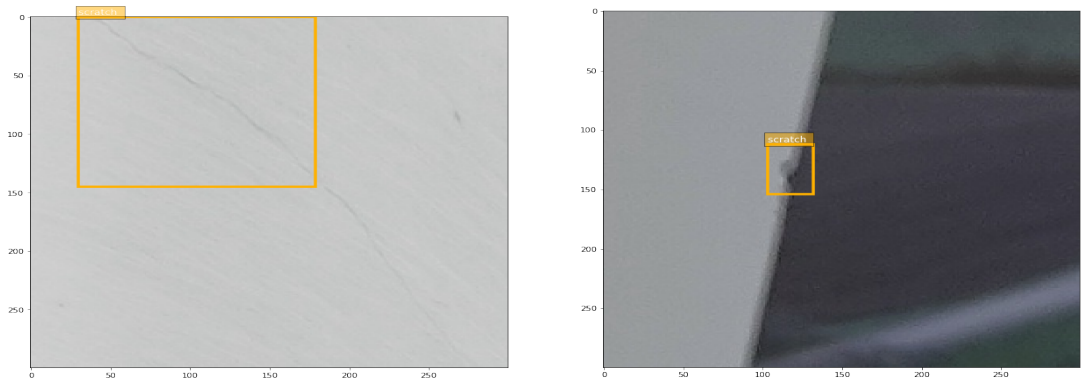


Figure B.14: Scratch

APPENDIX C
Results Graphs for Object Detection

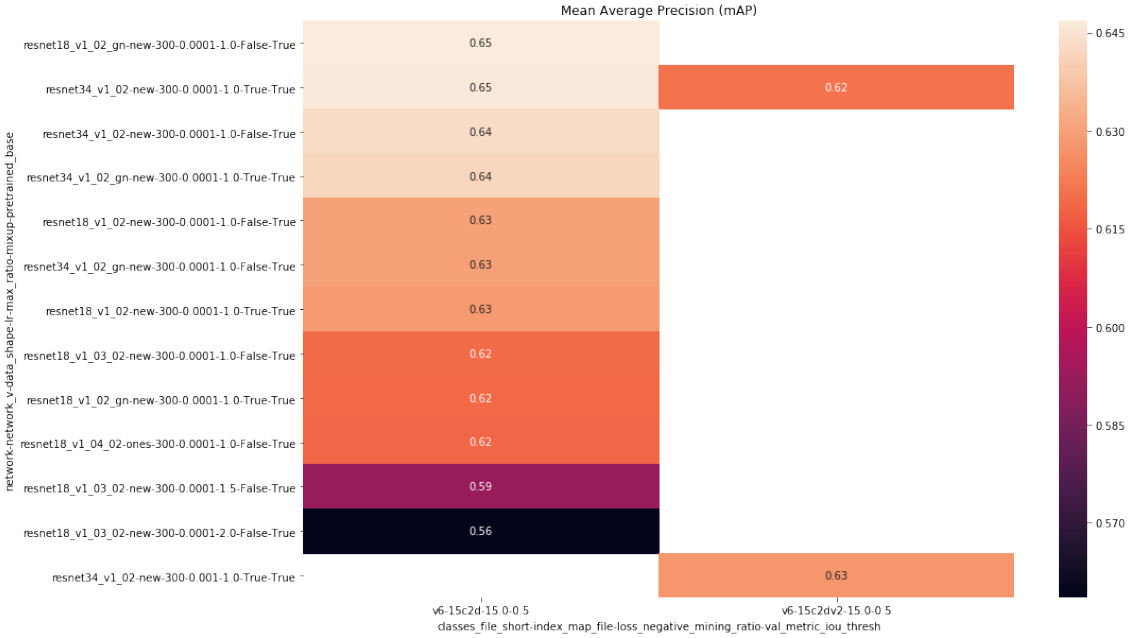


Figure C.1: Results for v6

APPENDIX D
Results Graphs for Image Classification

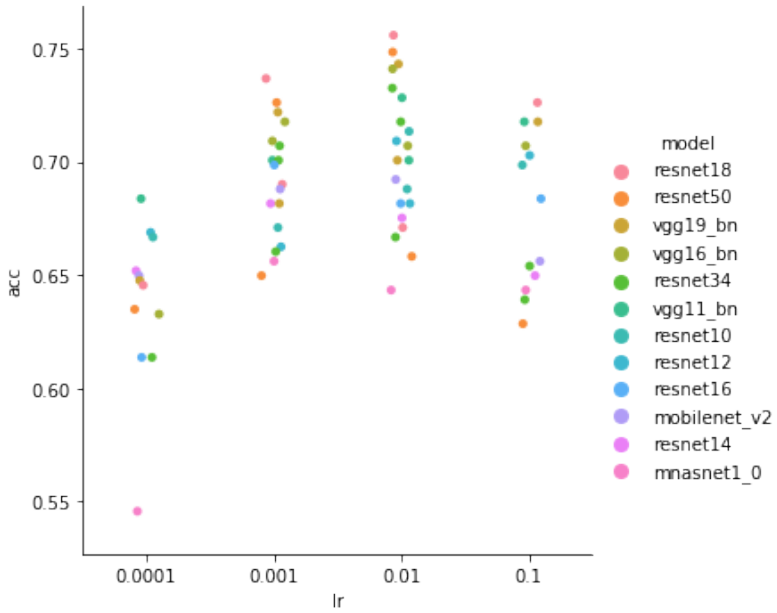


Figure D.1: Learning Rate for 5c.

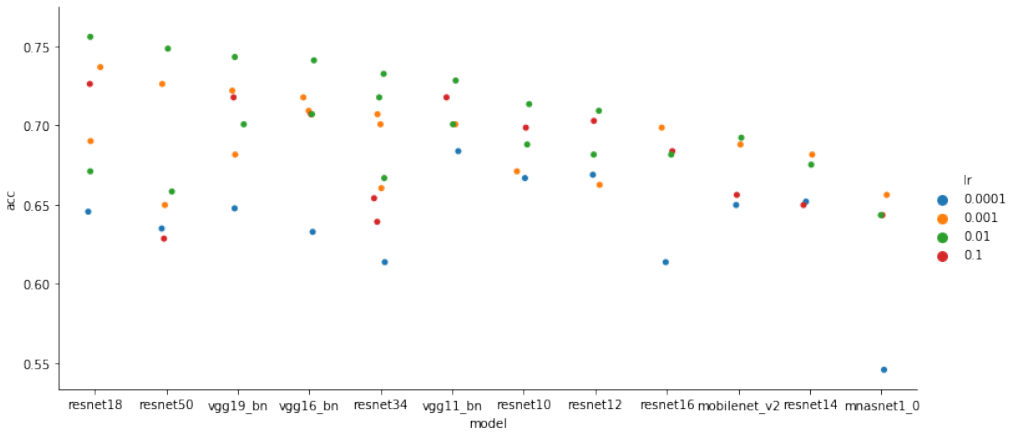


Figure D.2: Architecture for 5c.

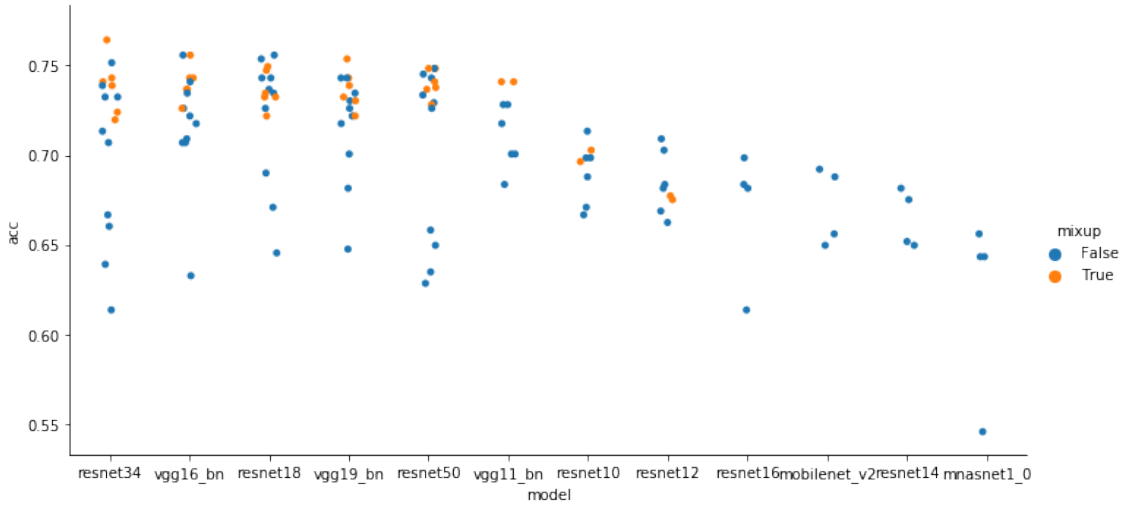


Figure D.3: mixup Data Augmentation for 5c

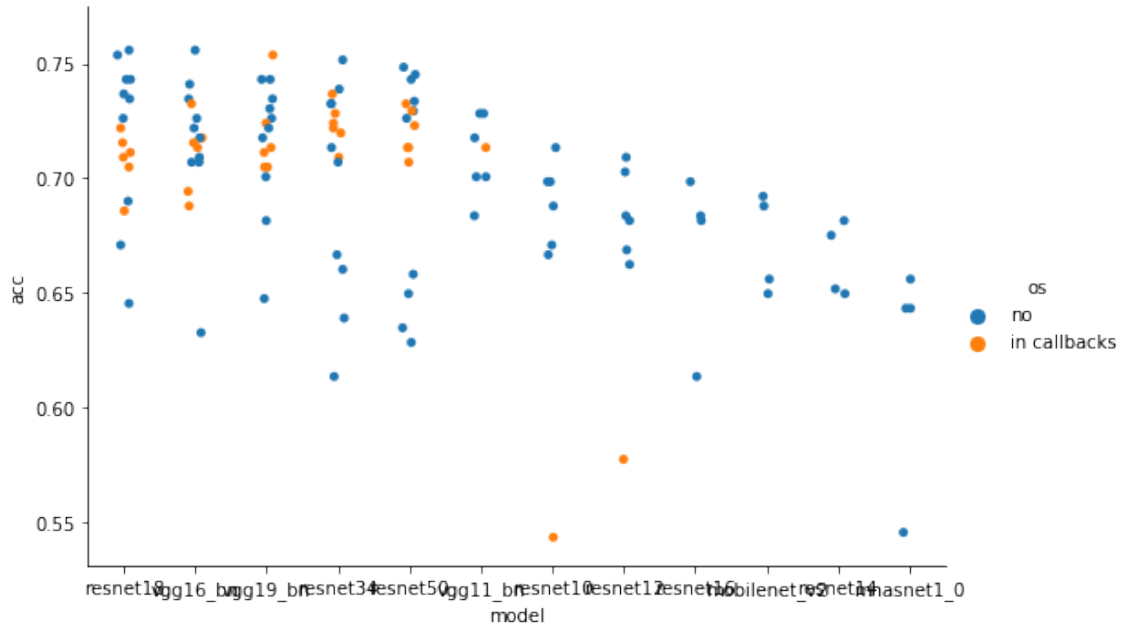


Figure D.4: over-sampling Data Augmentation for 5c

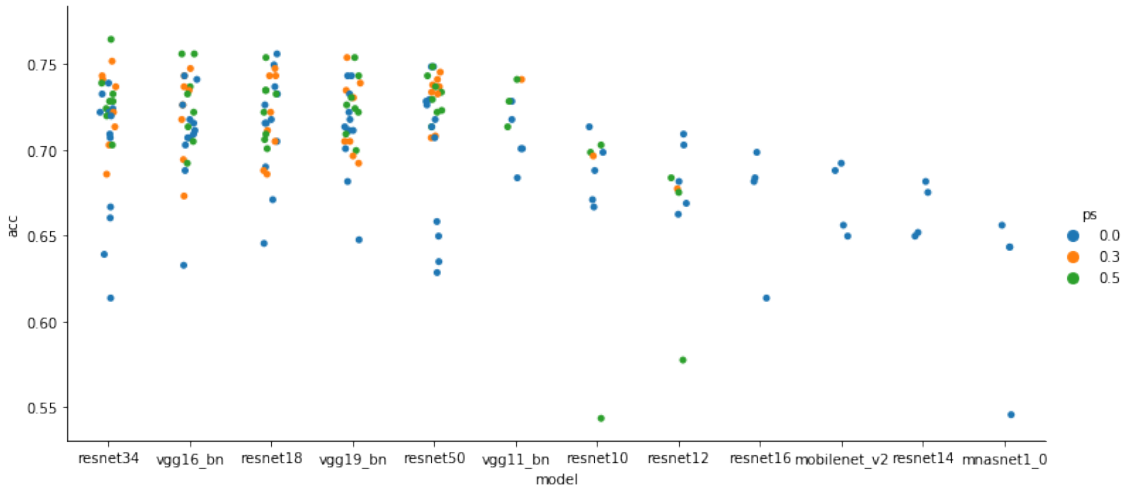


Figure D.5: Dropout for 5c

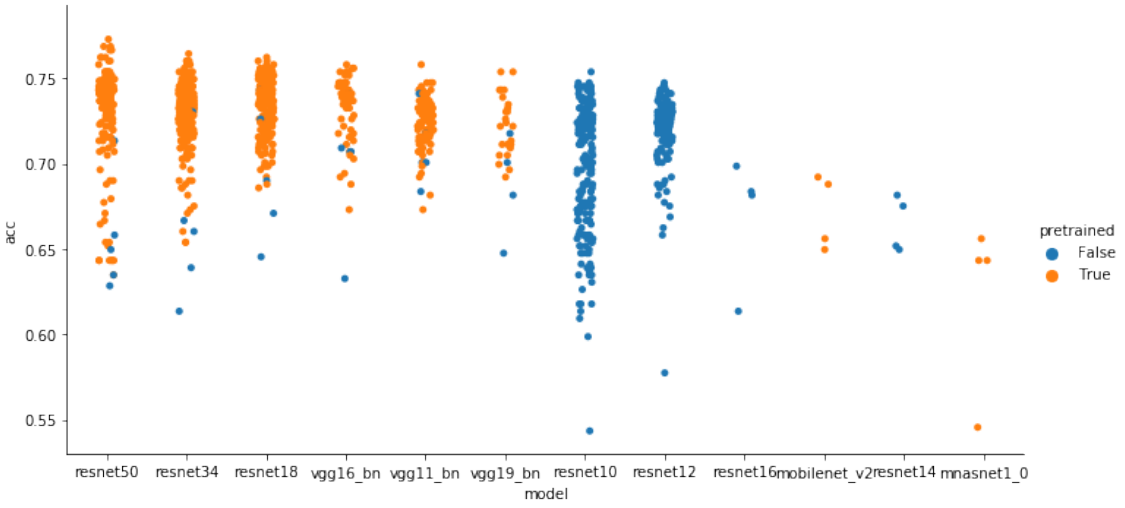


Figure D.6: Fine Tune for 5c