A Study of Active Object Detection for Composite Labelling

Connor Clarkson
868419

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Masters of Science

Department of Computer Science
Swansea University

November 20, 2020

Abstract

This project proposes an active learning framework on object detectors. We support any imagery-based dataset and any object detector from TensorFlow's Model zoom achieve. We put special focus on steel defect detection and composite labelling. Due to steel production having a continuous flow on steel needing to automatically be checked, we apply online learning elements to our investigation. We explore various object detectors in this unique setting of active-online learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Context and Motivation

For this thesis, we developed an active learning framework with a focus on object detectors. For this purpose we aim for detecting defects within steel but this framework can be used on any imagery-based dataset. Some of the biggest advances within AI have come from deep learning [1]. Since AlexNet winning the ImageNet competition [2], the main mentality and objective of many deep learning researchers and petitioners are to use more data and parameters in achieving complex tasks, which has resulted in models like OpenAI's GPT-3 with 175 billion parameters [3]. The hopeful goal of these models is to create them so large that they can be used on any task within its domain. The approach has become the de facto way in object detection, where we apply a transfer learning technique on one of these large models to a specific dataset. Allowing us to utilise the knowledge of the domain, which saves time and boosts performance. Two of the largest bottlenecks with these models - and generally within machine learning - is data labelling due to large/complex data and the need to have high-quality annotations. Other challenges such as a change in data distribution over time or even interpreting model outcomes are becoming more problematic. Within this thesis, we combine advances in online learning - where the model learns in real-time - with the advances of active learning to bring humans into the loop of AI decision-making, to reduce the need to label all the data.

1.2 Project Specification

Within this project we will be using two datasets, "Labelled Face Parts in the Wild" (LFPW) [4] and "Severstal: Steel Defect Detection" [5]. LFPW is a keypoint estimation dataset which makes it useful for our composite label experiments where we want to predict faces as well as the composite features of a face such as eyes and noses. Defects in steel often consists of composite labels that make up the final defect. However, we were unable to find an open-source steel dataset that consists of composite defects, so we choose to use the LFPW dataset for this experiment. The steel dataset is still vital for experimentation on hard to spot defects and low variance between classes of defects. The labels in these datasets are still useful to use even though we have our labelling application, as we can remove human error in our experiments and speeds up the process of experimenting. The experiments are dependent on previously completed experiments, however in general we will explore:

- Inter-class performance of composite labels
- Object detector performance on small intraclass labels
- Object detection in an active learning setting
- Influx of a class becoming more common in an online learning setting

The application specification consists of creating dataset-specific profiles. Users can select any number of images from the image viewer to then label. The labeller consists of 2 types of brushes and any number of classes. Users can create, rename, or remove labels on the fly and the configuration of the object detector is updated accordingly.

1.3 Objectives

To accomplish our project specification we aim to have computed the following objectives in stages:
Stage 1 Labelling Application:

- Dataset-specific profiler.
- An image viewer to select images for labelling.
- A labeller with multiple brushes.
- Users should be able to add, remove, rename labels on the fly.

Stage 2 Adapt application for machine learning environment:

- Generate TFRecord for each training session.
- The configuration file of any object detector should be updated to any changes made by the user.
- Object detectors should apply fine-tuning from the previous training session.

Stage 3 Experimentation:

- 1 and 2-stage object detector experiments; exploring composite labels and low intra-class labels.
- Active and online learning experiments with a focus on the performance of object detectors.

1.4 Contributions

- Active learning framework for object detectors with support for any imagery based dataset.
- Adaptable object detectors for changing or removing classes.
- Investigation in the use of object detectors in active and online learning settings.
- Investigation of composite labels and low intra-class labels such as in steel defects.

1.5 Outline

The chapters of this dissertation are outlined as follows:

Chapter 2 Literature Review:

We introduce the background literature of the project and dissertation with the introduction of neural networks and building towards deep learning. We end the chapter by exploring relevant literature in object detection.

Chapter 3 Responsibly Innovated Human Centric Design:

In this chapter we introduce responsible innovation and human centric design.

Chapter 4 Methodology:

We introduce the methodology and the software deliverable for this project. This chapter is split between our proposed labelling application and object detection.

Chapter 5 Project Management:

In this chapter we discuss the plan of the project and how it was planned.

Chapter 6 Evaluation:

We draw from our methodology and perform experiments to evaluate our deliverable .

Chapter 7 Conclusions and Future Work:

Finally we conclude the project and reflect back on the presented work and project in general.

Chapter 2

Literature Review

Contents

2.1 Introduction

Since 2015 when Girshick et al proposed the first deep learning object detector to successfully beat the state of the art metrics in PASCAL VOC dataset [6], object detection exploded in popularity in a similar way to deep learning in 2012 with competitions like ImageNet. Current object detectors rely heavily on large amounts of data and a tedious labelling process. This process becomes even more of a challenge when objects have composite elements that also need to be labelled, or inter-class variance is large. A common problem in many surface/texture detection tasks. Learning scenarios like active learning hope to address some of these challenges by only labelling samples that will have the greatest statistical inference on the object detectors, as result there should less of a need to label a lot of data and reducing the amount of data needed to train. Other challenges like when we need to train in real-time, due to deep learning following a greedy data principle. In this chapter, we present the knowledge and literature that is required for our experiments. We show the different types of learning and learning techniques used within machine learning to the development of neural networks and deep learning. Finally, we show some of the key developments in object detection and the process of how we compare such detectors.

2.2 Types of Learning

Machine learning is the purpose of gaining new knowledge from a domain feature space by learning the general patterns of that space. This experience is often in the form of historical datasets. This idea drives the concept of data-driven approaches. The three main tasks of machine learning are supervised, unsupervised and reinforcement learning. Each task differs in the type of data they input, output, and the approach in learning from the dataset. The lines between these tasks can get blurry as a result of domain-specific problems. In situations where the practitioner wants to use supervised learning but has small to no data, they could attempt to use semi/self-supervised learning. These hybrid tasks attempt to merge the main tasks in some capacity. We can also apply common learning techniques to any of the tasks, however applying these techniques are dependent on the problems being addressed. Multi-task, active, online, transfer and ensemble learning roughly fall under learning techniques.

2.2.1 Supervised Learning

One of the three pilers of machine learning, supervised learning consists of a labelled training set $\{(x_1, y_1), ..., (x_N, y_N)\}$. Where x_i is a sample(or feature vector) and y_i is the label (or class) of the sample. A sample is a vector of features that we show to a model and the result is a prediction of the label $\hat{y}_i$, generally defined as $f(x_i, \theta) = x_i\theta^T$. Where $f(\cdot)$ is our model and θ are the weights. We then define some loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i L(f_\theta(x_i), y_i)$, where we measure how close the prediction $\hat{y}_i$ was to y_i and compute the sum of all measures calculated in the training set. We learn over time by selecting a good θ that minimizes the loss $\mathcal{L}(\theta)$. To find a good θ we use one of a family of methods known as computational methods of optimization, the most common being gradient descent [7, 8]:

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t) \tag{2.1}$$

Where α_t is our gradient step value at timestamp t. The gradient step is scaler value that defines how large a jump we make towards the local minimum. Notice that we use previous θ_t to update $\theta_t + 1$, thus we must have a starting θ_0. Wolpert and Maccready prove that the best way to learn in any combinatorial optimization algorithm is to use a random θ_0 to allow for expectation, the only exception to this general rule is to have weights specially designed for a class of problems and we train for a subclass of problems(known as transfer learning within the machine learning literature and the transfer of learning in psychological literature) [9]. We continue to train until $\mathcal{L}(\theta)$ converges or we use some other early stopping criteria is used.

Within supervised learning there is two categories of problems, known as regression and classification. Regression is used to predict a continuous $\hat{y}_i$ while classification is used when we want to predict a discrete $\hat{y}_i$. The general form of regression is known as linear regression. In regression we learn to make a line of best fit given our training data, then predict $\hat{y}_i$ from our test dataset by getting the value where our test sample crosses the line on the y axis.

$$\hat{y}_i = c + mx_i \tag{2.2}$$

where c is the y-intercept and m is the slope of the line. Thus we need to learn a good c and m to get an accurate $\hat{y}_i$. A common loss function in regression to achieve this is means squared error:

$$\mathcal{L}(c, m) = \frac{1}{2N} \sum_{i=1}^{N} ((c + mx_i) - y_i)^2 \tag{2.3}$$

where N is the number of training samples. We then use gradient descent to minimise this loss function (equation 2.1) If input samples have more than one feature than we replace $(c + mx_i)$ in equation 2.3 with $h_\theta(x)$ defined as:

$$h_\theta(x_i) = \theta_0 + \theta_1 x_i^1 + ... + \theta_n x_i^N \tag{2.4}$$

where x_i is a our input sample vector of features $x_i^1, x_i^2, ..., x_i^n$, n is the number of features within the sample vector, N is the number of input samples in the dataset, θ_0 replaces c and finally $\theta_1 ... \theta_n$ replaces m.

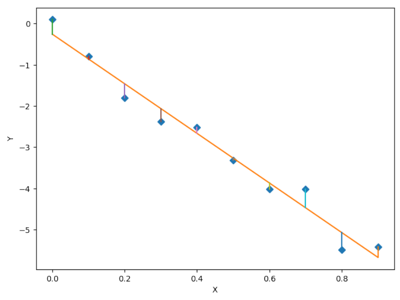


Figure 2.1: Illustration of linear regression where the orange line is the line of best fit, and the blue line between the line of best fit and the samples is the error of that specific sample.

Logistic regression applies the principles of regression in a classification setting. In this setting our $\hat{y}_i$ is a binary value which is the numerical mapping of our label. We fit a sigmoid line to our data:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{2.5}$$

where x is a input sample vector and θ is our current weights. We define a new loss function known as cross-entropy loss for our binary labels:

$$\mathcal{L}(\theta) = -\frac{1}{N} \left[\sum_{i=1}^{N} y_i \, log \, h_\theta(x_i) + \sum_{i=1}^{N} (1 - y_i) log(1 - h_\theta(x_i)) \right] \tag{2.6}$$

where the term $\sum_{i=1}^{N} y_i \, log \, h_\theta(x_i)$ is loss if $y_i = 1$ while the term $\sum_{i=1}^{N}(1 - y_i) log(1 - h_\theta(x_i))$ is the loss if $y_i = 0$ and N is the number of samples in the training dataset. The loss is being computed by taking negative log of all probabilities of both $y = 1$ and $y = 0$. The result of the loss function increases as the predictions $\hat{y}$ moves away from their true labels y. Finally we use gradient descent (equation 2.1) to minimise our new loss function.

2.2.2 Unsupervised Learning

In the context of unsupervised learning we do not provide any labels for each of our samples, thus our training dataset is defined as $\{x_1, ..., x_N\}$, where N is the number of samples in the set. This setting often uses the priori probability distribution of the dataset for the given unsupervised learning task. These tasks could include dimensionality reduction, cluster analysis or association analysis. Within cluster analysis our aim is to group the data into clusters based on the natural structure of the dataset given, kmeans is one such appouch. K-means gets its name from k number of centroids and the mean position of a centroid given all data samples within the associated cluster.

Algorithm 1: k-Means [10]

```
Result: Clusters of dataset
Step 1: Randomly initialize cluster centroids $\mu_1, \mu_2, ..., \mu_k$ in facture space
Step 2: while not converging do
    Step 2.a:
    foreach sample i do
        $c_i = argmin_j \|x_i - m_j\|^2$
    end
    Step 2.b:
    foreach centroid j do
        $m_j = \frac{\sum_{i=1}^{m} 1[c_i=j]x_i}{\sum_{i=1}^{m} 1[c_i=j]}$
    end
end
```

2.a defines the nearest centroid to each data sample x_i. While in 2.b, μ_j is the current guesses for the positions of the centroids at timestamp j. A good clustering of the data is dependent on the initial positions of the centroids. To check if clusters are good clusters at we can use the distortion function:

$$\mathcal{J}(c, \mu) = \sum_{i=i}^{N} \|x_i - \mu_{c_i}\|^2 \tag{2.7}$$

Where $\mathcal{J}$ computes the sum of squared distances for each data sample x_i and assigned centroid μ_{c_i}.

2.3 Learning Techniques

2.3.1 Transfer Learning

The aim of transfer learning is to use previously gained knowledge from one domain and apply it to a new but related domain. In the hope that the model learns the new domain quicker and with greater accuracy than simply training a model from scratch. In practice we reuse the generic layers (layers close to the input of the model) and create new task specific layers (the layers closer to the output of the model). During training we freeze the weights of the generaic layers allowing the task specific layers to learn. Generally transfer learning is defined as:

$$\mathcal{D} = \{\mathcal{X}, P(X)\}, \mathcal{T} = \{\mathcal{Y}, f(\cdot)\} \tag{2.8}$$

Where domain $\mathcal{D}$ holds our feature space $\mathcal{X}$ and a probabilistic distribution $P(X)$ where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. Task $\mathcal{T}$ holds our label space $\mathcal{Y}$ and some model function $f(\cdot)$. The task learns from data consisting of pairs $\{x_i, y_i\}$ from X and $\mathcal{Y}$ respectively, resulting in our function f predicting the corresponding label $\hat{y}$ from $f(x)$. Now that we have Domain $\mathcal{D}$ and our learning task $\mathcal{T}$, and we have a new target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$. We use knoweldge from D and T to help $f_T(\cdot)$ learn in $\mathcal{D}_T$.

Fine Tuning is a sub-technique of transfer learning in which we unfreeze some (or all) of the early layers in our model. During training, we use a smaller learning rate than normal since we are assuming the pre-trained weights are already good when compared to randomly initialized weights. If our learning rate is too large than the model can collapse and will be unable to learn.

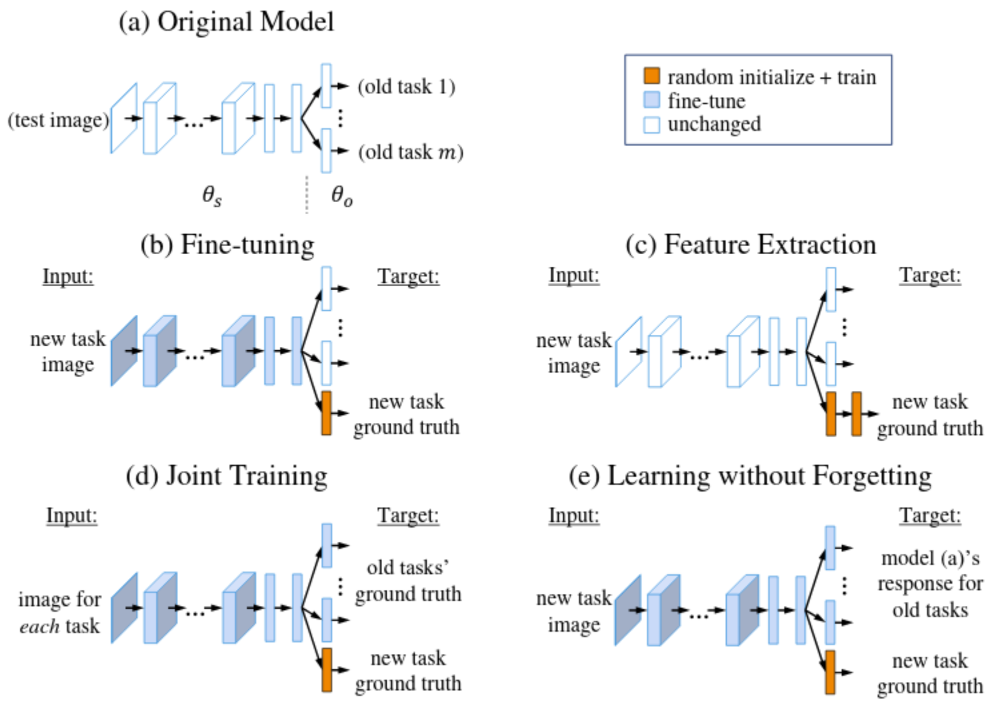


Figure 2.2: Fine Tuning is one of many sub-technique of transfer learning. This Illustration shows some of the other transfer learning methods [11]

The only criteria to using transfer learning and its variants are to ensure that the dataset for the specific task is simpler to the dataset that was used on the pre-trained model. The general rule for fine-tuning and transfer learning is as follows:

	Dataset is similar to pre-trained dataset	Dataset is not similar to pre-trained dataset
Large amounts of data	Fine-tune on all layers	Train model from scratch
Small amounts of data	Freeze all layers apart from last few layers and train, as described in standard transfer learning	Train on a smaller model from scratch and use pre-processing methods like data augmentation and post-processing methods like test-time augmentation

Table 2.1: Criteria for when we should use transfer learning, fine-tune or neither [12]

2.3.2 Active Learning

The active learning hypothesis states that if the learning algorithm is allowed to choose the data or be given more important data from a sample pool, then it will perform better with less training. One of the properties of this hypothesis is that we would require fewer data and -

more importantly - less labelled data. This is juxtaposed with traditional supervised learning, in which we are often training with thousands of labelled samples and long training times. This is becoming a desirable property in current literature with many notable researchers (including Yoshua Bengio and Geoffrey Hinton - 2 out of the 3 winners of the 2018 Turing Award) stating we need to move away from using more data and labelled samples in general [13, 14]. Their reasons include model overfitting and labelling of data getting more expensive [15].

The active learning scheme applies the hypothesis by querying unlabelled samples to be labelled by an oracle, often a human annotator but could be some information source. The aim is to achieve high accuracy with little labelled data, thus minimizing the cost of labelled data. How we select data (known as a query strategy in the literature) to be queried and the types of queries (known as active learning scenarios in the literature) are dependent on the active learning problem being proposed. The 3 main active learning scenarios are (1) membership query synthesis, (2) stream-based selective sampling, and (3) pool-based sampling.

Membership query synthesis is the scenario where the model generates its samples from the feature space distribution and then queue that generated sample to the oracle [16]. The advantage of this style of learning is that the dataset can be small and is efficient for finite problem domains [17, 18]. Lang et al used this scenario with human annotators to train a neural network in classifying handwritten characters [19]. However, they found that the generated samples contained many non-recognizable symbols, generating artificial hybrid characters instead. To fix this drawback an alternative was discovered named selective sampling. The assumption in this scenario is that the unlabelled sample can be obtained inexpensively, thus allowing the model to sample from the actual distribution and evaluate the informativeness of the unlabelled sample [20, 21]. The model then decides to request the sample to be labelled or not. The scenario is often called steam-based selective sampling as we evaluate each unlabelled sample one at a time. We can also ensure that each query given to the oracle is of a high standard as it is from the real input feature space distribution. Pool-based Sampling is the most common scenario within active learning due to situations where researchers have large amounts of unlabelled samples. In this scenario the model generates queries from a pool of samples, the model selects a query greedily. This is often in the form of an informativeness measure evaluated on all samples within the pool. The most informative samples are then queried to the oracle [22].

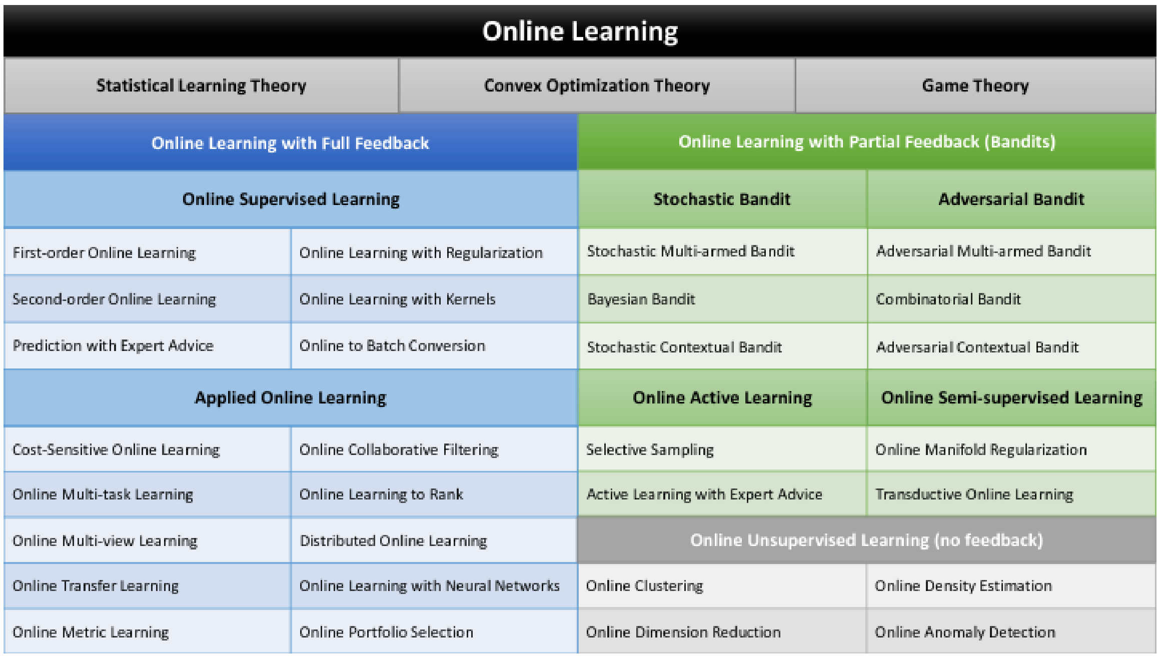
The query strategy refers to how we evaluate the informativeness of unlabelled samples. The query strategy used in the literature is uncertainty sampling [22]. The approach works by querying the samples which are the least confidence in labelling. General uncertainty sampling is defined as:

$$x^*_{LC} = \arg\max_x 1 - P_\theta(\hat{y}|x) \tag{2.9}$$

where x^*_{LC} is the least confident sample and $P_\theta(y|x)$ refers to the probability that sample x is label y using model distribution θ.

2.3.3 Online Learning

Online learning (also known as incremental or out-of-core learning) refers to a specific style of learning within machine learning where the training data becomes available in sequential order over time and we update our model at each step. Commonly in machine learning, we have access to the entire dataset to which we can train our models. Online machine learning involves a specific situation where it is infeasible to train on the entire dataset. A property of this situation is that we will have the model distribution which will change over time, thus normally these models are extremely sensitive to any change. The sensitively of the model risks a common error within this style of learning named catastrophic interference, in which a model abruptly forgets learned knowledge after getting new knowledge. However updating models after each new data sample can be a desirable property due to its scalability in real-world data analytics applications, where data is large and arriving fast [23]. In these situations traditional supervised learning would need to be re-trained once a collection of training samples had been collected, thus the model would have high time and space complexity.



Online Learning			
Statistical Learning Theory	Convex Optimization Theory		Game Theory
Online Learning with Full Feedback		**Online Learning with Partial Feedback (Bandits)**	
Online Supervised Learning		**Stochastic Bandit**	**Adversarial Bandit**
First-order Online Learning	Online Learning with Regularization	Stochastic Multi-armed Bandit	Adversarial Multi-armed Bandit
Second-order Online Learning	Online Learning with Kernels	Bayesian Bandit	Combinatorial Bandit
Prediction with Expert Advice	Online to Batch Conversion	Stochastic Contextual Bandit	Adversarial Contextual Bandit
Applied Online Learning		**Online Active Learning**	**Online Semi-supervised Learning**
Cost-Sensitive Online Learning	Online Collaborative Filtering	Selective Sampling	Online Manifold Regularization
Online Multi-task Learning	Online Learning to Rank	Active Learning with Expert Advice	Transductive Online Learning
Online Multi-view Learning	Distributed Online Learning	**Online Unsupervised Learning (no feedback)**	
Online Transfer Learning	Online Learning with Neural Networks	Online Clustering	Online Density Estimation
Online Metric Learning	Online Portfolio Selection	Online Dimension Reduction	Online Anomaly Detection

Figure 2.3: Taxonomy of Online Learning Techniques [23]

Online learning with full feedback is supervised learning tasks where the information is learnt by the model at the end of each online learning round. We further divide this topic into online supervised learning and applied online learning. Online supervised learning consists of the fundamental approaches of online learning in a supervised setting while applied online learning consists of non-traditional and often very problem-specific approaches. Online learning with partial feedback is the second main topic of online learning where the model makes a prediction on an incoming sample and is simply informed if that prediction is true or not. The is different from online supervised learning as the model is not informed what the correct label is if the prediction is wrong. For these styles of problems, the model makes the prediction based on balancing the distribution of disclosed knowledge with the exploration of unknown knowledge.

2.4 Deep Learning

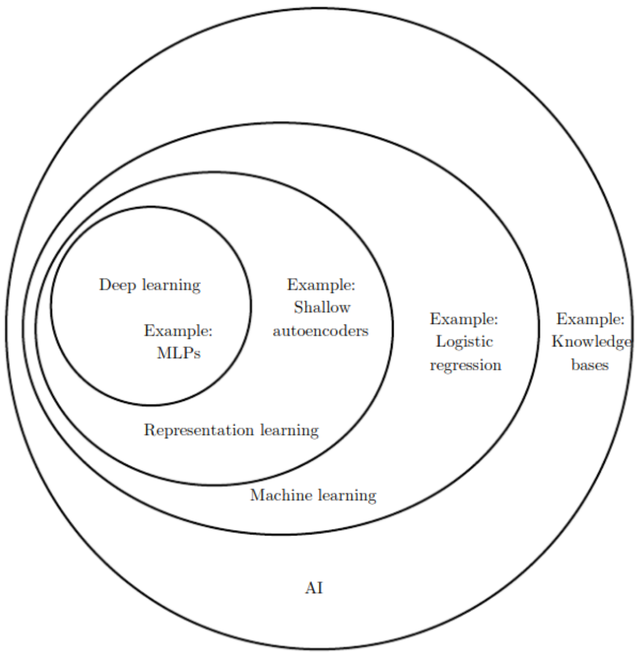


Figure 2.4: A Venn diagram illustrating the structure of AI to Deep Learning. We show that deep learning is a type of representation learning, which is a type of machine learning. Machine learning is only one branch of AI [24]

Deep Learning approaches create deeply structured neural networks (and variants) with some form of representation learning [25]. A philosophy of deep learning over the years has been to create larger structured neural networks with more data. This philosophy and approach have been successfully applied to many fields including computer vision, natural language processing, bioinformatics, and many more [26].

Representation learning is the process of learning the best way to represent the data so we can get better performance in our models. Often called feature learning as a sample X_i would have features $X_i = \{x_i^1, x_i^2, ..., x_i^N\}$, where N is the number of features and x_i^j is the j^{th} feature of X_i. A feature is a single characteristic of something being observed. For example, we can observe the sepal and petal length and width of a given species of Iris flowers [27]. A single representation is a collection of observing a single flower. Before representation learning, we would design - by hand - all the representations before attempting to get a model to learn the patterns in the features. The hypothesis in representation learning is that there are many different representations (or features) from a given sample, and it is believed that by applying optimization algorithms we can find hidden representations that greatly improve learning in the data [28]. These learnt representations might not all be task-specific (like in hand-made representations) but would still likely be useful for the general-purpose learning of the greater domain that the task is within.

2.4.1 Neural Networks

Artificial neural networks are the bases for a lot of deep learning architectures we have today. The original inspiration of neural networks comes from the information processing and distributed communication within biological neural networks and other biological systems [29]. To demonstrate neural networks we break its principles into the following sections;

- [Section 2.4.1.1] - The Perceptron and the Structure of Neural Networks: The founding building block of neural networks the perception and how we use many perceptrons to create the structure of the neural network
- [Section 2.4.1.2] - Activation Functions: Discuss if information should be passed through a particular area in a neural network or if we should discord it
- [Section 2.4.1.3] - Softmax Function: The final layer of a neural network which results in our predictions
- [Section 2.4.1.4] - Backpropagation: The widely used method for calculating derivatives in neural networks that we use to add newly learnt knowledge throughout the neural network
- [Section 2.4.1.5] - Regularization: Methods for ensuring that neural networks learn the general principles of the training data and not learn residual variation (which we call noise) [30]

2.4.1.1 The Perceptron and the Structure of Neural Networks

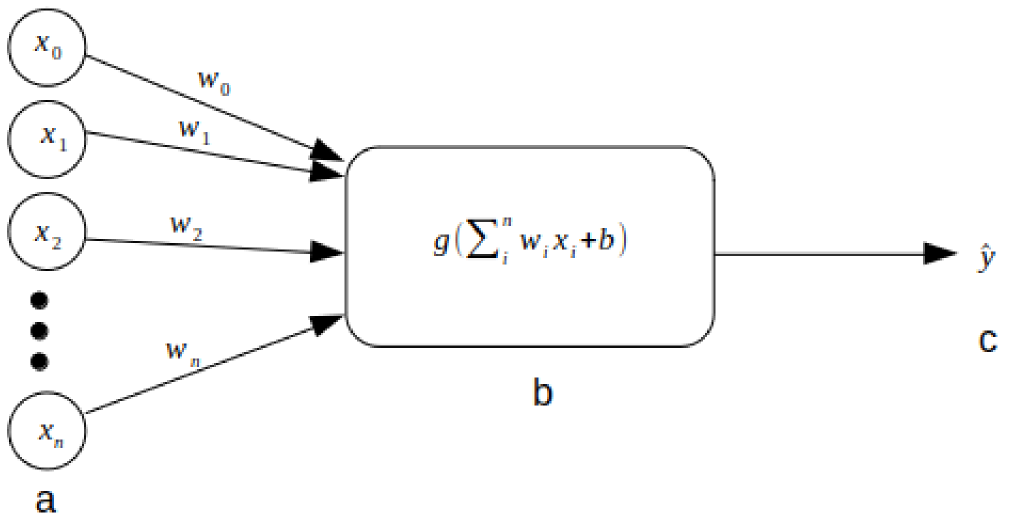


Figure 2.5: The perceptron is the weighted summation of an input vector plus some bias. We pass this value into some activation function $g(\cdot)$, this is illustrated in b. a: the input vector X where $x_0, x_1, ..., x_n$ is a set of features up to size n. c is our prediction for if our input is positive or negative [31].

A single perceptron decides if some real-valued vector X is a positive or a negative instance [31]. To determine if a vector is positive or negative we take the weighted summation of X and pass into some activation function, this is illustrated with figure 2.5. The main takeaway from the perceptron is the main contributor to determining if the input should be positive or negative is the value of our weights. Given we went some desired output or to be as close as possible to some output, we can change our weights accordingly. We often call a single perceptron a node and we can have many nodes in a layer we often call this a single layer perceptron, as to separate it from a multilayer perceptron. We can add many perceptrons in a single layer and have multiple layers within our model structure. Finally, we can formally write our neural network:

$$\begin{gathered} h^1 = g^1(W^1X + b^1) \\ h^2 = g^2(W^2h^1 + b^2) \\ . \\ . \\ . \\ \hat{y}(X) = W^N h^{N-1} + b^N \end{gathered} \tag{2.10}$$

where h^i is the output of layer i, X is our input vector, W^i our weights for layer i, b^i is our bias for layer i, and finally $g^i(\cdot)$ is our activation function for layer i.

2.4.1.2 Activation Functions

Activation functions determine if the output of a given node is positive or negative. The main category of activation functions used in neural networks is non-linear, meaning the output cannot be reproduced from a linear combination of inputs. We use non-linear activation functions to allow us to map our inputs to our outputs. If we do not have any non-linearity in our networks, then all neurons behave the same and we would not be able to perform some of the complex tasks that are common in deep learning. A common activation function is the sigmoid function [32]:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.11}$$

where x is our weighted sum of inputs. Sigmoid returns either 1 or 0 to determine if the weighted sum (which is a real-valued scalar) to be positive or negative. This can be an issue and is known as non-zero-centred, this is because the output is always a positive number and results in large gradient updates in a certain direction during backpropagation, making optimization harder [33]. Sigmoid can also stop weight updates which effectively stops the learning process. This is because often the local gradient is or is near zero when there are large negative values [33]. Another common activation function is the Rectified Linear Unit [34]:

$$max(0, x) \tag{2.12}$$

where x is our weighted sum, thus removing only negative weighted summations. When are model is learning we do not want all our neurons to be activated (that is to return a positive result for each neuron), as the computational cost of such a task is extremely high. However, having all our neurons activated can be perceived as a good because we would be not be disregarding any knowledge that we learnt in during a single iteration. To help with this computational limit we would want only a selection of the most important neurons to be activated. ReLU has the benefit of around 50% of the neurons would be activated, thus the model becomes lighter [33]. This is why ReLU become the "go-to" activation function for large deep learning models. The drawback to ReLU - just like sigmoid - is that it is zero centred. To help with this issue Leaky ReLU was proposed, which takes into account a small number of negative values [35]:

$$max(0.1x, x) \tag{2.13}$$

2.4.1.3 Softmax Function

The softmax function is the final layer of the neural network, where it is used to normalize the output of the network to a probability distribution. This distribution consists of k probabilities where k is the number of labels. Softmax is required as the neural network penultimate layout outputs real-valued scores that do not scale, thus is challenging to work out which label the network is predicting. Generally the softmax function is as follows [36, 37]:

$$\sigma(\overrightarrow{Z})_i = \frac{e^{Z_i}}{\sum_{j=1}^{K} e^{Z_j}} \tag{2.14}$$

where $\overrightarrow{Z}$ is our input vector from the neural network consisting of $\{Z_0, ..., Z_i\}$, e^{Z_i} is the standard exponential function which gives a positive value above 0. The bottom term is our normalization term that ensures we have a valued probability distribution, and finally, K is the number of classes. The drawback of softmax is that it tends to produce results close to 0 or 1, thus should be careful in using it as the true probability, especially when measuring uncertainty or other Bayesian machine learning tasks. [38]

The softmax function is a smoothed and differentiable alternative to the argmax function [24]. During training, we use softmax as its differentiable allowing us to optimize a loss function. However for inference, we often just need a single predicted label as our output. In these situations, we argmax, which converts the input vector to zeros apart from the largest probability, which is set to 1.

2.4.1.4 Backpropagation

Backpropagation (short for Backward Propagation of Errors) is the key algorithm in supervised learning and brought the recent surge in popularity of deep learning [24]. At the end of each training iteration a loss function is calculated, this is a scalar representation of how far the predictions are from the ground truth labels. We use backpropagation in calculating the gradient of the loss function concerning each of the weights in the neural network. This ensures each weight is updated to gradually reduce the loss function over time. We refer to this process as going backwards as the calculation of the gradient goes backwards from the last year to the first layer of the neural network. At any particular layer in the neural network, we calculate the gradient using the gradients of all the following layers using the chain rule.

The first stage in backpropagation is to accept input vector X_i and feed it through the layers of the network to produce $\hat{y}$, as shown in equation 2.10. This process is known as forward propagation. We then compute our error with the loss function $\mathcal{L}(y, \hat{y})$. The lower the value the closer $\hat{y}$ is to y. When the neural network first starts training the result of the loss function is high and over the training iterations, we should have a low error rate. To lower the result of our loss function we gradually adjust the weights. This requires us to calculate the derivate of $\mathcal{L}$ with respect to every weight in the neural network:

$$\frac{\partial \mathcal{L}}{\partial W^i_{jk}}, \tag{2.15}$$

where $W^i_j k$ is the weight of neural network going from neuron j in layer $i-1$ to neuron k in layer i.

We could compute each derivative of $\mathcal{L}$ with respect to each weight using the chain rule, however, this is extremely inefficient as we could have already computed the derivative of the layer $i-1$. Therefore in backpropagation we first calculate the derivate of the last layer N, we then use this derivate in the chain rule formula for layer $N-1$. So we work back through the neural network, each time using the last derivate calculation to obtain the current layers derivative using the chain rule.

2.4.1.5 Regularization

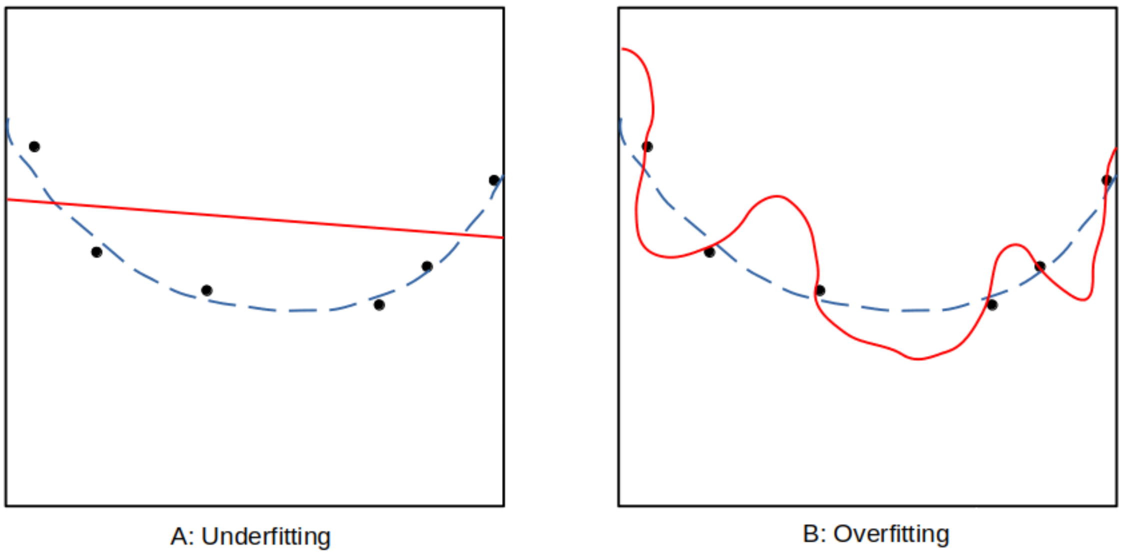


Figure 2.6: A: Underfitting - Occurs when models cannot obtain a low training error. B: Overfitting - Occurs when the gap between training and testing error is large. The dots are the training samples and the red line is our predicted line by a trained model.

The objective in machine learning - for a supervised learning setting - is to perform well on previously unseen data. This notion is called generalization. During training we show the training dataset to the model and compute some loss function to measure the error, this is called the training error. However, we also want the testing error (also called generalization error) to be as low as possible. We estimate this error by measuring the performance on the testing dataset. In the loss function of linear regression (as shown in equation 2.3) we compute the loss on our training data (x_i, y_i), however, what we care about is the loss on the testing dataset:

$$\mathcal{L}(c,m) = \frac{1}{2N}\sum_{i=1}^{N}((c + mx_i^{test}) - y_i^{test})^2 \tag{2.16}$$

To affect the performance on the testing data when we are only observing training data, we make a few assumptions: (1) we assume the data of both training and testing sets are not collected arbitrarily. (2) We assume that each sample (x_i, y_i) from both datasets are independent of each other. (3) Finally, we assume that both training and testing datasets are identically distributed, meaning that we collect the data from the same probability distributions.

Under the supervised learning process we sample our training dataset and choose the weights that reduce the training error, then we sample the testing dataset. We can expect that the testing error to be greater or equal to the training error. To determine how well a model generalised is directly related to the models' ability to make the training error small, while also making the gap between the training and testing error small [24, 26].These 2 objectives correspond to the 2 challenges of machine learning: underfitting and overfitting, as illustrated by figure 2.6. To address underfitting a common strategy is to use deeper neural networks with more layers and more neurons per layer. This is because it is believed that the model may not have enough neurons to capture the patterns in the training dataset [39]. It is also possible that the model has not found any of the optimal weights yet, thus could require more training time. Overfitting occurs as a result of fitting to the training data too well and not learning the general patterns of the training dataset. A way to address this is to add noise to the data or add more diversity to the dataset. This is known as data augmentation and is one of many regularization techniques. Another common regularization technique is to add a regularization term to the loss function, which is a type of penality to our model f:

$$\sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i) + \lambda R(W) \tag{2.17}$$

where $\mathcal{L}$ is our loss function, λ is a hyperparameter which determines how important the regularization term is and finally $R(W)$ is our regularization term where we pass the weights of our model f. The 2 main regularizers that are used are L1 and L2 regularization:

$$R(W) = \sum_{i=1}^{N} w_i \tag{2.18}$$

$$R(W) = \sum_{i=1}^{N} w_i^2 \tag{2.19}$$

2.4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is one of many types of deep learning algorithms that are designed to be used with structured multidimensional arrays of data. originally gaining huge popularity in computer vision due to images being a type of multidimensional data and its representation learning properties. At the time many approaches in computer vision used hand-made approaches or algorithms like Histogram of Oriented Gradients (HoG) [40]. There was a huge desire to have the features be learnt instead of using an algorithm or hand-made features. CNN's take advantage of the spatial domain of images by learning localized spatial relationships of features. This is accomplished by using kernel-based convolutions to extract the localised features, named convolutional layer. During training we learn the formulation of the kernel space which acts as our weights, providing a small parameter set to optimize with backpropagation. This is opposed to neural networks which have many more weights when using multidimensional arrays. Within a convolutional layer, it is common to stack many kernels together in sequential design, it is this design that allows us to learn the complex hierarchical features of some images. Each convolutional layer is followed by an activation layer, while some may also have a pooling layer.

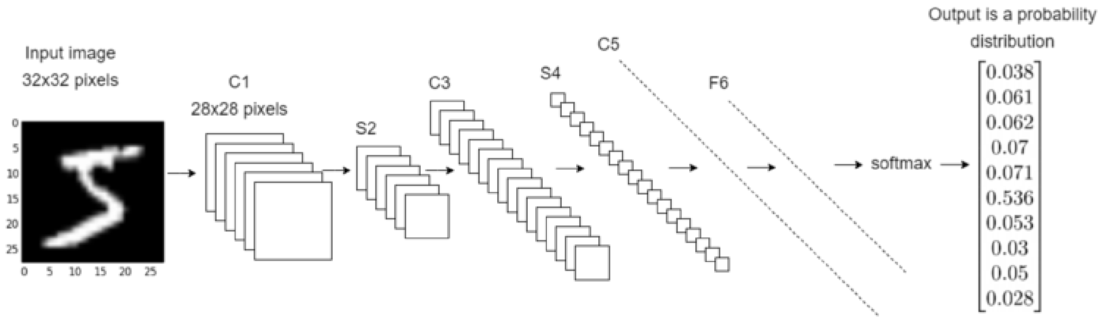


Figure 2.7: C1: The convolutional layer has 6 kernels of size 5x5 which ”walk over” the raw input image, and outputs 6 28x28 images(known as feature maps). The goal of the first few layers is to find basic features like edges and corners. S2: The average pooling layer takes a square of 4 pixels (from the feature map) and averages them to a single pixel. Thus the pooling layer scales down the image by a factor of 2. C3: the second convolutional layer that has 16 kernels of size 5x5. S4 is another pooling layer with the same specification as S2. C5 is a fully connected convolutional layer which has a 1D array of length 120 as the output. Each element of the output array is connected to all 400 weights in S4 (5x5x16). F6 is a fully connected layer mapping the 120 elements array to a 10 element array, where each element corresponds to a digit. [41]

LeNet is one of the earliest convolutional neural network architectures which was proposed by Yann LeCun et al in 1998 [41]. It is often used today to aid in understanding the design principles of CNNs. LeNet was designed to recognize handwritten digits 0-9, this the input is the raw image of a digit and the output is the probability distribution representing the confidence score of the input being a digit 0-9. LeNet-5 is a very small CNN - by today's standard - with 3 convolutional layers, 2 pooling layers and a fully connected layer. It is powerful enough to classify digits but not, for example, all 26 letters of the alphabet. Today many of the CNN architectures have millions of parameters and tens of layers.

The convolutional layer is the key principal in CNNs that utilising the representation learning scheme. Each kernel ”walks over” the feature map producing an output that describes the result of the learnt kernel. Each kernel is a square matrix where each element is a learnt weight of the features within the kernels region of interest. In training, we optimize the kernel weights to learn various localized features. The early kernels in the network learn basic features while later kernels learn more complex features of the image like texture.

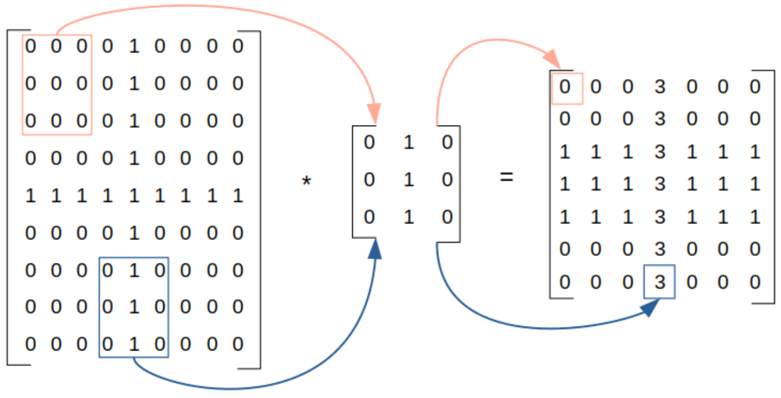


Figure 2.8: The first operator of the CNN architecture, the convolutional layer: the kernel is placed in the top left corner and slides over the feature map multiplying each element in the regain of the kernel with the element value in that kernel.

We formally define the feature map as a result of a kernel which contains the weights to be optimized:

$$a^{l}_{i',j'k'} + 1 = \sum_{i\ j\ k} W_{ijkk'}\ a^{l}_{i+j',\ j+j',\ k'}, \tag{2.20}$$

where i, j, k are the array height, width and channel indexes of the input, kernel W is convolved over the input a^{l} and returns a^{l+1} [42].

2.4.2.1 Architectures

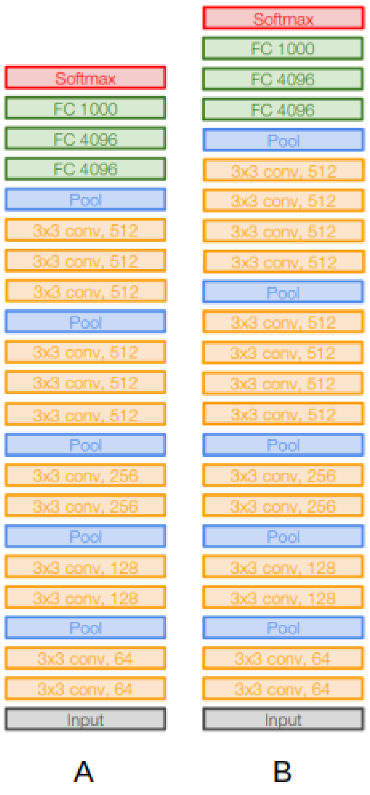


Figure 2.9: A is the 16 layer variant of the VGG Network while B is the 19 layer variant [43, 44]. All convolutional layers have 3x3 kernels with a stride of 1 and a padding of 1. All pooling layers use max-pooling and have a kernel of 2x2 and a stride of 2.

The architecture of a CNN refers to the design of the network structure. There are many different designs but generally, the sophisticated designs we have today are inspired by either the VGG Network, ResNet or MobileNet. VGG16/19 are designs that were originally proposed for the ImageNet challenge in 2014 and are inspired by AlexNet [43, 45, 2]. The ImageNet challenge was a competition between 2010 and 2017 to evaluate algorithms for object detection and image classification on the scale of 1000 classes. While developing the VGG Network the authors found that 2 3x3 kernels have the same receptive field as a 5x5, this is a useful insight as 2 3x3 kernels have fewer parameters and thus less computation than 1 5x5 kernel. With the idea that bigger models perform better, computation became more important and thus insights like this helped in authors winning the 2014 ImageNet challenge.

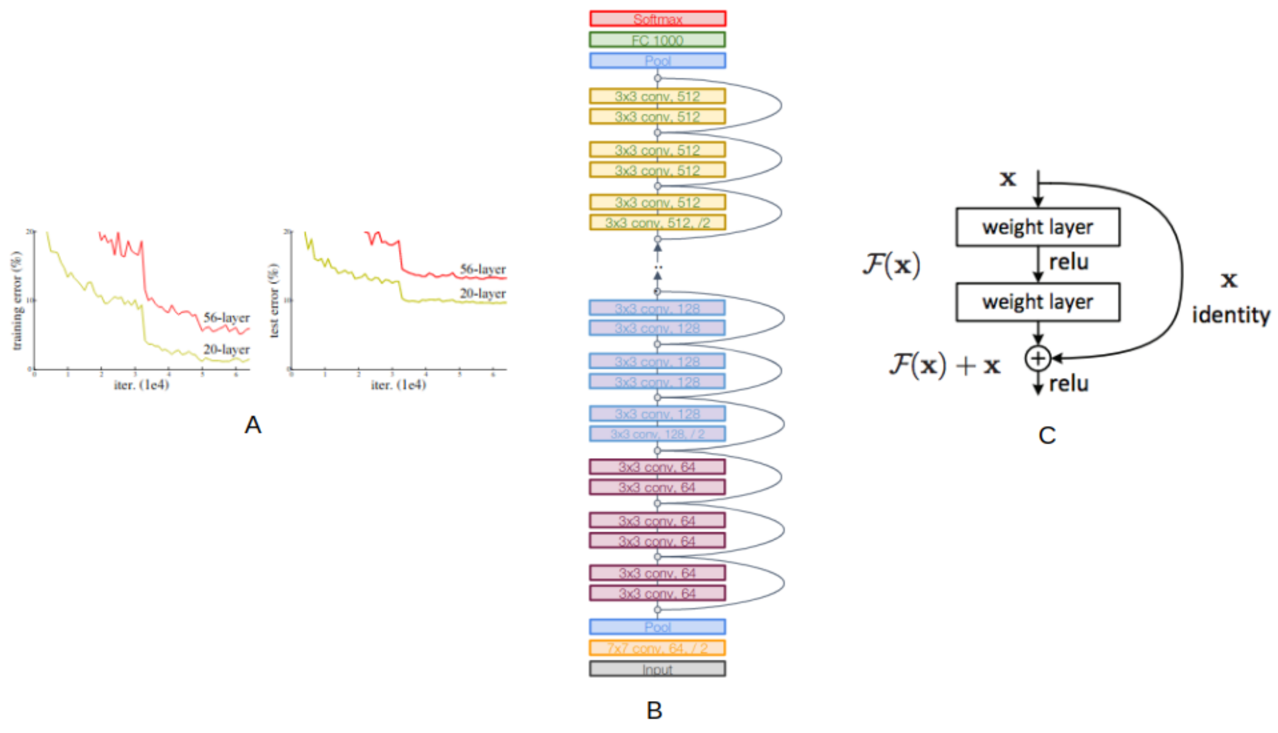


Figure 2.10: Each of these diagrams represent the 3 contributions that He et al proposed for CNN architecture designs. In Diagram A they show the training error(on the left) and the testing error (on the right) of 2 different networks, a 20 layer and a 56 layer network. They show that the deeper the model the more it begins to underfit to the dataset. As shown by the 20 layer model performing better both in training and testing. This resulted in the construction insight. Diagram C is the Residual block which allows us to skip convolutional layers, allowing deeper models to emulate shallower models. Diagram B is the resulting network that was built using stacks of residual blocks. [44, 46]

He et al found that if we kept stacking convolutional layers, the training will eventually get worse, this is illustrated in charts A of figure 2.10 [46]. They took a 20 layer and a 56 layer network and trained it on the CIFAR dataset [47]. The charts seem to show that the deeper model is underfitting to the data, and as a result, they hypothesised that deeper models are harder to optimize and not learn to identify functions to emulate shallow models [46, 44]. This leads them to the following construction insight:

Considering a shallow and deeper model, the deeper model needs to copy the shallow model, with identify mappings. This suggests that deeper models should result in no higher training error than the shallow model [46]

As models were getting larger they began to overfit and as a result, He et al proposed the Residual Block and Residual Network (ResNet) as illustrated in image B and C of figure 2.10 [46]. The residual block is a way to make it easier for deeper networks to learn to emulate shallower networks in the situation where the deeper network has more layers than it needed. A residual block consists of 2 convolutional layers, in image c of figure 2.10 this is referred to as $\mathcal{F}(X)$. Where X is a feature map. At the end of the block, we add the input X to the result of $\mathcal{F}(X)$, this adding of X to our function is known as the residual additive shortcut. As a result of adding this shortcut to our model we can now learn the identity function, this is accomplished by setting weights in the convolutional layers to 0. Allowing use to emulate shallower networks.

A Residual Network is a stack of these residual blocks where the design of the convolutional layer structure is inspired by the VGG networks. The ResNet design divides its convolutional layers into stages, between each stage we double the number of channels starting with 64 channels, each layer also has a 3x3 kernel simpler to the VGG network. For each pair of layers, we add a residual block, allowing us to skip layers if needed. The result of this network is that we have larger networks but have the same or less computational complexity compared to the other architecture designs of its day. These deeper networks while utilising less computational complexity helped beat the state of the art results and the authors won the ImageNet challenge that year.

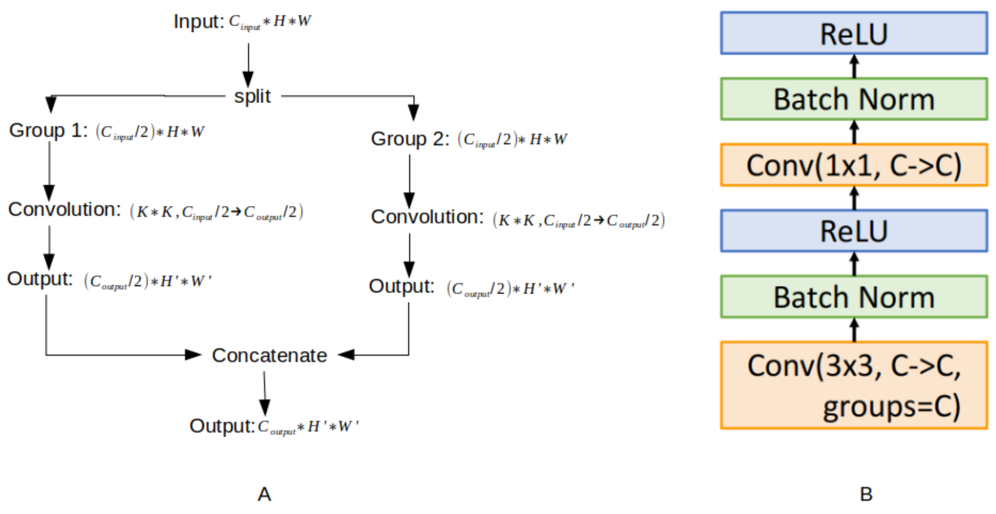

Input: $C_{input} * H * W$
split
Group 1: $(C_{input}/2) * H * W$
Group 2: $(C_{input}/2) * H * W$
Convolution: $(K * K, C_{input}/2 \rightarrow C_{output}/2)$
Convolution: $(K * K, C_{input}/2 \rightarrow C_{output}/2)$
Output: $(C_{output}/2) * H' * W'$
Output: $(C_{output}/2) * H' * W'$
Concatenate
Output: $C_{output} * H' * W'$
A
ReLU
Batch Norm
Conv(1x1, C->C)
ReLU
Batch Norm
Conv(3x3, C->C, groups=C)
B

Figure 2.11: Image A displays a flowchart of standard grouped convolutions with 2 groups, in which we take input and split the channels in half. We then perform our convolutions on each slide in parallel. Finally, we concatenate the 2 slides which result in our output. Image B is the design architecture of MobileNet. It consists of 2 blocks that we can repeat as many times as needed. Block 1 is a depthwise convolution with a 3x3 kernel, followed by batch normalization and a ReLU activation function. Block 2 is the same as block 1 apart from we replace the depthwise convolution with a pointwise convolution. [44, 48, 49]

As the development of new and better architecture designs where proposed, a bigger focus was spent on computational complexity, therefore questions like "Can we sacrifice a small portion of accuracy for much better computational complexity?" or "Are we able to train and test our models on embedded devices?". This inspired Howard et al and proposed an efficient design called MobileNet [48]. MobileNet uses a special case of grouped convolutions in which we split the task of convoluting over an image in parallel [49]. This is accomplished by dividing the channels of our input by the number of groups, which we set as a hyperparameter. The flowchart A in figure 2.11 illustrates a convolution with 2 groups. We divide our channels by 2, perform a convolution with a kxk kernel on each side and finally concatenate the 2 results. We can measure the computational complexity by using the Floating Point Operations per Second(FLOPs) measure. In a normal convolution we measure FLOPs by computing $C_{output}C_{input}K^2HW$, where C_{output} and C_{input} are the number of channels for the output and input, K is the size of the kernel, and finally, HW is the height and width of the input image.

In a grouped setting we divide the FLOPs buy G where G is the number of groups:

$$FLOPs = \frac{C_{output}C_{input}K^2HW}{G} \tag{2.21}$$

MobileNet uses a special variant of grouped convolutions where G groups are equal to the number of channels, known as depth-wise convolutions [50]. Along with depthwise convolutions we often use another type of convolution that uses a $1x1$ kernel to iterate through every pixel. Known as pointwise convolution, it forms a class of convolutions known as depthwise-separable convolutions [50]. The larger a model gets the more sensitive to the initial weights the model becomes. A common hypothesis for this is that the distribution of inputs to layers in the model may change with each minibatch as the weights get updated. This hypothesis is referred to as internal covariate shift. To solve this the technique known as batch normalization is used to normalize the inputs to a layer for each minibatch [51]. A nice property of this technique is that it reduces the amount of training time required as a result of normalizing the learning process. With techniques like depthwise-separable convolutions and batch normalizations lead use MobileNet, as illustrated in image B of figure 2.11. MobileNet uses 2 blocks, the first is a depthwise convolution as input followed by batch normalisation and a ReLU activation function, the second is the same apart from the use of a pointwise convolution instead of a depthwise convolution. This structure repeats until we get to our standard task-specific block which is dependent on the task, however, it often includes a fully connected convolutional layer to fully connected layer and finally a softmax layer.

2.5 The Challenges of Combining Deep Learning, Active Learning and Online Learning

In recent years there has been many workshops and talks by many researchers on the challenges of deep learning techniques learn and if these techniques are underfitting, while others have talked about the changing distribution of data and complexities. Deep learning has a strong ability to process large amounts of high-dimensional data with an accurate feature representation, while active learning can reduce the cost of labelling and online learning can minimise the use of more data while also having a changing distribution. There is hope to combine these approaches with the obvious benefit of expanding the application potential due to each approach solving each other's drawbacks. Applying deep learning to an active online learning setting is difficult to apply this is due to:

- Minimal data and labels: Active learning uses a small amount of data and online learning has an ever-changing pool of samples, meaning that different classes will appear more often than others. Deep learning is very greedy with the amount of data it requires, contradicting active and online learning [52].
- Uncertainty Modelling in selecting a sample for querying: Many active learning techniques use an uncertainty metric to find which samples or classes the model is least confident about. These samples are then queried to an oracle. Many deep learning techniques use a softmax layer to obtain the probability distribution on a sample. Softmax has been proven to be too confident, as a response the active learning community find it unreliable as a measure confidence [53].
- Representation Learning: in Active Learning: Active learning mostly focuses on training the classifier with some query strategy. Many strategies assume that there is a fixed feature representation. As discussed in section 2.4.2 deep learning techniques learn the

feature representation and the model training at the same time. Current solutions only propose fine-tuning models for active learning frameworks [54, 55].

Some proposes have been made such as applying data augmentation to the minimal data issue or use a multi-stage model to deal with the representation learning issue [54, 56, 57]. Others have proposed learnable query strategies by using Bayesian active learning techniques [58]. No approach solves all these problems and continues to be an open problem today [55].

Some have proposed deep learning solutions that can solve some of the challenges above. One of the largest areas of active learning has been applied successfully in computer vision. The main challenge in this context is that traditional active learning methods fail to gain good performance on small labelling when using high-dimensional data [55]. Wang et al [54] proposed assigning pseudo-labels to unlabelled samples that have high confidence from some classifier, to then add them to an uncertain sample set. which is queried by a deep Bayesian method for measuring uncertainty. Uncertain samples are queried to oracle. The model trains end to end with a multi loss consisting of the classifier loss and the Bayesian method loss functions. This approach is also well suited for online learning as we do not know the labels beforehand, but over the training, the model replaces the pseudo-labels with the labels written by the oracle. Object detection has also seen a lot of use in active learning due to the high label costs [59, 60, 61]. However, due to the computational complexity cost and training time, object detection is rare to see in online learning settings. Many of these techniques require a fixed representation space, therefore many use transfer learning methods to first learn the representation and then apply to an active learning framework. This results in some methods using a two-stage approach, the drawback is that the second model is highly dependent on the first models' results.

2.6 Object Detection

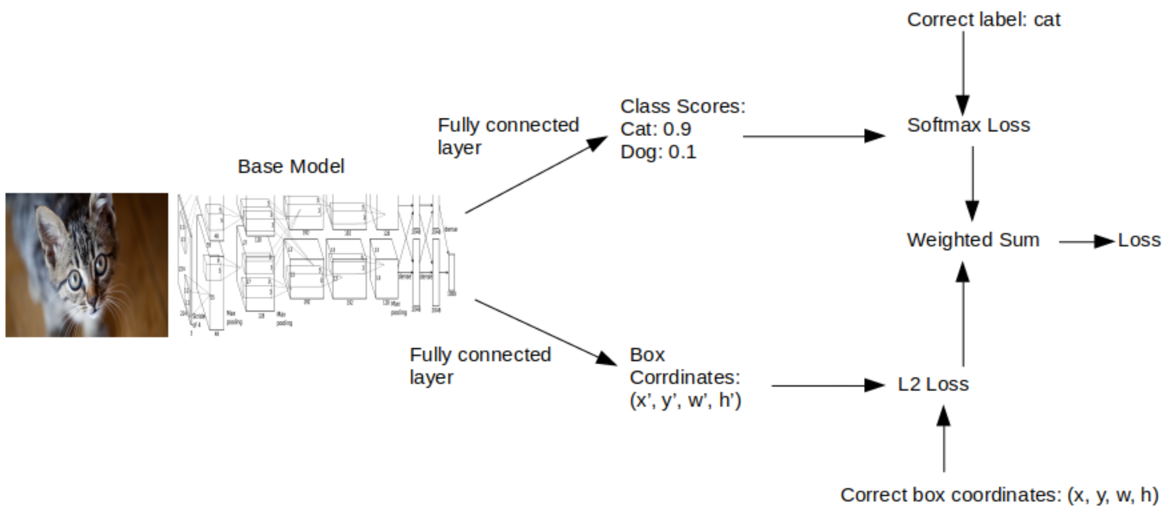


Figure 2.12: General model structure for an object detector in deep learning. The base model for finding features (hence the name feature executor) in this example is AlexNet, followed by 2 fully connected layers that branch into their different tasks of what the object is and where the object is in the image

One of the most studied and popular areas in the field of computer vision, object detection is the task of localising and classifying each semantic object within an image or video. The challenges of object detection are (1) multiple objects that could be anywhere in the image, (2) objects are overlapping, (3) 2 types of output for each object (a category label and a bounding box), (4) large images result in large processing time.

Generally, in a deep learning setting, we have some feature executor as our base model which branches into 2 layers; a fully connected layer with softmax for classification of the object, and a regressor layer to find "where" the object is in the image. Therefore the final model has 2 outputs and can be trained end to end. This is illustrated in figure 2.12. There are many base models to select as we can take any model that was designed for classification and modify the last fully connected layers. Thus any model from subsection 2.4.2.1 can be used for a detection task. As we have progressed within this field bigger models with more data have been used to get better accuracy, however at the cost of efficiency in training and testing time. Girshick et al were the first to propose a method which uses a CNN for gaining greater accuracy in object detection tasks, named Region-Based Convolutional Network Network(RCNN) and uses two stages within the detection [6]. The first stage finds regions within the image that the model thinks might have an object in it, while the second stage extracts feature from the regions which are then used to predict a label and a bounding box. With each variant of RCNN better accuracy and quicker testing time resulting in high localization and object recognition. However, even with the fastest variant of two-stage detectors were still slow when being used with video. Some approaches have been used to combat efficiency issues such as MobileNet and PeleeNet but are a current research problem being pursued [62, 63]. One-stage detectors are alternatives to the two-stage detectors in which they propose methods to directly predict bounding boxes from the image, this no first stage of getting regions. Generally, one-stage detectors are more time-efficient and are used for real-time object detection tasks. The trade-off of using one-stage detectors is that they are often less accurate in localising and classifying an object [64].

2.6.1 Two Stage Object Detectors

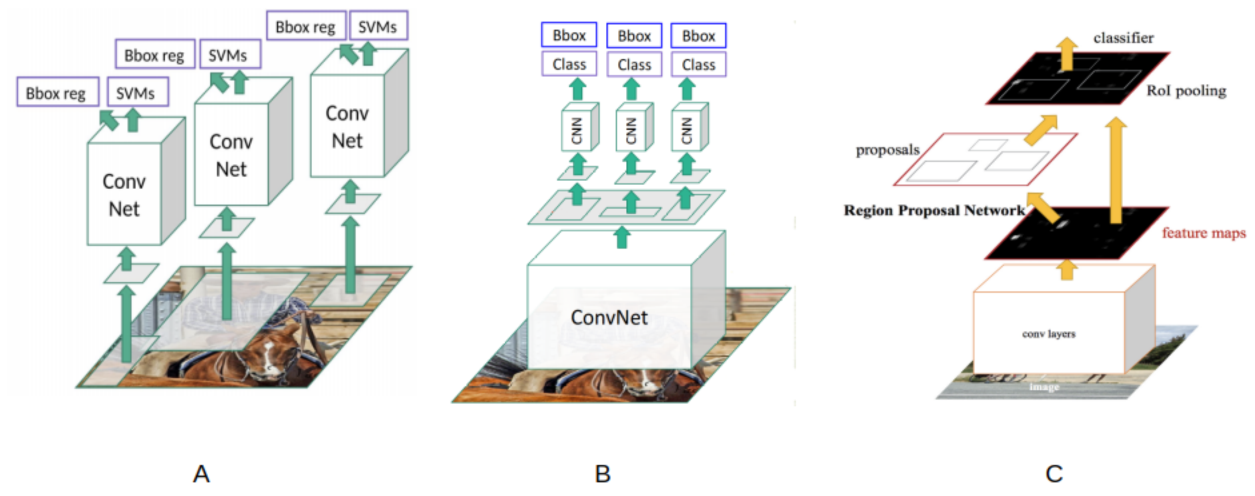


Figure 2.13: The RCNN design splits process of object detection into 4 modules: (1) generate region proposals, (2) extract features from regions, (3) classify modules in the regions, (4) predict a bounding box for that region. A is the first design in the RCNN family which uses selective search on the raw image, wraps the resulting regions, extracts the features of the regions with a pre-trained CNN and finally predict the object and bounding box. Fast RCNN (image B) changes RCNN by extracting the features from the raw image and then use the selective search on the final feature map. Each of the resulting regions is resized with an ROI pooling layer so we can keep the spatial information of the region. We then get a feature vector from each of the regions bypassing them through a fully connected layer. Finally given each vector we predict the object and the bounding box of in the region. Lastly image C is the Faster RCNN variant where we completely replace selective search with a learnable method called region proposal network [6, 65, 66].

RCNN was the first to propose a CNN architecture that could perform well on object detection tasks on the PASCAL VOC dataset [6, 67]. The RCNN design consists of 4 modules:

1. Generate region proposals
2. Extract fractures from regions
3. Multiple Support Vector Machines (SVM) to classify objects from each feature vector
4. Bounding box regressor for accurate placement of the bounding box

For module 1 Girshick et al propose using the selective search algorithm, which generates initial sub-segmentations of the image [68]. The algorithm then selects all the sub-segments that are the most similar and combine them. This is an iterative process where it keeps combining the sub-segmentations, thus we need a hyperparameter to state how many times we iterative. Each segment of the image becomes a proposed region. In module 2 we use a pre-trained CNN to extract a 4096-dimensional feature vector. The CNN requires a fixed size input, thus Girshick et al wrap each region proposal to be $227x227$. Within module 3 we use categorical independent SVM where each SVM is a binary classifier for background or the specific class that the individual SVM is designed to detect. Each SVM do not share the weights and therefore are independently trained. Finally, in module 4 we use a bounding box regressor to predict the correct (x', y', h', w') given the (x, y, h, w) from the position and the size of the region proposals. This step is required as the authors found that while testing the bounding boxes were offset due to the region proposal process. RCNN has a very high competition time due to using selective search [64]. Each image has around 2000 regions proposed in both training and testing time that then has to be passed to a CNN to extract its features. This process is very slow at around 40-50 seconds to predict each new image using some of the highest-end hardware on the market at the time. A result of this slow prediction time meant that this approach was almost impractical on large datasets like COCO and PASCAL VOC.

With the limitations of RCNN Ross Girshick proposed a faster variant of the same design named Fast RCNN [65]. As RCNN performs a forward pass on the base model for each region proposal without sharing computation, the SVM classification can take a long time. TO speed this process up Fast RCNN extracts the features from the entire image. Then perform a selective search on the final feature map. Fast RCNN still needs a fixed size region proposes, therefore the author utilizes a region of interest (ROI) pooling layer. This layer takes as input all regain proposal coordinates and we take the feature map sections that correspond to the coordinates and converts it into a fixed dimension. These fixed size regions are then passed to a fully connected layer resulting in a 4096-dimensional vector. Finally just like RCNN we pass the vector to our SVMs to classify the object (if there is an object in the region proposal) and predict the bounding box with a regressor layer. Fast RCNN was a huge improvement on RCNN with a large amount of time saved for the feature extraction and a large reduction in disk storage due to not storing the features to disk. Fast RCNN also uses an end to end training process due to a multi-task loss on each region proposals. ROI pooling layer has the added improvement of not needing to wrap the image and therefore we reserve the spatial information of the region proposals. The experiment results in Fast RCNN having a 7% increase in mAP(the metric used to evaluate object detectors, see section 2.6.3) and is 9 times faster than RCNN in both training and testing time [65]. Due to the computational speed of selective search, this approach still could not be used for real-time object detectors, or when using very large datasets. There was also a desire to experiment on if we can learn to find regain proposals during the training stage. This lead Girshick et al to create Faster RCNN with the use of a region proposal network (RPN) [66], which replaces selective search in Fast RCNN. The RPN is a fully convolutional network that can predict region proposals with many different scales and aspect ratios., The network decreases the generating region process speed due to it sharing fully convolutional features and a common set of convolutional layers. The RPN can predict regions by ranking the region boxes(known as anchors) and propose the regions that are the most likely to contain an object. This accomplished by generating 9 anchor boxes at 3 different scales and aspect ratios. Each of the region proposals is parameterized relative to one of these other boxes. We then measure the distance between a predicted region proposal and the ground truth bounding box, we then optimize this metric during training. The experiment results of Faster RCNN show that there is a 4% increase in performance and it is 10 times faster than Fast RCNN using the same base model and dataset.

2.6.2 One Stage Detectors

You Only Look Once (YOLO) is one of the first proposed one stage object detectors by Redmon et al in 2015 [69]. They were able to successfully show an accurate real-time object detector. In there, experiments YOLO performs at 45 fps compared to Faster RCNN which performs at 7 fps on the same dataset and hardware. They were able to accomplish this in one stage by thinking of detection as a regression problem and predicting all bounding boxes of the image in one forward pass of there model.

The YOLO pipeline consists of dividing the input into an SxS grid, where each cell is responsible for detecting a single object. Each cell in the grid predicts B bounding boxes $(x, y, w, h, confidence)$. x, y are the coordinates of the centre pixel in the cell. w, h refers to the width and height of the bounding box prediction. The confidence score is the probability that some class y_i is in the bounding box multiplied by the IoU metric of the predicted box by the ground truth bounding box. As all bounding boxes are predicted in one forward pass of the YOLO model the output tensor is $SxSx(Bx5 + C)$, where C is the number of classes. The network itself consists of 24 convolutional layers followed by 2 fully connected layers, one for predicting the bounding box coordinates and the other for object probabilities. The YOLO family of methods have become one standard way to accomplish real-time object detection with each iteration adding different design choices from previous works such as batch normalization, and adding independent logistic classifiers to the ends of the models, useful for overlapping labels.
The second family of methods for use in real-time object detection are the Single-shot detectors (SSD) proposed by Liu et al in 2015 [70]. The method behind SSDs is to use a fixed set of default bounding boxes with many different scales and aspect ratios for different feature maps within the network. The prediction is the class source and box offsets for one of these default bounding boxes. For each feature map, a different scale is used for the default bounding boxes as each feature map will also have a different scale. The feature maps learn to be responsive to a particular scale of each object. During training, we optimize the default bounding boxes for each given class by using IoU with the ground truth bounding box. This should mean that we get many negative bounding boxes as most of the image will be the background. To accommodate for this the authors choose to add a hard negative mining technique where they use the highest confidence loss for each default box out of all classes to optimize those boxes. The results of this method are that it can outperform YOLO and Faster RCNN on detection metrics but have a slower computational speed than YOLO, therefore YOLO still performs better in real-time settings.

2.6.3 mAP Metric for Object Detectors

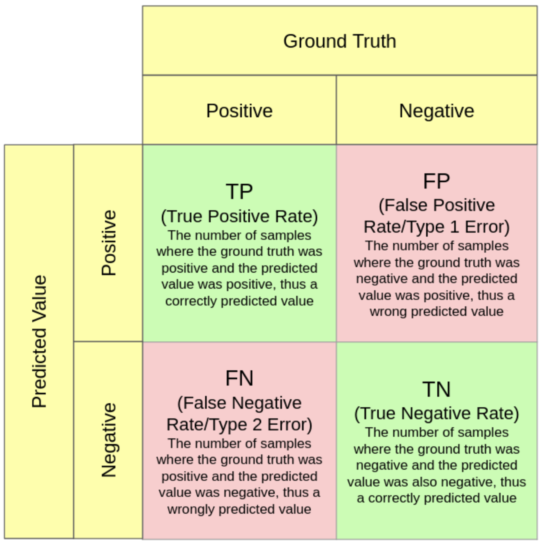

Ground Truth
Positive
Negative
Predicted Value
Positive
Negative
TP
(True Positive Rate)
The number of samples where the ground truth was positive and the predicted value was positive, thus a correctly predicted value
FP
(False Positive Rate/Type 1 Error)
The number of samples where the ground truth was negative and the predicted value was positive, thus a wrong predicted value
FN
(False Negative Rate/Type 2 Error)
The number of samples where the ground truth was positive and the predicted value was negative, thus a wrongly predicted value
TN
(True Negative Rate)
The number of samples where the ground truth was negative and the predicted value was also negative, thus a correctly predicted value

Figure 2.14: Illustration of a Confusion Matrix [71]

A popular metric for evaluating and comparing different object detectors is the Mean Average Precision (mAP), which is computed by first calculating the area under the Precision-Recall Curve (PR Curve). The precision of a given class is the ratio of true positive(TP) over the total number of predicted positives [72]. While recall (sensitivity) of a given class is the ratio of true positive(TP) over the total ground truth positives [73]. We use both precision and recall in a lot of different evaluation metrics for the performance of a classification task. We also merge them in the PR Curve as another form of metric, this is performed by calculating the precision and recall values at different confidence score thresholds.

$$precision = \frac{TP}{TP + FP} = \frac{retrieved\ and\ relevant\ documents}{all\ retrieved\ documents}, \tag{2.22}$$

$$recall = \frac{TP}{TP + FN} = \frac{retrieved\ and\ relevant\ documents}{all\ relevant\ documents}, \tag{2.23}$$

We use the PR Curve to visualise the trade-off between the two metrics. To achieve a high precision the number of false positives needs to decrease, however by doing this we also decrease our recall metric. Furthermore, by achieving a high recall the number of false negatives needs to decrease, however, this would decrease our precision. Object detectors should find all ground-truth objects (low false navigates) while identifying only the relevant object (low false positives) [74]. Therefore if the precision is high while the recall increases then the object detector is considered accurate.

In standard classification tasks it is trivial to find the TP, FP, FN and TN metrics as its purely based on what label the classifer predicted. In object detectors we predict anchors that form a bounding box around our objects, thus we need to measure how close the predicted bounding box is to the ground truth box. This is accomplished as illustrated in figure 2.15. We use IOU to check if a sample is a TP, FP, FN or TN, using the following criteria:

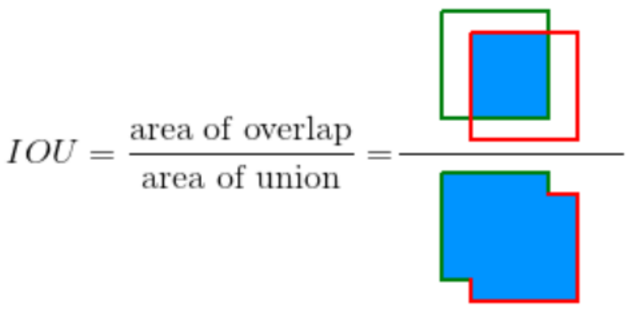


Figure 2.15: Illustration of Intersection Over Union (IOU), the green box is the ground truth while the red box is the prediction. The area that intersects the two boxes is our IOU metric. [74]

Metric	Criteria
TP	IOU >0.5
FP	IOU <0.5 or duplicated bounding box
FN	No detection at all or IOU >0.5 but is the wrong classification
TN	We ignore TN for object detectors as all background pixels are TN

Table 2.2: IoU Metric Criteria

Finally we can compute the area under the curve (or Average Precision in object detection contexts) of the PR Curve. This area is generally considered the average of precision values for each recall threshold. The general definition of Average Precision (AP) is:

$$AP = \int_0^1 p(r)dr \tag{2.24}$$

where $p(r)$ is the plotting precision as a function of recall r, as precision and recall are between 0 and 1 so is AP. A popular trend within academic literature is to interpolate the $p(r)$ due to the ”zigzag” problem in PR curve [74]. We use the interpulate precission to reduce computation for computing the area, the result being a curve which decreases monotonically. A caveat for using this method is that our curve will be less susceptible to small variations in the ranking.

$$p_interp(r) = \max_{\hat{r} \geq r} p(\hat{r}) \tag{2.25}$$

where we get the largest precision over all recalls greater than r [71].

As AP computes a score for a given label we compute the mean AP over all labels, known as mean average precision (mAP). This is the final metric we use to compare object detectors. Table 2.3 displays a few different criteria for computing the mAP, where the COCO challenge variant being the most rigorous. In this case there are 9 different IOU values and we compute the average over all 9 to give a final mAP score.

mAP	Criteria	Notes
AP	IOU = .50: 0.05: 0.95	COCO Challenge Metric
$AP^{IOU=.50}$	IOU = .50	Pascal VOC Metric. Often consisted the standard
$AP^{IOU=.75}$	IOU = .75	Strict Metric

Table 2.3: 3 different criteria that is often seen in literature to define mAP. Note that AP and mAP are used interchangeably [75, 76, 74]

Object detectors often have the issue where there will be many overlapping detections given as output. To find which is the best detection out of the overlapping detections we can apply a post-processing technique known as non-max suppression (NMS). This technique will remove all but the most detections which the detector is the most confident in. The technique works as follows [77]:

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IOU $>$ threshold
3. If any boxes remain, Go to step 1

This technique must be noted that it will have a positive effect on the mAP and is not always clear if a particular paper is using it when calculating their results. However, just like test-time argumentation is valued highly once deployed in application purposes. A drawback to using this technique is NMS may remove good bounding boxes when objects are highly overlapping.

2.6.4 Advances Object Detectors

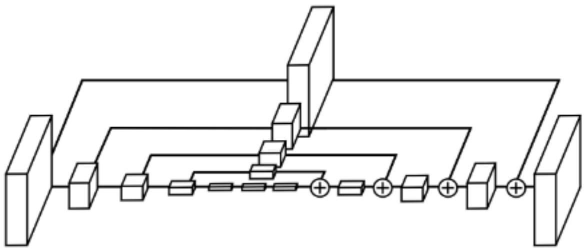


Figure 2.16: The Hour Glass module is very simpler to the residual block, each box consists of a skip connection but with the addition of a pooling layer [78]

One of the most important subproblems of computer vision, object detection, has gained huge amounts of popularity in the deep learning community in recent years. Since the initial proposal of RCNN showing that we could apply deep learning techniques to successfully beat traditional methods in the Pascal VOC metrics [6]. As discussed previously in section 2.6 there are two main types of techniques: two-stage and one-stage object detectors. Both of these require capturing regions within the image, the difference being how they capture the regains. IN recent years a new idea has been gaining popularity, in that instead of predicting a bound box, we predict the location of a set of points, representing the top left and bottom right corners of the bounding box. Known as a keypoint estimation problem, methods like CornerNet and CenterNet have seen success in gaining better metric scores than most one stage and two-stage object detectors [79, 80]. However, keypoint object detectors tend to be slower in both training and testing. CornerNet achieves a 42.2 % mAP source on the COCO dataset, which at the time of its publication was the best performing one-stage object detector. Both CornerNet and CenterNet use the Hour Glass network as the feature extractor, which was specially designed to be used for predicting human poses [78]. The network consists of a stack of hourglass modules, which is illustrated in figure 2.16. This module works like an autoencoder, where we downsample features to then upsample them once we reach a minimal latent feature space. This difference is that the hourglass module consists of a skip connection after each downsampling, allowing the model to learn features across all scales of the input. In CornerNet the Hour Glass Network leads to two final prediction modules for each keypoint of the bounding box. Each prediction module has a corner pooling layer - one of the contributions by the authors of CornerNet - along with predictors for the heatmap, embedding vector and an offset vector. The heatmap has a shape of H by W by C where C is the number categories and each C is a binary mask indicating the location of the corner for that category. CornerNet predicts embeddings for the corners such that the distance of the two corners from the same object is small. A bounding box is only computed if the distance is less than some threshold, and the confidence score is the average of both corner point confidence scores. During training the authors do not penalize all negative locations, instead of the further away the prediction is to the ground truth, the larger the penalty given. This is due to the bounding box of a pair of false corners can still be a good enough bounding box to the ground truth bounding box. CenterNet has the added benefit of a third prediction module to the CornerNet pipeline. As the name suggests this is for predicting the centre key point of an object. The use of such a point inherits simpler functionality as the ROI pooling layer while still keeping the network as a one-stage object detector [80]. The difference between the 2 corner prediction modules the centre prediction module is the use of a centre pooling layer. Center points normally do not have many recognizable visual patterns, therefore the authors propose a pooling layer to capture richer visual patterns. This all results in a model that performs better than CornerNet at roughly 7% better mAP score. However, both CenterNet and CornerNet have extremely slow training and testing time.

Training Time Friendly Network (TTFN) was proposed to have a balance between training time, testing time and computational complexity [81]. The authors achieved this by having one prediction module as opposed to the two and three prediction modules in CornerNet and CenterNet. This single prediction module has one head for the heatmap - which is 1/4 of the size of the input - and a regressor head for predicting the distance in 4 directions from the centre to the edges of the object. Training is accomplished in the same way as CenterNet and CornerNet by using a further away from ground truth penalty. These small but effective changes add up to a model which performs seven times faster than other real-time detectors on the same dataset and feature extractor. The authors showed that in general that their approach is more accurate while also being roughly 10 hours quicker in training [81]. This area is being very promising as more object detectors move to keypoint estimation. This is especially useful in active online learning settings where we require faster training times to minimise the training time bottleneck on live systems.

2.7 Summary

Within this chapter, we reviewed the relevant literature and knowledge required for creating the methodology of the project. We started by discussing the types of learning and learning techniques like transfer, online and active learning. Many of the modern object detectors use deep learning techniques to achieve state of the art results, therefore we explore literature by starting from the perceptron to the structure of neural networks to convolutional neural networks. We also discuss some of the common architectures, which are used as the feature extractor in modern object detectors. Finally, we discuss the most notable object detectors from RCNN to Training Time Friendly Networks.

Keypoint estimation seems to be making a resurgence in object detection in recent years [79, 80, 78, 81]. Allowing us to create composite labels, for example in human pose estimation we can generate bounding boxes over different parts of the body like a persons head or legs. These can be seen as composite labels to the greater label of person. We then use these as the ground truth boxes which training object detectors. Applying composite labels to different object detectors is important to the greater goal of the project. In which we apply composite steel defects to object detectors in an active-online learning setting. This is the situation where the data is given to the model in real-time - hence the online - and due to the complexity of the data and the need to have humans in the loop. We hope to apply this in an active setting. Given the challenges of combining these different styles of learning, there is a need to explore and experiment within the area.

Chapter 3

Responsibly Innovated Human Centric Desgin

Contents

3.1 Introduction

Now that we have established an initial idea for a methodology by find gapes and potential areas of improvement in chapter 2. We move to how we can be responsible for our research and put humans at the core of our project by developing a suitable project methodology. We first discuss the need for this methodology and it was developed by breaking the key areas. We also show how it differs from a human-centric design. Finally, we develop our human-centric methodology for this project based on modified software methodologies.

3.2 History of Responsible Innovation and Human-Centric Design and Why it is Needed

Responsible innovation was established by the European Union as part of their framework programme: "Framework programmes for Research and Technological Development" [82]. Responsible innovation aims to ensure that the process of any scientific research or technological development takes into account any effects and potential impacts on the environment and society [83]. This ensures that we as researchers are kept to a high ethical standard by avoiding any damaging effects of the newly created innovation, working with policymakers to engage with communities which ensure they know the necessary knowledge for further science education, and finally creating a fairer and more inclusive scientific community [84]. Engineering and Physical Sciences Research Council (EPSRC) adopted the responsible innovation programmes and commissioned the AREA framework. This will be the focus of this project [84], which includes the following key areas:

- Anticipation: The main key area of responsible innovation, in which we consider all possible consequences as a result of created innovation

- Reflection: Researchers should always be evaluating all aspects of the research process, this includes the type of data, the process of analysing that data and how we demonstrate our findings
- Ethics: The process of applying principles such as the General Data Protection Regulation and other established ethical policies into our work [85].
- Science Education: Research should be accessible and understandable to a large group of people, while also engaging with the younger generation to inspire new talent
- Equality: By actually engaging equality, diversity and inclusion in the sciences we ensure that the research process and the final innovation is available and suitable to all. Without this process it affects the research such the questions we ask or the models we create, this is because it would only be aimed at a select group of people.
- Open Access: Ensuring that all research is transparent and results are reproducible
- Governance: Institutions that the researchers are apart of must have suitable practices to promote responsible innovation
- Public Engagement: Before actively engaging with the public - such as deploying research studies - we must have a justifiable motivation and reflect on a suitable audience for our research

Human-Centric Design - in the context of research and project management methodology - is the process of actively involving the human perspective in all stages and aspects of the project [86]. This can consist of adopting an agile methodology where humans are involved in conceptualizing, developing, testing and deploying solutions. If we come across specific aspects that we find not to be human-centred we require to move back in the project pipeline to revaluate our steps and make changes, thus can be adapted into any agile methodology. The results of such an approach allow for enhancing solution effectiveness and efficiency, well-being improvements, as well as accessibility and suitability of human use [86]. Which is accomplished by applying human factor and usability techniques into the core solution.

3.3 Developing a Suitable Project Methodology

Now that we have established responsible innovation and human-centric design, we move to applying them to our project. Within this section, we show how we apply such principles to our project management and solution methodology. Anticipation states that we should be aware of all possible negative consequences of our research. This could potentially be a challenge as we cannot predict every future consequence of our project. To help with this we prepared a risk assessment within our project management chapter(5). We split the risk assessment into personal, technical, general project and responsible innovation risks. Where each risk has a likelihood, the impact it will have and a possible remedy for the risk. Reflection and ethics is an iterative process where we will reflect at each step of the project, ensuring that the steps that we make are suitable for the human-centric approach and responsible. This is achieved by weekly supervisory meetings and individual reflective step at the end of each completed step. The different steps of the project are discussed in our schedule section of project management(5). We will also be following GDPR guidelines as well as our institution ethical procedures.

The developed application - as a result of this project - allows for some miner learning of machine learning. Users can select different datasets, label a few samples and train on various models. Users can then compare with different model results, all within a graphical user interface. Thus our goal within the science education principle is to inspire potential students into driving deeper into object detectors and active online learning. As part of our methodology, we have a few datasets with one being based on faces and individual parts of faces. These samples are then trained on different object detectors. A principle goal of this technology is to generalise over a dataset, just like any machine learning task. Thus our task is to generalise over all ethnicities and genders while ensuring that there is as little bias as possible. Therefore we will be evaluating each dataset before adding such a dataset to our project. All datasets will also be publicly available with no requirement to pay for access. The aim of the project is for our code to able to be open source and accessible to everyone. We aim to add our code and how to run the application on a publicly available GitHub page.

Chapter 4

Methodology

Contents

4.1 Introduction

Now that we have set out areas to explore in our literature review (chapter 2) and discussed the need to add human-centric design while also being responsible during the development process (chapter 3). We move to discussing the methodology of our project. In this chapter, we discuss the process of how we developed our active online learning framework with the use of our labelling application, where we break down the various aspects of the different application windows. Following this in datasets (section 4.6) we analysis our selected face and steel dataset to test our framework and other object detectors. There are many different datasets that we could have selected, thus we perform a small evaluation on our datasets to ensure they meet our project criteria. There are many different types of object detectors that we could select, fortunately, they all follow the same configuration and environment pipeline. Therefore in section 4.4 we adapt our labelling framework to support this environment and automatically generate configuration files for any given object detector. Finally we move to the final stages of the project and set out to perform our experiments. An experiment simply starts by first asking a question, within this section we setup some experiment questions and why there is a need for them to be answered.

4.2 Framework Overview

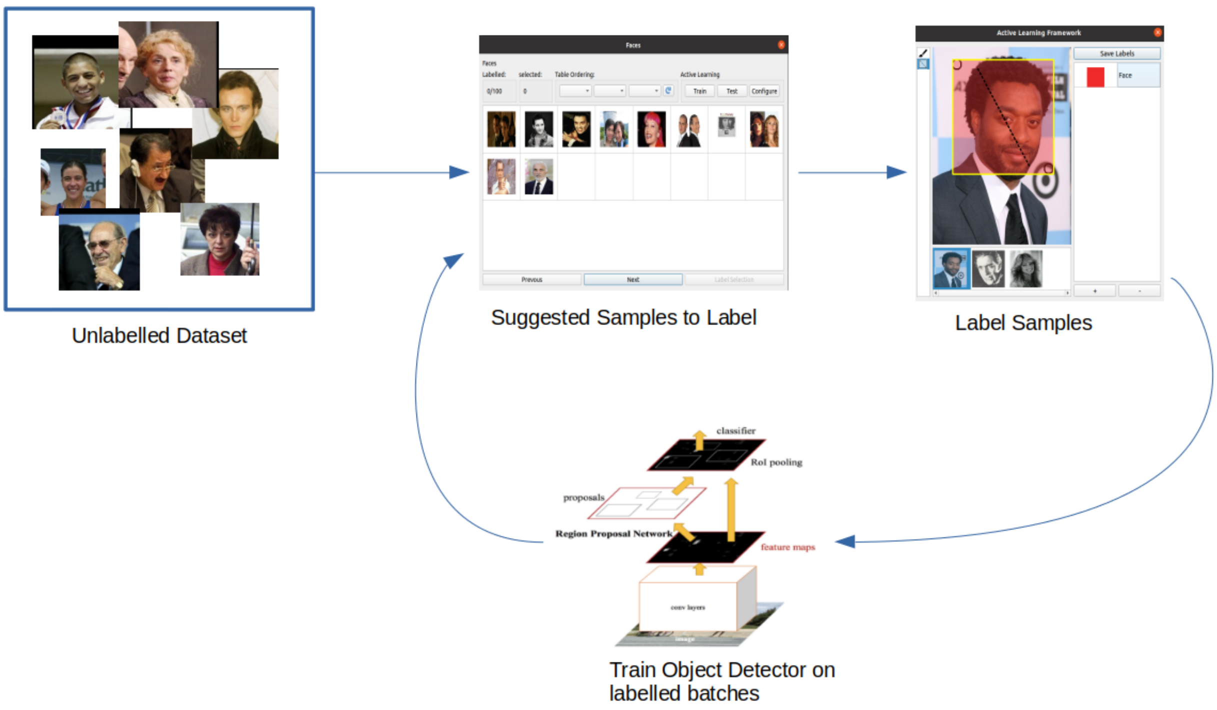


Figure 4.1: Application pipeline starts with some unlabelled dataset and we create a new project in our application. After loading the dataset into the new project, users can select samples to label and train a selected object detector. This is an iterative process of labelling samples and training.

The first stage of this project was to create a labelling application that can support many different types of image datasets. The goals of this early stage were to create project profiles, upload an image dataset, have an image viewer and finally select images to then label them. At the start of the project creation, we generate an XML file which records all our labels for each image. There is support for multiple objects of any class in a single image, the configuration file (of that given object detector) will adapt accordingly. This leads us to the second stage of the project which was to set up our application for use with TensorFlow's object detection library [87]. The library has the following steps for training an object detector:

1. Select an object detector from the model zoo [88]
2. Create a TFRecord for both training and testing sets
3. Create a configuration file in the same format as Protocol Buffer [89]
4. Run library train and evaluate scripts in parallel

There are many ways we can prepare our data for a machine learning task and when using thousands of samples there are a few optimization tricks we can utilise for reading each sample. During training reading, each file individually can have significant overhead because of the magnetic head within a hard disk drive (HDD). HDDs needs to seek the start of each file and jump around the disk finding different parts until the full file has been read. This problem becomes even worst when we store our files on a remote storage service. This problem results in a drastic slowdown of the training process. Tensorflow proposes one idea to speed up the reads of samples with its TFRecord. This file stores our data in a protocol buffer object, which speeds up the training process as all data or a few batches of our data are already stored in memory. In the situation where we can not fit our data into memory, TFRecord will create read batches and models train on those batches. This is all done without much effect from the developer. TFRecord also helps reduce a common problem in deep learning known as GPU starvation. This is when our GPU is waiting for data to be read or proposed. The data should already be processed so the GPU can start training our model straight away. As a result of TFRecord the CPU creates the next batch of samples while the GPU is training on the current batch.

The configuration files is another important file for the process of training and testing object detectors in TensorFlow. It consists of all hyperparameters and paths for its associated object detector. In our framework, users can add new classes at any point in time, even after some training iterations. This requires us to generate a new configuration file at the start of each training session as we would need the number of classes, TFRecord paths and the path of the previous training iteration model. Once we have done at least one training iteration we can finally test our object detector. The detector will give its best prediction and we can adjust its prediction box as well as its class prediction if needed within the application.

4.3 Labelling Application

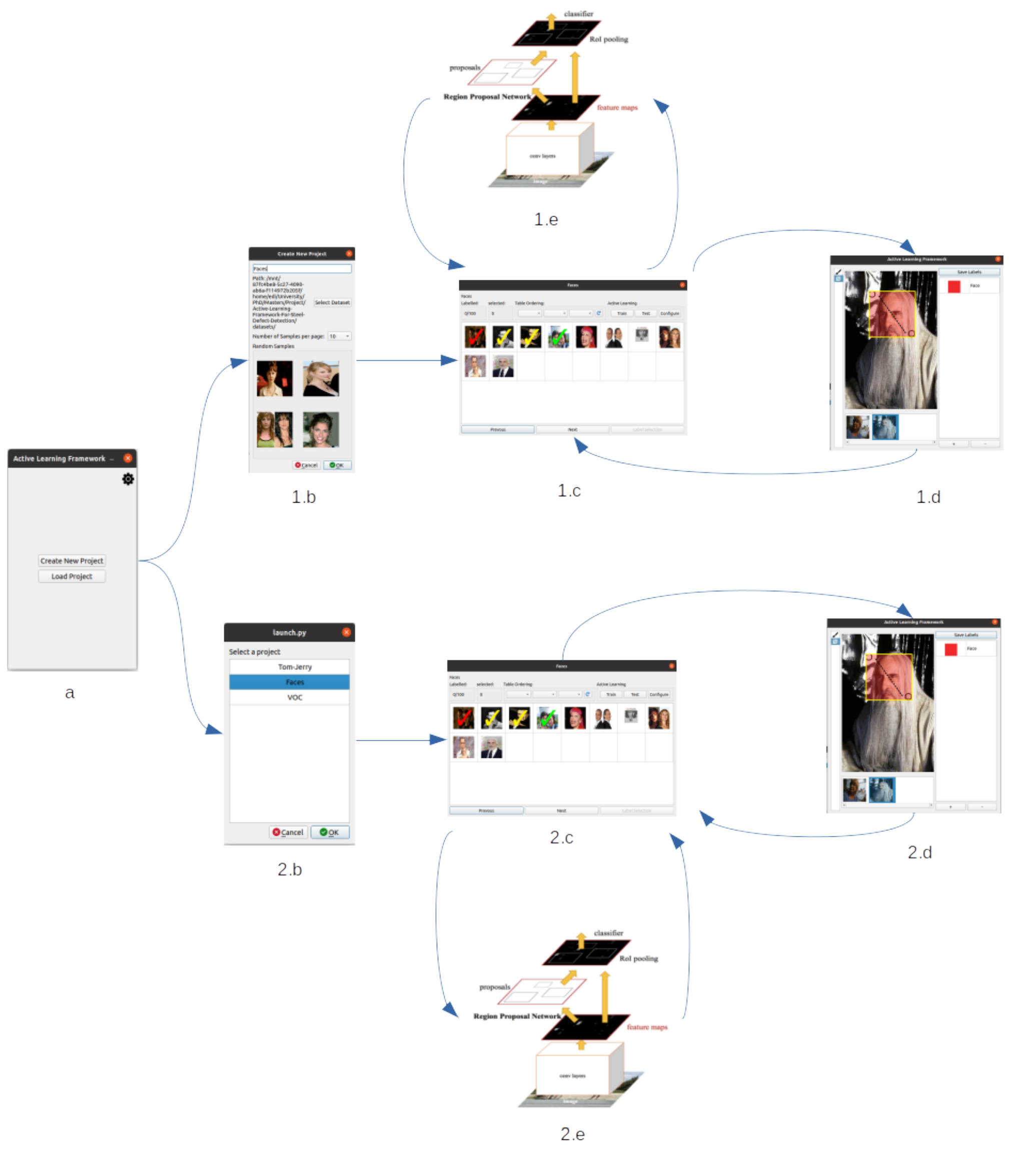


Figure 4.2: An illustrated flowchart of the application where window a is the launching main menu for the application. The user has the choice to create a new project (path 1.x) or lead to an existing project (path 2.x). If the user creates a new project window 1.b is shown. Users give the project a name and upload a local dataset. If users want to load an existing project profile then window 2.b is shown. In whichever case the user is lead to the image viewer (both 1.c and 2.c). From this the user can label samples (1.d or 2.d) or train a selected object detector (1.e or 2.e)

The labelling application was the first stage of the project. We developed the interface with Qt and the image processing with OpenCV [90, 91]. All code in this project was developed in python 3.8. Figure 4.2 illustrates a flowchart of actions the user can do from launching the application to labelling a sample to training labelled samples.

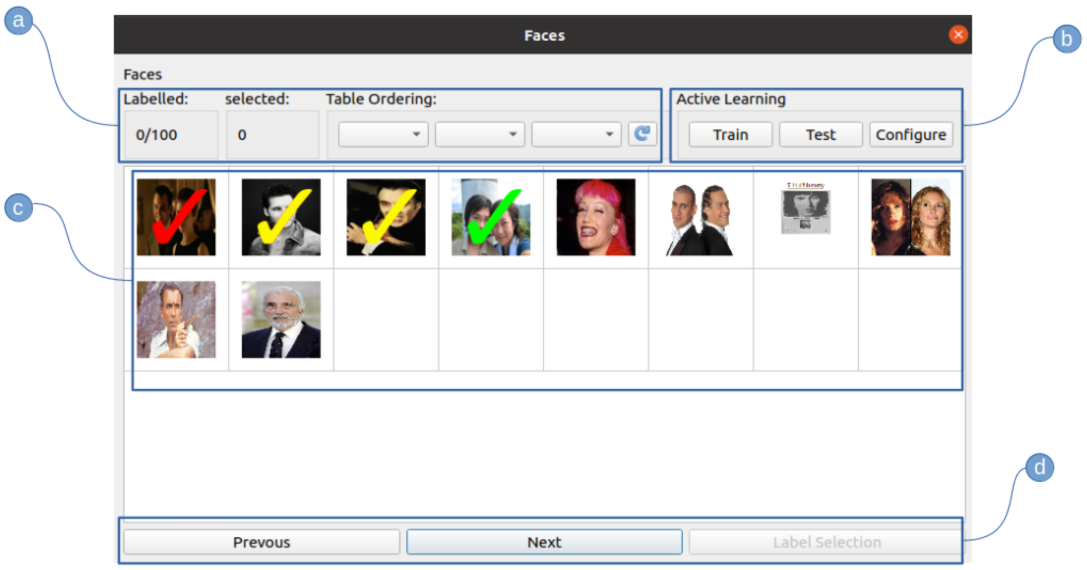


Figure 4.3: The image viewer window is where users will spend most of their time. Users can select samples to label, train or test models menu a gives basic information about the dataset. Menu b is for all the active learning tasks like training or testing a model. While in testing all samples are passed to the model. To see results users click on the image and can adjust labels if needed. Menu d are triggers for moving back and forth between the pages of the dataset. Users can also click label selection to move to the labeller window with all currently selected samples

The most important window of the application, the image viewer displays all the images of the data in a set of pages. The user can configure the image viewer with the use of a YAML config file. In figure 4.3 there are 10 images in 2 rows but this can be changed to any number, however greater this number is the slower the application performs. Users can select any number of samples to label by dragging the mouse of the samples or also individually select samples by holding the control key, just like any table-based application. The selections are stored across all pages, this also means that users will need to go back to a page to unselect a sample.

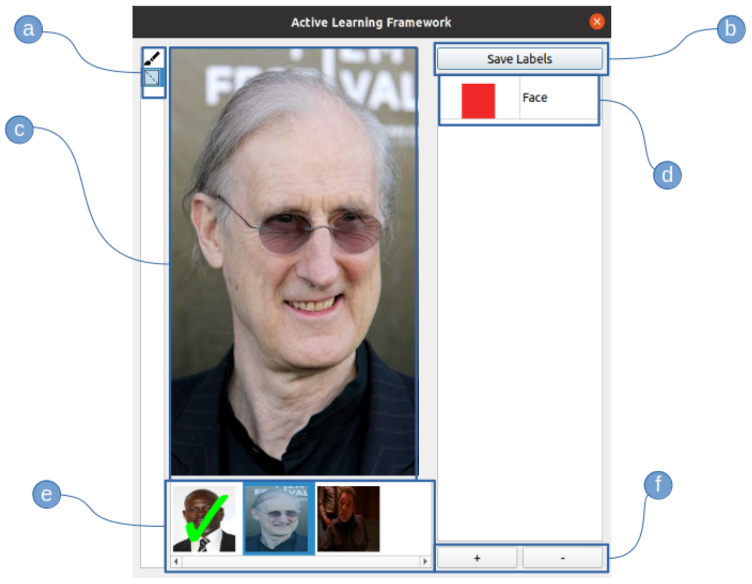


Figure 4.4: Users can select different brushes (a) and create boxes on the currently selected image (c). Different labels can be selected or edited (shown in d) as well as remove or add labels with the buttons from menu f. Once the users are happy they click "save label", highlighted as b.

Within the labeller (shown in figure 4.4) users can create, remove or edit labels. Each label is assigned a colour and a unique identifier (UID). During training sessions the UID is the given label, this allows users to change label names without removing labels from the object detector and thus losing previously learnt knowledge. Users can also change colour which updates all bounding boxes for that given label. There are 2 types of branches for labelling an object in an image:

- Paint Brush: Users can paint over the area and a bounding box is generated around that area.
- Point to Point Brush: Users drag their mouse from one point to another. As the distance between the 2 points grows so does the size of the bounding box.

Once a box has been created users can adjust its size by moving the anchors in the corners of the box. The position of the box can also be adjusted, once the user is happy they click "save label" and they automatically move to the next sample in the queue. If there are no more samples then we move back to the image viewer.

4.4 Object Detectors in the Labelling Application

For each profile in the application an XML file is generated to store both the ground truth boxes (labelled by a user) and the prediction box given by an object detector. The structure of the XML for a given image is as follows:

```
<?xml version="1.0" encoding="utf-8"?>

<image path = "/absolute/path/to/image">
<image_width>w</image_width>
```

```
<image_height>h</image_height>
<rect id: x>
<human_labelled>bool</human_labelled>
<label>label of object</label>
<colour>colour of object</colour>
<topLeft>(x,y)</topLeft>
<bottomRight>(x,y)</bottomRight>
<width>width of box</width>
<height>height of box</height>
</rect>
.
.
.
</image>
```

When starting a training session we iterate over the XML file, getting only the human labelled annotations and generating a TFRecord. If we have done some training sessions before the current session then we fine-tune otherwise we start with random weights on a fresh object detector. We config this by generating a configuration file. There are 3 types of learning supported by the object detection library:

- Classification: standard supervised learning on images where we label the whole image to a given label. We start from random weights
- Detection: The main type of learning for our task, where we localize and classify each object in an image. We start from random weights
- Fine-Tune: We load the weights of a given model path. The task of the object detecter depends on what the loaded model was trained on.

For our purposes only the first training session is defined as detection while all further sessions are fine-tuned, to ensure we keep the previously learned knowledge. TensorFlow provides many different types of object detectors in their zoo, however not all support detection or classification. For example, ResNet based CenterNet only support classification and fine-tuning while HourGlass based CenterNet supports detection and fine-tuning.

When users run an object detection test we load all our data into a TFRecord and load the model of the previous training session. The object detector produces several predictions and we apply the post-processing technique NMS to prune similar predictions [77]. After which we save the rest of the predictions to the project's XML file. If there is a prediction when loading the image in the labeller, this will also get displayed to the user. Allowing users to make adjustments if needed. In the image viewer there are 3 trick indicators for the type of annotation:

- Green Tick: Indicates that the image is human labelled and is consisted of ground truth.
- Yellow Tick: Indicates that the boxes are a prediction from an object detector and that the confidence in its prediction is good.
- Red Tick: Low confidence prediction from the object detector. These images are suggested to be labelled by a user.

4.5 Summary

The deliverable of this project is a labelling application where we apply object detectors in an active learning setting. The application has support for many file types and any size image. The general propose use of the application allows for testing on various types of imagery like surface texture detection or composite labelling datasets. With such an application we can perform active online learning experiments on object detectors.

We started this chapter by giving a general overview of the application and its two purposes of its use: labelling objects and training models based on those labelled objects. We then further brake these purposes down in their sections; Labelling Application (section 4.3) and Object Detectors in the Labelling Application (section 4.4). In labelling application, we demonstrated how a user creates a project, loads a dataset of choice and finally starts labelling. Our labeller consists of two types of brushes and how labels can be edited or removed. Useful for when specific labels can have multiple names. In section 4.4 we showed the process of applying the object detection environment to the application as well as the structure of how we store and use the annotations. Finally, we gave an overview of the datasets used for our experiments.

4.6 Datasets

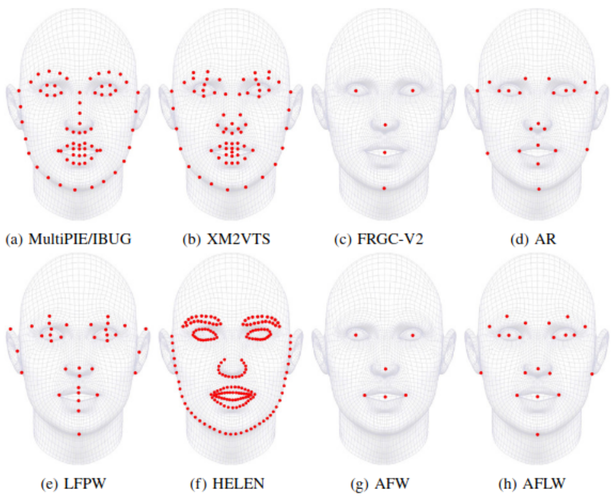


Figure 4.5: Layout of different keypoint locations on various face datasets [92]

Labelled Face Parts in the Wild (LFPW) and Severstal Steel Defect Detection are the 2 datasets being used for our experiments [4, 5]. LFPW was used for our composite label experiments where we train object detectors on faces to then transfer that knowledge to also detect facial features. The result is a detector capable of detecting faces and the composite labels of a face like eyes and noses. This is a common problem within steel defection where there are many classes of defects that also consist of subclasses of small defects. Unfortunately, we did not have access to a steel dataset with composite labels. The steel dataset is still very useful due to the challenge of each category of class looking simpler to each and the high inter-class variance due to defects consisting of a set of smaller defects.

Keypoint estimation datasets are especially useful for composite datasets as they typically have keypoints in many areas of an object. We can hand-select these points to build a box to represent a composite label. For example, in LFPW there are 4 key points for the nose and therefore we wrap a bounding box around those points. Figure 4.5 displays various different keypoint datasets for faces. We require a dataset that has enough detail to create multiple composite labels while also have many different facial expressions and ethnic groups. LFPW is a reasonable choice as it had enough information to create bounding boxes around different fascial features while - as the name suggests - have different faces in various contexts. LFPW is also open source while some other datasets like XM2VTS are not [93]. The authors of LFPW gathered 3000 faces by using text queries on websites such as Google, Flicker and Yahoo. The 35 ground truth key points were labelled by workers from Amazon Mechanical Turk (MTurk). While using this service is beneficial for labelling datasets, people can still make mistakes. The authors considered this and each keypoint was labelled by three different workers and they took the average position as the ground truth.

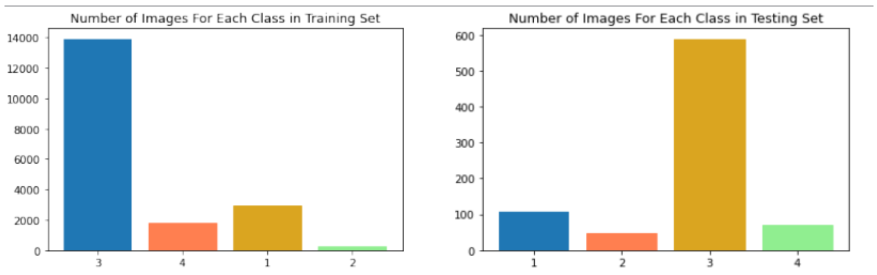


Figure 4.6: These charts show the number of defects for each class over both the training and testing sets. [5]

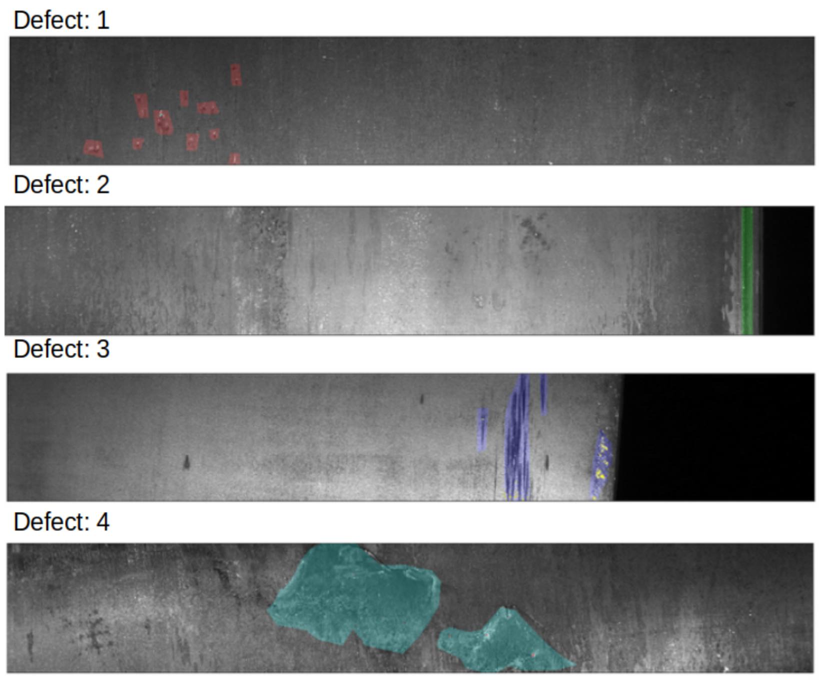


Figure 4.7: Visualization of each defect in the dataset. [5]

The only publicly available steel defect detection dataset that we could is the Severstal dataset. Originally used for a Kaggle competition [5], it consists of 12,568 steel panels. Where each panel can have one or more of the four types of defects, 5902 of which do not have any defects. Figure 4.6 shows the number of defects for each class over both training (left chart) and testing sets (right chart). We also show the 4 different types of defects by figure 4.7. Notice they are contours and not bound boxes and so for each contour, we generate a bounding box. This can be an issue for some defects where they are long strips (such as with defect 3) as opposed to large areas like defect 4. Defect 3 is also oversampled in the dataset but we will not be balancing the dataset or taking even samples. This is because during the production of the steel some defects are more common than others. This is then a natural bias of the dataset and we want our models to generalized over real-world production. We use both these datasets for use in our object detection experiments to not only test object detectors for their given domain but also in an active online learning setting.

Chapter 5

Project Management

Contents

5.1 Introduction

Planning our project and how we manage it are instrumental to the success of the project. We first create a project management methodology, where we apply human-centric design to an agile development cycle. Allowing us to fix any arising problems quickly with the added benefit of not needing to go back on previously completed stages or project milestones. Secondly, we will discuss the project schedule, which embraces the selected project methodology by its iterative nature. We will also set out our major and secondary milestone. By creating this schedule we perform an initial reflection of the planned stages and aspects to ensure that we innovate responsibly and that the proposal (that was discussed in chapter 4) follows a human-centric design. Finally, we will evaluate the project and initial proposal further by constructing and performing a risk assessment. This assessment is split into general, technical, personal and responsibly innovative risks. Where each risk is evaluated based on a likelihood and negative impact score.

5.2 Project Development Methodology

The underlining principle of this project is to have a human-centric focus while also ensuring that we research, develop, test and responsibly deploy our project solutions. This requires us to always be thinking about these principles at every stage of the project. Which naturally deduct to an iterative process and therefore an obvious approach is to an agile methodology, this includes but not limited to:

- Agile Scrum Methodology [94]
- Lean Software Development [95]

- Extreme Programming (XP) [96]
- Dynamic Systems Development Method(DSDM) [97]
- Feature Driven Development (FDD) [98]

Each of these methodologies has a focus on software development, thus any of these approaches will require some modification for our experiment stages. Deploying our experiments requires a rapid methodology as our next few experiments are completely dependent on the results of the previous experiment. As well as the experiments we also create an active online learning framework with a graphical user interface, which allows us to label samples and perform some training of different object detectors. It will be relatively linear with only miner feedback changes during these early stages of the project. Due to these challenges in both early and late stages, we will be using a modified version of extreme programming, this is because of its continuous delivery by its responsiveness to ever-changing situations in the project. This is useful for experiments stages but also adapts well to situations where there are few iterations such as in the active online learning framework stages.

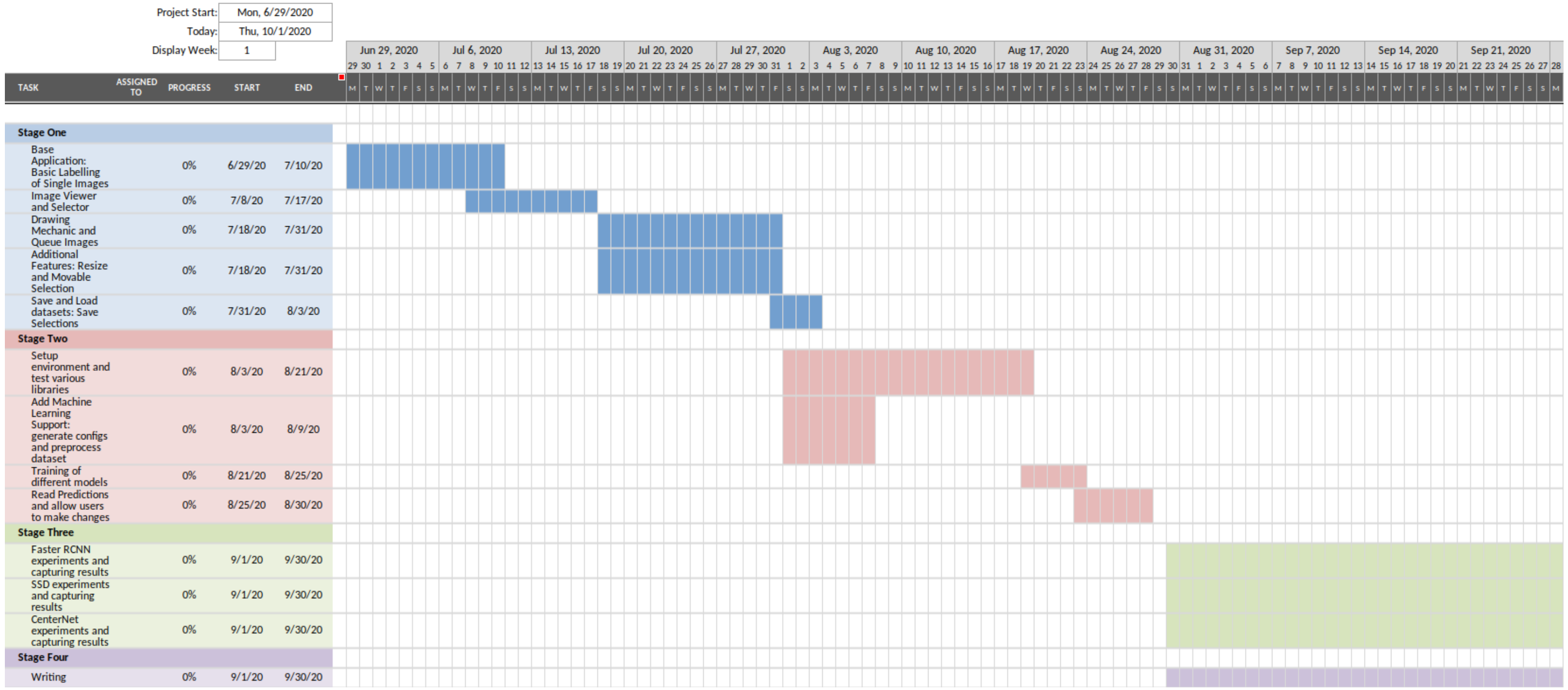
Originally introduced by Ken Beck in 1990, extreme programming allows us to create higher quality software with each iteration while also being adaptable to changes in the project as they arise. The approach is designed to have rapid feedback and response to that feedback as quickly as possible. We will have a weekly supervisory meeting where we will be able to adapt to the feedback in these meetings quicker than other methodologies. As this approach uses a test-driven principle we can change different parts of the experiment depending on previous experiments.

5.3 Schedule

Our schedule starts from 29th June to the 30th of September 2020 and is split amount 4 stages:

- Stage 1: We develop a labelling application that can support any image dataset.
- Stage 2: Setup a machine learning environment and the ability to train and test different object detectors.
- Stage 3: Online and Active learning experiments along with object detection experiments on composite label datasets.
- Stage 4: Writing the dissertation.

The end of each stage acts as our milestones. At which we reflect and evaluate our work to ensure we are on the track of the schedule and following the human-centred methodology. Each stage is further broken down into tasks that we must complete by their associated end date. Our schedule follows a linear path for the first 2 stages, then stages 3 and 4 are set in parallel. This is because we first build our labelling application and added machine learning support before we move into the iterative process of experimentation. To help in having better use of our time we will be preparing and writing the dissertation while training sessions are running.



Project Start:	Mon, 6/29/2020
Today:	Thu, 10/1/2020
Display Week:	1

TASK	ASSIGNED TO	PROGRESS	START	END
Stage One				
Base Application: Basic Labelling of Single Images		0%	6/29/20	7/10/20
Image Viewer and Selector		0%	7/8/20	7/17/20
Drawing Mechanic and Queue Images		0%	7/18/20	7/31/20
Additional Features: Resize and Movable Selection		0%	7/18/20	7/31/20
Save and Load datasets: Save Selections		0%	7/31/20	8/3/20
Stage Two				
Setup environment and test various libraries		0%	8/3/20	8/21/20
Add Machine Learning Support: generate configs and preprocess dataset		0%	8/3/20	8/9/20
Training of different models		0%	8/21/20	8/25/20
Read Predictions and allow users to make changes		0%	8/25/20	8/30/20
Stage Three				
Faster RCNN experiments and capturing results		0%	9/1/20	9/30/20
SSD experiments and capturing results		0%	9/1/20	9/30/20
CenterNet experiments and capturing results		0%	9/1/20	9/30/20
Stage Four				
Writing		0%	9/1/20	9/30/20

Figure 5.1: Project Schedule split into the four different stages

5.4 Risk Assessment

To conduct our risk assessment we first establish potential risks in 4 areas:

- Personal Risks: Risks that affect us individually and as a result harm the process of the project. Examples include sickness or failing to follow a schedule.
- Technical Risks: Risks with some form of technical elements such as badly labelled dataset or bugs in libraries.
- Project Risks: These are general risks or risks that do not fit into other risk categories. Risks like mismanagement of planning or resources.
- Responsible Innovation Risks: Identified areas of the project that could potentially be a risk of breaking one or more areas of responsible innovation.

For each potential risk, we state how it could affect the project and give a solution. We also give each risk a likelihood of happening and an impact on the project score. The resulting score is the average of the 2 scores. Finally, we order the risks in descending order of the results score. This shows us the more important risks that we should be especially aware of.

Legend:	
🟦	Personal Risks
🟨	Technical Risks
🟥	Project Risks
🟪	Responsible Innovation Risks

Risks	Risk Affects on Project	Solution	Likelihood	Impact	Result Score
Fixing bugs delay progress	Failing to meet project requiremen and milestones	Dedicate days for r bugs and prioritise	80	80	80
Failing to keep track of schedule	Failing to meet project requiremen and milestones	Follow a project de methodology and p	80	70	75
Sickness	Results in delays and potentially missing deadlines	Task regular breaks covid-19 guidelines	50	90	70
Library was found suitable due to faili of responsible innov	Results in failing one of the underlining principles of the project	Use backup library	50	90	70
PC hardware was r for the type of moc we want to train	Fails to complete a main requirement of the project	Use models that ar computationally sn or use Google Cola	70	70	70
Specific task was fo harder than what v	Results in delays and potentially missing of deadlines	Add extra time for than we originally t	65	70	67.5
Dataset is not ethic fails a key area of r	Not being able to use the dataset for the project, results in failing to train models	Have backup datasets in place	70	60	65

Not prioritising tasks	Lose scope of project, resulting in not completing important tasks	Use schedule and u based to do list	60	60	60
Unexpected anoma such as bugs in libr	Delays tasks which could mean some tasks are not completed	Add extra time to than otherwise tho	40	70	55
Loss of Data	Will require us to restart the project	Backup with an on or version control s	20	90	55
Specific task was fo not be human-cent	Affects project in a number of ways including strong AI bias to failing to meet project requirements	Evaluate each task being completed an on the task after co	20	70	55
Dataset Errors	Poor object detection performance	Use preprocessing a postprocessing tech	50	20	35

5.5 Summary

In this chapter, we presented how the project was managed and originally planned. We first discussed the need for a project development methodology by modifying extreme programming to work with our human-centric design. We then created a schedule with individual tasks grouped into stages. The end of these stages is the major milestones of the project. Finally, we performed a risk assessment to ensure that if anything was to happen that could negatively impact our plan, there was a backup plan. We also ordered our risk assessment by importance to make special attention to risks that have the largest impact or the most likely to occur.

Chapter 6

Evaluation

Contents

6.1 Introduction

In this chapter, we investigate the use of object detectors in an active setting. We use a composite label and highly intra-class datasets for these experiments and all run on an Nvidia GeForce GTX 1070. The chapter is split into our active learning experiments and object detection experiments, for each given experiment we show our results and evaluate the performance. Due to the many challenges of combining different learning scenarios with the added complexity of focusing on object detectors, we choose to split the problem topic into their sections.

6.2 Online and Active Learning Experiments

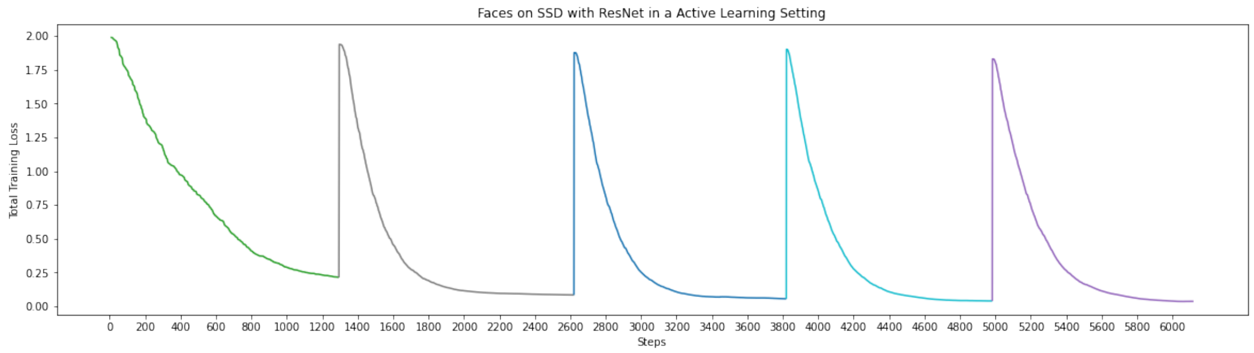


Figure 6.1: Each colour represents a different training session. For each new session we add 10 new samples to the training pool. These samples are selected based on the average losses confidence class.

We first perform an active online learning experiment where random samples are given to the object to simulate real-life data retrieval. The random nature is sightly wrong to real-life production lines like in steel. We do not have access to data on its distribution through time, which can be found in a live scenario, therefore it is impossible for us to know. After a few of the training sessions, we applied an active element to the sessions, where sample classes that had the average lowest confidence score. As we already had the labels made before the start of the training sessions, our sampling acted as the oracle. This reduces the training time as we do not need to hand label or constantly need to monitor the training. Figure 6.2 illustrates the training loss of our first experiment. For each training session, we add 10 new samples to the training set. We select these 10 new samples based on the average confidence score of the classes. The figure shows the first 5 out of 10 sessions. As we get new samples added the pool, a large spike representing a large loss. Over time we should see the spike decrease, however, in our experiments, the spike is roughly the same. There are a few hypotheses for this:

- We used the confidence based on softmax. As we found in the literature softmax tends to be overconfident. Therefore we may be given the model wrong samples to be training on
- Object detectors are large/complex models and require a lot more data and time than we provided in the experiments. This could result in overfitting to the data.

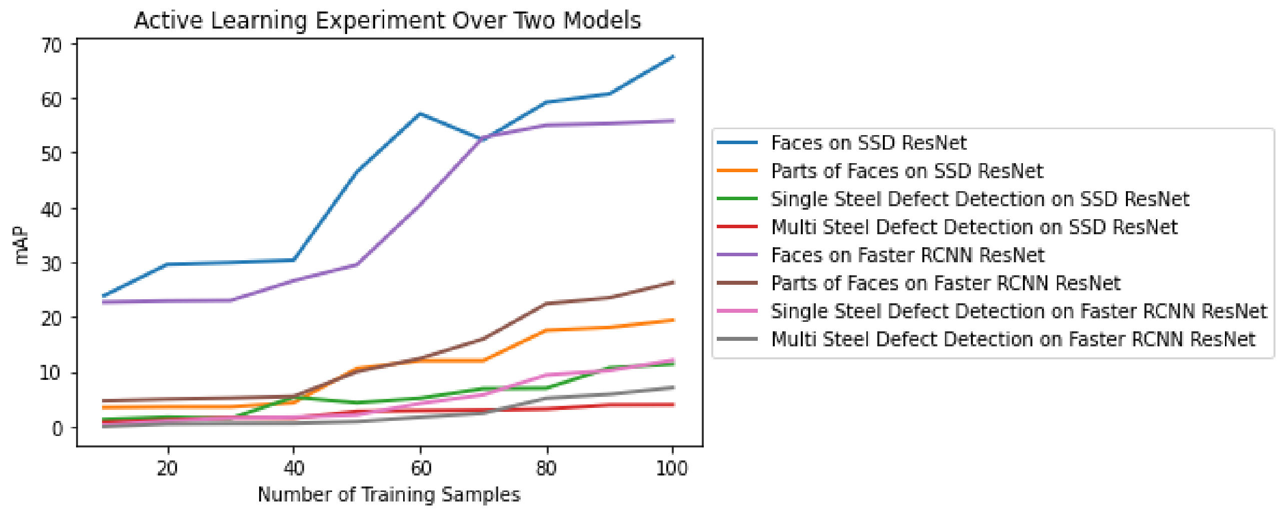


Figure 6.2: Overview of the active learning experiments with 2 different models and 4 types of training data.

Figure 6.2 we show all 8 of our active learning experiments. We used 2 meta-architectures: SSD and Faster RCNN. We choose to test a 1 stage and 2 stage model to test the trade-off between accuracy over speed of training time. We also broke up the datasets into a single class and a multi-class variant. For the LFPW dataset, we generated the bounding boxes of a face by getting the largest and smallest points, which represents the bottom right corner and the top left corner of a box. For parts of the face, we generated boxes based around the eyes, noses and mouths. The single class steel defect dataset consisted of only the first class of defects. As shown in table 6.1 our best performing object detector was the SSD with ResNet on the single class of face, which outperformed even the Faster RCNN with the same configurations and training data. This could be an anomaly but this model also performed well in the object detection experiments. We assume that due to how large these 2 stage detectors are, it was underfitting to a simple face detection task.

Dataset	Model	mAP	mAP^{50}	Training Time (Hours)
Faces	SSD with ResNet	67.48	67.84	11
Parts of Faces	SSD with ResNet	19.42	26.97	12
Single Class Steel Defect Detection	SSD with ResNet	11.42	14.81	28
Multi Class Steel Defect Detection	SSD with ResNet	3.97	9.5	28
Faces	Faster RCNN with ResNet	55.79	56.21	26
Parts of Faces	Faster RCNN with ResNet	26.3	30.38	26
Single Class Steel Defect Detection	Faster RCNN with ResNet	12.1	12.86	32
Multi Class Steel Defect Detection	Faster RCNN with ResNet	7.10	8.24	32

Table 6.1: Final results of the active learning experiments.

6.3 Object Detector Experiments

All of the steel defect detection experiments performed poorly overall. However, the detections are still accurate in their placement but most predictions had very low confidence. As confidence is used to remove some predictions, many accurate ones where removed. This can be edited by lowing the threshold but we wanted to keep all our experiments consistent with each other for a fairer comparison among methods.

Training time is heavily important in online learning scenarios as there will be a consistent flow of data and thus the bottleneck is the training time. Many approaches stop production or retrieval of data to train. This has the obvious drawback of some finance loses, there is also the drawback of not learning and therefore adapting to current situations that may naturally occur. For example if a steel mile is not cleaned then dust and other artefacts go onto the steel. The object detector would need to adapt to this in real time. The detector would only learn to adapt to these artefacts when training starts. Therefore it is desired to have a model that can train as quickly as possible. In the object detection experiments we focus on testing composite labels and highly intra class datasets. We found the best preforming detector to be EfficientDet D1 with a resolution of 640x640 and an mAP score of 71.13. Its training time was also one of the best due to how computationally efficient the model is. Our fastest performing detector was the SSD with MobileNet, this was not a surprise due to the feature extractor having excellent speed complexity.

Dataset	Meta-Architecture	Base Model	Resolution	mAP	Training Time (Hours)
Faces	Faster RCNN	ResNet50	640x640	92.75	2
Parts of Faces	Faster RCNN	ResNet50	640x640	35.54	2
Single-Class Steel Defect Detection	Faster RCNN	ResNet101	1024x1024	15.13	7
Multi-Class Steel Defect Detection	Faster RCNN	ResNet101	1024x1024	9.97	7
Single-Class Steel Defect Detection	Faster RCNN	ResNet50	640x640	13.21	4
Multi-Class Steel Defect Detection	Faster RCNN	ResNet50	640x640	8.17	4
Faces	SSD	MobileNet	640x640	94.91	1
Parts of Faces	SSD	MobileNet	640x640	34.13	1
Single-Class Steel Defect Detection	SSD	MobileNet	640x640	5.71	2
Multi-Class Steel Defect Detection	SSD	MobileNet	640x640	3.94	2
Faces	SSD	ResNet50	640x640	93.71	3
Parts of Faces	SSD	ResNet50	640x640	32.54	3
Single-Class Steel Defect Detection	SSD	ResNet50	640x640	7.18	4
Multi-Class Steel Defect Detection	SSD	ResNet50	640x640	5.71	4
Faces	EfficientDet	D1	640x640	71.13	3
Parts of Faces	EfficientDet	D1	640X640	64.59	3
Single-Class Steel Defect Detection	EfficientDet	D1	640x640	11.71	4
Multi-Class Steel Defect Detection	EfficientDet	D1	640x640	5.13	4

Table 6.2: Object Detection Experiment Results with 4 types of training sets 3 types of object detectors.

Chapter 7

Conclusions and Future Work

Contents

7.1 Conclusions

In this thesis, we present an active learning framework that can use any imagery-based datasets. Users can create dataset-specific profiles and label any number of images at once. Labels can be edited without the need to configure object detectors. Users can select any of the object detectors provided by TensorFlow's model zoo. They can also configure different parts of the application like the text of all buttons or the number of images being displayed by the image viewer in a given page.

The second part of the project was to explore object detectors in an active-online learning setting on 2 datasets; "Severstal: Steel Defect Defection" and "Labelled Face Parts in the Wild" (LFPW). LFPW is a keypoint estimation dataset which has 35 labelled points around faces in many different backgrounds and facial expressions. We use these labelled points to generate ground truth boxes for eyes, noses and mouths. We then use this dataset to predict composite labels on various object detectors like SSD and Faster RCNN. The performance and training time of these methods are very dependent on the feature extractor. We test our models on ResNet and MobileNet to measure the trade-off between training time and accuracy, an important attribute of online learning.

We first started this thesis by introducing the background literature by exploring deep learning. We first introduced neural networks and built towards modern object detectors. Many of which use a feature extractor to learn a representation of the data. We explore the most notable extractors and object detectors. Following the literature, we introduce the human-centric design and later adjust a software development methodology with this new design. In Chapter 4 we introduced the main deliverable of the project as well as the datasets used for our experiments. In which we found that both SSD and Faster RCNN with ResNet50 are the best for our composite label experiments (1% difference). While our quickest trainable model was EfficientDet D1 with an mAP of 64.59.

7.2 Future Work

The current approach to steel defect detection is to use object detection, this is where we get regions of the image and make predictions on those regions. However, the defects found in steel does not fit well in a bounding box. This meant that our ground truth boxes have many empty regions. This resulted in a more difficult classification task as we are - in general - pointing to an area where there is a defect as opposed to directly feeding the defect pixels into our model. An instance segmentation approach seems more reasonable as we can classify the individual pixels to get a natural outline of the defect. This requires a larger and more complex labelling process, due to the requirement of labelling each pixel. We would need to adjust our application for pixel-wise labelling as we already have pixel brush mechanics.
All object detectors in TensorFlow's zoo uses non-maximum suppression (NMS). This is a problem for composite labelling where you have labels overlapping other labels. There is no option to remove this without major editing of the source code. Therefore the construction of our models would be required. Another area to explore could involve decreasing the training time with key point estimation methods. Training time friendly networks seem promising in there training speeds while still able to have similar performance to 1 stage detectors [81]

Bibliography

[1] Hatcher W, Yu W. A Survey of Deep Learning: Platforms, Applications and Emerging Research Trends. IEEE Access. 2018 04;PP:1–1.

[2] Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, editors. Advances in Neural Information Processing Systems 25. Curran Associates, Inc.; 2012. p. 1097–1105. Available from: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[3] Brown TB, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, et al. Language Models are Few-Shot Learners. arXiv. 2020;.

[4] Belhumeur PN, Jacobs DW, Kriegman DJ, Kumar N. Localizing Parts of Faces Using a Consensus of Exemplars. 24th IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2011;.

[5] Severstal. Severstal, editor. Severstal: Steel Defect Detection. kaggle; 2019. Available from: `"https://www.kaggle.com/c/severstal-steel-defect-detection"`.

[6] Girshick R, Donahue J, Darrell T, Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition; 2014. p. 580–587.

[7] Lemarechal C. Lemarechal C, editor. Cauchy and the Gradient Method; 1847.

[8] Andrychowicz M, Denil M, Gomez S, Hoffman MW, Pfau D, Schaul T, et al. Learning to learn by gradient descent by gradient descent. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. Curran Associates Inc.; 2016. p. 39883996.

[9] Wolpert DH, Macready WG. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation. 1997;.

[10] Lloyd S. Least squares quantization in PCM. IEEE Transactions on Information Theory. 1982;.

[11] Li Z, Hoiem D. Learning without Forgetting. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2016;p. 2935–2947.

[12] Koul A, Ganju S, Kasam M. Practical Deep Learning for Cloud, Mobile and Edge: Real-World AI and Computer Vision Projects Using Python, Keras and TensorFlow. O'Reilly Media, Incorporated; 2019. Available from: `https://www.oreilly.com/library/view/practical-deep-learning/9781492034858/`.

[13] Bengio Y. Bengio Y, editor. Public Lecture: Towards Deep Learning 2.0. IEEE WCCI 2020; 2020. Available from: https://2020.wcci-virtual.org/presentation/public-lecture/public-lecture-towards-deep-learning-20.

[14] Lillicrap TP, Santoro A, Marris L, Akerman CJ, Hinton G. Backpropagation and the brain. Nature Reviews Neuroscience. 2020;Available from: https://doi.org/10.1038/s41583-020-0277-3.

[15] Bengio Y, Bastien F, Bergeron A, BoulangerLewandowski N, Breuel T, Chherawala Y, et al. Deep Learners Benefit More from Out-of-Distribution Examples. Journal of Machine Learning Research - Proceedings Track. 2011 11–13 Apr;15:164–172. Available from: http://proceedings.mlr.press/v15/bengio11b.html.

[16] Angluin D. Queries and Concept Learning. Machine Language. 1988;p. 319–342. Available from: https://doi.org/10.1023/A:1022821128753.

[17] Angluin D. Queries Revisited. Theoretical Computer Science. 2004;313.

[18] Wang L, Hu X, Yuan B, Lu J. Active learning via query synthesis and nearest neighbour search. Neurocomputing. 2015;p. 426–434.

[19] Lang K, Baum E. Query learning can work poorly when a human oracle is used; 1992. .

[20] Chen SF, Rosenfeld R. A survey of smoothing techniques for ME models. IEEE Transactions on Speech and Audio Processing. 2000;.

[21] Dagan I, Engelson SP. Committee-Based Sampling For Training Probabilistic Classifiers. In: In Proceedings of the Twelfth International Conference on Machine Learning. Morgan Kaufmann; 1995. p. 150–157.

[22] Lewis DD, Gale WA. Lewis DD, editor. A Sequential Algorithm for Training Text Classifiers. Springer, London; 1994.

[23] Hoi SCH, Sahoo D, Lu J, Zhao P. Online Learning: A Comprehensive Survey. arXiv. 2018;.

[24] Goodfellow I, Bengio Y, Courville A. Deep Learning. The MIT Press; 2016.

[25] Schmidhuber J. Deep learning in neural networks: An overview. Neural Networks. 2015 Jan;61:85117. Available from: http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[26] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;Available from: https://doi.org/10.1038/nature14539.

[27] FISHER RA. THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS. Annals of Eugenics. 1936;7(2):179–188. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x.

[28] Bengio Y, Courville A, Vincent P. Representation Learning: A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2013;35(8):1798–1828.

[29] Marblestone AH, Wayne G, Kording KP. Toward an Integration of Deep Learning and Neuroscience. Frontiers in Computational Neuroscience. 2016;10:94. Available from: https://www.frontiersin.org/article/10.3389/fncom.2016.00094.

[30] Burnham K, Anderson D. Model Selection and Multimodel Inference. A Practical Information-theoretic Approach. 2004 01;.

[31] Rosenblatt F. A Perceiving and Recognizing Automaton. Cornell Aeronautical Laboratory. 1957;.

[32] Narayan S. The generalized sigmoid activation function: Competitive supervised learning. Information Sciences. 1997;99(1):69 – 82. Available from: http://www.sciencedirect.com/science/article/pii/S0020025596002009.

[33] Nwankpa C, Ijomah W, Gachagan A, Marshall S. Nwankpa C, editor. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. arXiv; 2018.

[34] Agarap AF. Agarap AF, editor. Deep Learning using Rectified Linear Units (ReLU). arXiv; 2018.

[35] Maas AL. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In: in ICML Workshop on Deep Learning for Audio, Speeh and Language Processing; 2013. .

[36] Boltzmann L. Studien ber das Gleichgewicht der lebendigen Kraft zwischen bewegten materiellen Punkten; 1868. .

[37] Gibbs JW. Elementary Principles in Statistical Mechanics: Developed with Especial Reference to the Rational Foundation of Thermodynamics. Cambridge Library Collection - Mathematics. Cambridge University Press; 2010.

[38] Gao B, Pavel L. On the Properties of the Softmax Function with Application in Game Theory and Reinforcement Learning; 2017.

[39] Kleppmann M. Designing Data-Intensive Applications: The big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly; 2017.

[40] Dalal N, Triggs B. Histograms of oriented gradients for human detection. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). vol. 1; 2005. p. 886–893 vol. 1.

[41] Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;86(11):2278–2324.

[42] Edwards M. Representation Learning in Irregular Domains; 2018.

[43] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition; 2014.

[44] Johnson J. CNN Architecutres. University of Michigan; 2019.

[45] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, et al. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV). 2015;115(3):211–252.

[46] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition; 2015.

[47] Krizhevsky A. Learning Multiple Layers of Features from Tiny Images. University of Toronto. 2012 05;.

[48] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, et al.. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications; 2017.

[49] Xie S, Girshick R, Dollr P, Tu Z, He K. Aggregated Residual Transformations for Deep Neural Networks; 2016.

[50] Chollet F. Xception: Deep Learning with Depthwise Separable Convolutions; 2016.

[51] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift; 2015.

[52] Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR. Improving neural networks by preventing co-adaptation of feature detectors; 2012.

[53] Wang D, Shang Y. A new active labeling method for deep learning. In: 2014 International Joint Conference on Neural Networks (IJCNN); 2014. p. 112–119.

[54] Wang K, Zhang D, Li Y, Zhang R, Lin L. Cost-Effective Active Learning for Deep Image Classification. IEEE Transactions on Circuits and Systems for Video Technology. 2017 Dec;27(12):25912600. Available from: `http://dx.doi.org/10.1109/TCSVT.2016.2589879`.

[55] Ren P, Xiao Y, Chang X, Huang PY, Li Z, Chen X, et al.. A Survey of Deep Active Learning; 2020.

[56] Hossain HMS, Roy N. Active Deep Learning for Activity Recognition with Context Aware Annotator Selection. KDD '19. New York, NY, USA: Association for Computing Machinery; 2019. p. 18621870. Available from: `https://doi.org/10.1145/3292500.3330688`.

[57] Simoni O, Budnik M, Avrithis Y, Gravier G. Rethinking deep active learning: Using unlabeled data at model training; 2019.

[58] Houlsby N, Huszr F, Ghahramani Z, Lengyel M. Bayesian Active Learning for Classification and Preference Learning; 2011.

[59] Holub A, Perona P, Burl M. Entropy-based active learning for object recognition; 2008. p. 1–8.

[60] Joshi AJ, Porikli F, Papanikolopoulos N. Multi-class active learning for image classification. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition; 2009. p. 2372–2379.

[61] Kapoor A, Grauman K, Urtasun R, Darrell T. Gaussian Processes for Object Categorization. International Journal of Computer Vision; 2010. .

[62] Wang RJ, Li X, Ling CX. Pelee: A Real-Time Object Detection System on Mobile Devices; 2018.

[63] Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC. MobileNetV2: Inverted Residuals and Linear Bottlenecks; 2018.

[64] Jiao L, Zhang F, Liu F, Yang S, Li L, Feng Z, et al. A Survey of Deep Learning-Based Object Detection. IEEE Access. 2019;7:128837128868. Available from: `http://dx.doi.org/10.1109/ACCESS.2019.2939201`.

[65] Girshick R. Fast R-CNN; 2015.

[66] Ren S, He K, Girshick R, Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks; 2015.

[67] Everingham M, Gool L, Williams CK, Winn J, Zisserman A. The Pascal Visual Object Classes (VOC) Challenge. USA: Kluwer Academic Publishers; 2010. Available from: `https://doi.org/10.1007/s11263-009-0275-4`.

[68] Uijlings J, Sande K, Gevers T, Smeulders A. Selective Search for Object Recognition. International Journal of Computer Vision. 2013 09;104:154–171.

[69] Redmon J, Divvala S, Girshick R, Farhadi A. You Only Look Once: Unified, Real-Time Object Detection; 2015.

[70] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, et al. SSD: Single Shot MultiBox Detector. Lecture Notes in Computer Science. 2016;p. 2137. Available from: `http://dx.doi.org/10.1007/978-3-319-46448-0_2`.

[71] Ting KM. In: Sammut C, Webb GI, editors. Confusion Matrix. Boston, MA: Springer US; 2017. p. 260–260. Available from: `https://doi.org/10.1007/978-1-4899-7687-1_50`.

[72] Olson DL, Delen D. Advanced Data Mining Techniques. 1st ed. Springer Publishing Company, Incorporated; 2008.

[73] Goutte C, Gaussier" E. A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. In: Advances in Information Retrieval. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005. p. 345–359.

[74] Padilla R, Netto SL, da Silva EAB. A Survey on Performance Metrics for Object-Detection Algorithms. In: 2020 International Conference on Systems, Signals and Image Processing (IWSSIP); 2020. p. 237–242.

[75] Lin TY, Maire M, Belongie S, Bourdev L, Girshick R, Hays J, et al.. Microsoft COCO: Common Objects in Context; 2014.

[76] Everingham M, Gool L, Williams CK, Winn J, Zisserman A. The Pascal Visual Object Classes (VOC) Challenge. USA: Kluwer Academic Publishers; 2010. Available from: `https://doi.org/10.1007/s11263-009-0275-4`.

[77] Rothe R, Guillaumin M, Van Gool L. Non-Maximum Suppression for Object Detection by Passing Messages between Windows. vol. 9003; 2015. .

[78] Newell A, Yang K, Deng J. Stacked Hourglass Networks for Human Pose Estimation; 2016.

[79] Law H, Deng J. CornerNet: Detecting Objects as Paired Keypoints; 2018.

[80] Duan K, Bai S, Xie L, Qi H, Huang Q, Tian Q. CenterNet: Keypoint Triplets for Object Detection; 2019.

[81] Liu Z, Zheng T, Xu G, Yang Z, Liu H, Cai D. Training-Time-Friendly Network for Real-Time Object Detection; 2019.

[82] Commission E. Options for Strengthening Responsible Research and Innovation - Report of the Expert Group on the State of Art in Europe on Responsible Research and Innovation; 2013.

[83] von Schomberg R. 3. In: A Vision of Responsible Research and Innovation. John Wiley and Sons Ltd; 2013. p. 51–74. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118551424.ch3.

[84] Orbit. The Keys of Responsible Research and Innovation; 2020. Available from: https://www.orbit-rri.org/resources/keys-of-rri/.

[85] Voigt P, Bussche Avd. The EU General Data Protection Regulation (GDPR): A Practical Guide. 1st ed. Springer Publishing Company, Incorporated; 2017.

[86] Ergonomics of human-system interaction - Part 210: Human-centred design for interactive systems. International Organization for Standardization; 2019.

[87] Google. TensorFlow Object Detection API; 2020. Available from: "https://github.com/tensorflow/models/tree/master/research/object_detection".

[88] Google. TensorFlow 2 Detection Model Zoo; 2020. Available from: "https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md".

[89] Google. Protocol Buffers; 2020. Available from: "https://developers.google.com/protocol-buffers".

[90] Qt. Qt; 2020. Available from: "https://www.qt.io/".

[91] Bradski G. The OpenCV Library. Dr Dobb's Journal of Software Tools. 2000;.

[92] Sagonas C, Antonakos E, Tzimiropoulos G, Zafeiriou S, Pantic M. 300 Faces In-The-Wild Challenge: database and results. Image and Vision Computing. 2016 01;47.

[93] Messer K, Matas J, Kittler J, Luettin J, Maitre G. XM2VTSDB: The Extended M2VTS Database. In: Second International Conference on Audio and Video-based Biometric Person Authentication; 1999. Available from: 'http://www.ee.surrey.ac.uk/CVSSP/xm2vtsdb/'.

[94] Carvalho B, Henrique C, Mello C. Scrum agile product development method -literature review, analysis and classification. Product: Management and Development. 2011 01;9:39–49.

[95] Poppendieck M, Poppendieck T. Lean Software Development: An Agile Toolkit. Addison-Wesley; 2003. Available from: https://www.safaribooksonline.com/library/view/ean-software-development/0321150783/.

[96] Beck K, Andres C. Extreme Programming Explained: Embrace Change (2nd Edition). Addison-Wesley Professional; 2004.

[97] Agile Business Consortium. The DSDM Agile Project Framework (2014 Onwards); 2014. Available from: https://www.agilebusiness.org/resources/dsdm-handbooks/the-dsdm-agile-project-framework-2014-onwards.

[98] Palmer SR, Felsing M. A Practical Guide to Feature-Driven Development. Pearson Education; 2001.