Tampere University

Umer Hameed

DIGITIZING INDUSTRIAL TECHNICAL LAYOUTS USING COMPUTER VISION AND MACHINE LEARNING

Engineering and Natural Sciences
Master of Science Thesis
December 2021

ABSTRACT

Umer Hameed: Digitizing Industrial Technical Layouts using Computer Vision and Machine Learning
Master of Science Thesis
Tampere University
Automation Engineering
December 2021

Recently there have been rapid advancements in the field of image recognition. Researchers are looking for better ways to utilize the machine learning capabilities and advancements in computer vision to extract data from digital images. Industrial systems are moving towards automation and the use of computers and machines is ever increasing. An important aspect of ensuring personnel and product safety is the design and planning of industrial layouts. Technical layouts of various aspects of an industry need to be designed and implemented. These are traditionally not readable by machines and would require some form of digitizing.

This thesis focuses on the process of digitizing technical layouts into computer readable formats and useful data. The thesis proposes an approach to creating custom datasets that will be used to train a chosen model for object detection. The training and tuning of the model for object detection on custom dataset is implemented using a Jupyter notebook platform.

The proposed method was successfully implemented and tested on electrical layout. The results show that the model detects objects with high accuracy and speed. The thesis provides a guideline to develop similar techniques for various technical layouts.

Keywords: Computer Vision, Digitizing, Machine Learning, object detection

The originality of this thesis has been checked using the Turnitin Originality Check service.

PREFACE

I would like to take this opportunity to thank the people who have motivated me throughout my journey in this master's program. Firstly, I am deeply thankful to my parents for their unconditional love and support. I wouldn't be where I am without them. Their prayers kept me motivated in the toughest of times. I am also thankful to my brothers and sister for being so supportive.

I would like to express my deep gratitude to my supervisor Luis Gonzalez for his guidance and supervision despite his busy schedule. I would especially like to thank Professor Jose Martinez Lastra for giving me the opportunity to work on this thesis. Their guidance and supervision were essential to completing this thesis.

I would also like to thank my friends Osama, Jehanzeb, and Ismail for always motivating me. A special thank you to Zaighum Sultan for being a true friend, for which I am forever grateful.

Tampere, 11 December 2021

Umer Hameed

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
BiFPN	Bi-directional Feature Pyramid Network
CNN	Convolutional Neural Network
COCO	Common Objects in Context
ConvNet	Convolutional Network
CSV	Comma Separated Values
FPN	Feature Pyramid Network
GB	Gigabyte
GPU	Graphics processing unit
GUI	Graphical User Interface
mAP	Mean Average Precision
MBE	Mean Bias Error
MEA	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
MSLE	Mean Squared Logarithmic Error
NAS-FPN	Neural Architecture Search Feature Pyramid Network
NLP	Natural Language Processing
RAM	Random Access Memory
ReLU	Rectified Linear Unit
TFX	TensorFlow Extended
UI	User Interface
XLA	Accelerated Linear Algebra
XML	Extensible Markup Language

1 INTRODUCTION

1.1 Background

Recently there have been rapid advancements in the field of image recognition. Researchers are looking for better ways to utilize the machine learning capabilities and advancements in computer vision to extract data from digital images. There are a wide variety of practical applications for the data extracted from these techniques. For example, smartphone devices use it for face recognition and self-driving cars can utilize it for autonomous driving.

Nowadays, industrial systems are moving towards automation and the use of computers and machines is ever increasing. An important aspect of ensuring personnel and product safety is the design and planning of industrial layouts. Technical layouts of various aspects of an industry need to be designed and implemented. An industry may have architectural layouts, electrical layouts, piping and instrumentation diagrams etc. These layouts are usually complex and mainly used by humans. They are traditionally not readable by machines and would require some form of digitizing.

1.2 Problem statement

This thesis focuses on the process of digitizing technical layouts into computer readable formats and useful data. The primary focus is on electrical layouts and can be extended to other technical layouts in future works. To solve this problem, machine learning and computer vision will be used to train an object detection model to detect symbols in a layout.

1.3 Objective and scope

The objective of the thesis is to find a solution to digitize industrial technical layouts. The research and case study aims to answer the following questions:

- How can an object detection model be trained to detect objects and symbols from technical layouts?
- Which models should be selected for object detection on layout images?

- How fast and how accurately can the detection be output into machine readable format?

The scope of the thesis does not include all possible technical layouts. This thesis builds the foundation for digitizing any technical layouts, but the demonstration is done using only electrical layouts. The approach that is proposed in this thesis can be applied to other types of technical layouts using the basis outlined.

1.4 Thesis outline

The thesis structure consists of 6 chapters. Chapter 1 introduces the thesis and outlines the scope of work. Chapter 2 discusses the theoretical background relevant to the thesis to provide an understanding of the work. Chapter 3 discusses the approach and justifies the technologies and tools used. Chapter 4 provides the implementation to the approach. Chapter 5 provides the tests done during implementation and the results to verify the implementation. Chapter 6 concludes the thesis and discusses future work.

2 THEORETICAL BACKGROUND

This section discusses the theoretical concepts and state of the art. It also provides an overview of related technologies that are relevant to this study.

2.1 Machine Learning

Machine learning is the investigation of giving devices the capacity to learn and generate their own programs in order to make them more human-like in their behaviour and choices [1]. This can be achieved with the minimum amount of human intervention. It also facilitates in the automation and efficient development of data analysis models. Many sectors depend on massive amounts of data to improve their processes and they makes smart decisions [2]. There are three main elements in ML system.

- The model is a system that makes predictions.
- The elements are known as parameters the model examines these parameters in making predictions.
- To get the results of the study, the learner matches the predictions, with parameters and the model.

Figure 1. *Machine learning*

2.1.1 Introduction

Machine Learning is used to automate many tasks. Historical data is used as an input. ML is concerned with the creation of computer programs that can collect data and learn on their own. Statistics are used by machine-learning systems to discover patterns in large volumes of data. It generally provides better, more correct information in identify-

ing profitable possibilities. Computational techniques are used by machine learning algorithms to obtain information directly from data, rather than depending on a model based on a predetermined equation. Machine learning also helps in the analysis of large amounts of data [3].

Machine learning has seven steps.

- Data collection
- Data preparation
- Model selection
- Training
- Evaluating
- Tuning hyperparameters
- Predictions

Machine learning is becoming more important as the amount and diversity of data growing with the computational power, as well as the provision of high-speed Internet. The digitalization elements allow for the rapid and automatic development of models that can evaluate extremely massive and complicated data sets efficiently and precisely. Machine learning has a number of practical applications that lead to real-world commercial outcomes. Many sectors are now building more powerful models that can analyse more and more complicated data while providing faster and more precise findings. Machine learning automates operations that would usually require the assistance of a live agent, such as resetting a password or verifying an account balance [4].

There are three major types of ML as illustrated in figure 2.

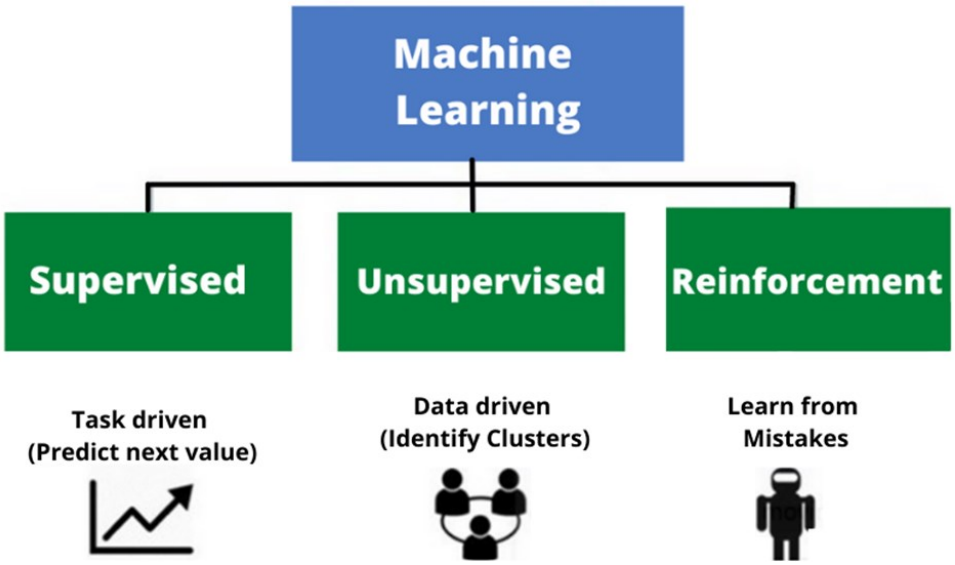


Figure 2. *Types of Machine Learning*

2.1.2 Supervised Machine Learning

Supervised machine learning creates a model that produces results which are based on evidence. The supervisor is the output in the data for a particular set of inputs, and the learner agent is the machine learning (ML) algorithm or models. The learning algorithm's objective is to predict how a particular set of inputs will result in a given level of output. The computers are trained using well-labelled training data to predict the output. Initially, the ML management takes the inputs and guesses the outputs which are unpredictable. At the output end, the supervisor reveals the inaccuracy in prediction, and the learning agent is guided to minimize the error once more. It requires time and technical knowledge from a team of highly qualified data scientists to successfully create correct supervised ML models [5].

The methods in Supervised Machine learning are:

- All supervised learning methods begin with a data matrix as input. Make a dataset for training.
- Create a feature vector from the input object. Some characteristics that represent the object are included in the feature vector
- Choose the learning algorithm you want to use and test it on the training data.
- Run the algorithm on the training set of data. Verification sets, which are a subset of training datasets, sometimes are required as control parameters.
- Use the test dataset to assess the model's accuracy, and then implement the model to forecast the results of unanticipated data

There are two types of supervised ML Algorithms: Regression and Classification.

Regression

If it is a connection between input and output variables, regression procedures are applied. It is applied in the forecasting of continuous data, such as temperature. Using training sets, the regression model predicts a single output value [6].

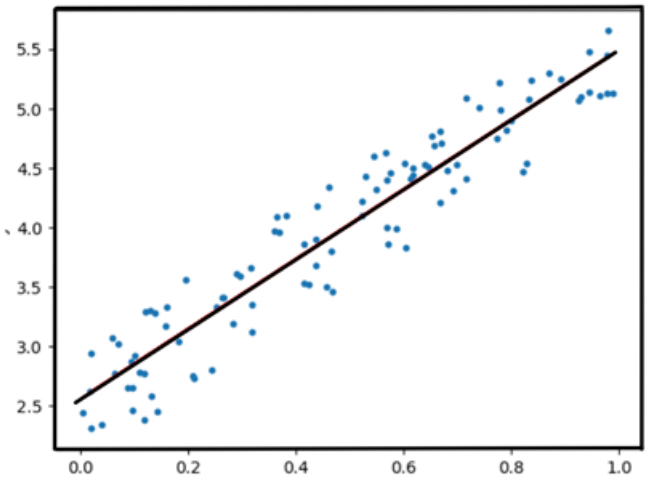


Figure 3. *Predicting numerical values from several data points*

Regression finds and guesses ongoing outcomes by identifying patterns in the sample data. Quantities, numbers, relationships, and groups are all understood through this algorithm. This approach works excellent for product and stock forecasting [6].

There are two types of regression, Linear and Logistic regression.

- Linear regression: In this technique it is assumed that the input and output have a linear relationship. The input vector is called an independent variable, while the output variable is considered a dependent variable. It applies the function, analyses the result, and shows the output as a continuous number [7].
- Logistic regression: The algorithms in logistic regression predict discrete values for the collection of independent variables on the list. The method forecasts the probability of new data, resulting in an output that ranges from 0 to 1 [8].

Classification

Classification is the process of categorizing output into different groupings. Binary classification is when an algorithm attempts to divide data into two separate classes.

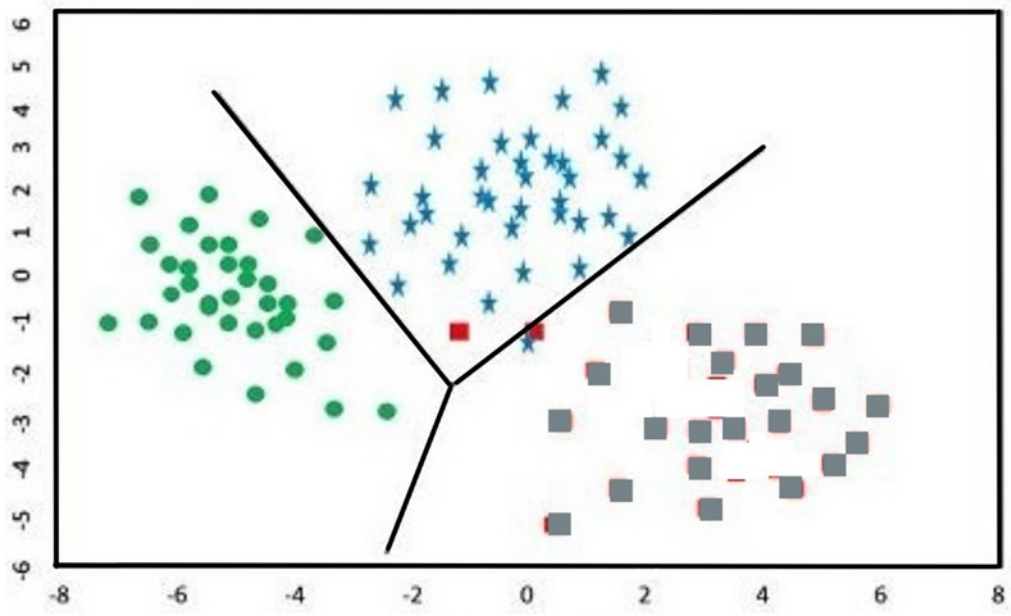


Figure 4. *Making different grouping of elements using classification*

Based on previous data, the input data is labelled. These algorithms have been specifically developed to recognize specific categories of things. It's straightforward to process and analyse the labelled sample data, forecast the weather, and identify images [9].

2.1.3 Unsupervised Machine Learning

A dataset is presented without labelling in unsupervised learning, and a model learns beneficial aspects of the dataset's structure. Unsupervised Learning is a sort of machine learning wherein the algorithms are given data with no labels. Unsupervised Machine Learning identifies patterns in a set of data without the use of pre-defined labels or categories.

Since we have little or no knowledge about the data, unsupervised learning algorithms are more complex than supervised learning algorithms. Unsupervised machine learning is particularly beneficial for data information, such as classifying potential consumers into categories based on common characteristics for more positive benefit or detecting the characteristics that distinguish one set of customers from another. Grouping comparable cases together, dimension reduction, are common unsupervised learning problems [10].

Unsupervised learning is utilized for more complex problems because there is no labelled input data in unsupervised learning. This is often used before supervised learning to uncover characteristics in data exploration and classify data into groups. Unsu-

pervised learning can sometimes be preferred over supervised learning for a number of reasons. Here are a few of the benefits. There are two types of unsupervised learning: Clustering and Association.

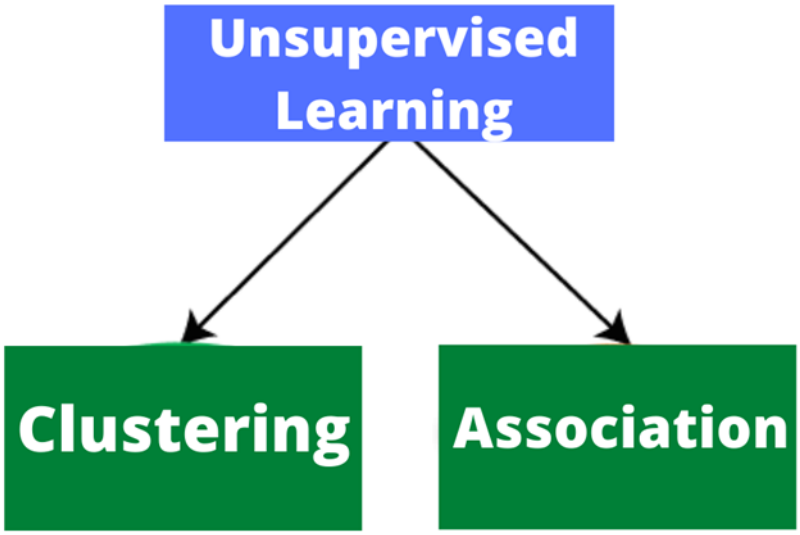


Figure 5. *Types of unsupervised learning.*

Clustering is a type of unsupervised machine learning which use cluster analysis, and clustering techniques to examine data and identify underlying patterns or categories, is the most frequent unsupervised learning method. Clustering helps to divide a dataset into subgroups based on similarities. Cluster analysis, on the other hand, typically magnifies the connection among groups and fails to consider data points as individuality. Association mining detects groups of things in collection that commonly occur together [11].

2.1.4 Reinforced Machine Learning

Reinforcement learning is a kind of dynamic programming that uses a progressive discipline system to train algorithms in ML. It is a deep learning technique that allows users to optimize a portion of the total reward.

Unlike supervised learning, Reinforcement Learning allows the agent to learn on its own through feedback. Because no labelled data is available, the agent must rely only on its own experiences to learn [12].

Reinforcement Learning Terminologies:

- Agent: It is considered as an entity capable of exploring the environment in order to obtain a reward

- An agent's environment (e): It is a situation that they must deal with which is random in nature. It's also known as an agent's actions in the environment
- Reward (r): It is either an immediate reward given to an agent when the agent executes a certain action or task, or it is a report given to the agent from the environment
- State (s): The current status as reported by the environment is referred to as the state.
- Policy (π): A policy is a technique that the agent uses to choose the next act based on the present condition.
- Value (v): Value is the predicted long-term return, discounted from the short-term gain.
- Q value, often known as action value (q): It is a term which is quite comparable to value. The only distinction between the two is that the current action takes an additional parameter.

Transfer Learning

Traditional ML models take longer to achieve optimal performance than transfer learning models. That's because models that use previously trained algorithms' knowledge currently know what the characteristics are. A model learned on one activity is reused on a second, similar work as an optimization that allows for faster modelling development on the second activity. Transfer learning can achieve much better performance than training with a small amount of data when applied to a new task [13].

Transfer learning is so frequently used that training a model for image or natural language processing problems from scratch is quite rare. Transfer learning has several advantages, the most prominent of which are shorter training times, better neural network performance (in most cases), and the lack of a huge amount of data. Neural networks are a type of machine learning technique that uses numerous hidden units and non-linear training algorithm to describe complicated patterns in datasets. Iterative refinement techniques such as gradient descent is used to train neural networks. A lot of data is normally required to train a neural network from scratch, but accessibility to that data isn't always feasible. This is where transfer learning comes in very handy [14].

2.2 Deep Learning

Neural network architecture is used in most deep learning approaches. Deep learning is commonly described to as "deep neural networks" because of this. A practical ex-

ample of deep learning is the enabling of voice recognition in electronics used by people.

Deep learning necessitates a significant amount of processing power. Deep learning creates an "artificial neural network" which can learn by itself and make even make smart decisions by layering algorithms. The inspiration of Artificial neural networks methods comes from the human brain, learning from enormous volumes of data in deep learning. Deep learning systems learn by recognizing complex patterns in the data they receive. The networks can construct several degrees of abstraction to describe the data by constructing computer simulations that are made up of many processing layers. Neural networks are made up of layers upon layers of variables that change to the qualities of the data they're taught on and may perform tasks like picture classification and audio to text conversion [15]. An illustration of a typical dep learning network is given in figure 6 below.

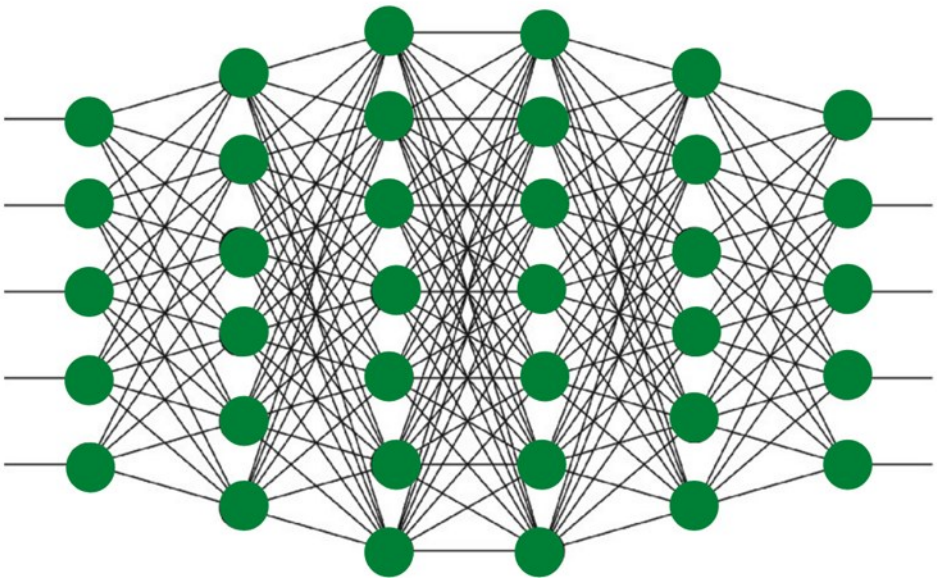


Figure 6. *Deep learning layers*

2.2.1 Image Classification

The input to neural networks passes via underlying layers of nodes. These nodes process the data and propagate the outcomes to the subsequent layers. The main neural networks used during computer vision and as image classifiers are CNN, or Convolutional Neural Networks. Convolutional layers in these networks acquire relevant details information of characteristics by moving a kernel or filters across the input image. This continues until it reaches an output layer, at which point the machine responds. Convo-

lutional neural networks are commonly used in deep learning for classification purposes. It can consist of hidden layers. Machines can recognize and extract information from photographs using deep learning. As a result, programmers do not have to manually enter these filters [16].

2.2.2 Object Detection

A branch of computer vision, object recognition is an automated method for locating interesting things in an image concerning the background. Because object detection and picture recognition are frequently confounded. Object detection is a supervised machine learning issue, which means it uses labelled samples to train your models. Every image in the training data must be supplemented by a file containing the boundary and class information for the objects it comprises. Object detection annotations can be created using a variety of open-source technologies [17, 18]. An example of a typical object detection use is depicted in figure 7.

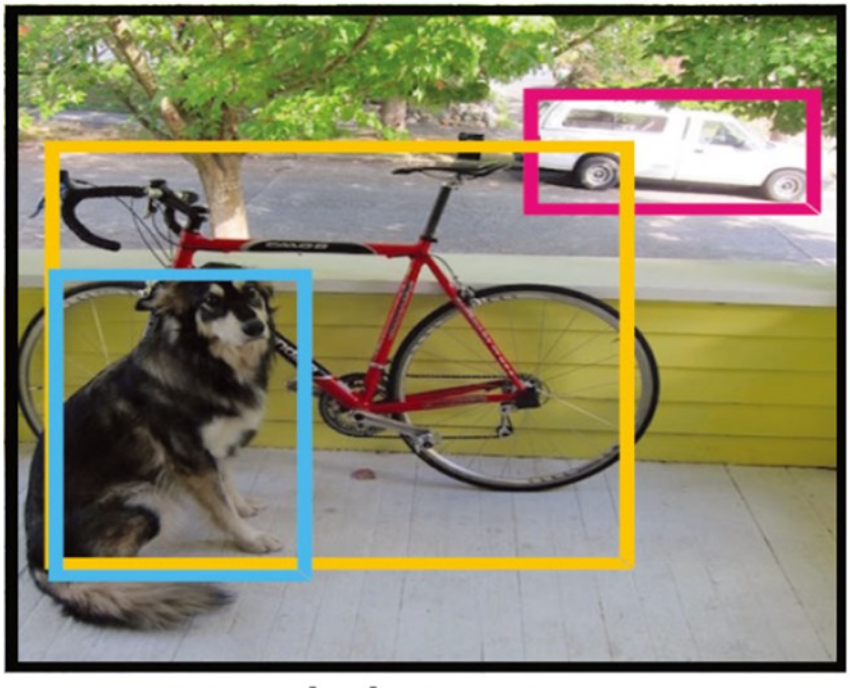


Figure 7. *Object detection example.*

To get started with object detection utilizing deep learning, there are two main approaches.

- Make a customized object detector and train it. To build a customized object detector from the ground up, first, the architecture of the network needs to be built that can learn the characteristics of the objects of interest. To train CNN, there is a need for a big amount of labelled data. A custom object detector can pro-

duce incredible results. However, manually configuring the layers and parameters in the CNN, which takes a long time and a lot of datasets [19].

- Use an object detector that has been pre-trained. Transfer learning is used in many deep learning object recognition processes, and it is possible to start with a pre-trained model. This method can offer faster results because object analysers have historically been trained on hundreds, if not thousands, of images.

2.2.3 Datasets

A dataset comprises of many different segments of data, it could be used to train an algorithm with the purpose of identifying predictable patterns within the dataset. The discipline of machine learning relies heavily on datasets. The difficult task in deep learning is acquiring the proper data in the right format, which has nothing to do with neural nets. Gathering or finding the data that correlates with the results you wish to forecast is what getting the appropriate data entails [20].

The selection of appropriate dataset is one of the most critical steps for solving any machine learning based problem. The type of dataset to be selected is based upon the type of problem being solved (i.e., supervised, or unsupervised learning problems). Usually, the datasets in machine learning are categorized in two major categories.

- Labelled dataset.

  Labelled data in machine learning refers to the type of data, that incorporates labels or ground-truths corresponding to each instance of dataset. These datasets are usually employed for solving supervised machine learning based problems (i.e., classification or regression) [21].

- Unlabelled dataset.

  Unlabelled datasets in machine learning refers to the type of data that only consists of the instance of datasets without their corresponding labels. To deal with such datasets, unsupervised machine learning based algorithms are being employed i.e., clustering. In such algorithms, the similarity or patterns in datasets are analysed for their categorization [21].

Additionally, machine learning works with two types of data: training and testing. A dataset may need to be split into these two categories.

Training

Accessibility to high-quality training data is required for AI and machine learning models. When a training set is executed through a neural network, it trains it how to priori-

ties various characteristics and modify their parameters based on their chances of reducing errors in output. Those values, also referred to as factors, will be stored in tensors. These are collectively referred to as the model, as they represent a model of the data they are training on.

Testing

Testing data helps to evaluate the progress of algorithm's training and optimize it to get better results [22]. To put it another way, some part of the data is used to analyse whether the training is being done correctly or not.

2.2.4 Dataset Augmentation

Applying basic adjustments to your existing dataset, such as adding noise, interpreting the image, and changing the dimension of each image, all contribute to the size and variety of training dataset. The process of performing basic and complicated modifications to data, such as flipping or style transfer, might help to meet the growing demands of Deep Learning models. Deep learning uses geometric modifications, flipping, colour alteration, cropping, rotating, noise injecting, and randomized erasing to optimize images.

The ImageDataGenerator class in the Keras deep learning toolkit supports image data augmentation. For training data, computer vision programs use typical data augmentation approaches. For picture identification and natural language processing, there are both basic and complex data supplementation methods [23]. These include the following.

- Adversarial training.
- GANs (generative adversarial networks).
- Neural style transfer.
- Reinforcement learning models.

Data augmentation techniques are commonly used in image recognition and natural language processing (NLP) models. Data augmentation is also used in the medical imaging sector to perform changes to images and provide variety to datasets.

2.2.5 Image Segmentation

Image segmentation is an important area in computer vision that has a lot of research behind it, both in terms of image processing algorithms and learning methodologies. It allows us to mark the image into different parts based on certain characteristics. This

makes it simpler for inspection. The technique of segmenting one picture into several segments is known as image segmentation [24]. A typical example of image segmentation is illustrated in the figure below.

Figure 8. *Image segmentation example.*

Face detection is used with image segmentation in face analysis to decide which sections of a video should be targeted on to assess age, gender, and emotions facial emotion recognition. The image segmentation approach is highly beneficial for the study of diverse image modalities in the medical areas because it produces robust and high-accuracy results [25].

Image Segmentation methods can be divided in two broad categories. Approach based and Technique based classifications.

Approach Based Classification

In this method, the primary criterion is the way that an algorithm can recognize objects [26]. This can involve grouping similar pixels and differentiating them from another group of different pixels. The two ways this can be achieved are listed below.

- The Region-based method uses area combining, area extending, and region growing to identify similar objects.
- The boundary-based method is used to detect the boundaries between objects so that they can be differentiated.

Technique Based Classification

Technique-Based Classification has three categories.

- Structural Techniques.

  These algorithms are based to have access to the image's structural information. Pixel intensities, distribution, scatter plot, pixel resolution, colours distributions, and other important data are all included [27].

- Stochastic Techniques.

  These algorithms are based on the latest graph theoretical approach for selecting cuts in graphs that uses pairwise proximity of components. Rather than the architecture of the necessary region of the image, these methods demand information about the image's continuous adjacent pixels [28].

- Hybrid Techniques.

  Both stochastic as well as structural approaches are used in these algorithms.

Types of image segmentation techniques

Thresholding is an image segmentation technique in which the pixels of an image are altered to make the picture simpler and faster. Thresholding is the process of converting a colour or grayscale images into binary images, which is just black and white. It's a

straightforward method of image segmentation which is used for creating binary or multi-colour image as input a threshold value to the source image's intensity values [29].

Edge-based segmentation algorithms use numerous discontinuities in grey level, colouring, pattern, intensity, brightness, contrasts, and other factors to locate edges in an image. Image segmentation includes edge detection. The accuracy of identifying significant edges is critical to the success of several image processing and computer vision jobs. It's one of the methods for identifying digital image intensity discontinuity. These operators assist in detecting edge discontinuities and, as a result, identifying edge bounds [30].

In region-based approach all pixels that relate to an object are gathered and labelled to show that they correspond to one area in the region-based method. Segmentation is the term for this procedure. Pixels are allocated to regions based on a criterion that sets them apart from the current frame [31].

Watershed algorithms are generally employed in image analysis for object segmentation, or distinguishing different items in a single image. This enables object counting or additional evaluation of the split components [32].

Clustering is the process of detecting closeness in data so that they can be categorized and split. The process of segmentation is the method of placing objects into different commonalities [33].

A deep learning technique named semantic segmentation uses convolutional neural networks (CNNs) to correlate each pixel of an image with a class name. Self-vehicles, industrial automation, diagnostic imaging, and satellite picture processing are some of the applications for semantic segmentation [34].

2.3 Convolutional Neural Network

Convolutional Neural network is one of the well-known deep learning algorithms, which is also known as ConvNet or CNN. Its primarily used is for the classification or differentiation of images which belons to different categories or classes. To do so it assigns different importance values (i.e., biases and weights) to the numerous objects found in the image that is given as input to the network. In traditional machine learning classification algorithms, several pre-processing techniques need to be applied on the input images to get better results, however in case of CNNs its necessity is much lower. In addition to this, in traditional methods the task of features extraction is done using hand-engineered filters, while in case of CNNs they incorporate the learning ability of these filters in an automated way. The connectivity of neurons in CNN architecture is

arranged similar to the neuronal pattern in brain. Due to the high accuracy and adorable performance of ConvNet for complex images (i.e., images with pixel dependencies), it has replaced traditional multi-layered perceptron. By employing different filters, these networks could easily detect the temporal and spatial dependencies of pixels in a digital image. By reusing network weights and due to the less usage of network parameters, this architecture performs better training over image datasets i.e., it better understands the sophisticated details of images. In case of images with large number of dimensions, CNNs transform them into such a shape that is easier to get processed, without any reduction of features, which is one of the most crucial tasks for getting better results. A general architecture of CNN has been depicted in figure 9 below. This architecture incorporates several types of layers which include Convolutional, ReLu, Max-pooling, and fully connected layers [35].

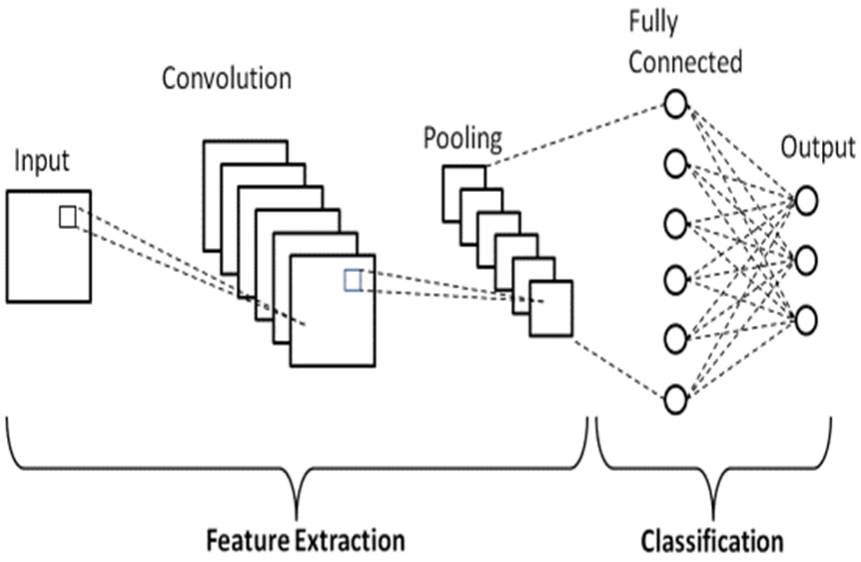


Figure 9. *CNN Architecture*

2.3.1 Convolutional Layer

One of the major constituents of ConvNet is its convolutional layer, by which this network is named as Convolutional Neural Network. The prime purpose of this layer is to execute a task named as “convolution”. Convolution is basically a linear operation, in which an array of weights, which is also known as kernel or filter, gets multiplied with the input image. The filter is smaller than the input image, so that filter gets multiplied multiple times at different positions of the input image. This repetitive application of filter over input image, results into a 2-Dimensional output array also known as “feature map”.

Traditionally, computer vision experts design different types of filters or kernels (e.g., horizontal, or vertical filters) for the extraction of feature maps from the input images (i.e., to perform image analysis task). However, in case of CNNs the values or weights of these kernels is automatically learned during the training of network. The main aim of employing convolution operation in CNN is the extraction of high-level and low-level features from images. There is no limitation the on number of convolutional layers in CNNs i.e., it could be start from one to as many as you want the depth of network to be. In general, the first convolutional layer of CNN is supposed to extract low-level feature details of input images (i.e., orientation, colour, and edges etc.), while the later layers are intended to extract high-level features details from images (i.e., objects).

An example of the calculations performed while applying convolutional operation has been depicted in the figure below [36].

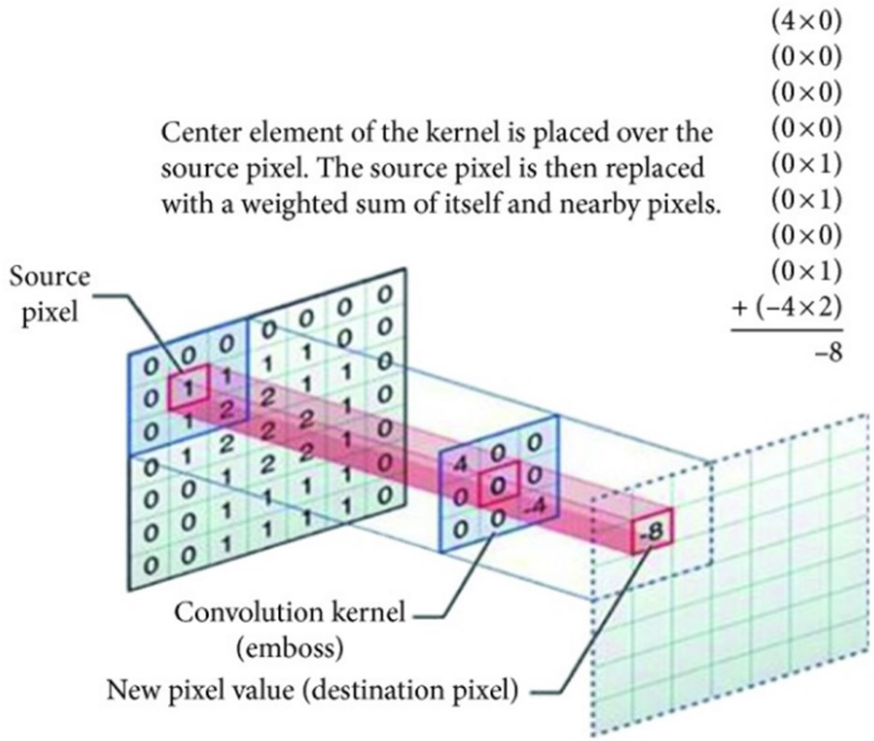


Figure 10. *Calculation in Convolution Operation.*

2.3.2 Pooling Layer

Another major constituent of CNN architecture is its pooling layer. The major aim of employing this layer in the network is the reduction of spatial size of the feature maps extracted by employing convolutional operation. Pooling layer basically reduces the dimensions of input images and hence assists in minimizing the computational cost required to process the input image dataset. In addition to this, pooling layer also assists in the extraction of dominant features from the input image, which are both invariant to

rotation and position, thus helping in the efficient training of CNN. There are two types of operations which are employed in pooling layers: Average pooling and Max-Pooling. Max pooling operation also acts as a noise suppressor in addition to the dimensionality reducer, as it assists in image denoising by discarding the noisy activations from the input image. Therefore, max pooling operation is preferred compared to the average pooling operation, due to its better performance [37]. A computational example of average and max pooling operations has been depicted in the figure below.

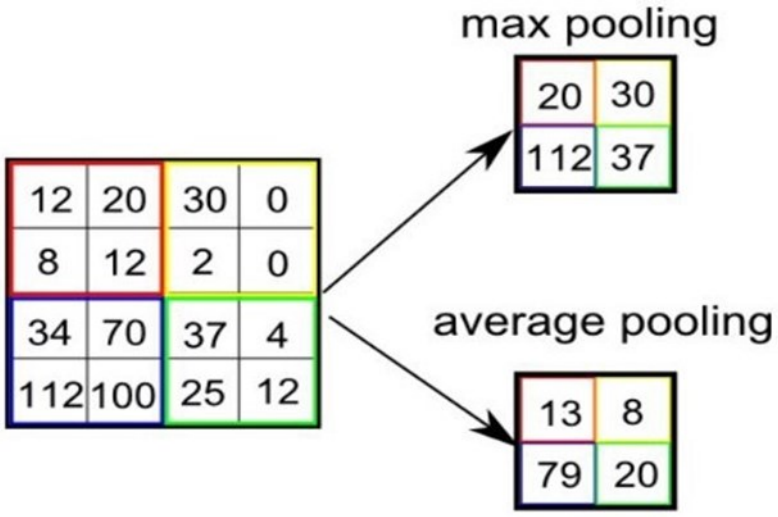


Figure 11. *Types of Pooling techniques.*

Collectively, both convolutional and pooling layers of Conv-Net form the i-th layer of the network. For getting the more low-level details of the complex input images, the number of these blocks can be increased in the network. However, this increment in number of layers increase the computational cost.

After performing the above-mentioned process, the final output is finally flattened and fed to the regular neural network (i.e., fully connected layer) to perform classification task.

2.3.3 Fully Connected Layer

For learning non-linear combination from the high-level features space extracted from convolutional blocks of Conv-Net, one of the cheapest ways is the application of fully connected layers. For feeding the feature maps to such fully connected layers, it is necessary to flatten them into a column vector first. This flattened feature vector is subsequently fed to the subsequent one or more fully connected layers, which are also named as “dense layers” [38]. In a single dense layer, each input is linked with every output, while employing a learnable weight. The major task of the dense layers is to map the down-sampled flattened feature maps to the network’s final output (i.e., class

probabilities). The number of neurons in the final dense layer are typically same as that of number of classes in the dataset. An example of the calculations in a typical fully connected layer has been depicted in the figure below [39].

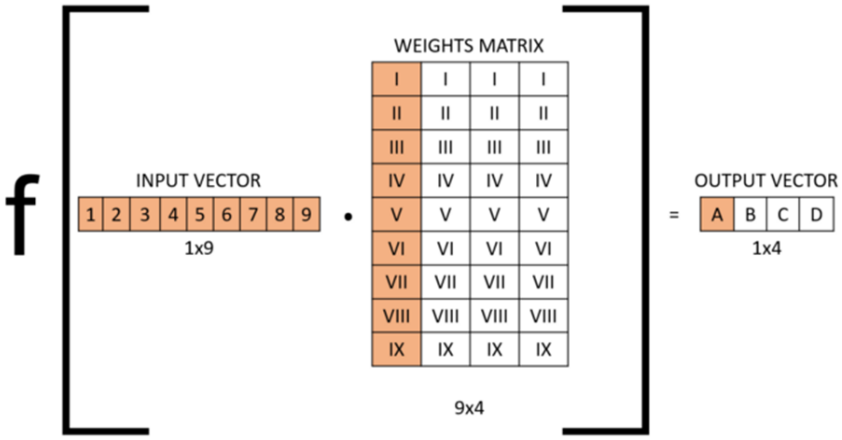


Figure 12. *Calculation in Fully Connected Layer.*

2.3.4 ReLU Layer

Rectified Linear Unit (ReLU) is a layer that is applied on Conv-Nets, after the implication of each convolution layer. The main aim of this layer is the introduction of non-linearity in the linear output of convolutional layers.

The basic function that is applied in ReLU layer is: f(x) = max (0, x) over all the values of convolutional layer's feature maps [40]. Basically, all the negative values are transformed to 0 by employing this activation function [41].

2.3.5 Loss Layer

The last fully connected layer of deep learning architecture (i.e., Conv-Net) is followed by a loss layer. The prime motive of this layer is the adjustment of all network weights. As we know, random weights are assigned to different layers of CNN (i.e., convolution and max-pooling layers), before starting the training process. During the training process, the task of loss layer is to check the difference between the prediction made by the final fully connected layer of model and the actual prediction value, while the motive behind doing this task is to minimize the calculated difference between the network's prediction and actual value or goal value. To minimize this difference, the loss layer adjusts the weights of CNN layers (i.e., convolutional, and fully connected) during each training iteration or epoch. Different categories of loss functions are there, which are being utilized by the network's loss layer. Thus, while configuring or designing any deep learning network, one the most crucial tasks is the selection of appropriate loss

function [42]. The selection of these loss functions typically depends upon the type of machine learning problem being solved i.e., in case of regression problem, the loss functions that are usually employed includes the following.

- Mean Absolute Error (MEA)

  For the model's loss metric estimation, MAE calculates the absolute difference between the actual value or class and the predicted value by the model. The formula for the calculation of this metric can be expressed as:

$$Mean\ Absolute\ Error\ (MAE) = \frac{\sum_{j=1}^{t} |c_j - p_j|}{t}$$

  Where t is the total number of instances in dataset, c_j is the actual class label of jth instance and p_j is the model's predicted value for jth instance of dataset [43].

- Mean Squared Error (MSE)

  This metric is also named as L2 loss, while it calculates the average or mean of squared differences between the class label and model's predicted values for the estimation of model's loss. The formula for the calculation of this metric can be expressed as [43]:

$$Mean\ Squared\ Error\ (MSE) = \frac{i}{t}\sum_{j=1}^{t}(c_j - p_j)^2$$

- Mean Squared Logarithmic Error (MSLE)

  The calculation of MSLE is done same as that of MSE, except instead of actual class label and actual model's predicted value their natural logarithms are used. The formula for the calculation of this metric can be expressed as [43]:

$$Mean\ Squared\ Logarithmic\ Error\ (MSLE) = \frac{i}{t}\sum_{j=1}^{t}(\log(c_j) - \log(p_j))^2$$

- Mean Bias Error (MBE)

  For the estimation of model's bias (i.e., underestimation or overestimation of model's parameters) this metric is calculated. This metric is calculated same as that of MAE except actual difference of class label and model's predicted value is utilized instead of their absolute difference. The formula for the calculation of this metric can be expressed as [43]:

$$Mean\ Bias\ Error\ (MBE) = \frac{\sum_{j=1}^{t} c_j - p_j}{t}$$

2.4 TensorFlow

TensorFlow is one of the most popular open-source platforms for solving machine learning problems. It provides a complete set of flexible tools, resources and libraries that assists developers and researchers in the deployment of different Machine learning based applications. This core machine learning library enable researchers in building and training of different machine learning and deep learning-based models, while using Keras application programming interface (API) [44]. In addition to this, in case of big machine learning based projects, the process of model's training could be made faster and flexible by employing the distribution strategy of this API. In distribution strategy, the training task is basically distributed over different hardware's without making any alteration in the definition of model. Regardless to the programming language or platform that is being utilized by a developer, this library assists in the direct training and deployment of machine learning models. TensorFlow Extended (TFX) is designed for the developers who want to deploy complete machine learning pipeline, while for running machine learning applications on edge devices and mobiles TensorFlow Lite has been designed. Moreover, for dealing with java-script based environments TensorFlow.js has been introduced [44].

TensorFlow provide developers several well-known ready to use datasets. These different categories datasets are being utilized for solving different nature of machine learning problems i.e., audio, image, text and video-based datasets that are used for tasks like classification, regression, speech recognition and object detection etc.

The TensorFlow library also provides a number of pretrained transfer learning-based CNN models, that could be directly utilized for performing tasks like image classification and segmentation. Based on the nature of problem being solved these networks could be customized i.e., by removing the classification layers for using these pretrained models as features extractor or by fine-tuning these models over new datasets. Some of these pretrained models include EfficientNet, MobileNet, ResNet, NasNet and InceptionNet etc [45].

For performing machine learning based experiments, TensorBoard provides tooling and visualization facility e.g., using this tool different classification and regression metrics i.e., accuracy and loss could be tracked and visualized, The layer structure of implemented deep models could be visualized, the parameters of models i.e., their

weights and biases could be visualized in the form of graphs as they vary with time, different dataset instances i.e., text, images and audios could be visualized, profiling of different TensorFlow based programs can be done [46].

Accelerated Linear Algebra tool or XLA is a domain specific compiler that is intended for solving problems related to linear algebra. This compiler tool assists in accelerating the task of TensorFlow based models without any change in its source code [47].

3 APPROACH

This section described the approach used for the implementation of digitizing industrial layouts. It also provides an overview and justification of the technologies and tools used to implement the system.

3.1 Datasets

The first part of any object detection process is to establish a dataset of images. These sets of images can be used to train a model to detect the required objects. For digitizing electrical layouts, it is necessary to detect the various electrical symbols. An example of such symbols is given in figure 13.

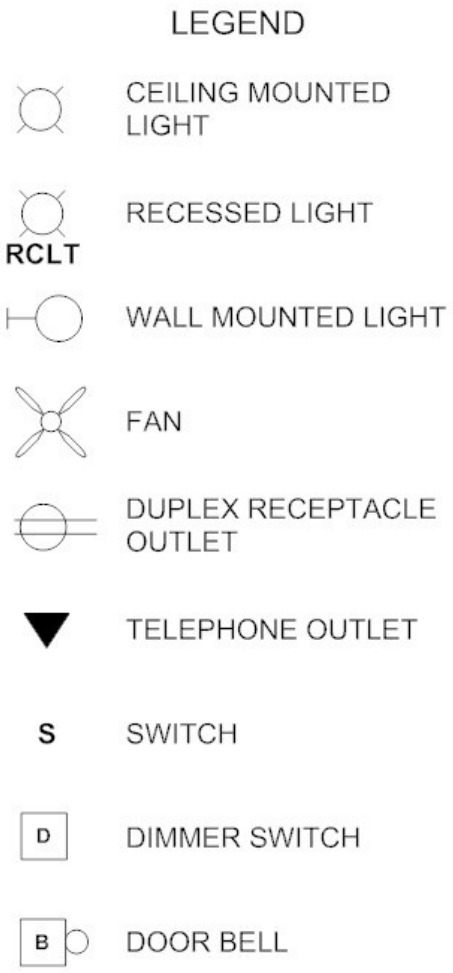


Figure 13. *Example of electrical symbols*

The dataset needs to consist of images which contain the required symbols. These symbols can be in various size, orientations, and placements etc. Therefore, the first step was to acquire different technical layouts that have a wide variety of these electrical symbols. The datasets used in this research were built by acquiring various technical layout through the internet and some were drawn from scratch. The important factor was the quality and variety of symbols contained within these layouts. It turned out to be quite difficult to find high-quality, copyright-free layouts. To artificially increase the size of the dataset, augmentation techniques were used.

Dataset augmentation

The dataset had to be augmented using various transformations. In this way, it was possible to create a large collection of images from a smaller subset. This is a common technique to make a small set of images into a sufficiently varied and adequately numerous sets of images. In many cases it might not be possible to have enough images of high quality to create your dataset.

For this thesis, a simple image augmentation tool called Image Augmentor [48] was used. New images are created by applying a set of transformations to each image file in the chosen directory. Some of the transformations that can be applied are illustrated in figure 14.

Code	Description	Example Values
`fliph`	Horizontal Flip	`fliph`
`flipv`	Vertical Flip	`flipv`
`noise`	Adds random noise to the image	`noise_0.01`, `noise_0.5`
`rot`	Rotates the image by the specified amount	`rot_90`, `rot_-45`
`trans`	Shifts the pixels of the image by the specified amounts in the x and y directions	`trans_20_10`, `trans_-10_0`
`zoom`	Zooms into the specified region of the image, performing stretching/shrinking as necessary	`zoom_0_0_20_20`, `zoom_-10_-20_10_10`
`blur`	Blurs the image by the specified amount	`blur_1.5`

Figure 14. *Transformation examples.*

It is always a good idea to have a high number of images to train the model and after these transformations were applied, there were a total of 500 images for the training task, with each image containing 30-50 electrical symbols.

Dataset annotation

The next step is to prepare the images for use by the model to train on. To train and test a model, accurate annotations are required. This means that every symbol in the image needs to be labelled for the model to identify. An example of annotating images is illustrated in the figure below.

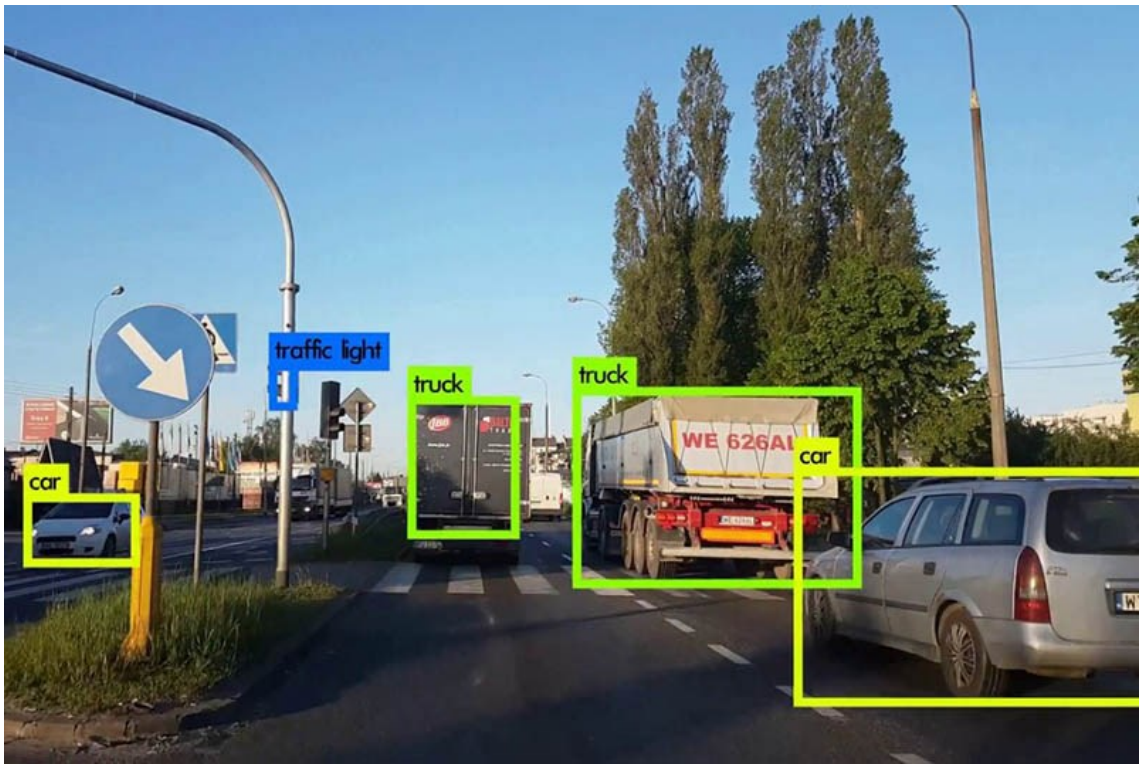


Figure 15. *Example of an annotated image.*

Annotating every image in a dataset is a very time-consuming and laborious task. Tzutalin's "LabelImg" [49] is an intuitive GUI tool that can be used for creating and editing these annotations using an XML format. This tool allows for creation of classes according to your needs. This means that for each individual symbol that needs to be detected, a different class will be created, for example a "Ceiling Mounted Light" class. Then we can draw the bounding boxes for each object in an image. This box will mark the coordinates in which a certain object is located. A step of the process is illustrated in figure 16.

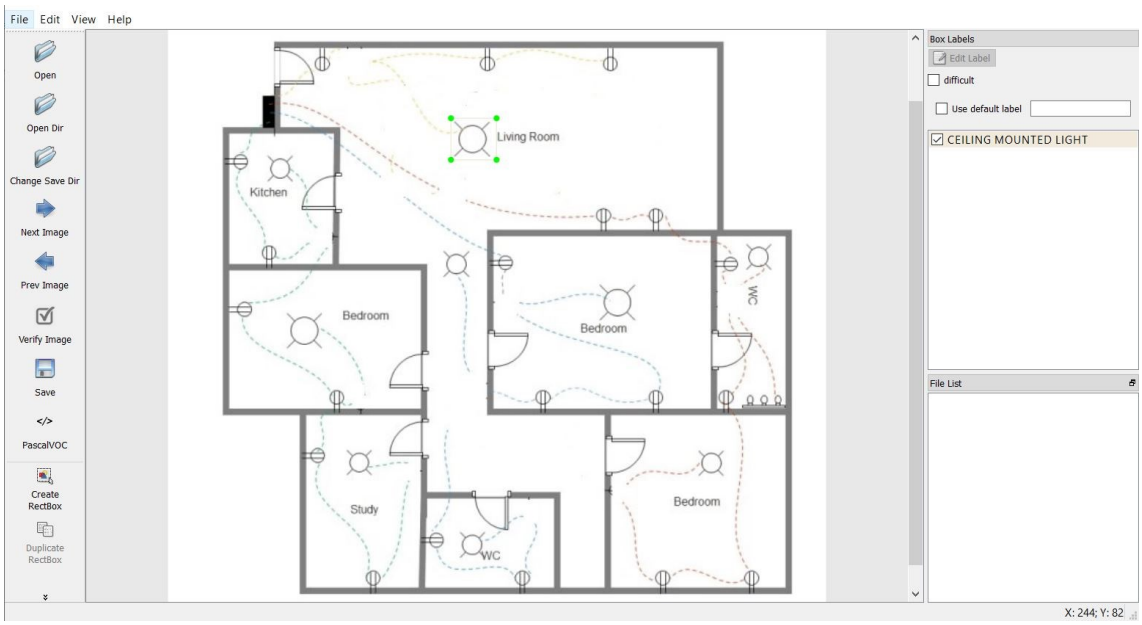


Figure 16. *Annotating an image.*

Once every image is annotated, they can be used to train an object detection model.

3.2 Simulation Environment

There are several different programming languages available to use in machine learning with Python, R, C/C++, and Java being the most widely used. Python was chosen for this task as it is the most common and widely used language.

Due to the shortage of GPUs at the time of this study, the training and inference tasks were done on Google Colab [50]. It offers cloud computing in a Jupyter notebook environment, free of cost. Jupyter is intuitive and easy to use platform for programming. Google Colab provides a NVIDIA Tesla K80 GPU with 12GB RAM which was sufficient for this task.

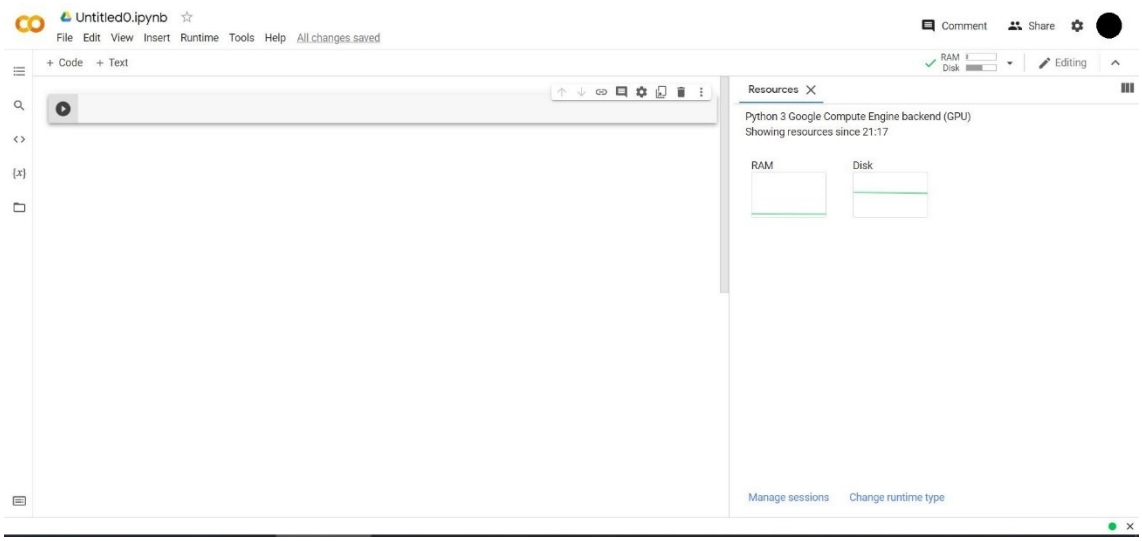


Figure 17. *Google Colab notebook GUI.*

3.3 TensorFlow

TensorFlow [51] provides free and open-source library for machine learning. TensorFlow APIs allow users to develop machine learning models using Keras. TensorFlow provides a diverse collection of models for various machine learning implementations. The major focus of this study was object detection which is why the TensorFlow Object Detection API was used. There are constantly additional methods being introduced to use more efficient models for tackling the task of object detection. Google and TensorFlow have been the leading developers in this field.

3.3.1 TensorFlow models

There are several pre-trained object detection models [52] available for use that are developed and maintained by TensorFlow. These models are tested on the COCO (Common Objects in Context) [53] dataset which contains over 330 thousand images and 80 classes. These are good for initializing models when training on new datasets. The images in the datasets are all labelled and an example of one such image from this dataset, is illustrated in the figure below.

Figure 18. *An image from COCO dataset with image segmentation applied.*

The models each have their own architectures and provide various procedures for object detection. These can be evaluated on different aspects of their results. The TensorFlow model zoo provides a comparison for these models.

Model selection criteria

The primary criterion for evaluation for these models is the mAP (Mean Average Precision). This is a widely used metric for the accuracy of a model, and usually comes with a trade-off in terms of execution speed as well as image resolution limitations. This will tell us how accurately a model can detect objects.

Another criterion is the speed of detection of objects. This is a concern mainly for detection in videos and not that important when detecting objects on static images. It is desirable to have this detection speed be low.

The image resolution is also an important factor to consider when choosing a model. Usually, the model will resize any input image to its own required image resolution.

However, it should be noted that doing so may reduce quality and make it harder to detect objects. Having your dataset images be as close to the model's resolution is desirable for optimal performance.

A section of the library of models is shown in figure 19 [52].

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet HourGlass104 512x512	70	41.9	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40.0/61.4	Boxes/Keypoints
CenterNet HourGlass104 1024x1024	197	44.5	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Boxes/Keypoints
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V1 FPN Keypoints 512x512	30	29.3/50.7	Boxes/Keypoints
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
CenterNet Resnet50 V2 Keypoints 512x512	30	27.6/48.2	Boxes/Keypoints
CenterNet MobileNetV2 FPN 512x512	6	23.4	Boxes
CenterNet MobileNetV2 FPN Keypoints 512x512	6	41.7	Keypoints
EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes
EfficientDet D3 896x896	95	45.4	Boxes
EfficientDet D4 1024x1024	133	48.5	Boxes
EfficientDet D5 1280x1280	222	49.7	Boxes
EfficientDet D6 1280x1280	268	50.5	Boxes

Figure 19. *TensorFlow model zoo.*

Model consideration and comparison

Some of the more commonly used models that were considered for this task with their names, version, and image resolution are listed below.

- EfficientDet D2 768x768
- EfficientDet D3 896x896
- EfficientDet D4 1024x1024
- EfficientDet D6 1280x1280

- EfficientDet D7 1536x1536
- CenterNet HourGlass104 512x512
- CenterNet HourGlass104 1024x1024
- CenterNet HourGlass104 Keypoints 1024x1024
- CenterNet Resnet101 V1 FPN 512x512
- CenterNet Resnet50 V2 512x512
- Faster R-CNN ResNet50 V1 640x640
- Faster R-CNN ResNet50 V1 1024x1024
- Faster R-CNN ResNet101 V1 1024x1024
- Faster R-CNN Inception ResNet V2 640x640
- Faster R-CNN Inception ResNet V2 1024x1024
- Mask R-CNN Inception ResNet V2 1024x1024

There were 41 models available for use at the time of this study and a comparison was made to best determine the most suitable model for this task. Various factors determine the feasibility of using one model over the other. These include the input resolution, the execution speed, and most importantly the mAP score.

As can be seen from the comparison of models [52], it might be prudent to assume that CenterNet HourGlass104 Keypoints 1024x1024 is the best for this task as it has the highest mAP. However, it must be noted that its mAP score is for keypoints and it has a lower score for boxes. As the aim of this thesis is to detect the box coordinates in which a symbol is located, any model with keypoint detection is not a viable option.

Another factor to consider is the image resolution. Each model will change the input image to its own required resolution as stated in the name, for example, 1024x1024. For more efficient object detection, it is recommended to have the resolution of your input image as close to the model's resolution to prevent inaccuracies. Lastly, speed is also a factor to consider here. Faster detection is not a priority for this task, but excessively high times are not desirable and should be avoided.

Considering the factors mentioned above, it was determined that EfficientDet D3 would be the best match for our input image resolution of 800x600. It has a relatively high mAP score of 45.4 for boxes while having a detection time of under 100 ms.

3.3.2 EfficientDet D3

EfficientDet architecture consists of three main stages. A backbone model, a feature network, and a class and box prediction network. Figure 20 provides an overview of the architecture [54].

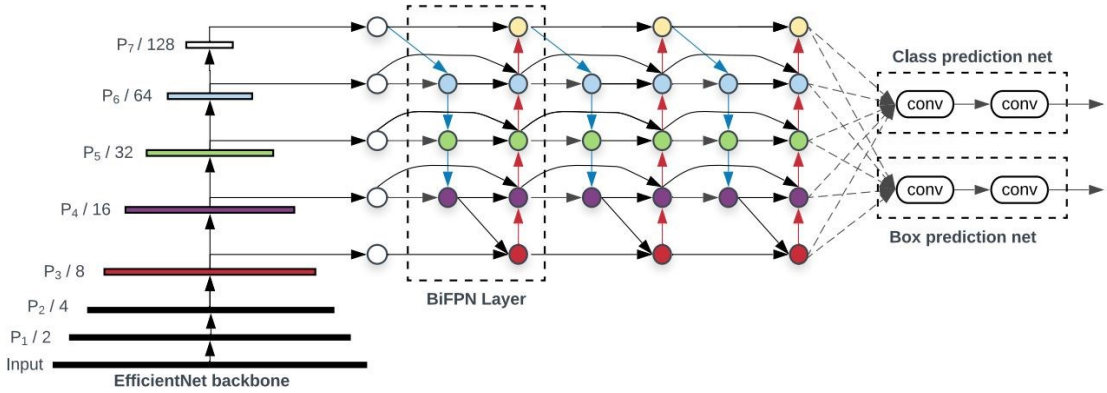


Figure 20. *EfficientDet architecture.*

Backbone model

EfficientDet builds on the EfficientNet model as its backbone. CNNs can be scaled in several ways. You can make the layers wider, deeper or increase resolution. EfficientNet proposed scaling all these factors by a single compound coefficient [55]. Scaling each dimension individually can be laborious and inefficient which is why scaling all of them with the same ratio is a simple yet elegant approach. Figure 21 illustrates this scaling process [55].

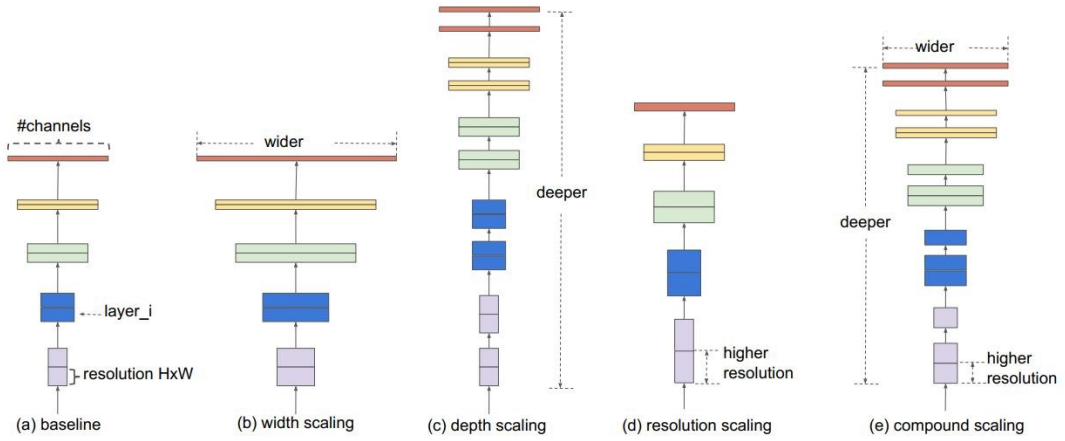


Figure 21. *EfficientNet model scaling.*

Feature network

The features extracted from the backbone model are then used as inputs to a feature network that fuses them to produce an output with the most important features of the

image. BiFPN is a modified version of NAS-FPN and has bidirectional paths in its feature network topology. EfficientDet utilizes the feature network by stacking these BiFPN blocks. The number of blocks vary with each model. The features of an input image at different resolutions can be represented in an aggregated manner and scaled uniformly. A comparison of some feature networks [54] is given in figure 22.

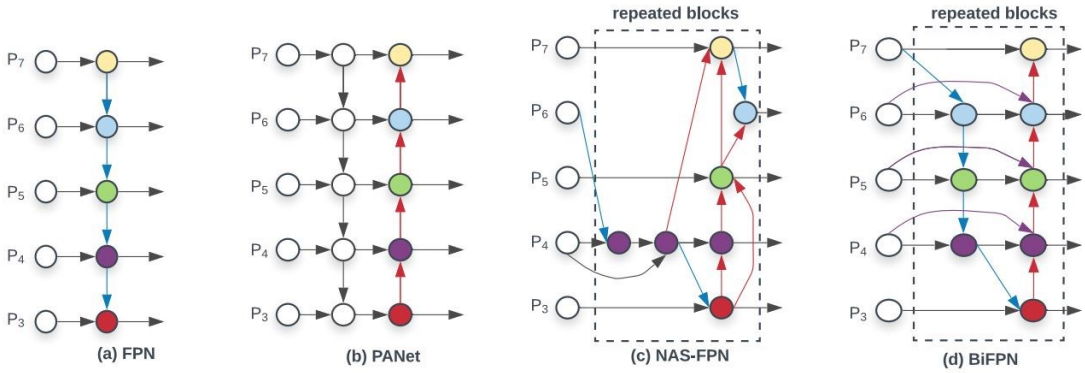


Figure 22. *A comparison of FPN, PANet, NAS-FPN, and BiFPN Feature networks.*

Head models

The last stage of the EfficientDet model is the head models consisting of class and box predictors. These consist of convolution layers that utilize batch normalization and Swish activation functions [54]. This network is used to produce an output for predictions using anchor boxes.

3.3.3 Visualization toolkit

As mentioned in chapter 2.4, TensorFlow provides a visualization toolkit called TensorBoard which allows users to visualize various aspects of models. It can display graphs, histograms, images, and audio data. These are useful to track loss, accuracy, and other metrics. TensorBoard acquires data through logs and Event Files. Monitoring loss is an important part of this study and for that purpose, TensorBoard Scalars Dashboard was used to monitor the loss data during training of the chosen model. The graph that is important for this task, is the total loss graph which needs to be as low as possible for more accurate results.

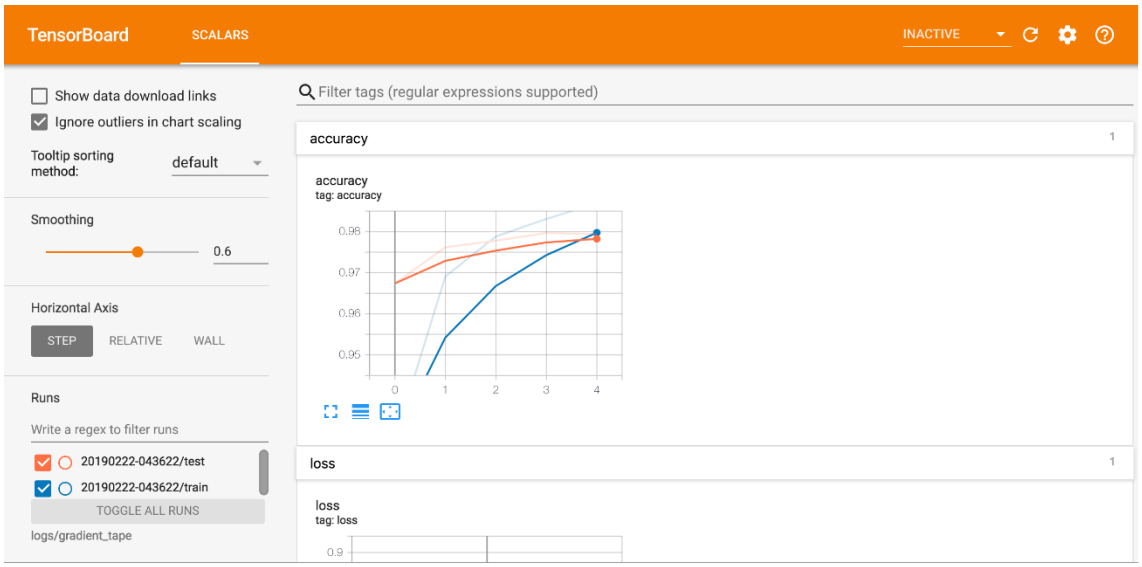


Figure 23. *TensorBoard UI example.*

There are a wide range of visual tools available with TensorBoard that include graphs, histograms, scalars, distributions etc. For training a model, the main tool to observe is the scalars data of loss. This data helps us analyse the accuracy of the model and gives a parameter to determine when to terminate the training process. An example of the loss data representation in TensorBoard is illustrated in figure 24.

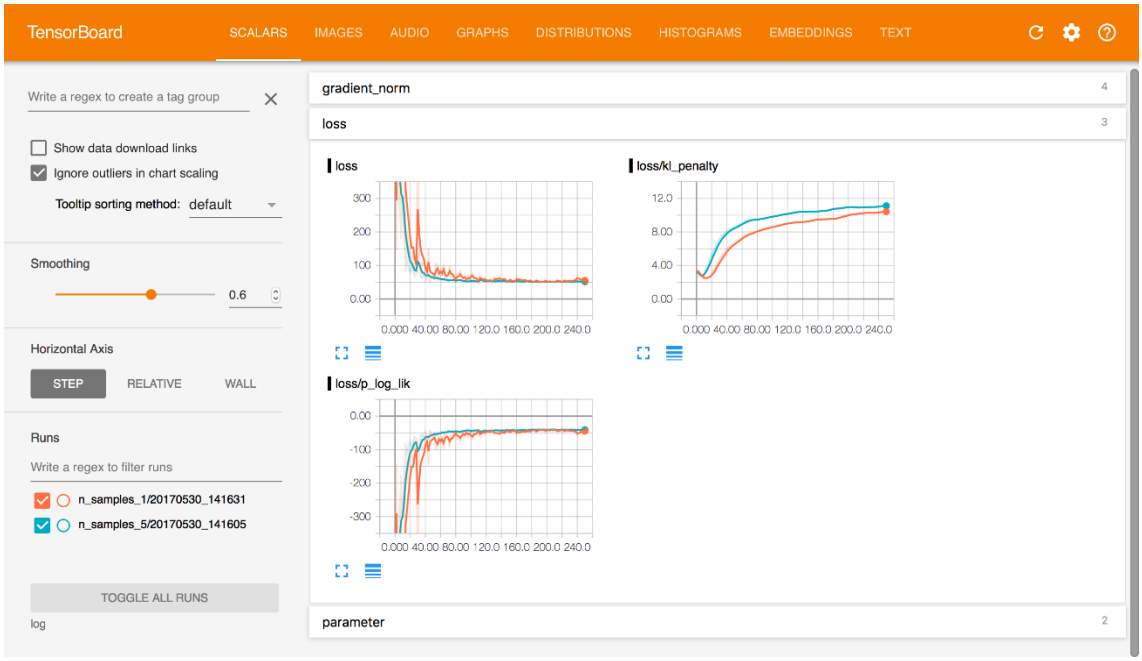


Figure 24. *TensorBoard scalar UI for loss data.*

The data generated throughout the training process can be saved for later use. This is very useful as it allows the user to tune subsequent trainings by adjusting the parameters to the model. A visual representation of the data helps determine whether the model is overfitting or not and the point at which training should be stopped to prevent

this issue. A model is said to be overfitting if the testing losses start increasing whereas the training losses keep decreasing. This essentially means that the model is starting to learn irrelevant data and will not perform well on new input data. TensorBoard visualization tools provide significant help to monitor the process and prevent issues like this from occurring.

4 IMPLEMENTATION

This section discusses the actual implementation of approach presented in section 3. The implementation consists of two main parts. The first part describes the dataset construction and augmentation. The second part describes the implementation of the chosen model using the constructed dataset. This involves the training parameters and the procedure for training and monitoring the process.

4.1 Building Datasets

As discussed previously, constructing a custom dataset is the first step in the object detection process. The steps to create one are detailed in the following sections.

4.1.1 Image collection

Constructing a custom dataset requires a large amount of data. This task requires a diverse set of images for electrical layouts with assorted symbols for each class that is required to be detected. Images were downloaded from copyright free sources, and some were modified or made from scratch. Some of the images needed to be modified so as to retain only the necessary information. Unnecessary information was cropped out and images were rescaled if necessary to 800x600.

Finding copyright free images was more difficult than initially thought, therefore it was necessary to augment this dataset to produce one that was adequate for the training process.

4.1.2 Dataset augmentation and Annotation

The Image Augmentor tool was used to apply certain transformation to the image set. The transformations used are horizontal flip, vertical flip, adding noise to the image, rotating the image, and applying blur to the image. A small sample of two images with applied transformations is illustrated in figure 25.

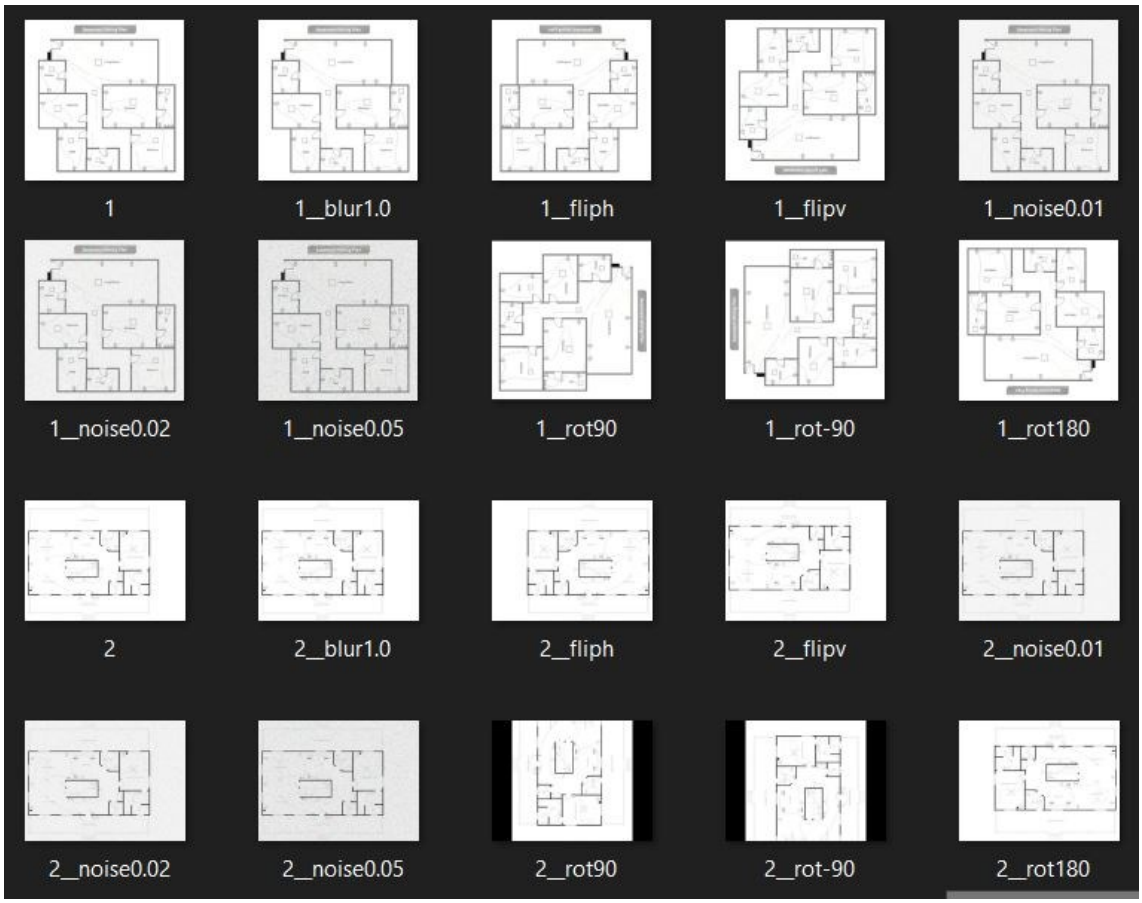


Figure 25. *Transformations applied to images.*

Using this augmentation technique, a dataset of 500 images was constructed. Now that a sufficient dataset has been generated, the next step is to annotate or label the images. The LabelImg tool allows for the creation of classes and to draw rectangular boxes to label each instance of that object. A total of 18 classes were created and are listed in figure 26.

AFF
CEILING_MOUNTED_LIGHT
DIMMER_SWITCH
DOOR_BELL
DUPLEX_RECEPTACLE_OUTLET
DUPLEX_SPECIAL_RECEPTACLE
EMERGENCY_LIGHT
FAN
NURSE_CALL
QUADRUPLEX_OUTLET
RECESSED_DOWN_LIGHT
RECESSED_LIGHT
SINGLE_RECEPTACLE
SMOKE_DETECTOR
SWITCH
TELEPHONE_OUTLET
TV_OUTLET
WALL_MOUNTED_LIGHT

Figure 26. *List of classes in dataset.*

The images were then annotated and each instance of the object to be detected was manually labelled. A category was created for each class like fans, wall lights etc. Each object will be categorised into their respective class and their position marked on the image. An example of one image with the annotations is given in figure 27.

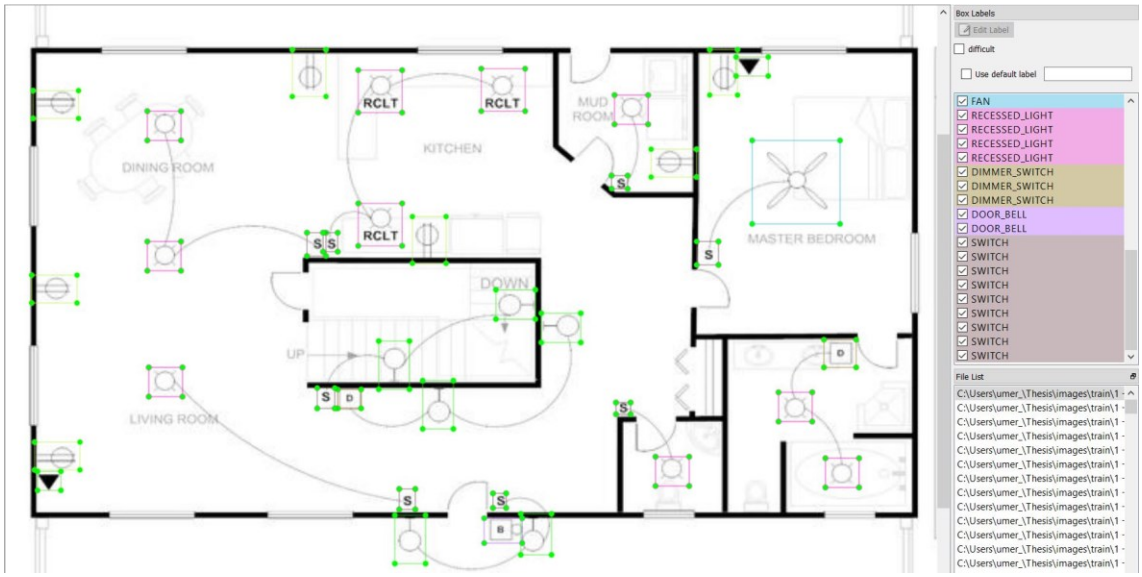


Figure 27. *Annotated image.*

Now each input image will have the objects labelled for the model to train on.

4.1.3 Preprocessing

Once the dataset is compiled and annotated, the next step is to prepare it for the training process. As discussed in chapter 2.2.3, the data needs to be split into two sections. One section for training and another for testing. Generally, 10-20% of the dataset is used for testing so this dataset was split into 400 images for training and 100 images for testing.

TensorFlow uses the TFRecord format which is an easy format to combine large datasets. The files produced from LabelImg are in XML format and need to be converted to the TFRecord format. There are several ways to achieve this. First, the XML files were combined into one CSV file using a simple python script. A sample of the generated CSV file is illustrated in figure 28.

filename	width	height	class	xmin	ymin	xmax	ymax
1 - Copy.jpg	800	600	SWITCH	210	280	216	291
1 - Copy.jpg	800	600	SWITCH	219	282	227	292
1 - Copy.jpg	800	600	SWITCH	212	385	224	397
1 - Copy.jpg	800	600	SWITCH	270	455	280	469
1 - Copy.jpg	800	600	SWITCH	334	456	342	468
1 - Copy.jpg	800	600	SWITCH	474	288	485	297
1 - Copy.jpg	800	600	SWITCH	416	391	426	404
1 - Copy.jpg	800	600	SWITCH	415	237	425	251
1 - Copy.jpg	800	600	RECESSED_LIGHT	242	170	272	197
1 - Copy.jpg	800	600	RECESSED_LIGHT	322	168	358	197
1 - Copy.jpg	800	600	RECESSED_LIGHT	243	263	271	285
1 - Copy.jpg	800	600	DUPLEX_RECEPTACLE_OUTLET	278	268	300	294
1 - Copy.jpg	800	600	DUPLEX_RECEPTACLE_OUTLET	22	180	49	198
1 - Copy.jpg	800	600	DUPLEX_RECEPTACLE_OUTLET	20	307	46	326
1 - Copy.jpg	800	600	DUPLEX_RECEPTACLE_OUTLET	22	426	51	440
1 - Copy.jpg	800	600	DUPLEX_RECEPTACLE_OUTLET	443	222	471	239
1 - Copy.jpg	800	600	DUPLEX_RECEPTACLE_OUTLET	478	155	503	185
1 - Copy.jpg	800	600	DUPLEX_RECEPTACLE_OUTLET	199	155	217	182
1 - Copy.jpg	800	600	CEILING_MOUNTED_LIGHT	95	192	125	216
1 - Copy.jpg	800	600	CEILING_MOUNTED_LIGHT	412	181	442	205
1 - Copy.jpg	800	600	CEILING_MOUNTED_LIGHT	93	370	123	394
1 - Copy.jpg	800	600	CEILING_MOUNTED_LIGHT	94	282	124	306
1 - Copy.jpg	800	600	CEILING_MOUNTED_LIGHT	523	385	553	409
1 - Copy.jpg	800	600	CEILING_MOUNTED_LIGHT	439	427	469	451

Figure 28. *Sample of the csv file generated for training dataset.*

Next, the label map file needs to be created. This allows us to map each label or class to an integer which is used in the training and detecting process. This is a simple text file in the pbtxt format. Each of the 18 classes needs to be mapped to its unique integer. The structure is shown in figure 29.

```
item {
    id: 1
    name: 'AFF'
}
item {
    id: 2
    name: 'CEILING_MOUNTED_LIGHT'
}
item {
    id: 3
    name: 'DIMMER_SWITCH'
}
item {
    id: 4
    name: 'DOOR_BELL'
}
item {
    id: 5
    name: 'DUPLEX_RECEPTACLE_OUTLET'
}
item {
    id: 6
    name: 'DUPLEX_SPECIAL_RECEPTACLE'
}
item {
    id: 7
    name: 'EMERGENCY_LIGHT'
}
item {
    id: 8
    name: 'FAN'
}
item {
    id: 9
    name: 'NURSE_CALL'
}
item {
    id: 10
    name: 'QUADRUPLEX_OUTLET'
}
```

Figure 29. *Label map text file.*

Each object can now be identified by its corresponding identification number to make it easier to categorize.

The CSV file can now be converted to TFRecords format using a simple python script. TFRecord files were then generated for both the training and testing datasets. The required files are now ready to begin training.

4.2 Model

4.2.1 Configure and Train Model

For the training and inference processes, a Jupyter notebook was used on Google Colab. First the TensorFlow model repository needs to be imported.

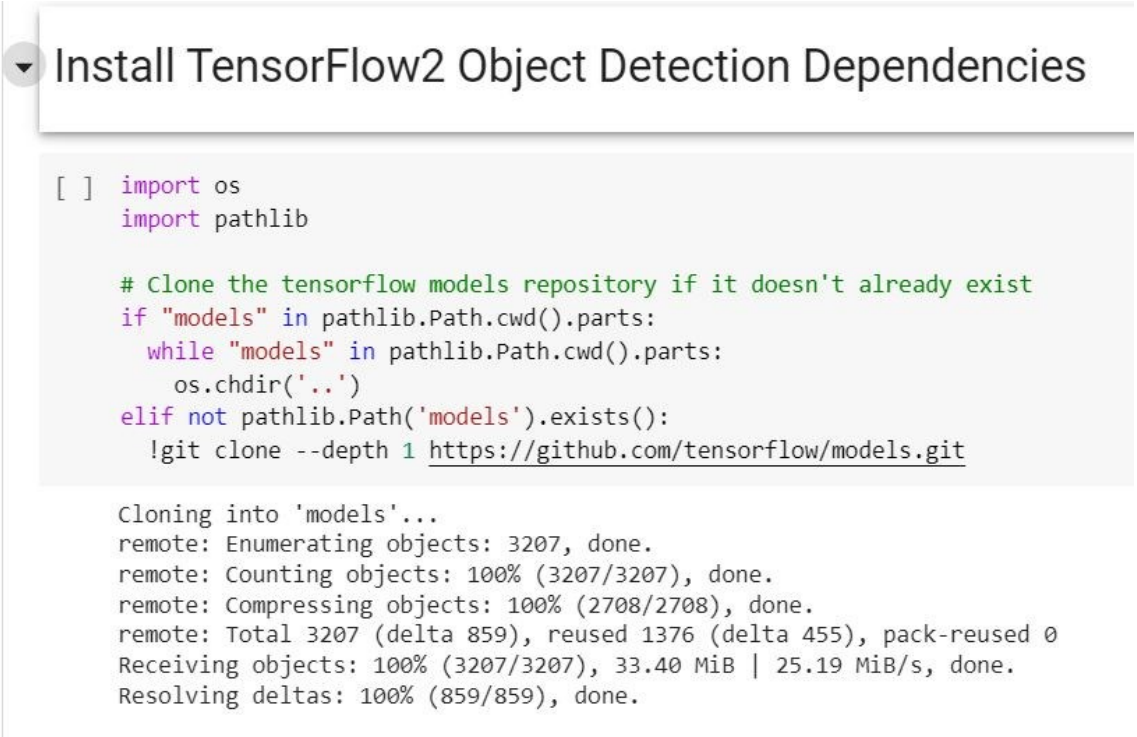


Figure 30. *Importing TensorFlow model repository.*

The training and test TFRecord files and the Label map files were uploaded to Colab. The input path to train and test TFRecord files was set to the uploaded files.

Figure 31. *Configuring path to uploaded user files.*

Next, the model chosen for the process, EfficientDet D3, needs to be configured. The batch size can be adjusted according to the hardware capabilities available. With 12 GB of RAM available, batch size was kept at 2 to keep within hardware limitations. Choosing a higher batch size would require more powerful hardware and would result in faster training. The number of steps depend on the training process and for this task 19500 steps was a good starting point. The number of classes are defined from our label map file already. The paths to pre-trained checkpoint and label map were configured as well. Figure 32 shows the configuration for the chosen model.

```
[ ] MODELS_CONFIG = {
        'efficientdet-d0': {
            'model_name': 'efficientdet_d0_coco17_tpu-32',
            'base_pipeline_file': 'ssd_efficientdet_d0_512x512_coco17_tpu-8.config',
            'pretrained_checkpoint': 'efficientdet_d0_coco17_tpu-32.tar.gz',
            'batch_size': 16
        },
        'efficientdet-d1': {
            'model_name': 'efficientdet_d1_coco17_tpu-32',
            'base_pipeline_file': 'ssd_efficientdet_d1_640x640_coco17_tpu-8.config',
            'pretrained_checkpoint': 'efficientdet_d1_coco17_tpu-32.tar.gz',
            'batch_size': 16
        },
        'efficientdet-d2': {
            'model_name': 'efficientdet_d2_coco17_tpu-32',
            'base_pipeline_file': 'ssd_efficientdet_d2_768x768_coco17_tpu-8.config',
            'pretrained_checkpoint': 'efficientdet_d2_coco17_tpu-32.tar.gz',
            'batch_size': 16
        },
            'efficientdet-d3': {
            'model_name': 'efficientdet_d3_coco17_tpu-32',
            'base_pipeline_file': 'ssd_efficientdet_d3_896x896_coco17_tpu-32.config',
            'pretrained_checkpoint': 'efficientdet_d3_coco17_tpu-32.tar.gz',
            'batch_size': 2
        }
    }

    chosen_model = 'efficientdet-d3'

    num_steps = 19500
    num_eval_steps = 500

    model_name = MODELS_CONFIG[chosen_model]['model_name']
    pretrained_checkpoint = MODELS_CONFIG[chosen_model]['pretrained_checkpoint']
    base_pipeline_file = MODELS_CONFIG[chosen_model]['base_pipeline_file']
    batch_size = MODELS_CONFIG[chosen_model]['batch_size'] #if you can fit a large batch in memory, it may speed up your training
```

Figure 32. *overview of chosen model with parameters and baselines.*

The configuration of the models using the above-mentioned parameters is illustrated in figure 33.

```
import re

%cd /content/models/research/deploy
print('writing custom configuration file')

with open(pipeline_fname) as f:
    s = f.read()
with open('pipeline_file.config', 'w') as f:

    # fine_tune_checkpoint
    s = re.sub('fine_tune_checkpoint: ".*?"',
               'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)

    # tfrecord files train and test.
    s = re.sub(
        '(input_path: ".*?)(PATH_TO_BE_CONFIGURED/train)(.*?")', 'input_path: "{}"'.format(train_record_fname), s)
    s = re.sub(
        '(input_path: ".*?)(PATH_TO_BE_CONFIGURED/val)(.*?")', 'input_path: "{}"'.format(test_record_fname), s)

    # label_map_path
    s = re.sub(
        'label_map_path: ".*?"', 'label_map_path: "{}"'.format(label_map_pbtxt_fname), s)

    # Set training batch_size.
    s = re.sub('batch_size: [0-9]+',
               'batch_size: {}'.format(batch_size), s)

    # Set training steps, num_steps
    s = re.sub('num_steps: [0-9]+',
               'num_steps: {}'.format(num_steps), s)

    # Set number of classes num_classes.
    s = re.sub('num_classes: [0-9]+',
               'num_classes: {}'.format(num_classes), s)

    #fine-tune checkpoint type
    s = re.sub(
        'fine_tune_checkpoint_type: "classification"', 'fine_tune_checkpoint_type: "{}"'.format('detection'), s)

    f.write(s)
```

Figure 33. *Model configuration file.*

We are now ready to begin training. The training can be started using the code shown in figure 34.

```
[ ]  !python /content/models/research/object_detection/model_main_tf2.py \
        --pipeline_config_path={pipeline_file} \
        --model_dir={model_dir} \
        --alsologtostderr \
        --num_train_steps={num_steps} \
        --sample_1_of_n_eval_examples=1 \
        --num_eval_steps={num_eval_steps}
```

Figure 34. *Start training using configured parameters.*

The training process can take several hours depending on the hardware, parameters, and configurations. With a batch size of 2, and 12 GB of RAM, this particular model took approximately 6 hours to reach an acceptable loss. At that point the training process can be terminated, and we can save our checkpoints and fine-tuned model. The result of one training process is shown in figure 35.

```
 'learning_rate': 0.0793647}
INFO:tensorflow:Step 19500 per-step time 1.156s
I0612 04:40:33.057120 139807150659456 model_lib_v2.py:700] Step 19500 per-step time 1.156s
INFO:tensorflow:{'Loss/classification_loss': 0.07963552,
 'Loss/localization_loss': 0.018678721,
 'Loss/regularization_loss': 0.05689582,
 'Loss/total_loss': 0.15521006,
 'learning_rate': 0.079357184}
I0612 04:40:33.057427 139807150659456 model_lib_v2.py:701] {'Loss/classification_loss': 0.07963552,
 'Loss/localization_loss': 0.018678721,
 'Loss/regularization_loss': 0.05689582,
 'Loss/total_loss': 0.15521006,
 'learning_rate': 0.079357184}
```

Figure 35. *Training results.*

4.2.2 Visualization Tool

To help with visualizing the training process and monitor the loss, we can make use of TensorBoard. This can be run at the same time as the training process in taking place. It can be loaded using the commands shown in figure 36 which can be adjusted to the directory in which the data is being logged.

```
[ ]  %load_ext tensorboard
     %tensorboard --logdir '/content/training/train'
```

Figure 36. *Loading TensorBoard UI in notebook.*

The training takes several hours so it is important to observe the loss data towards the later stages. The different losses at the end of the training are illustrated in figure 37.

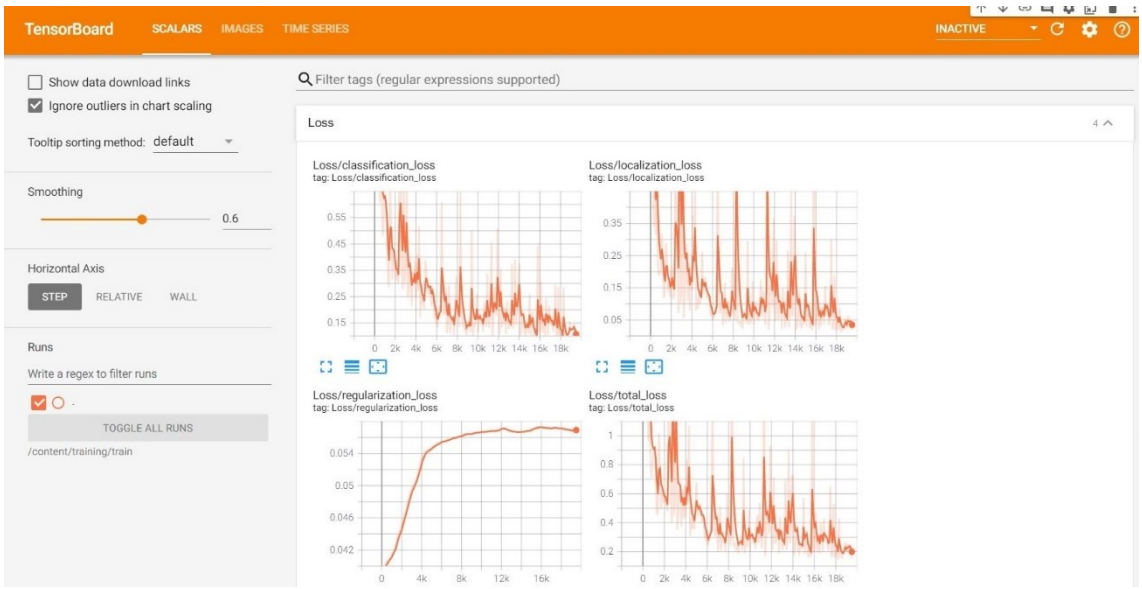


Figure 37. *TensorBoard GUI showing the different losses.*

These scalar graphs are updated every 500 steps as was set up during configuration of the model. The main graph to observe here is the total loss graph. As the value of loss approaches zero, the model is better at classifying objects and has less errors. As can be seen from figure 35 and figure 37, the total loss approached 0.155. For our purpos-

es, this is a sufficiently good value to terminate the training process. We can now proceed to test the model.

5 TEST AND RESULTS

This section describes the testing procedure and the results of the process. The first part describes the procedure to input the test image for inference in the model. The second part discusses the results of the test image.

5.1 Testing

Once the model has been trained, it is possible to test it using any test layout image. As with the training process, the label map needs to be configured and the final checkpoint from training needs to be restored. The code is set up to draw the anchor box on detected symbols. Minimum score threshold for successful detection was kept at 50% or 0.5. A snippet of the code that provides the required output is given in figure 38.

```
TEST_IMAGE_PATHS = glob.glob('/content/drive/MyDrive/Symbol Detector/8.jpg')
image_path = random.choice(TEST_IMAGE_PATHS)
image_np = load_image_into_numpy_array(image_path)

input_tensor = tf.convert_to_tensor(
    np.expand_dims(image_np, 0), dtype=tf.float32)
detections, predictions_dict, shapes = detect_fn(input_tensor)



label_id_offset = 1
image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
      image_np_with_detections,
      detections['detection_boxes'][0].numpy(),
      (detections['detection_classes'][0].numpy() + label_id_offset).astype(int),
      detections['detection_scores'][0].numpy(),
      category_index,
      use_normalized_coordinates=True,
      max_boxes_to_draw=200,
      min_score_thresh=.5,
      agnostic_mode=False,

)
```

Figure 38. *Parameters for detection and drawing bounding boxes.*

A simple code was used to display the detected objects with their coordinates and class. Additionally, a counter was displayed for the number of objects detected in each class. The code snippet is presented in figure 39.

```
plt.figure(figsize=(12,16))
plt.imshow(image_np_with_detections)
plt.show()
boxes = detections['detection_boxes'][0].numpy()

# get all boxes from an array
max_boxes_to_draw = boxes.shape[0]
# get scores to get a threshold
scores = detections['detection_scores'][0].numpy()

min_score_thresh=.5
class_names = list()
# iterate over all objects found
for i in range(min(max_boxes_to_draw, boxes.shape[0])):

  if scores is None or scores[i] > min_score_thresh:
    class_name = category_index[(detections['detection_classes'][0].numpy() + label_id_offset).astype(int)[i]]['name']
    class_names.append(class_name)

    print ("Detected", boxes[i], class_name)
import collections
occurrences = collections.Counter(class_names)
print("Symbols Detected")
print(occurrences)
```

Figure 39. *Code for displaying a machine readable output.*

Multiple images were tested using this fine-tuned model and their results were recorded. One of the test images used to evaluate the model is demonstrated in figure 40.

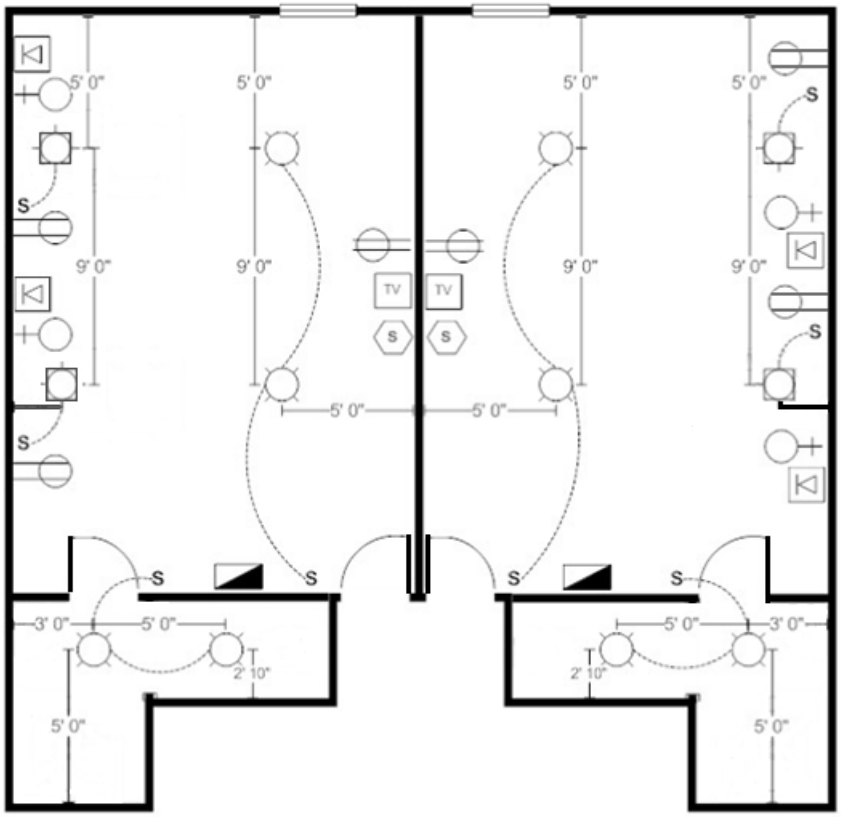
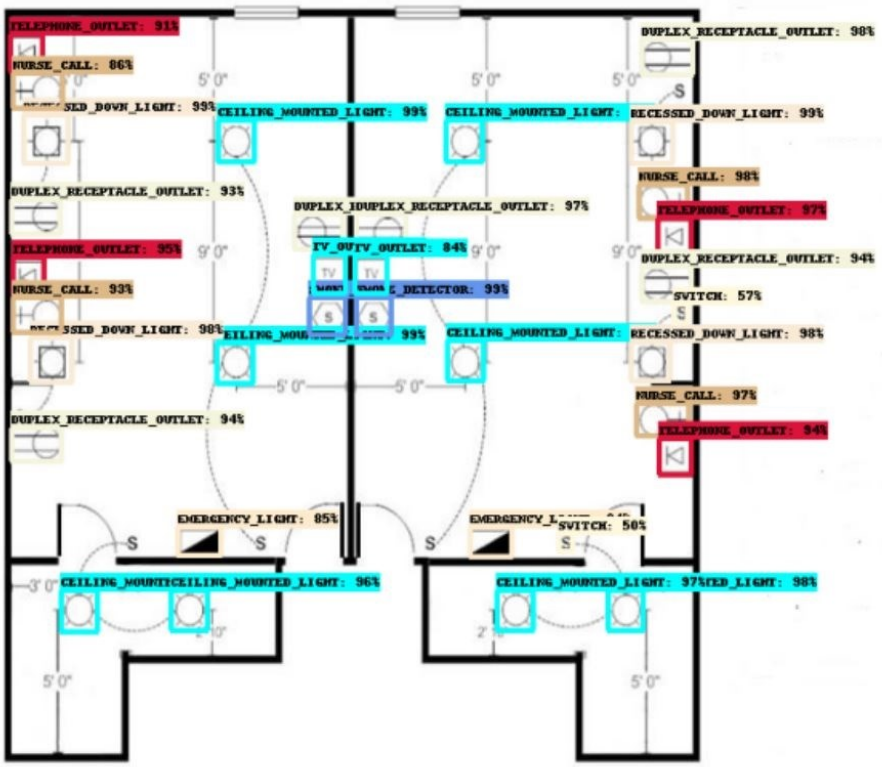


Figure 40. *Input test image.*

The expected outcome from this image is that each object present in the image will be marked with an anchor box that displays the confidence level of detection. The text output should have every symbol that is detected with its coordinates listed. Finally, a counter of the number of instances of each object detected should be displayed.

5.2 Results

The process of inference on the image from figure 40 is illustrated in figure 41 below. These results can be saved in a separate image file with a corresponding text file for mapped coordinates.




```
Detected [0.3999407  0.30693725 0.44427654 0.3413742 ] SMOKE_DETECTOR
Detected [0.29591715 0.29336983 0.3382874  0.33936304] DUPLEX_RECEPTACLE_OUTLET
Detected [0.17279397 0.04064617 0.2312678  0.08329089] RECESSED_DOWN_LIGHT
Detected [0.18048333 0.60645163 0.23059228 0.6474432 ] RECESSED_DOWN_LIGHT
Detected [0.452306   0.43572134 0.50149417 0.4702894 ] CEILING_MOUNTED_LIGHT
Detected [0.39979184 0.35034877 0.44485193 0.38418323] SMOKE_DETECTOR
Detected [0.44927812 0.04749852 0.50241417 0.08655459] RECESSED_DOWN_LIGHT
Detected [0.4540972  0.6069914  0.50158787 0.64453053] RECESSED_DOWN_LIGHT
Detected [0.25984094 0.61395395 0.29853284 0.66094923] NURSE_CALL
Detected [0.7626976  0.07591581 0.812068   0.11039143] CEILING_MOUNTED_LIGHT
Detected [0.7624449  0.58547133 0.81155866 0.61970156] CEILING_MOUNTED_LIGHT
Detected [0.07894572 0.6172544  0.12082309 0.6654246 ] DUPLEX_RECEPTACLE_OUTLET
Detected [0.29621708 0.3505107  0.33888984 0.3978044 ] DUPLEX_RECEPTACLE_OUTLET
Detected [0.30139983 0.6320653  0.34581074 0.66376865] TELEPHONE_OUTLET
Detected [0.530509   0.6123844  0.56889004 0.6614793 ] NURSE_CALL
Detected [0.7625753  0.48161224 0.8115835  0.5168656 ] CEILING_MOUNTED_LIGHT
Detected [0.76229584 0.17827214 0.81052613 0.21283738] CEILING_MOUNTED_LIGHT
Detected [0.34979746 0.03025473 0.39411288 0.06056774] TELEPHONE_OUTLET
Detected [0.56038946 0.02923722 0.60146797 0.07639654] DUPLEX_RECEPTACLE_OUTLET
Detected [0.3608975  0.61743796 0.40251333 0.6644175 ] DUPLEX_RECEPTACLE_OUTLET
Detected [0.57342386 0.63348377 0.61675453 0.664685  ] TELEPHONE_OUTLET
Detected [0.3987314  0.03095918 0.44081283 0.07647569] NURSE_CALL
Detected [0.27757376 0.02926043 0.31962907 0.07616956] DUPLEX_RECEPTACLE_OUTLET
Detected [0.07253531 0.02859134 0.11692542 0.05978265] TELEPHONE_OUTLET
Detected [0.34648573 0.31014985 0.3927878  0.34174788] TV_OUTLET
Detected [0.12096789 0.03032156 0.16328478 0.07861567] NURSE_CALL
Detected [0.6835716  0.18468508 0.7166795  0.22521529] EMERGENCY_LIGHT
Detected [0.6842973  0.45705625 0.71885216 0.49717948] EMERGENCY_LIGHT
Detected [0.34698993 0.3499301  0.3926564  0.38034946] TV_OUTLET
Detected [0.4074633  0.64733267 0.42839918 0.66121554] SWITCH
Detected [0.6926593  0.53945786 0.7125115  0.55338234] SWITCH
Symbols Detected
Counter({'CEILING_MOUNTED_LIGHT': 8, 'DUPLEX_RECEPTACLE_OUTLET': 6, 'RECESSED_DOWN_LIGHT': 4, 'NURSE_CALL': 4,
'TELEPHONE_OUTLET': 4, 'SMOKE_DETECTOR': 2, 'TV_OUTLET': 2, 'EMERGENCY_LIGHT': 2, 'SWITCH': 2})
```

Figure 41. *Output of test image.*

Accuracy

As can be seen from the result, all the symbols were correctly identified with confidence level near or above 95% for most cases. These results mean that the model is

highly accurate in detecting the electrical symbols. The only exception is the switch symbol which is essentially an alphabet.

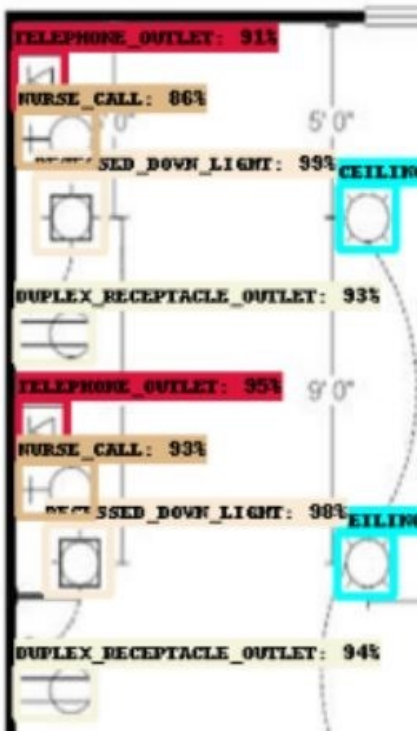


Figure 42. *Anchor boxes showing high confidence level.*

The EfficientDet D3 model chosen for this task was successful in detecting the required symbols with the accuracy expected.

Machine-readable output

The coordinates are in a format that is easily read by machines and also contains the label or identifier for each object. This was the desired outcome and is sufficient for digitizing purposes. The text file generated for the above test image is show in figure 43.

```
Detected [0.3999407  0.30693725 0.44427654 0.3413742 ] SMOKE_DETECTOR
Detected [0.29591715 0.29336983 0.3382874  0.33936304] DUPLEX_RECEPTACLE_OUTLET
Detected [0.17279397 0.04064617 0.2312678  0.08329089] RECESSED_DOWN_LIGHT
Detected [0.18048333 0.60645163 0.23059228 0.6474432 ] RECESSED_DOWN_LIGHT
Detected [0.452306   0.43572134 0.50149417 0.4702894 ] CEILING_MOUNTED_LIGHT
Detected [0.39979184 0.35034877 0.44485193 0.38418323] SMOKE_DETECTOR
Detected [0.44927812 0.04749852 0.50241417 0.08655459] RECESSED_DOWN_LIGHT
Detected [0.4540972  0.6069914  0.50158787 0.64453053] RECESSED_DOWN_LIGHT
Detected [0.25984094 0.61395395 0.29853284 0.66094923] NURSE_CALL
Detected [0.7626976  0.07591581 0.812068   0.11039143] CEILING_MOUNTED_LIGHT
Detected [0.7624449  0.58547133 0.81155866 0.61970156] CEILING_MOUNTED_LIGHT
Detected [0.07894572 0.6172544  0.12082309 0.6654246 ] DUPLEX_RECEPTACLE_OUTLET
Detected [0.29621708 0.3505107  0.33888984 0.3978044 ] DUPLEX_RECEPTACLE_OUTLET
Detected [0.30139983 0.6320653  0.34581074 0.66376865] TELEPHONE_OUTLET
Detected [0.530509   0.6123844  0.56889004 0.6614793 ] NURSE_CALL
Detected [0.7625753  0.48161224 0.8115835  0.5168656 ] CEILING_MOUNTED_LIGHT
Detected [0.76229584 0.17827214 0.81052613 0.21283738] CEILING_MOUNTED_LIGHT
Detected [0.34979746 0.03025473 0.39411288 0.06056774] TELEPHONE_OUTLET
Detected [0.56038946 0.02923722 0.60146797 0.07639654] DUPLEX_RECEPTACLE_OUTLET
Detected [0.3608975  0.61743796 0.40251333 0.6644175 ] DUPLEX_RECEPTACLE_OUTLET
Detected [0.57342386 0.63348377 0.61675453 0.664685  ] TELEPHONE_OUTLET
Detected [0.3987314  0.03095918 0.44081283 0.07647569] NURSE_CALL
Detected [0.27757376 0.02926043 0.31962907 0.07616956] DUPLEX_RECEPTACLE_OUTLET
Detected [0.07253531 0.02859134 0.11692542 0.05978265] TELEPHONE_OUTLET
Detected [0.34648573 0.31014985 0.3927878  0.34174788] TV_OUTLET
Detected [0.12096789 0.03032156 0.16328478 0.07861567] NURSE_CALL
Detected [0.6835716  0.18468508 0.7166795  0.22521529] EMERGENCY_LIGHT
Detected [0.6842973  0.45705625 0.71885216 0.49717948] EMERGENCY_LIGHT
Detected [0.34698993 0.3499301  0.3926564  0.38034946] TV_OUTLET
Detected [0.4074633  0.64733267 0.42839918 0.66121554] SWITCH
Detected [0.6926593  0.53945786 0.7125115  0.55338234] SWITCH
Symbols Detected
Counter({'CEILING_MOUNTED_LIGHT': 8, 'DUPLEX_RECEPTACLE_OUTLET': 6, 'RECESSED_DOWN_LIGHT': 4, 'NURSE_CALL': 4,
'TELEPHONE_OUTLET': 4, 'SMOKE_DETECTOR': 2, 'TV_OUTLET': 2, 'EMERGENCY_LIGHT': 2, 'SWITCH': 2})
```

Figure 43. *Text file with machine readable output.*

Exceptions

This model is pretrained for a certain dataset type and that does not include alphabets. Detecting text is not the primary purpose of these models which explains why it had a hard time detecting the switch symbol.

The minimum confidence level for detection was set at 50% so even though the switch symbols may be detected, they will not be shown here because of low confidence. Only 2 switch symbols that were at 50% confidence are shown. It might be beneficial to exclude all alphabetical symbol for this task as these models are not suited for it.

Testing and retraining times

The testing is quick as it takes only a couple of seconds to display the results. More images can be tested one at a time provided they contain only the classes of objects already trained. This test proved to be effective for the desired outcome with successful detection of symbols. Introducing a new class of objects would require retraining the model which would take roughly the same amount of time as the previous training of 6 hours. This is a limitation to the models' capabilities as you need to determine all the classes that need to be included before training the model. Adding additional classes has a significant time requirement which is why it was so important to determine the scope of the task at the start.

Image resolution considerations

It is also important to note that input images should be close to 896x896 resolution of the model. There is a limit to the size of bounding box that can be drawn on the image by the model. Therefore, small images when scaled up to the required resolution, will make it impossible for the symbol to be detected and marked. The same is true for downsizing vary large images to the required size. It will essentially make the object very small and therefore hard to detect.

6 CONCLUSIONS

This section provides a conclusion to the thesis. The first part summarizes the work done in this thesis and provides answers to the research questions posed in section 1. The second part discusses the potential for future work using the approach in this thesis.

6.1 Summary

This thesis presents an approach to solving the problem of digitizing technical image layouts using machine learning and computer vision. It was proven during the implementation that a pre-trained model can be used to train a custom dataset comprising of the desired images.

The designed system provides instructions that allow the user to create their own custom dataset and to train it using TensorFlow models. The system allows for the selection of desired model with personalised requirements. The model uses the user dataset to train for customized object detection. The training process is visualized using TensorBoard and losses are easily monitored. The fine-tuned model is saved and ready for use in testing. The testing interface can be accessed by the user to input test images and obtain results.

The thesis provides guidelines to create and augment a custom dataset. It also provides guidelines to select a model based on the required image resolution and precision scores. Configuration of the model and preparing dataset for training are also outlined. The model was trained in approximately 6 hours with the available hardware but can be scaled up to be faster with more powerful computational hardware.

The system was capable of detecting required objects with high accuracy and is adaptable in nature according to the inputs. The adaptability is a key factor in this thesis as the possibilities of technical layouts are ever expanding. The output is in a machine-readable format and can be utilized for future work.

6.2 Future work

The current system is tuned for detecting symbols on an electrical layout and achieved its goals. However, it can still be developed further to provide useful data in other forms of technical layouts.

The models available in TensorFlow model zoo are adaptable to a wide variety of objects. With the correct custom datasets and model, any other technical layout may be digitized. For example, Piping and instrumentation diagrams can be digitized by creating a dataset using layout images containing the symbols used in them.

Another way to improve this system would be to use more computing power. The system was implemented on a cloud computing environment with limitation to the hardware available. More computing power can yield more accurate and faster results while utilizing complex models.

The technology used in this thesis can work with video as well. In the future there may be opportunities to incorporate those elements into the digitization of technical layouts. There could be an option to record videos of a structure and possibly make a digitized version of the layout.

7 REFERENCES

[1] J. Alzubi, A. Nayyar, and A. Kumar, "Machine learning from theory to algorithms: an overview," in *Journal of physics: conference series*, 2018, vol. 1142, no. 1: IOP Publishing, p. 012012.

[2] H. J. N. m. IJ, "Statistics versus machine learning," vol. 15, no. 4, p. 233, 2018.

[3] S. Liu, X. Wang, M. Liu, and J. J. V. I. Zhu, "Towards better analysis of machine learning models: A visual analytics perspective," vol. 1, no. 1, pp. 48-56, 2017.

[4] P. Tubaro, A. A. Casilli, M. J. B. D. Coville, and Society, "The trainer, the verifier, the imitator: Three ways in which human platform workers support artificial intelligence," vol. 7, no. 1, p. 2053951720919776, 2020.

[5] J. Carifio, J. Halverson, D. Krioukov, and B. D. J. J. o. H. E. P. Nelson, "Machine learning in the string landscape," vol. 2017, no. 9, pp. 1-36, 2017.

[6] G. Kostopoulos, S. Karlos, S. Kotsiantis, O. J. J. o. I. Ragos, and F. Systems, "Semi-supervised regression: A recent review," vol. 35, no. 2, pp. 1483-1500, 2018.

[7] P. Ranganathan, C. Pramesh, and R. J. P. i. c. r. Aggarwal, "Common pitfalls in statistical analysis: logistic regression," vol. 8, no. 3, p. 148, 2017.

[8] X. Zou, Y. Hu, Z. Tian, and K. Shen, "Logistic regression model optimization and case analysis," in *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*, 2019: IEEE, pp. 135-139.

[9] F. Thabtah, "Autism spectrum disorder screening: machine learning adaptation and DSM-5 fulfillment," in *Proceedings of the 1st International Conference on Medical and health Informatics 2017*, 2017, pp. 1-6.

[10] M. Usama *et al.*, "Unsupervised machine learning for networking: Techniques, applications and research challenges," vol. 7, pp. 65579-65615, 2019.

[11] H. U. Dike, Y. Zhou, K. K. Deveerasetty, and Q. Wu, "Unsupervised learning based on artificial neural network: A review," in *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, 2018: IEEE, pp. 322-327.

[12] J. N. Foerster, Y. M. Assael, N. De Freitas, and S. J. a. p. a. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," 2016.

[13] F. Zhuang *et al.*, "A comprehensive survey on transfer learning," vol. 109, no. 1, pp. 43-76, 2020.

[14] S. Sharma, S. Sharma, and A. J. t. d. s. Athaiya, "Activation functions in neural networks," vol. 6, no. 12, pp. 310-316, 2017.

[15] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 843-852.

[16] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 international interdisciplinary PhD workshop (IIPhDW)*, 2018: IEEE, pp. 117-122.

[17] H. Ahn, Y.-H. J. J. o. A. I. Lee, and H. Computing, "Performance analysis of object recognition and tracking for the use of surveillance system," vol. 7, no. 5, pp. 673-679, 2016.

[18] Z.-Q. Zhao, P. Zheng, S.-t. Xu, X. J. I. t. o. n. n. Wu, and l. systems, "Object detection with deep learning: A review," vol. 30, no. 11, pp. 3212-3232, 2019.

[19] L. Huang, J. Li, H. Hao, X. J. T. Li, and U. S. Technology, "Micro-seismic event detection and location in underground mines by using Convolutional Neural Networks (CNN) and deep learning," vol. 81, pp. 265-276, 2018.

[20] M. Fatima, M. J. J. o. I. L. S. Pasha, and Applications, "Survey of machine learning algorithms for disease diagnostic," vol. 9, no. 01, p. 1, 2017.

[21] X. Wang, J. Gao, M. Long, and J. Wang, "Self-tuning for data-efficient deep learning," in *International Conference on Machine Learning*, 2021: PMLR, pp. 10738-10748.

[22] Q. Liu, P. Li, W. Zhao, W. Cai, S. Yu, and V. C. J. I. a. Leung, "A survey on security threats and defensive techniques of machine learning: A data driven view," vol. 6, pp. 12103-12117, 2018.

[23] C. Shorten and T. M. J. J. o. B. D. Khoshgoftaar, "A survey on image data augmentation for deep learning," vol. 6, no. 1, pp. 1-48, 2019.

[24] E. Goceri, "Challenges and recent solutions for image segmentation in the era of deep learning," in *2019 ninth international conference on image processing theory, tools and applications (IPTA)*, 2019: IEEE, pp. 1-6.

[25] P. A. Yushkevich and G. J. I. p. Gerig, "ITK-SNAP: an intractive medical image segmentation tool to meet the need for expert-guided segmentation of complex medical images," vol. 8, no. 4, pp. 54-57, 2017.

[26] D. S. Prabha and J. S. J. I. J. S. T. Kumar, "Performance evaluation of image segmentation using objective methods," vol. 9, no. 8, pp. 1-8, 2016.

[27] O. Bernard *et al.*, "Deep learning techniques for automatic MRI cardiac multi-structures segmentation and diagnosis: Is the problem solved?," vol. 37, no. 11, pp. 2514-2525, 2018.

[28] J. Lee, E. Kim, S. Lee, J. Lee, and S. Yoon, "Ficklenet: Weakly and semi-supervised semantic image segmentation using stochastic inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5267-5276.

[29] M. Abd El Aziz, A. A. Ewees, A. E. Hassanien, M. Mudhsh, and S. Xiong, "Multi-objective whale optimization algorithm for multilevel thresholding segmentation," in *Advances in soft computing and machine learning in image processing*: Springer, 2018, pp. 23-39.

[30] R. Priyadharsini and T. S. J. P. C. S. Sharmila, "Object detection in underwater acoustic images using edge based segmentation method," vol. 165, pp. 759-765, 2019.

[31] C. Zhu, Y. Zheng, K. Luu, and M. Savvides, "Cms-rcnn: contextual multi-scale region-based cnn for unconstrained face detection," in *Deep learning for biometrics*: Springer, 2017, pp. 57-79.

[32] S. Beucher and F. Meyer, "The morphological approach to segmentation: the watershed transformation," in *Mathematical morphology in image processing*: CRC Press, 2018, pp. 433-481.

[33] X. Jia, T. Lei, X. Du, S. Liu, H. Meng, and A. K. J. I. A. Nandi, "Robust self-sparse fuzzy clustering for image segmentation," vol. 8, pp. 146182-146195, 2020.

[34] I. Saetchnikov, "Image segmentation using deep learning methods," 2019.

[35] K. O'Shea and R. J. a. p. a. Nash, "An introduction to convolutional neural networks," 2015.

[36] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017: Ieee, pp. 1-6.

[37] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *International conference on rough sets and knowledge technology*, 2014: Springer, pp. 364-375.

[38] H. Nakahara, T. Fujii, and S. Sato, "A fully connected layer elimination for a binarizec convolutional neural network on an FPGA," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017: IEEE, pp. 1-4.

[39] D. Unzueta. "Convolutional Layers vs Fully Connected Layers." https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b (accessed.

[40] A. F. J. a. p. a. Agarap, "Deep learning using rectified linear units (relu)," 2018.

[41] P. Petersen and F. J. N. N. Voigtlaender, "Optimal approximation of piecewise smooth functions using deep ReLU neural networks," vol. 108, pp. 296-330, 2018.

[42] L. Tao, C. Zhu, G. Xiang, Y. Li, H. Jia, and X. Xie, "LLCNN: A convolutional neural network for low-light image enhancement," in *2017 IEEE Visual Communications and Image Processing (VCIP)*, 2017: IEEE, pp. 1-4.

[43] P. Saravanan. "Understanding Loss Functions in Machine Learning." https://www.section.io/engineering-education/understanding-loss-functions-in-machine-learning/ (accessed.

[44] D. Baylor *et al.*, "Continuous Training for Production {ML} in the TensorFlow Extended ({TFX}) Platform," in *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*, 2019, pp. 51-53.

[45] L. Liu, M. Ji, and M. J. S. Buchroithner, "Transfer learning for soil spectroscopy based on convolutional neural networks and its application in soil clay content mapping using hyperspectral imagery," vol. 18, no. 9, p. 3169, 2018.

[46] M. Dahl *et al.*, "Private machine learning in tensorflow using secure computation," 2018.

[47] P. Chadha and T. Siddagangaiah, "Performance Analysis of Accelerated Linear Algebra Compiler for TensorFlow."

[48] R. Dawson. "Image Augmentor." GitHub repository. https://github.com/codebox/image_augmentor (accessed December, 2021).

[49] Tzutalin. "Labelimg." GitHub repository. https://github.com/tzutalin/labelImg (accessed December, 2021).

[50] E. Bisong. "Building Machine Learning and Deep Learning Models on Google Cloud Platform." Google Colaboratory. https://doi.org/10.1007/978-1-4842-4470-8_7 (accessed December, 2021).

[51] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/.

[52] H. Yu *et al.* "TensorFlow Model Garden." https://github.com/tensorflow/models (accessed December, 2021).

[53] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," 2014.

[54] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," 2019.

[55] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," 2019.