DEEP LEARNING-BASED OBJECT DETECTION IN WOUND IMAGES

by

Yash Patel

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Computer Science

at

The University of Wisconsin-Milwaukee

May 2020

ABSTRACT

DEEP LEARNING-BASED OBJECT DETECTION IN WOUND IMAGES

by

Yash Patel

The University of Wisconsin-Milwaukee, 2020
Under the Supervision of Professor Zeyun Yu

Developing a deep neural network for wound localization was the first step towards an efficient and fully automated wound healing system. A wound localizer was developed in this research using the YOLOv3 model, and an iOS mobile app was also created with the developed localization algorithm. The developed system can detect the wound and its surrounding tissue and isolate the portion of the localized wound for future care. This will support the segmentation and classification of wound by eliminating a lot of redundant details from photos of wound. A lighter variant of YOLOv3 called tiny-YOLOv3 is used for mobile device video processing. The model is trained and tested on an independently created dataset, designed in collaboration with AZH Wound and Vascular Center, Milwaukee, Wisconsin. Model YOLOv3 is contrasted with model SSD, showing that YOLOv3 gives 93.9% of the mAP value, which is much better than the SSD model (86.4%). These models' robustness and reliability are shown to be very good when evaluated on a dataset that is publicly available.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Artificial Intelligence	AI
Machine Learning	ML
Deep Learning	DL
Neural Nets	NN
Convolutional Neural Network	CNN
Deep Convolution Neural Network	DCNN
Regional based Convolution Network	RCNN
You Only Look Once version 3	YOLOv3
Single Shot multi-box Detector	SSD
Computed Tomography	CT
Magnetic Resonance Imaging	MRI
Mean Average Precision	mAP
Intersection Over Union	IoU

ACKNOWLEDGMENTS

This thesis project benefited from conversations with many intelligent people in the Big Data Analytics and Visualization Laboratory at the University of Wisconsin- Milwaukee. Professor Zeyun Yu gave me lots of high-level advice from his years of experience. I would also like to thank our lab group for providing their GPU, their knowledge, help, and their insightful suggestions.

CHAPTER 1: Introduction

The art of diagnosing a medical condition by analyzing medical data has only been possible by humans in the past. Medical experts were the only interpreters of medical data. The vast growth in medical images that are quite complex and subjective to interpret has led health care experts to turn their attention towards the computer industry for solutions. The advancement in technology, computational power, and high availability of medical data has allowed experts to use various machine and deep learning techniques to train a computer system as an expert, which can be used further for prediction and decision making [3]. Deep learning is a branch of machine learning that uses artificial intelligence to retrieve and process networked data that is unstructured for decision making.

The abilities of learning algorithms to choose edges and features to analyze an image and give the best results, make them highly appropriate for medical imaging. The introduction of Deep learning techniques has made the diagnosis process easier and more accurate in healthcare. Along with the widely open data movement, the rapid development of deep learning methods has pushed computer-assisted diagnosis to a great extent.

Medical image analysis constitutes four fundamental tasks, which include, object classification, localization, detection and segmentation. Object classification predicts the class of an object in an image. Object localization locates the position of object in the image using bounding box technique. Object detection locates the objects with bounding box and classifies the type of objects in the image. Object segmentation indicates the recognized object by highlighting the specific pixels of object instead of bounding box [3].

Initially, object detection methods constituted handcrafted features that had narrow architectures with low performance, which generated multiple images of low-level features. The rapid development of deep learning has helped in the advancement of powerful tools that provided a higher quality image with advanced features that could tackle problems faced in the past.

Object detection algorithms are used in various applications, such as, wound segmentation, which is automatic wound area measurement and tissue analysis for wound observation and wound healing. Steps of wound treatment can include wound grading that involves describing the wound by its dimensions, the color of its composition to identify the type of tissue. Deep Neural Network (DNN) is used to determine the time progress of wound healing in terms of its color composition. Data, based on the color of the tissue wound as it heals at different stages, is fed to the dataset such as light red for unhealed injury [1], brown to almost healing and black to fully healing. DNN is used for feature extraction instead of classifying the tissue color type for an accurate measure of healing time. Apart from these, object detection has many applications in the health sector, such as, computer-aided diagnosis, image interpretation, image fusion, image registration, image segmentation [1], image-guided therapy, image retrieval and efficient representation of extracted information.

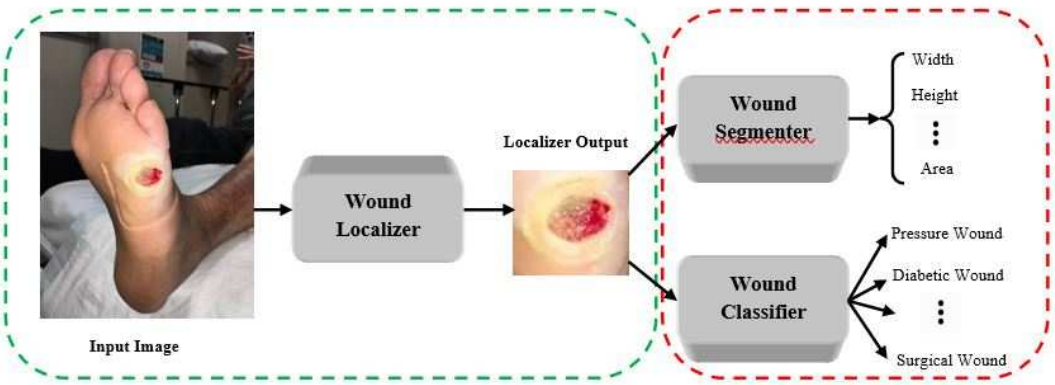


Figure 1: Overview of Wound detection system

The introduction of YOLOv3 object detection model helped medical experts to predict the locations of objects in the CT images, MRI images and 2D images of patients in a very short time. YOLOv3 is nearly as accurate as Single-shot MultiBox (SSD). Keeping this in mind, we have decided to use YOLOv3 for wound localization. Wound localization is the process of identifying the wound location in a given image. Once the wound is located, we can crop around the area bounded by the prediction box and use the cropped image for further classification, as shown in the figure 1.

YOLOv3 model uses deep learning algorithms for object detection [2]. It works by generating a one by one detection kernel on the featured map of three places of a given network. YOLOv3 uses multi-labeled classification such as independent logistic classifiers to act on complex datasets that have many overlapping labels. YOLOv3 can achieve close to 58% mAP compared to other forms of detection such as Deep Convolutional Neural of 53% and Retina Net of 51%. Its immensity, 30 frames per second, makes it one of the fastest object detection algorithms.

The goal of this project is to create a real-time and image-based software system for localizing, segmenting and classifying the wounds in images, running on mobile iOS

devices. In order to construct this system, we must also implement an object detection model suitable in terms of size and speed to run on a mobile device and detect wounds in real time as well as in static images. As the scope of this thesis, we will be implementing the stage one of this system, i.e. wound localizing.

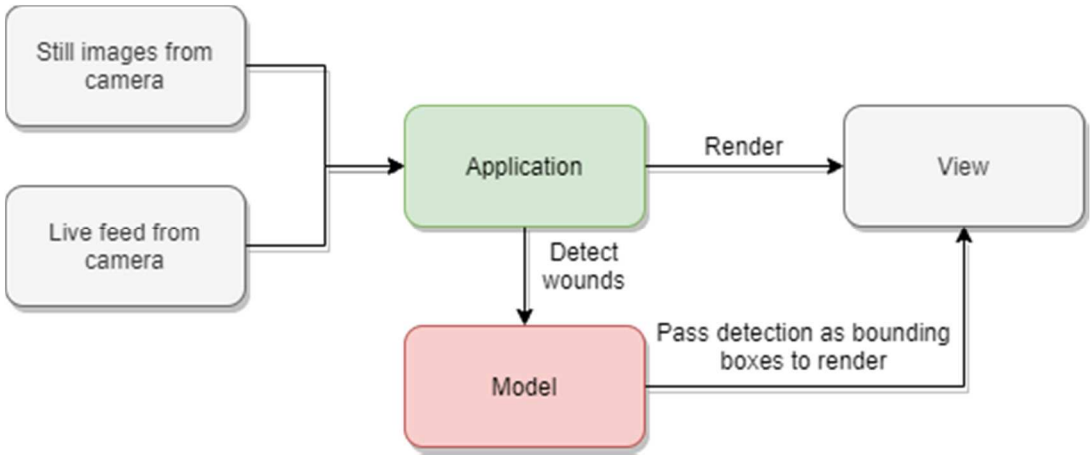


Figure 2 : A logical overview of the components of the application system

For the wound localization, the system architecture is developed. Logical components are found, and their interactions with each other can be seen in Figure 2. Required libraries and tools for the program are obtained. The machine learning process is also settled by defined main steps: Data collection, Data preparation (data labelling, verifying labels and data augmentation), and Model training and testing which will be discussed in detail in later chapters.

CHAPTER 2: Related Work

2.1 Overview

The terms Artificial Intelligence (AI), Machine Learning (ML), Deep Learning (DL) and Neural Nets (NN) can be found in numerous documents or articles. Since they are usually seen together people may think they are the same term. Hence, the following figure shows the actual difference between AL, ML and DL and NN.

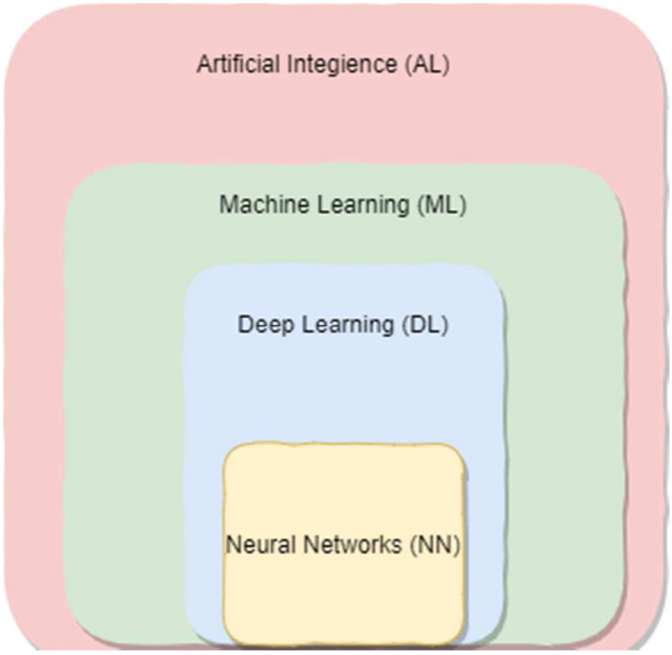


Figure 3: Artificial Intelligence, Machine Learning, and Deep Learning connections

Figure 3 shows the relation between all the four components of Computer Vision. Let us begin with an introduction to Machine Learning which includes DL and NNs.

2.2 Machine learning

Machine learning is the study of algorithms and models that can be utilized by machines to perform certain tasks effectively without guidance, relying on examples instead [4].

Machine learning models learn from data. This data consists of features and ground truths or labelled output. In image data, features can be pixels of an image. In some cases, these features are extracted or modified as per training needs. Training is the process of machine learning algorithms, where datasets are used to teach a model. Datasets are often split, as discussed later, into training, testing and validation sets. Test dataset is used to evaluate the model accuracy. Overfitting and underfitting are concepts where either model is unable to generalize on new data or is unable to generalize on training data [15].

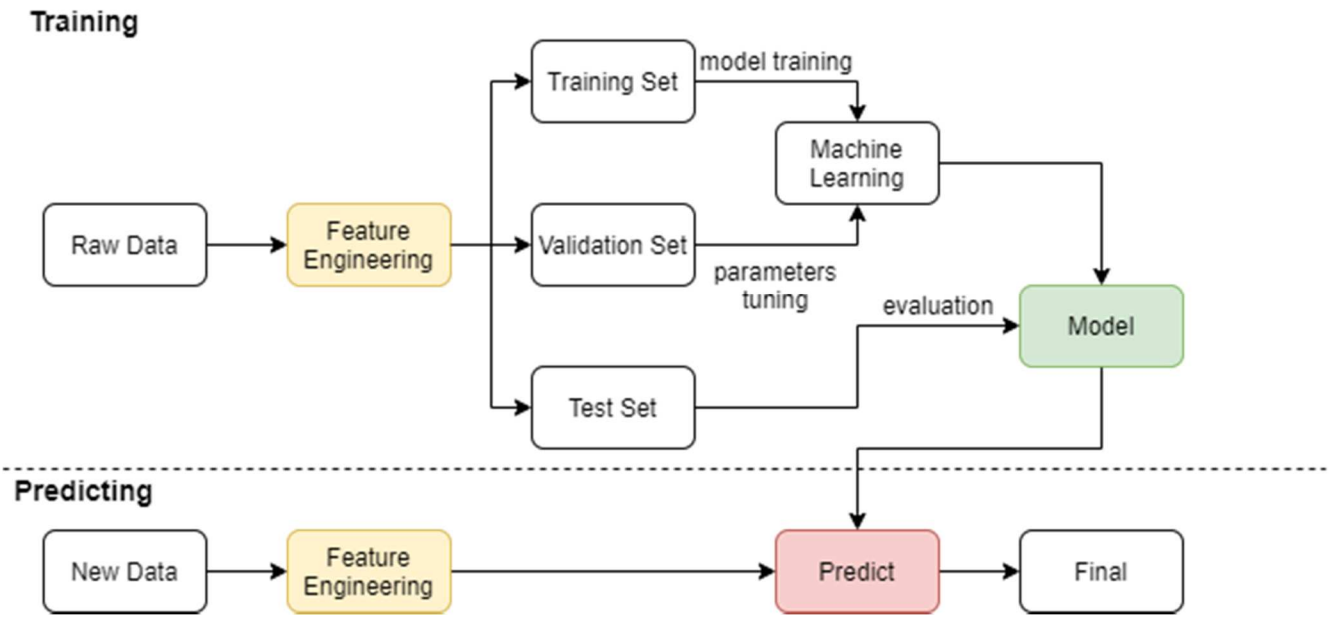


Figure 4: Machine learning process [28]

The most common methods for training are supervised learning and unsupervised learning.

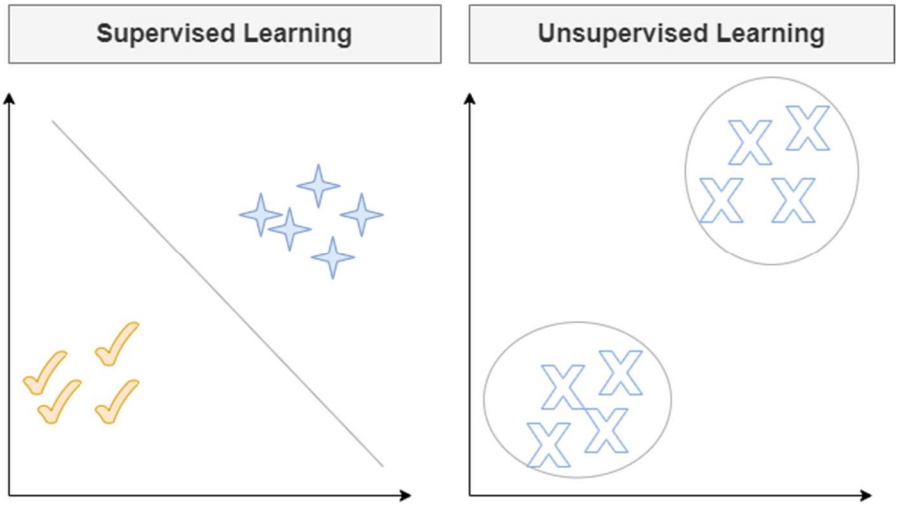


Figure 5: Supervised Learning vs Unsupervised Learning

Left graph in Figure 5 shows that if a model is trained with label data, it can classify the new data into one of the labelled categories. Right graph shows that if a model is trained on unlabeled data, it can create clusters of similar features from data and can tell us the difference between the data.

2.2.1 Supervised learning

Supervised learning is the task of concluding a function from labelled training data set. In supervised learning, the model is trained on a specific task using training data that consist of features vectors and labelled output. The training data consists of a set of examples [25]. Each example is a pair of features and labelled output. The model analyzes the training data and produces an inferred function, which can be utilized for mapping new sets of examples [25]. Classification is the best example of Supervised learning.

2.2.2 Unsupervised Learning

Unlike supervised learning, training data in unsupervised learning does not have labelled output values. Thus, the model decides the more relevant patterns in the training data and output what it thinks is important. This output is basically in the form of clusters or patterns. For example, output can be any kind of anomaly in given data. It can be financial record anomaly or high voltage cable anomaly or categorizing different groups of consumer customers for marketing purposes, etc.

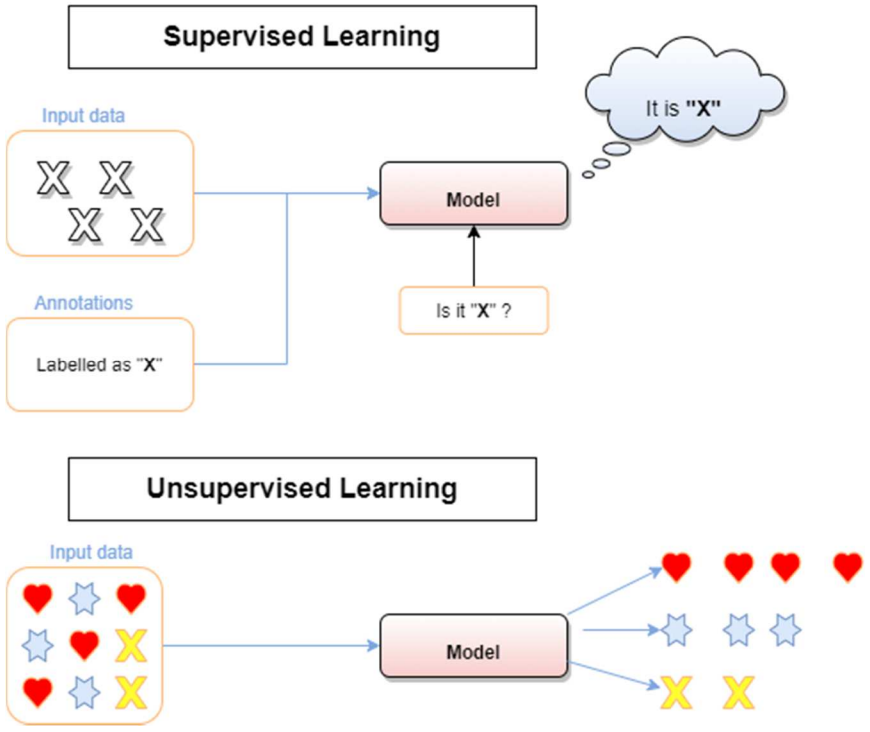


Figure 6: Supervised Learning vs Unsupervised Learning

The upper half in Figure 6 is based on supervised learning, where model is trained on labelled images and then asked to classify a new image. The lower half of Figure 6 is based on unsupervised learning, where model is trained on unlabeled data and is asked to draw an inference on what it sees.

2.3 Performance

To be able to evaluate the performance of the model a performance metric P is used. For classification problems, P might be a value that represents if the model classified the object correctly or not. Using this we can calculate the accuracy or error rate of the model. Every task will have a different performance metrics P. The detail of definitions for different P metrics will be given in later chapters when specific neural networks are presented.

2.4 Dataset

Dataset is a collection of data that is used for training a model. The data is usually split into two sets, called training data and testing data. Distribution for training data and testing data is generally 80% and 20%. The training data is used by the model to learn a specific task and the test data is used to verify how well the model performed on that task, given new or unseen data. Just splitting the data into training set and testing set can make it hard to deal with overfitting and underfitting, particularly when the dataset is small.

Overfitting happens when a model learns the features and noise in the training data to the degree that it contrarily impacts the exhibition of the model on new data. This implies that the noise or irregular changes in the training data is picked up and learned as a feature by the model. The issue is that these features do not make a difference to new data and adversely affect the models' capacity to sum up. Overfitting is practically certain with nonparametric and nonlinear models that have more adaptability when learning a target function. All things considered, numerous nonparametric machine learning

algorithms incorporate parameters or techniques to restrain and compel how much detail the model learns. Underfitting refers to a model that can neither model the training data nor sum up or generalize the new data [34]. Underfitting occurs because a model is too simple compared to the data it is trying to learn.

Overfitting and underfitting comes from the bias-variance tradeoff. This means that there exists a conflict when trying to reduce two error sources at the same time. As the name suggests, these two errors are called bias and variance. Bias is the error that describes bad assumptions made within the learning algorithm. High bias can mean the model has a hard time learning the important features of data set, in other words underfitting. Variance on the other hand, is the error that happens from small differences within the input compared to the training data. A high variance can mean that the model is extremely sensitive to noise, or in other words overfitting.

To deal with overfitting and underfitting, there are two important techniques namely,

1) Using a resampling technique to estimate model accuracy, or
2) Holding back a validation dataset

The k-fold cross validation is a popular resampling technique. It allows us to train and test the model k-times on different instances of training dataset and build up a performance estimation of the model on new data.

A validation data is solely a subset of training data that is preserved. This data is never used to train the model, but instead is used to check how well the model performs on unseen data after every training step. At each training step, the model is evaluated, and the most accurate model is saved.

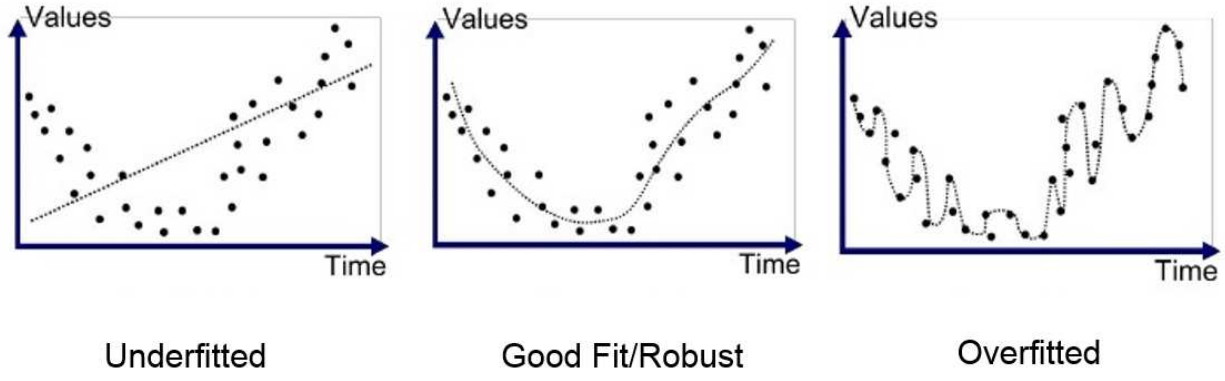


Figure 7: Underfitted, Good Fit and Overfitted [7]

2.5 Data Augmentation

Another way to deal with overfitting is by increasing the amount of data within a given dataset. This is called data augmentation. Data augmentation means adding new data to given dataset by changing some properties using techniques such as rotating, flipping, resizing, etc. For our project we have used 4 type of augmentation, rotation by 25 degree, flip horizontally, flip vertically and blurring as seen in following table.

Original		
Rotate (-25,25)		

Flip horizontally		
Flip vertically		
Blur		

Table 1: Augmentation types

2.6 Neural Networks

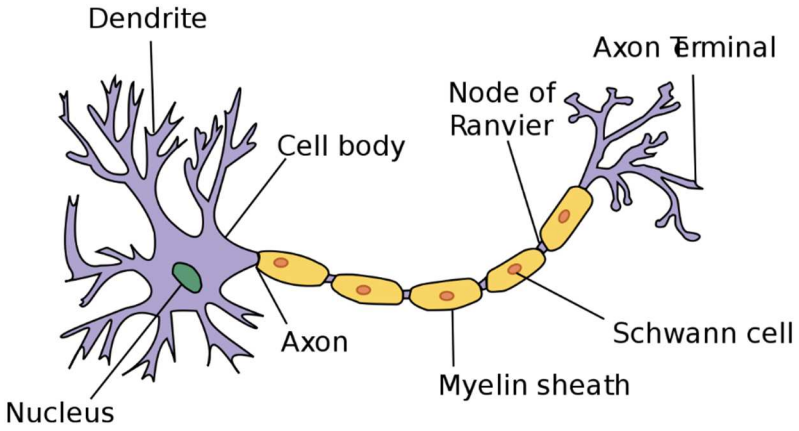


Figure 8: Neuron [10]

Neural networks get their name from the way they represent neurons in the human brain as shown in the figure 8. The way the human body works is still an enigma. However, we know information is passed along the human nervous system using neurons. Essentially, neurons help transmit information to and fro from the human brain to other muscles, cells and glands in the body. They send and receive messages from the neighboring neurons and carry the message to its destination. So, our body communicates and processes information in the form of a network.

2.6.1 Neural network structure

The perceptron was initially proposed by Minsky- Papert in 1969 [13].

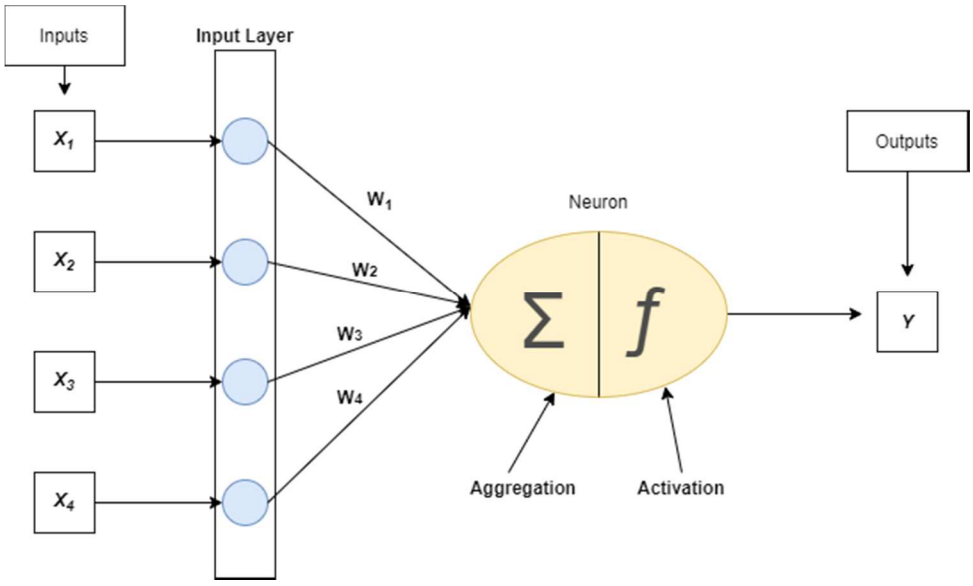


Figure 9: Architecture of a single neuron (Perceptron) [12]

Perceptron takes in some amount of inputs and returns a single output (figure 9). It performs three main steps [16]:

1. Take inputs vector x_i and multiply with their weights w_i,
2. Then sum them up,
3. Apply activation function f to the sum

The inputs are usually features from the data or they can be the output of another neuron. Each feature is assigned weights depending on how relevant it is to predict the target class. In some cases, we add biases too for acquiring better results. We use activation functions to help map input, weights and bias to a specific output. In terms of nervous system, activation functions choose whether a neuron will be fired or not, or whether it will be activated. Activation function can have values ranging from $-\infty$ to $+\infty$ and performs certain operations on it [17]. The most used activation functions are given in figure 10. We typically use non- linear functions as activation functions so that they can approximate any complex function. The accuracy of the neural network depends on how well it can assign the weights.

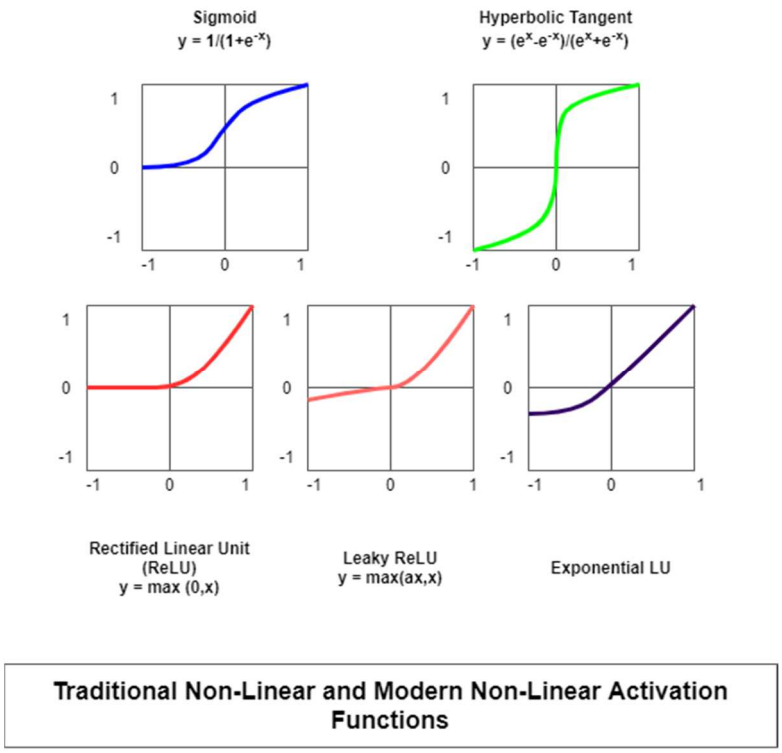


Figure 10: Commonly used activation functions [29]

Deep neural networks consist of multi-layered structures (as shown in figure 11) where each layer has a fixed number of neurons. The number of neurons in the first layer is equal to the size of input vectors and that of output layer equals the classes number [35]. As shown in figure 11, the connections between the neurons do not form cycles. This type of network is called feedforward neural network as the data always flows forward in the network.

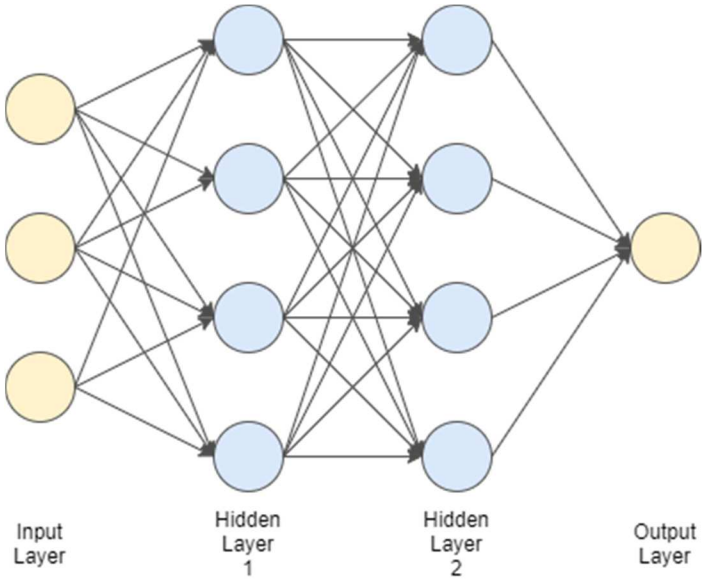


Figure 11: Neural network with 3 layers and 2 fully connected layers

For object detection, image pixels are used as features while training a model or making predictions. If we use fully connected neural networks, where every neuron in one layer is connected to all the neurons in the previous layer then, for a single image of size 512x512x3, a single neuron in the first fully connected layer will have 786,432 weights. This number will keep growing as the number of pixels increases and can results in overfitting [17]. To overcome such gigantic calculations, we need some better architecture that can bypass them.

Convolutional Neural Networks, commonly known as CNN, are a type of neutral networks. CNN consists of an input layer, some hidden layers and an output layer. The main difference between convolution neural network and artificial neural network is that CNN consists of at least one layer that uses convolution in place of normal matrix multiplication. Convolution allows the input size to vary. CNN layers are built on three facts, which are sparse interaction, parameter sharing and equivalent representation [21].

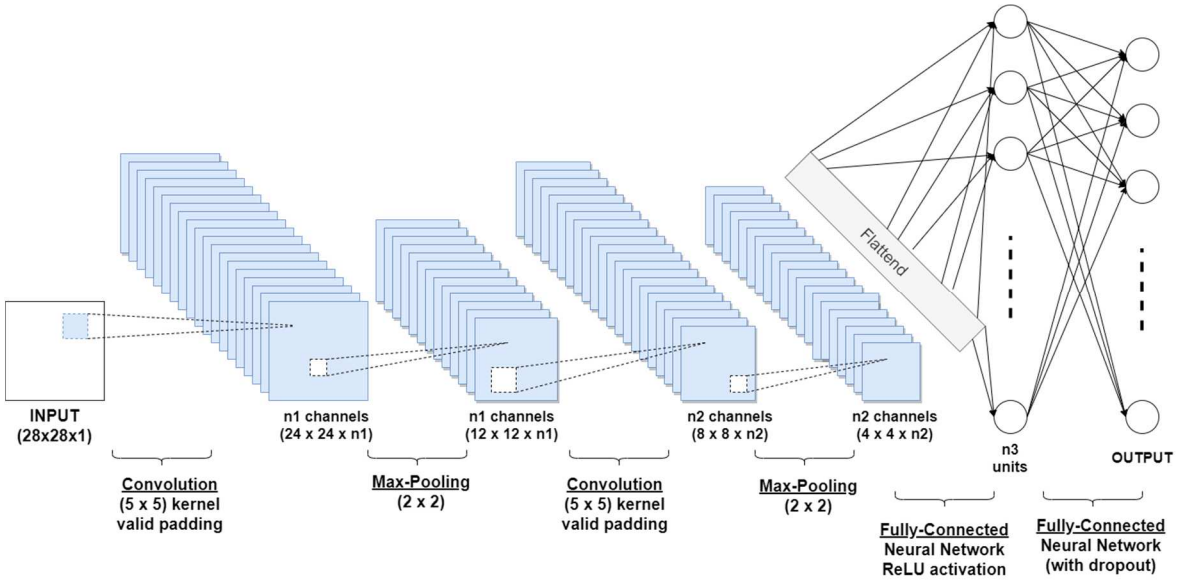


Figure 12: Convolution Neural Network Structure

A CNN model can take in an input image, assign weights and biases to various objects in the image and is able to differentiate one from other. The pre-processing cost required in CNN is much lower compared to other algorithms. While in primitive methods, filters are hand-engineered, with enough training, CNNs have the ability to learn these filters [18]. Figure 12 shows a CNN structure example for number classification. The operations performed are a series of convolution and pooling operations, followed by a few fully connected layers. If multiclass classification is to be done, then output is SoftMax.

2.7 Object Detection in real time

Real time object detection is one of the categories of computer vision where our aim is to detect and classify objects in real time. This suggests that the inference time of the deep learning models should be fast enough to handle input under real time constraints, such as video, whilst also maintaining a good accuracy.

Real time object detection has quite a short history because the hardware a few years ago was lacking the computational power to perform inference in real time. The paper Faster RCNN: Towards Real-Time Object Detection with Region Proposal Networks [19] showed that there was an opportunity to try and do inference fast enough to reach 15 fps. Some months after the first YOLO paper [19)] was released. YOLO was extremely fast and did achieve inference speeds to permit for 45 fps, while accomplishing a mAP value of 60. Real time object detectors were incrementally improved by SSD [23], YOLO 9000 [20] and YOLOv3 [22].

CHAPTER 3: Wound Detection with Deep Learning

3.1 Architecture

3.1.1 You Only Look Once (YOLOv3)

You only look once, or YOLO is one of the fast object detection algorithms. YOLOv3 was introduced by Joseph Redmon and Ali Farhadi in 2018 as an improvement for YOLO 9000. YOLOv3 predicts both bounding boxes and classifies the object within the bounding box in one pass. Bounding box are boxes that encompass one object; thus, every detected object will have a bounding box around it to indicate where it is. Old object detectors had to first locate the bounding box and then classify the object within the image. As the name suggests, YOLOv3 is the third generation of the YOLO family. YOLOv3 does prediction on the pre-frame basis and no temporal information is employed. YOLOv3 architecture consists of three different networks. The first one is Darknet-53 which is the backbone of YOLOv3 [22], second is upsampling and third is YOLO layers or detection layers.

3.1.2 Darknet-53

YOLOv2 used darknet-19 which suggests a 19-layer network augmented with 11 more layers for object detection. With only 30 layers, YOLOv2 struggled detecting smaller objects. This was the result of loss of fine-grained features as the layers downsampled the input. To overcome that YOLOv2 used feature maps from preceding layers to capture low rank features. YOLOv2 lacked residual blocks, skill connections and upsampling. But YOLOv3 includes all of them. YOLOv3 uses Darknet-53 which has 53 layers. So YOLOv3

consist of a total 106 layers fully convolutional. Due to its large size, YOLOv3 is slower compared to YOLOv2.

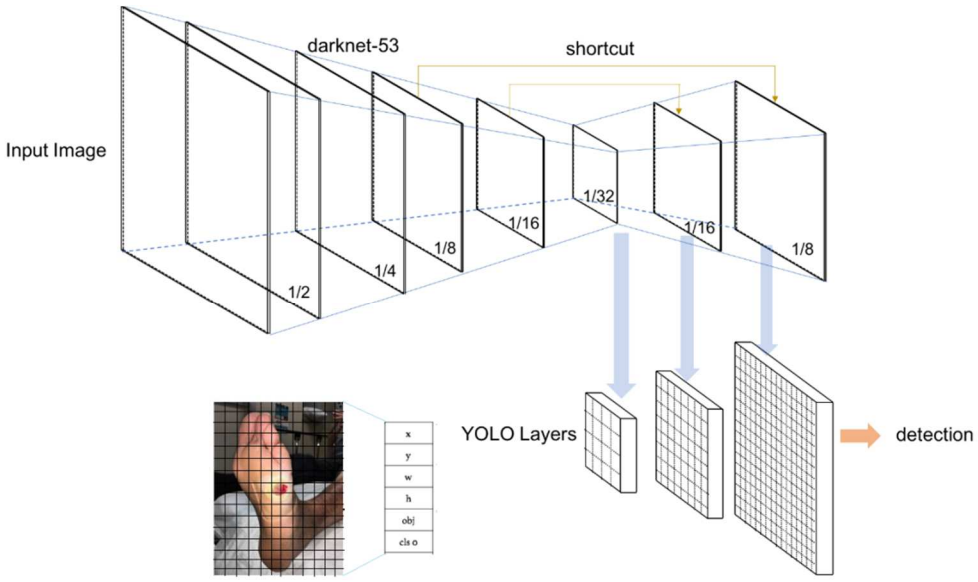


Figure 13: YOLOv3 network architecture [30]

The darknet-53 network is used to extract features from the input image. Darknet-53 consists of residual blocks as the basic component. Each residual block consists of a pair of 3 x 3 and 1x 1 convolutional layers together with shortcut connections. As the name suggests, there are 53 convolutional layers. A complete overview of Darknet-53 can be found in figure 13. The numbers below each layer are the dimensions by which the input is decreased.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 14: Darknet-53 Architecture [24]

The YOLO layers are responsible for detecting objects at different scales using features extracted by Darknet-53 layers. So, at the initial YOLO layer the grid size is 1/32 of the input image size. The final YOLO layer the grid size is 1/8 of the input image size. The benefit of three YOLO layers is that smaller objects are now detected. Each YOLO layer consists of a few convolution layers with batch normalization and leaky ReLU activation. There are shortcut connections that connects darknet-53 intermediate layers to the layer after upsampling layer.

Due to three different scales, the output of the network is 1 x 1 [3 x (4 + 1 + N)] tensor, where N is the number of classes, ‘4’ indicates the bounding box attributes and inner ‘1’ indicates object confidence. YOLOv3 uses an independent logistic classifier to find the class probability of detected objects. In YOLOv3, there is no more softmaxing the classes. Softmaxing classes assumes that classes are mutually exclusive meaning if an object belongs to at least one class then it cannot belong to a different one [26]. So, when classes like Person and Women are there in dataset, then the softmaxing fails. Hence, authors of YOLO have refrained from softmaxing the classes. Instead logistic regression is used to predict the class score and a threshold is used to predict multilabel classes.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2$$

$$+ \sum_{i=0}^{S^2} 1_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Figure 15: Loss function [19]

3.1.3 Loss function

YOLOv3 loss function consists of bounding box error, confidence error and classification error. The bounding box error is nothing but the difference between predicted box and ground truth. The confidence error predicts whether the bounding box has object or not. The classification error describes incorrect prediction of classes inside bounding boxes. Earlier, YOLOv2 loss function looked like shown in Figure 15

We can see that the last three terms of the loss function are squared error terms in YOLOv2. In YOLOv3 these squared error terms are replaced by cross-entropy error terms [26]. That is, logistic regression is used to predict class and object confidence. The loss function uses λ_{coord} and λ_{noobj}, where λ_{coords} is used to increase the loss of the bounding box coordinate predictions and λ_{noobj} is used to decrease the loss of the predicted boxes containing no objects.

3.1.4 Performance Metrics

Performance metrics is used to determine the model performance over a given dataset. Performance metrics include IOU, Recall and Precision, mAP and F1 value.

3.1.4.1 Intersection Over Union (IOU)

Intersection over union is a metric that evaluates the overlapping of two bounding boxes as shown in Figure 16. One box is the predicted bounding box and the other one is ground truth.

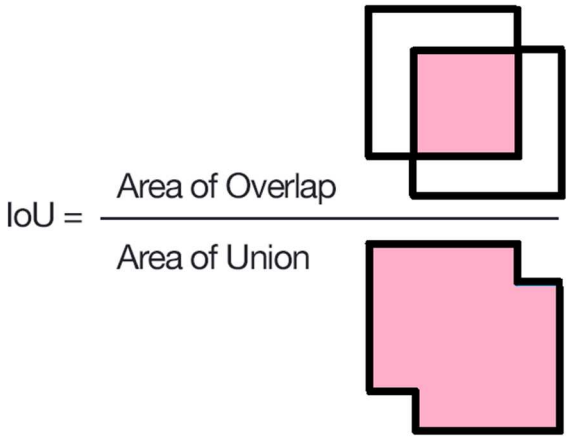


Figure 16: Intersection over Union

We predefine a threshold value for IOU = 0.5 which is the most used value.

If IOU > 0.5 then it is True Positive,

If IOU < 0.5 then it is False positive,

If IOU > 0.5 and object is wrongly classified, then it is False Negative

3.1.4.2 Recall & Precision

Precision is the proportion of information that model predicted as relevant are actually relevant. Recall is the ability of the model to find all the relevant information from the given dataset. Calculation of precision and recall is shown in the Figure 17. Accuracy is the measure of truly predicted values over total value.

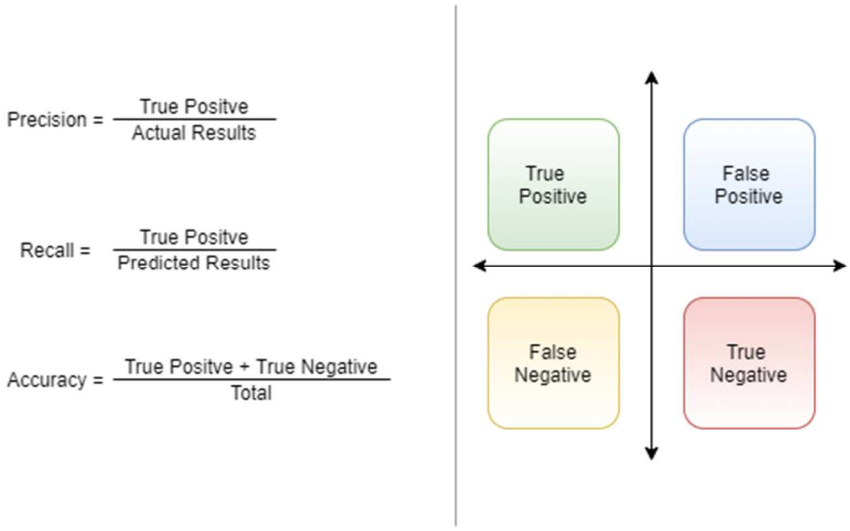


Figure 17: Recall & Precision

3.1.4.3 Mean Average Precision (mAP)

Mean average precision is a metric that is used to compare different object detectors over multiple datasets. To calculate mAP, we need to know the Interpolated Precision [27]. Interpolated Precision is simply the highest precision value for a specific recall value. For example, if recall value is 0.4 for two precision values 0.92 and 0.89 then interpolated precision for both recall values will be highest precision value i.e. 0.92. Interpolated Precision is calculated as follow,

$$P_{interp}(r) = \max_{r' \geq r} p(r')$$

Finally, to calculate mAP, we will take summation of interpolated precision values,

$$mAP = \sum_{r=0}^{1} (r_n - r_{n-1})\, P_{internp}(r_n)$$

3.1.4.4 F1 Score

F1 score is the weighted average of precision and recall [27]. If, F1 score is higher, then the model is considered to perform better.

$$F1 = 2\,x\,\frac{precision\,x\,recall}{precision + recall}$$

3.1.5 Model Training

Ultralytics and Lufficc are the GitHub repository developed by Glenn Jocher and Congcong Li respectively. These repositories are available at [31,32] and are designed for object detection using PASCAL VOC dataset [33]. We customized the code for our wound detection application.

For training, we have used a single class, named “wound” for our model training. For YOLOv3 model, we have used the YOLO annotations. This model is trained for 273 epochs with 8 batch size. YOLOv3 model is trained with a 0.001 learning rate, and stochastic gradient descent (SGD) optimizer has been used. We have used the YOLOv3-416 model for wound bounding box detection. In SSD, we have used the Pascal VOC annotations. The YOLO annotations are converted to Pascal VOC format for the training of this model. The SSD model is trained for 475 epochs with a batch size of 8. For SSD we have used stochastic gradient descent (SGD) optimizer and the learning rate parameter is set to 0.001. We have used the SSD300 model for our detection. These models are written in Python programming language by using the Pytorch deep learning

framework. The models are trained on an Nvidia GeForce RTX 2080Ti GPU platform. These models are trained in the “Big Data Analytics and Visualization Lab” server.

3.2 Result and Discussion

3.2.1 Data Collection

The wound dataset has been collected from AZH Wound and Vascular Center, Milwaukee, WI, USA. This dataset contains a total of 1010 wound images. Three types of ulcers have been included in the dataset: Diabetic foot ulcer (DFU), Pressure Ulcer (PU), and Venous Ulcer (VU). All the images are captured with iPad and DSLR camera. No specific environmental or illumination condition has been applied on image capturing. These images are further processed and used as training and test data. For testing the robustness and reliability of our models, 56 images have been collected from Medetec Wound Database [7]. Though this database contains all types of open wound images such as abdominal wounds, burn and scalds, diabetic foot ulcers, haemangiomas, venous ulcers and arterial ulcers, malignant wounds etc.; we have only collected diabetic foot ulcer, venous ulcer and pressure ulcer images.

3.2.2 Data Preparation

As our models can take different width-height ratio of images as training and test input; we do not make the image size uniform. To increase the number of images for our dataset, we have applied some augmentations. On the AZH Wound and Vascular Center dataset, we have applied rotation, flipping (up and right), and blurring augmentations. After augmentation we have a total of 4050 image data. We have used 3645 images as training dataset, 405 images are used as test dataset. All the collected images have been labeled manually for training and evaluation of our models. We have used a MIT licensed

free graphical image annotation tool, named labelImg [12] for our data labeling. Annotations are saved as YOLO format, which produce a text file for each image; containing the class number, center coordinates of bounding box(s), and height and width of bounding box(s). Annotations are further changed to Pascal VOC format, according to models need. These images and annotations are passed as the inputs of our models.

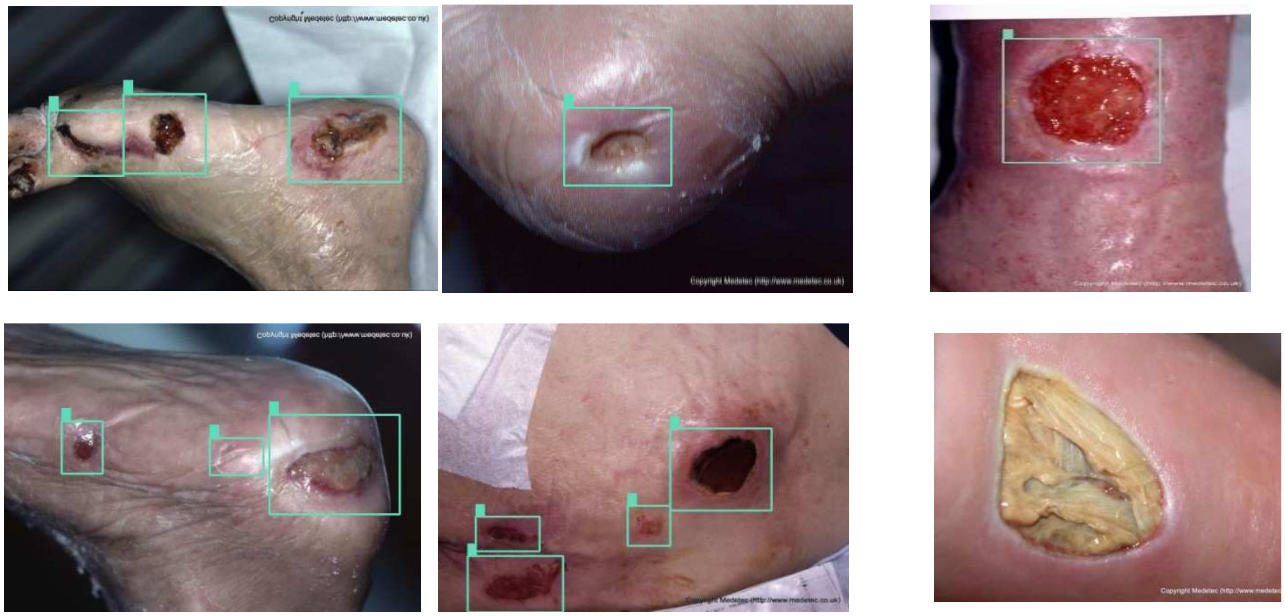
3.2.3 Results

We have tested the performance of both of our models with the test set that contains 405 wound images. For YOLOv3, with IoU of 0.5 and non-maximum suppression of 1.00, we get the mAP value of 0.939. The precision, recall and f1 score of this model is 0.925, 0.905, and 0.915 respectively. In SSD, by using an IoU of 0.5 and non-maximum suppression of 0.45, we get the mAP value of 0.864. The precision, recall and f1 score of SSD model is 0.902, 0.584, and 0.709 respectively. For mobile application, we have also implemented a light version of YOLOv3 (with reduced convolutional layers), named tiny-YOLOv3. With IoU threshold set to 0.5, we get the mAP value of 0.926 for tiny-YOLOv3. The precision, recall and f1 score of this model is 0.902, 0.899, and 0.9 respectively. Table 2 shows a brief summary of our evaluation results.

	Precision	Recall	F1-Score	mAP
YOLOv3	0.925	0.905	0.915	0.939
SSD	0.902	0.584	0.709	0.864
Tiny-YOLOv3	0.902	0.899	0.9	0.926

Table 2: Result Summary

The robustness and reliability testing on Medetec dataset shows very promising result. Some of the testing outputs with both models are shown in Figure 18.



(a) YOLOv3 output

(b) SSD output

Figure 18: Robustness and Reliability testing

3.2.4 Discussion

From the findings of the result section, it is clear that YOLOv3 gives us the best results. The mAP value is relatively much higher than SSD model with a difference of 7.5%. All the evaluation matrices (precision, recall and f1-score) reflect better value for YOLOv3 than SSD model. From table 1, we can see that SSD reflects a low recall and high precision, which leads to the decision that SSD is a very picky or fault-finding model. All the images it thinks are wounds - are really “wounds”, but it also misses a lot of actual wounds. In the other hand, from table 1 we can also see that YOLOv3 has a high precision (0.925) and high recall (0.905) value, which indicates a better and stable model.

Table 2 also indicates that tiny-YOLOv3 gives very good result, considering its speed. All the mAP, recall, precision, and f1-score values are better than SSD, and close to YOLOv3 results. So for mobile platform, keeping the lightness and speed of tiny-YOLOv3 in mind; we can consider the small differences in results: precision (2.3%), recall (0.6%), f1-score (1.5%), and mAP (1.3%).

In the robustness and reliability test, from Figure 18 we can see that YOLOv3 reflects good results than SSD model. In Figure 18, from (a) – 1, 4, 5 and (b) – 1, 4, 5 we can see that SSD model misses some wounds. It is dangerous for our localization task, as we are going to pass these cropped bounding boxes to the wound segmenter and wound classifier. We can also see from (a) – 6 and (b) – 6 that YOLOv3 completely miss the wound, but SSD captures it; but it is a less dangerous case than the previous one, as we are going to pass the whole image to segmenter and classifier if we do not find a bounding box. Passing a whole image is better than passing partial wounds from an image. From

(a) – 2, 3 and (b) – 2, 3, we can see that, both models do good job; but SSD captures slightly more healthy skins than YOLOv3. So, we can confidently say that, YOLOv3 does a god job for wound localization than SDD model.

Chapter 4: Mobile Application

4.1 Overview

The system is in the form of an iOS application. At the start of the application, the user is greeted with a login interface. Once the user is authenticated, application opens the live camera feed view showing whatever is in view of the camera at a given moment as demonstrated in Figure 19. To detect objects (wound ROI), we have used a deep convolutional neural network with an architecture called YOLOv3. In this section, we have demonstrated how to work in iOS with neural networks using the Core ML framework, and how to use and process the outputs of this network. Later we will discuss how this application can be improvised for further analysis of the wound. For this application, we have used our model that is trained on around 4000 images. Threshold IoU and object confidence is set to 0.5 and 0.2 respectively. Images shown below Figure 20, are the screenshot of iOS application detecting wound ROI. This code follows the implementation from Moonl1ght [5].

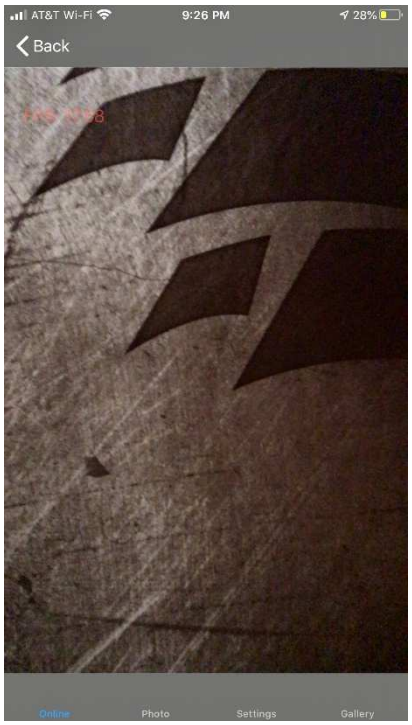


Figure 19: The live camera feed view

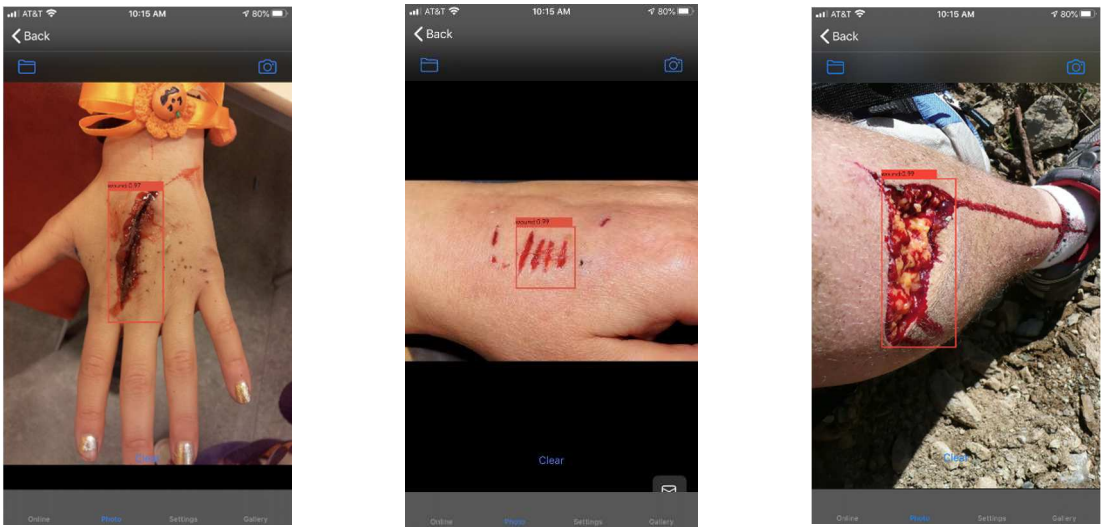


Figure **20: three examples of wound ROI**

4.2 Core ML Framework

Core ML is an Apple-developed machine-learning system. Core ML is the basis for the features and domain-specific frameworks. Core ML supports Vision for image analysis, Natural language for text processing, and more. Core ML builds on top of low-level primitives such as Accelerate and BNNS as well as Metal Quality Shaders themselves. [4]. It is available for iOS 11 and above. Core ML framework provides a unified representation for all models. Before we are able to integrate the model into the application with Core ML, a conversion to the Core ML format is important. Core ML format can only be used within the Apple system. By leveraging the CPU, GPU, and Neural Engine, Core ML optimizes on-device performance while reducing its memory footprint and power consumption. Running a model strictly on the device of the user eliminates the need for a network connection which helps to keep the data of the user private and sensitive to the app.

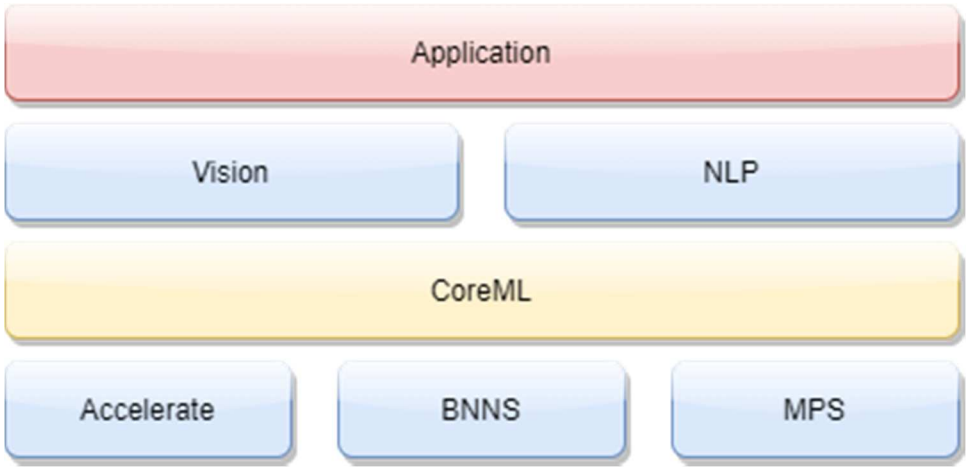


Figure 21: Core ML stack [4]

Core ML platform is a versatile platform for machine learning, but it comes with straight-forward use. To start implementing the machine learning framework, a Core ML model

file needs to be added to the project. The framework will generate the model class, which includes all utilities which help to predict the output with selected input.

4.3 Implementation

4.3.1 Conversion

There's a wonderful Core ML library on iOS, starting with version 11.0, that lets you run machine learning models directly on the computer. But there's one limitation: Only a mobile system running iOS 11 and above can run the program.

The problem is that Core ML only understands the Core ML model's unique format. For most common libraries, such as Tensorflow, Keras, or XGBoost, direct conversion to Core ML format is possible. Yet there is no such chance in Darknet. You may use different options to migrate the saved and qualified model from Darknet to Core ML, such as saving Darknet to ONNX, and then converting it from ONNX to Core ML.

We can find a simplified approach and use YOLOv3 implementing Keras. The action algorithm is this: load the Darknet weights into the Keras model, save it in the Keras format, and then convert it directly to Core ML. Since we are using pytorch to train the YOLOv3 model, we first convert pytorch model to Darknet weights and latter follows.

```
python convert.py yolov3.cfg yolov3.weights yolo.h5

import coremltools
coreml_model = coremltools.converters.keras.convert(
    'yolo.h5',
    input_names='image',
    image_input_names='image',
    input_name_shape_dict={'image': [None, 416, 416, 3]},
    image_scale=1/255.)

coreml_model.input_description['image'] = 'Input image'

coreml_model.save('yolo.mlmodel')
```

Figure 22: Model Conversion [5]

Our application has two main functionalities. They are:

1) Permitting the user to take a picture utilizing the camera of the device and detecting the ROI in the taken picture.
2) Permitting the user to detect ROI in a live video feed.

These requirements are satisfied by three UIViewControllers, namely: OnlineViewController, PhotoViewController and SettingsViewController. The first is the online ROI detection for each frame. You may take a picture in the second, or pick a picture from the list, and check the network on those images. The third one contains the settings, you can choose the model YOLOv3 or YOLOv3-tiny, as well as the thresholds.

4.3.2 Loading model into Core ML

After converting the trained model from Darknet format to Core ML, we've got a file with the .mlmodel extension. In my case, I built two files for the models YOLOv3 and YOLOv3-tiny, respectively: yolo.mlmodel and yolo-tiny.mlmodel. Now in Xcode you can load those files into a project. We build the ModelProvider class; it will store the current model and methods used to invoke the neural network asynchronously for execution. YOLO class is primarily responsible for loading files from .mlmodel and managing outputs from model.

```
private func loadModel(type: YOLOType) {
  do {
    self.model = try YOLO(type: type)
  } catch {
    assertionFailure("error creating model")
  }
}

var url: URL? = nil
  self.type = type
  switch type {
  case .v3_Tiny:
    url = Bundle.main.url(forResource: "yolo-tiny", withExtension:"mlmodelc")
    self.anchors = tiny_anchors
  case .v3_416:
    url = Bundle.main.url(forResource: "yolo", withExtension:"mlmodelc")
    self.anchors = anchors_416
  }
  guard let modelURL = url else {
    throw YOLOError.modelFileNotFound
  }
  do {
    model = try MLModel(contentsOf: modelURL)
  } catch let error {
    print(error)
    throw YOLOError.modelCreationError
  }
```

Figure 23: ModelProvider and YOLO class [5]

4.3.3 Processing the outputs of a neural network

If you select a model file in Xcode, you can see what they are, and see the output layers.

Name	Type	Description
▼ Inputs		
image	Image (Color 416 x 416)	Input image
▼ Outputs		
output1	MultiArray (Double 18 x 13 x 13)	
output2	MultiArray (Double 18 x	Show or hide Code Review

Figure 24: Yolo-tiny model input and output layers

Name	Type	Description
▼ Inputs		
image	Image (Color 416 x 416)	Input image
▼ Outputs		
output1	MultiArray (Double 18 x 13 x 13)	The 13x13 grid (Scale1)
output2	MultiArray (Double 18 x 26 x 26)	The 26x26 grid (Scale2)
output3	MultiArray (Double 18 x 52 x 52)	The 52x52 grid (Scale3)

Figure 25: Yolov3 input and output layers

As you can see in the image above, we have three for YOLOv3-416 and two for output layers YOLOv3-tiny in each of which bounding boxes for various objects are predicted. The YOLOv3 model uses three layers as output to break the image into a separate row, these grids 'cell sizes have the following values: 8, 16, and 32. Suppose we have an image in the input size of 416x416 pixels, so the output matrices will be 52x52, 26x26 and 13x13 (416/8 = 52, 416/16 = 26 and 416/32 = 13). In the case of YOLO-tiny, all is the same, but we have two in place of three grids: 16 and 32, that is, 26x26 and 13x13 dimensional matrices. After the loaded Core ML model is started, we get two (or three) MLMultiArray class objects on the display. And if you look at the shape properties of these objects, we can see the following for YOLO-tiny,

[1, 1, 18, 26, 26] and [1, 1, 18, 13, 13]

The dimensions of the matrices are predicted to be 26x26 and 13x13. But what does the number 18 mean? As already mentioned, the output layers are matrices 52x52, 26x26 and 13x13. The fact is that not every element of that matrix is a number, it's a vector.

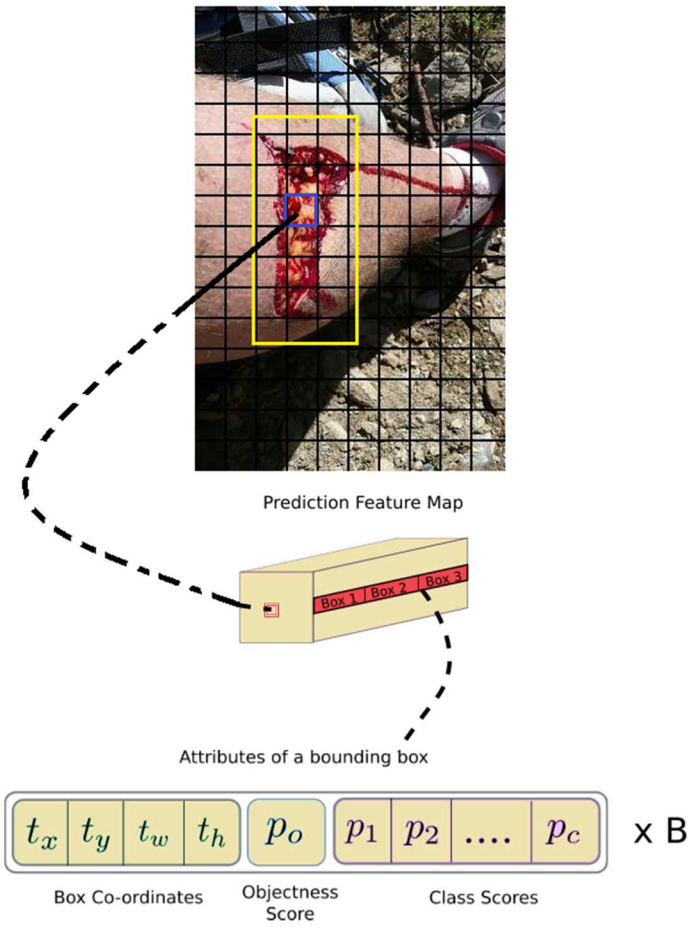


Figure 26: Feature Map

That is, three-dimensional matrix is the output sheet. This vector has dimensions B x (5 + C), where B is the number of bounding boxes in the cell, C is class number. Where does number 5 come from? The explanation is this: the probability of an object (object confidence) being predicted for each box is one number, and the remaining four are x, y, the width and height of the predicted box. For our network trained we had 1 class. So using the above formula B x (5 + C) will be 3 x (5 + 1) = 18.

Once we received the coordinates and sizes of bounding boxes and the corresponding probabilities for all found objects in the image, we can start drawing them on top of the image. But there occurs a problem when one entity or object is expected to have many

boxes with very large probabilities. We use simple algorithm called Non maximum suppression in that case.

The algorithm is as follows:

- We are finding a boundung box with the highest probability of belonging to the object.
- We run through all the bounding boxes which belong to this object as well.
- We remove them if the intersection of the first bounding box over Union (IoU) is greater than the defined threshold. Working directly with the results of neural network prediction can then be considered complete.

```
static func IOU(a: CGRect, b: CGRect) -> Float {
    let areaA = a.width * a.height
    if areaA <= 0 { return 0 }
    let areaB = b.width * b.height
    if areaB <= 0 { return 0 }
    let intersection = a.intersection(b)
    let intersectionArea = intersection.width * intersection.height
    return Float(intersectionArea / (areaA + areaB - intersectionArea))
  }

  private func nonMaxSuppression(boxes: inout [Prediction], threshold: Float) {
    var i = 0
    while i < boxes.count {
      var j = i + 1
      while j < boxes.count {
        let iou = YOLO.IOU(a: boxes[i].rect, b: boxes[j].rect)
        if iou > threshold {
          if boxes[i].score > boxes[j].score {
            if boxes[i].classIndex == boxes[j].classIndex {
              boxes.remove(at: j)
            } else {
              j += 1
            }
          } else {
            if boxes[i].classIndex == boxes[j].classIndex {
              boxes.remove(at: i)
              j = i + 1
            } else {
              j += 1
            }
          }
        } else {
          j += 1
        }
      }
      i += 1
    }
  }
```

Figure 27: IoU and NMS [5]

Conclusion

This work focused on the development of an automatic localizer for wounds. Wound localizer is the first step in developing an intelligent wound healing system, since the localizer output would be the input of future wound processing systems, such as wound segmenter and wound classifier. As already stated, all previous works used the localized wound by manually extracting it from the original image. And as the first automatic wound localizer, this tool is of great significance for future wound healing studies. With a maximum mAP value of 0.94 our system performs well. We also broadened the reach of our work further by developing a mobile channel for it. It will benefit patients who have no access to specialist wound care remotely, which will also significantly reduce the cost of treatment. We are progressing towards the development of a full automated wound care program, with our team working on the task of wound segmentation and classification using the found wound patches prepared by this experiment.

Reference

1) Latif, J., Xiao, C., Imran, A., & Tu, S. (2019, January). Medical Imaging using Machine Learning and Deep Learning Algorithms: A Review. In 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET) (pp. 1-5). IEEE.
2) Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., & Gao, R. X. (2019). Deep learning and its applications to machine health monitoring. Mechanical Systems and Signal Processing, 115, 213-237
3) Brownlee, J. (2019, July 5). A Gentle Introduction to Object Recognition With Deep Learning. Retrieved January 1, 2020, from https://machinelearningmastery.com/object-recognition-with-deep-learning/
4) Core ML. Apple developer framework. Retrieved March 5, 2020, from https://developer.apple.com/documentation/coreml
5) moonl1ght, A. (2019, July 25). moonl1ght/iOS-ObjectDetection. Retrieved March 20, 2020, from https://github.com/moonl1ght/iOS-ObjectDetection
6) Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016, Retrieved March 20, 2020, from. http://www.deeplearningbook.org.
7) Thomas, S. (2014, February 6). Medetec Wound Database: stock pictures of wounds. Retrieved March 3, 2020, from http://www.medetec.co.uk/files/medetec-image-databases.html
8) Thomas, S. (2014, February 6). Medetec Wound Database: stock pictures of wounds. Retrieved March 3, 2020, from http://www.medetec.co.uk/files/medetec-image-databases.html
9) Marr, B. (2018, September 24). What Are Artificial Neural Networks - A Simple Explanation For Absolutely Anyone. Retrieved March 20, 2020, from https://www.forbes.com/sites/bernardmarr/2018/09/24/what-are-artificial-neural-networks-a-simple-explanation-for-absolutely-anyone/
10) Wikipedia contributors, Neuron. Retrieved March 20, 2020, from https://simple.wikipedia.org/w/index.php?title=Neuron&oldid=6834394
11) Dertat, A. (2017, October 9). Applied Deep Learning - Part 1: Artificial Neural Networks. Retrieved March 20, 2020, from https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

12) Tzutalin. (2020, January 30). tzutalin/labelImg. Retrieved March 20, 2020, from https://github.com/tzutalin/labelImg

13) Marvin L. Minsky and Seymour A. Papert. Perceptrons: Expanded Edition. MIT Press, Cambridge, MA, USA, 1988

14) Contributors to Wikimedia projects, “Machine learning - Wikipedia,” Wikimedia Foundation, Inc., Retrieved March 20, 2020. from https://en.wikipedia.org/wiki/Machine_learning

15) “Machine Learning Glossary | Google Developers,” Google Developers. Retrieved March 20, 2020, from https://developers.google.com/machine-learning/glossary/.

16) Obuchowski, A. (2019, March 26). Understanding neural networks 1: The concept of neurons. Retrieved March 20, 2020, from https://becominghuman.ai/understanding-neural-networks-1-the-concept-of-neurons-287be36d40f

17) CS231n Convolutional Neural Networks for Visual Recognition, Retrieved February 3, 2020, from http://cs231n.github.io/neural-networks-1/

18) Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. Retrieved March 15, 2020, from http://www.deeplearningbook.org.

19) Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 779–788, 2016

20) J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6517–6525, July 2017

21) Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 779–788, 2016.

22) Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. arXiv, 2018.

23) Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In ECCV, 2016.

24) Zhang, Mingjiang & Zhao, Weihu & Li, Hongwei & Wang, Feng & Zhang, Shuai. (2019). Research on the Application of Deep Learning YOLOv3 in Aerial Patrol Inspection of Optical Cable Lines. Journal of Physics: Conference Series. 1345. 032068. 10.1088/1742-6596/1345/3/032068.

25) Wikipedia contributors, "Supervised learning," Wikipedia, The Free Encyclopedia, Retrieved from March 20, 2020, from https://en.wikipedia.org/w/index.php?title=Supervised_learning&oldid=940235979

26) Kathuria, A. (2018, April 29). What's new in YOLO v3? Retrieved March 20, 2020, from https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

27) Joshi, R. (2016, November 11). Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures. Retrieved March 20, 2020, from https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/

28) Cdiscount Data Science, A brief overview of Automatic Machine Learning solutions (AutoML), Hacker Noon, 25-May-2018. Retrieved March 20, 2020, from https://hackernoon.com/a-brief-overview-of-automatic-machine-learning-solutions-auto ml-2826c7807a2a.

29) Sze, Vivienne & Chen, Yu-Hsin & Yang, Tien-Ju & Emer, Joel. (2017). Efficient Processing of Deep Neural Networks: A Tutorial and Survey. Proceedings of the IEEE.105.10.1109/JPROC.2017.2761740.

30) Honda, H. (2019, July 10). Reproducing training performance of YOLOv3 in PyTorch (Part1). Retrieved March 20, 2020, from https://medium.com/@hirotoschwert/reproducing-training-performance-of-yolov3-in-pytorch-part1-620140ad71d3

31) Glenn Jocher, guigarfr, perry0418, Ttayu, Josh Veitch-Michaelis, Gabriel Bianconi, … IlyaOvodov. (2019, April 24). ultralytics/yolov3: Rectangular Inference, Conv2d + Batchnorm2d Layer Fusion (Version v6). Zenodo. https://github.com/ultralytics/yolov3/tree/v6

32) Congcong Li, High quality, fast, modular reference implementation of SSD in Pytorch, 2018, https://github.com/lufficc/SSD

33) Pascal VOC. (n.d.). Retrieved March 20, 2020, from http://host.robots.ox.ac.uk/pascal/VOC/

34) Brownlee, J. (2019, August 12). Overfitting and Underfitting With Machine Learning Algorithms. Retrieved March 20, 2020, from https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

35) Turesson, Hjalmar, et al. “Machine Learning Algorithms for Automatic Classification of Marmoset Vocalizations.” PLoS One, vol. 11, no. 9, Public Library of Science, Sept. 2016, p. e0163041.