



**INTELLIGENT VEHICLE DAMAGE  
ASSESSMENT AND COST  
ESTIMATOR FOR INSURANCE COMPANY APPLIED IN  
ARTIFICIAL INTELLIGENCE**

**A PROJECT REPORT**

*Submitted by*

**PRANAV R (192071062)**

**RAJENDRAN V (192071068)**

**RAHUL K (192071067)**

**SANDEEP KUMAR R (192071076)**

*In partial fulfilment for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**IN**

**INFORMATION TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**MAHENDRA ENGINEERING COLLEGE**

**(AUTONOMOUS)**

**Mahendhirapuri, Mallasamudram, Namakkal-637 503.**

**NOVEMBER - 2022**

## **ABSTRACT**

Vehicle damage detection is one of the important prime activities in the insurance and vehicle rental industries. These kinds of systems are widely used by the driver and also by the insurance company to identify the damage of the vehicle. Once an accident happens and in order to detect and determine a suitable appraisal as per the damage and for vehicle rental companies to assign the damage of the vehicle to a guilty customer. The core technique of this system is object recognition, and this presents a novel approach to measure the vehicle body damage and to make a cost prediction using 2D images. The model will show how much the damage is done to vehicle and cost estimator for the damaged vehicles. Using our model we can predict the car model and analysis the damage with more accurate result.

# TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	<b>ABSTRACT</b>	<b>v</b>
	<b>LIST OF FIGURES</b>	<b>viii</b>
	<b>LIST OF ABBREVIATION</b>	<b>x</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 Overview	1
<b>2</b>	<b>SYSTEM STUDY</b>	<b>3</b>
	2.1 Convolutional Neural Network	3
	2.2 You Only Look Once	7
<b>3</b>	<b>SYSTEM ANALYSIS</b>	<b>11</b>
	3.1 Existing System	11
	3.2 Drawbacks of Existing System	11
	3.3 Proposed System	12
	3.4 Advantages of Proposed System	14
<b>4</b>	<b>SYSTEM SPECIFICATIONS</b>	<b>15</b>
	4.1 Hardware Requirements	15
	4.2 Software Requirements	15
<b>5</b>	<b>SOFTWARE DESCRIPTION</b>	<b>16</b>
	5.1 <b>PYTHON DISTRIBUTION</b>	<b>16</b>
	5.1.1 Anaconda Distribution	16

	5.1.2 Anaconda Navigator	17
	5.1.3 Jupyter	18
5.2	<b>Packages and Libraries</b>	<b>18</b>
	5.2.1 NumPy	19
	5.2.2 Pandas	19
	5.2.3 Matplotlib	20
	5.2.4 Scikit-learn	20
	5.2.5 TensorFlow	21
<b>6</b>	<b>SYSTEM DESIGN</b>	<b>22</b>
	6.1 Control Flow Diagram	22
	6.2 Description about the system	23
<b>7</b>	<b>CONCLUSION</b>	<b>26</b>
	7.1 Conclusion	26
	7.2 Future Enhancement	26
<b>8</b>	<b>APPENDIX</b>	<b>27</b>
	8.1 Sample Code	27
<b>9</b>	<b>REFERENCES</b>	<b>48</b>

# LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
2.1	Neural Network Layers	4
2.2	CNN example	4
2.3	Convolutional patch example	5
2.4	Layer transformation	6
2.5	Fully connected layer	7
2.6	Intersection over Union (IoU)	8
2.7	YOLO working principle	9
2.8	YOLO Architecture	10
5.1	Anaconda Logo	18
5.2	Jupyter Logo	18
5.3	Numpy Logo	19
5.4	Pandas Logo	20
5.5	Matplotlib Logo	20
5.6	Scikit Learn Logo	21
5.7	TensorFlow Logo	21
6.1	Control Flow	22
6.2	VGG16 architecture	24

6.3	VGG16 architecture Map	24
A.1	Level model accuracy	37
A.2	Level model loss	38
A.3	Damage model accuracy	45
A.4	Damage model loss	45

## **LIST OF ABBREVIATION**

AI	Artificial Intelligence
CNN	Convolutional Neural Network
YOLO	You Only Look Once
VGG	Visual Geometry Group
RELU	Rectified Linear Unit
IoU	Intersection over Union
AP	Average Precision
mAP	mean Average Precision
RAM	Random Access Memory
SSD	Solid State Drive
GUI	Graphical User Interface
CLI	Command Line Interface
ILSVRC	ImageNet Large Scale Visual Recognition Challenge

# **CHAPTER - 1**

## **INTRODUCTION**

### **1.1 OVERVIEW**

The proliferation of automobile industries is directly related to the number of claims in insurance companies. Those companies are facing many simultaneous claims and solving claims leakage. In Advanced Artificial Intelligence (AI), machine learning and deep learning algorithms can help to solve these kinds of problems for insurance industries. In this paper, we apply deep learning-based algorithms, VGG16 and VGG19, for car damage detection and assessment in real world datasets. The algorithms detect the damaged part of a car, assess its location and severity.

Typically, training a CNN requires an extremely large amount of training data, and can be very time-consuming to perform image classification tasks, taking days or even weeks to complete it. What is more, the purpose of training CNN is to identify the correct weights for the network by multiple forward and backward iterations. Pre-trained CNN models, which have been previously trained on large benchmark datasets like ImageNet dataset, can save some time consuming, directly get their weights and apply their architectures for implementing the learning on the specific tasks via transfer learning.

There's a good performance for car damaged classification using transfer learning with pre-trained CNN model. Moreover, pre-trained CNN models can be used as a feature extract or and a fine-tuned. But, their frameworks are very complicated to understand because the variance is intensity. So there is away how to focus on the impact of certain hyper-parameters and exploring theory to adapt them. In this paper, we choose to use the pre-trained VGG models of VGG16 and VGG19 trained on the ImageNet dataset because of their simple and effective architectures for the capability of object detection and classification in



photographs, their model weights are freely available to load, and use for our specific tasks. In addition to this, we don't need to prepare our new datasets into annotation ones by using them. Then, we utilise our models with transfer learning and L2 regularization to reduce the over-fitting problem. After that, we apply fine-tuning to adjust some hyper-parameters. Following that, we use data augmentation to improve our small dataset creating into a large one. To sum up, we try to improve our system to get our specific tasks.

AI in automotive insurance holds significant potential to quickly estimate vehicle damages. Soon with the advancement in AI algorithms, assessment done manually would be a thing of the past. Traditionally the damage assessment was carried out by multiple parties which were time-consuming, highly prone to human error, leading to inaccurate cost estimations

## CHAPTER - 2

### SYSTEM STUDY

#### 2.1 CONVOLUTIONAL NEURAL NETWORK

It is assumed that the reader knows the concept of Neural networks. When it comes to Machine Learning, Artificial Neural Networks perform really well. Artificial Neural Networks are used in various classification tasks like image, audio, words. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN. Before diving into the Convolution Neural Network, let us first revisit some concepts of Neural Network. In a regular Neural Network there are three types of layers:

- **Input Layers:** It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).
- **Hidden Layer:** The input from the Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.
- **Output Layer:** The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

The data is then fed into the model and output from each layer is obtained this step is called feedforward, we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. After that, we backpropagate into the model by calculating the derivatives. This step is called Backpropagation which basically is used to minimize the loss.

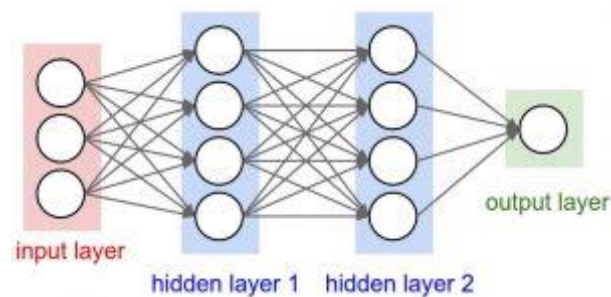


Figure 2.1 –Neural Network Layers

### 2.1.1 Convolution Neural Network

Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (as images generally have red, green, and blue channels).

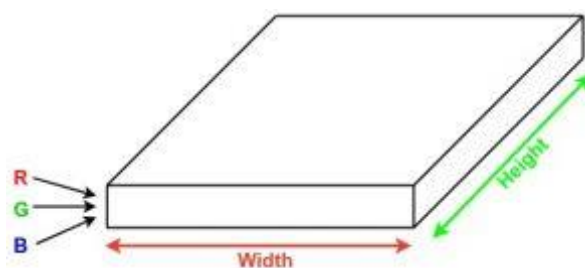


Figure 2.2 – CNN example

Now imagine taking a small patch of this image and running a small neural network on it, with say, k outputs and represent them vertically. Now slide that neural network across the whole image, as a result, we will get another image

with different width, height, and depth. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called Convolution. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

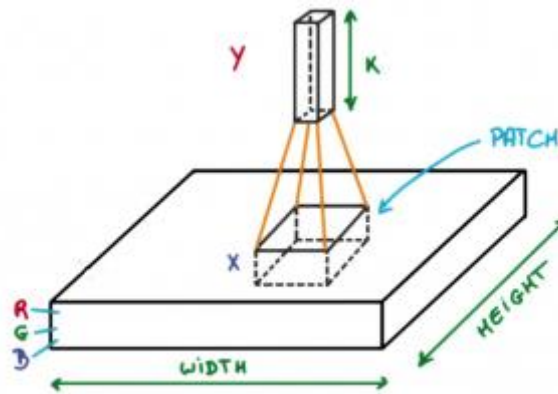


Figure 2.3 – Convolutional patch example

### 2.1.2 Layers used to build ConvNets

A convnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

#### Types of layers:

Let's take an example by running a convnets on of image of dimension 32 x 32 x 3.

1. **Input Layer:** This layer holds the raw input of the image with width 32, height 32, and depth 3.
2. **Convolution Layer:** This layer computes the output volume by computing the dot product between all filters and image patches. Suppose we use a total of 12 filters for this layer we'll get output volume of dimension 32 x 32 x 12.

3. **Activation Function Layer:** This layer will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU:  $\max(0, x)$ , Sigmoid:  $1/(1+e^{-x})$ , Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimension  $32 \times 32 \times 12$ .
4. **Pool Layer:** This layer is periodically inserted in the convnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are max pooling and average pooling. If we use a max pool with  $2 \times 2$  filters and stride 2, the resultant volume will be of dimension  $16 \times 16 \times 12$ .

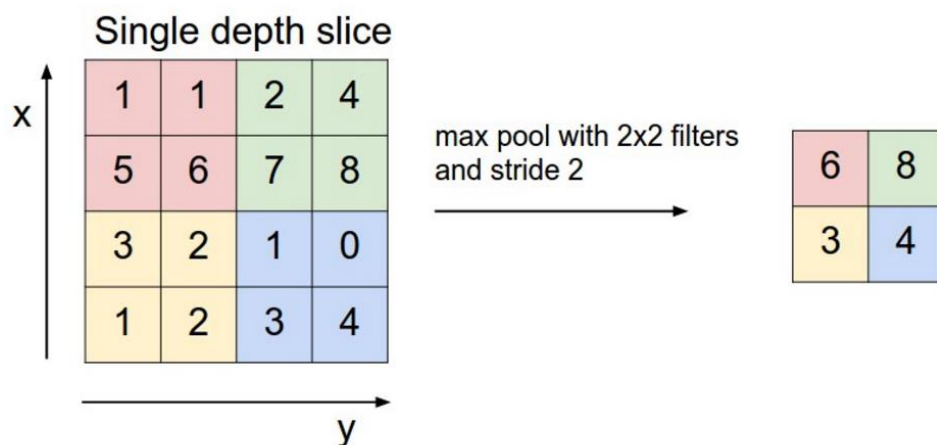


Figure 2.4 – Layer transformation

**Fully-Connected Layer:** This layer is a regular neural network layer that takes input from the previous layer and computes the class scores and outputs the 1-D array of size equal to the number of classes.

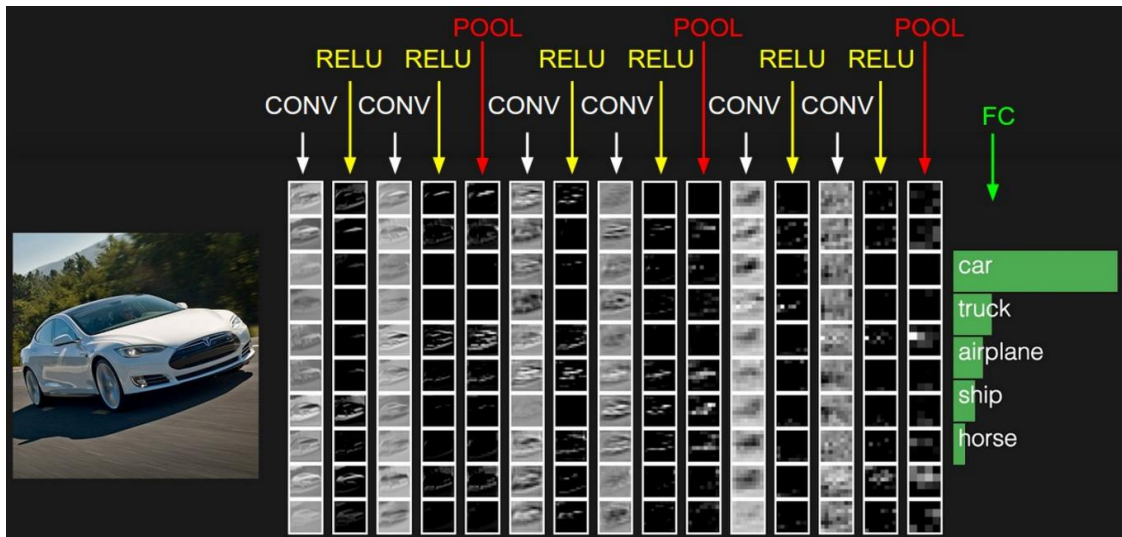


Figure 2.5 – Fully connected layer

## 2.2 YOU ONLY LOOK ONCE

### 2.2.1 What is YOLO?

Compared to the approach taken by object detection algorithms before YOLO, which repurpose classifiers to perform detection, YOLO proposes the use of an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once. Following a fundamentally different approach to object detection, YOLO achieves state-of-the-art results beating other real-time object detection algorithms by a large margin.

### 2.2.2 YOLO vs. other detectors

In addition to increased accuracy in predictions and a better Intersection over Union in bounding boxes (compared to real-time object detectors), YOLO has the inherent advantage of speed. YOLO is a much faster algorithm than its counterparts, running at as high as 45 FPS.

While algorithms like Faster RCNN work by detecting possible regions of interest using the Region Proposal Network and then perform recognition on

those regions separately, YOLO performs all of its predictions with the help of a single fully connected layer. Methods that use Region Proposal Networks thus end up performing multiple iterations for the same image, while YOLO gets away with a single iteration.

### 2.2.3 Intersection over Union (IoU)

Intersection over Union is a popular metric to measure localization accuracy and calculate localization errors in object detection models. To calculate the IoU with the predictions and the ground truth, we first take the intersecting area between the bounding boxes for a particular prediction and the ground truth bounding boxes of the same area. Following this, we calculate the total area covered by the two bounding boxes—also known as the Union. The intersection divided by the Union, gives us the ratio of the overlap to the total area, providing a good estimate of how close the bounding box is to the original prediction.

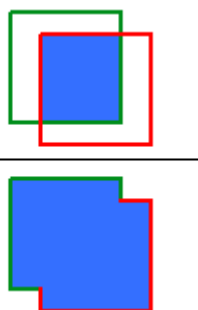
$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{area of overlap}}{\text{area of union}}$$
The diagram shows two overlapping rectangles. The top rectangle has a green border and the bottom rectangle has a red border. The area where they overlap is shaded blue. Below this, the same two rectangles are shown again, but the entire area covered by both rectangles (the union) is shaded blue, while the individual rectangles are outlined in green and red.

Figure 2.6 – Intersection over Union (IoU)

### 2.2.4 Average Precision (AP)

Average Precision is calculated as the area under a precision vs recall curve for a set of predictions. Recall is calculated as the ratio of the total predictions made by the model under a class with a total of existing labels for the class. On the other hand, Precision refers to the ratio of true positives with respect to the total predictions made by the model.

The area under the precision vs recall curve gives us the Average Precision per class for the model. The average of this value, taken over all classes, is termed as mean Average Precision (mAP). In object detection, precision and recall are not for class predictions, but for predictions of boundary boxes for measuring the decision performance. An IoU value  $> 0.5$  is taken as a positive prediction, while an IoU value  $< 0.5$  is a negative prediction.

### 2.2.5 How does YOLO work?

The YOLO algorithm works by dividing the image into  $N$  grids, each having an equal dimensional region of  $S \times S$ . Each of these  $N$  grids is responsible for the detection and localization of the object it contains. Correspondingly, these grids predict  $B$  bounding box coordinates relative to their cell coordinates, along with the object label and probability of the object being present in the cell.

This process greatly lowers the computation as both detection and recognition are handled by cells from the image, but it brings forth a lot of duplicate predictions due to multiple cells predicting the same object with different bounding box predictions. YOLO makes use of Non Maximal Suppression to deal with this issue.

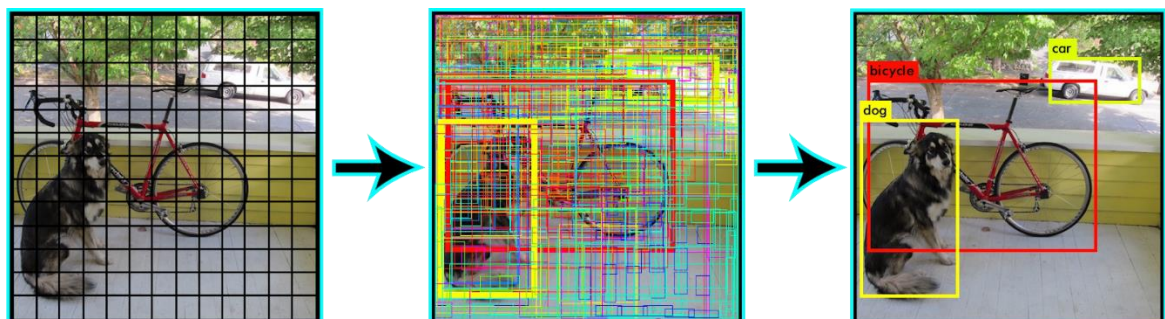


Figure 2.7 – YOLO working principle



In Non Maximal Suppression, YOLO suppresses all bounding boxes that have lower probability scores. YOLO achieves this by first looking at the probability scores associated with each decision and taking the largest one. Following this, it suppresses the bounding boxes having the largest Intersection over Union with the current high probability bounding box. This step is repeated till the final bounding boxes are obtained.

## 2.2.6 YOLO Architecture

Inspired by the GoogleNet architecture, YOLO's architecture has a total of 24 convolutional layers with 2 fully connected layers at the end.

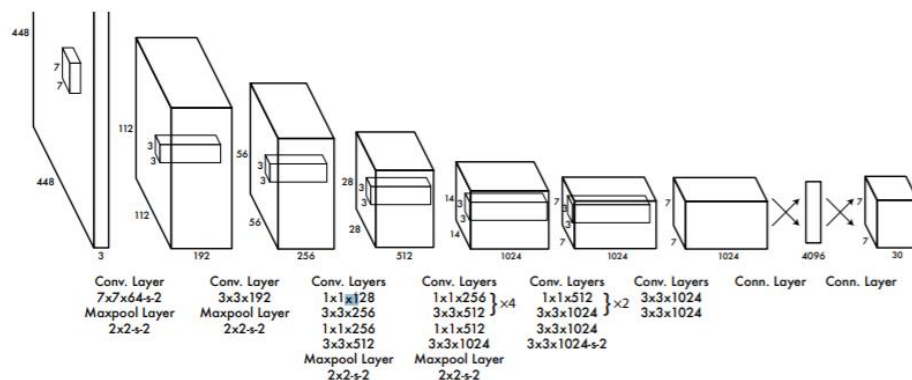


Figure 2.8 – YOLO Architecture

Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 x 1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the layers on the ImageNet classification task at half the resolution (224 x 224 input image) and then double the resolution for detection.

## **CHAPTER - 3**

### **SYSTEM ANALYSIS**

#### **3.1 Existing System**

In this existing study, we present a traditional method that every insurance companies follows for every insurance claims, which is the manual method where the one of the insurance company employee has to verify the damaged car by himself to analyse the cost with him the company's mechanic will also come along to check the damaged parts in the vehicle to make a estimated cost to repair those damaged parts. The details are shared to insurance company to claim the insured amount to repair the vehicle.

#### **3.2 Drawbacks of Existing System**

In this existing system, it takes more than one day to get the analysis of the damage report of the vehicle from the mechanic or the insurance company. There is a possibility of excess amount added by the insurance company to get the amount for themselves from the customers to get benefit. The customer support in this existing method is poor, where the customers will not get the proper feedback, response or anything needed from the insurance company, sometimes they will treat the customers as low life instead of to treat them as equal human beings.

### **3.3 Proposed System**

In this section, we explain our facing problems and how we solve those problem with effective ways in our systems.

#### **3.3.1 Transfer Learning**

Every deep learning model trains and places each task from the ground up, while transfer learning concentrates on feature extraction and appropriate data from source tasks and then applies the required data to a target task. When the source and the target data are similar, transfer learning may improve the performance of the target tasks as a result.

We choose to apply it with the pre-trained VGG models to defeat the over-fitting problems on our small dataset and solve for classification, regression and clustering problems. What is more, it has demonstrated significant progress on how to solve classification problems when the small dataset is not enough to train a CNN model. Also, it can significantly reduce the training time when we use the weights of pre-trained VGG models. The classes of the pre-trained VGG models are the source tasks, and the detected damaged parts of their locations, and their damaged levels are the target tasks in our system.

#### **3.3.2 Influence of Hyper-Parameters**

Adapting the hyper-parameters based on diagnostics in a theoretically way can help to obtain good results with a limited number of tasks. Keeping track of the loss and metric functions during our training datasets can determine good specifications of the learning rate, batch size and the amount of data augmentation in our system. The smaller the learning rate, the larger the batch size and the more stable the learning process, but it is more expensive to use a larger batch size in our system. Fine-tuning transferred parameters can give great the results

according to other related papers. Thus we fine-tune the last layers of pre-trained VGG models to adjust hyper-parameters in our models.

### **3.3.3 Regularization**

Regularization is the controlling of the model complexity. Generally, it is to prevent the over-fitting problem. The most usual way is to add it to the loss function. There are many techniques of regularization such L1 regularization, L2 regularization, dropouts, early stopping, batch normalization, and data augmentation. Among them, L2 regularization was expected to give the best performance, no need to concern with explicit feature selection. Therefore, we use L2 regularization to fit the over-fitting problem in our systems.

### **3.3.4 Dataset Description**

Using cross-validation to estimate our models would need too much computational time because it is very expensive to train CNNs for a long time. Therefore, we decided to split the dataset randomly into separate sets for training (80%) and validation (20%). We randomly put train and validation sets, because creating and training with an ensemble of models would have taken very much time. After training multiple times with different splits, this split proved useful. Finally, the train and test were split such that they have similar images. We create our three datasets based on 1150 car damaged images, which consist of different types of car damages when there is no openly obtainable dataset for car damage classification. To reach our classification procedure, we need to have our three datasets which are manually collected from google images using selenium, which must include respective images without cars, with undamaged cars, and with damaged cars, and ImageNet dataset. While we are dealing with small datasets, we require to use data augmentation to artificially expand and adapt our datasets to improve their performance and decrease their tolerance to the over-fitting issue

during training. Therefore, we utilise randomly rotation, zooming, dimension shift and flipping renovation plans to differ the generated data. We explain more about our three datasets in the dataset subsection.

### **3.4 Advantages of Proposed System**

In the proposed system, we use the machine learning, VGG16 model to get very quick result of damage assessment on the damaged vehicle instead of waiting for a long time. With the help of this model, we are able to easily classify how much the damage is in vehicle. It can be able categorize the proportions of damaged parts and determine whether they need to be replaced or repaired. It aids the user in expediting the process of filling an insurance claim for his/her vehicle. It is process which saves money and time for the customer and the estimation report will be near to 100% accuracy.

## **CHAPTER – 4**

### **SYSTEM SPECIFICATIONS**

#### **4.1 HARDWARE REQUIREMENTS**

Hardware refers to the computer's tangible components or delivery systems that store and run the written instructions provided by the software. The software is the intangible part of the device that lets the user interact with the hardware and command it to perform specific tasks.

- Processor – Intel core i3 10<sup>th</sup> gen
- RAM – 4GB (min)
- Storage – 100GB SSD

#### **4.2 SOFTWARE REQUIREMENTS**

Software Requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed. With the help of this tool are quite user-friendly.

The software's that are required for this project are:

- Spyder
- Anaconda
- Python

## **CHAPTER - 5**

### **SOFTWARE DESCRIPTION**

#### **5.1 PYTHON DISTRIBUTION**

Python is all about Distribution. It can be defined as the collection of files that together allows to build, package and distribute a module. Aside from the official CPython distribution available from python.org, other distributions based on CPython include the following:

- ActivePython
- Anaconda
- ChinesePython Project
- PocketPython
- Portable Python
- PythonwarePython
- StacklessPython

##### **5.1.1 Anaconda Distribution**

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012. As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition, while other products from the company are Anaconda Team Edition and Anaconda Enterprise Edition, both of which are not free.

Package versions in Anaconda are managed by the package management system conda. This package manager was spun out as a separate open-source package as it ended up being useful on its own and for things other than Python. There is also a small, bootstrap version of Anaconda called Miniconda, which includes only conda, Python, the packages they depend on, and a small number of other packages. Anaconda distribution comes with over 250 packages automatically installed, and over 7,500 additional open-source packages can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command-line interface (CLI).

### **5.1.2 Anaconda Navigator**

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- Spyder
- Glue
- Orange
- RStudio
- Visual Studio Code



### 5.1.3 Jupyter

Project Jupyter is a project with goals to develop open-source software, open standards, and services for interactive computing across multiple programming languages. It was spun off from IPython in 2014 by Fernando Pérez and Brian Granger. Project Jupyter's name is a reference to the three core programming languages supported by Jupyter, which are Julia, Python and R. Its name and logo are an homage to Galileo's discovery of the moons of Jupiter, as documented in notebooks attributed to Galileo. Project Jupyter has developed and supported the interactive computing products Jupyter Notebook, JupyterHub, and JupyterLab. Jupyter is financially sponsored by NumFOCUS



Figure 5.1 - PyCharm Logo



Figure 5.2 – Jupyter Logo

## 5.2 Packages and Libraries

A package is a collection of related modules that work together to provide certain functionality. These modules are contained within a folder and can be imported just like any other modules. This folder will often contain a package, potentially containing more modules nested within subfolders.

A library is an umbrella term that loosely means “a bundle of code.” These can have tens or even hundreds of individual modules that can provide a wide range of functionality. Pandas is a file handling library. The Standard Library contains hundreds of modules for performing common tasks, like sending emails or reading JSON data. What’s special about the Standard Library is that it comes

bundled with your installation of programming language, so you can use its modules without having to download them from anywhere.

### 5.2.1 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.



Figure 5.3 - Numpy Logo

### 5.2.2 Pandas

Pandas is a Python library used for working with data sets. It has functions for analysing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008. Pandas allows us to analyse big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.



Figure 5.4 - Pandas Logo

### 5.2.3 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.



Figure 5.5 - Matplotlib Logo

### 5.2.4 Scikit-learn

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.



Figure 5.6 - Scikit Learn Logo

### 5.2.5 TensorFlow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow can be used in a wide variety of programming languages, including Python, JavaScript, C++, and Java. This flexibility lends itself to a range of applications in many different sectors.



Figure 5.7 – TensorFlow Logo

## CHAPTER – 6

### SYSTEM DESIGN

#### 6.1 CONTROL FLOW DIAGRAM

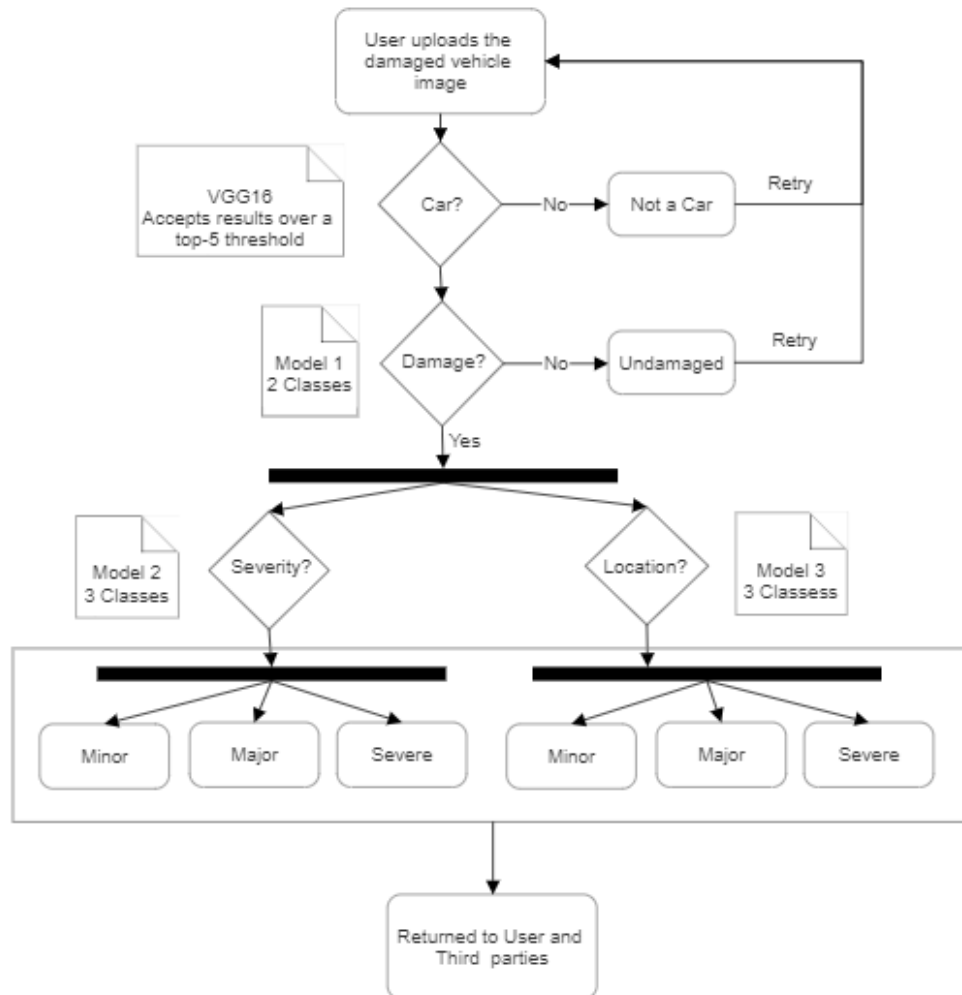


Figure 6.1 - Control Flow

## **6.2 Description about the system**

In the system design, the process have six steps.

### **6.2.1 Input Data**

In our proposed system, the input data is in the form of the image, because we use the 2D images for damage assessment and cost estimation. So, we are using Convolutional Neural Network(CNN) which classifies the image data to get the desired output based on the given input.

The input can be uploaded from the local storages of your mobile phones and Computers, or even can be uploaded directly by using camera to take picture from your mobile phones. The image must be in good quality if it is not the algorithm may the lead the classification to occur an error.

### **6.2.2 VGG16 model**

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual computer vision competition. Each year, teams compete on two tasks. The first is to detect objects within an image coming from 200 classes, which is called object localization. The second is to classify images, each labeled with one of 1000 categories, which is called image classification. VGG 16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University in 2014 in the paper “VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION”. This model won 1st and 2nd place in the above categories in the 2014 ILSVRC challenge.

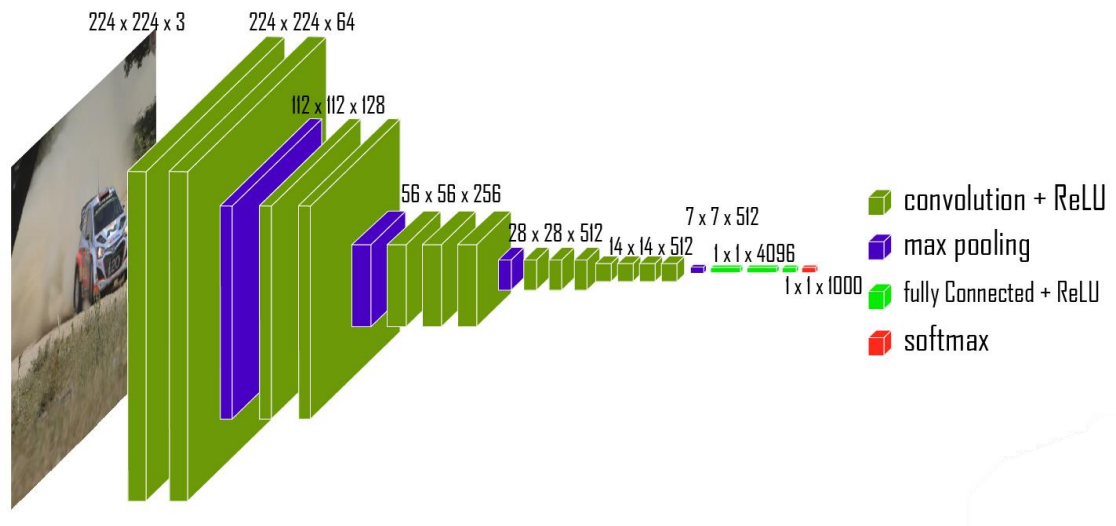


Figure 6.2 – VGG16 architecture

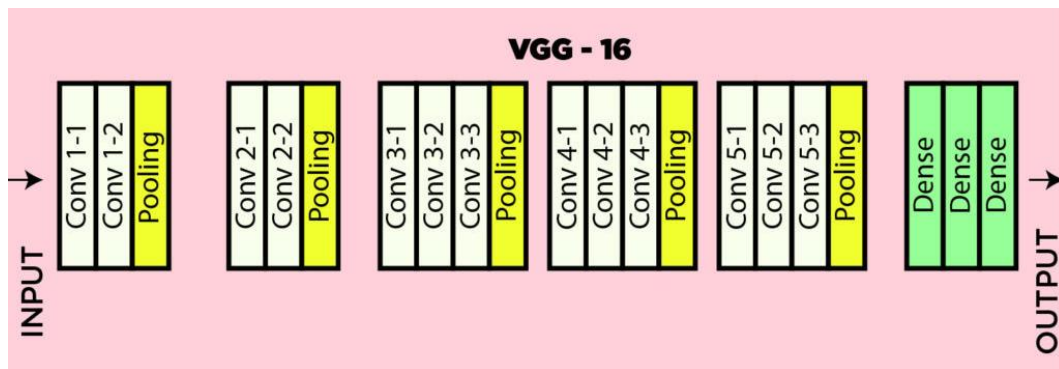


Figure 6.3 – VGG16 architecture Map

The VGG16 model will identify the given input based on the given dataset which we have used to train the model for testing and validation. The VGG16 has different types of activation function which are Threshold function, ReLU (Rectified Linear Unit) function, Sigmoid function, Tanh function, softmax function.

### **6.2.3 Model 1 (Damage analyse)**

The CNN model is used here to identify the input image with the non-damaged vehicle dataset and give the results as No Damage, if there is any damage in the vehicle the model is then separated into two classes namely Severity and Location. Every model in this project uses VGG16 to classify.

### **6.2.4 Model 2 (Severity)**

The first sub model in the Model 1 is the Severity analysis model which classify the damage level into 3 sub classes, those 3 classes are varied by severity level of the damage namely minor, major and severe.

### **6.2.5 Model 3 (Location)**

The second sub model in the Damage classification is Location analysis, it analyse and finds the location of the damage in the vehicle. It further divides into 3 classes namely front, rear and side. This classification is fully based to the view of the vehicle in the input image.

### **6.2.6 Output**

After the classification and analyse the report is returned to user who uploaded the image and can be also send the report to third party companies like insurance company or vehicle service center. The output will show the 2 models result view as Front/Rear/Side and damage as Minor/Major/Severe and also analyse the damage and give the estimated cost to repair the damage.



## **CHAPTER – 7**

### **CONCLUSION AND FUTURE ENHANCEMENT**

#### **7.1 CONCLUSION**

Vehicle damages are often seen in the modern world due to the immense usage of vehicles. The drivers or the vehicle owners should spend a considerable amount of time on finding the severity of the damage and the cost to recover the damage according to the existing approaches. Next the extracted features are taken into consideration and are classified into severity classes. Based on the severities of the damages, they are given an approximate cost utilizing a rule-based engine. After analyzing our models, we find out that the results of using transfer learning and regularization can work better than those of fine-tuning. All of the above, our pre-trained models not only detect damaged part but also assess its location and severity. That why this solution can help the asset for insurance companies to solve claims leakage problems.

#### **7.2 FUTURE ENHANCEMENT**

As the future enhancement of our project, we are decided to add some features like open AI's to get more improved results in the damage assessment. The feature includes that after the enhancement it requires the user to upload 4 or more images of the vehicle from different sides such as front, rear, left side, right side and from top (if possible) to get make a 3D model of the damaged vehicle from the uploaded images, and also be able to find and compare with the original 3D model of the car (undamaged) to get more analysis in the damage assessment. The users can able to move and rotate the rendered 3D model.

## CHAPTER - 8

### APPENDIX

#### 8.1 SAMPLE CODE

##### 8.1.1 Python Code

```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import
Dense,Convolution2D,MaxPooling2D,Flatten

from tensorflow.keras.preprocessing import image

import numpy as np

import keras

from matplotlib import pyplot as plt

import io

import urllib.request
```

##### *#Location Classification*

```
data_level_train = ImageDataGenerator(rescale =
1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=Tr
ue,vertical_flip=True) data_level_test =
ImageDataGenerator(rescale=1./255)
```

### *#Training Data*

```
Location_train =  
data_level_train.flow_from_directory(r"C:\Users\  
ELCOT\Desktop\IBM\Final project\Car  
damage\body\training",  
  
target_size=(76,76),batch_size=30,class_mode="categorical  
")
```

### *#Testing Data*

```
Location_test =  
data_level_test.flow_from_directory(r"C:\Users\ELCOT\  
Desktop\IBM\Final project\Car damage\body\validation",  
target_size=(76,76),batch_size=30,class_mode="categorical  
")
```

Found 979 images belonging to 3 classes.

Found 171 images belonging to 3 classes.

### *#Damage Assessment*

```
data_damage_train = ImageDataGenerator(rescale =  
1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=Tr  
ue,vertical_flip=True)
```

```
data_damage_test =
```

```
ImageDataGenerator(rescale=1./255)
```

### *#Training Data*

```
Damage_train =  
  
data_damage_train.flow_from_directory(r"C:\Users\ELCO  
T\Desktop\IBM\Final project\Car damage\level\training",  
target_size=(76,76),batch_size=30,class_mode="categorical  
")
```

### *#Testing Data*

```
Damage_test =  
  
data_damage_test.flow_from_directory(r"C:\Users\ELCOT\  
Desktop\IBM\  
Final project\Car damage\level\validation",  
target_size=(76,76),batch_size=30,class_mode="categorical  
")
```

Found 979 images belonging to 3 classes.

Found 171 images belonging to 3 classes.

### *#Location CNN*

```
location = Sequential()  
  
location.add(Convolution2D(32,(3,3),activation='relu',input  
_shape=(76,76,3)))  
  
location.add(MaxPooling2D(pool_size=(2, 2)))
```

```
location.add(Flatten())

location.add(Dense(500,activation='relu'))

location.add(Dense(300,activation='relu'))

location.add(Dense(3,activation='softmax'))
```

### *#Damage CNN*

```
damage = Sequential()

damage.add(Convolution2D(32,(3,3),activation='relu',input
_shape=(76,76,3)))

damage.add(MaxPooling2D(pool_size=(2, 2)))

damage.add(Flatten())

damage.add(Dense(500,activation='relu'))

damage.add(Dense(300,activation='relu'))

damage.add(Dense(3,activation='softmax'))
```

### *#Compiling the models*

#### *#location model*

```
location.compile(loss="categorical_crossentropy",optimizer
="adam",metrics=["accuracy"])
```

*#Damage model*

```
damage.compile(loss="categorical_crossentropy",optimizer  
="adam",metrics=["accuracy"])
```

## Training the model

*#Location model training*

*#s\_p\_e => (979/30) == 33*

*#v\_s => (171/30) == 6*

```
Location_Train =  
location.fit(Location_train,steps_per_epoch=33,epochs=30,  
validation_data=Location_test,validation_steps=6)
```

Epoch 1/30

33/33 [=====] - 63s

1s/step - loss: 2.2914 -

accuracy: 0.3626 - val\_loss: 1.1260 - val\_accuracy: 0.4269

Epoch 2/30

33/33 [=====] - 32s

971ms/step - loss: 1.0917

- accuracy: 0.4280 - val\_loss: 1.0687 - val\_accuracy: 0.4211

Epoch 3/30

33/33 [=====] - 39s

1s/step - loss: 1.0791 -

accuracy: 0.4127 - val\_loss: 1.0865 - val\_accuracy: 0.4211

Epoch 4/30

33/33 [=====] - 33s

982ms/step - loss: 1.0585

- accuracy: 0.4423 - val\_loss: 1.0706 - val\_accuracy: 0.4211

Epoch 5/30

33/33 [=====] - 32s

973ms/step - loss: 1.0499

- accuracy: 0.4494 - val\_loss: 1.1198 - val\_accuracy: 0.4386

Epoch 6/30

33/33 [=====] - 34s

1s/step - loss: 1.0111 -

accuracy: 0.4974 - val\_loss: 1.0905 - val\_accuracy: 0.4795

Epoch 7/30

33/33 [=====] - 33s

979ms/step - loss: 1.0145

- accuracy: 0.5158 - val\_loss: 1.0914 - val\_accuracy: 0.4854

Epoch 8/30

33/33 [=====] - 33s

993ms/step - loss: 0.9759

- accuracy: 0.5312 - val\_loss: 1.1071 - val\_accuracy: 0.5439

Epoch 9/30

33/33 [=====] - 33s

987ms/step - loss: 0.9711

- accuracy: 0.5301 - val\_loss: 1.1298 - val\_accuracy: 0.4854

Epoch 10/30

33/33 [=====] - 32s  
979ms/step - loss: 0.9440

- accuracy: 0.5669 - val\_loss: 1.0939 - val\_accuracy: 0.4678  
Epoch 11/30

33/33 [=====] - 33s  
995ms/step - loss: 0.9361

- accuracy: 0.5526 - val\_loss: 1.2094 - val\_accuracy: 0.4854  
Epoch 12/30

33/33 [=====] - 33s  
988ms/step - loss: 0.9123

- accuracy: 0.5761 - val\_loss: 1.1241 - val\_accuracy: 0.4971  
Epoch 13/30

33/33 [=====] - 35s  
1s/step - loss: 0.8780 -

accuracy: 0.5996 - val\_loss: 1.2161 - val\_accuracy: 0.4971  
Epoch 14/30

33/33 [=====] - 33s  
994ms/step - loss: 0.8684

- accuracy: 0.5986 - val\_loss: 1.1626 - val\_accuracy: 0.4503  
Epoch 15/30

33/33 [=====] - 38s  
1s/step - loss: 0.8609 -

accuracy: 0.6210 - val\_loss: 1.3011 - val\_accuracy: 0.4795  
Epoch 16/30



33/33 [=====] - 34s  
 1s/step - loss: 0.8447 -  
 accuracy: 0.6139 - val\_loss: 1.3229 - val\_accuracy: 0.5029  
 Epoch 17/30

33/33 [=====] - 32s  
 972ms/step - loss: 0.8191  
 - accuracy: 0.6170 - val\_loss: 1.2998 - val\_accuracy: 0.4269  
 Epoch 18/30

33/33 [=====] - 32s  
 974ms/step - loss: 0.8148  
 - accuracy: 0.6415 - val\_loss: 1.3024 - val\_accuracy: 0.5088  
 Epoch 19/30

33/33 [=====] - 33s  
 990ms/step - loss: 0.7798  
 - accuracy: 0.6599 - val\_loss: 1.2260 - val\_accuracy: 0.4795  
 Epoch 20/30

33/33 [=====] - 35s  
 1s/step - loss: 0.7823 -  
 accuracy: 0.6415 - val\_loss: 1.2835 - val\_accuracy: 0.5029  
 Epoch 21/30

33/33 [=====] - 33s  
 976ms/step - loss: 0.7301  
 - accuracy: 0.6864 - val\_loss: 1.3868 - val\_accuracy: 0.4795  
 Epoch 22/30

33/33 [=====] - 32s  
961ms/step - loss: 0.7282

- accuracy: 0.6915 - val\_loss: 1.2598 - val\_accuracy: 0.5263  
Epoch 23/30

33/33 [=====] - 33s  
986ms/step - loss: 0.6653

- accuracy: 0.7222 - val\_loss: 1.3745 - val\_accuracy: 0.4971  
Epoch 24/30

33/33 [=====] - 33s  
1s/step - loss: 0.6918 -  
accuracy: 0.7120 - val\_loss: 1.3745 - val\_accuracy: 0.5146  
Epoch 25/30

33/33 [=====] - 33s  
995ms/step - loss: 0.6724

- accuracy: 0.7120 - val\_loss: 1.4253 - val\_accuracy: 0.5088  
Epoch 26/30

33/33 [=====] - 34s  
1s/step - loss: 0.6219 - accuracy: 0.7324 - val\_loss: 1.3387 -  
val\_accuracy: 0.5322  
Epoch 27/30

33/33 [=====] - 32s  
975ms/step - loss: 0.6533

- accuracy: 0.7324 - val\_loss: 1.4048 - val\_accuracy: 0.4561  
Epoch 28/30

```
33/33 [=====] - 38s
1s/step - loss: 0.6526 -
accuracy: 0.7242 - val_loss: 1.3891 - val_accuracy: 0.4854

Epoch 29/30

33/33 [=====] - 39s
1s/step - loss: 0.6031 -
accuracy: 0.7426 - val_loss: 1.4374 - val_accuracy: 0.5380

Epoch 30/30

33/33 [=====] - 43s
1s/step - loss: 0.5996 - accuracy: 0.7497 - val_loss: 1.4803 -
val_accuracy: 0.5146
```

*#plotting the accuracy and loss of Location Model*

*#Accuracy*

```
plt.plot(Location_Train.history['accuracy'])
plt.plot(Location_Train.history['val_accuracy'])
plt.title('Level model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')

plt.show()
```

*#Loss*

```
plt.plot(Location_Train.history['loss'])
```

```
plt.plot(Location_Train.history['val_loss'])
```

```
plt.title('Level model loss')
```

```
plt.ylabel('loss')
```

```
plt.xlabel('epoch')
```

```
plt.legend(['train', 'val'], loc='upper left')
```

```
plt.show()
```

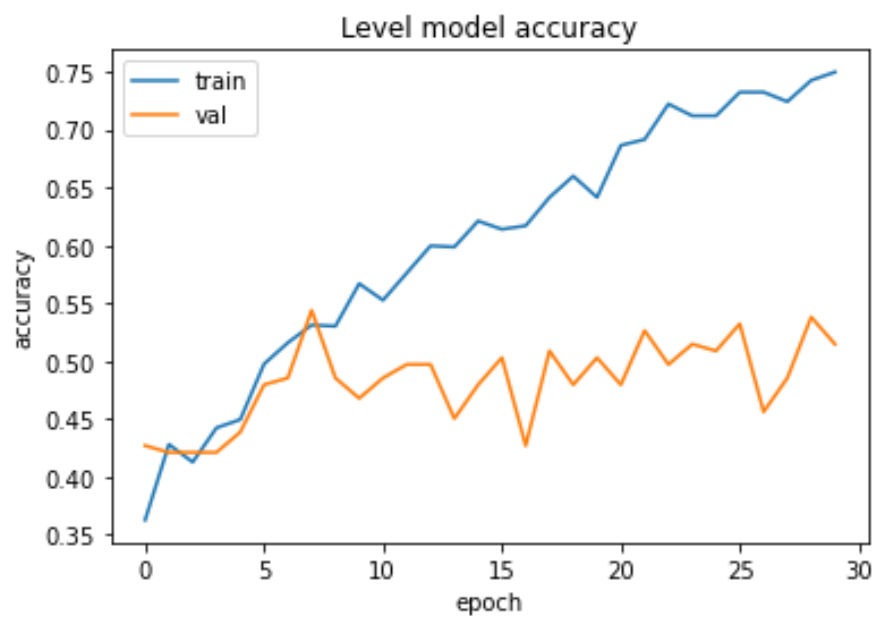


Figure A.1 – Level model accuracy

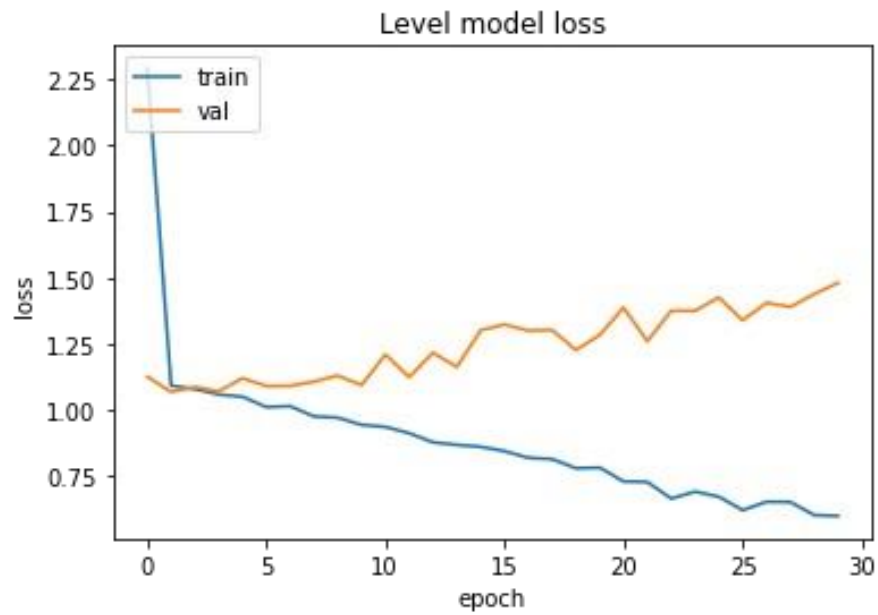


Figure A.2 – Level model Loss

*#Damage model training*

*#s\_p\_e => (979/30) == 33*

*#v\_s => (171/30) == 6*

Damage\_Train =

damage.fit(Damage\_train,steps\_per\_epoch=33,epochs=30,validation\_data=Damage\_test,validation\_steps=6)

Epoch 1/30

33/33 [=====] - 47s

1s/step - loss: 1.5142 -

accuracy: 0.3820 - val\_loss: 1.1120 - val\_accuracy: 0.3977

Epoch 2/30

33/33 [=====] - 48s

1s/step - loss: 1.0369 -

accuracy: 0.4637 - val\_loss: 1.2644 - val\_accuracy: 0.3977

Epoch 3/30

33/33 [=====] - 45s

1s/step - loss: 1.0091 -

accuracy: 0.4883 - val\_loss: 1.2730 - val\_accuracy: 0.4152

Epoch 4/30

33/33 [=====] - 45s

1s/step - loss: 0.9801 -

accuracy: 0.5005 - val\_loss: 1.0369 - val\_accuracy: 0.4971

Epoch 5/30

33/33 [=====] - 36s

1s/step - loss: 0.9522 -

accuracy: 0.5189 - val\_loss: 1.0641 - val\_accuracy: 0.4795

Epoch 6/30

33/33 [=====] - 39s

1s/step - loss: 0.9307 -

accuracy: 0.5495 - val\_loss: 1.0548 - val\_accuracy: 0.4678

Epoch 7/30

33/33 [=====] - 38s

1s/step - loss: 0.9024 - accuracy: 0.5812 - val\_loss: 1.0847 -

val\_accuracy: 0.4620

Epoch 8/30

33/33 [=====] - 64s  
2s/step - loss: 0.9150 -  
accuracy: 0.5434 - val\_loss: 1.1425 - val\_accuracy: 0.4737  
Epoch 9/30  
33/33 [=====] - 69s  
2s/step - loss: 0.8955 -  
accuracy: 0.5853 - val\_loss: 1.1233 - val\_accuracy: 0.4561  
Epoch 10/30  
33/33 [=====] - 67s  
2s/step - loss: 0.8571 -  
accuracy: 0.6180 - val\_loss: 1.1477 - val\_accuracy: 0.4620  
Epoch 11/30  
33/33 [=====] - 70s  
2s/step - loss: 0.8678 -  
accuracy: 0.5914 - val\_loss: 1.1796 - val\_accuracy: 0.4503  
Epoch 12/30  
33/33 [=====] - 65s  
2s/step - loss: 0.8404 -  
accuracy: 0.6016 - val\_loss: 1.1327 - val\_accuracy: 0.4678  
Epoch 13/30  
33/33 [=====] - 69s  
2s/step - loss: 0.8085 -  
accuracy: 0.6415 - val\_loss: 1.1858 - val\_accuracy: 0.4795  
Epoch 14/30

33/33 [=====] - 56s

2s/step - loss: 0.8483 -

accuracy: 0.6078 - val\_loss: 1.1295 - val\_accuracy: 0.4211

Epoch 15/30

33/33 [=====] - 46s

1s/step - loss: 0.8118 -

accuracy: 0.6231 - val\_loss: 1.2709 - val\_accuracy: 0.4561

Epoch 16/30

33/33 [=====] - 45s

1s/step - loss: 0.7777 -

accuracy: 0.6364 - val\_loss: 1.2445 - val\_accuracy: 0.5029

Epoch 17/30

33/33 [=====] - 59s

2s/step - loss: 0.7613 -

accuracy: 0.6558 - val\_loss: 1.3651 - val\_accuracy: 0.4152

Epoch 18/30

33/33 [=====] - 48s

1s/step - loss: 0.7729 -

accuracy: 0.6517 - val\_loss: 1.3303 - val\_accuracy: 0.4854

Epoch 19/30

33/33 [=====] - 39s

1s/step - loss: 0.7196 - accuracy: 0.6803 - val\_loss: 1.3915 -

val\_accuracy: 0.4737

Epoch 20/30



33/33 [=====] - 44s

1s/step - loss: 0.7726 -

accuracy: 0.6721 - val\_loss: 1.3091 - val\_accuracy: 0.4503

Epoch 21/30

33/33 [=====] - 47s

1s/step - loss: 0.7135 -

accuracy: 0.6956 - val\_loss: 1.4631 - val\_accuracy: 0.4737

Epoch 22/30

33/33 [=====] - 42s

1s/step - loss: 0.6661 -

accuracy: 0.7160 - val\_loss: 1.5048 - val\_accuracy: 0.3977

Epoch 23/30

33/33 [=====] - 50s

2s/step - loss: 0.6671 -

accuracy: 0.7232 - val\_loss: 1.5726 - val\_accuracy: 0.4327

Epoch 24/30

33/33 [=====] - 47s

1s/step - loss: 0.6569 - accuracy: 0.7191 - val\_loss: 1.4662 -  
val\_accuracy: 0.4503

Epoch 25/30

33/33 [=====] - 47s

1s/step - loss: 0.6579 -

accuracy: 0.6987 - val\_loss: 1.2992 - val\_accuracy: 0.3743

Epoch 26/30

33/33 [=====] - 47s  
1s/step - loss: 0.6463 -  
accuracy: 0.7232 - val\_loss: 1.4134 - val\_accuracy: 0.4971

Epoch 27/30

33/33 [=====] - 46s  
1s/step - loss: 0.6001 -  
accuracy: 0.7692 - val\_loss: 1.6205 - val\_accuracy: 0.4444

Epoch 28/30

33/33 [=====] - 53s  
2s/step - loss: 0.6009 -  
accuracy: 0.7508 - val\_loss: 1.3528 - val\_accuracy: 0.4094

Epoch 29/30

33/33 [=====] - 50s  
1s/step - loss: 0.5835 -  
accuracy: 0.7549 - val\_loss: 1.4194 - val\_accuracy: 0.4561

Epoch 30/30

33/33 [=====] - 47s  
1s/step - loss: 0.4999 - accuracy: 0.8069 - val\_loss: 1.7530 -  
val\_accuracy: 0.3743

*#plotting the accuracy and loss of Damage model*

*#Accuracy*

```
plt.plot(Damage_Train.history['accuracy'])
```

```
plt.plot(Damage_Train.history['val_accuracy'])
```

```
plt.title('Damage model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'val'], loc='upper left')

plt.show()
```

### *#Loss*

```
plt.plot(Damage_Train.history['loss'])

plt.plot(Damage_Train.history['val_loss'])

plt.title('Damage model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'val'], loc='upper left')

plt.show()
```

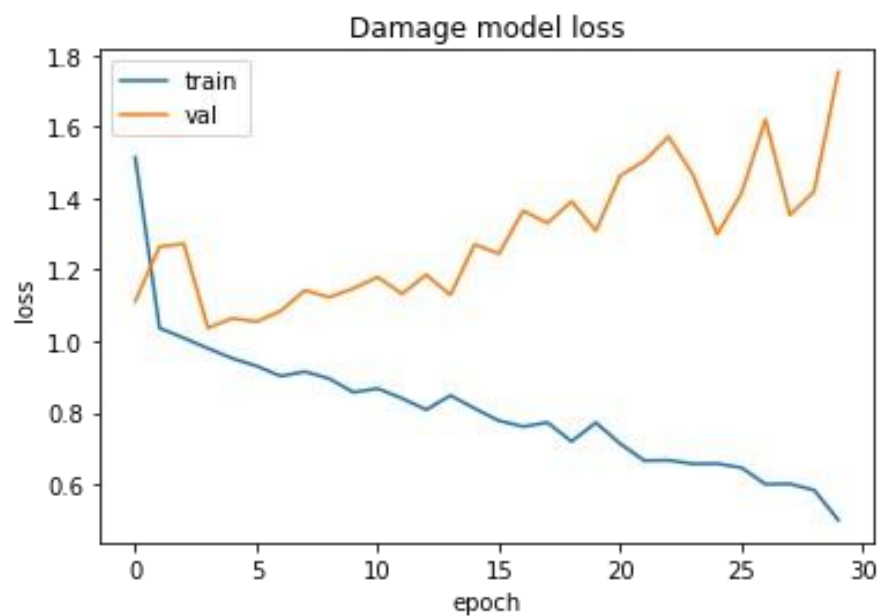


Figure A.3 – Damage model accuracy

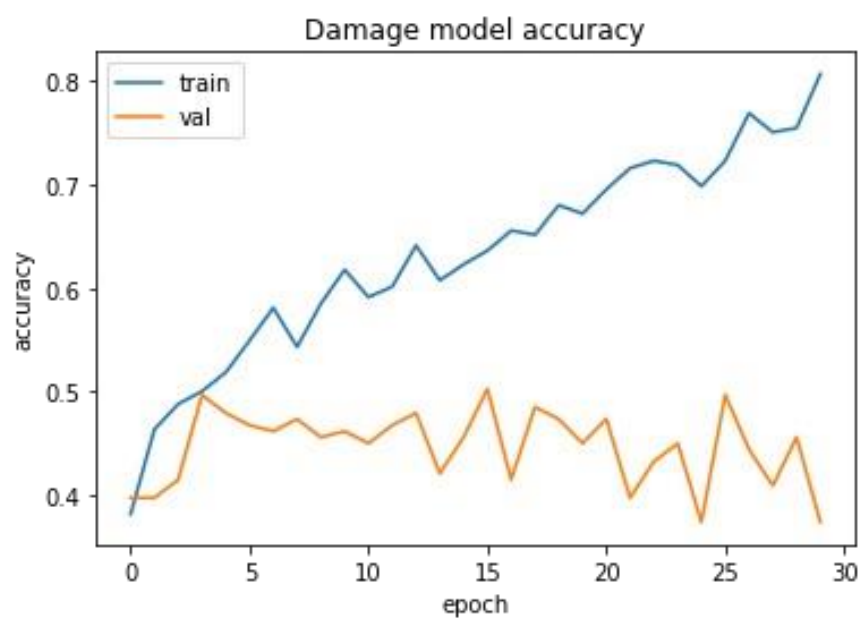


Figure A.4 – Damage model loss

*#Saving the Location Model*

```
location.save("Location.h5")
```

*#Saving the Damage Model*

```
damage.save("Damage.h5")
```

## Testing the Model

```
Location_train.class_indices
```

```
{'00-front': 0, '01-rear': 1, '02-side': 2}
```

```
Damage_train.class_indices
```

```
{'01-minor': 0, '02-moderate': 1, '03-severe': 2}
```

*#Location Model Testing*

```
img =
```

```
image.load_img(r"C:\Users\ELCOT\Desktop\IBM\Final  
project\
```

```
Front.jpg",target_size=(76,76))
```

```
x = image.img_to_array(img)
```

```
x = np.expand_dims(x,axis=0)
```

```
pred_prob = location.predict(x)
```

```
class_name = ['00-front', '01-rear', '03-side']
```

```
pred_id = pred_prob.argmax(axis=1)[0]
```

```
class_name[pred_id]
```

```
1/1 [=====] - 5s 5s/step
```

```
'00-front'
```

### *#Damage Model Testing*

```
img =
```

```
image.load_img(r"C:\Users\ELCOT\Desktop\IBM\Final  
project\
```

```
Front.jpg",target_size=(76,76))
```

```
x = image.img_to_array(img)
```

```
x = np.expand_dims(x,axis=0)
```

```
pred_prob = damage.predict(x)
```

```
class_name = ['01-minor', '02-moderate', '03-severe']
```

```
pred_id = pred_prob.argmax(axis=1)[0]
```

```
class_name[pred_id]
```

```
1/1 [=====] - 1s 1s/step
```

```
'02-moderate'
```

## REFERENCES

- [1] J. D. Deijin, “Automatic Car Damage Recognition using Convolutional Neural Networks,” March 2018.
- [2] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li. and L. F. Fei, “Image Net: A Large-Scale Hierarchical Image Database,” IEEE, 2019.
- [3] N. Dhieb, H. Ghazzai, H. Besbes, and Y. Massoud, ”A Very Deep Transfer Learning Model for Vehicle Damage Detection and Localization,” in IEEE International Conference on Vehicular Electronics and Safety (ICVES’19), Cairo, Egypt, Sept. 2019
- [4] M. Dhieb, H. Ghazzai, H. Besbes, and Y. Massoud, ”Extreme gradient boosting machine learning algorithm for safe auto insurance operations,” in IEEE International Conference on Vehicular Electronics and Safety (ICVES’19), Cairo, Egypt, Sept. 2019.
- [5] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio, “Why does unsupervised pre-training help deep learning?” Journal of Machine Learning Research, vol. 11, no. Feb, pp. 625 – 660, 2010.
- [6] M. Dwivedi, M. H. Shadab, S. Omkar, E. B. Monis, B. Khanna, S.R. S. A. Tiwari, and A. Rathi, “Deep Learning Based Car Damage Classification and Detection,” DOI:10.13140/ RG.2.2.18702.51525, Sep.2019.
- [7] Ellen Rathje and Melba Crawford, “Using high resolution satellite imagery to detect damage from the2003 northern Algeria earthquake,” in 13th World Conference on Earthquake Engineering, August, 2004, pp. 1 – 6.

- [8] W. A. R. Harshani, and K. Vidanage, “Image Processing based Severity and Cost Prediction of Damages in the Vehicle Body: A Computational Intelligence Approach,” National Information Technology Conference(NITC), 13-15 September, 2017.
- [9] A. Karpathy, “Course Notes of Convolutional Neural Networks for Visual Networks for Visual Recognition,” Standard University classCS231n. Available: [w.w.w.cs231n.github.io](http://w.w.w.cs231n.github.io), December, 2017.
- [10] K Kouchi and F Yamazaki, “Damage detection based on object-based segmentation and classification from high resolution satellite images for the 2003 boumerdes, Algeria earthquake,” in Proceedings of the 26th Asian conference on Remote Sensing, Hanoi, Vietnam, 2005.
- [11] Michael Giering Mark R. Gurvich Soumalya Sarkar, Kishore K. Reddy, “Deep learning for structural health monitoring: A damage characterization application,” in Annual Conference of the Prognostics and Health Management Society, 2016.
- [12] K. Patil, M. Kulkarni, A. Sriraman and S. Kerande, “Deep Learning Based Car Damage Classification,” in IEEE Int. Conf. Machine Learning App. (ICMLA’17), Cancun, Mexico, Dec. 2017.
- [13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 779-788, 2016.
- [14] F Samadzadegan and H Rastiveisi, “Automatic detection and classification of damaged buildings, using high resolution satellite imagery and vector data,” The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. 37, pp. 415 – 420,2008.
- [15] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv preprint arXiv: 1409.1556,2014.



- [16] R. Singh, M. P. Ayyar, and T. V. S. Pavan, S. Gosain, and R. R. Shah, "Automating Car Insurance Claims Using Deep Learning Techniques," in IEEE fifth International Conference on Multimedia Big Data(BigMM), 2019.
- [17] P. Li, B. Y. Shen, and W. Dong, "An Anti-Fraud System for Car Insurance claim Based on Visual Evidence," arXiv: 1804.11207v1, April,2018.
- [18] Srimal Jayawardena et al., "Image based automatic vehicle damage detection", Ph.D. thesis, AustralianNational University, 2013.
- [19] M. Wassel, "Property Casualty: Deterring Claims Leakage in the Digital Age," Cognizant Insurance Practice, Tech. Rep., 2018.
- [20] M. Wassel, "Property Casualty: Deterring Claims Leakage in the Digital Age," Cognizant Insurance Practice, Tech. Rep., 2018.
- [21] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How Transferable are Features in Deep Neural Networks?," in Advances in Neural Information Processing Systems 27 (Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K. Q. Weinberger, eds.), pp. 3320/3328, Curran Associates, Inc.,2014.