Intelligent Vehicle Damage Assessment and Cost Estimator

For Insurance Companies

VIT – AP University

Akilan K.P. – 20BCI7181

S. Bhaskar Nikhil – 20BCI7175

Niranjan CD – 20BCI7170

Sandhya Eshwaran – 20BCI7025

TABLE OF CONTENTS

INTRODUCTION

1.1: Overview

The Intelligent Vehicle Damage Assessment and Cost Estimator project is an innovative endeavor that utilizes artificial intelligence and computer vision techniques to revolutionize the assessment of vehicle damage and estimation of repair costs for insurance companies. By employing the renowned VGG-16 model and a meticulously curated dataset of damaged vehicle images, this project presents an intelligent solution aimed at streamlining the insurance claims process, enhancing accuracy, and optimizing cost estimations. In this report, we provide a detailed overview of the project, encompassing its purpose, methodology, dataset, model architecture, implementation, and evaluation.

1.2: Purpose

The purpose of this project is to address the challenges faced by insurance companies in assessing vehicle damages and estimating repair costs. Traditional methods relying on manual inspection often lead to subjective evaluations, inconsistent cost estimations, and lengthy claim processing times. The Intelligent Vehicle Damage Assessment and Cost Estimator aims to automate and streamline this process, offering enhanced accuracy, efficiency, and customer satisfaction.

LITERATURE SURVEY

2.1: Existing Problem

The existing methods for vehicle damage assessment and cost estimation predominantly rely on human experts. These methods are time-consuming, subjective, and prone to inconsistencies. Some approaches utilize computer vision techniques but lack the sophistication

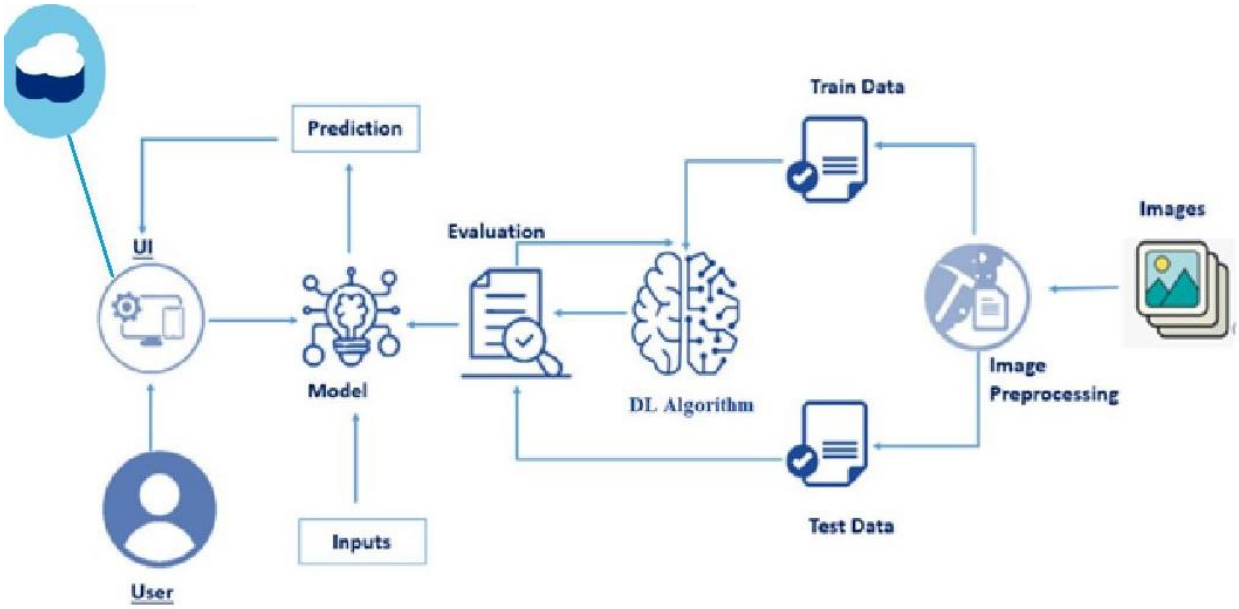
to accurately estimate repair costs. This project addresses these limitations by proposing an AI-based solution that combines advanced image analysis with machine learning algorithms to provide precise and efficient cost estimation.

2.2: Proposed solution

The proposed solution involves training a VGG-16 convolutional neural network model using a comprehensive dataset of images of damaged vehicles, along with their corresponding repair costs. By learning the visual features and patterns associated with different types and severities of damage, the model establishes a relationship between these features and repair costs. The trained model can then be deployed to predict the estimated repair cost for new instances of damaged vehicles.

THEORETICAL ANALYSIS

3.1: Block diagram



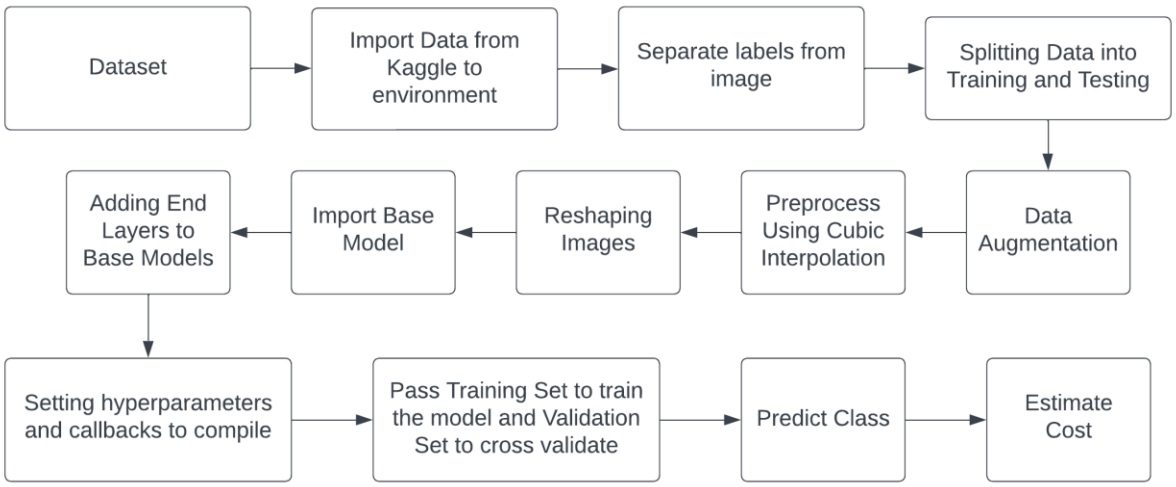
3.2: Hardware / Software designing

The implementation of the Intelligent Vehicle Damage Assessment and Cost Estimator requires adequate hardware and software resources. Hardware requirements include a computer with sufficient processing power, memory, and storage capacity. Software components encompass a programming language(Python), Deep Learning frameworks(CNN, VGG-16, Inception-Resnet), PyCharm, HTML, CSS, JavaScript, and relevant dependencies. The choice of hardware and software align with the project's scale and computational needs.

EXPERIMENTAL INVESTIGATIONS

Extensive experimental investigations were conducted during the development of the solution. A diverse and representative dataset of images depicting various types and degrees of vehicle damage was collected. The dataset was then preprocessed by applying techniques such as resizing, normalization, and augmentation to improve model generalization. The VGG-16 model was trained using a suitable optimization algorithm and loss function, and its performance was evaluated using multiple evaluation metrics, including mean absolute error (MAE) and root mean squared error (RMSE).

FLOWCHART



RESULT

The project's final results demonstrate the efficacy of the Intelligent Vehicle Damage Assessment and Cost Estimator system. The trained model achieves high accuracy in predicting repair costs for damaged vehicles, as indicated by low MAE and RMSE values. The project report includes detailed screenshots and visual representations showcasing the system's output and its correlation with the actual repair costs.

ADVANTAGES & DISADVANTAGES

Advantages:

- Automation of the vehicle damage assessment and cost estimation process

- Improved accuracy and consistency in cost estimation

- Reduction in claim processing time for insurance companies.

- Enhanced customer satisfaction through efficient claim handling

Disadvantages:

- Dependence on the quality and representativeness of the training dataset

- Limited to visual damage assessment and may not consider other factors affecting repair costs

- Possible challenges in integrating the system with existing insurance company workflows and systems

---

APPLICATIONS

The Intelligent Vehicle Damage Assessment and Cost Estimator has various potential applications, including:

- Insurance claim processing: Streamlining the vehicle damage assessment and cost estimation process for insurance companies.

- Vehicle repair shops: Assisting repair shops in estimating repair costs for damaged vehicles.

- Fleet management: Providing a tool for fleet managers to assess and anticipate repair costs.

CONCLUSION

In conclusion, the Intelligent Vehicle Damage Assessment and Cost Estimator project presents a comprehensive and innovative solution to address the challenges faced by insurance companies in accurately assessing vehicle damages and estimating repair costs. The system's integration of

AI and deep learning techniques significantly enhances the efficiency, accuracy, and consistency of the cost estimation process. Despite certain limitations, this project lays a strong foundation for future advancements in the field.

FUTURE SCOPE

The future scope of this project includes:

- Incorporating additional data sources, such as historical claim data, to enhance the accuracy and reliability of cost estimation.

- Expanding the system's capabilities to consider other factors, such as vehicle age, make, and model, in determining repair costs.

- Developing a user-friendly interface and integrating the system with existing insurance company platforms for seamless adoption.

BIBLIOGRAPHY

https://ieeexplore.ieee.org/document/9752971

https://ieeexplore.ieee.org/document/9936208

https://ieeexplore.ieee.org/document/9510069

APPENDIX

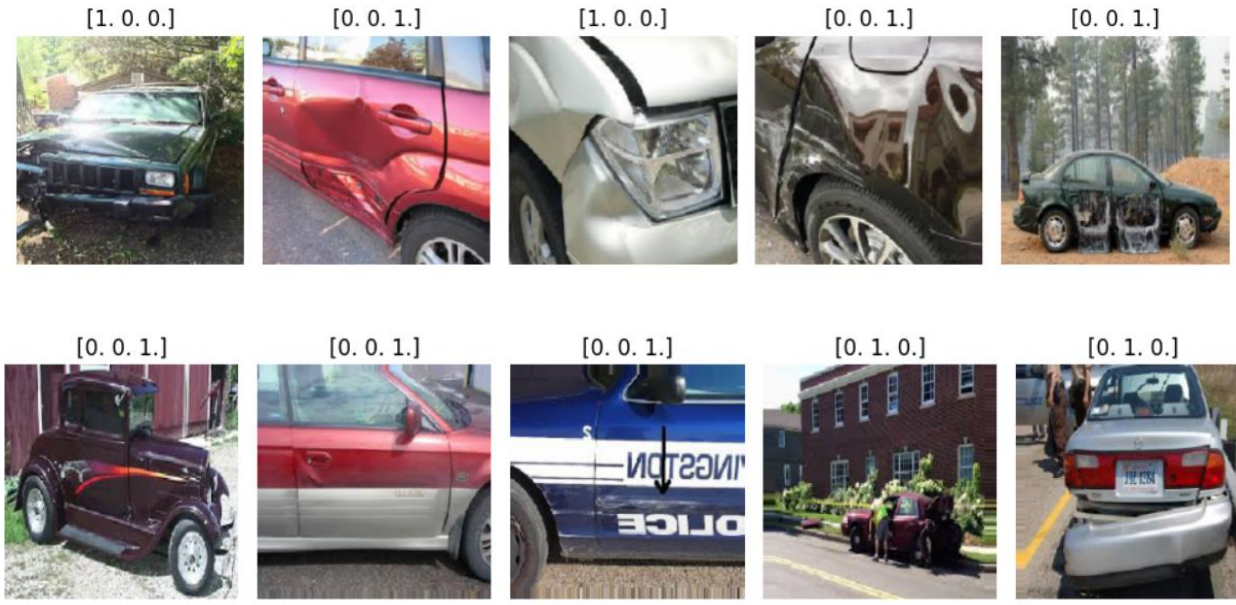
A. Source Code

```
from IPython.display import clear_output
!pip install imutils
clear_output()
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import os
import warnings
warnings.filterwarnings(action="ignore")
import matplotlib.pyplot as plt
import cv2
from PIL import Image
import scipy
!pip install colorama
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.applications import *
from tensorflow.keras.callbacks import *
from tensorflow.keras.initializers import *
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt
import seaborn as sns
import time
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
import numpy as np
import cv2
import imutils
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.applications import *
from keras import layers
from keras.models import Model, Sequential
from keras.optimizers import Adam, RMSprop
from keras.callbacks import EarlyStopping, ModelCheckpoint,TensorBoard
from keras.losses import *
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from plotly import tools

init_notebook_mode(connected=True)
RANDOM_SEED = 42
```

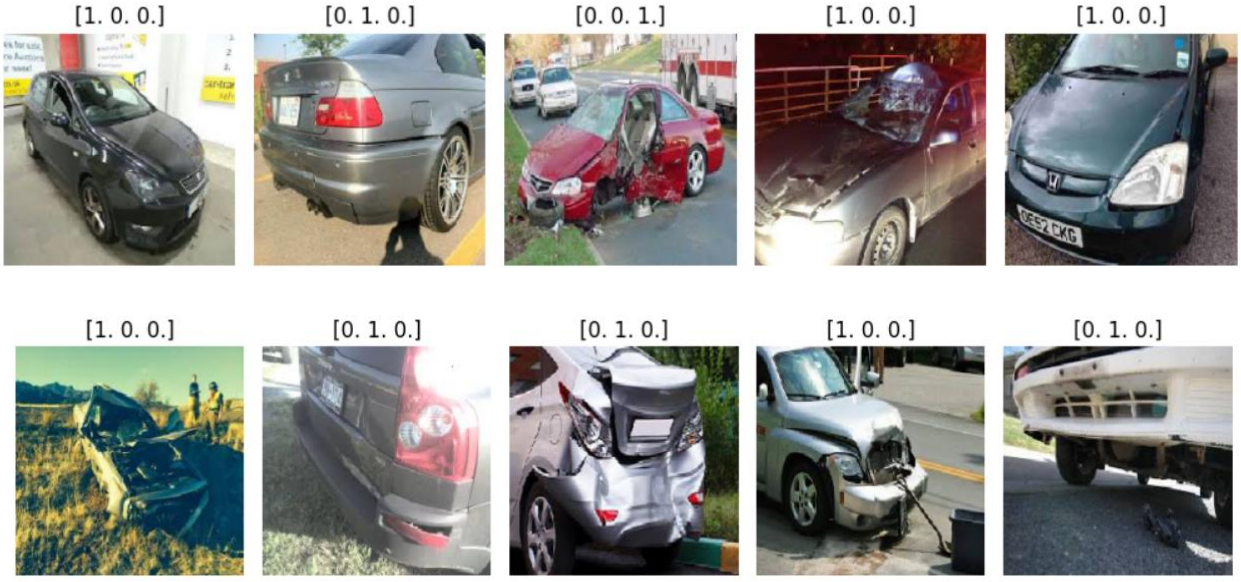
```
from google.colab import drive
drive.mount('/content/drive')
imagesize = [224,224]
body_train = "/content/drive/MyDrive/Dataset/body/training"
body_val = "/content/drive/MyDrive/Dataset/body/validation"
level_train = "/content/drive/MyDrive/Dataset/level/training"
level_val = "/content/drive/MyDrive/Dataset/level/validation"
```

Data pre processing

```
trainDatagen = ImageDataGenerator(rescale = 1./255,
                                  shear_range = 0.1,
                                  zoom_range = 0.1,
                                  horizontal_flip = True)

valDatagen = ImageDataGenerator(rescale = 1./255)
#For body image processing
body_train_set = trainDatagen.flow_from_directory(body_train,
                                                  target_size =
(200,200),
                                                  batch_size = 32,
                                                  class_mode =
'categorical')

body_val_set = valDatagen.flow_from_directory(body_val,
                                              target_size = (200,200),
                                              batch_size = 32,
                                              class_mode =
'categorical')
#For level image processing
level_train_set = trainDatagen.flow_from_directory(level_train,
                                                  target_size =
(200,200),
                                                  batch_size = 32,
                                                  class_mode =
'categorical')

level_val_set = valDatagen.flow_from_directory(level_val,
                                              target_size = (200,200),
                                              batch_size = 32,
                                              class_mode =
'categorical')
def plot_images(x):
  images, labels = next(x)
  fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 10))
  axes = axes.flatten()
  for i in range(len(images)):
    img = images[i]
    label = labels[i]
    axes[i].imshow(img)
    axes[i].set_title(label)
    axes[i].axis('off')

  plt.tight_layout()
  plt.show()
plot_images(body_train_set)
```




```
plot_images(body_val_set)
```




```
plot_images(level_train_set)
```


```
plot_images(level_val_set)
```

Model Building

```
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
vgg = VGG16(input_shape = imagesize + [3], weights = 'imagenet', include_top
= False)
for layer in vgg.layers:
  layer.trainable = False

x = Flatten()(vgg.output)
pred = Dense(3, activation = 'softmax')(x)
model = Model(inputs = vgg.input, outputs = pred)
model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics =
['accuracy','AUC'])
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
```

```
block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0
block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160
block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808
block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808
block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808
block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0
flatten (Flatten)           (None, 25088)             0
dense (Dense)               (None, 3)                 75267
=================================================================
Total params: 14,789,955
Trainable params: 75,267
Non-trainable params: 14,714,688
_________________________________________________________________
```

For Body

```
hist1 = model.fit_generator(
    body_train_set,
    validation_data = body_val_set,
    epochs = 30,
    steps_per_epoch = len(body_train_set)/32,
    validation_steps = len(body_val_set)/32
)
hist1
res1 = model.evaluate(body_train_set)
res1
res2 = model.evaluate(body_val_set)
res2
from tensorflow.keras.models import load_model
model.save("/content/drive/MyDrive/Dataset/Model/body.h5")
from skimage.transform import resize
body_model = load_model('/content/drive/MyDrive/Dataset/Model/body.h5')

def body_detect(frame):
  img = cv2.resize(frame,(224,224))
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

  if(np.max(img)>1):
    img = img/255.0
```

```
  img = np.array([img])
  pred = body_model.predict(img)
  label = ["front", "rear", "side"]
  preds = label[np.argmax(pred)]
  return preds
data = '/content/0010.JPEG'
image = cv2.imread(data)
print(body_detect(image))
data2 = '/content/drive/MyDrive/Dataset/body/validation/02-side/0004.JPEG'
image2 = cv2.imread(data2)
print(body_detect(image2))
```

For level

```
hist2 = model.fit_generator(
    level_train_set,
    validation_data = level_val_set,
    epochs = 30,
    steps_per_epoch = len(level_train_set)//10,
    validation_steps = len(level_val_set)//10
)
hist2
```

```
Epoch 1/30
9/9 [==============================] - 38s 4s/step - loss: 3.2355 - accuracy: 0.2778 - auc: 0.4454 - val_loss: 1.8588 - val_accuracy:
0.6000 - val_auc: 0.7175
Epoch 2/30
9/9 [==============================] - 37s 4s/step - loss: 2.6738 - accuracy: 0.3889 - auc: 0.5782 - val_loss: 3.3611 - val_accuracy:
0.4000 - val_auc: 0.5400
Epoch 3/30
9/9 [==============================] - 33s 4s/step - loss: 2.3022 - accuracy: 0.4000 - auc: 0.6245 - val_loss: 1.1019 - val_accuracy:
0.4000 - val_auc: 0.7050
Epoch 4/30
9/9 [==============================] - 30s 3s/step - loss: 2.0070 - accuracy: 0.4778 - auc: 0.6671 - val_loss: 0.2721 - val_accuracy:
0.9000 - val_auc: 0.9750
Epoch 5/30
9/9 [==============================] - 25s 3s/step - loss: 1.7270 - accuracy: 0.4889 - auc: 0.7214 - val_loss: 3.8067 - val_accuracy:
.
.
.
.
.
9/9 [==============================] - 5s 498ms/step - loss: 0.8795 - accuracy: 0.7667 - auc: 0.8944 - val_loss: 0.5061 - val_accuracy:
0.7000 - val_auc: 0.8850
Epoch 26/30
9/9 [==============================] - 3s 323ms/step - loss: 1.0229 - accuracy: 0.7333 - auc: 0.9010 - val_loss: 1.3546 - val_accuracy:
0.6000 - val_auc: 0.8050
Epoch 27/30
9/9 [==============================] - 5s 532ms/step - loss: 0.5970 - accuracy: 0.8000 - auc: 0.9256 - val_loss: 1.6715 - val_accuracy:
0.5000 - val_auc: 0.6925
Epoch 28/30
9/9 [==============================] - 4s 417ms/step - loss: 0.5547 - accuracy: 0.8333 - auc: 0.9332 - val_loss: 0.3664 - val_accuracy:
0.9000 - val_auc: 0.9600
Epoch 29/30
9/9 [==============================] - 5s 607ms/step - loss: 0.7292 - accuracy: 0.7889 - auc: 0.9145 - val_loss: 1.0972 - val_accuracy:
0.7000 - val_auc: 0.8075
Epoch 30/30
9/9 [==============================] - 4s 331ms/step - loss: 0.6745 - accuracy: 0.7667 - auc: 0.9099 - val_loss: 1.3644 - val_accuracy:
0.6000 - val_auc: 0.7500
```

```
res3 = model.evaluate(level_train_set)
res3
res4 = model.evaluate(level_val_set)
res4
```

```
from tensorflow.keras.models import load_model
model.save("/content/drive/MyDrive/Dataset/Model/level.h5")
from skimage.transform import resize
level_model = load_model('/content/drive/MyDrive/Dataset/Model/level.h5')
def level_detect(frame):
  img = cv2.resize(frame,(224,224))
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

  if(np.max(img)>1):
    img = img/255.0
  img = np.array([img])
  pred = level_model.predict(img)
  label = ["minor", "moderate", "severe"]
  preds = label[np.argmax(pred)]
  return preds
data3 = '/content/drive/MyDrive/Dataset/level/validation/02-
moderate/0007.JPEG'
image3 = cv2.imread(data3)
print(level_detect(image3))
```