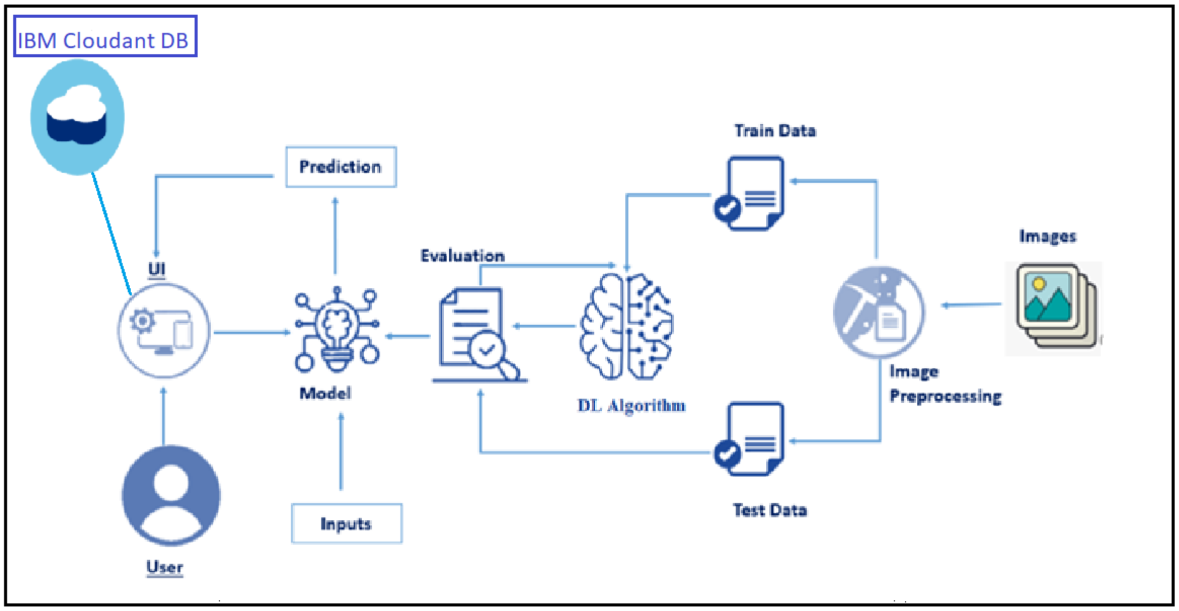
Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies Using IBM Cloud

Project Description:

Nowadays, a lot of money is being wasted in the car insurance business due to leakage claims. Claims leakage /Underwriting leakage is characterized as the discrepancy between the actual payment of claims made and the sum that should have been paid if all of the industry's leading practices were applied. Visual examination and testing have been used to may these results. However, they impose delays in the processing of claims.

The aim of this project is to build a VGG16 model that can detect the area of damage on a car. The rationale for such a model is that it can be used by insurance companies for faster processing of claims if users can upload pics and the model can assess damage(be it dent scratch from and estimates the cost of damage. This model can also be used by lenders if they are underwriting a car loan especially for a used car.

Technical Architecture:



Prerequisites:

To complete this project, you must require the following software's, concepts and packages

- **Anaconda navigator and PyCharm / Spyder:**
  - Refer the link below to download anaconda navigator
  - Link (PyCharm) : https://youtu.be/1ra4zH2G4o0
  - Link (Spyder) : https://youtu.be/5mDYijMfSzs
- **Python packages:**
  - Open anaconda prompt as administrator
  - Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter...
  - Type "pip install tensorflow==2.3.2" and click enter.
  - Type "pip install keras==2.3.1" and click enter.
  - Type "pip install Flask" and click enter.

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- **Deep Learning Concepts**
  - **CNN:** https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
  - **VGG16:** https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615
  - **ResNet-50:** https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
  - **Inception-V3:** https://iq.opengenus.org/inception-v3-model-architecture/
  - **Xception:** https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
- **Computer Vision**
  - https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html
- **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

  Link: **https://www.youtube.com/watch?v=lj4I_CvBnt0**

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of VGG16.
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- Know how to build a web application using the Flask framework.

Project Flow:

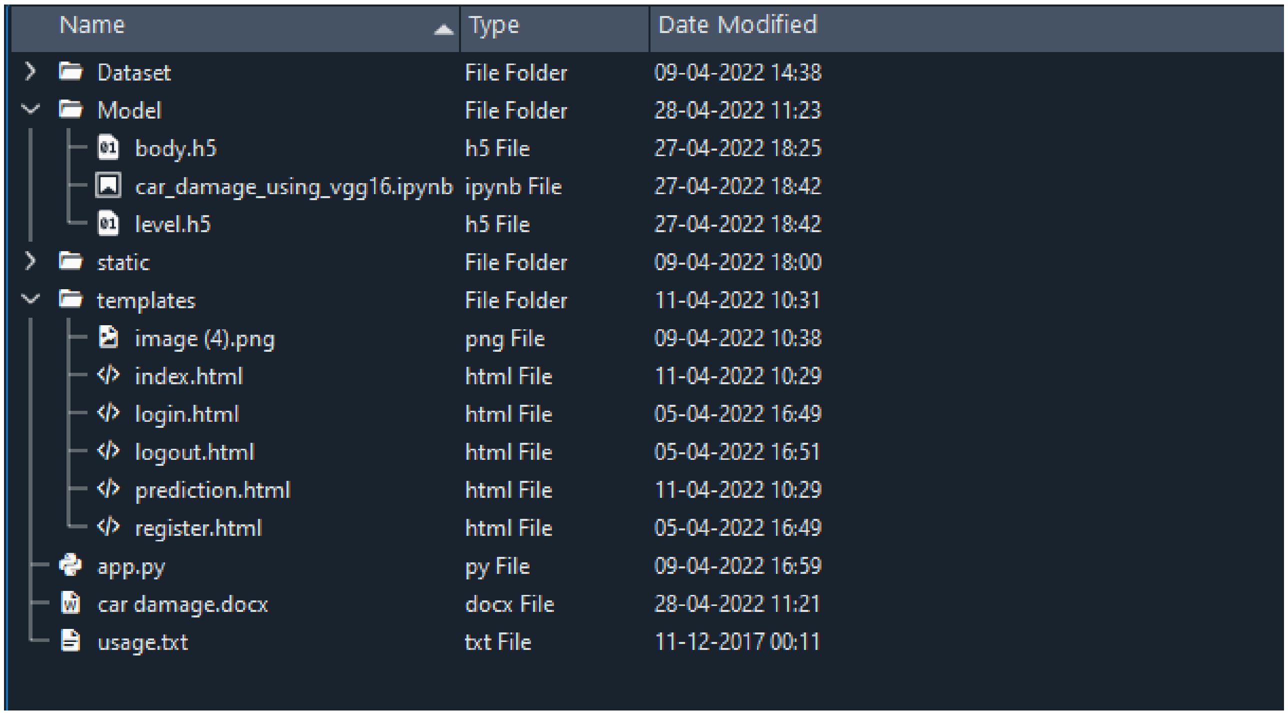
- The user interacts with the UI (User Interface) to choose the image.
- The chosen image is analyzed by the model which is integrated with the flask application.
- VGG16 Model analyzes the image, then the prediction is showcased on the Flask UI.

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
  - Create Train and Test Folders.
- Image Preprocessing.
  - Import the ImageDataGenerator library
  - Configure ImageDataGenerator class
  - ApplyImageDataGenerator functionality to Trainset and Testset
- Model Building
  - Import the model building Libraries
  - Loading the model
  - Adding Flatten layers
  - Adding Output Layer
  - Creating Model Object
  - Configure the Learning Process
  - Train the Model
  - Save the Model
  - Test The Model
- Cloudant DB
  - Register & Login to IBM Cloud
  - Create Service Instance
  - Creating Service Credentials
  - Launch Cloudant DB
  - Create Database
- Application Building
  - Building HTML Pages
  - Build Python Code
  - Run The Application

Project Structure:

Create a Project folder which contains files as shown below



Name	Type	Date Modified
Dataset	File Folder	09-04-2022 14:38
Model	File Folder	28-04-2022 11:23
body.h5	h5 File	27-04-2022 18:25
car_damage_using_vgg16.ipynb	ipynb File	27-04-2022 18:42
level.h5	h5 File	27-04-2022 18:42
static	File Folder	09-04-2022 18:00
templates	File Folder	11-04-2022 10:31
image (4).png	png File	09-04-2022 10:38
index.html	html File	11-04-2022 10:29
login.html	html File	05-04-2022 16:49
logout.html	html File	05-04-2022 16:51
prediction.html	html File	11-04-2022 10:29
register.html	html File	05-04-2022 16:49
app.py	py File	09-04-2022 16:59
car damage.docx	docx File	28-04-2022 11:21
usage.txt	txt File	11-12-2017 00:11

- The Dataset folder contains the training and testing images for training our model.
- We are building a Flask Application that needs HTML pages stored in the **templates** folder and a python script **app.py** for server-side scripting
- we need the model which is saved and the saved model in this content is a **body.h5 and level.h5**
- templates folder contains base.html,index.html pages.

Milestone 1: Data Collection

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

The dataset is divided 2 sub folders such as Body and Level. You can download the dataset used in this project using the below link.

Dataset: https://drive.google.com/drive/folders/1lpOrcULlx5mHuSbbLRCrvmCwXKkbDVKE?usp=sharing

Note: For better accuracy train on more image

Milestone 2: Image Pre Processing

In this milestone, we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, etc.

Reference: Link

Activity 1: Import The ImageDataGenerator Library

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

Let us import the ImageDataGenerator class from TensorFlow Keras

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

Activity 2: Configure ImageDataGenerator Class

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation

There are five main types of data augmentation techniques for image data; specifically:

Image shifts via the width_shift_range and height_shift_range arguments.
The image flips via the horizontal_flip and vertical_flip arguments.
Image rotations via the rotation_range argument
Image brightness via the brightness_range argument.
Image zoom via the zoom_range argument.

An instance of the ImageDataGenerator class can be constructed for train and test.

```
#setting parameter for image data augmentation to the training data
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

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

Activity 3: Apply ImageDataGenerator Functionality To Trainset And Testset

Let us apply ImageDataGenerator functionality to Trainset and Testset by using the following code.For Training set using flow_from_directory function.

This function will return batches of images from the subdirectories

Arguments:

directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.

batch_size: Size of the batches of data which is 64.

target_size: Size to resize images after they are read from disk.

class_mode:

- ‘int': means that the labels are encoded as integers (e.g. for sparse_categorical_crossentropy loss).

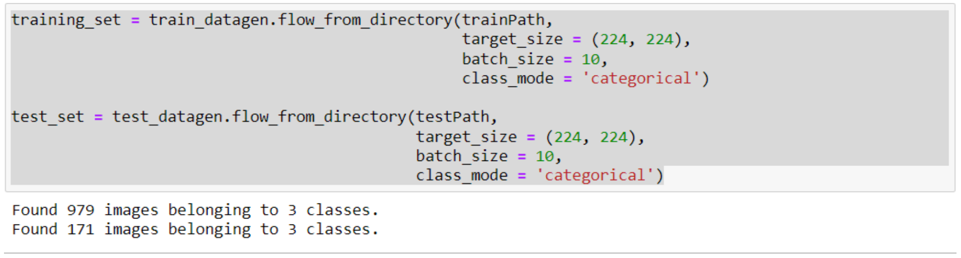
- 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).

- 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).

- None (no labels).

Loading our data and performing Data Augmentation

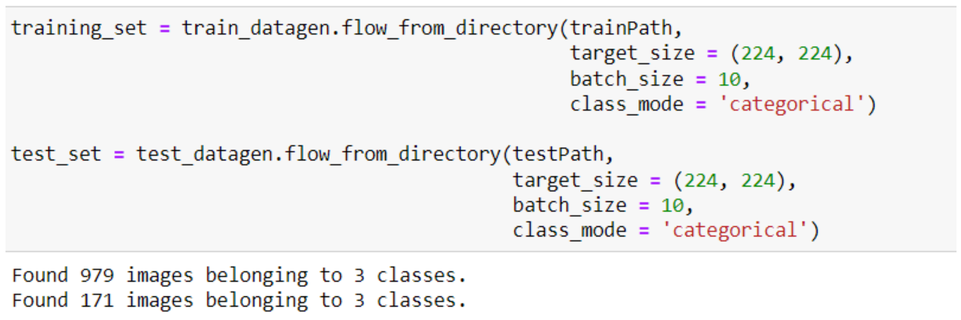
For Body Damage:



```
training_set = train_datagen.flow_from_directory(trainPath,
                                                 target_size = (224, 224),
                                                 batch_size = 10,
                                                 class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(testPath,
                                            target_size = (224, 224),
                                            batch_size = 10,
                                            class_mode = 'categorical')
```

```
Found 979 images belonging to 3 classes.
Found 171 images belonging to 3 classes.
```

For Level of Damage:



```
training_set = train_datagen.flow_from_directory(trainPath,
                                                 target_size = (224, 224),
                                                 batch_size = 10,
                                                 class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(testPath,
                                            target_size = (224, 224),
                                            batch_size = 10,
                                            class_mode = 'categorical')
```

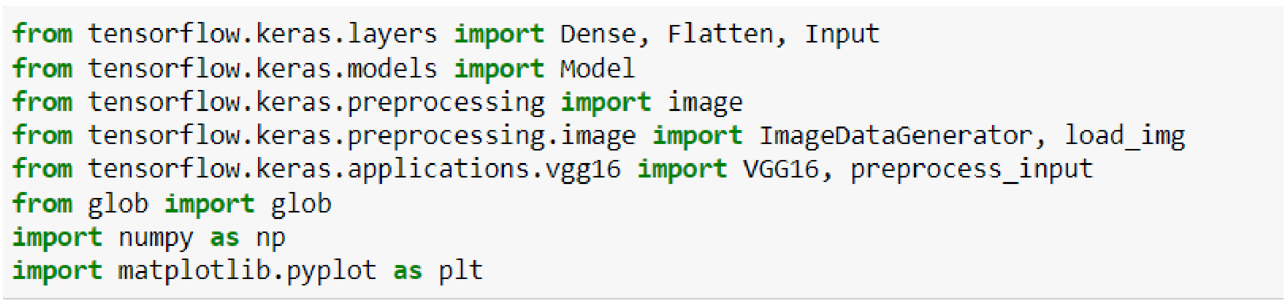
```
Found 979 images belonging to 3 classes.
Found 171 images belonging to 3 classes.
```

Milestone 3: Model Building

Now it's time to Build input and output layers for the VGG16 model. Hidden layers are frozen because they have trained sequences, so changing the input and output layers.

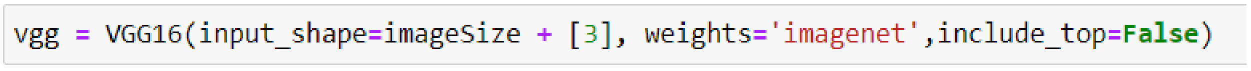
Activity 1: Importing The Model Building Libraries

Import the necessary libraries as shown in the image
.



```
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from glob import glob
import numpy as np
import matplotlib.pyplot as plt
```

Activity 2: Initializing The Model

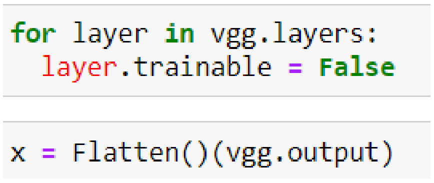


```
vgg = VGG16(input_shape=imageSize + [3], weights='imagenet',include_top=False)
```

The vgg16 model needs to be loaded and we are storing that into a variable called vgg.

Activity 3: Adding Flatten Layer

For the VGG16 model, we need to keep the Hidden layer training as false, because it has trained weights.

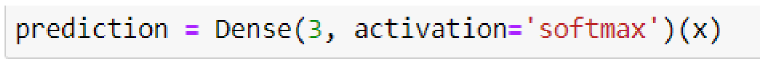


```
for layer in vgg.layers:
  layer.trainable = False
```

```
x = Flatten()(vgg.output)
```

Activity 4: Adding Output Layer

Our dataset has 3 classes, so the output layer needs to be changed as per the dataset.

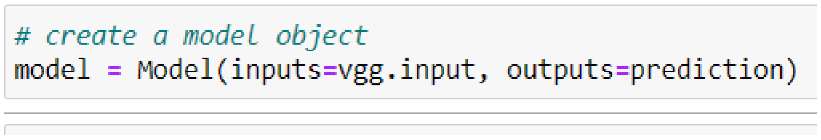


```
prediction = Dense(3, activation='softmax')(x)
```

3 indicates no of classes, softmax is the activation function we use for categorical output

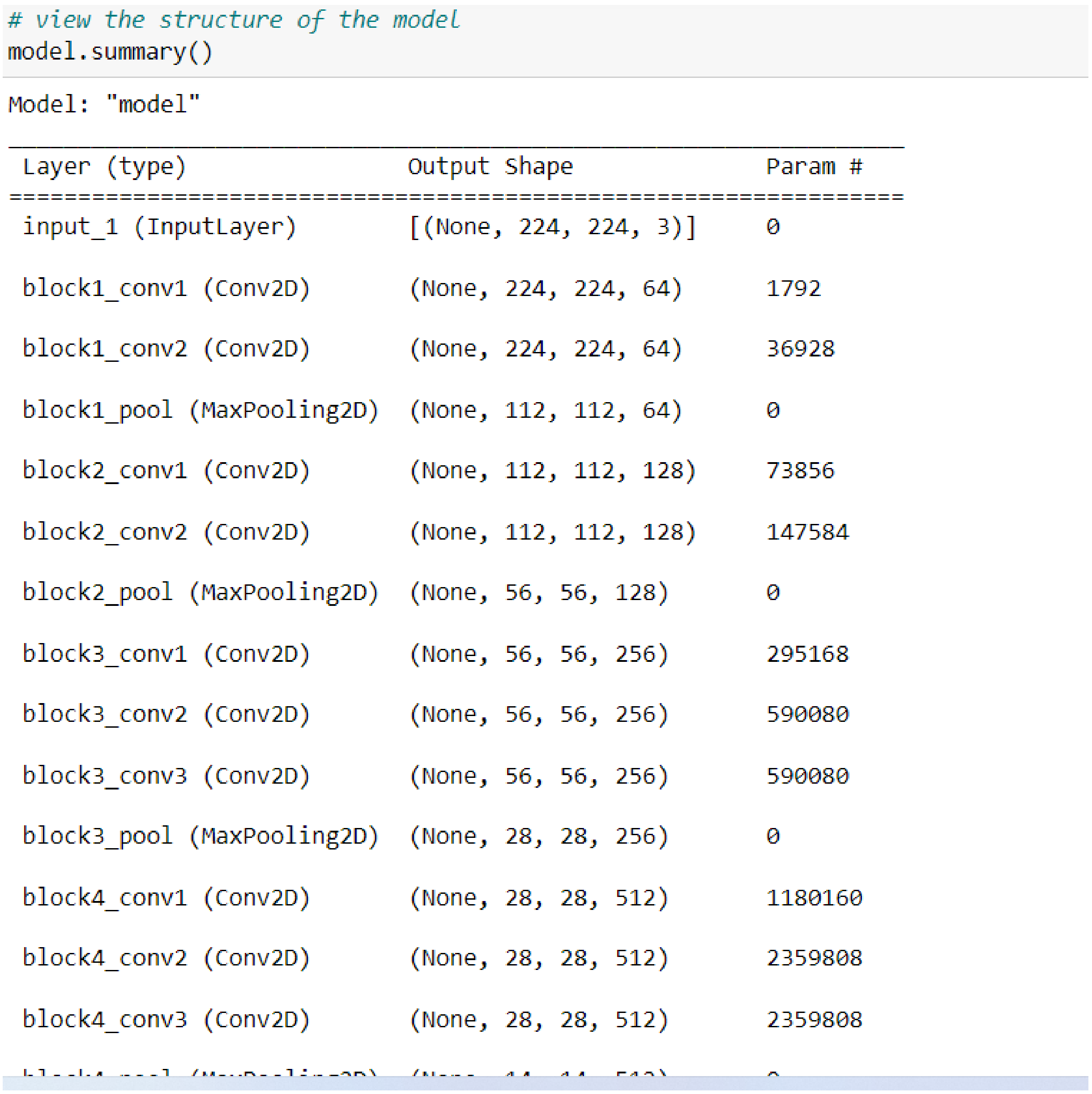
Adding a fully connected layer.

Activity 5: Creating A Model Object



```
# create a model object
model = Model(inputs=vgg.input, outputs=prediction)
```

We have created inputs and outputs in the previous steps and we are creating a model and fitting it to the vgg16 model so that it will take inputs as per the given and displays the given no of classes



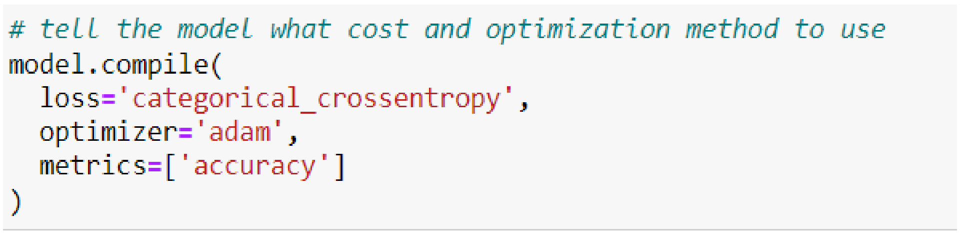
```python
# view the structure of the model
model.summary()
```

```
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
```

Activity 6: Configure The Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

Compiling The Model



```
# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

Activity 7: Train The Model

Now, let us train our model with our image dataset. The model is trained for 25 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 25 epochs and probably there is further scope to improve the model.

fit_generator functions used to train a deep learning neural network

Arguments:

steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

Epochs: an integer and number of epochs we want to train our model for.

validation_data can be either:

- an inputs and targets list

- a generator

- inputs, targets, and sample_weights list which can be used to evaluate

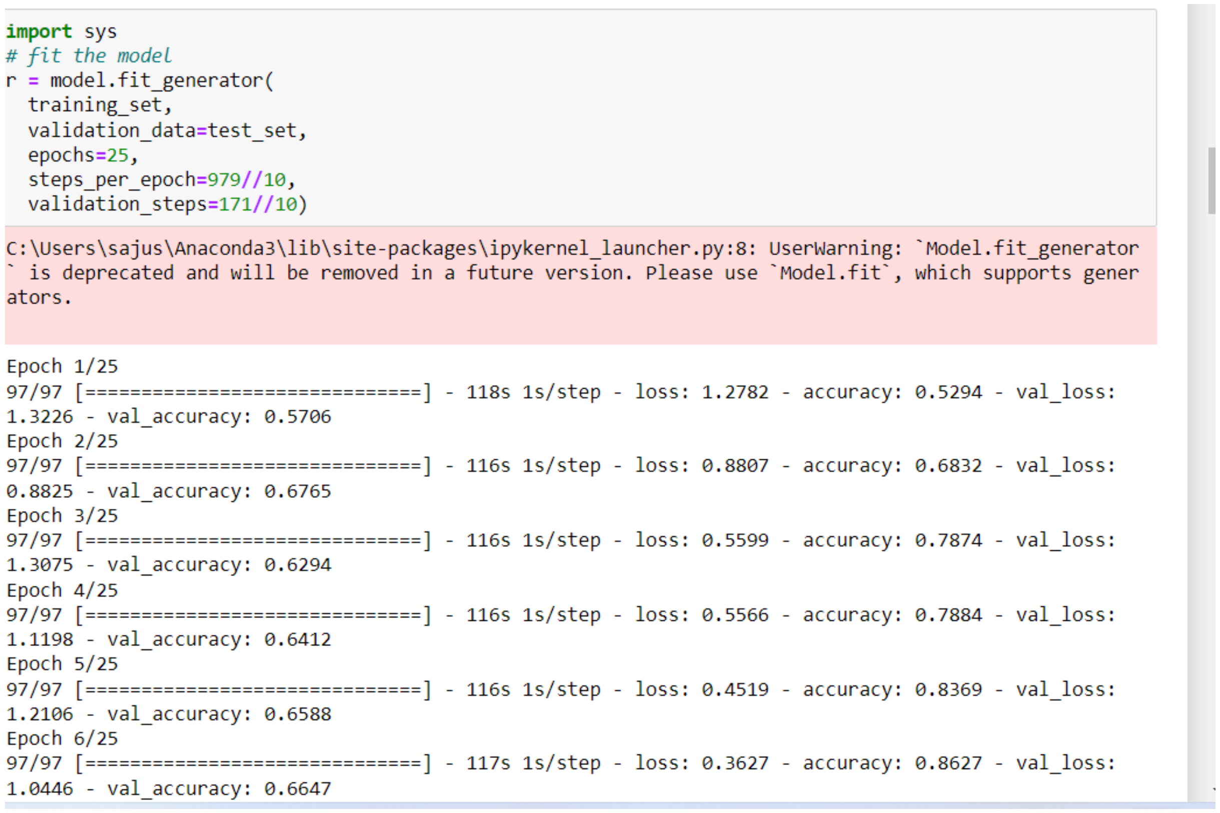
  the loss and metrics for any model after any epoch has ended.

validation_steps: only if the validation_data is a generator then only this argument

can be used. It specifies the total number of steps taken from the generator before it is

stopped at every epoch and its value is calculated as the total number of validation data points

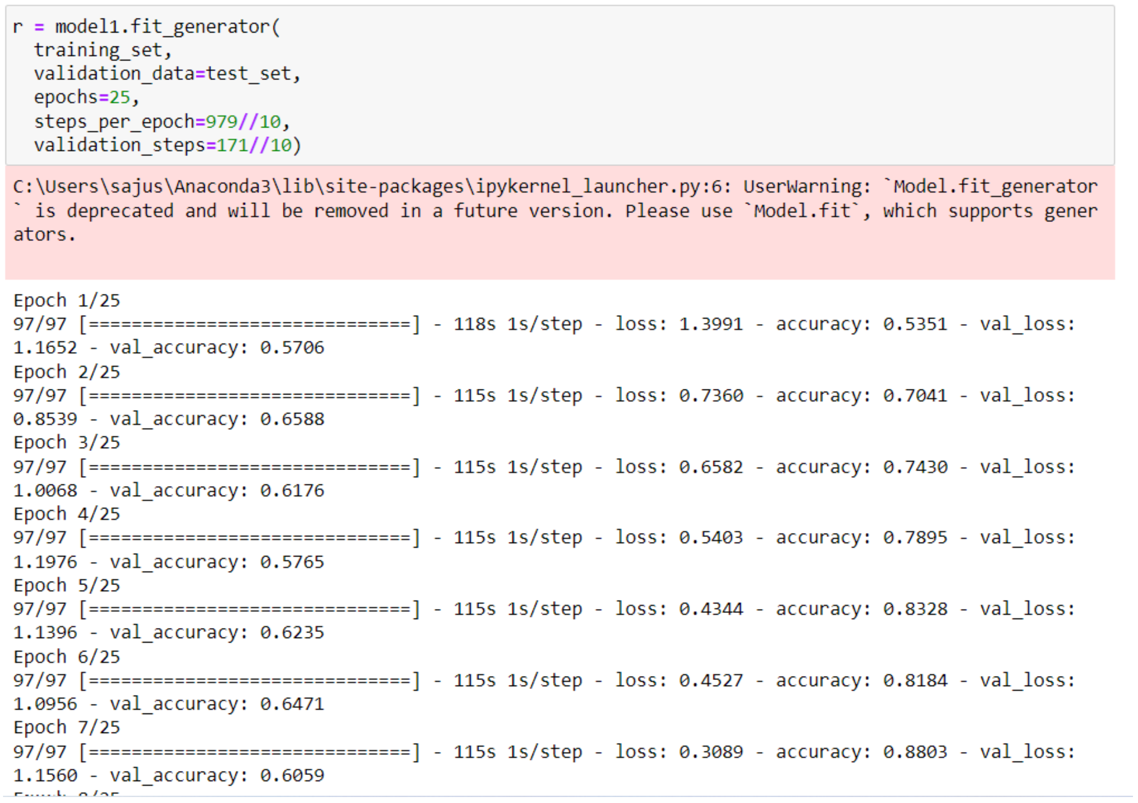
in your dataset divided by the validation batch size.



```
import sys
# fit the model
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=25,
  steps_per_epoch=979//10,
  validation_steps=171//10)
```

```
C:\Users\sajus\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: `Model.fit_generator
` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports gener
ators.
```

```
Epoch 1/25
97/97 [==============================] - 118s 1s/step - loss: 1.2782 - accuracy: 0.5294 - val_loss:
1.3226 - val_accuracy: 0.5706
Epoch 2/25
97/97 [==============================] - 116s 1s/step - loss: 0.8807 - accuracy: 0.6832 - val_loss:
0.8825 - val_accuracy: 0.6765
Epoch 3/25
97/97 [==============================] - 116s 1s/step - loss: 0.5599 - accuracy: 0.7874 - val_loss:
1.3075 - val_accuracy: 0.6294
Epoch 4/25
97/97 [==============================] - 116s 1s/step - loss: 0.5566 - accuracy: 0.7884 - val_loss:
1.1198 - val_accuracy: 0.6412
Epoch 5/25
97/97 [==============================] - 116s 1s/step - loss: 0.4519 - accuracy: 0.8369 - val_loss:
1.2106 - val_accuracy: 0.6588
Epoch 6/25
97/97 [==============================] - 117s 1s/step - loss: 0.3627 - accuracy: 0.8627 - val_loss:
1.0446 - val_accuracy: 0.6647
```



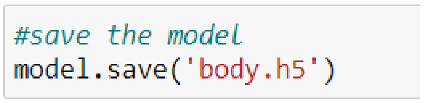
```
r = model1.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=25,
  steps_per_epoch=979//10,
  validation_steps=171//10)
```

```
C:\Users\sajus\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: UserWarning: `Model.fit_generator
` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports gener
ators.
```

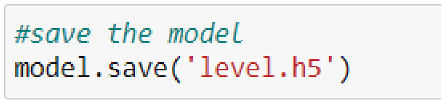
```
Epoch 1/25
97/97 [==============================] - 118s 1s/step - loss: 1.3991 - accuracy: 0.5351 - val_loss:
1.1652 - val_accuracy: 0.5706
Epoch 2/25
97/97 [==============================] - 115s 1s/step - loss: 0.7360 - accuracy: 0.7041 - val_loss:
0.8539 - val_accuracy: 0.6588
Epoch 3/25
97/97 [==============================] - 115s 1s/step - loss: 0.6582 - accuracy: 0.7430 - val_loss:
1.0068 - val_accuracy: 0.6176
Epoch 4/25
97/97 [==============================] - 115s 1s/step - loss: 0.5403 - accuracy: 0.7895 - val_loss:
1.1976 - val_accuracy: 0.5765
Epoch 5/25
97/97 [==============================] - 115s 1s/step - loss: 0.4344 - accuracy: 0.8328 - val_loss:
1.1396 - val_accuracy: 0.6235
Epoch 6/25
97/97 [==============================] - 115s 1s/step - loss: 0.4527 - accuracy: 0.8184 - val_loss:
1.0956 - val_accuracy: 0.6471
Epoch 7/25
97/97 [==============================] - 115s 1s/step - loss: 0.3089 - accuracy: 0.8803 - val_loss:
1.1560 - val_accuracy: 0.6059
```

Activity 8: Save The Model

The model is saved with .h5 extension as follows
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.



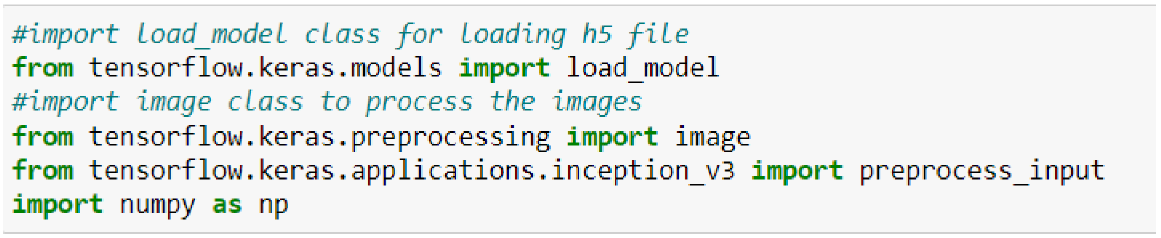
```
#save the model
model.save('body.h5')
```



```
#save the model
model.save('level.h5')
```

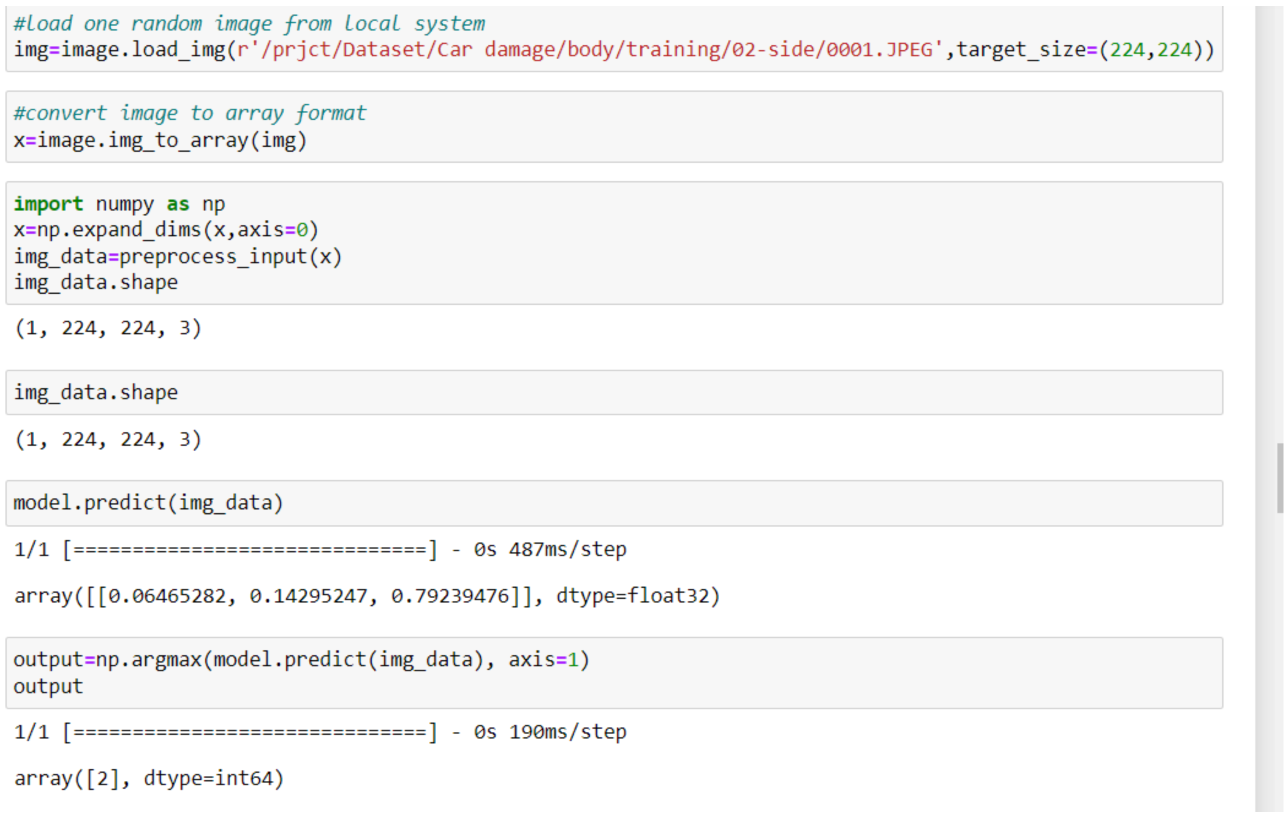
Activity 9: Test The Model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.
Load the saved model using load_model



```
#import load_model class for loading h5 file
from tensorflow.keras.models import load_model
#import image class to process the images
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import preprocess_input
import numpy as np
```

Taking an image as input and checking the results



```
#load one random image from local system
img=image.load_img(r'/prjct/Dataset/Car damage/body/training/02-side/0001.JPEG',target_size=(224,224))
```

```
#convert image to array format
x=image.img_to_array(img)
```

```
import numpy as np
x=np.expand_dims(x,axis=0)
img_data=preprocess_input(x)
img_data.shape
```

```
(1, 224, 224, 3)
```

```
img_data.shape
```

```
(1, 224, 224, 3)
```

```
model.predict(img_data)
```

```
1/1 [==============================] - 0s 487ms/step

array([[0.06465282, 0.14295247, 0.79239476]], dtype=float32)
```

```
output=np.argmax(model.predict(img_data), axis=1)
output
```

```
1/1 [==============================] - 0s 190ms/step

array([2], dtype=int64)
```

Milestone 4: Cloudant DB

Cloudant is a non-relational, distributed database service.

Below are steps that need to follow for creating and using Cloudant service.

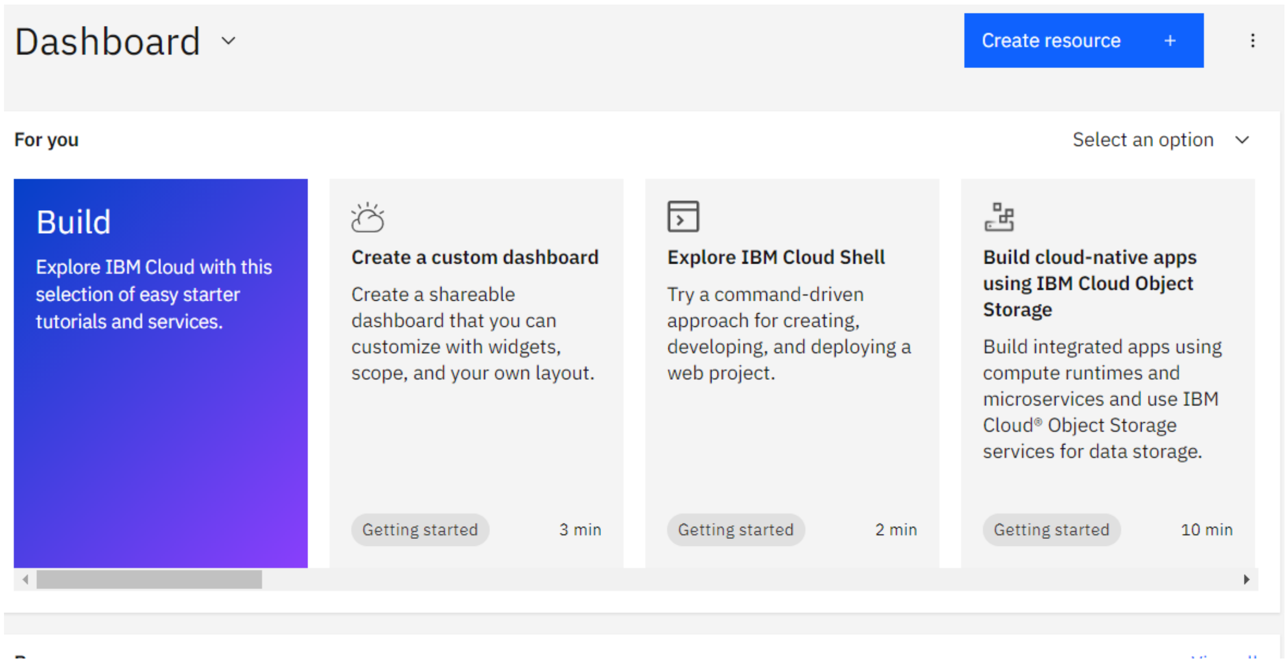
- Register & Login to IBM Cloud
- Create Service Instance
- Creating Service Credentials
- Launch Cloudant DB
- Create Database

Activity 1: Register & Login To IBM Cloud

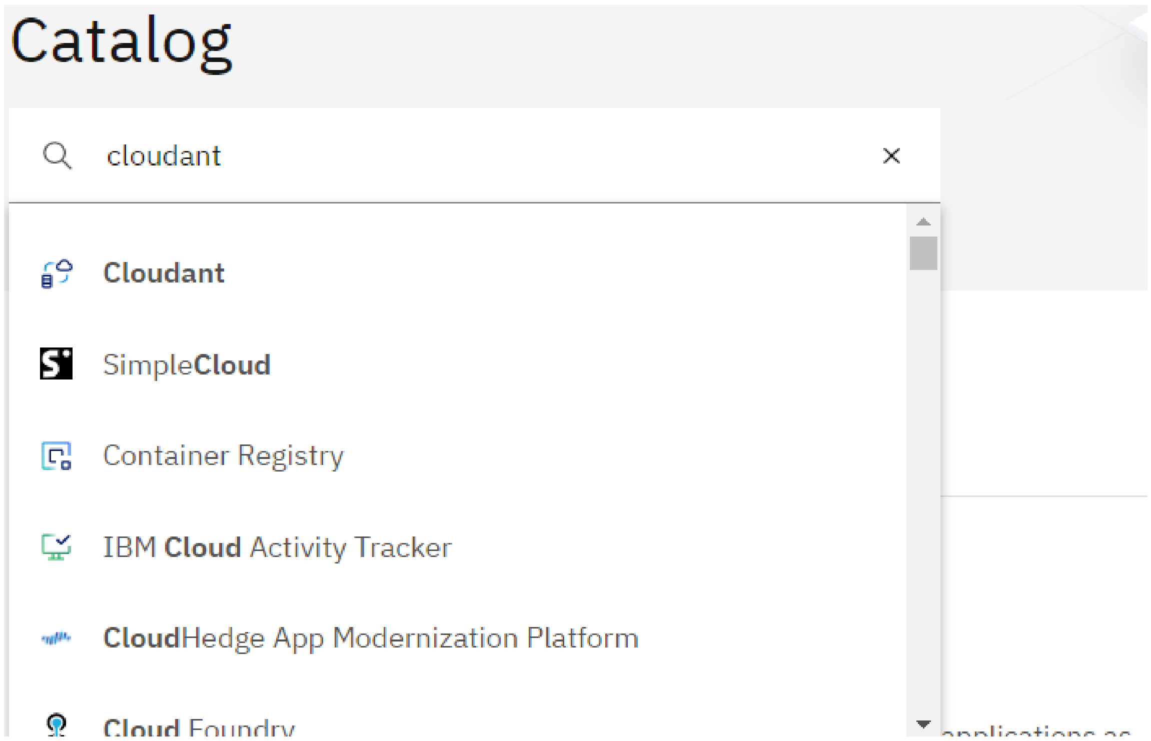
1. Register To IBM Cloud:- https://cloud.ibm.com/registration/trial
2. Sign in with your credentials: https://cloud.ibm.com/login

Activity 2: Create Service Instance

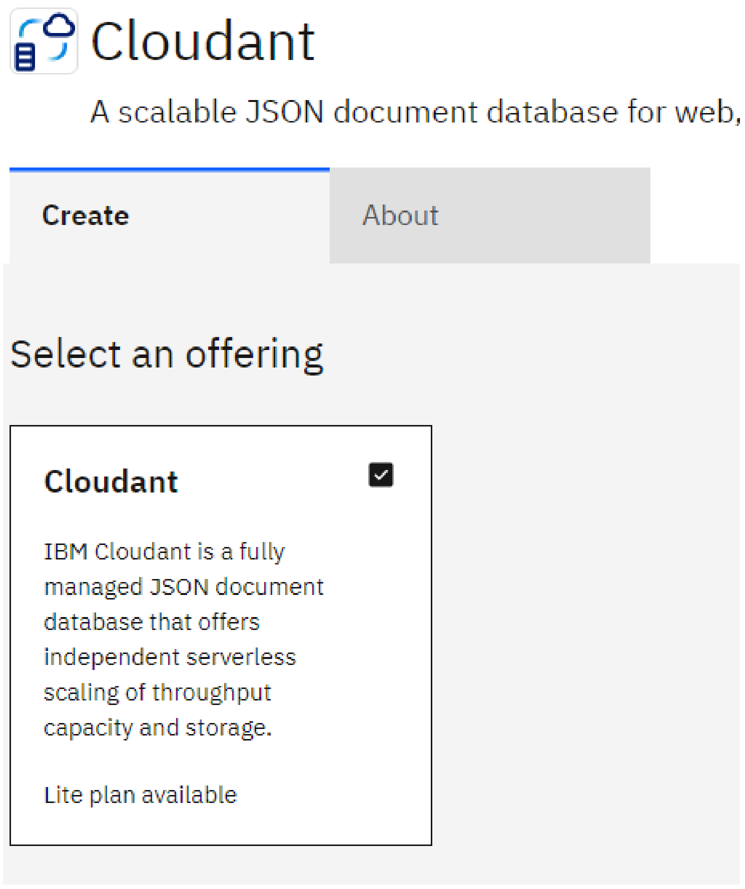
Log in to your IBM Cloud account, and click on Catalog


Dashboard
Create resource +
For you
Select an option
Build
Explore IBM Cloud with this selection of easy starter tutorials and services.
Create a custom dashboard
Create a shareable dashboard that you can customize with widgets, scope, and your own layout.
Getting started
3 min
Explore IBM Cloud Shell
Try a command-driven approach for creating, developing, and deploying a web project.
Getting started
2 min
Build cloud-native apps using IBM Cloud Object Storage
Build integrated apps using compute runtimes and microservices and use IBM Cloud® Object Storage services for data storage.
Getting started
10 min

Type Cloudant in the Search bar and click to open it.



Select an offering and an environment


Cloudant

A scalable JSON document database for web,

Create | About

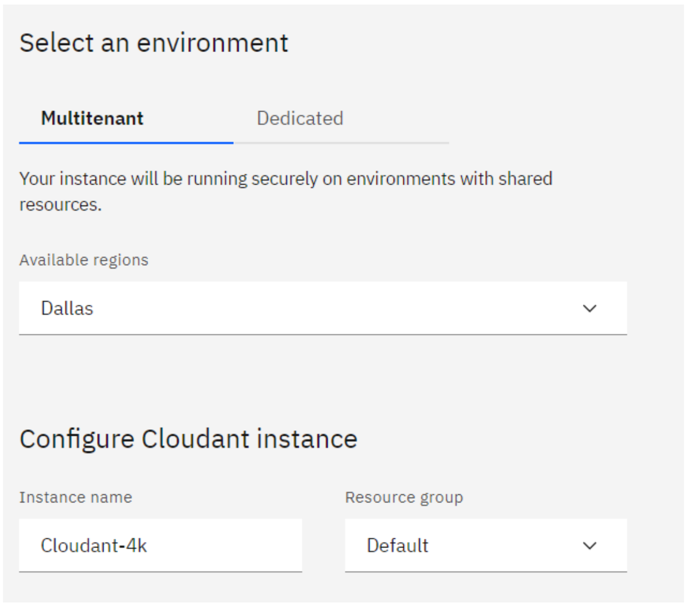
Select an offering

Cloudant ✓

IBM Cloudant is a fully managed JSON document database that offers independent serverless scaling of throughput capacity and storage.

Lite plan available

Select region as Dallas & Type an instance name then click on create service.


Select an environment

Multitenant | Dedicated

Your instance will be running securely on environments with shared resources.

Available regions: Dallas

Configure Cloudant instance

Instance name: Cloudant-4k

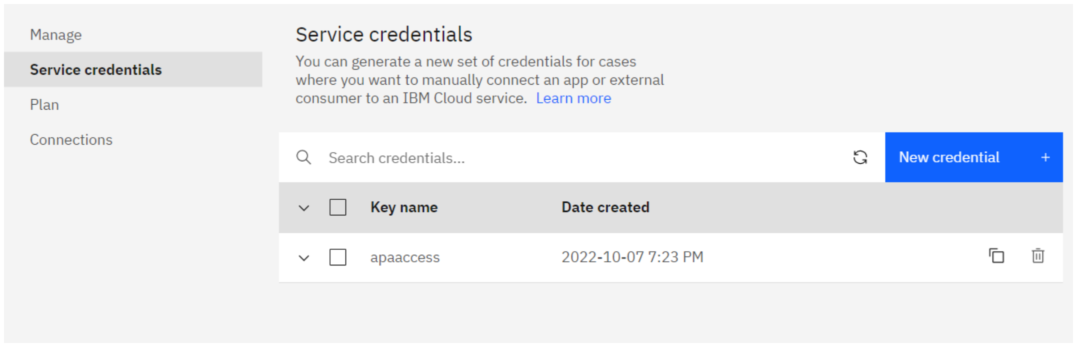
Resource group: Default

After you click create the system displays a message to say that the instance is being provisioned, which returns you to the Resource list. From the Resource list, you see that the status for your instance is, Provision in progress.

When the status changes to Active, click the instance.

Activity 3: Create Service Credentials

1. To create the connection information that your application needs to connect to the instance, click New credential.



Manage

Service credentials

Plan

Connections

Service credentials

You can generate a new set of credentials for cases where you want to manually connect an app or external consumer to an IBM Cloud service. Learn more

Search credentials...

New credential +

Key name	Date created
apaaccess	2022-10-07 7:23 PM

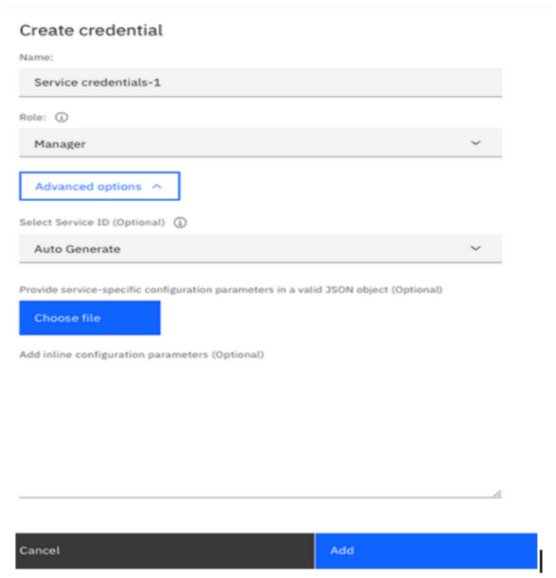
2. Enter a name for the new credential in the Add new credential window.

3. Accept the Manager role.

4. (Optional) Create a service ID or have one automatically generated for you.

5. (Optional) Add inline configuration parameters. This parameter isn't used by IBM Cloudant service credentials, so ignore it.

6. Click Add.



Create credential

Name:

Service credentials-1

Role:

Manager

Advanced options

Select Service ID (Optional)

Auto Generate

Provide service-specific configuration parameters in a valid JSON object (Optional)

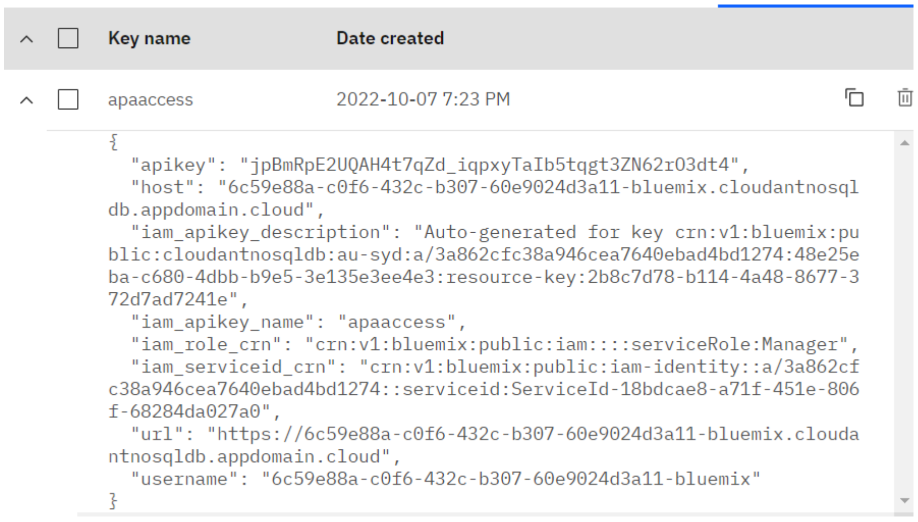
Choose file

Add inline configuration parameters (Optional)

Cancel

Add

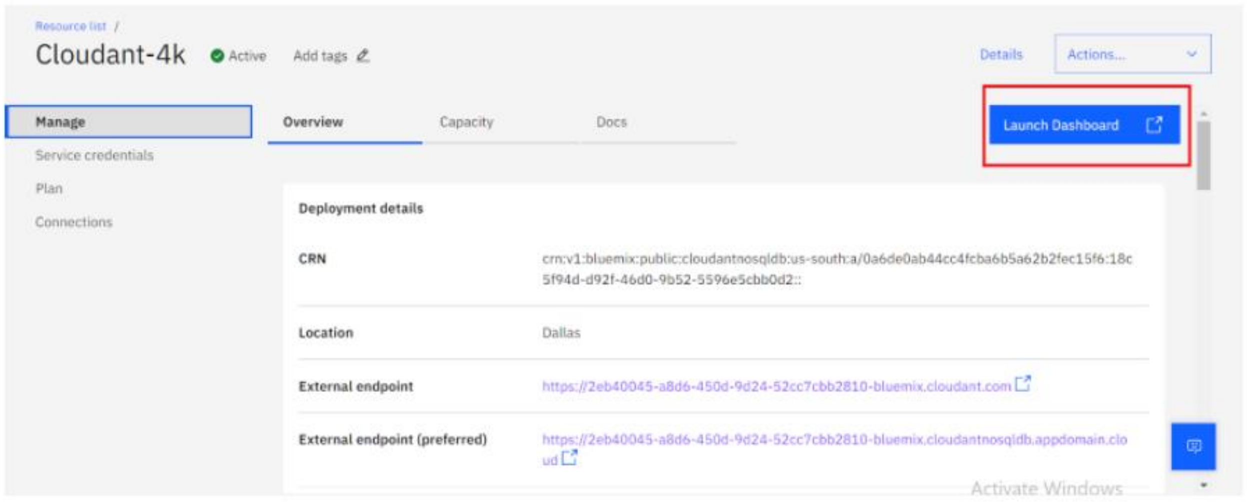
7. To see the credentials that are required to access the service, click the chevron.
8. The details for the service credentials open like the following example:



Key name	Date created
apaaccess	2022-10-07 7:23 PM

```
{
  "apikey": "jpBmRpE2UQAH4t7qZd_iqpxyTaIb5tqgt3ZN62r03dt4",
  "host": "6c59e88a-c0f6-432c-b307-60e9024d3a11-bluemix.cloudantnosql
db.appdomain.cloud",
  "iam_apikey_description": "Auto-generated for key crn:v1:bluemix:pu
blic:cloudantnosqldb:au-syd:a/3a862cfc38a946cea7640ebad4bd1274:48e25e
ba-c680-4dbb-b9e5-3e135e3ee4e3:resource-key:2b8c7d78-b114-4a48-8677-3
72d7ad7241e",
  "iam_apikey_name": "apaaccess",
  "iam_role_crn": "crn:v1:bluemix:public:iam::::serviceRole:Manager",
  "iam_serviceid_crn": "crn:v1:bluemix:public:iam-identity::a/3a862cf
c38a946cea7640ebad4bd1274::serviceid:ServiceId-18bdcae8-a71f-451e-806
f-68284da027a0",
  "url": "https://6c59e88a-c0f6-432c-b307-60e9024d3a11-bluemix.clouda
ntnosqldb.appdomain.cloud",
  "username": "6c59e88a-c0f6-432c-b307-60e9024d3a11-bluemix"
}
```

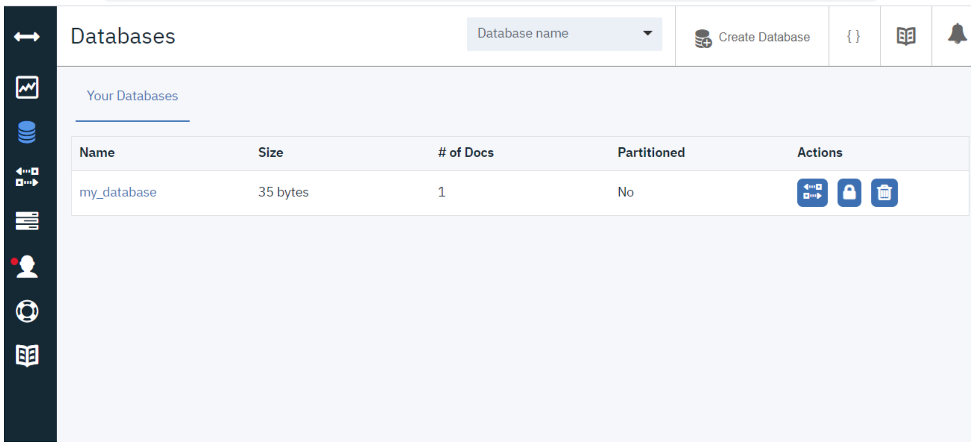
Activity 4: Launch Cloudant DB



Deployment details

CRN	crn:v1:bluemix:public:cloudantnosqldb:us-south:a/0a6de0ab44cc4fcba6b5a62b2fec15f6:18c5f94d-d92f-46d0-9b52-5596e5cbb0d2::
Location	Dallas
External endpoint	https://2eb40045-a8d6-450d-9d24-52cc7cbb2810-bluemix.cloudant.com
External endpoint (preferred)	https://2eb40045-a8d6-450d-9d24-52cc7cbb2810-bluemix.cloudantnosqldb.appdomain.cloud

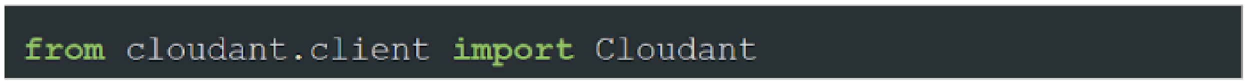
Your Cloudant DB launches



Name	Size	# of Docs	Partitioned	Actions
my_database	35 bytes	1	No	

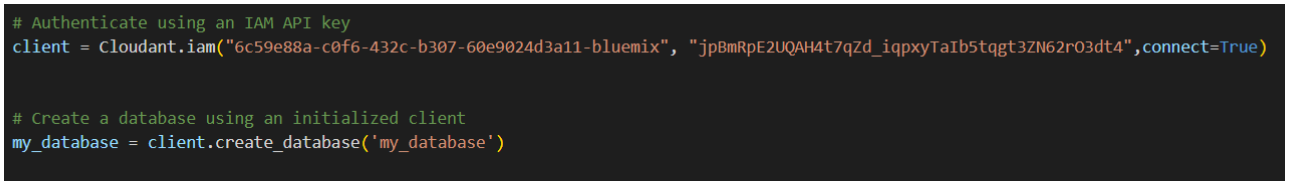
Activity 5: Create Database

In order to manage a connection from a local system, you must first initialize the connection by constructing a Cloudant client. We need to import the cloudant library.



```python
from cloudant.client import Cloudant
```

IBM Cloud Identity & Access Management enables you to securely authenticate users and control access to all cloud resources consistently in the IBM Bluemix Cloud Platform.



```python
# Authenticate using an IAM API key
client = Cloudant.iam("6c59e88a-c0f6-432c-b307-60e9024d3a11-bluemix", "jpBmRpE2UQAH4t7qZd_iqpxyTaIb5tqgt3ZN62rO3dt4",connect=True)


# Create a database using an initialized client
my_database = client.create_database('my_database')
```

In the above cloudant.iam() method we have to give a username & apikey to build the connection with cloudant DB.
Once a connection is established you can then create a database, and open an existing database.
Create a database as my_database.

Milestone 5: Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.
This section has the following tasks

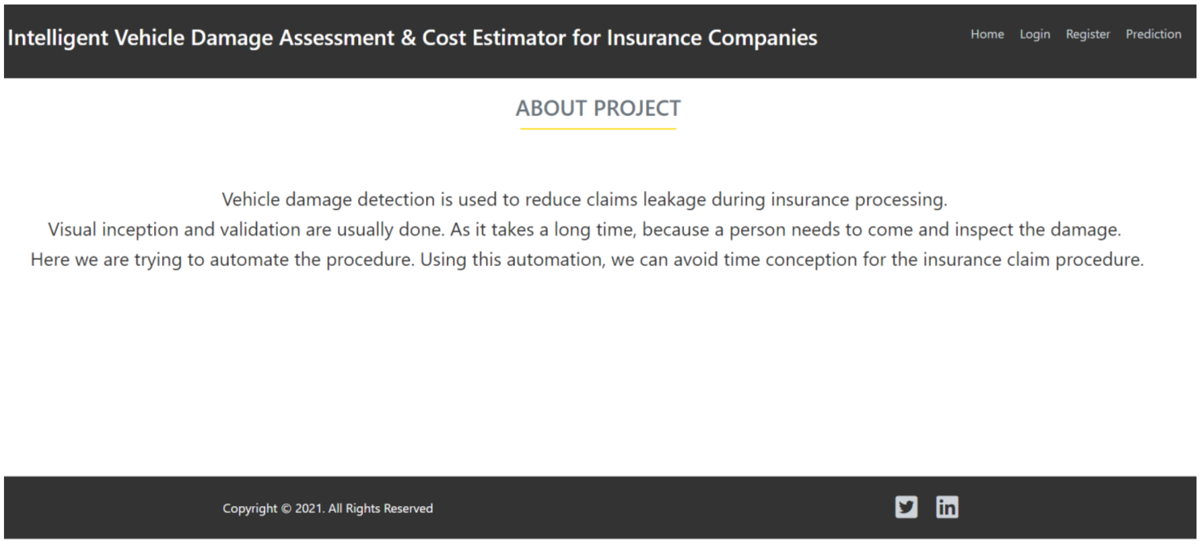
- Building HTML Pages
- Building server side script

Activity 1: Building Html Pages:

For this project create one HTML file namely

- Index.html

Let’s see how our index.html page looks like:

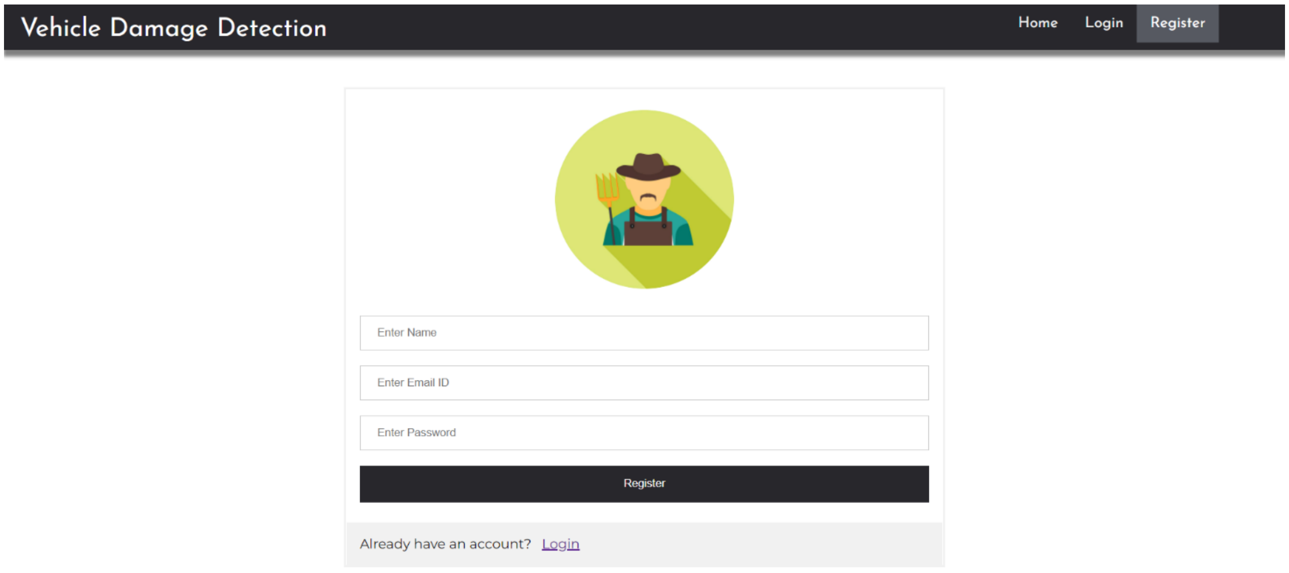

Intelligent Vehicle Damage Assessment & Cost Estimator for Insurance Companies

ABOUT PROJECT

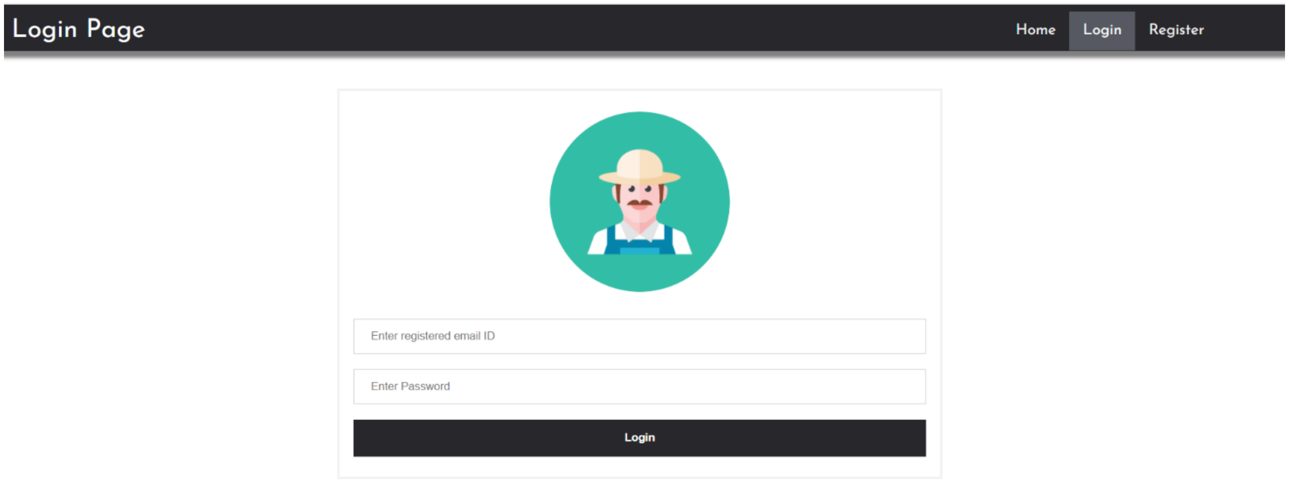
Vehicle damage detection is used to reduce claims leakage during insurance processing.
Visual inception and validation are usually done. As it takes a long time, because a person needs to come and inspect the damage.
Here we are trying to automate the procedure. Using this automation, we can avoid time conception for the insurance claim procedure.

Copyright © 2021. All Rights Reserved

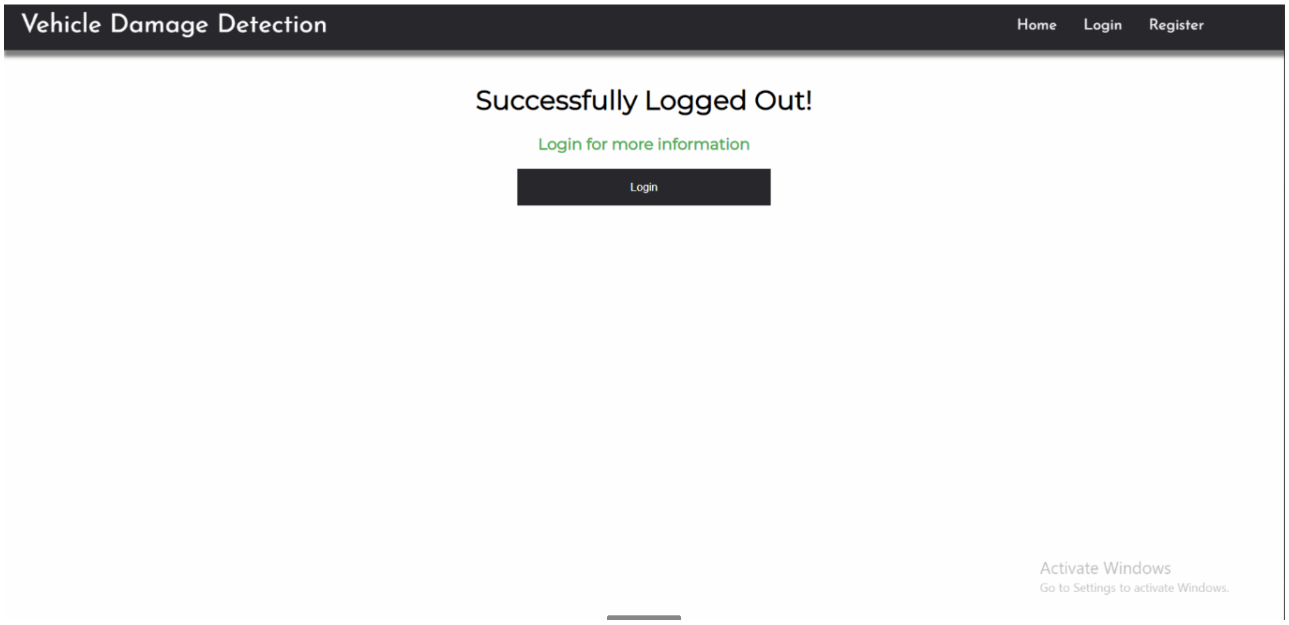
When you click on Register button on the top, you will be redirecting to the following page



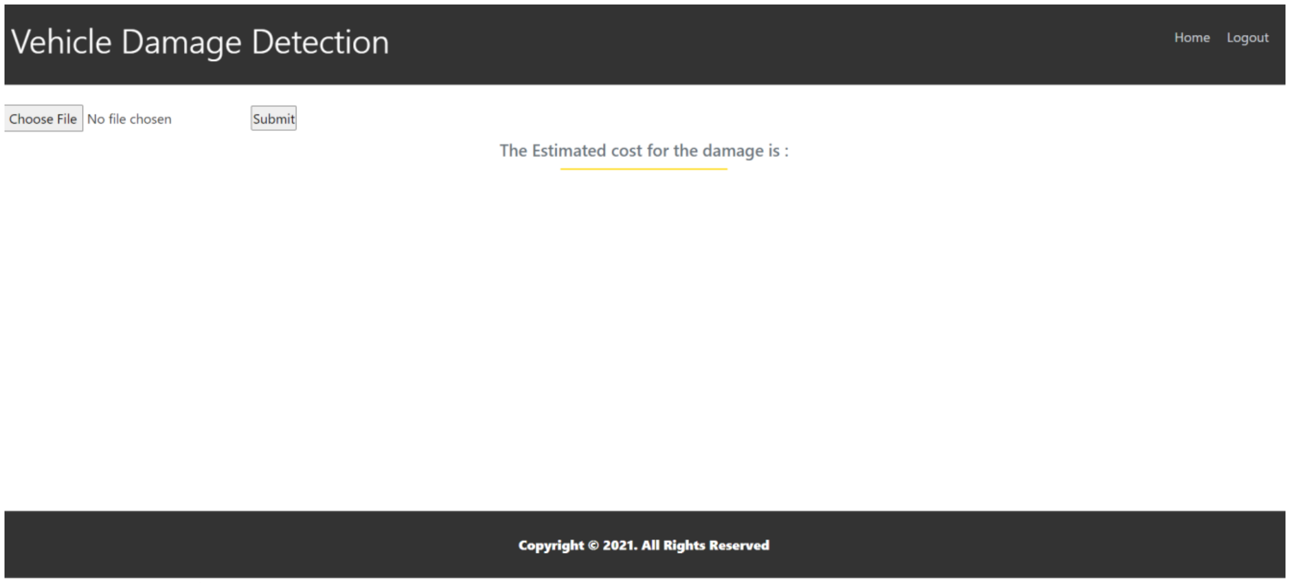
When you click on the login button, it will redirect you to the below page, when hover on the each model name you can expand and see the architecture and description of the model and link to study about the model.



Logout page:

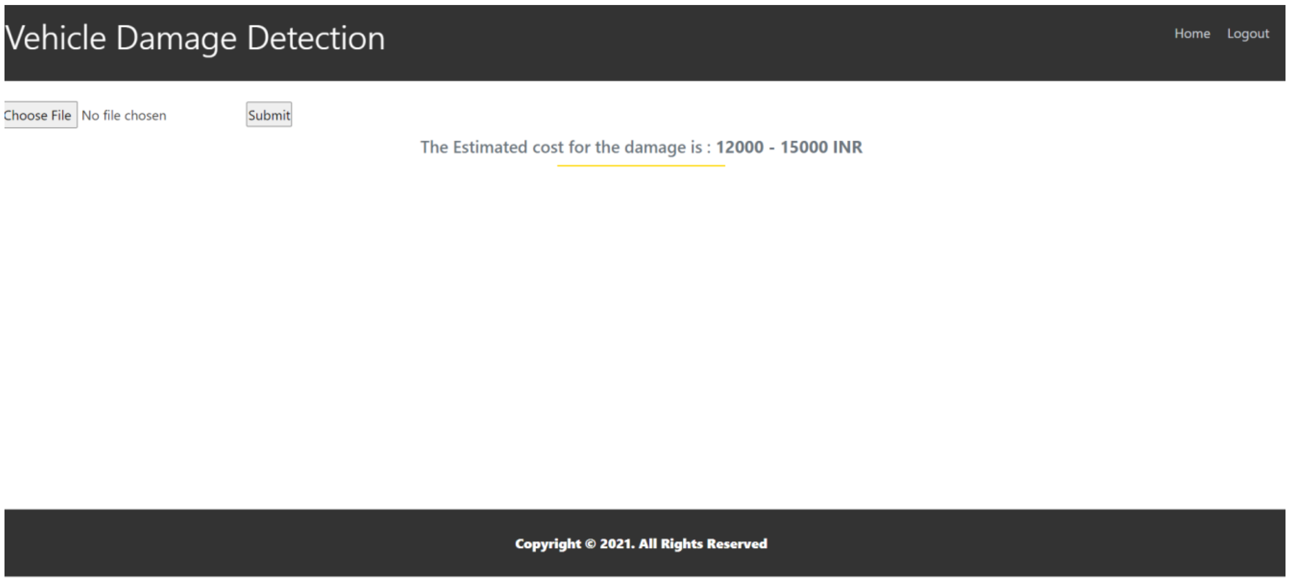


Prediction Page:



The Estimated cost for the damage is :

Copyright © 2021. All Rights Reserved

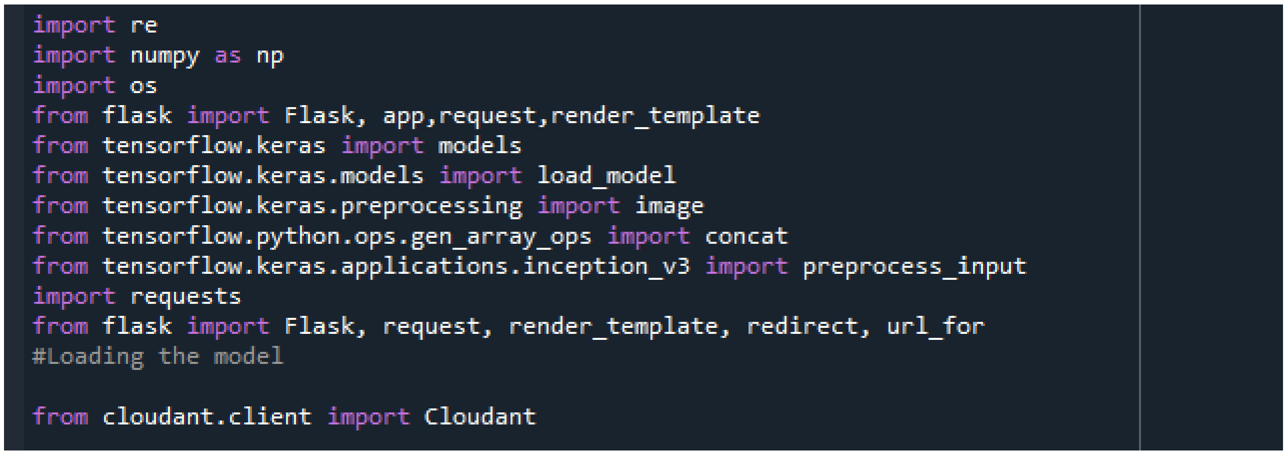


The Estimated cost for the damage is : **12000 - 15000 INR**

Copyright © 2021. All Rights Reserved

Activity 2: Build Python code:

Import the libraries



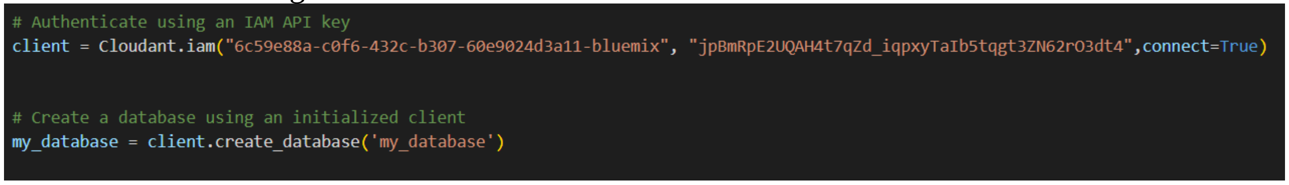
```python
import re
import numpy as np
import os
from flask import Flask, app,request,render_template
from tensorflow.keras import models
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.python.ops.gen_array_ops import concat
from tensorflow.keras.applications.inception_v3 import preprocess_input
import requests
from flask import Flask, request, render_template, redirect, url_for
#Loading the model

from cloudant.client import Cloudant
```

Loading the saved model and initializing the flask app

Importing the flask module in the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as an argument Pickle library to load the model file.

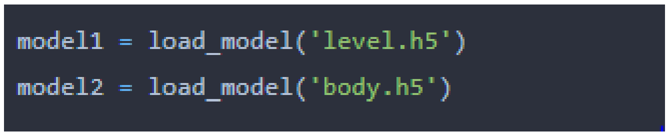
Once after loading the libraries, we need to authenticate with cloudant DB of IBM using IAM API key

Create a database using the below code

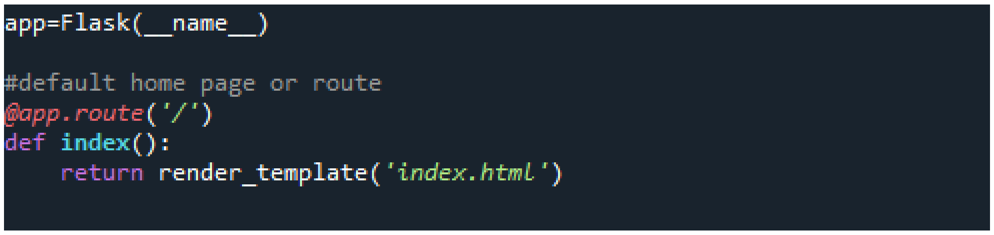


```python
# Authenticate using an IAM API key
client = Cloudant.iam("6c59e88a-c0f6-432c-b307-60e9024d3a11-bluemix", "jpBmRpE2UQAH4t7qZd_iqpxyTaIb5tqgt3ZN62rO3dt4",connect=True)


# Create a database using an initialized client
my_database = client.create_database('my_database')
```

Creating our flask application and loading our model by using the load_model method



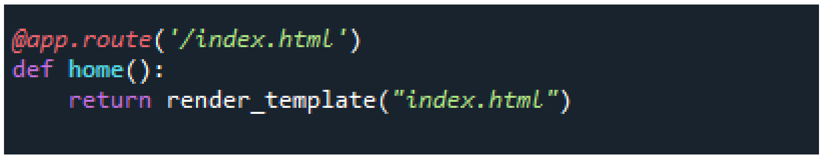
```python
model1 = load_model('level.h5')
model2 = load_model('body.h5')
```



```python
app=Flask(__name__)

#default home page or route
@app.route('/')
def index():
    return render_template('index.html')
```

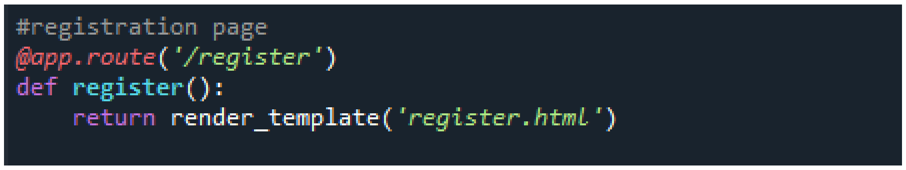
Render HTML pages:

Once we uploaded the file into the app, then verifying the file uploaded properly or not. Here we will be using declared constructor to route to the HTML page which we have created earlier.



```
@app.route('/index.html')
def home():
    return render_template("index.html")
```

When you click on register on the home page, it will redirect you to the register page

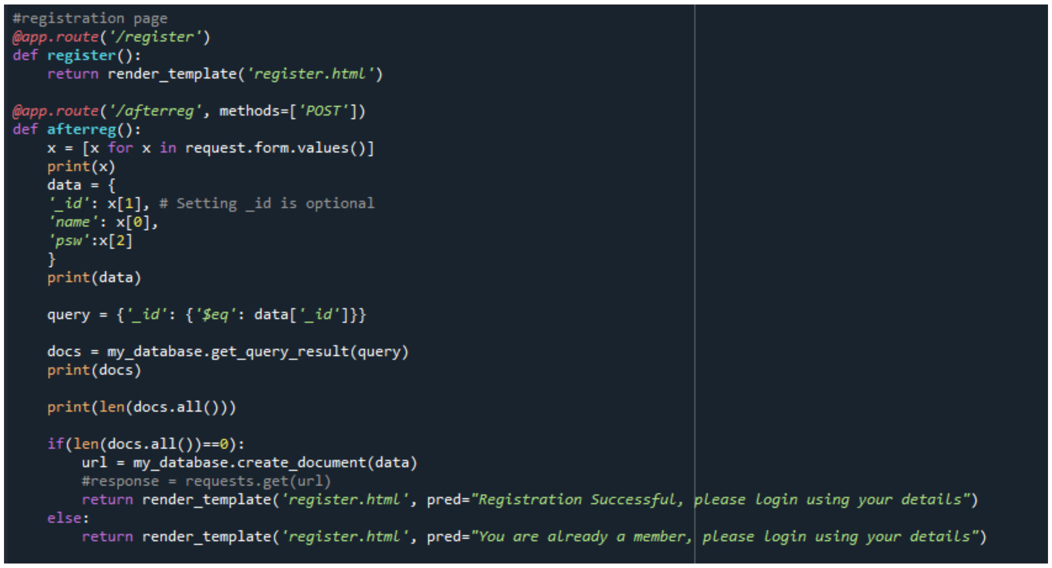


```
#registration page
@app.route('/register')
def register():
    return render_template('register.html')
```

When you click on register on the home page, it will redirect you to the register page

Configure the registration page

Based on user input into the registration form we stored it in a data dictionary then we can validate the data using _id parameter with user input that we can store it on the query variable then we can validate bypassing the query variable into the my_database.get_user_result() method.Then we can check the docs length by using len(docs.all()) function.If the length of docs is 0 then the user will register successfully on the platform and user data will store in the database. Otherwise its shows the message as a user already registered please login and use our web application for Car Damage prediction.

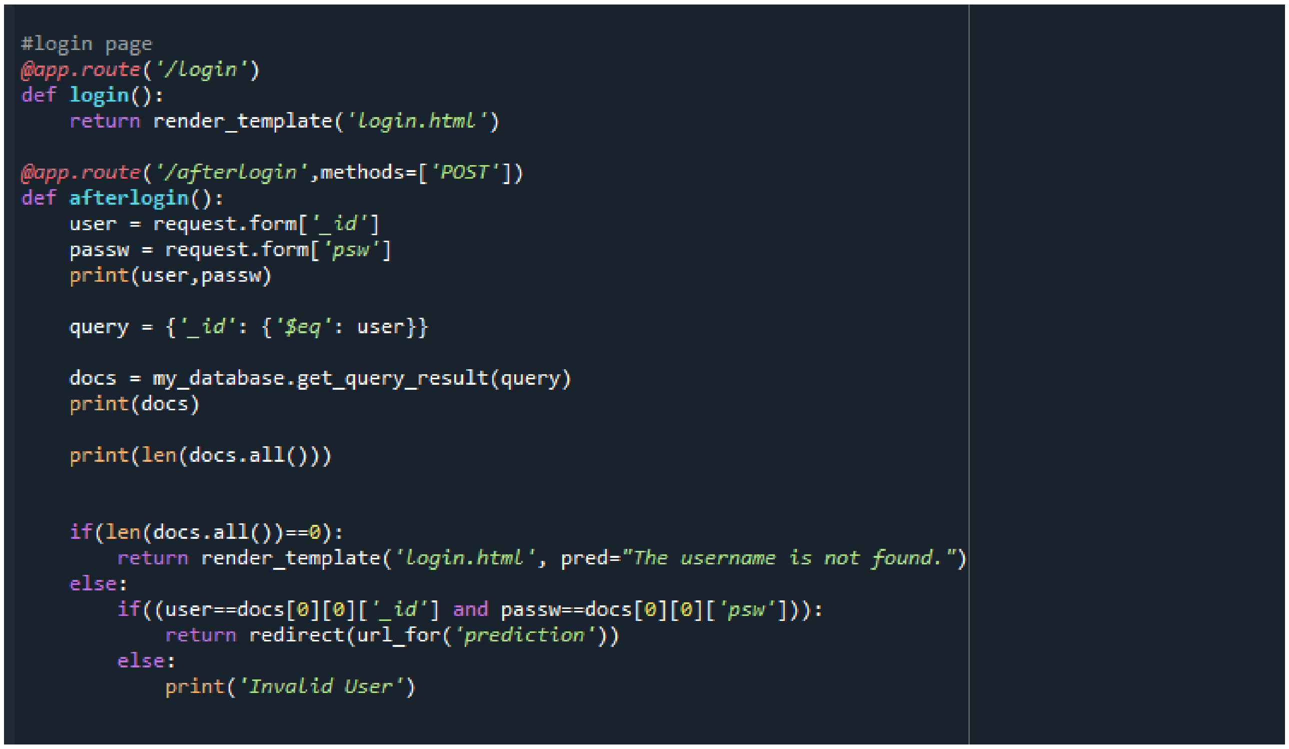


```
#registration page
@app.route('/register')
def register():
    return render_template('register.html')

@app.route('/afterreg', methods=['POST'])
def afterreg():
    x = [x for x in request.form.values()]
    print(x)
    data = {
    '_id': x[1], # Setting _id is optional
    'name': x[0],
    'psw':x[2]
    }
    print(data)

    query = {'_id': {'$eq': data['_id']}}

    docs = my_database.get_query_result(query)
    print(docs)

    print(len(docs.all()))

    if(len(docs.all())==0):
        url = my_database.create_document(data)
        #response = requests.get(url)
        return render_template('register.html', pred="Registration Successful, please login using your details")
    else:
        return render_template('register.html', pred="You are already a member, please login using your details")
```

Once you register, you need to login through the login page, for routing to the login page you need to use the below code

Configure the login page

Based on user input into the login form we stored user id and password into the (user,passw) variables. Then we can validate the credentials using _id parameter with user input that we can store it on query variable then we can validate by passing the query variable into the my_database.get_user_result() method.Then we can check the docs length by using len(docs.all()) function.If the length of doc is 0 then it means the username is not found. Otherwise its validate the data that is stored on the database and check the username & password. If it's matched then the user will be able to login and use our web application for Car Damage prediction. Otherwise the user needs to provide the correct credentials.

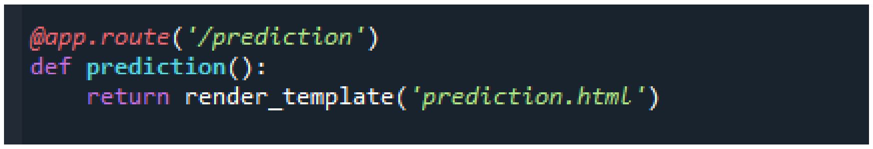


```python
#login page
@app.route('/login')
def login():
    return render_template('login.html')

@app.route('/afterlogin',methods=['POST'])
def afterlogin():
    user = request.form['_id']
    passw = request.form['psw']
    print(user,passw)

    query = {'_id': {'$eq': user}}

    docs = my_database.get_query_result(query)
    print(docs)

    print(len(docs.all()))


    if(len(docs.all())==0):
        return render_template('login.html', pred="The username is not found.")
    else:
        if((user==docs[0][0]['_id'] and passw==docs[0][0]['psw'])):
            return redirect(url_for('prediction'))
        else:
            print('Invalid User')
```

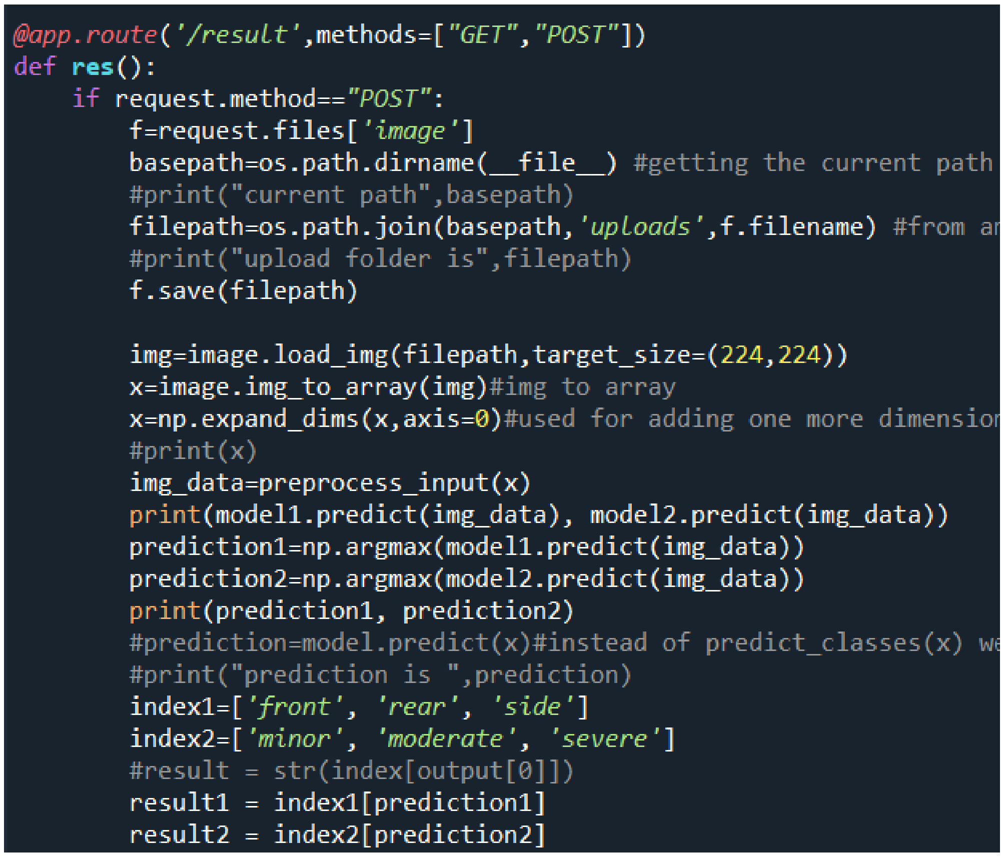
Routing to the html Page

Here, the declared constructor is used to route to the HTML page created earlier.

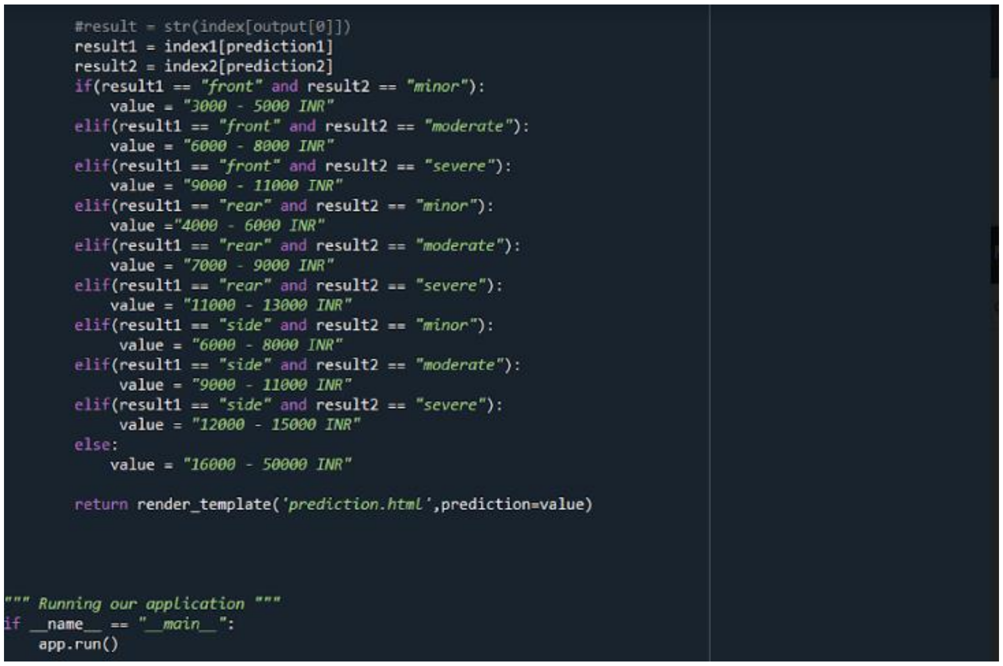
In the above example, ‘/’ URL is bound with index.html function. Hence, when the home page of web server is opened in browser, the html page will be rendered. Whenever you browse an image from the html page this photo can be accesses through POST or GET Method.



```python
@app.route('/prediction')
def prediction():
    return render_template('prediction.html')
```



```
@app.route('/result',methods=["GET","POST"])
def res():
    if request.method=="POST":
        f=request.files['image']
        basepath=os.path.dirname(__file__) #getting the current path
        #print("current path",basepath)
        filepath=os.path.join(basepath,'uploads',f.filename) #from ar
        #print("upload folder is",filepath)
        f.save(filepath)

        img=image.load_img(filepath,target_size=(224,224))
        x=image.img_to_array(img)#img to array
        x=np.expand_dims(x,axis=0)#used for adding one more dimensior
        #print(x)
        img_data=preprocess_input(x)
        print(model1.predict(img_data), model2.predict(img_data))
        prediction1=np.argmax(model1.predict(img_data))
        prediction2=np.argmax(model2.predict(img_data))
        print(prediction1, prediction2)
        #prediction=model.predict(x)#instead of predict_classes(x) we
        #print("prediction is ",prediction)
        index1=['front', 'rear', 'side']
        index2=['minor', 'moderate', 'severe']
        #result = str(index[output[0]])
        result1 = index1[prediction1]
        result2 = index2[prediction2]
```

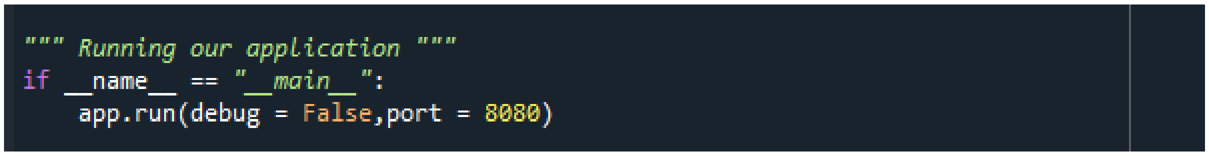


```
        #result = str(index[output[0]])
        result1 = index1[prediction1]
        result2 = index2[prediction2]
        if(result1 == "front" and result2 == "minor"):
            value = "3000 - 5000 INR"
        elif(result1 == "front" and result2 == "moderate"):
            value = "6000 - 8000 INR"
        elif(result1 == "front" and result2 == "severe"):
            value = "9000 - 11000 INR"
        elif(result1 == "rear" and result2 == "minor"):
            value ="4000 - 6000 INR"
        elif(result1 == "rear" and result2 == "moderate"):
            value = "7000 - 9000 INR"
        elif(result1 == "rear" and result2 == "severe"):
            value = "11000 - 13000 INR"
        elif(result1 == "side" and result2 == "minor"):
             value = "6000 - 8000 INR"
        elif(result1 == "side" and result2 == "moderate"):
             value = "9000 - 11000 INR"
        elif(result1 == "side" and result2 == "severe"):
             value = "12000 - 15000 INR"
        else:
            value = "16000 - 50000 INR"

        return render_template('prediction.html',prediction=value)



""" Running our application """
if __name__ == "__main__":
    app.run()
```

Here we are defining a function which requests the browsed file from the html page using the post method. The requested picture file is then saved to the uploads folder in this same directory using OS library. Using the load image class from Keras library we are retrieving the saved picture from the path declared. We are applying some image processing techniques and then sending that preprocessed image to the model for predicting the class. This returns the numerical value of a class (like 0,1 ,2 etc.) which lies in the 0th index of the variable preds. This numerical value is passed to the index variable declared. This returns the name of the class. This name is rendered to the predict variable used in the html page.

Finally, Run the application

This is used to run the application in localhost.



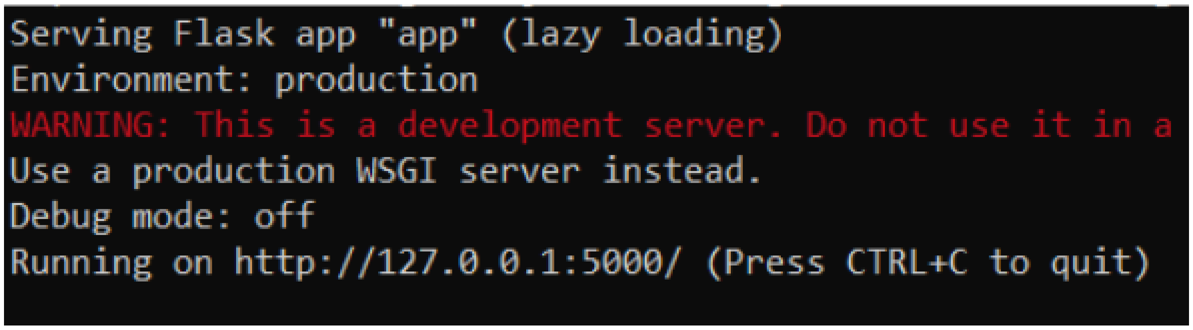
```
""" Running our application """
if __name__ == "__main__":
    app.run(debug = False,port = 8080)
```

Activity 3: Run The Application

- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type the "python app.py" command.
- It will show the local host where your app is running on **http://127.0.0.1.8080/**
- Copy that localhost URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

1: Run the application

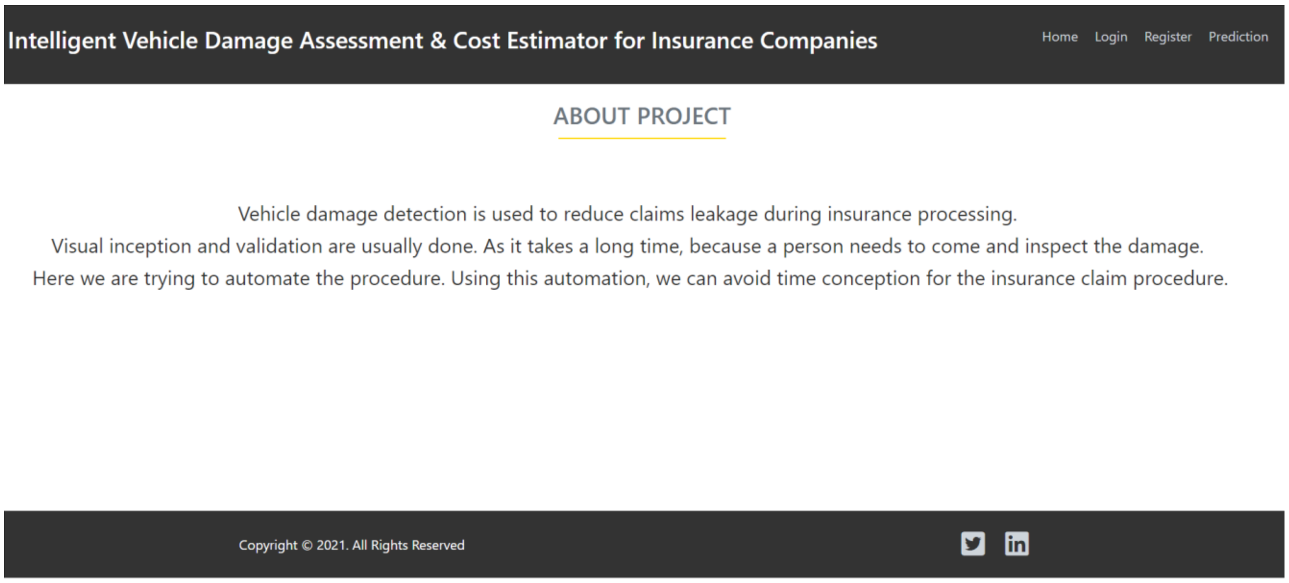
In the anaconda prompt, navigate to the folder in which the flask app is present. When the python file is executed the localhost is activated on 5000 port and can be accessed through it.



```
Serving Flask app "app" (lazy loading)
Environment: production
WARNING: This is a development server. Do not use it in a
Use a production WSGI server instead.
Debug mode: off
Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

2: Open the browser and navigate to localhost:5000 to check your application

The home page looks like this. You can click on login or register.


Intelligent Vehicle Damage Assessment & Cost Estimator for Insurance Companies

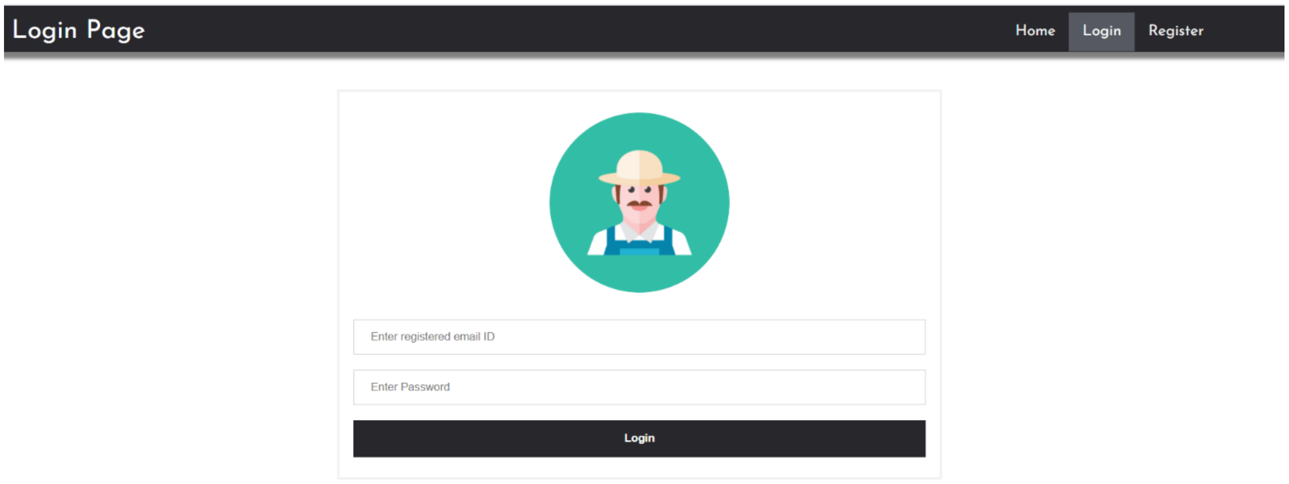
Home Login Register Prediction

ABOUT PROJECT

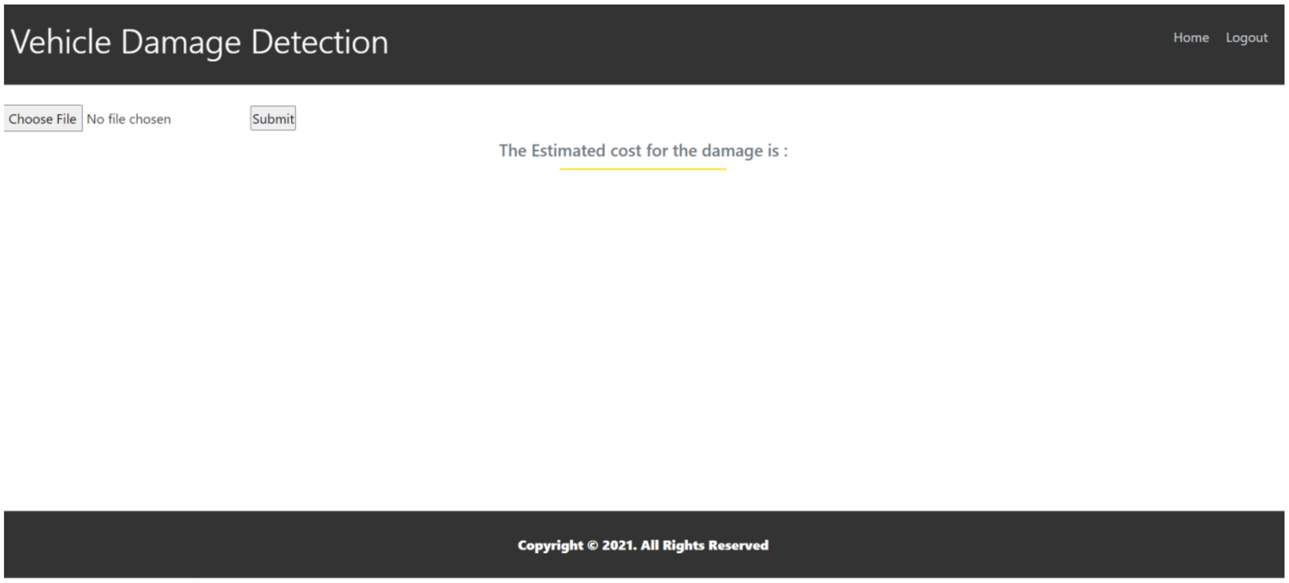
Vehicle damage detection is used to reduce claims leakage during insurance processing.
Visual inception and validation are usually done. As it takes a long time, because a person needs to come and inspect the damage.
Here we are trying to automate the procedure. Using this automation, we can avoid time conception for the insurance claim procedure.

Copyright © 2021. All Rights Reserved

While logging in you need to provide your registered credentials,



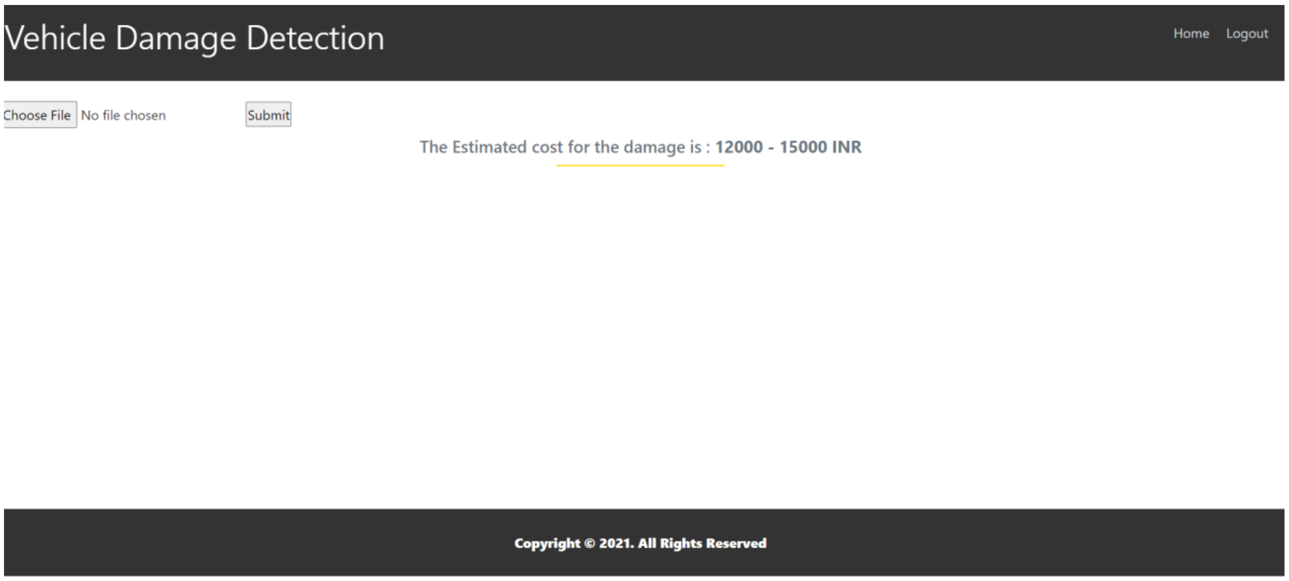
After successfully login , you will redirect to the prediction page where we have to upload the image to predict the outcomes.


Vehicle Damage Detection

The Estimated cost for the damage is :

Copyright © 2021. All Rights Reserved

Output:


Vehicle Damage Detection

The Estimated cost for the damage is : 12000 - 15000 INR

Copyright © 2021. All Rights Reserved