Car Damage Detection

To predict the damage of a car and it’s severity from it’s image and provide cost estimate

By:

Vishvambar Panth S

Rashmika B

Saravanakumar M

Saravanan B L

Vayalpati Monika

Sethuraman

TEAM 2

Motivation

01 Cars have a central role in today’s world. **Non automated visual inspection** of cars is a common task in some businesses,mainly insurance companies.

02 Two types of inspection- new coverage and issues reported. Most of the inspections are only done by sampling. This would **increase the cost to be given to the expert**, which would eventually cause delays to the customers and company alike.

03 The customer would **no longer have the convenience** of quickly buying an insurance, since he would have to wait for the inspection to be performed before the effectiveness of the insurance was granted.

04 These inconveniences would be mitigated if the client uploaded some **photos of the car** to be automatically inspected by a system that could assert if the car has damages or not.

Objective

- ➔ To detect the presence of damage with car images
- ➔ To predict the severity of the damage if found
- ➔ To give a rough estimate of the cost of the damage

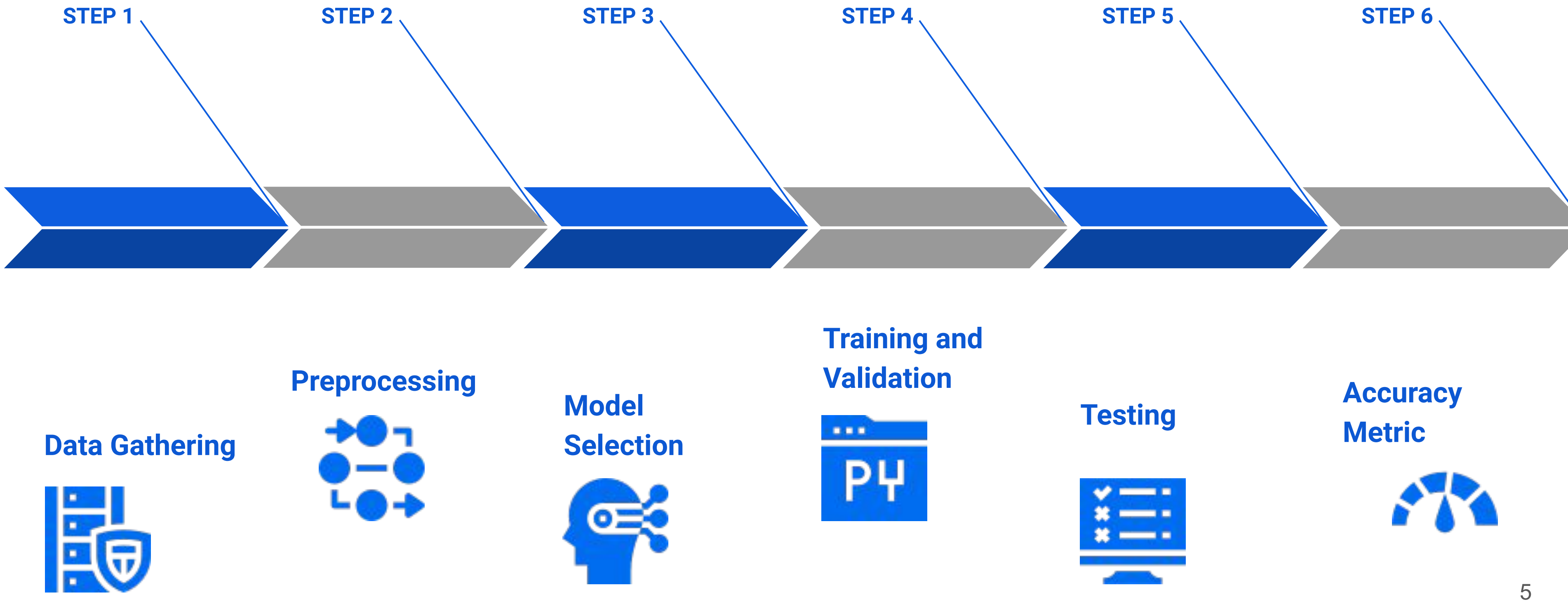
Abstract

The goal of this project is to develop a basic system capable of automatically identifying the presence of damages in cars and predicting the severity of the damages.

The recent advances in computer vision largely due to the adoption of fast, scalable and end to end trainable Convolution Neural Networks(CNN’s) makes it technically feasible to recognize vehicle damages using deep convolutional networks.

Using CNN models pretrained on the obtained dataset and applying various techniques to improve the performance of our system, we were able to achieve good accuracy of 85%, which was significantly better.

Workflow


STEP 1
Data Gathering
STEP 2
Preprocessing
STEP 3
Model Selection
STEP 4
Training and Validation
STEP 5
Testing
STEP 6
Accuracy Metric

Dataset

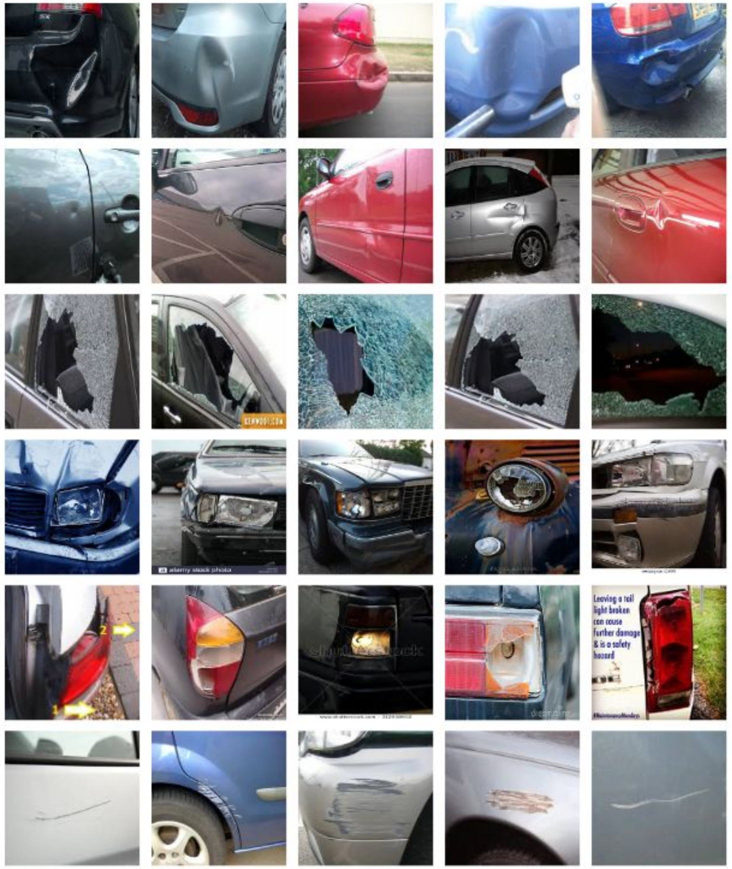
The dataset that we used has a collection of about 1000 images with two broad categories- Damage, Severity.

Damaged - This dataset contains about 2 classes with labels as ‘damaged’ and ‘whole’.

Severity - The mini dataset contains 2 classes with labels as ‘severe’ and ‘minor’.

Few images were manually collected from the web and annotated as severe and minor.

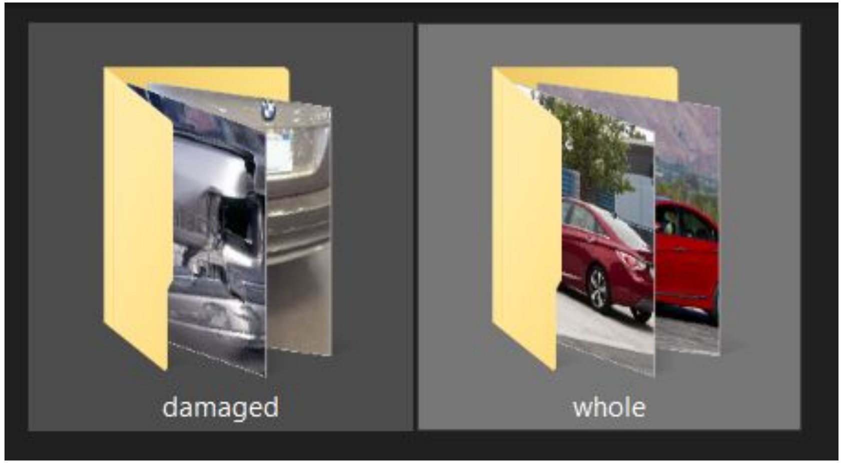
The labels are converted into binary classes and binary classification is performed.



Pre-processing

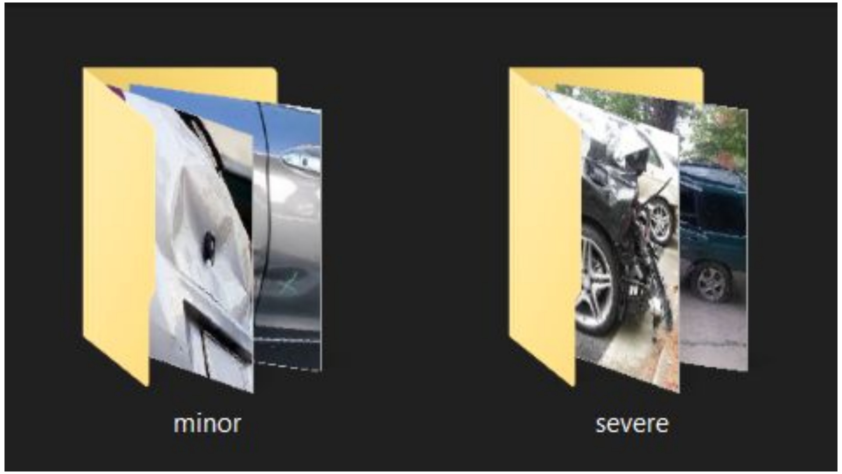
<u>Damage Detection :</u>

The dataset containing 1150 images of damaged and undamaged cars was collected from an utility website called Kaggle which had around 2000 images. The data was cleaned to have only Jpeg/Jpg images.



<u>Severity :</u>

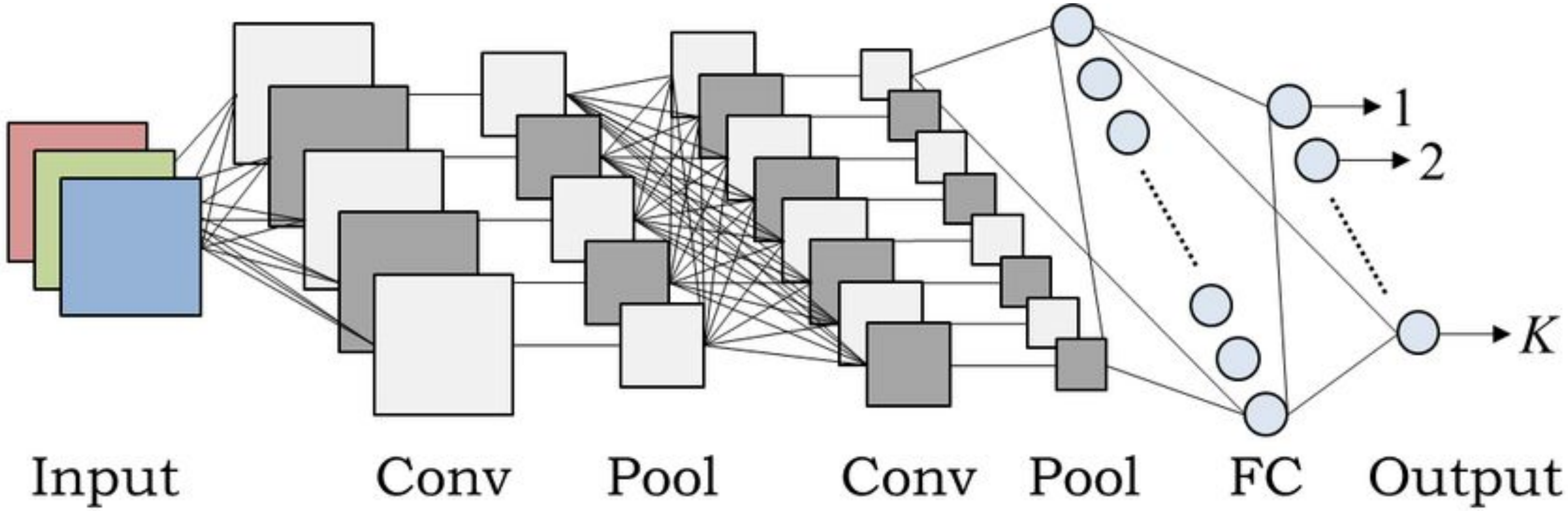
Around 170 images were collected from Internet manually and hand labelled into two categories - minor and severe damaged cars.



Model Used - CNN

- **Convolutional Neural Network.**
- More scalable approach to image classification and computer vision tasks.
- The architecture of a convolutional neural network is a multi-layered feed-forward neural network, made by stacking many hidden layers on top of each other in sequence.
- Three main types of layers:
  - **Convolutional layer:** Compute output of Neurons.
  - Pooling layer
    - **Max pooling-** As the filter moves across the input, it selects the pixel with the maximum value to send to the output array.
  - **Fully-connected (FC) layer-** Used to connect neurons between two different layers.

CNN - Simple Architecture



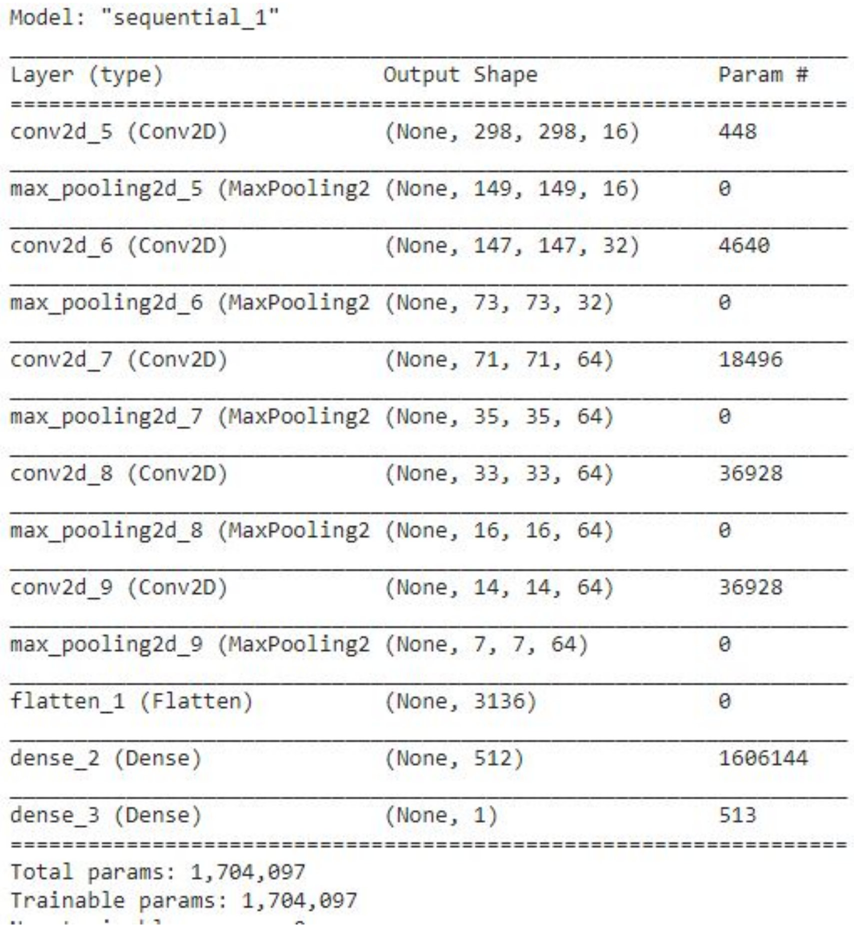
Advantages of the CNN Model

- Weight Sharing
- Accuracy of difficult classification tasks
- Automatically detects Important features without human supervision
- Deep convolutional networks are flexible and work well on image data.
- Memory Saving.
- Independent of Transformations
- Significant Speed advantage.

Architecture

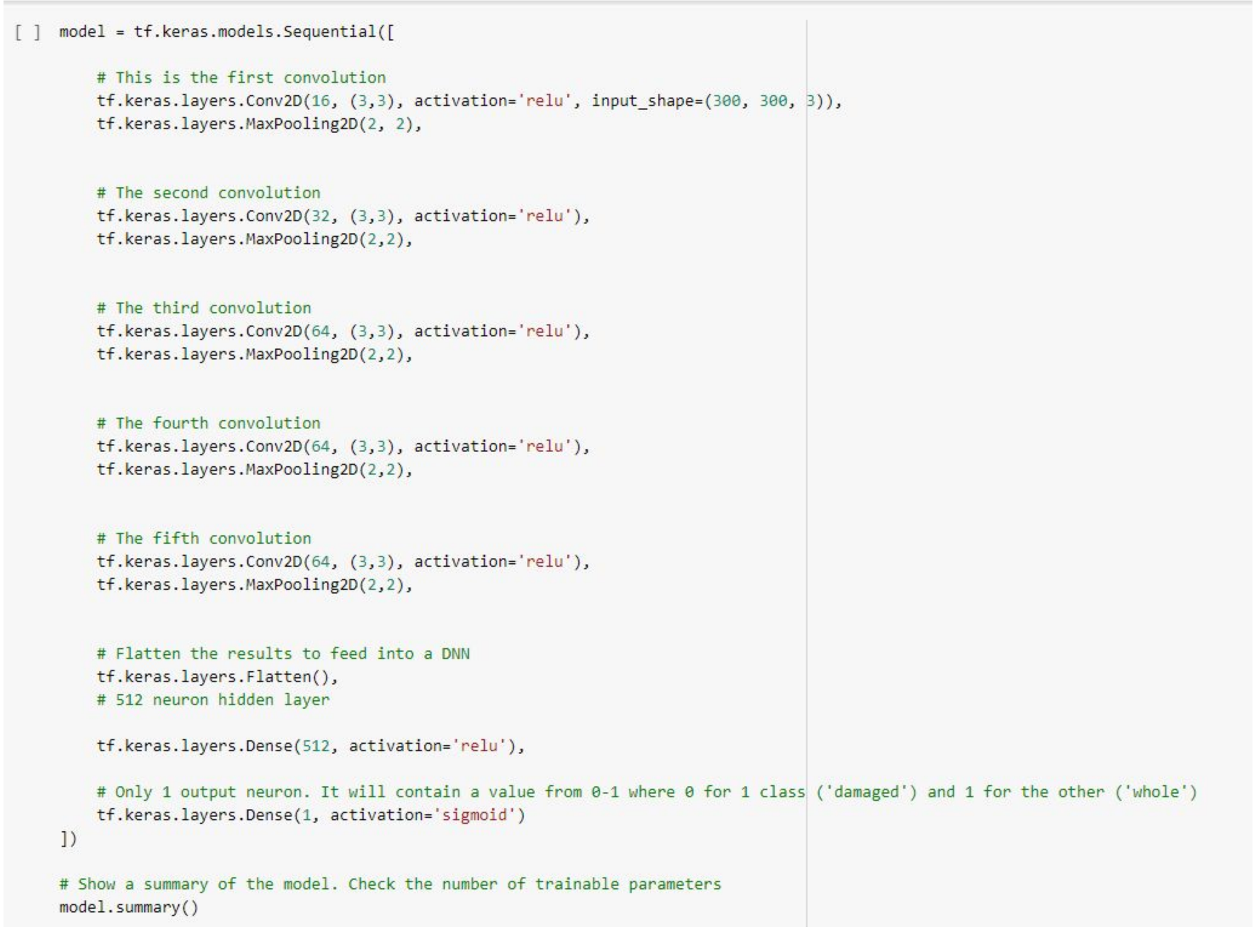
Contains:

- Convolution Layers
- Max Pooling
- Dense Layers
    - ReLu
    - Sigmoid



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 298, 298, 16)      448
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 149, 149, 16)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 147, 147, 32)      4640
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 73, 73, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 71, 71, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 35, 35, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 33, 33, 64)        36928
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               1606144
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 513
=================================================================
Total params: 1,704,097
Trainable params: 1,704,097
```

Important Code snippets



```
[ ]  model = tf.keras.models.Sequential([

         # This is the first convolution
         tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(300, 300, 3)),
         tf.keras.layers.MaxPooling2D(2, 2),


         # The second convolution
         tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
         tf.keras.layers.MaxPooling2D(2,2),


         # The third convolution
         tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
         tf.keras.layers.MaxPooling2D(2,2),


         # The fourth convolution
         tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
         tf.keras.layers.MaxPooling2D(2,2),


         # The fifth convolution
         tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
         tf.keras.layers.MaxPooling2D(2,2),


         # Flatten the results to feed into a DNN
         tf.keras.layers.Flatten(),
         # 512 neuron hidden layer

         tf.keras.layers.Dense(512, activation='relu'),

         # Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('damaged') and 1 for the other ('whole')
         tf.keras.layers.Dense(1, activation='sigmoid')
     ])

     # Show a summary of the model. Check the number of trainable parameters
     model.summary()
```

1) Import necessary libraries
2) Define Convolution Neural Network

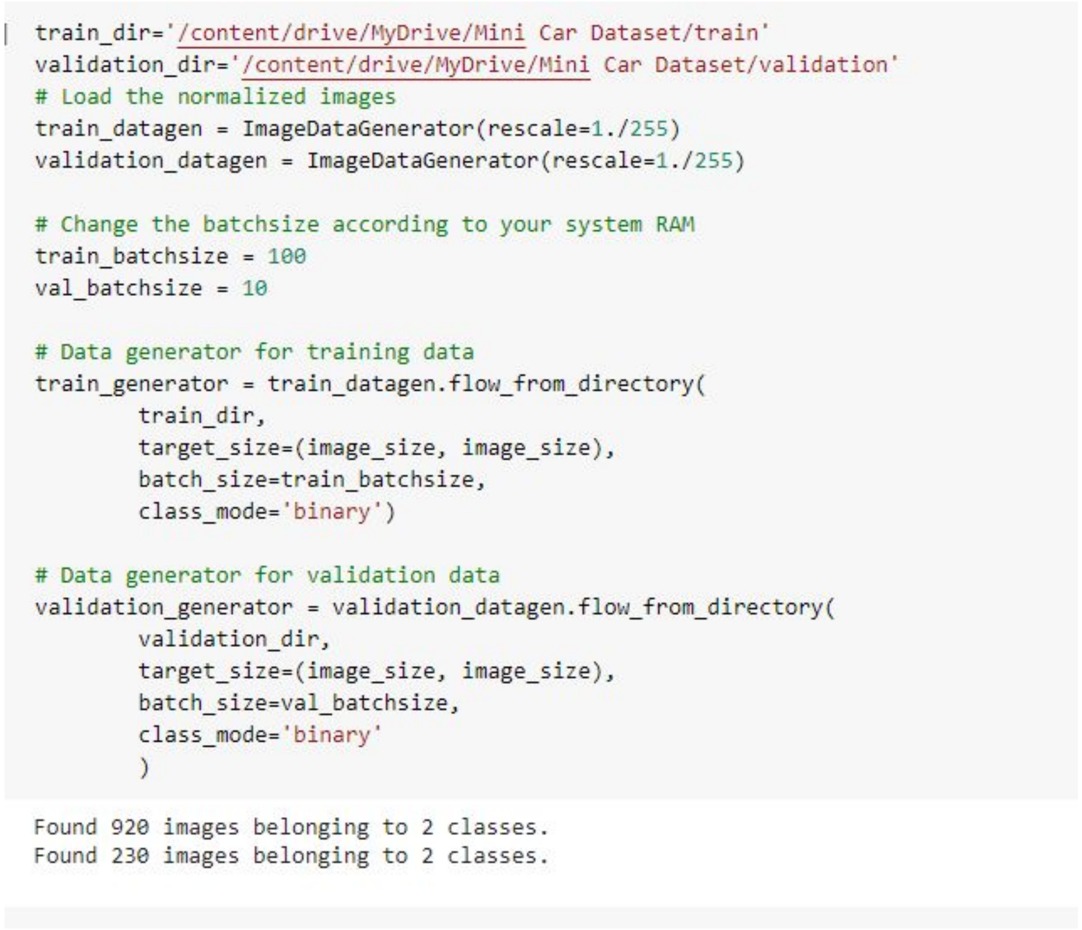
In the first set of experiments, we trained a CNN starting with the random initialization. Our CNN architecture consist of 5 layers: **Conv1-Pool1-Conv2-Pool2-Conv3-Pool3-Conv4-Pool4-Conv5-Pool5-FC-Softmax**

A RELU non-linearity is used for every convolutional layer.

The final layer has 1 neuron with sigmoid activation function.

The total number of parameters in the network are approx. 170K.

Loading Data



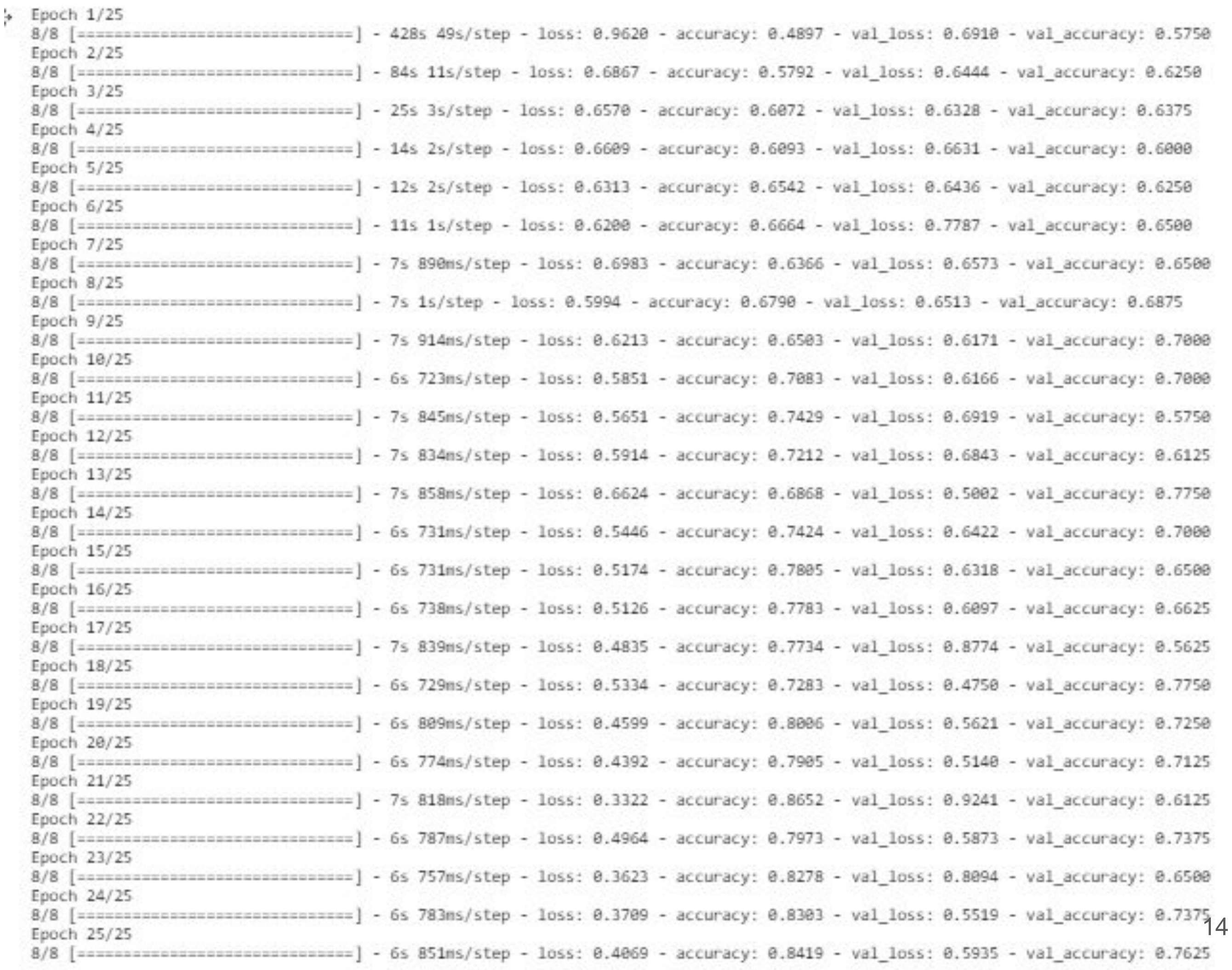
```
train_dir='/content/drive/MyDrive/Mini Car Dataset/train'
validation_dir='/content/drive/MyDrive/Mini Car Dataset/validation'
# Load the normalized images
train_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)

# Change the batchsize according to your system RAM
train_batchsize = 100
val_batchsize = 10

# Data generator for training data
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(image_size, image_size),
        batch_size=train_batchsize,
        class_mode='binary')

# Data generator for validation data
validation_generator = validation_datagen.flow_from_directory(
        validation_dir,
        target_size=(image_size, image_size),
        batch_size=val_batchsize,
        class_mode='binary'
        )
```

```
Found 920 images belonging to 2 classes.
Found 230 images belonging to 2 classes.
```

Using the ImageDataGenerator() function, we load the train_data and validation_data in to the code for training and validation.

Training

After loading the dataset , we train the model with discussed architecture for 25 epochs, considering the size of data available.



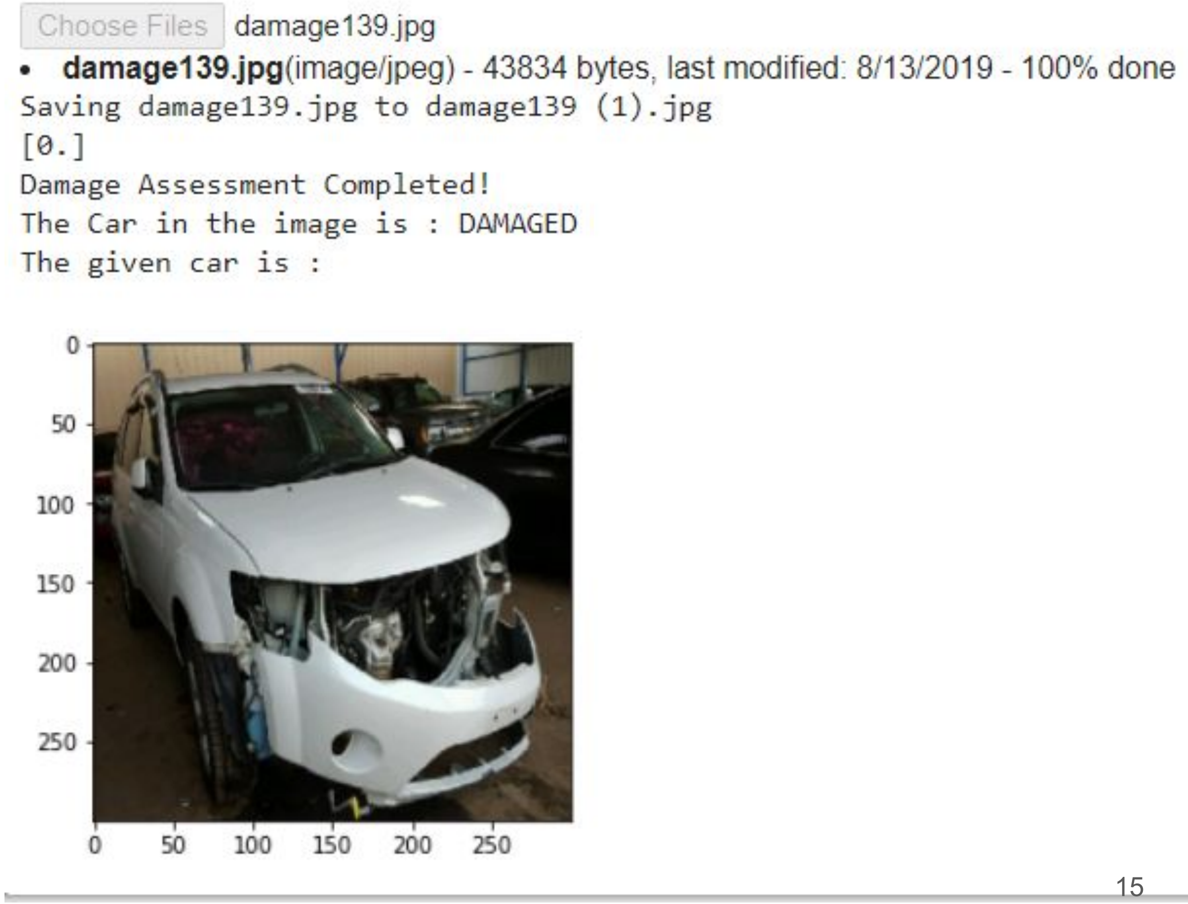
```
Epoch 1/25
8/8 [==============================] - 428s 49s/step - loss: 0.9620 - accuracy: 0.4897 - val_loss: 0.6910 - val_accuracy: 0.5750
Epoch 2/25
8/8 [==============================] - 84s 11s/step - loss: 0.6867 - accuracy: 0.5792 - val_loss: 0.6444 - val_accuracy: 0.6250
Epoch 3/25
8/8 [==============================] - 25s 3s/step - loss: 0.6570 - accuracy: 0.6072 - val_loss: 0.6328 - val_accuracy: 0.6375
Epoch 4/25
8/8 [==============================] - 14s 2s/step - loss: 0.6609 - accuracy: 0.6093 - val_loss: 0.6631 - val_accuracy: 0.6000
Epoch 5/25
8/8 [==============================] - 12s 2s/step - loss: 0.6313 - accuracy: 0.6542 - val_loss: 0.6436 - val_accuracy: 0.6250
Epoch 6/25
8/8 [==============================] - 11s 1s/step - loss: 0.6200 - accuracy: 0.6664 - val_loss: 0.7787 - val_accuracy: 0.6500
Epoch 7/25
8/8 [==============================] - 7s 890ms/step - loss: 0.6983 - accuracy: 0.6366 - val_loss: 0.6573 - val_accuracy: 0.6500
Epoch 8/25
8/8 [==============================] - 7s 1s/step - loss: 0.5994 - accuracy: 0.6790 - val_loss: 0.6513 - val_accuracy: 0.6875
Epoch 9/25
8/8 [==============================] - 7s 914ms/step - loss: 0.6213 - accuracy: 0.6503 - val_loss: 0.6171 - val_accuracy: 0.7000
Epoch 10/25
8/8 [==============================] - 6s 723ms/step - loss: 0.5851 - accuracy: 0.7083 - val_loss: 0.6166 - val_accuracy: 0.7000
Epoch 11/25
8/8 [==============================] - 7s 845ms/step - loss: 0.5651 - accuracy: 0.7429 - val_loss: 0.6919 - val_accuracy: 0.5750
Epoch 12/25
8/8 [==============================] - 7s 834ms/step - loss: 0.5914 - accuracy: 0.7212 - val_loss: 0.6843 - val_accuracy: 0.6125
Epoch 13/25
8/8 [==============================] - 7s 858ms/step - loss: 0.6624 - accuracy: 0.6868 - val_loss: 0.5002 - val_accuracy: 0.7750
Epoch 14/25
8/8 [==============================] - 6s 731ms/step - loss: 0.5446 - accuracy: 0.7424 - val_loss: 0.6422 - val_accuracy: 0.7000
Epoch 15/25
8/8 [==============================] - 6s 731ms/step - loss: 0.5174 - accuracy: 0.7805 - val_loss: 0.6318 - val_accuracy: 0.6500
Epoch 16/25
8/8 [==============================] - 6s 738ms/step - loss: 0.5126 - accuracy: 0.7783 - val_loss: 0.6097 - val_accuracy: 0.6625
Epoch 17/25
8/8 [==============================] - 7s 839ms/step - loss: 0.4835 - accuracy: 0.7734 - val_loss: 0.8774 - val_accuracy: 0.5625
Epoch 18/25
8/8 [==============================] - 6s 729ms/step - loss: 0.5334 - accuracy: 0.7283 - val_loss: 0.4750 - val_accuracy: 0.7750
Epoch 19/25
8/8 [==============================] - 6s 809ms/step - loss: 0.4599 - accuracy: 0.8006 - val_loss: 0.5621 - val_accuracy: 0.7250
Epoch 20/25
8/8 [==============================] - 6s 774ms/step - loss: 0.4392 - accuracy: 0.7905 - val_loss: 0.5140 - val_accuracy: 0.7125
Epoch 21/25
8/8 [==============================] - 7s 818ms/step - loss: 0.3322 - accuracy: 0.8652 - val_loss: 0.9241 - val_accuracy: 0.6125
Epoch 22/25
8/8 [==============================] - 6s 787ms/step - loss: 0.4964 - accuracy: 0.7973 - val_loss: 0.5873 - val_accuracy: 0.7375
Epoch 23/25
8/8 [==============================] - 6s 757ms/step - loss: 0.3623 - accuracy: 0.8278 - val_loss: 0.8094 - val_accuracy: 0.6500
Epoch 24/25
8/8 [==============================] - 6s 783ms/step - loss: 0.3709 - accuracy: 0.8303 - val_loss: 0.5519 - val_accuracy: 0.7375
Epoch 25/25
8/8 [==============================] - 6s 851ms/step - loss: 0.4069 - accuracy: 0.8419 - val_loss: 0.5935 - val_accuracy: 0.7625
```

Testing

DETECT DAMAGE

<u>Case I</u>: **Damaged**

In the first model, prediction of damage was successful with the help of CNN, running for 25 epochs



damage139.jpg

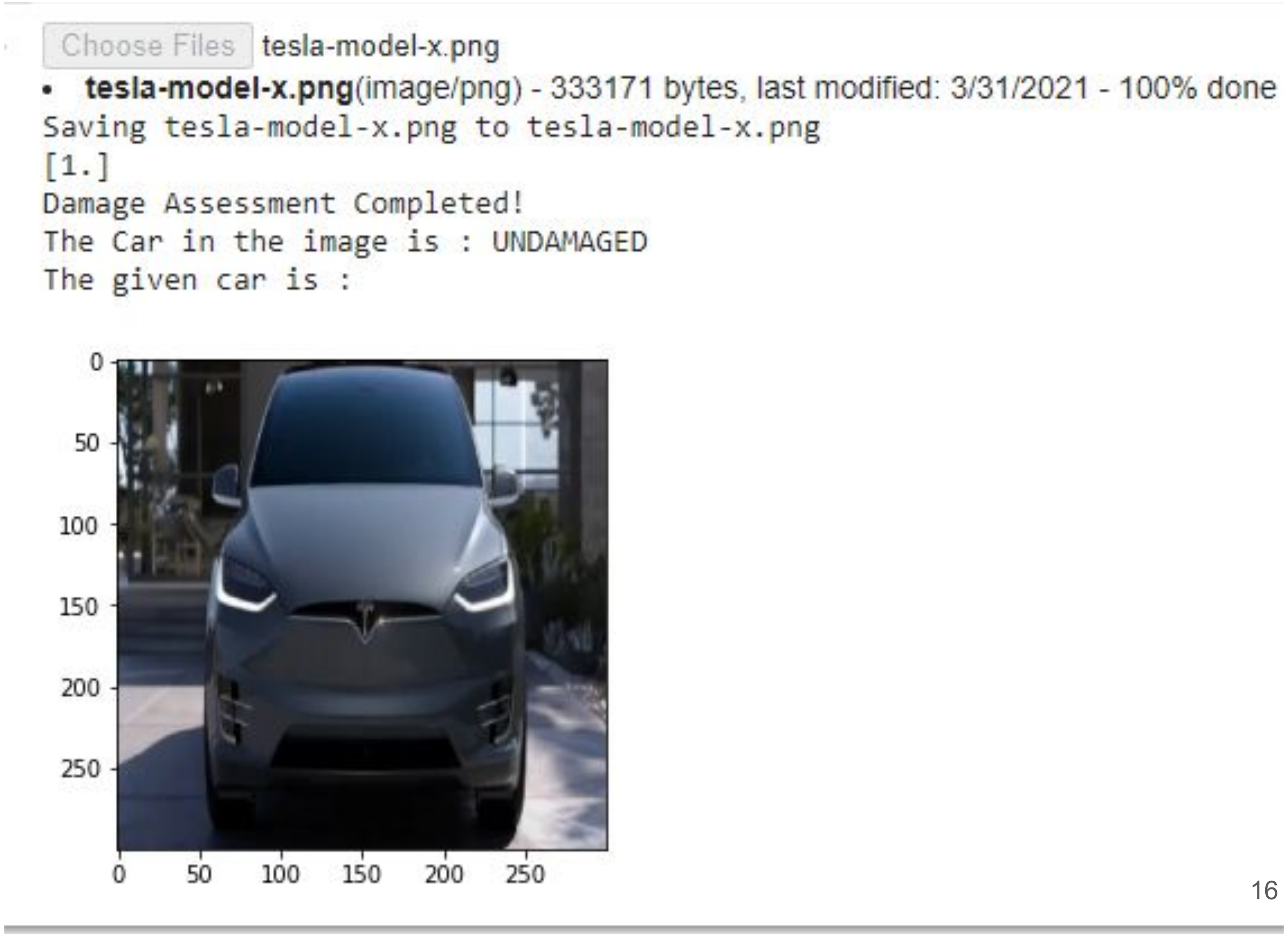
- **damage139.jpg**(image/jpeg) - 43834 bytes, last modified: 8/13/2019 - 100% done

```
Saving damage139.jpg to damage139 (1).jpg
[0.]
Damage Assessment Completed!
The Car in the image is : DAMAGED
The given car is :
```

Testing

DETECT DAMAGE

Case II: **Undamaged**



Choose Files tesla-model-x.png

- **tesla-model-x.png**(image/png) - 333171 bytes, last modified: 3/31/2021 - 100% done

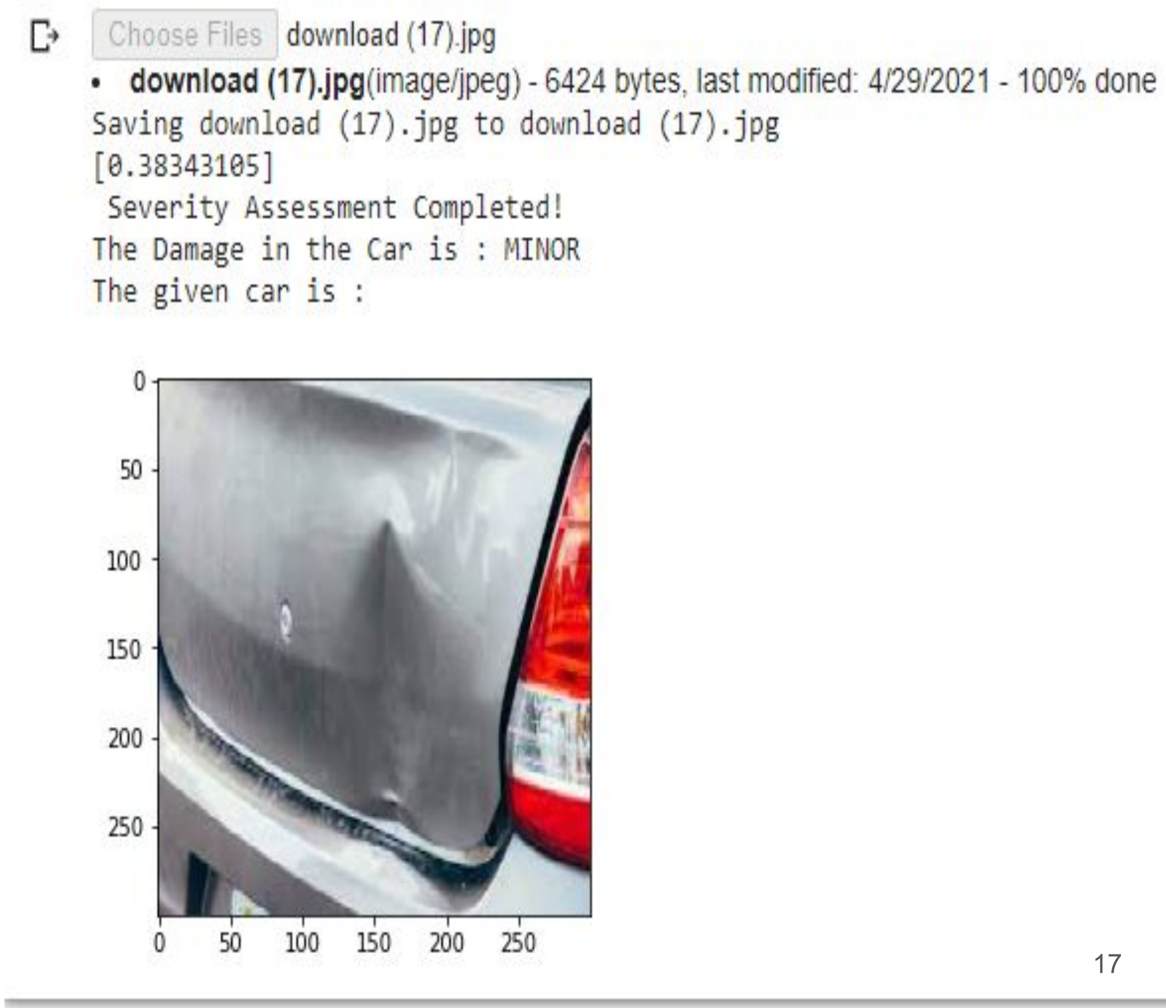
```
Saving tesla-model-x.png to tesla-model-x.png
[1.]
Damage Assessment Completed!
The Car in the image is : UNDAMAGED
The given car is :
```

Severity

DETECT SEVERITY

<u>Case I</u>: **Minor**

In this model, prediction of damage was successful with the help of CNN, running for 15 epochs.



```
download (17).jpg
• download (17).jpg(image/jpeg) - 6424 bytes, last modified: 4/29/2021 - 100% done
Saving download (17).jpg to download (17).jpg
[0.38343105]
 Severity Assessment Completed!
The Damage in the Car is : MINOR
The given car is :
```

Severity

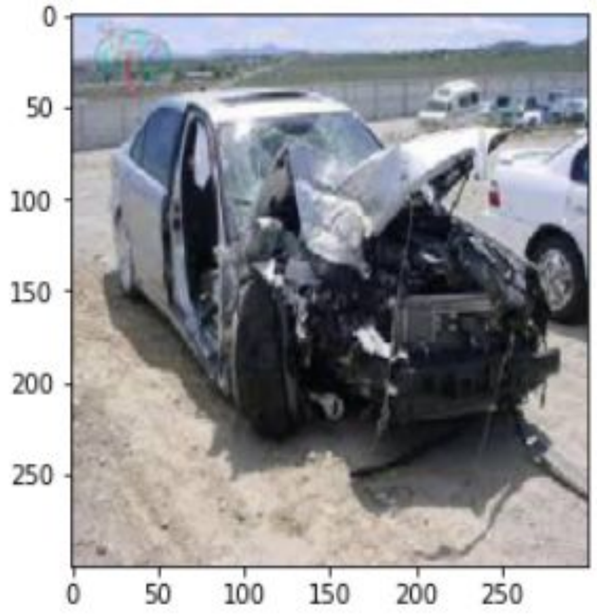
DETECT SEVERITY

Case II: **Severe**

Choose Files damage8.jpg

- **damage8.jpg**(image/jpeg) - 11260 bytes, last modified: 8/13/2019 - 100% done

```
Saving damage8.jpg to damage8.jpg
[1.]
 Severity Assessment Completed!
The Damage in the Car is : SEVERE
The given car is :
```



Results

The following results were obtained after training the model with the required modifications and additions to the CNN architecture:

- ❖ On the first task, an accuracy of **85%** was achieved **for 25 epochs**. On the second task, an accuracy of **85%** was achieved **for 15 epochs**. With this mark, this model can be placed in the same range of accuracy as expected from untrained human beings.
- ❖ As the epochs increases, it increases the computation resources and time, thereby giving better accuracy.

Constraints

1. **Backpropagation**

   Backprob is a method to find the contribution of every weight in the error after a batch of data is prepossessed and most of good optimization algorithms (SGD, ADAM …) uses Backpropagation to find the gradients backpropagation has been doing so good but is not an efficient way of learning, because it needs **huge dataset**

2. **Translation invariance**

   When we say translational invariance we mean that the same object with slightly change of orientation or position might not fire up the neuron that is supposed to recognize that object. Even though **CNN's are generally robust**, there is a **significant chance to have translational invariance.**

3. **Pooling layers**

   Pooling layers **loses a lot of valuable information** and it ignores the relation between the part and the whole. Here, we need to combine few features to say the damage is present, where CNN would say if those features are present with high probability, then it would be damaged.

Conclusion

1. This is how we proposed and worked on the Car Damage Detection and it’s severity to estimate the cost of repair. We used a data set along with some manually annotated pictures to train the CNNs by performing a pre-processing.

2. We detected the presence of damage and also its severity. Based on the severity, we suggest the estimated cost. For example, if it’s a mild damage like a scratch or a dent basic reasonable cost is advisable.

Improvements

The work presented here merely lays ground for further, more detailed work in this field. Further studies, relying on more computational power and bigger datasets might be desirable. This would make it possible to use even more powerful models to surpass human performance in related tasks.

This work also lays ground for the attempt of more complex tasks related to damage detection. These tasks might include repair cost estimation, determination of the driver at fault in car Work accidents, automated damage conditions estimation, among other tasks performed around car accidents. More new architectures like those of Transfer Learning and Ensemble methods could be deployed for best accuracy and performance.

References

1. Dataset: https://www.kaggle.com/anujms/car-damage-detection
2. https://www.ee.iitb.ac.in/student/~kalpesh.patil/material/car_damage.pdf
3. https://repositorio-aberto.up.pt/bitstream/10216/107814/2/219929.pdf
4. https://www.upgrad.com/blog/basic-cnn-architecture/
5. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53