Car Dent Detection using Deep learning Techniques

Sarathkumar Eswaran (G1801470F)
Wee Kim Wee School of Communication and Information
Nanyang Technological University

Abstract--In this project, we intend to apply the state-of-the-art Deep convolutional neural networks to identify the car has any dent in the image. Using transfer learning with available models pretrained on the ImageNet. A dataset of 1732 images belonging to two classes is used for training, validating and testing the model, in our approach, transfer learning with VGG-19 architecture is used to do the binary classification and achieved the accuracy of 92% with the validation set and using the object detection techniques such as Mask R CNN, model is developed with pretrained weights of COCO to locate the dent in the image which is classified as the image with dent. For Mask R CNN model, around 650 images with dent is used to train and validate the model. In future work, this scenario can be dealt as multi class classification problem such as where the dent is in front, rear and sideways, model can be developed to do the cost estimation along with the masking of the dent.

Keywords: Image classification, Convolutional neural networks, transfer learning, ImageNet, Deep learning

I. INTRODUCTION

With Unleashing power of computer vision, Deep learning techniques finding ways in different domains for different applications from detecting an object in a video to detecting the headcount of people in a crowded place. Deep learning techniques are not majorly popular among the automobile servicing sectors, one such application is using the Convolution neural networks to identify the dents in the car. Usually, people drive into the car servicing workshop and must wait until the supervisor of the Workshop acknowledges even for the general service. To overcome this, Computer vision techniques can be used to identify the Car dents and to raise the request for Supervisor for human inspection. This would help us in minimizing the efforts needed to inspect the car, for example, if the car doesn't have any dents it can be moved to general service directly without any further delay.

A. Problem Statement

The objective of this project is to use Deep learning Techniques such as Convolutional neural network (CNN) to automate the process of identifying car dents through car images. Car dents usually appear on the bumpers due to collision with other objects or automobiles, usually, it occurs in a parking lot. This may not affect the functionality of the car, hence people tend to ignore that and will try to get it fixed during the regular service of their cars. Due to this people are forced to wait even who are there for general servicing. To simplify the process of driving through vehicles in Workshops with least waiting time, an image classification model will be developed to classify the image and to locate the dent. This model will also help in other domains such as Insurance companies to validate the claims and in Bank car loan process, to evaluate the worth of the used cars.

B. The Problem-Working with Limited set of data

CNN's are known to achieve higher accuracy given that it has enough data to train the model, but it is a time-consuming process to collect such a huge volume of data. In this project, instead of collecting images from real time cars, images are scrapped from the shutterstock.com and using chrome extension called Fatkun batch image downloader is used to download the images from google images for each class i.e., 850 images for cars without any dents are obtained and 880 images for cars with dent, with these images CNN will be trained through a transfer learning and fine tuning the pre-trained models to see how much accuracy can be achieved. In the scrapped dataset, we have images of a car from the front, rear, and side view along with few images of a car which got crashed. Manually, the images are filtered which has the dent on the cars.

II. BACKGROUND

Among the deep learning techniques Convolution Neural Networks (CNN) is gaining popularity in computer vision tasks such as object detection and recognition. It is known that Convolutional neural networks (CNN's) require huge amounts of data and resources to train the model. To overcome this drawback, transfer learning has become popular among the researchers, as it drastically reduces the training time and data needed to build the model. A Neural network which is trained on the source task helps in extracting the useful features for target task. Pre-trained weights of the CNN models trained on ImageNet dataset, are available publicly such as VGG-16/19, Alexnet, Inception, and Resnet. Transfer learning helps to tune the model to fix the overfitting effect in case of a small dataset.

The study carried to use the Deep learning techniques to classify the damages in car images and locating it [1]. In this paper[1], an image is classified based on the type of damage such as a bumper dent, glass shatter, door dent, headlamp broken, tail lamp broken, smash and scratch using different techniques such as Convolution based auto-encoder, CNN based model and transfer learning with Resnet model. Based on this study results, transfer learning with Resnet model found to be performing extremely well compared to other models.

Research performed on detecting the damages with car in locating the damages, type and size using the Deep convolution networks using the VGG 16 architecture with transfer learning [2]. In this paper, they managed to achieve the accuracy of 75%, 68% and 54% respectively. In our project, problem statement is focused on a specific class i.e.., identifying the dents in the car and locating it, the approach followed is pretty much similar to the paper [1], the differences being transfer learning models are considered because of the dataset constraints and to make it more interesting using object detection techniques dent present in the image will be located.

A. VGG Network

Simonyan and Zisserman [3]investigated the effect of Convolutional Neural Network (CNN) depth from 16 to 19 weight layers on its accuracy using the ImageNet largescale visual recognition challenge (ILSVRC-2012) dataset. This dataset includes 1000 classes with 1.2 million training images, 500,000 validation images, and 100,000 test images. This network is popular for its simplicity, it uses 3*3 convolutional layers stacked on each other in increasing depths. Max-pooling layers are in place to reduce the volume size, two fully connected layers of 4096 nodes are followed by the SoftMax classifier [4]. As a result of the evaluation, it is found that the depth of the CNN improves the accuracy of large-scale image classification. This study confirms that this model generalizes well on the large scale of datasets and can be used for classifying the images from new domains. This ImageNet model weight have been made publicly available in the form VGG-16 and VGG-19 where 16 and 19 represent the depth of the model [3].

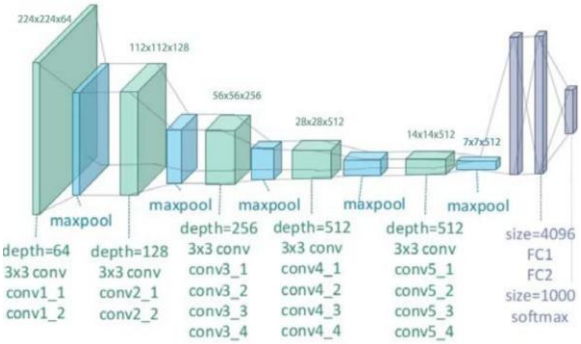

Figure 1: VGG 19 Architecture, Source: Adapted from [5]

B. ResNet

Increasing the depth of the convolution layers will always result in improved accuracy. Moreover, the loss in deeper networks is comparatively higher than the results of superficial neural networks [6]. To overcome that He, k., et al.[7] proposed a residual learning framework to ease the training of deeper networks using micro architecture modules nothing but a shortcut connection. In shortcut connections several layers of CNN are skipped [6]. Output of this shortcut connections is fused with the output of the stacked layers. The advantages of using the output of shortcut connections is having only the trivial numbers for computation. Another advantage of using He, k., et al. shortcut connections is deeper layers of CNN can be trained using the standard SGD [6]. Even though the ResNet is deeper than VGG 16 and VGG 19, the size of the model is significantly smaller than the VGG models, due to the presence of global average pooling

layer instead of the fully connected layers. This reduces the model size substantially to the size of 102 mb for ResNet 50 [4].

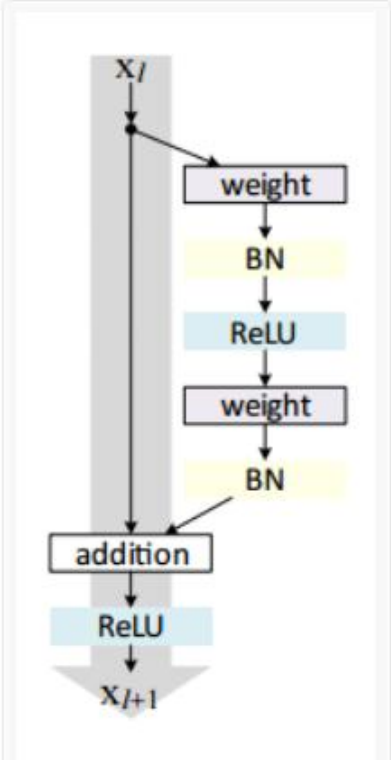

Figure 2: Residual module in Resnet, Source: adapted from [4]

C. Mask R CNN

Mask R CNN is the extended version of Faster R CNN only difference is that in Mask R CNN segmentation masks are predicted by adding a branch, on each Region of interest, we will see how the Faster R CNN works. Faster R CNN comprises of two stages. The first stage is called Region proposal network (RPN), scans the image and proposes a candidate object bounding boxes. In the second stage, features are extracted using Region of Interest (ROI) pooling layer from each proposal and fully connected network (FCN) is used to classify and output the bounding boxes for each of objects [8]. Mask R CNN works like Faster R CNN, has the same two stages, the difference, is the binary mask is generated for each ROI, in parallel to predicting the class and box offset. In stage two, RoI align is used to locate the relevant areas of feature map and an FCN branch for generating the mask for each object in pixel level [8].

Each of this stage is connected to backbone called Feature Pyramid Network (FPN) comprises of the bottom-up pathway, top-bottom pathway, and lateral

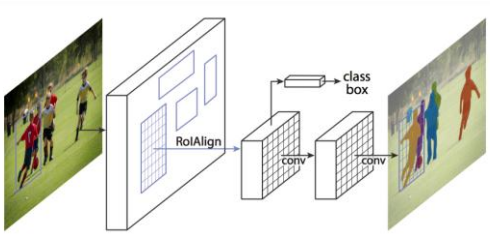

Figure 3: Mask R CNN framework for Instance Segmentation, Source: Adapted from [8]

connections. Bottom-up pathways in CNN such as Resnet, VGG used to extract features from the input images, FPN generates features pyramid map at various resolutions levels size of the feature map is like bottom-up pathways [9]. Lateral connections are the convolution and adding operations used between these 2 pathways. Using this combination Mask R CNN can achieve a good gain in terms of accuracy and speed in the feature extraction technique [9].

III DATA COLLECTION AND PRE-PROCESSING

Since the dataset is not readily available online, the dataset has been prepared manually. Car dent images were downloaded from the shutterstock.com using the web scrapping techniques such as Beautiful soup, code is mentioned in the appendix section A for the same. 880 car images with dent and 850 car images without dent were downloaded. Few images were downloaded using the chrome extension called Fatkun batch image downloader from google images. Few samples of the image obtained as shown in fig. 4a for the dent class and fig. 4b for the no dent class. Now the collected images are split into three sets training set, validation set, and test set. Training set has 650 images for dent class and 700 images for No dent class, Validation set has 176 images for Dent class and 150 images for No dent class and test set has 26 images for each class.

This data distribution is used for the binary classification of image using VGG 19 architecture. All images were in different size and brought to the same scale of 224 * 224 to feed in to the VGG 19 model, Image Data generator class was used as an pipeline to feed in the images and the image count is not enough to train the model, hence image augmenting

techniques were used along with pipeline to create multiple snaps using the image data generator options for both training and validation set with below parameters, the code snippet is shown in appendix section B for the same and few samples of augmented images were shown in fig 4c. Both images in training and test sets were rescaled, flipped horizontally, zoomed by the range of 0.5, width and height shifted by the range of 0.2, sheared by the range of 0.2 and rotated by the range of 20.

Rescale – Original Image pixels is in range of 0-255 and this value would be very high for the model to handle hence it is brought down to the range of 0-1 by multiplying the pixel values with 1/255.

Zoom range - Randomly zooming inside pictures

Shear range - Randomly applying shearing transformations

Horizontal flip - Flipping the images in horizontal directions which have no assumptions of horizontal symmetry

Width and height shift range - Fraction of total width and height to randomly translate pictures either horizontally or vertically.

Figure 4a: Dent class Image sample

Figure 4b: No Dent class Image sample

For object detection, around 530 car dent images were used as a training set to train the Mask-R-CNN model and 120 car dent images were used as a validation set. Annotations must be created for the Mask R CNN image data set, VGG image annotator online tool has been used to create the annotation and the annotations for all the images are saved in JSON file format as polygon points and fed into the network Fig. 5 shows how the annotation has been created on

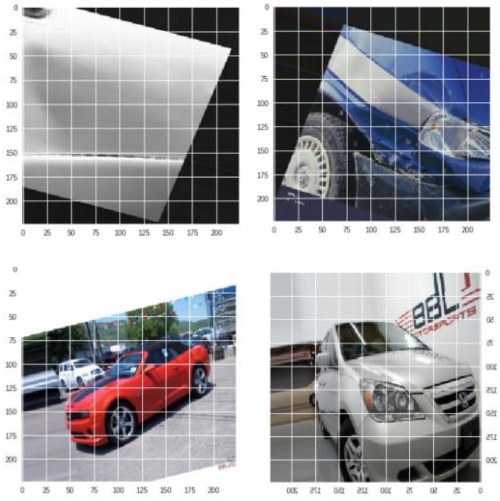

Figure 4c: Augmented image samples

Table 1 Image distribution

set	Class Nos	Total images
Training	2	1353
Validation	2	326
Test	2	53

dents. Separate annotation file has been created for the training and validation set. Dataset has images of different types such as PNG, JPEG to deal with such cases few functions in COCO.py file has been modified as shown in appendix section C refer the customDataset class and all images were resized to the same size of 1024 * 1024, to ensure that model can train multiple images per batch. If the available image is not a square it is padded with zero.

IV METHODS AND ALGORITHM

Instead of training a model from scratch, transfer learning technique has been used. Transfer learning is using the pretrained weights of VGG-19 and removes the last few layers such as Fully connected layers and SoftMax layers, is replaced with one fully connected layer along with dropout layer to overcome the overfitting issue, followed by the sigmoid classifier,

since it is a binary classifier, model summary is shown in fig.6. After doing the binary classification, object

Figure 5: Image Annotation using VGG image annotator

detection technique is used to identify the exact spot of the dent, we use Mask R CNN technique for our cause.

In transfer learning, the performance of the model depends on the following crucial step, as there are 3 options either to train the entire model, freeze the certain layers of the model or to freeze the entire convolution layers of the model. In this case, VGG 19 architecture trained using the ImageNet dataset and it has a class to identify the cars , but here in this scenario, the model is focused on detecting the dents in the images of car, hence we have frozen initial five layers of the model, which will extract the crucial features of the images such as edges and blobs and the rest of the layers are trained with our image data set to ensure that model learns about specific features related to the dataset i.e. dents in the images.

```
for layer in model.layers[:5]:
    layer.trainable = False
```

A. Optimization algorithm and loss function

Stochastic gradient descent is used as the optimizer with the learning rate of 0.0001 and momentum 0.9 is used for training the model. The Stochastic gradient is preferred as it tends to update the weights of the cost function by seeing the mini batch of training examples. The reason being it reduces variance in the weight update and leads to the stable convergence, and another reason being computation takes advantage of highly optimized matrix operations that should be used in a well vectorized computation of the cost and gradient [10].

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_conv4 (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_conv4 (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_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 512)	12845568
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 1)	513

Total params: 32,870,465
Trainable params: 32,757,889
Non-trainable params: 112,576

Figure 6: Model summary

In SGD, very small learning rate is used in initial epochs to get the stable convergence and it can be modified if it no further changes in error rate of the epochs. When the objective function has a long shallow ravine leading to the optimum and steep walls on the sides, standard SGD tends to oscillate across the narrow ravine since the negative gradient will point down one of the steep sides rather than along the ravine towards the optimum. Objective of the deep architectures have this form near local optima and standard SGD tends to be slower in convergence particularly after the initial steep gains. Momentum can be used with SGD to overcome this issue and it

enables the model to converge in the right direction, thus leading to faster converging. Moment update is given by the below equation

$$v = \gamma v + \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$
$$\theta = \theta - v$$

Source: Adapted from [10]

In above equation, v represents the current velocity which is of the same size as the parameter θ. Learning rate is represented as α, γ is the momentum, learning rate is set to a smaller value when used with momentum and momentum is set to 0.9, as it is known that lesser values of momentum produces more fluctuations and higher values of momentum produces the smoother curve but it averages on the large number of samples and optimal value to be 0.9 [11].

B. Cross Entropy

Cross entropy or log loss, used to measure the performance of the model which return the probability value between 0 and 1, below fig 7 is for interpretation. it does not replicate the values of the model. As we see that when predicted probability approaches 1 its log loss gradually decreases. Log loss penalizes both errors which is confident and wrong.

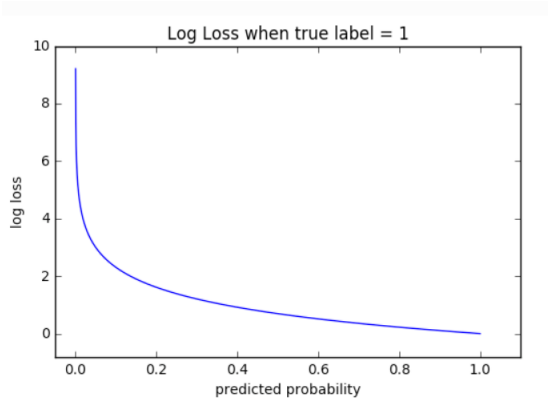

Figure 7: Log loss when true label=1, Source: Adapted from [12]

Cross entropy for binary classification is computed using the following formulae:
−(ylog(p)+(1−y)log(1−p))

Where p represents the predicted probability and y represents the actual probability

In this model, binary cross entropy is used as a loss function as the problem statement deals with binary classification along with sigmoid activation function.

Mask R CNN popular object detection techniques are used to detect the location of dent with Resnet 101 as backbone and pretrained weights of the model on Common objects in context (COCO) training set used to start the learning process, implementation part follows the same steps mentioned in Mask R CNN paper differences being the bounding boxes and masks are generated on the fly based on the mask instead loading the datasets with predefined boundary boxes, learning rate of 0.01 is used instead of 0.02 and remaining parameters such as weight decay and moment is the same value as used in the original paper i.e. 0.0001 and 0.9 respectively. Model is loaded with the images, which is resized to 1024*1024 along with annotation file in json format. It is found that the convergence rate is faster with a small learning rate in this case. Using Annotations, pixels in each image will be classified into background and dent as shown in Fig 8. Mask generated on the positive regions selected by the ROI classifier as shown in Fig 9.

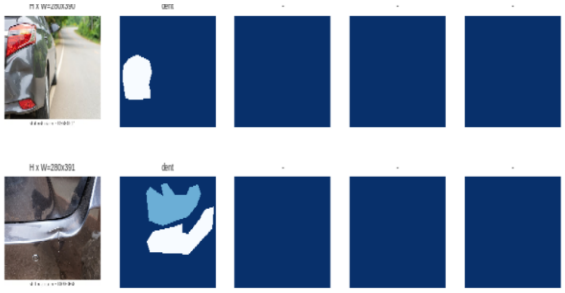

Figure 8: Annotated image

Figure 9: Masked image

V RESULTS

A. Binary Classifier

After training the model with image set of 1373 images belonging to both the classes for 250 epochs, performance of the model is evaluated is using the same validation set and a test set with images of 53 belonging to 2 classes. In earlier epochs, validation accuracy is high since inner variance with in the validation set is low. When the model is evaluated with the validation set of 326 images belonging to 2 classes, accuracy of the model went up to 92.33% and F1-score of the model stands around 92% which reflects that model is doing equally good with true negative class i.e. no dent class as shown in Fig. 10. Fig. 11 shows the confusion matrix of the model on the validation set, we can see that model is doing good in identifying the no dent class as it properly classified all the 150 images as no dent and in Dent class among 176 images, 151 images were classified properly and 25 images were misclassified as no dent. Fig. 14 shows the final VGG-19 model accuracy and loss on training and validation set on the last 100 epochs.

	precision	recall	f1-score	support
Dent	1.00	0.86	0.92	176
No_dent	0.86	1.00	0.92	150
micro avg	0.92	0.92	0.92	326
macro avg	0.93	0.93	0.92	326
weighted avg	0.93	0.92	0.92	326

Figure 10: Classification Report of model on validation set

True	Dent	No Dent
Predicted		
Dent	151	0
No Dent	25	150

Figure 11: Confusion matrix of model on validation set

And to validate the model further we have used the 53 images which were not used before, i.e., test set, and model is comparatively doing good in classifying the images among the 53 images, 26 images belong to the dent class and 27 images belong to the no dent class, model was able to classify all the images accurately. As the images are pretty much similar, due to which model can predict all the images accurately.

B. Masking the Dent spot

After doing the binary classification using the VGG 19 model, the images which were classified as dent can be used to identify the exact spot of the dent using

	precision	recall	f1-score	support
Dent	1.00	1.00	1.00	27
No_dent	1.00	1.00	1.00	26
micro avg	1.00	1.00	1.00	53
macro avg	1.00	1.00	1.00	53
weighted avg	1.00	1.00	1.00	53

Figure 12: Classification report of model on test set

True	Dent	No Dent
Predicted		
Dent	27	0
No Dent	0	26

Figure 13: Confusion matrix of model on test set

Figure14a: VGG-19 Model Accuracy

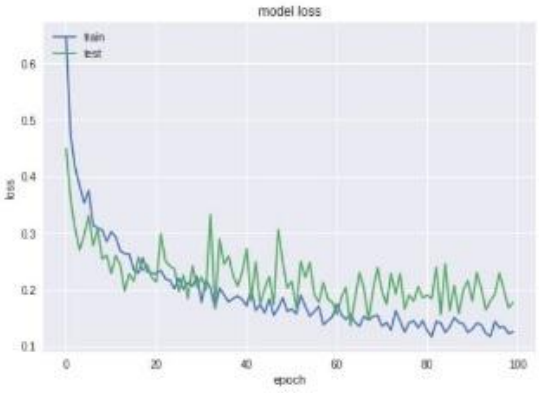

Figure 14b: VGG-19 model loss

Mask R CNN model, as mentioned earlier we used 530 images to train the model and 117 images as validation set. The model is evaluated on image from validation set and it can locate the dent exactly and moreover

model is validated by checking the distribution of weights and biases. Fig. 15. Shows the distribution of weights and biases of the model in the first few layers. Fig 16 shows the model prediction on the images from validation set.

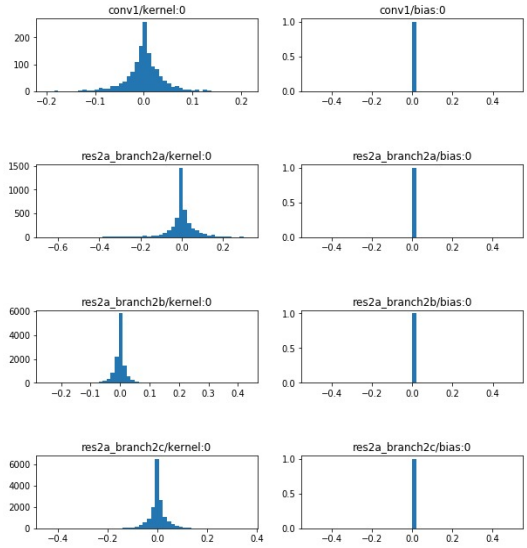

Figure 15: Weight distribution of the final model

Figure 16: Mask R CNN prediction on Images

VI DISCUSSION

In this section, we will discuss how the model is trained and optimized to get the good performance for binary classifier and Mask R CNN.

A. Binary classifier

Initially tried to build the model using Resnet architecture with pretrained weights of ImageNet, by freezing all the layers with one fully connected layer and dense layer with sigmoid activation function in the output layer, trained the model for 65 epochs with the batch size of 16. The model was found to be under fitted as shown in fig 17 training loss was getting reduced and reached a value of 0.09 and validation loss was around 0.8. In the figure, only the values of the last 15 epochs were used. To proceed further VGG 19 architecture has been used for the further development of the model since they perform better with new domain imageset and it was used in similar problems [2].

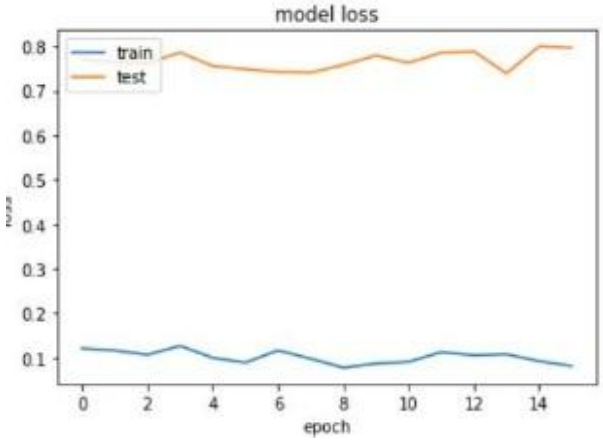

Figure 17a: Resnet model loss

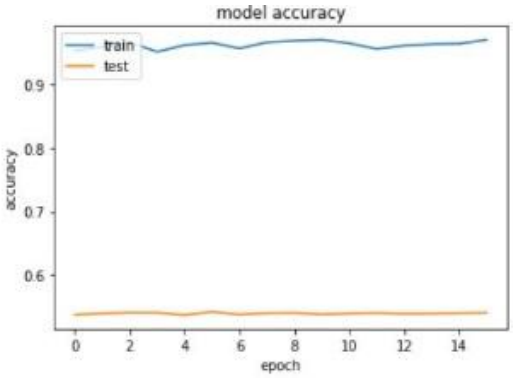

Figure 17b: ResNet Model accuracy

Using VGG 19 architecture with pre trained weights of ImageNet, all the layers were frozen in the beginning, and model is trained with this architecture for 50 epochs with a batch size of 16, the loss and accuracy difference between the training and validation is shown in fig 18. And this model didn't

perform well on the validation set, even though it managed to achieve the accuracy of 90%.

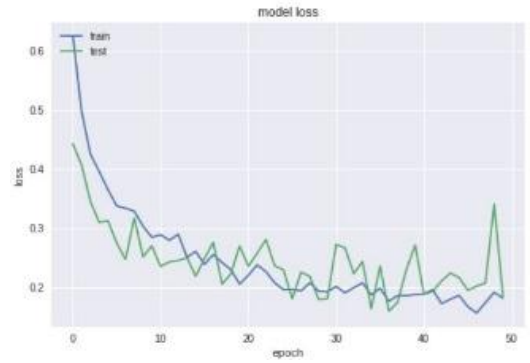

Figure 18a: Initial VGG-19 model loss

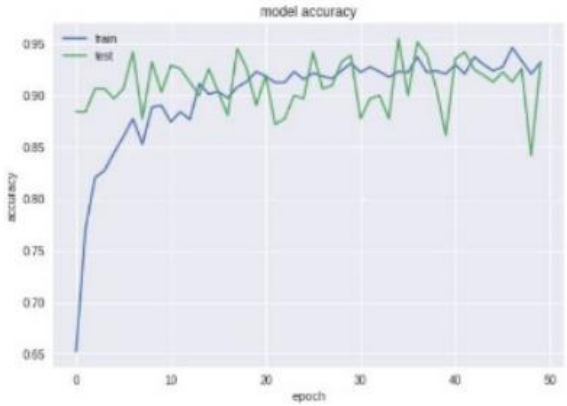

Figure 18b: Initial VGG-19 model accuracy

To overcome this, certain number of layers were frozen, as it is difficult to find the exact number of layers to freeze, by trying multiple epochs was able to find the layers to be frozen as 5, tried freezing 10 layers of architecture it is found to be underfitting as it didn't learn much about the dataset. Figure 19. Shows the learning curve for the loss of the model trained for 50 epochs with 10 layers frozen. When the model is tested on the validation set, the model performed extremely poor predicted all images as no dent class at that time training set has 700 images for No dent class and 520 images for dent class in training set.

Considering the model performance, 5 layers were kept intact, and 2 fully connected layers with 1024 neurons followed by dropout layers with 0.5 were used along with sigmoid activation function as output layer at the end of the architecture, from the sixth layers the weights of all layers is updated by using the training dataset. In this stage, SGD is used as optimizer and tried different learning rates with this architecture

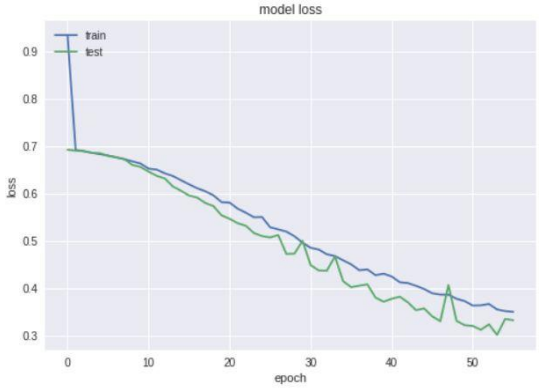

Figure 19: Loss of model with 8 layers frozen

to find the appropriate learning rate as shown in fig 22. Model with learning rate of 1e-4 is found to be doing good as it converges gradually comparing to other learning curves. Now, the model with learning rate of 1e-4 is trained for 50 more epochs and, the model was performing reasonably good but found to be overfitting, learning curve for loss of the model is shown in fig 20.

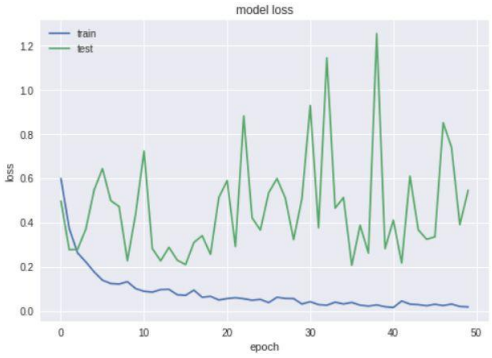

Figure 20: SGD loss with frozen layers

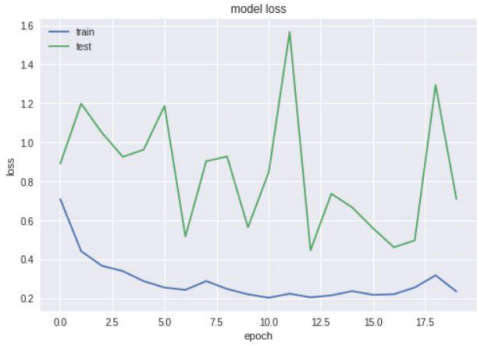

Figure 21: RMSprop loss with frozen layers

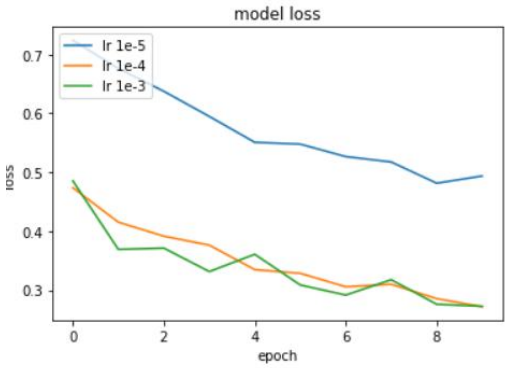

Figure 22: Different learning rates with SGD

To fix this issue, optimizer was changed to rmsprop as it uses the moving average of squared gradient with learning rate of 1e-4 and still, the model performance was found to be overfitting as shown in fig 21 and to improve the performance learning rate is adjusted to 1e-3. Still, it didn't improve the model performance much. Then, tried replacing the dropout layer with the L2 regularization to the layers which made the model perform very poor as it was taking longer time to converge to the local minima. Hence, optimizer is changed back to the SGD and to fix the overfitting issue, 100 more images were added to the image set in the training sample to the dent class and it was having around 550 images previously, 40 more images added to the validation set to bring the count to 176 from 130 in dent class, as there were too many fluctuations in the learning curve, enabled the early stopping for the model. And when the model is trained with this image set, the model is still overfitting with this sample shown in fig 23.

Finally, the model architecture was modified by removing the last one fully connected layer and reduced the number of neurons in a fully connected layer to 512. When the model is trained with this new architecture for 170 epochs, training error got reduced to 0.02 and validation error reduced to 0.08 and it produced reasonably good results comparing to the previous instances learning curve for the model is shown in Fig 24. When the model is tested on the validation set and test set model, it achieved accuracy of 92% in the validation set and able to classify all the images in test set appropriately. This is due to the fact that the test set has reasonably simple images to classify, that's why model was able to achieve the good accuracy with test set.

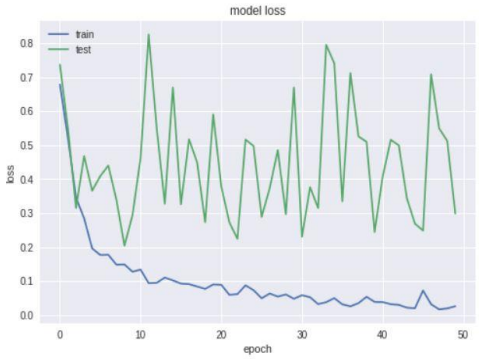

Figure 23: SGD loss with few more images

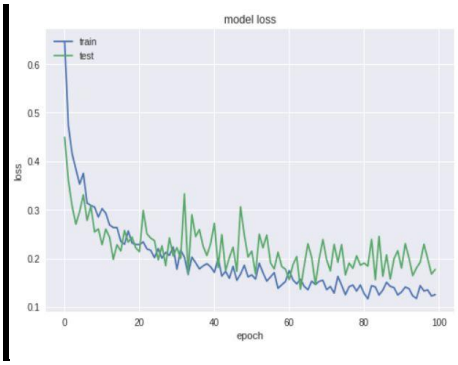

Figure 24: SGD loss with modified architecture

B. Object detection using Mask R CNN

As mentioned earlier after the binary classification object detection technique is used to identify the exact spot. Mask R CNN technique with Resnet 101 as backbone is used as object detection technique. Using transfer learning, the pre-trained weights of COCO has been used instead of training the model from scratch, since the COCO dataset has a car class which helps in the feature extractors. Binary masks which are generated in the run can get large with high resolution images, it may take more memory. To improve the training speed, binary masks generated are optimized, by storing the mask pixels that are inside the boundary box instead of the mask in the whole image and resized the mask size to a smaller size. Mini-mask generated for the image is shown in the fig. 25. The resized masks are augmented as shown in Fig 26. And model is trained with the learning rate of 0.01, as we are dealing with small batches of images to ensure it doesn't overshoot the local minima. In object detection techniques, annotating the class plays a significant role in the performance of the model.

Figure 25: Mini Mask generated on the image

Figure 26: Augmented Mask

Initially, annotations were poorly made using different shapes instead of a polygon and trained the model for only 50 epochs with learning rate of 0.01, due to which the model couldn't learn much about the data set and the masking was shown in unintended places and when dataset is visualized with annotations, this issue is found, to fix this issue annotation has been done again using polygon consistently for all images and trained the model for around 250 epochs. This reflected in model performance along with the score and it reduced the Mrcnn_mask_loss on training data to 0.15 and on validation data to 0.63. And the model is validated by checking the weight distribution of the model as shown in fig 15.

VII FUTURE WORK

Binary classifier achieved the accuracy of 92.33% but the dataset used is very small, this may not be the true representative of real-world scenario. More data should be collected pertaining to this issue and the model must be trained with these images to improve the performance of the model further. In this method, binary classification has been done, in future work, it can be resolved as the multi class problem like dent appearing in front, rear, and sideways. Along with object detection techniques, an estimate to replace the dent part can also be predicted and can be displayed along with the part name. And all this feature combining, an application can be developed which can also be used to get the cost estimation of the defective parts. This model has a wide range of applications starting from manufacturing plant to the servicing industry.

VIII CONCLUSION

In ImageNet, model is trained to identify the cars and its type, identifying the dent in the image is a tedious task since the models are not trained on such classification techniques. With the limited dataset, using VGG-19 with transfer learning, model able to achieve the accuracy of 92.3% since it is a smaller data set, model is doing good with validation and test set. This model can be developed further with rich and quality dataset to make the model more robust. And using the object detection techniques such as Mask R CNN were able to spot the dent in images. This model can be deployed after training with rich data set and it has wide range of applications from manufacturing company to insurance company. As discussed earlier in future work, model can be improved further in identifying the location of the dent, its severity and the cost estimation of the replacement part.

IX REFERENCES

[1] K. Patil, et al., "Deep Learning Based Car Damage Classification," ed: IEEE, 2017, p. 50.

[2] J. d. Deijn, "Automatic Car Damage Recognition using Convolutional Neural Networks."

[3] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," ed, 2014.

[4] A. Rosebrock, "ImageNet: VGGNet, ResNet, Inception, and Xception with Keras," ed, 2017.

[5] Y. Zheng, C. Yang, and A. Merkulov, *Breast cancer screening using convolutional neural network and follow-up digital mammography*. 2018, p. 4.

[6] H. Qassim, D. Feinzimer, and A. Verma, "Residual Squeeze VGG16," ed, 2017.

[7] K. He, et al., "Deep Residual Learning for Image Recognition," ed: IEEE, 2016, p. 770.
[8] k. He, et al., "Mask R-CNN," 2017 2017.
[9] X. Zhang, "Simple Understanding of Mask RCNN," ed, 2018.
[10] Stanford.edu, "Optimization: Stochastic Gradient Descent," ed.
[11] V. Bushaev, "Stochastic Gradient Descent with momentum," ed, 2017.
[12] ML-Cheatsheet, "Loss Functions."
[13] W. Abdulla, "Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow," ed, 2017.

APPENDIX

Section A- Web scrapping

In [ ]:

```
import bs4 as bs
import urllib.request as request

#file path
folder=r'C:\Users\Sarath\Desktop\term-2\AI\car-damage-dataset\Shutter_stock' + '\\'
#source url
url='https://www.shutterstock.com/search?page=7&section=1&searchterm=car%20dent&measure
ment=px&sort=popular&image_type=all&safe=true&search_source=base_search_form&language=e
n&saveFiltersLink=true'
response = request.urlopen(url)
soup = bs.BeautifulSoup(response, 'html.parser')

img_class=soup.findAll('div',{'class':'img-wrap'} )

for enum,i in enumerate(img_class):
    k=str(enum+500)
    request.urlretrieve(i.img['src'],folder+k+'.jpeg')
```

Section B- Binary Classification of image

In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

In [0]:

```
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D
from keras import backend as k
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyS
topping
```

In [0]:

```
#Resizing the images to 224*224 and assigning the value for batch size
import os

img_width, img_height = 224, 224
train_data_dir = "/content/drive/My Drive/train_AI_new"
validation_data_dir = "/content/drive/My Drive/validation_AI_new"

train_samples = [len(os.listdir(train_data_dir+'/'+i)) for i in sorted(os.listdir(train
_data_dir))]
nb_train_samples = sum(train_samples)

validation_samples = [len(os.listdir(validation_data_dir+'/'+i)) for i in sorted(os.lis
tdir(validation_data_dir))]
nb_validation_samples = sum(validation_samples)

batch_size = 16
```

In [0]:

```
#Downloading the VGG 19 model architecture
model = applications.VGG19(weights = "imagenet", include_top=False, input_shape = (img_
width, img_height, 3))
```

In [0]:

```
#Freezing the first five layers of VGG-19 model
for layer in model.layers[:5]:
    layer.trainable = False
```

In [0]:

```
#defining fully connected layers and output layers
from keras.regularizers import l2
x = model.output
x = Flatten()(x)
x = Dense(512, activation="relu")(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation="sigmoid")(x)
```

In [0]:

```
#Defining the input and output to the model
model_final = Model(input = model.input, output = predictions)
model_final.load_weights("/content/drive/My Drive/imagenew_sgd1.h5")
```

In [0]:

```
#summarising the model
model_final.summary()
```

In [0]:

```
#compiling the model with loss function and optimizer
from keras.utils.vis_utils import plot_model
model_final.compile(loss = "binary_crossentropy", optimizer = optimizers.SGD(lr=0.0001,
momentum=0.9), metrics=["accuracy"])
plot_model(model_final, to_file='model_plot.png', show_shapes=True, show_layer_names=Tr
ue)
```

Data Preprocessing

In [0]:

```
train_datagen = ImageDataGenerator(
rescale = 1./255,
shear_range=0.2,
horizontal_flip = True,
fill_mode = "nearest",
zoom_range = 0.2,
width_shift_range = 0.2,
height_shift_range=0.2,
rotation_range=20)
```

In [0]:

```
test_datagen = ImageDataGenerator(
rescale = 1./255,
shear_range=0.2,
horizontal_flip = True,
fill_mode = "nearest",
zoom_range = 0.2,
width_shift_range = 0.2,
height_shift_range=0.2,
rotation_range=20)
```

In [0]:

```
path = '/content/drive/My Drive/image_aug'
train_generator = train_datagen.flow_from_directory(
train_data_dir,
target_size = (img_height, img_width),
batch_size = batch_size,
class_mode = "binary")
```

In [0]:

```
validation_generator = test_datagen.flow_from_directory(
validation_data_dir,shuffle=True,batch_size = batch_size,
target_size = (img_height, img_width),
class_mode = "binary")
```

In [0]:

```
import matplotlib.pyplot as plt
x_batch, y_batch = next(validation_generator)
for i in range (0,16):
    image = x_batch[i]
    plt.imshow(image)
    plt.show()
```

In [0]:

```
checkpoint = ModelCheckpoint("/content/drive/My Drive/imagenew_sgd1.h5", monitor='val_a
cc', verbose=1, save_best_only=True, save_weights_only=False, mode='auto', period=1)
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=10, verbose=1, mode='aut
o')
```

Training the model

In [0]:

```
history = model_final.fit_generator(
train_generator,
steps_per_epochs = nb_train_samples//batch_size,
epochs = 50,
validation_data = validation_generator,
validation_steps = nb_validation_samples//batch_size,
callbacks = [checkpoint,early])
```

Validating the model on Random image

In [0]:

```
from tensorflow.keras.preprocessing import image
img = image.load_img("/content/drive/My Drive/validation_AI_new/No Dent/0044.jpg", targ
et_size = (224, 224))
```

In [0]:

```
import matplotlib.pyplot as plt
plt.imshow(img)
#plt.title(classname)
plt.show()
import numpy as np
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)

preds = model_final.predict(x)
```

In [0]:

```
preds = [0 if preds < 0.5 else 1]
```

In [0]:

```
preds
```

Evaluating the performance of the model

In [0]:

```
from keras.models import load_model
from keras.preprocessing import image
import numpy as np
import os

folder_path = '/content/drive/My Drive/validation_AI_new/Dent'
model_final.load_weights("/content/drive/My Drive/imagenew_sgd1.h5")
img_width, img_height = 224,224
```

In [0]:

```
# load all images into a list
images = []
for img in os.listdir(folder_path):
    img = image.load_img(folder_path+'/'+img, target_size=(img_width, img_height))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    images.append(img)
```

In [0]:

```
len(images) #number of dent images
```

In [0]:

```
folder_path = '/content/drive/My Drive/validation_AI_new/No Dent'
for img in os.listdir(folder_path):
    img = image.load_img(folder_path+'/'+img, target_size=(img_width, img_height))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    images.append(img)
```

In [0]:

```
len(images)#150 no dent images
```

In [0]:

```
np.save(open('/content/drive/My Drive/validation_set.npy', 'wb'),images)
```

In [0]:

```
validation_data = np.load(open('/content/drive/My Drive/validation_set.npy', 'rb'))
validation_labels = np.array([0] * 176 + [1] * 150)
```

In [0]:

```
classes = []
for i in validation_data:
  preds = model_final.predict(i)
  classes.append(preds)
```

In [0]:

```
res = []
for i in classes:
  if i<0.5:
    res.append(0)
  else:
    res.append(1)
```

In [0]:

```
np.mean(res==validation_labels)
```

In [0]:

```
#confusion matrix for the validation set
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix
cm = confusion_matrix(validation_labels, res)
cm_df = pd.DataFrame(cm.T, index=['Dent', 'No Dent'], columns=['Dent', 'No Dent'])
cm_df.index.name = 'Predicted'
cm_df.columns.name = 'True'
print(cm_df)
```

In [0]:

```
#classification report for the validation set
target_names =['Dent', 'No_dent']
print(classification_report(validation_labels, res, target_names=target_names))
```

In [0]:

```
#27 dent #26 no dent
folder_path = '/content/drive/My Drive/test_AI_new/dent'

# load all images into a list
images = []
for img in os.listdir(folder_path):
    img = image.load_img(folder_path+'/'+img, target_size=(img_width, img_height))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    images.append(img)
```

In [0]:

```
len(images)#dent images count = 27
```

In [0]:

```
folder_path = '/content/drive/My Drive/test_AI_new/No dent'
for img in os.listdir(folder_path):
    img = image.load_img(folder_path+'/'+img, target_size=(img_width, img_height))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    images.append(img)
```

In [0]:

```
len(images)#26 no dent images
```

In [0]:

```
np.save(open('/content/drive/My Drive/test_set.npy', 'wb'),
            images)
```

In [0]:

```
test_data = np.load(open('/content/drive/My Drive/test_set.npy', 'rb'))
print(test_data.shape)
test_labels = np.array([0] * 27 + [1] * 26)
```

In [0]:

```
classes_2 = []
for i in test_data:
  preds = model_final.predict(i)
  classes_2.append(preds)
```

In [0]:

```
res_2 = []
for i in classes_2:
  if i<0.5:
    res_2.append(0)
  else:
    res_2.append(1)
```

In [0]:

```
np.mean(res_2==test_labels)
```

In [0]:

```
# confusion matrix for the test set
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix
cm = confusion_matrix(test_labels, res_2)
cm_df = pd.DataFrame(cm.T, index=['Dent', 'No Dent'], columns=['Dent', 'No Dent'])
cm_df.index.name = 'Predicted'
cm_df.columns.name = 'True'
print(cm_df)
```

In [0]:

```
#classification report for the test set
target_names =['Dent', 'No_dent']
print(classification_report(test_labels, res_2, target_names=target_names))
```

Visualising the Loss and accuracy of the model

In [0]:

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

In [0]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Section C- Spotting the dent in images

In [ ]:

```
#CUSTOM.PY file used to train the model

import os
import sys
import json
import datetime
import numpy as np
import skimage.draw
import cv2
from mrcnn.visualize import display_instances
import matplotlib.pyplot as plt

# Root directory of the project
ROOT_DIR = os.path.abspath("../../")

# Import Mask RCNN
sys.path.append(ROOT_DIR)
from mrcnn.config import Config
from mrcnn import model as modellib, utils

COCO_WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")

DEFAULT_LOGS_DIR = os.path.join('/content', "logs")

############################################################
#  Configurations
############################################################


class CustomConfig(Config):
    """Configuration for training on the toy  dataset.
    Derives from the base Config class and overrides some values.
    """

    NAME = "dent"

    IMAGES_PER_GPU = 2

    # Number of classes (including background)
    NUM_CLASSES = 1 + 1  # Background + toy

    # Number of training steps per epoch
    STEPS_PER_EPOCH = 100

    # Skip detections with < 90% confidence
    DETECTION_MIN_CONFIDENCE = 0.9


############################################################
#  Dataset
############################################################

class CustomDataset(utils.Dataset):

    def load_custom(self, dataset_dir, subset):
        # updating the class
        self.add_class("dent", 1, "dent")
        #choosing the train or val set
        assert subset in ["train", "val"]
```

```
        dataset_dir = os.path.join(dataset_dir, subset)
        annotations1 = json.load(open(os.path.join(dataset_dir, "via_region_data.json"
)))
        annotations = list(annotations1.values())
        annotations = [a for a in annotations if a['regions']]

        # Add images
        for k in annotations:
            polygons = [r['shape_attributes'] for r in k['regions'].values()]
            image_path = os.path.join(dataset_dir, k['filename'])
            image = skimage.io.imread(image_path)
            height, width = image.shape[:2]

            self.add_image(
                "dent",  # adding the name of class
                image_id=k['filename'],  # image id as file name
                path=image_path,
                width=width, height=height,
                polygons=polygons)

    def load_mask(self, image_id):
        # If not a balloon dataset image, delegate to parent class.
        image_info = self.image_info[image_id]
        if image_info["source"] != "dent":
            return super(self.__class__, self).load_mask(image_id)

        # Convert polygons to a bitmap mask of shape
        # [height, width, instance_count]
        info = self.image_info[image_id]
        mask = np.zeros([info["height"], info["width"], len(info["polygons"])],
                        dtype=np.uint8)
        for i, p in enumerate(info["polygons"]):
            # Get indexes of pixels inside the polygon and set them to 1
            rr, cc = skimage.draw.polygon(p['all_points_y'], p['all_points_x'])
            mask[rr, cc, i] = 1

        # Return mask, and array of class IDs of each instance. Since we have
        # one class ID only, we return an array of 1s
        return mask.astype(np.bool), np.ones([mask.shape[-1]], dtype=np.int32)

    def image_reference(self, image_id):
        """Return the path of the image."""
        info = self.image_info[image_id]
        if info["source"] == "dent":
            return info["path"]
        else:
            super(self.__class__, self).image_reference(image_id)


def train(model):
    """Train the model."""
    # Training dataset.
    dataset_train = CustomDataset()
    dataset_train.load_custom(args.dataset, "train")
    dataset_train.prepare()

    # Validation dataset
    dataset_val = CustomDataset()
    dataset_val.load_custom(args.dataset, "val")
    dataset_val.prepare()
```

```
    # *** This training schedule is an example. Update to your needs ***
    # Since we're using a very small dataset, and starting from
    # COCO trained weights, we don't need to train too long. Also,
    # no need to train all layers, just the heads should do it.
    print("Training network heads")
    model.train(dataset_train, dataset_val,
                learning_rate=config.LEARNING_RATE,
                epochs=10,
                layers='heads')


def color_splash(image, mask):
    """Apply color splash effect.
    image: RGB image [height, width, 3]
    mask: instance segmentation mask [height, width, instance count]

    Returns result image.
    """
    # Make a grayscale copy of the image. The grayscale copy still
    # has 3 RGB channels, though.
    gray = skimage.color.gray2rgb(skimage.color.rgb2gray(image)) * 255
    # We're treating all instances as one, so collapse the mask into one layer
    mask = (np.sum(mask, -1, keepdims=True) >= 1)
    # Copy color pixels from the original color image where mask is set
    if mask.shape[0] > 0:
        splash = np.where(mask, image, gray).astype(np.uint8)
    else:
        splash = gray
    return splash


def detect_and_color_splash(model, image_path=None, video_path=None):
    assert image_path or video_path

    # Image or video?
    if image_path:
        # Run model detection and generate the color splash effect
        print("Running on {}".format(args.image))
        # Read image
        image = skimage.io.imread(args.image)
        # Detect objects
        r = model.detect([image], verbose=1)[0]
        # Color splash
        splash = color_splash(image, r['masks'])
        # Save output
        file_name = "splash_{:%Y%m%dT%H%M%S}.png".format(datetime.datetime.now())
        skimage.io.imsave(file_name, splash)
    elif video_path:
        import cv2
        # Video capture
        vcapture = cv2.VideoCapture(video_path)
        width = int(vcapture.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(vcapture.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = vcapture.get(cv2.CAP_PROP_FPS)

        # Define codec and create video writer
        file_name = "splash_{:%Y%m%dT%H%M%S}.avi".format(datetime.datetime.now())
        vwriter = cv2.VideoWriter(file_name,
                                  cv2.VideoWriter_fourcc(*'MJPG'),
                                  fps, (width, height))
```

```
        count = 0
        success = True
        while success:
            print("frame: ", count)
            # Read next image
            success, image = vcapture.read()
            if success:
                # OpenCV returns images as BGR, convert to RGB
                image = image[..., ::-1]
                # Detect objects
                r = model.detect([image], verbose=0)[0]
                # Color splash
                splash = color_splash(image, r['masks'])
                # RGB -> BGR to save image to video
                splash = splash[..., ::-1]
                # Add image to video writer
                vwriter.write(splash)
                count += 1
        vwriter.release()
    print("Saved to ", file_name)

############################################################
#  Training
############################################################

if __name__ == '__main__':
    import argparse

    # Parse command line arguments
    parser = argparse.ArgumentParser(
        description='Train Mask R-CNN to detect custom class.')
    parser.add_argument("command",
                        metavar="<command>",
                        help="'train' or 'splash'")
    parser.add_argument('--dataset', required=False,
                        metavar="/path/to/custom/dataset/",
                        help='Directory of the custom dataset')
    parser.add_argument('--weights', required=True,
                        metavar="/path/to/weights.h5",
                        help="Path to weights .h5 file or 'coco'")
    parser.add_argument('--logs', required=False,
                        default=DEFAULT_LOGS_DIR,
                        metavar="/path/to/logs/",
                        help='Logs and checkpoints directory (default=logs/)')
    parser.add_argument('--image', required=False,
                        metavar="path or URL to image",
                        help='Image to apply the color splash effect on')
    parser.add_argument('--video', required=False,
                        metavar="path or URL to video",
                        help='Video to apply the color splash effect on')
    args = parser.parse_args()

    # Validate arguments
    if args.command == "train":
        assert args.dataset, "Argument --dataset is required for training"
    elif args.command == "splash":
        assert args.image or args.video,\
               "Provide --image or --video to apply color splash"

    print("Weights: ", args.weights)
    print("Dataset: ", args.dataset)
```

```
print("Logs: ", args.logs)

# Configurations
if args.command == "train":
    config = CustomConfig()
else:
    class InferenceConfig(CustomConfig):
        # Set batch size to 1 since we'll be running inference on
        # one image at a time. Batch size = GPU_COUNT * IMAGES_PER_GPU
        GPU_COUNT = 1
        IMAGES_PER_GPU = 1
    config = InferenceConfig()
config.display()

# Create model
if args.command == "train":
    model = modellib.MaskRCNN(mode="training", config=config,
                              model_dir=args.logs)
else:
    model = modellib.MaskRCNN(mode="inference", config=config,
                              model_dir=args.logs)

# Select weights file to load
if args.weights.lower() == "coco":
    weights_path = COCO_WEIGHTS_PATH
    # Download weights file
    if not os.path.exists(weights_path):
        utils.download_trained_weights(weights_path)
elif args.weights.lower() == "last":
    # Find last trained weights
    weights_path = model.find_last()[1]
elif args.weights.lower() == "imagenet":
    # Start from ImageNet trained weights
    weights_path = model.get_imagenet_weights()
else:
    weights_path = args.weights

# Load weights
print("Loading weights ", weights_path)
if args.weights.lower() == "coco":
    # Exclude the last layers because they require a matching
    # number of classes
    model.load_weights(weights_path, by_name=True, exclude=[
        "mrcnn_class_logits", "mrcnn_bbox_fc",
        "mrcnn_bbox", "mrcnn_mask"])
else:
    model.load_weights(weights_path, by_name=True)

# Train or evaluate
if args.command == "train":
    train(model)
elif args.command == "splash":
    detect_and_color_splash(model, image_path=args.image,
                            video_path=args.video)
else:
    print("'{}' is not recognized. "
          "Use 'train' or 'splash'".format(args.command))
```

Section D- Spotting the dent in images

In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

In [0]:

```
!git clone https://github.com/matterport/Mask_RCNN.git
```

In [0]:

```
cd /content/Mask_RCNN/
```

In [0]:

```
!pip install -e .
```

Visualising the data

In [0]:

```
import os
import sys
import itertools
import math
import logging
import json
import re
import random
from collections import OrderedDict
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lines
from matplotlib.patches import Polygon

# Root directory of the project
ROOT2_DIR = '/content/drive/My Drive/'

# Import Mask RCNN
sys.path.append(ROOT2_DIR)  # To find local version of the library
from mrcnn import utils
from mrcnn import visualize
from mrcnn.visualize import display_images
import mrcnn.model as modellib
from mrcnn.model import log
import custom


%matplotlib inline
```

In [0]:

```
#configurations
config = custom.CustomConfig()
CUSTOM_DIR = os.path.join(ROOT2_DIR, "customImages")
print(CUSTOM_DIR)
```

In [0]:

```
#dataset

import json
# Load dataset
# Get the dataset from the releases page
# https://github.com/matterport/Mask_RCNN/releases
dataset = custom.CustomDataset()
dataset.load_custom(CUSTOM_DIR, "train")

# Must call before using the dataset
dataset.prepare()

print("Image Count: {}".format(len(dataset.image_ids)))
print("Class Count: {}".format(dataset.num_classes))
for i, info in enumerate(dataset.class_info):
    print("{:3}. {:50}".format(i, info['name']))
```

In [0]:

```
# Load and display random samples
image_ids = np.random.choice(dataset.image_ids, 4)
for image_id in image_ids:
    image = dataset.load_image(image_id)
    mask, class_ids = dataset.load_mask(image_id)
    visualize.display_top_masks(image, mask, class_ids, dataset.class_names)
```

In [0]:

```
#Bounding Boxes

image_id = random.choice(dataset.image_ids)
image = dataset.load_image(image_id)
mask, class_ids = dataset.load_mask(image_id)
# Compute Bounding box
bbox = utils.extract_bboxes(mask)

# Display image and additional stats
print("image_id ", image_id, dataset.image_reference(image_id))
log("image", image)
log("mask", mask)
log("class_ids", class_ids)
log("bbox", bbox)
# Display image and instances
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

In [0]:

```
#Resize the images

image_id = dataset.image_ids[:]
for i in image_id:
    image = dataset.load_image(i)
    mask, class_ids = dataset.load_mask(i)
    # Compute Bounding box
    bbox = utils.extract_bboxes(mask)

    # Display image and additional stats
    print("i ", i, dataset.image_reference(i))
    log("image", image)
    log("mask", mask)
    log("class_ids", class_ids)
    log("bbox", bbox)
    # Display image and instances
    visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

In [0]:

```
#Mini Masks

image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
    dataset, config, image_id, use_mini_mask=False)

log("image", image)
log("image_meta", image_meta)
log("class_ids", class_ids)
log("bbox", bbox)
log("mask", mask)

display_images([image]+[mask[:,:,i] for i in range(min(mask.shape[-1], 7))])
```

In [0]:

```
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

In [0]:

```
# Add augmentation and mask resizing.
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
    dataset, config, image_id, augment=True, use_mini_mask=True)
log("mask", mask)
display_images([image]+[mask[:,:,i] for i in range(min(mask.shape[-1], 7))])
```

In [0]:

```
mask = utils.expand_mask(bbox, mask, image.shape)
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

Training the model

In [0]:

```
cd /content/drive/My Drive/
```

In [0]:

```
!pwd
```

In [0]:

```
#initially training the model wit coco weights
!python /custom.py train --dataset=/content/drive/My\ Drive/customImages --weights=coco
#Training the model with last trained weights
!python /content/custom.py train --dataset=/content/drive/My\ Drive/customImages --weig
hts=mask_rcnn_dent_0015_3.h5
```

Evaluating the model

In [0]:

```
import os
import cv2
import sys
import random
import math
import re
import time
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import skimage
import glob

# Root directory of the project
ROOT2_DIR = '/content/drive/My Drive/'

# Import Mask RCNN
sys.path.append(ROOT2_DIR)  # To find local version of the library
from mrcnn import utils
from mrcnn import visualize
from mrcnn.visualize import display_images
import mrcnn.model as modellib
from mrcnn.model import log

import custom

%matplotlib inline

# Directory to save logs and trained model
MODEL_DIR = os.path.join('/content', "logs")

custom_WEIGHTS_PATH = "/content/drive/My Drive/mask_rcnn_dent_0015_3.h5"  # TODO: updat
e this path
```

In [0]:

```
config = custom.CustomConfig()
custom_DIR = os.path.join(ROOT2_DIR, "customImages")
```

In [0]:

```
# Override the training configurations with a few
# changes for inferencing.
class InferenceConfig(config.__class__):
    # Run detection on one image at a time
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

config = InferenceConfig()
config.display()
```

In [0]:

```
# Device to load the neural network on.
# Useful if you're training a model on the same
# machine, in which case use CPU and leave the
# GPU for training.
DEVICE = "/gpu:0"  # /cpu:0 or /gpu:0

# Inspect the model in training or inference modes
# values: 'inference' or 'training'
# TODO: code for 'training' test mode not ready yet
TEST_MODE = "inference"
```

In [0]:

```
def get_ax(rows=1, cols=1, size=16):
    """Return a Matplotlib Axes array to be used in
    all visualizations in the notebook. Provide a
    central point to control graph sizes.

    Adjust the size attribute to control how big to render images
    """
    _, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
    return ax
```

In [0]:

```
# Load validation dataset
dataset = custom.CustomDataset()
dataset.load_custom(custom_DIR, "val")

# Must call before using the dataset
dataset.prepare()

print("Images: {}\nClasses: {}".format(len(dataset.image_ids), dataset.class_names))
```

In [0]:

```
# Create model in inference mode
with tf.device(DEVICE):
    model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR,
                              config=config)
```

In [0]:

```
# load the last model you trained
# weights_path = model.find_last()[1]

# Load weights
print("Loading weights ", custom_WEIGHTS_PATH)
model.load_weights(custom_WEIGHTS_PATH, by_name=True)
```

In [0]:

```
from importlib import reload # was constantly changin the visualization, so I decided t
o reload it instead of notebook
reload(visualize)
```

In [0]:

```
image_id =random.choice(dataset.image_ids)
image, image_meta, gt_class_id, gt_bbox, gt_mask =\
    modellib.load_image_gt(dataset, config,image_id, use_mini_mask=False)
info = dataset.image_info[image_id]
print("image ID: {}.{} ({}) {}".format(info["source"], info["id"], image_id,
                                       dataset.image_reference(image_id)))

# Run object detection
results = model.detect([image], verbose=1)

# Display results
ax = get_ax(1)
r = results[0]
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                            dataset.class_names, r['scores'], ax=ax,
                            title="Predictions")
log("gt_class_id", gt_class_id)
log("gt_bbox", gt_bbox)
log("gt_mask", gt_mask)
```

validating the model weights

In [0]:

```
import os
import sys
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
import keras

# Root directory of the project
ROOT2_DIR = os.path.abspath("../../")

# Import Mask RCNN
sys.path.append(ROOT2_DIR)  # To find local version of the library
from mrcnn import utils
import mrcnn.model as modellib
from mrcnn import visualize
from mrcnn.model import log

%matplotlib inline

# # Directory to save logs and trained model
MODEL_DIR = os.path.join('/content', "logs")
```

In [0]:

```
import custom
config = custom.CustomConfig()
```

In [0]:

```
DEVICE = "/gpu:0"
def get_ax(rows=1, cols=1, size=16):
    """Return a Matplotlib Axes array to be used in
    all visualizations in the notebook. Provide a
    central point to control graph sizes.

    Adjust the size attribute to control how big to render images
    """
    _, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
    return ax
```

In [0]:

```
# Create trainmodel in inference mode
with tf.device(DEVICE):
    model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR,
                              config=config)
# Load weights
weights_path = '/content/drive/My Drive/mask_rcnn_dent_0015_3.h5'

# Load weights
print("Loading weights ", weights_path)
model.load_weights(weights_path, by_name=True)
```

In [0]:

```
# Pick layer types to display
LAYER_TYPES = ['Conv2D', 'Dense', 'Conv2DTranspose']
# Get layers
layers = model.get_trainable_layers()
layers = list(filter(lambda l: l.__class__.__name__ in LAYER_TYPES,
                layers))
# Display Histograms
fig, ax = plt.subplots(len(layers), 2, figsize=(10, 3*len(layers)),
                       gridspec_kw={"hspace":1})
for l, layer in enumerate(layers):
    weights = layer.get_weights()
    for w, weight in enumerate(weights):
        tensor = layer.weights[w]
        ax[l, w].set_title(tensor.name)
        _ = ax[l, w].hist(weight[w].flatten(), 50)
```