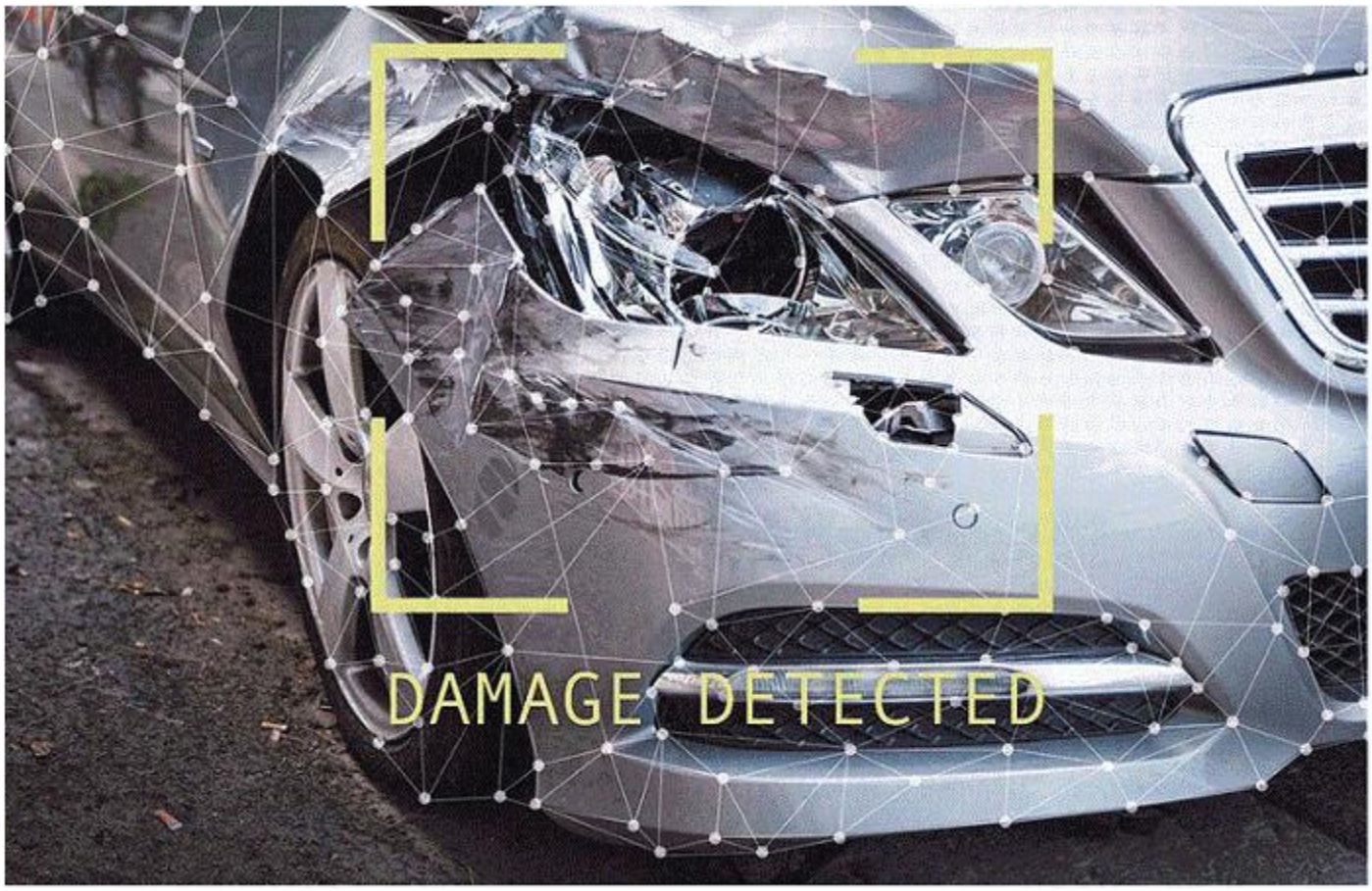
Car Damage Assessment for Image Data Genarator



Overview:

In Car Insurance industry, a lot of money is being wasted on Claims leakage. Claims leakage is the gap between the optimal and actual settlement of a claim. Visual inspection and validation are being used to reduce claims leakage. But doing inspection might take a long time and result in delaying of claims processing. An automated system for doing inspection and validation will be of great help in speeding up the process.

1. Business Use Case:

To reduce Claims leakage during Insurance processing. Visual inspection and validation are being done. As this takes a long time because the person needs to come and inspect the damage. We are trying to automate this procedure. Using this automation will result in Claims processing faster.

2. Data Source:

This consists of Train and Validation which each folder has Damage cars pics and whole car pics. A total of 2300 images are present in both train and validation combined.

3. Existing Approaches:

In above mentioned paper the team collected the images and sorted the dataset into 8 classes (Bumper dent, Door dent, Glass shatter, Head lamp Broken, Tail lamp broken, Scratch, Smash, No damage).

Since the dataset containing images are less so they synthetically enlarged the dataset 5 times using Augmentation of Random rotation between -20 and 20 with Horizontal flip.

Method	Without Augmentation			With Augmentation		
	Acc	Prec	Recall	Acc	Prec	Recall
CNN	71.33	63.27	52.5	72 46	64 03	61 01
AE-CNN	73.43	67.21	55.32	72 30	63 69	59 48

Model	Params	Dim	Without Augmentation						With Augmentation					
			Linear SVM			Softmax			Linear SVM			Softmax		
			Acc	Prec	Recall	Acc	Prec	Recall	Acc	Prec	Recall	Acc	Prec	Recall
Cars [14]	6.8M	1024	57.33	47 24	56 46	60 38	47 23	32 39	58 45	48.58	56 97	64 25	52 73	39 16
Inception [13]	5M.	2048	68.12	57 46	55 53	71 82	61 75	56 71	68 60	58.50	54 44	71 50	69 47	52 81
Alexnet	60 M.	4096	70.85	61 68	64 60	70 85	61 42	58 09	73 26	62.83	61 72	73 91	66 83	63 36
VGG-19 [12]	144M.	4096	82.77	78 62	73 16	84 22	80 76	73 60	82 29	76.30	70 60	83 90	80 74	73 41
VGG-16 [12]	138M.	4096	83.74	77 79	75 41	84 86	81 91	73 56	82 93	78.62	71 96	82 72	78 99	70 30
Resnet [15]	25.6M.	2048	86.31	80 87	78 30	**88.24**	84 38	81 10	87 92	84.40	78 94	87 92	83 68	79 47

They enlarged the dataset 4 times by applying rotation 20 degrees, shear of 0.2, zoom of range 0.2 and horizontal flip.

Instead of training CNN without pretrained weights. They went for Pre trained models (Alexnet, Inception V3, VGG19, Resnet50, Mobile nets). They trained only FC layers and All layers for every pretrained model.

Table 3. Test accuracy using these techniques

Model	Acc(training FC layers only)	Acc(training all layers with differential learning rate annealing)	Precision	Recall	F-beta Score	Acc(Using test time augmentation)
VGG16	90.97% (10 epochs)	93.81% (15 epochs)	0.907	0.886	0.891	94.84%
VGG19	90.46% (10 epochs)	95.36% (15 epochs)	0.922	0.908	0.914	95.87%
Resnet34	90.20% (10 epochs)	93.29% (15 epochs)	0.857	0.830	0.837	93.88%
Resnet50	91.75% (10 epochs)	96.13% (15 epochs)	0.928	0.922	0.922	96.39%

In all the above models only pretrained models are performing better than training from scratch

4. Exploratory Data Analysis:

We have three types of Data:

1. Training and Test folders of Car damaged, not damaged images.
2. Training and Test folders of Damage on Front, Rear, Side.
3. Training and Test folders of Damage severity Minor, Moderate, Severe.

Stage 1 (Damaged or Not Damaged):

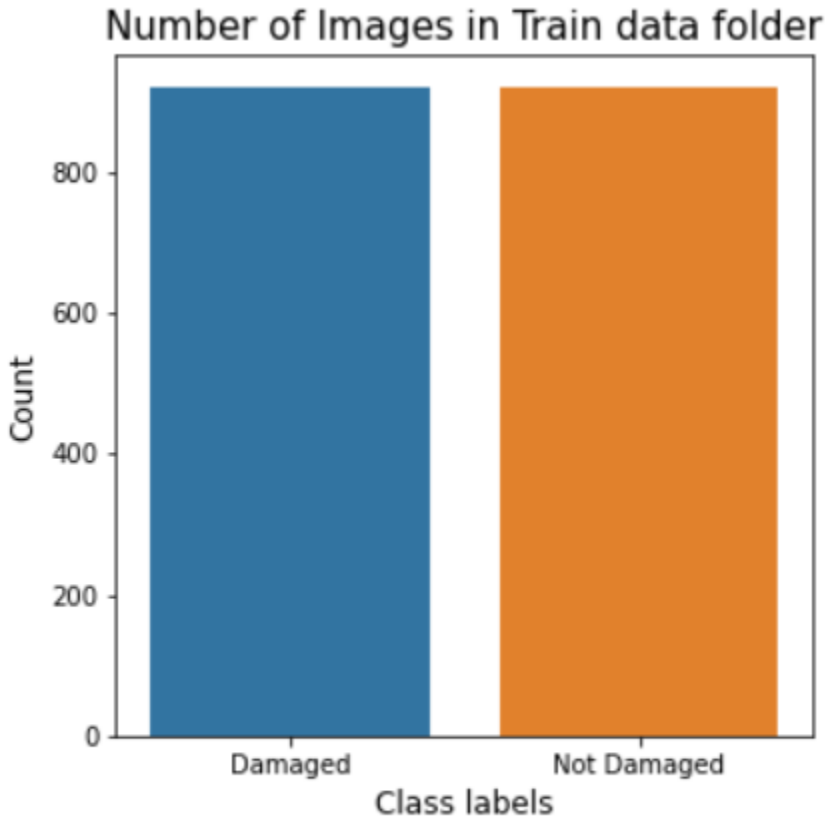
In EDA we will look at how many files each folder has and the top image height and width in our data.

Bar Plot:

```
def plot_bar(class_labels,counts,name):
    plt.figure(figsize = (5,5))
    f = sns.barplot(x = classes,y =counts)
    plt.xlabel("Class labels", fontsize=12)
    plt.ylabel('Count', fontsize=12)
    plt.title("Number of Images in "+name+' folder', fontsize=15)
    plt.show()


classes = ['Damaged','Not Damaged']
counts = [len(train_damaged_list),len(train_not_damaged_list)]
```

```
plot_bar(classes,counts,'Train data')

for i in range(len(classes)):

    print('Number of '+classes[i]+' images in
train is '+str(counts[i]))
```

```
classes = ['Damaged','Not Damaged']

counts =
[len(test_damaged_list),len(test_not_damaged_l
ist)]

plot_bar(classes,counts,'Test data')

for i in range(len(classes)):

print('Number of '+classes[i]+' images in test
is '+str(counts[i]))
```




```
Number of Damaged images in train is 920
Number of Not Damaged images in train is 920
```

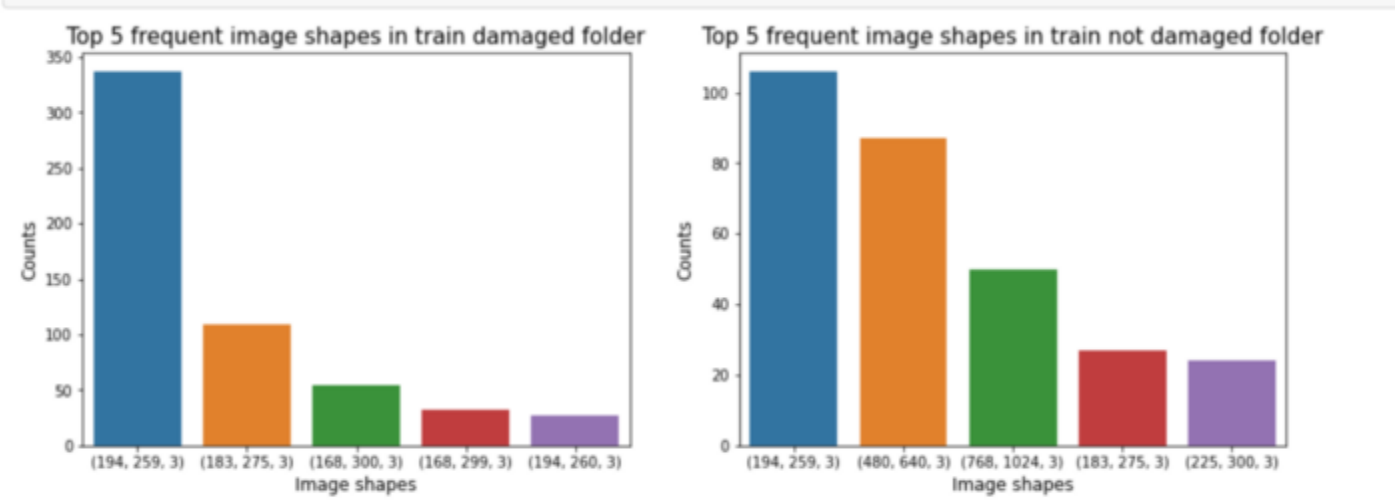

```
Number of Damaged images in test is 230
Number of Not Damaged images in test is 230
```

Image Sizes plot:

```
def
img_shapes(files,path):
                            shapes = []
                            for i in files:
                                img = cv2.imread(path+'/'+i)
                                shapes.append(img.shape)
                            return shapes


                            train_dmg =
                        img_shapes(train_damaged_list,'drive_data/data1a/training/00-damage')
```

```
    train_no_dmg =
img_shapes(train_not_damaged_list,'drive_data/data1a/training/01-
whole')

    train_df = pd.DataFrame(list(zip(train_dmg,train_no_dmg)),columns =
classes)


    #Damaged train

num = [str(i) for i in train_df['Damaged'].value_counts().index[:5]]

counts_val = [i for i in train_df['Damaged'].value_counts()[:5]]

plt.figure(figsize = (15,5))

plt.subplot(1,2,1)

sns.barplot(x = num,y =counts_val)

plt.xlabel("Image shapes", fontsize=12)

plt.ylabel('Counts', fontsize=12)

plt.title("Top 5 frequent image shapes in train damaged folder",
fontsize=15)



#Not Damaged train

num = [str(i) for i in train_df['Not
Damaged'].value_counts().index[:5]]

counts_val = [i for i in train_df['Not Damaged'].value_counts()[:5]]

plt.subplot(1,2,2)

sns.barplot(x = num,y =counts_val)

plt.xlabel("Image shapes", fontsize=12)

plt.ylabel('Counts', fontsize=12)

plt.title("Top 5 frequent image shapes in train not damaged folder",
fontsize=15)

plt.show()
```

```
print('No of unique image shapes in train damaged are
',len(train_df['Damaged'].unique()))

print('No of unique image shapes in train not damaged are
',len(train_df['Not Damaged'].unique()))
```



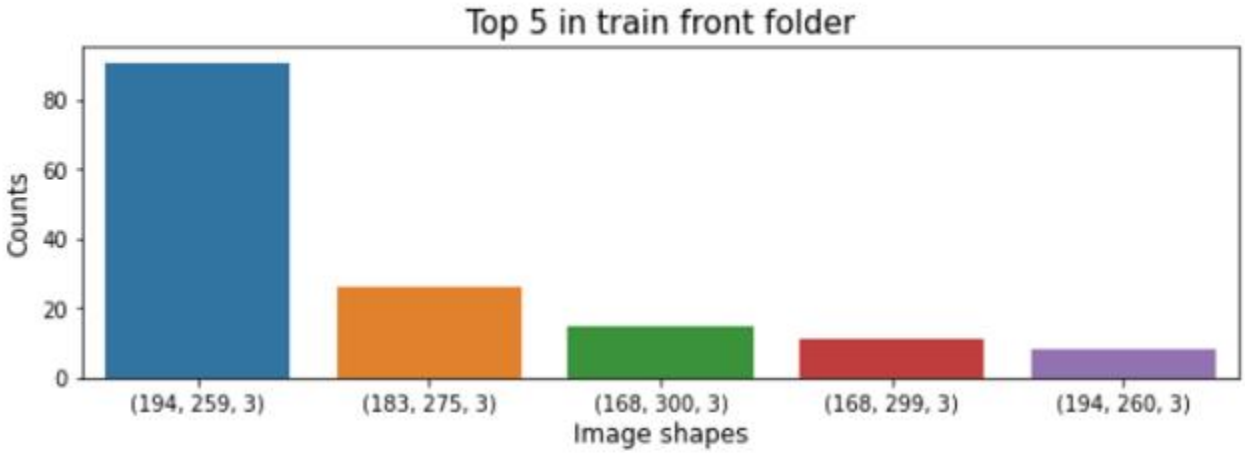

```
No of unique image shapes in train damaged are  132
No of unique image shapes in train not damaged are  385
```

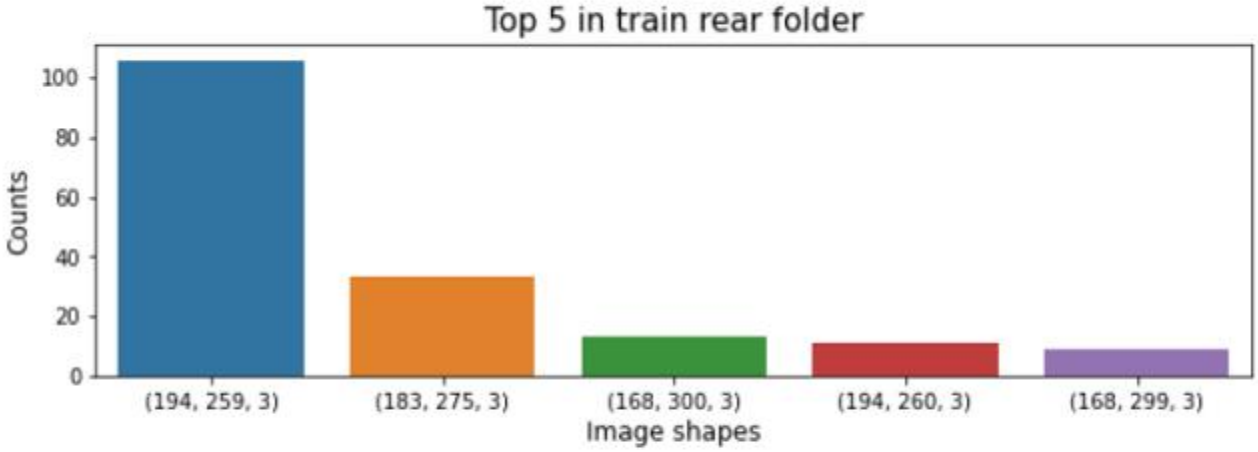
Stage 2 (Front, Rear and Side):

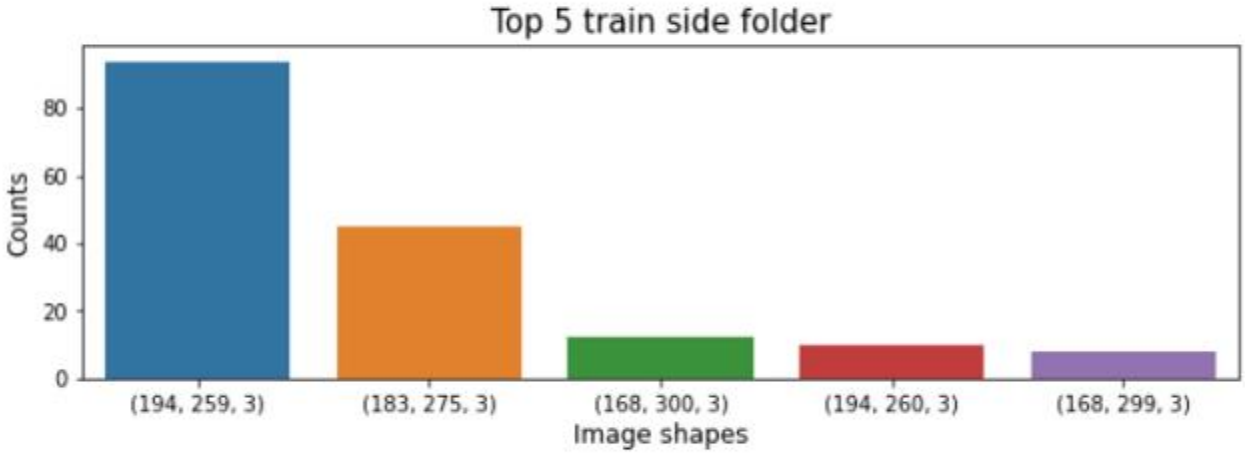
Bar plot:


```
Number of Front images in test is 73
Number of Rear images in test is 58
Number of Side images in test is 48
```

Image plots:








```
No of unique image shapes in train front are  70
No of unique image shapes in train rear are  67
No of unique image shapes in train side are  60
```

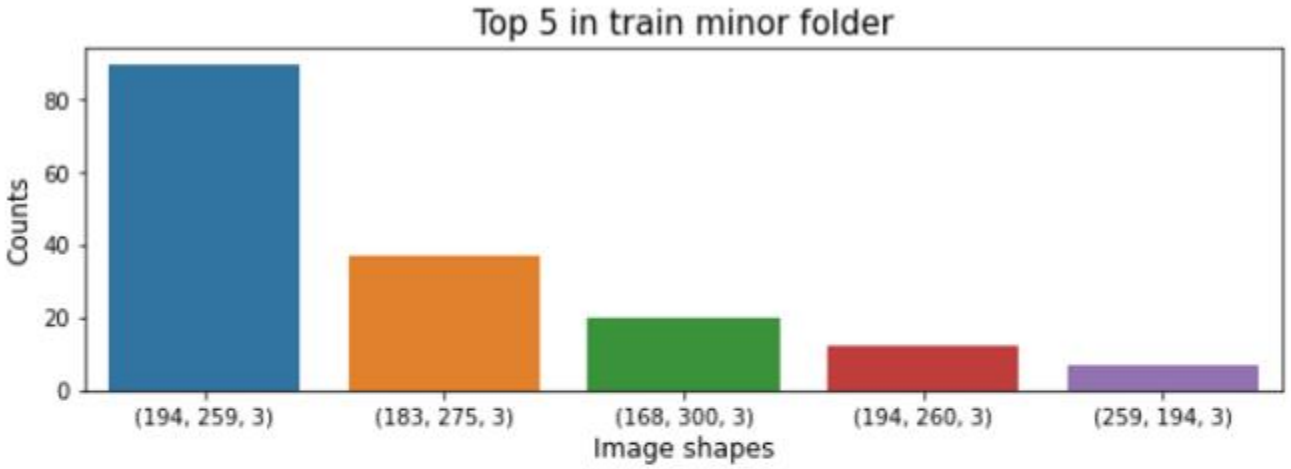
Stage 3(Minor, Moderate and Severe):

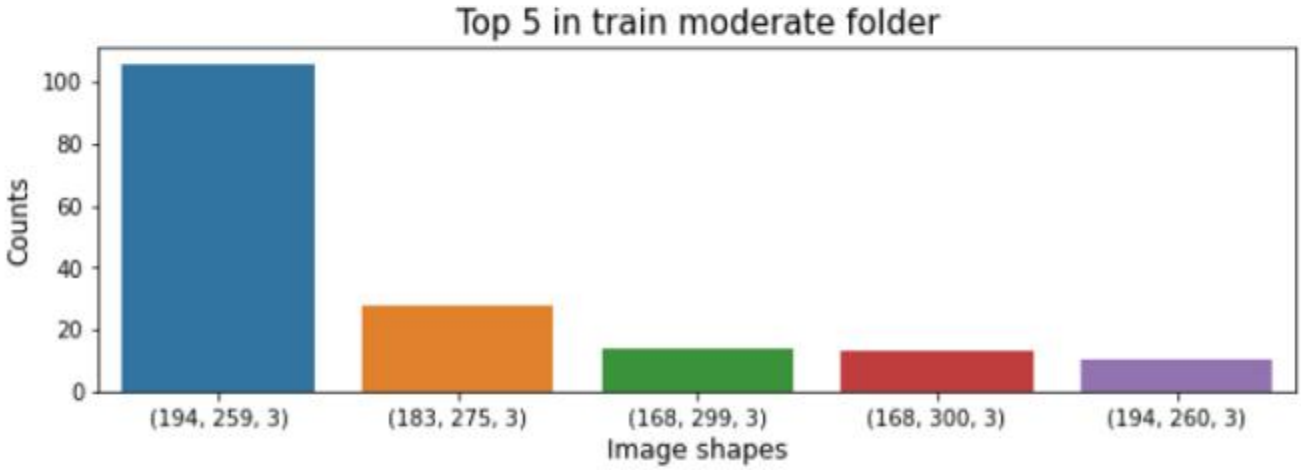
Bar plots:

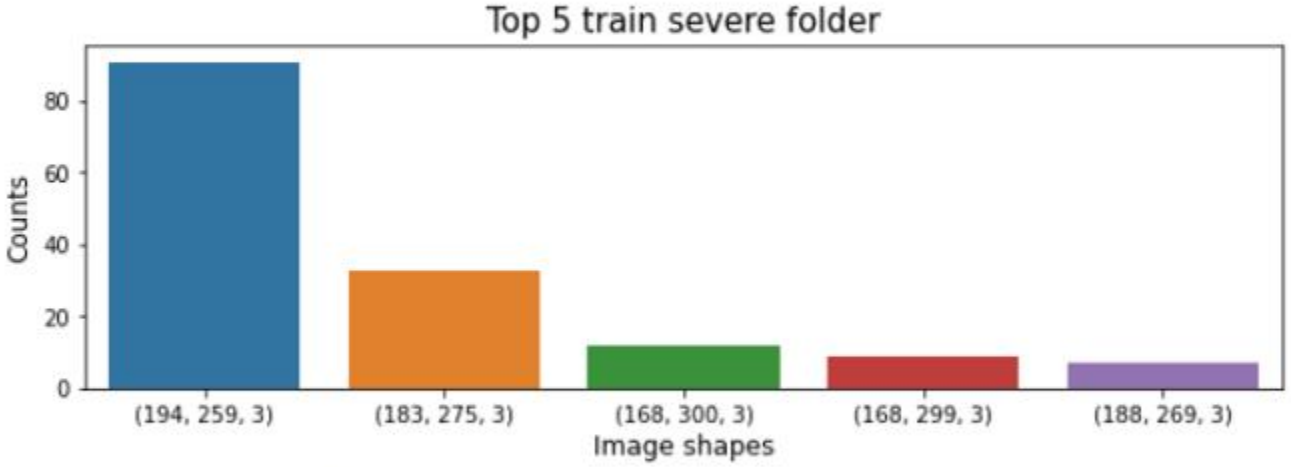

```
Number of Minor images in train is 278
Number of Moderate images in train is 315
Number of Severe images in train is 386
```


```
Number of Minor images in test is 48
Number of Moderate images in test is 55
Number of Severe images in test is 68
```

Image plots:








```
No of unique image shapes in train minor are  66
No of unique image shapes in train moderate are  63
No of unique image shapes in train severe are  70
```

5.First Cut Approach:

Since the dataset we have is small we will synthetically enlarge the dataset twice using Data Augmentation. Various papers shows different types of data augmentation. We will use below two types of data augmentation and see which performs better.

1. Random rotation between -20 and 20 degrees and horizontal flip transformations.
2. Random rotation between -20 and 20, shear-range of 0.2, zoom-range of 0.2 and horizontal-flip.

We will create required training folders to save the data augmented images.

Data Augmentation:

Since the data we have are less we will use Data augmentation to synthetically enlarge the dataset. As mentioned in existing approaches, Iwill be using the two types of augmentation and enlarging the dataset twice.

1. Enlarging the dataset twice using random rotation between -20 and 20 degrees and horizontal flip transformations.

2. Enlarging the dataset twice using random rotation between -20 and 20, shear-range of 0.2, zoom-range of 0.2 and horizontal-flip.

Visualization of Data Augmentation:

Data Augmentation 1:

```
#Getting
a sample
file
name
from
list of
file
names
        sample_file = random.choice(train_damaged_list)



        #Getting a random rotation between -20 and 20
        rotation = random.randint(-20,20)


        augmentation = ImageDataGenerator(rotation_range=rotation,horizontal_flip=True)


        sample_img = image.load_img('drive_data/data1a/training/00-damage'+'/'+sample_file)
        sample_img_array = image.img_to_array(sample_img)
        sample_img_array = sample_img_array.reshape((1,) + sample_img_array.shape)


        samples = []
        for arr,val in zip(augmentation.flow(sample_img_array, batch_size=1,
        save_format='jpg'),range(1)):
            img_save = image.array_to_img(arr[0], scale=False)
            samples.append(img_save)


        plt.imshow(samples[0])
```

Original

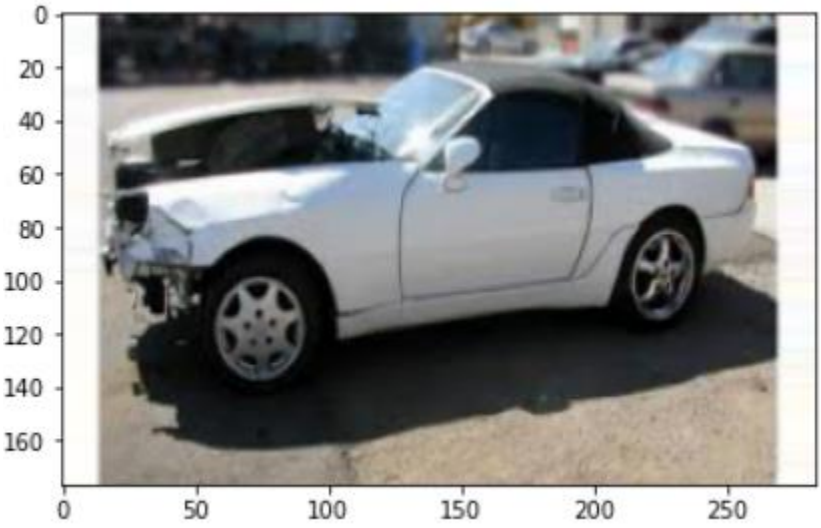
Data Augmented Image

Data Augmentation 2:

```
#Getting
a sample
file
name
from
list of
file
names
        sample_file = random.choice(train_damaged_list)


        #Getting a random rotation between -20 and 20
        rotation = random.randint(-20,20)
```

```
augmentation = ImageDataGenerator(rotation_range=rotation, shear_range = 0.2,
zoom_range = 0.2, horizontal_flip=True)
sample_img = image.load_img('drive_data/data1a/training/00-damage'+'/'+sample_file)
sample_img_array = image.img_to_array(sample_img)
sample_img_array = sample_img_array.reshape((1,) + sample_img_array.shape)
samples = []
for arr,val in zip(augmentation.flow(sample_img_array, batch_size=1,
save_format='jpg'),range(1)):
    img_save = image.array_to_img(arr[0], scale=False) #scale=False did the trick
    samples.append(img_save)
plt.imshow(samples[0])
```

Original



Data Augmented

So we will be doing data augmentation on all data we have and create folders which consists of original image and generated images in a single folder.

I have created three types of data for each stage we have. Lets see the number of files we have for each stage separately.

Check Damaged or Not:

1. Original Data (1840 Train files, 460 Test files)
2. Original Data + Data Augmentation 1 (3680 Train files, 460 Test files)
3. Original Data + Data Augmentation 2 (3680 Train files, 460 Test files)

Check Front, Rear or Side:

1. Original Data (985 Train files, 179 Test files)
2. Original Data +Data Augmentation 1 (1970 Train files, 179 Test files)
3. Original Data + Data Augmentation 2 (1970 Train files, 179 Test files)

Check Minor, Moderate, Severe:

1. Original Data (979 Train files, 171 Test files)
2. Original Data +Data Augmentation 1 (1958 Train files, 171 Test files)
3. Original Data + Data Augmentation 2 (1958 Train files, 171 Test files)

After training models on above data I will be using YOLO V3 to categorize the image into one of the following using bounding boxes.

1. Glass and Light Broken
2. Car Dent and Scratch
3. Smash

Since these models does not have FC layer, I have used below configuration for all models.

```
#Flatten
        flatten = Flatten(data_format='channels_last',name='Flatten')(vgg16.output)


        #FC layer
        FC1 = Dense(units=512,activation='relu',name='FC1')(flatten)


        #FC layer
        FC2 = Dense(units=256,activation='relu',name='FC2')(FC1)


        #Dropout layer
        droput1 = Dropout(0.5)(FC2)
```

```
        #output layer
         Out = Dense(units=n_classes,activation=output_activation,name='Output')(droput1
```

I have created functions which are generalized and can be used with any pretrained model.

```
#Creating
a
funciton
which
sets
layers to
non
trainable
if
required
         def non_trainable(model):
             for i in range(len(model.layers)):
               model.layers[i].trainable = False
             return model
```

Input data for NN and baseline models.

```
#Data
for
Neural
Networks
        input_shapes = (256,256,3)
        batch_size = 8


        #Classes
        stage1_class_labels = ['damaged','not_damaged']
        stage2_class_labels = ['front','rear','side']
        stage3_class_labels = ['minor','moderate','severe']
```

Baseline Model (Logistic Regression):

Below function gives an array of features predicted using pretrained model. I have created a x_train, y_train, x_test, y_test using above features.

```
def
model_features(model,train_dir,test_dir):
                                                    train_gen = ImageDataGenerator()
                                                    test_gen = ImageDataGenerator()

                                                    train_fd =
                                                train_gen.flow_from_directory(train_dir,target_size =
                                                (256,256),batch_size = 1,shuffle = False)
                                                    test_fd =
                                                test_gen.flow_from_directory(test_dir,target_size =
                                                (256,256),batch_size = 1,shuffle = False)

                                                    train_features = model.predict(train_fd)
                                                    test_features = model.predict(test_fd)
```

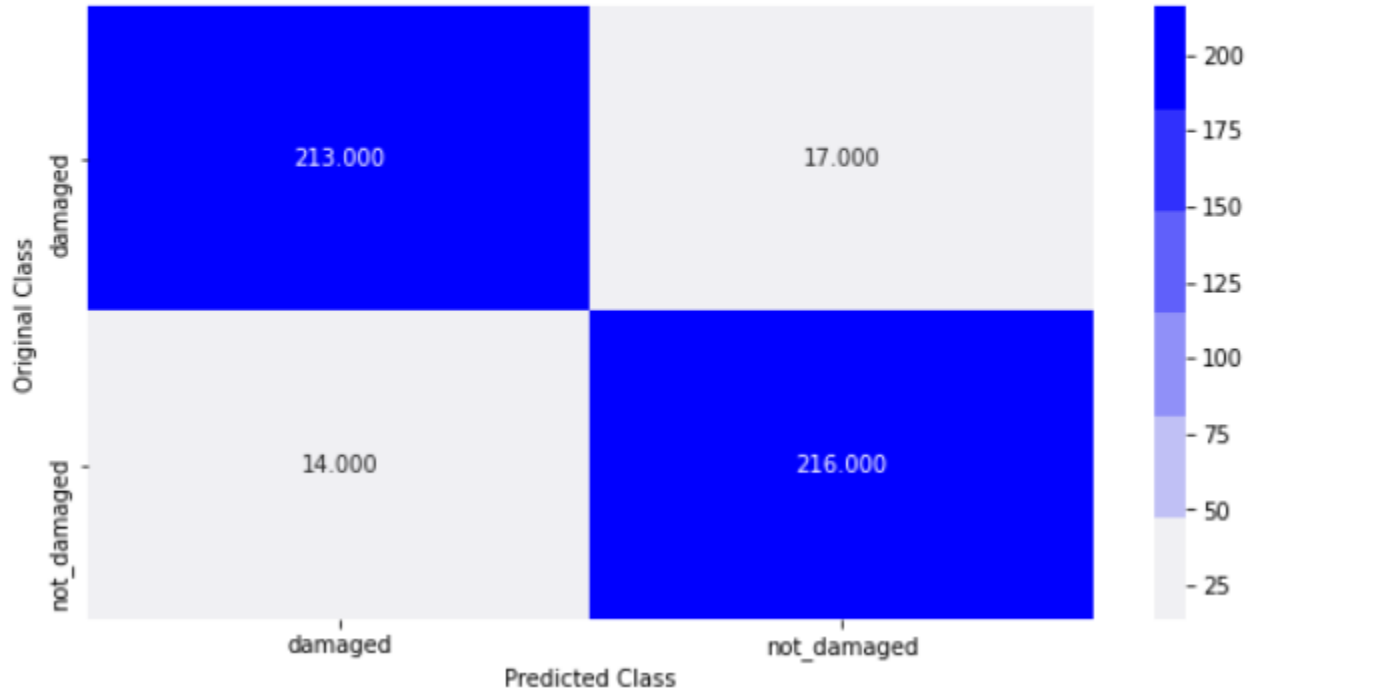
Using above code for hyperparameter tuning of Logistic Regression. Using best alpha from above to create model and saving model to disk.

```
#Final model
using best
paramaters
from
hyperparameter
tuning.
              base_final=LogisticRegression(penalty='l2',C=alpha[best_alpha])
              base_final.fit(x_train,y_train)


              #saving the model
              save_path = 'vgg16/vgg16_stage1_baseline.sav'
              pickle.dump(base_final,open(save_path,'wb'))


              #Predcitions and visualization of Confusion matrix
              predict_y = base_final.predict(x_test)
              print ('Accuracy on test data for final baseline model',accuracy_score(y_test,
              predict_y))
              precision,recall = compute_precision_recall(y_test,predict_y)
              print ('Precision on test data for final baseline model',precision)
              print ('Recall on test data for final baseline model',recall)
```

Confusion Matrix:

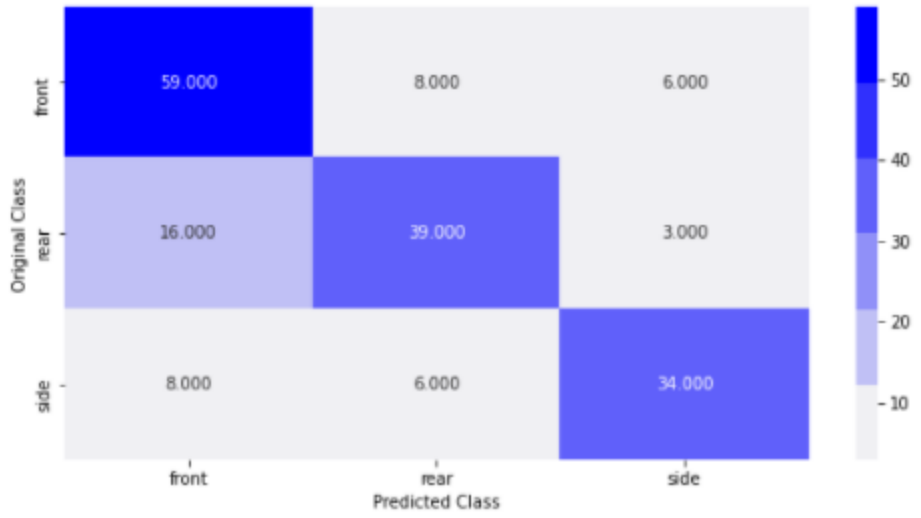
For binary values Stage 1(Check Damaged or Not)

```
Accuracy on test data for final baseline model 0.9326086956521739
Precision on test data for final baseline model 0.9383259911894273
Recall on test data for final baseline model 0.9260869565217391
---------------------------------------------------- Confusion matrix ------------
```

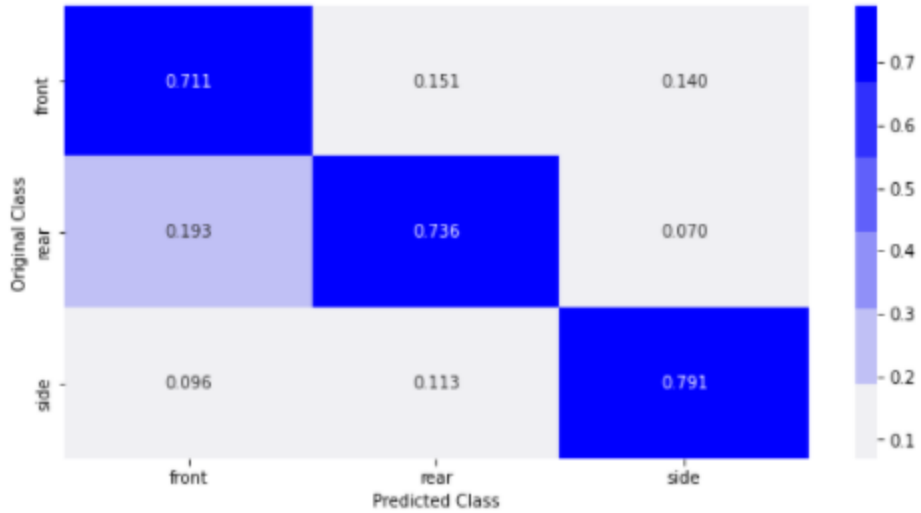


For multiclass I will be showing Confusion, Precision and Recall matrix and compute average precision and Recall.

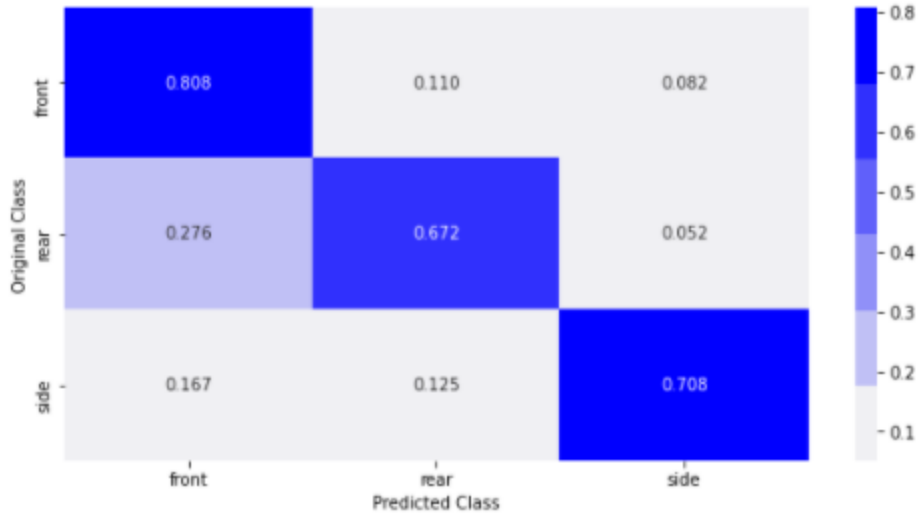
```
Accuracy on test data for final baseline model 0.7374301675977654
Average Precision is 0.7457967015054514
Average Recall is 0.7296554348396578
---------------------------------------------------- Confusion Matrix --------
```




```
---------------------------------------------------- Precision Matrix --------
```




```
---------------------------------------------------- Recall Matrix -----------
```



Code for Baseline models:

Neural Networks:

I have created below functions which are Training on FC layers only and Training on all layers.

```
def
create_model(n_classes,output_acti
vation):
                                    os.environ['PYTHONHASHSEED'] = '0'

                                    tf.keras.backend.clear_session()



                                    ## Set the random seed values to regenerate the model.

                                    np.random.seed(0)

                                    rn.seed(0)



                                    #Input layer

                                    input_layer =
                                Input(shape=(256,256,3),name='Input_Layer')



                                    #Adding pretrained model

                                    vgg16 = applications.VGG16(include_top = False,weights =
                                'imagenet',input_tensor = input_layer)



                                    #Flatten
```

```
    flatten =
Flatten(data_format='channels_last',name='Flatten')(vgg16.ou
tput)



    #FC layer

    FC1 =
Dense(units=512,activation='relu',name='FC1')(flatten)



    #FC layer

    FC2 = Dense(units=256,activation='relu',name='FC2')(FC1)



    #Dropout layer

    droput1 = Dropout(0.5)(FC2)



    #output layer

    Out =
Dense(units=n_classes,activation=output_activation,name='Out
put')(droput1)



    #Creating the Model

    model = Model(inputs=input_layer,outputs=Out)



    return model
```

After creating model using above functions I have compiled using Binary cross entropy loss for Stage 1 and Categorical Cross entropy loss for Stage 2 and Stage 3. Used SGD optimizer and accuracy as metric.

```
model =
create_model(1,'sigmoi
d')
```

```
#Compiling the model

model.compile(loss = 'binary_crossentropy', optimizer =
optimizers.SGD(lr=0.00001, momentum=0.9), metrics=["accuracy"])
```

```
#Model saving based on validation accuracy score

filepath="vgg16/vgg16_stage1_fc-{val_accuracy:.3f}.hdf5"

checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',
verbose=1, save_best_only=True, mode='auto')
```

```
traindatagen = ImageDataGenerator()

testdatagen = ImageDataGenerator()
```

```
train_data_dir = 'data_augmentation_1/data_1/train'

test_data_dir = 'data_augmentation_1/data_1/test'
```

```
train_generator =
traindatagen.flow_from_directory(train_data_dir,target_size =
(256,256),batch_size = batch_size, class_mode = "binary")
```

```
test_generator =
testdatagen.flow_from_directory(test_data_dir,target_size =
(256,256),batch_size = batch_size, class_mode = "binary")



n_validation_steps = 460/batch_size

n_steps_epoch = 3680/batch_size







model.fit(train_generator ,validation_data =
test_generator,validation_steps =
n_validation_steps,steps_per_epoch=n_steps_epoch,epochs=50,callbacks=[ch
eckpoint])



for i in ['loss','accuracy']:

  plot_metrics(model,i)


x_test = testdatagen.flow_from_directory(test_data_dir,target_size =
(256,256),batch_size = 1, class_mode = "binary",shuffle = False)

best_model = load_model('vgg16/vgg16_stage1_fc-0.930.hdf5')

y_pred = best_model.predict(x_test)

y_true = target_feat(x_test)

y_predicted = []

for i in y_pred:

  if i <= .5:

    y_predicted.append('damaged')

  elif i > .5:
```

```
        y_predicted.append('not_damaged')



precision,recall = compute_precision_recall(y_true,y_predicted)

print ('Precision on test data for final baseline model',precision)

print ('Recall on test data for final baseline model',recall)

binary_confusion_matrix(y_true,y_predicted,stage1_class_labels)
```

6.Model Comparison and Selection:

Original Data:

Vgg16

	Model	Accuracy	Precision	Recall
0	Vgg16_stage1_baseline	0.932	0.916	0.952
1	Vgg16_stage2_baseline	0.698	0.709	0.689
2	Vgg16_stage3_baseline	0.626	0.626	0.616
3	Vgg16_stage1 FC	0.930	0.913	0.952
4	Vgg16_stage2 FC	0.687	0.698	0.677
5	Vgg16_stage3 FC	0.649	0.640	0.646
6	Vgg16_stage1 all	0.943	0.969	0.961
7	Vgg16_stage2 all	0.631	0.646	0.616
8	Vgg16_stage3 all	0.538	0.525	0.524

Vgg19

	Model	Accuracy	Precision	Recall
0	Vgg19_stage1_baseline	0.915	0.903	0.930
1	Vgg19_stage2_baseline	0.732	0.746	0.730
2	Vgg19_stage3_baseline	0.693	0.698	0.691
3	Vgg19_stage3_baseline	0.614	0.605	0.612
4	Vgg19_stage1 FC	0.928	0.938	0.917
5	Vgg19_stage2 FC	0.721	0.732	0.710
6	Vgg19_stage3 FC	0.637	0.670	0.636
7	Vgg19_stage1 all	0.928	0.919	0.939
8	Vgg19_stage2 all	0.564	0.553	0.542
9	Vgg19_stage3 all	0.637	0.622	0.625

Densenet

	Model	Accuracy	Precision	Recall
0	Densenet_stage1_baseline	0.856	0.856	0.856
1	Densenet_stage2_baseline	0.508	0.485	0.491
2	Densenet_stage3_baseline	0.503	0.494	0.497
3	Densenet_stage1 FC	0.883	0.873	0.896
4	Densenet_stage2 FC	0.609	0.603	0.597
5	Densenet_stage3 FC	0.538	0.526	0.525
6	Densenet_stage1 all	0.963	0.949	0.978
7	Densenet_stage2 all	0.765	0.768	0.744
8	Densenet_stage3 all	0.661	0.657	0.642

Resnet

	Model	Accuracy	Precision	Recall
0	Resnet_stage1_baseline	0.926	0.922	0.930
1	Resnet_stage2_baseline	0.749	0.777	0.736
2	Resnet_stage3_baseline	0.672	0.684	0.664
3	Resnet_stage1 FC	0.939	0.917	0.965
4	Resnet_stage2 FC	0.698	0.715	0.693
5	Resnet_stage3 FC	0.649	0.642	0.633
6	Resnet_stage1 all	0.950	0.936	0.965
7	Resnet_stage2 all	0.721	0.734	0.705
8	Resnet_stage3 all	0.678	0.685	0.673

1. Stage 1 (Check Damaged or Not):

   For Stage 1 we can see that Densenet trained on all layers is performing better than other models. We have got an accuracy of 96.3 % on this model, precision of 94.9% and recall of 97.8%

2. Stage 2 (Damage Localization):

   For Stage 2 we can see that Densenet trained on all layers is performing better than other models. We have got an accuracy of 76.5 % on this model, precision of 76.8% and recall of 74.4%

3. Stage 3 (Damage Severity)

   For Stage 2 we can see that Resnet trained on all layers is performing better than other models. We have got an accuracy of 67.8 % on this model, precision of 68.5% and recall of 67.3%

Original Data + Data Augmentation 1:

Vgg16

	Model	Accuracy	Precision	Recall
0	Vgg16_stage1_baseline	0.933	0.938	0.926
1	Vgg16_stage2_baseline	0.737	0.746	0.730
2	Vgg16_stage3_baseline	0.661	0.650	0.652
3	Vgg16_stage1 FC	0.930	0.919	0.943
4	Vgg16_stage2 FC	0.709	0.717	0.707
5	Vgg16_stage3 FC	0.643	0.650	0.637
6	Vgg16_stage1 all	0.939	0.935	0.943
7	Vgg16_stage2 all	0.693	0.712	0.678
8	Vgg16_stage3 all	0.608	0.608	0.587

Vgg19

	Model	Accuracy	Precision	Recall
0	Vgg19_stage1_baseline	0.930	0.927	0.935
1	Vgg19_stage2_baseline	0.732	0.746	0.730
2	Vgg19_stage3_baseline	0.656	0.641	0.646
3	Vgg19_stage1 FC	0.926	0.930	0.922
4	Vgg19_stage2 FC	0.693	0.710	0.678
5	Vgg19_stage3 FC	0.637	0.633	0.626
6	Vgg19_stage1 all	0.946	0.944	0.948
7	Vgg19_stage2 all	0.693	0.712	0.678
8	Vgg19_stage3 all	0.608	0.608	0.587

Densenet

	Model	Accuracy	Precision	Recall
0	Densenet_stage1_baseline	0.880	0.875	0.887
1	Densenet_stage2_baseline	0.598	0.590	0.579
2	Densenet_stage3_baseline	0.550	0.540	0.545
3	Densenet_stage1 FC	0.893	0.888	0.900
4	Densenet_stage2 FC	0.581	0.583	0.564
5	Densenet_stage3 FC	0.573	0.578	0.569
6	Densenet_stage1 all	0.959	0.949	0.970
7	Densenet_stage2 all	0.804	0.807	0.789
8	Densenet_stage3 all	0.696	0.692	0.685

Resnet

	Model	Accuracy	Precision	Recall
0	Resnet_stage1_baseline	0.930	0.923	0.940
1	Resnet_stage2_baseline	0.782	0.791	0.779
2	Resnet_stage3_baseline	0.637	0.633	0.621
3	Resnet_stage1 FC	0.933	0.938	0.926
4	Resnet_stage2 FC	0.771	0.788	0.759
5	Resnet_stage3 FC	0.643	0.628	0.630
6	Resnet_stage1 all	0.961	0.945	0.978
7	Resnet_stage2 all	0.749	0.762	0.740
8	Resnet_stage3 all	0.643	0.635	0.633

1. Stage 1 (Check Damaged or Not):

   For Stage 1 we can see that Resnet and Densenet trained on all layers are performing better than other models. We have got an accuracy of 96.1 % on resnet model but precison is lower than the densenet model. For Densenet model accuracy is 95.9%, precision of 94.9% and recall of 97.8%.

2. Stage 2 (Damage Localization):

   For Stage 2 we can see that Densenet trained on all layers is performing better than other models. We have got an accuracy of 80.4 % on this model, precision of 80.7% and recall of 78.9%.

3. Stage 3 (Damage Severity)

   For Stage 2 we can see that Resnet trained on all layers is performing better than other models. We have got an accuracy of 69.6 % on this model, precision of 69.2% and recall of 68.5%.

Original Data + Data Augmentation 2:

Vgg16

	Model	Accuracy	Precision	Recall
0	Vgg16_stage1_baseline	0.922	0.915	0.930
1	Vgg16_stage2_baseline	0.743	0.740	0.733
2	Vgg16_stage3_baseline	0.661	0.650	0.651
3	Vgg16_stage1 FC	0.926	0.908	0.948
4	Vgg16_stage2 FC	0.726	0.732	0.723
5	Vgg16_stage3 FC	0.655	0.654	0.641
6	Vgg16_stage1 all	0.935	0.954	0.913
7	Vgg16_stage2 all	0.665	0.667	0.669
8	Vgg16_stage3 all	0.608	0.607	0.586

Vgg19

	Model	Accuracy	Precision	Recall
0	Vgg19_stage1_baseline	0.915	0.909	0.922
1	Vgg19_stage2_baseline	0.704	0.711	0.702
2	Vgg19_stage3_baseline	0.620	0.616	0.612
3	Vgg19_stage1 FC	0.922	0.929	0.913
4	Vgg19_stage2 FC	0.687	0.702	0.683
5	Vgg19_stage3 FC	0.655	0.668	0.643
6	Vgg19_stage1 all	0.943	0.936	0.952
7	Vgg19_stage2 all	0.637	0.630	0.631
8	Vgg19_stage3 all	0.655	0.636	0.631

Densenet

	Model	Accuracy	Precision	Recall
0	Densenet_stage1_baseline	0.867	0.875	0.856
1	Densenet_stage2_baseline	0.553	0.563	0.539
2	Densenet_stage3_baseline	0.544	0.538	0.532
3	Densenet_stage1 FC	0.878	0.868	0.891
4	Densenet_stage2 FC	0.598	0.616	0.575
5	Densenet_stage3 FC	0.544	0.524	0.519
6	Densenet_stage1 all	0.954	0.941	0.969
7	Densenet_stage2 all	0.777	0.778	0.766
8	Densenet_stage3 all	0.684	0.686	0.680

Resnet

	Model	Accuracy	Precision	Recall
0	Resnet_stage1_baseline	0.930	0.923	0.939
1	Resnet_stage2_baseline	0.760	0.774	0.776
2	Resnet_stage3_baseline	0.649	0.649	0.635
3	Resnet_stage1 FC	0.943	0.928	0.961
4	Resnet_stage2 FC	0.721	0.741	0.715
5	Resnet_stage3 FC	0.684	0.685	0.678
6	Resnet_stage1 all	0.954	0.937	0.974
7	Resnet_stage2 all	0.732	0.768	0.720
8	Resnet_stage3 all	0.632	0.628	0.626

1. Stage 1 (Check Damaged or Not):

   For Stage 1 we can see that Resnet and Densenet trained on all layers are performing better than other models. We have got an accuracy of 95.4 % on resnet model but precison is lower than the densenet model. For Densenet model accuracy is 95.4%, precision of 94.1% and recall of 96.9%.

2. Stage 2 (Damage Localization):

   For Stage 2 we can see that Densenet trained on all layers is performing better than other models. We have got an accuracy of 77.7 % on this model, precision of 77.8% and recall of 766.6%

3. Stage 3 (Damage Severity)

   For Stage 2 we can see that Resnet trained on all layers is performing better than other models. We have got an accuracy of 68.4 % on this model, precision of 68.6% and recall of 68.0%

Conclusion:

1. From above all models we can see that the Densenet (Trained on all layers) model which is trained on the Original data with augmentation 1 is performing best in Stage 2 and Stage 3.
2. For Stage 1 densenet (Trained on all layers) trained on Orginal data with Augmentation 1 is very close to the model trained on Orginal data and precision of both models are same. So we can use the Densenet (Trained on all layers) trained on Orginal data with Augmentation 1 instead of model trained on original data.

So Densenet (Trained on all layers) with Augmentation 1 worked best for all 3 stages.