Full Stack Milestone Project

Car Damage Detection App

[Github]

Group PDSDS 4:
Hui Wan - 101384469
Lee Ying Nam - 101386434
Huynh Kha Man - 101374770
Tushar Kapoor-101212308

1. Introduction

Because of the increasing expense of evaluating car loss by humans, our car insurance company has to develop an AI application that is available to recognize and detect the car damages automatically in a web application.

Due to the pandemic, the human force is demanding and not all of our staff can work in the office to check the damaged cars of the clients. In order to solve this program, we are trying to develop an AI web application where the client can take a picture of their damaged car and our AI will detect and categorize the type of damage.

Also, the insurance companies take a long time to process claims, they take time to process even small damage to the car by human judgment. As traveling may not be convenient for all clients, just getting information from the description of the damaged car may not be accurate to estimate the claim. In order to solve this problem, the AI can process a faster and more accurate claim.

The proposed application will help to reduce the cost of evaluating the loss manually as our application automatically detects damages to the car. The application will help ease the process of claiming insurance from insurance companies as the output of the application might be used to estimate the loss.

The application will also help insurance companies to provide claims faster to their users as the damage will be automatically detected.

2. Dataset

There will be 2 dataset used in this project.

The first dataset is from kaggle: https://www.kaggle.com/anujms/car-damage-detection. This dataset consists of Train and Validation which each folder has Damage cars pics and whole car pics and there are a total 2300 images present in both train and validation combined. There will be mainly 3 types of data. The first type is for classifying whether the car is damaged or not. There are Training and Test folders of Car damaged, not damaged images label pictures. The second type is for classifying where the damaged part is. There are Training and Test folders of Damage on Front, Rear, Side lebel pictures. The third one is for classifying the severity of the damage. There are Training and Test folders of Damage severity Minor, Moderate, Severe.

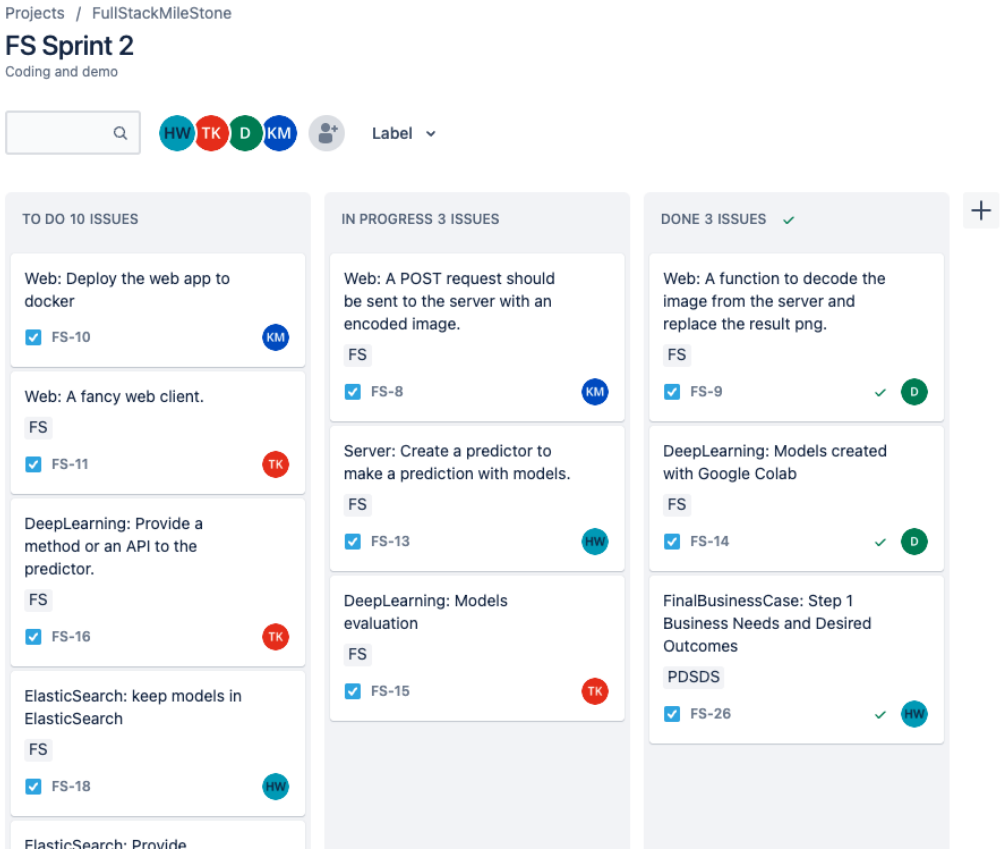
Another dataset is from roboflow: https://universe.roboflow.com/yash-sawant/car-dent-detection2. This dataset consists of train, test and validation files with a total of 3746 images. All the images are annotated which can be used for object detection.

For the dataset responsible for classifying the car damage, we synthetically enlarged the dataset using Augmentation of Random rotation between -20 and 20 with Horizontal flip.

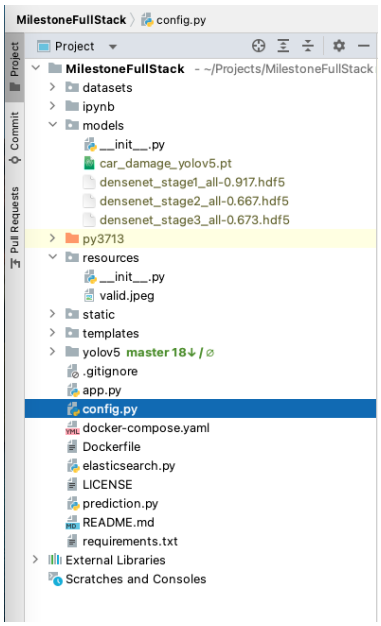
```
#Getting a random rotation between -20 and 20
rotation = random.randint(-20,20)


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

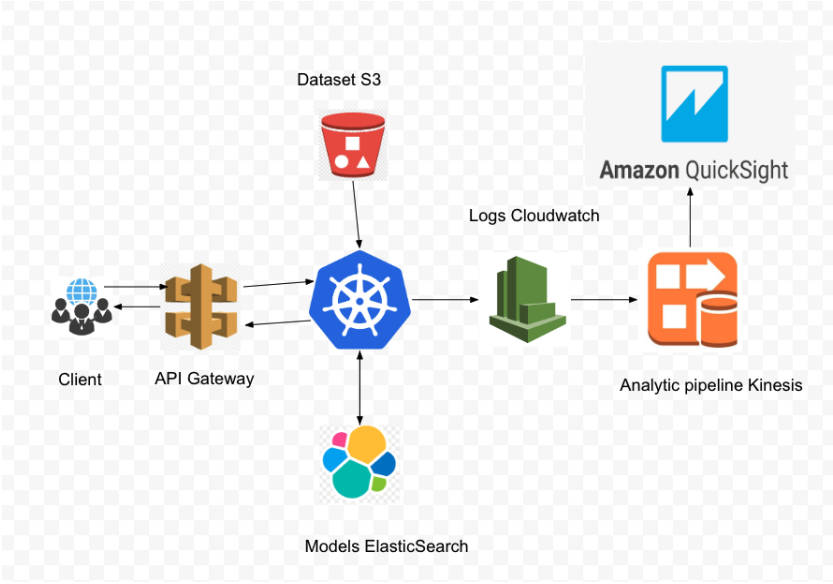
3. Agile

Here is our project report from Agile.
This is a project with 2 sprints. It contains 2 weeks in every sprint.



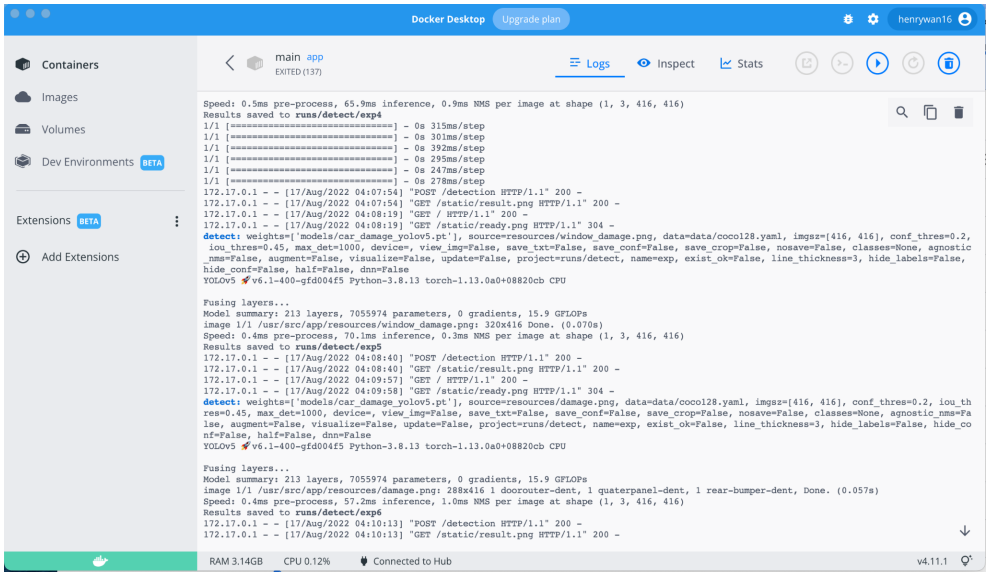
4. Project Outline





For the front end, we will work with Angular which is a famous framework from Google. An application for users to upload images will be created. API Gateway will provide multiple endpoints which will be accessed by users and development teams. Our application will be able to run on docker locally and then be deployed to AWS EKS which is scalable and stable. The application running on the server can train better models and update models in the ElasticSearch. Those updated models will be used for prediction. All the logs will be captured and kept in AWS CloudWatch. There will be an analytic pipeline connecting to the CloudWatch. We'd like to use AWS Kinesis to hold our data pipeline. Those analytic data can be shown in QuickSight which is a tool like a Tableau running on the cloud.

We would like to deploy our backend to AWS EKS with Kubernetes. We will have a VPC and a public subnet for our application. Then the EKS will be running on multiple availability zones to guarantee scalability and fault tolerance. If one of the availability doesn't work, our application can run successfully in another zone.



Here are our Flask APIs endpoints:
GET /index #Access the web application
POST /training # Train the model
POST /prediction # Get a report for the car damage including severity, general description and a marked image.
POST /severity # Severe evaluation
POST /description # Car damage description

5. Models Training

There will be several models trained and used in this project. Three models for classification and one model for object detection.

- We used Densenet and Resnet model pre trained on imagenet dataset
- The models were trained in 3 stages
- Stage 1 is for detecting the damage
- Stage 2 is for damage localization
- Stage 3 is for calculating the extent of damage

Stage 1

```
[ ] model = create_model(1,'sigmoid')
```

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

```
[ ] model.summary()
```

```
    #Model saving based on validation accuracy score
    filepath="densenet/densenet_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 = '../input/jjjjjjjj/content/data_augmentation_1/data_1/train'
    test_data_dir = '../input/jjjjjjjj/content/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
```

Stage 2

```
[ ] model = create_model(3,'softmax')
```

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

```
[ ] model.summary()
```

```
[ ] #Model saving based on validation accuracy score
    filepath="densenet/densenet_stage2_all-{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 = '../input/jjjjjjjj/content/data_augmentation_1/data_2/train'
    test_data_dir = '../input/jjjjjjjj/content/data_augmentation_1/data_2/test'

    train_generator = traindatagen.flow_from_directory(train_data_dir,target_size = (256,256),batch_size = batch_size)
    test_generator = testdatagen.flow_from_directory(test_data_dir,target_size = (256,256),batch_size = batch_size)

    n_validation_steps = 179/batch_size
    n_steps_epoch = 1970/batch_size
```

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

6. Models Evaluation

Models Evaluation(Original Data)

	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

Models Evaluation (With Augmentation)

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

7. Result

Here is the interface of the webpage that we allow users to upload the images. We have 2 buttons: one to select an image locally to upload and another one to get the detection output.

Click on the "Choose File" button to upload a car damage image or video:

Car Damage Detection

Choose File valid.jpeg ShowDamage

Ready Go

8. Conclusion

To conclude, we have achieved something good by the end of the project. First, our model accurately detects the exterior damage part of the car and classifies its level. As we know, it goes through 3 different stages to make the final result, thus making helpful references for users and insurance companies to use. Moreover, it's possible to be able to handle a high amount of client requests and return responses fastly.
We think that the project will have more room for improvements as well as challenges that we need to solve.