CAR DAMAGE INTENSITY DATASET

Author: Suman Paul Choudhury , IIT GUWAHATI

The dataset contains 3 folders containing images which describes the intensity of damages as minor , moderate and severe accordingly. The task was to analyse and explore the different feature extraction techniques along with classifiers to check upon the performance of the system. Hereby I worked with 3 different types of approaches to tackle the task

APPROACH 1:

Using Feature Extraction:

Features are the information that are extracted from an image. When deciding upon the features, we can possibly think upon Colour, Texture and shape as the primary ones. However, examples like "Sunflower" or "Daffodil" can be similar in terms of colours. So, we need different descriptors to describe our image more effectively.

Global Feature Vectors: These are the features that quantifies an image globally. These don't have the concept of interest points and thus, takes in the entire image for processing.
Colour: Colour Histogram, Colour statistics
Shape: Hu Moments, Zernike Moments
Texture: Haralick Features, LBP Features

For global feature vectors, we just concatenate each feature vector to form a single global feature vector.
So here, I have used Colour Histogram, Hu Moments and Haralick features (GLCM) as global features vectors for the task.

HuMoments compute the moments of the image and convert it to a vector using flatten (). While Haralick Texture is used to quantify an image based on texture. Reason behind using Haralick features is the Gray Level Co-occurrence Matrix or GLCM. Gray Level Co-occurrence matrix (GLCM) uses adjacency concept in images. The basic idea is that it looks for pairs of adjacent pixel values that occur in an image and keeps recording it over the entire image.

TO RUN THE PROGRAM **(Folder name : task_vision)**

1. **python featureExtractionDamage.py**
2. **python train_test_carDamage.py**

(Note: Please change the path accordingly for the folders in the python file)

Step1:

Since the images are matrices we need an efficient way to save our features locally. The function "**featureExtractionDamage.py**" extract three global features from the images, concatenates the global features into a single global feature and save the feature along with its label in a HDF5 format.

For each of the training label name, we iterate through the corresponding folder (minor , moderate and severe) to get all the images inside it. For each image that we iterate, we first resize the image into a fixed size. Then, we extract the three global features and concatenate these three features using NumPy's np.hstack() function. We keep track of the feature with its label using those two lists we created above - ***labels*** and ***global_feature***

Step2:
After extracting features and concatenating it, we need to save this data locally. Before saving this data, we use something called ***LabelEncoder()*** to encode our labels in a proper format. This is to make sure that the labels are represented as unique numbers. As we have used different global features, one feature might dominate the other with respect to it's value. In such scenarios, it is better to normalize everything within a range (say 0-1). Thus, we normalize the features using ***MinMaxScaler()*** function. After doing these two steps, we use h5py to save our features and labels locally in .h5 file format

Run the program as : **python featureExtractionDamage.py**
(Note: Please change the path accordingly for the folders)

There are 532 columns in the global feature vector which tells us that when we concatenate color histogram, haralick texture and hu moments, we get a single row with 532 columns. So, for 979 images, we get a feature vector of size (979, 532). Also, you could see that the target labels are encoded as integer values in the range (0-2) denoting the 3 classes of car dataset.

Step3:
After extracting the global features, its time to train the classifiers. To create the machine learning model, we take the help of scikit-learn. We will choose Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbours, Decision Trees, Random Forests, Gaussian Naive Bayes, Support Vector Machine , MLP Neural Network as our machine learning models.

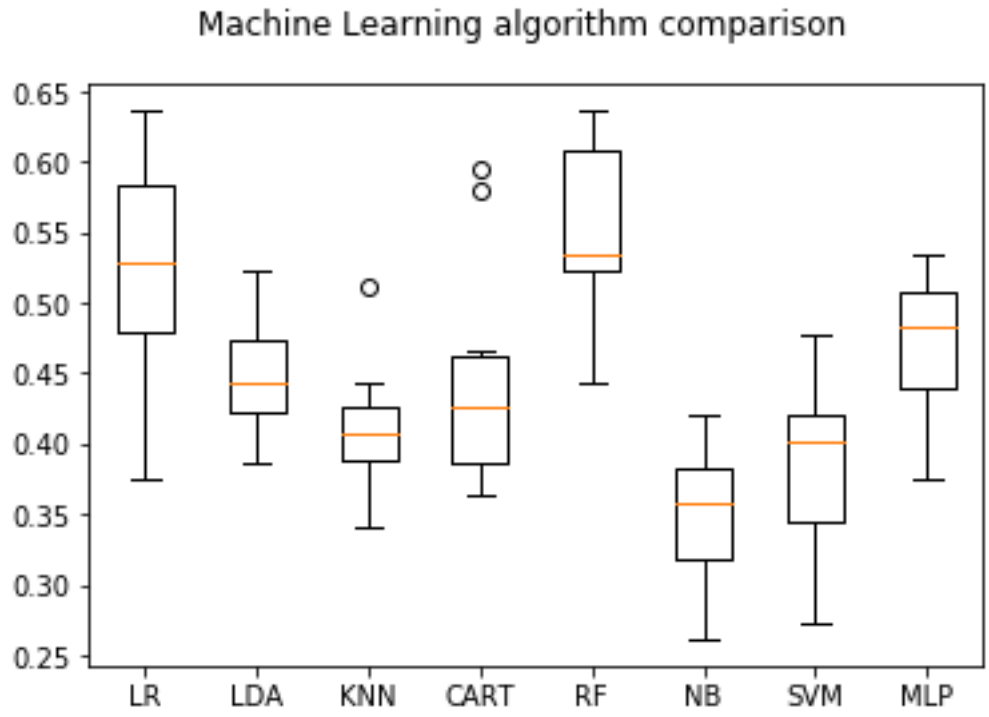
I have used *train_test_split* function provided by scikit-learn to split our training dataset into train_data and test_data. I have also use a technique called K-Fold Cross Validation, a model-validation technique which is the best way to predict ML model's accuracy. I have imported all the necessary libraries to work with and create a models list. This list will have all our machine learning models that will get trained with our locally stored features. Our target will be to train our model with more data so that our model generalizes well. So, we keep test_size variable to be in the range (0.10 - 0.30).

Finally, we train each of our machine learning model and check the cross-validation results. Here, we have used only our train_data.

Run the program as : **python train_test_carDamage.py**
(Note: Please change the path accordingly for the folders)

Results:

LR: 0.522089 (0.074325) → Logistic Regression
LDA: 0.450549 (0.043395) → Linear Discriminant Analysis
KNN: 0.408631 (0.045475) → K – Nearest Neighbour
CART: 0.445914 (0.078079) → Decision Tree Classifier
RF: 0.552707 (0.057119) → Random Forest Classifier
NB: 0.350804 (0.047380) → Naïve Bayes Classifier
SVM: 0.384780 (0.059284) → Support Vector Machine Classifier
MLP: 0.466547 (0.051940) → MultiLayer Perceptron Classifier



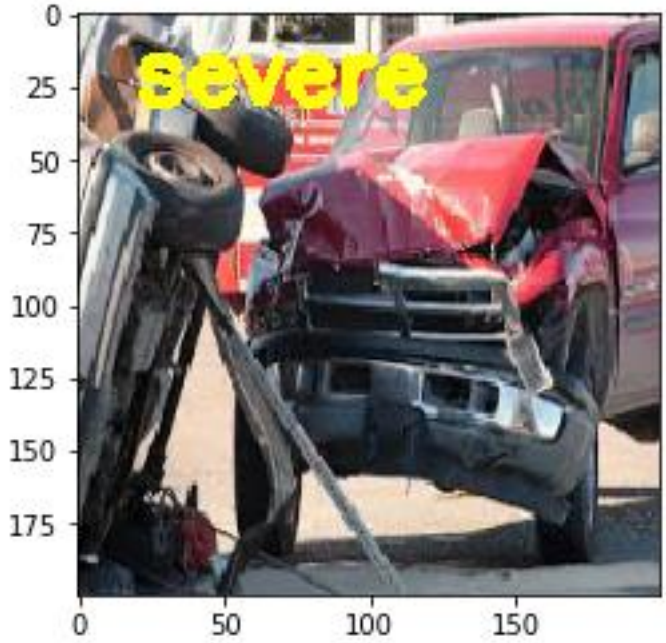
As we can see RandomForest Classifier performs best among all the classifiers but the accuracy is not that high.

Analysis/Reason: This is mainly due to the number of images we use per class. **We need large amounts of data to get better accuracy.** For example, for a single class, we atleast need around 500-1000 images which is indeed a time-consuming task.

Step 4: Finally , we quickly build a Random Forest model, train it with the training data and test it on some unseen car damage images.

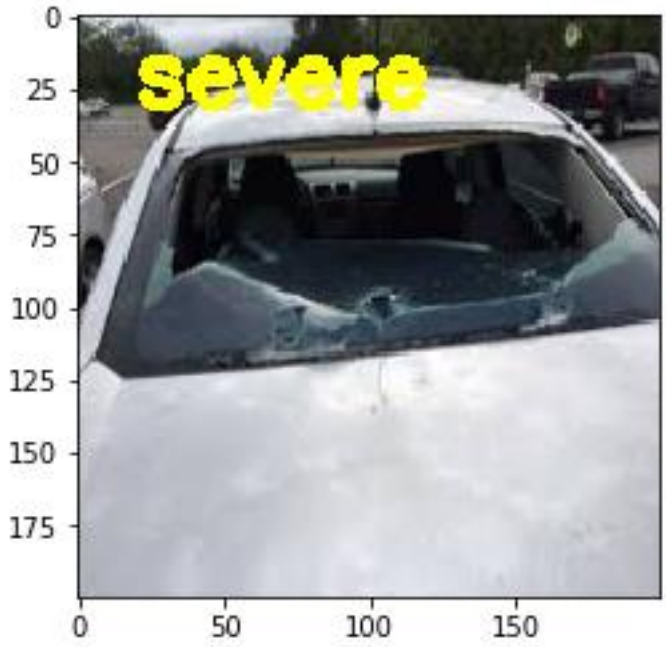
Some of the test images got correctly classified as shown in the picture below:

True Label : Severe , Predicted : Severe

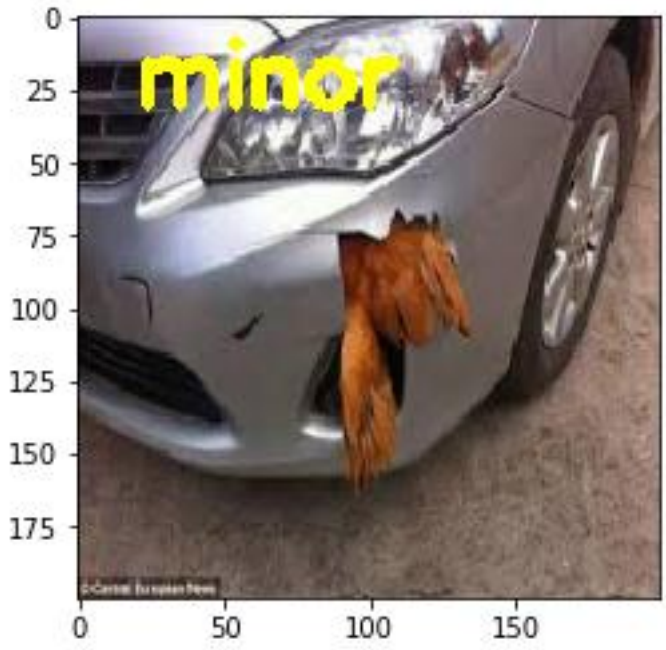


True Label : Severe , Predicted : Severe

While a majority of the images got misclassified also as shown in the picture below



True Label : Moderate , Predicted : Severe



True Label: Moderate , Predicted : Minor

Analysis :

The idea of moderate, minor or severe damages totally depends/varies upon human perception. In the above image where the glass shields seem to be broken appears to be a severe damage while the ground truth is given as moderate.

So Local features like SIFT (Scale Invariant Feature Transform) , SURF (Speeded Up Robust Features), SURF can be a very good addition to global features while extracting feature vectors which is a further exploration area of feature extraction but I could not explore it owing to time constraints.

IMPROVING CLASSIFIER ACCURACY

There are multiple techniques to achieve better accuracy

1. Gather more data for each class. (500-1000) images per class.

2. We can use Data Augmentation to generate more images per class.

3. Global features along with local features such as SIFT, SURF or DENSE could be used along with Bag of Visual Words (BOVW) technique.

4. Local features alone could be tested with BOVW technique.

5. Convolutional Neural Networks - State-of-the-art models when it comes to Image Classification and Object Recognition.

APPROACH 2:

Using Convolutional Neural Networks (CNN).

The reason for using CNN is since we are looking for different feature extraction techniques. CNN provides that leverage to focus on different features by using different filters of varying sizes such that all the features such as low level features (edge, corners), mid-level features (faces , blobs etc) or high level features such as (object detection etc.) are captured with respect to different stacks of convolutional layers. However, a perfect CNN architecture totally depends on the proper initialisation of weights of the neural network.

I Tried to implement a **4 stack layers of CNN** networks for the given task. The entire program is written in **TensorFlow**.

At the initial stages, the weights of filters are initialised with *Xavier Initialisation.*

I have defined certain functions to make the readability of the program better.

initialize_parameters() – function to initialise the parameters

random_mini_batches() – function to divide the data into certain mini batches

convert_to_one_hot() – to convert into one hot encoding

forward_propagation() –function to forward propagate the CNN

compute_cost() – function to compute the cost of the loss function

model() – function to train the model and returning the train accuracy , test accuracy and cost after each 100 epochs

All the images intensities are passed on to the CNN architecture having the following architecture.

CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL →FLATTEN -> FULLYCONNECTED -> FULLYCONNECTED

TO RUN THE PROGRAM SEQUENTIALLY AS: (Folder Name: task_vision)

python DatagenFinal.py (used to generate the dataset required for CNN)

cnnMainCarDetection.py (used to run the entire CNN model and find accuracies)

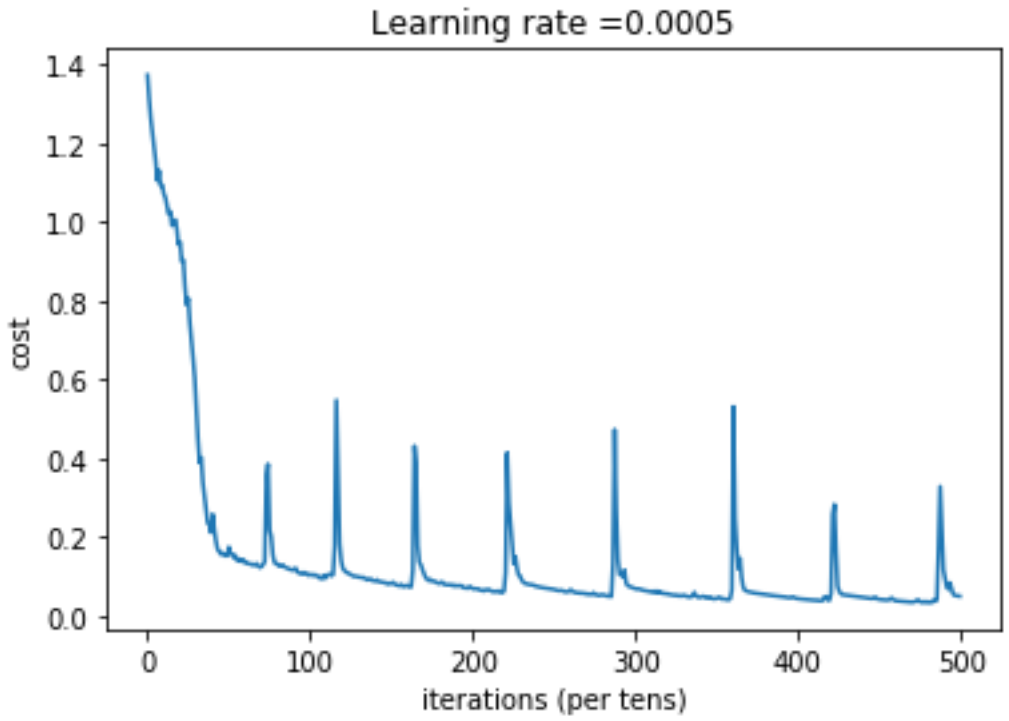
(Note: Please change the path accordingly for the folders in the python file)

The model is run for 1000 epochs with learning rate as 0.0005 , keeping the mini batch size as 32.

The results are as follows:

Train Accuracy: 1.0

Test Accuracy: 0.4502924



Analysis:

Since CNN requires huge amount of data for its training and as we can see from the training and testing dataset accuracy that the model is overfitting. That implies that the model is not generalising well. Thus the model accuracy will increase with the following suggestions

1. **Increase in the number of images per classes**
2. **Try for Dropouts , L1 or L2 regularization**
3. **We can also use some pretrained models, custom train it for our task and use it for our classification purposes.**

APPROACH 3:

Using PreTrained models approach (best one)

The pre-trained models we will consider are VGG16, VGG19, Inception-v3, Xception, ResNet50, InceptionResNetv2 and MobileNet. Instead of creating and training deep neural nets from scratch (which takes time and involves many iterations), what if we use the pre-trained weights of these deep neural net architectures (trained on ImageNet dataset) and use it for our own car damage intensity detection.

Instead of making a CNN as a model to classify images, what if we use it as a Feature Extractor by taking the activations available before the last fully connected layer of the network (i.e. *before* the final softmax classifier). These activations will be acting as the feature vector for a machine learning model (classifier) which further learns to classify it. This type of approach is well suited for Image Classification problems, where instead of training a CNN from scratch (which is time-consuming and tedious), a pre-trained CNN could be used as a Feature Extractor - Transfer Learning

5 steps involved in this task

1. Prepare the training dataset with car damage images and its corresponding labels.
2. Specify our own configurations in **conf.json** file.
3. Extract and store features from the last fully connected layers (or intermediate layers) of a pre-trained Deep Neural Net (CNN) using **extractFeatures.py**.
4. Train a Machine Learning model such as Logisitic Regression, Random Forest using this CNN extracted features and labels using ***train.py***.
5. Evaluate the trained model on unseen data and make further optimizations if necessary.

conf.json is the configuration file or the settings file we will be using to provide inputs to our system. This is just a json file which is a key-value pair file format to store data effectively. Open up the file to understand the settings used. The data path needs to be changed in the json file for the program to run properly.

I have decided to use **VGG16** architecture pre-trained on Imagenet including the top layers.

TO RUN THE PROGRAM AS: **(Folder Name: task_vision2)**

1. **python extractFeatures.py** (to extract the features of VGG16 net architecture and use it for our task)
2. **python train.py** (to run the model)

(Note: Please change the path accordingly for the folders in the python file)

Now the model is trained using the features of VGG16 using Random Forest classifier and the accuracies are measured

Run the program as

python train.py (to run the model)

(Note: Please change the path accordingly for the folders)

Results :

Accuracy is 69.05% (Rank 1 accuracy)

The results are saved in **data/output folder** as results.txt

Analysis:

Several other architectures can be explored upon to check for the accuracies. Such architectures are VGG19, Inception-v3, Xception, ResNet50, InceptionResNetv2 and MobileNet can be explored upon which will surely boost up the performance for this task.

Conclusion:

This experiments actually concludes that Pretrained model for CNNs performs better than the conventional feature extraction techniques for any type of complex classification task and also reduce the time consumed to construct a deep neural net from scratch.