Divya Parmar
Deep Learning
Final Project – Individual Report

Introduction

In the automobile and adjacent industries, much of the communication between customers and companies has been redirected online as of recently. In these industries, such as insurance, dealerships, and resale, customers are often required to upload images of their vehicles to the companies' websites. In order to confirm the vehicle matches the description given by the customer, a checkpoint must be established. With the overwhelming number of images being uploaded on a regular basis, this is a job that must be automated to keep up with demand. By adopting our automobile image classification system, companies can save thousands of human hours annually and thus reduce business expenses.

The dataset that we used was sourced via Stanford University and a paper written by Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei titled *3D Object Representation for Fine-Grained Categorization*. It contains 16,185 images of 196 classes of cars and is split into 8,144 training images and 8,041 testing images with each class being split roughly 50-50 between the two subsets. Classes are typically at the level of *Make, Model, Year*, e.g. 2012 Tesla Model S or 2012 BMW M3 coupe.

Description of Individual Work

- EDA
  - I did exploratory data analysis on the class imbalance of our dataset. I found the data was balanced, but we could use more data to help us classify better.
- Model Metrics
  - I wrote a model metric logging framework to compare different models and hyperparameters. Unfortunately, we weren't able to use it as our models took too long to run.
- Other Coding Tasks
  - Adding third model
    - I suggested adding the third model (Inception V3) to our work for additional comparison purposes.
  - Bounding Boxes
    - I wrote the code to use the bounding boxes for image cropping.
- Code Review
  - I reviewed others pre-processing and data augmentation to make sure it achieved its intended aims.
- Report and Presentation
  - I wrote the conclusion for the final report and the presentation PowerPoint.

Description of Portion of My Work

See code section in appendix.

Results

In order to arrive at the final results, the model training required much more GPU and memory. As stated earlier, one of the techniques we used was to preprocess all the images and save them to a numpy file. The result of the numpy file is a list that contains the numpy arrays representing the preprocessed images.This reduced training time, although the .npy files were in large sizes of about 19GB for the train set and 6GB for the test set.

The next set of results obtained were from the data augmentation. After images were resized, they were randomly rotated or flipped and lastly the backgrounds were reduced using the salt pepper contour technique. Below are some image samples of augmented images and background removed images.

The average run time using 3 epochs was 1 hour 47mins training over 42,000+ images.

The best metrics obtained are as follows:

1. Hamming loss : 0.654
2. Accuracy Score : 0.40
3. Cohen Kappa Score : 0.3425
4. F1-macro Average : 0.41
5. Last loss value : 0.941

Although the scores are very low, from our analysis, this is because some classes are almost not recognized and this because the images are not large enough to be resized into 244 by 244 thereby losing some information. Additionally, we had 197 classes to predict, so an accuracy of 0.40 is respectable.

To achieve the metrics above during training, we used the following hyper parameters

1. Activation : Softmax
2. Weights : Imagenet
3. Epochs : 5
4. Batch Size : 64
5. Optimizer : Adam

Model	Parameters	Hamming	Accuracy	Cohen Kappa	F1 score

VGG19	Imagenet Batch:64 Epochs : 5 Adam	0.654	0.40	0.3425	0.41
Inception V3	Imagenet Batch:64 Epochs : 5 Adam	0.719	0.281	0.277	0.27
Custom	Random weights Batch:64 Epochs : 5 Adam	0.970	0.03	0.0252	0.02

Summary and Conclusion

After testing all three models, the VGG19 model had the best results with an accuracy score of 0.40. These results, at first glance, do not seem great, but considering we were dealing with 197 classes

The VGG19 model with pre-trained weights greatly outperformed our custom model with random weights. This demonstrates the value of architecture tested by research scientists and the value of time/computing power spent on pre-training. In addition, using data augmentation and image pre-processing improved our results. This speaks to the power of (1) more data and (2) removing noise from the image so the model can learn from the correct information.

However, we identified various areas for improvement. These included image quality, pipeline optimization, and hyperparameter tuning.

Due to the inconsistent size of the images, smaller images were hardly classified properly. To improve the network significantly, the smaller images will have to have their pixels improved to become much bigger. However, we did not have the time or knowledge to apply image enhancement. Also there is still some noise when the image background is removed. A more precise noise remover will help obtain better results.

We realized late in the process that we could save intermediate files to avoid duplicating pre-processing steps. This cut down our model run time from ~16 hours to under 1 hour. This would have allowed us to test more models. With this more efficient model pipeline, we could have tested more model architectures, optimizers, learning rates, etc. to optimize our final metrics.

Percent of Code

Around 10% of code was copied. That was for cropping images using bounding boxes and the cv2 library.

Reference

EDA Plot

```
import
pandas
as pd
        import os
        import matplotlib.pyplot as plt

        # Get all images, approximately 8000 each in train and test
        SAMPLE_SIZE = 9000

        module_dir = os.path.dirname(__file__)  # Set path to current directory

        # Train Dataset creation
        train_meta_data_file_path = os.path.join(module_dir, 'Dataset/Metadata/train-meta.xlsx')
        train_data = pd.read_excel(train_meta_data_file_path).head(SAMPLE_SIZE)
        train_counts = train_data.groupby(['class']).size()

        # Test Dataset creation
        test_meta_data_file_path = os.path.join(module_dir, 'Dataset/Metadata/test_meta.xlsx')
        test_data = pd.read_excel(test_meta_data_file_path).head(SAMPLE_SIZE)
        test_counts = test_data.groupby(['class']).size()


        # Change to output directory
        os.chdir("../Images")

        # Train plot
```

```
train_counts.plot(kind='bar',use_index=True)
plt.xlabel("Class Label")
plt.ylabel("Sample Count")
plt.title("Training Sample by Class")
plt.tick_params(
   axis='x',         # changes apply to the x-axis
   which='both',     # both major and minor ticks are affected
   bottom=False,     # ticks along the bottom edge are off
   top=False,        # ticks along the top edge are off
   labelbottom=False) # labels along the bottom edge are off
plt.savefig('train_class_distribution.jpeg', dpi=100)
plt.show()

#Test plot
test_counts.plot(kind='bar',use_index=True)
plt.xlabel("Class Label")
plt.ylabel("Sample Count")
plt.title("Test Sample by Class")
plt.tick_params(
   axis='x',         # changes apply to the x-axis
   which='both',     # both major and minor ticks are affected
   bottom=False,     # ticks along the bottom edge are off
   top=False,        # ticks along the top edge are off
   labelbottom=False) # labels along the bottom edge are off
plt.savefig('test_class_distribution.jpeg', dpi=100)
```

Image cropping

```
try:
        y1 = data[1]
        y2 = data[3]
        x1 = data[0]
        x2 = data[2]
        image = cv2.imread(image_file_path, cv2.IMREAD_COLOR)
        cropped_image = image[y1:y2, x1:x2]
```