Bradley Reardon
Deep Learning
Final Project – Individual Report

Introduction

As technology advances, we continue to find labors that can be automated and in return free up humans to work on things that cannot be automated. With the power of machine learning and computer vision, the classification of images is one of those labors.

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.

Individual Contribution

- Code:
  - Wrote Model.py script. Had help from Salim after for bug fixes and changes to parameters
  - Implemented data augmentation into DataPreprocessing.py file
  - Sourced code for BackgroundRemoval.py and worked through it to fix bugs from original code and make it work for out preprocessing script. Salim helped with bug fixes
- Presentation:
  - Created powerpoint and added intro and model architecture slides.
  - Presented on model architecture.
- Group Final Report:
  - Intro, model architecture, conclusion

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

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 preprocessing 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:
~25% of code was sourced from online. This was mainly in the BackgroundRemoval.py code.

Citations:

1. https://iq.opengenus.org/vgg19-architecture/
2. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
3. https://theailearner.com/tag/cv2-medianblur/
4. https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomFlip
5. https://blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/
6. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939