6203 – Deep Learning
Final Project Proposal
Bradley Reardon, Salim Haruna, Divya Parmar
12/4/21

Automobile Image Classification

Table of Contents

Introduction

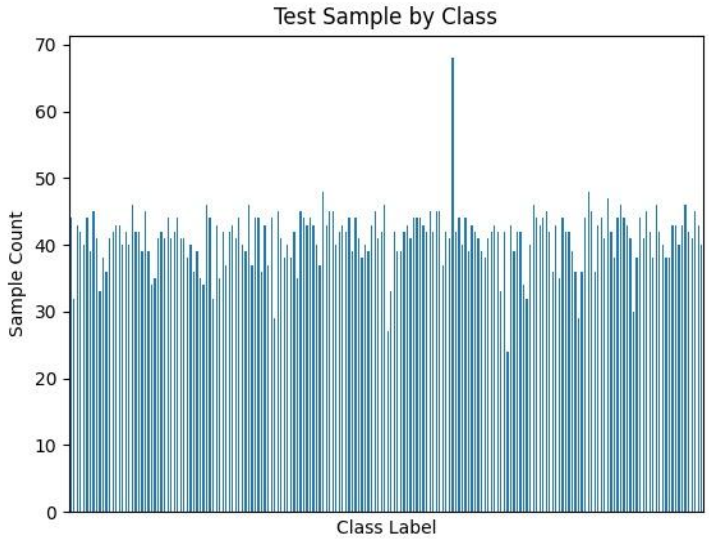
As technology advances, we continue to find labors that can be automated and in return free up humans to work on things that cannot be automated. With the power of machine learning and computer vision, the classification of images is one of those labors.

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.

Dataset Overview

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. The training dataset is fairly balanced, with approximately

30-50 images per class. However, with such a larger number of classes, the model was unable to classify well given the initial dataset size. Augmentation of images was implemented in order to increase the amount of images available per class for our model to train on.



Pre-Processing

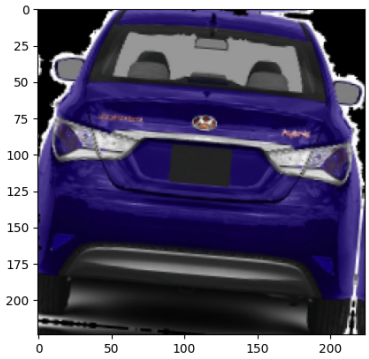
To prepare for our model, we conducted data extraction, data optimization, and data augmentation.

Data extraction consisted of reading in our image files, cropping them in accordance with bounding boxes provided by the research authors, and assigning labels.

Following extraction, we optimized our images by performing background removal, which will reduce noise by removing irrelevant information and allow our model to focus on the information provided by the car itself. To do the background removal, we first performed a median blur to remove salt and pepper noise (https://theailearner.com/tag/cv2-medianblur/). We then find significant contours, and use those contours to identify background pixels and white them out.

To augment our samples, we flip the image horizontally and vertically while randomly applying a rotation (https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomFlip). However, the image size stays the same so it can be passed to our model.

Lastly to save the time running the training, we implemented a technique where you can preprocess the images and save the numpy arrays to a numpy .npy file. Although the files saved are large due to the number of images, they also load in seconds.



Network and Models

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image (https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939). A

CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer. The convolutional layer performs a dot product between two matrices, one being the kernel, also known as a set of learnable parameters which is smaller than the image matrix, and the other being a predetermined portion of the image matrix.

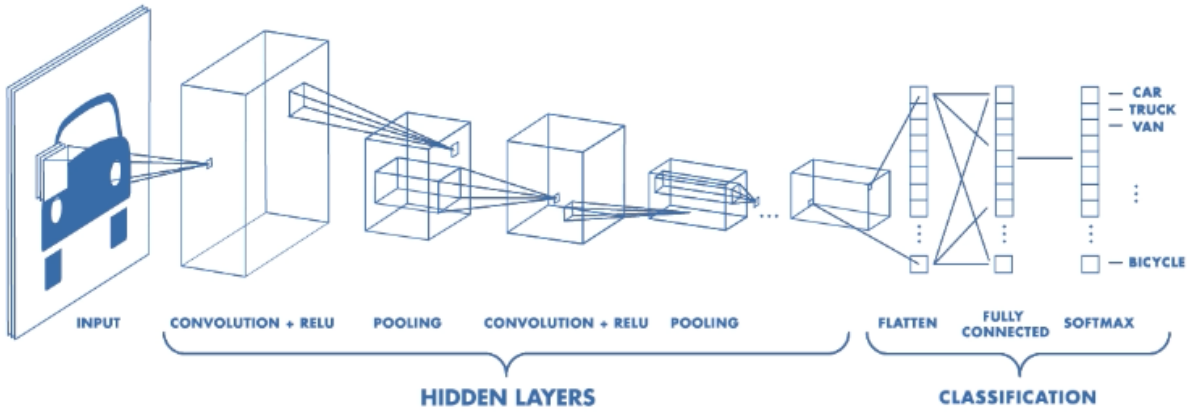


Figure 3: Convolutional Neural Network Architecture (Source)

The artificial neural networks we tested are the pretrained VGG19 network, the pretrained Inception V3 model, and a custom convolutional neural network (CNN) designed by us using tensorflow. The VGG19 architecture was built by a group named Visual Geometry Group at Oxford's, hence the name VGG, and is a deep CNN model pretrained on the ImageNet database, an image database consisting of 14,197,122 images organized according to the WordNet hierarchy. Figure 4 represents the VGG19 architecture.

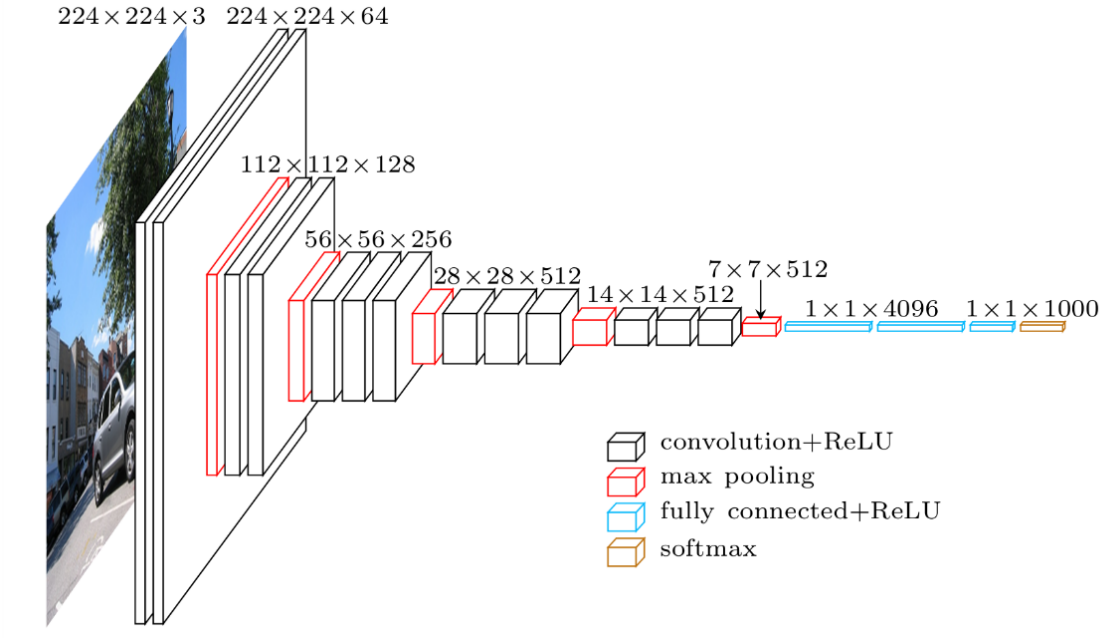
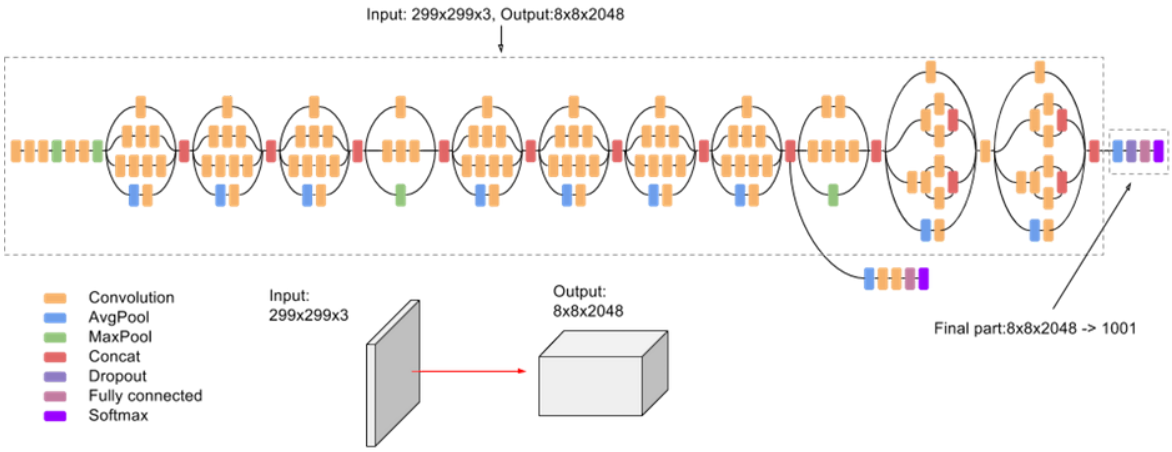


Figure 4: VGG19 Architecture (Source)

The VGG network, being pretrained on such a large dataset of images, provides the ability to initialize weights that have already been updated on the pretrained dataset, meaning we do not need to randomly initialize weights and use some of our training data to reach the performance provided by the VGG weights. This allows all of our training data to be used for further optimization and thus improves overall performance.

The Inception V3 model contains 48 layers, consisting of convolution, average pooling, max pooling, concatenation, dropout, softmax layers. This model is trained on over a million images and can classify up to 1000 classes. What makes the Inception V3 unique is that it includes auxiliary classifiers in between each layer, auxiliary classifiers being a small CNN inserted between layers during training, and the loss incurred is added to the main network loss. The following image represents the Inception V3 architecture.



Since we had such a large number of classes, the custom CNN we tried building was not classifying with much success. We concluded that the custom CNN model would not perform well, so we left it as a shallow model since it had no chance of competing with the pretrained models. The architecture of the custom CNN is Conv3x3 (16), BatchNorm, MaxPool, Conv3x3 (43), BatchNorm, AvgPool, Fully Connected (400), Dropout, BatchNorm, Fully Connected (197).

Experimental Setup

In order to load in the dataset, we ran a matlab script that stored all image file names in an excel file as a list, and used a dataloader to iterate through the list loading each image matrix as a numpy array. Once stored as an object, we split the dataset into train and test sets at a 90/10 ratio and the target feature was converted to categorical data.

Using tensorflow, we created a sequential model and added the VGG19 architecture as a layer. The VGG19 model parameters included using the “imagenet” trained values for weight initialization and setting the input dimensions to 224 x 224 x 3 to match the input the model was

pre trained on. The hyperparameters for all models are: batch size = 64 to prevent running out of memory per iteration, dropout = 0.5 to decrease chances of overfitting, loss = categorical cross entropy since we dealt with multi-class classification, epoch = 3 considering our train dataset was so large after augmentation that each epoch took extremely long, and number of classes = 197 since there are 197 differing classes in the Stanford dataset. The performance was measured by the F1-score since there were varying degrees of performance across all 197 classes during classification.

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

Conclusion

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.

Citations:

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