IMAGE CLASSIFICATION OF CARS USING CONVOLUTIONAL NEURAL NETWORK

Rachel Filderman
School of Data Science
University of Virginia
Charlottesville, VA 22904
raf2dh@virginia.edu

Jae Yoon Sung
School of Data Science
University of Virginia
Charlottesville, VA 22904
js2yp@virginia.edu

Congxin (David) Xu
School of Data Science
University of Virginia
Charlottesville, VA 22904
cx2rx@virginia.edu

July 26, 2021

ABSTRACT

Computer vision and image recognition are increasingly relevant deep learning topics entwined in society. Computers can be used to recognize objects in nontrivial tasks such as medical diagnosis and in everyday details such as facial recognition in unlocking iPhone screens. The dataset of interest in this study is labelled car images, from which class participation can be identified. Our focus in this project is in improving the accuracy of detecting the class (Make-Model-Year) of a car from an image through Convolutional Neural Networks (CNNs) and exploring which specific hyperparameters or regularization techniques aid in this goal. Though our team is interested specifically in recognizing the class of a car from its image, the technologies used and insights gained in this project in tuning CNNs can be extended to other image recognition tasks. In this project, we will integrate current knowledge from the deep learning community in image processing with our intended experiments, such as data augmentation, in order to improve model accuracy.

1 Motivation

One of the actively researched areas in the deep learning field is computer vision. The main idea of computer vision is to make computers recognize objects as well as normal human beings. By automating the object recognition step with deep learning algorithms, we are able to achieve a lot of tasks that we will not be able to do in the past. One may not realize this, but computer vision technology has already been a part of our everyday life. Every time we pickup our iPhone and put the front camera towards our face to unlock the screen, computer vision technology is used to recognize our faces. As of right now, computer vision has been used not only in facial recognition, but also augmented reality and self-driving cars as well as many areas. Therefore, in our final project, we will implement a computer vision model using CNNs in order to detect the model and make of a car. This model can be applied to many classes of images for processing tasks, but classifying images of cars has implications for security enforcement, traffic management, and other automated systems.

2 Method

2.1 Data Collection

Our data is collected and organized by a Kaggle user and it is provided as a public Kaggle dataset, *Stanford Car Dataset*, that can be downloaded by any Kaggle user. We downloaded all the car images as well as the class names of each car (Make, Model, Year) into our Google Colab disk space. All the car images belonging to the same class are organized in the same folder, which makes the implementation fairly straightforward. There are a total of 196 different Make-Model-Year combinations. We have 8,144 images in the training dataset and the 8,041 images in the testing dataset. A preview of some of the car images and their respective labels is shown in Figure 1.

Figure 1: Example Car Images and Corresponding Class Labels

2.2 Model Design

Our main goal of this project is to accurately classify the Make-Model-Year of a car based on the images of that particular Make-Model-Year. If humans were asked to perform the same task of classifying a car's Make-Model-Year from an image, they would focus on specific features of a car across the image, such as logo, shape, lights, wheels and etc., to make an educated guess about the Make-Model-Year of the car. To simulate this human behavior in classifying an image, we will use the kernel maps within the CNN. Given the massive success of existing CNNs, we will start modeling using some of the top pre-trained CNNs in order to create baseline models. The list of the existing pre-trained models within the Tensorflow package that we are going to use on our dataset and their respective characteristics is shown in Table 1.

Pre-trained CNN	# of Layers	Input Image Size	Year
ResNet50 [1]	50	224 x 224	2015
Xception [2]	36	299 x 299	2017
VGG19 [3]	19	224 x 224	2015
Inception [4]	50	299 x 299	2015
DenseNet201 [5]	201	224 x 224	2018

Table 1: Comparison of Pre-trained CNN Features

Regarding the implementation of these CNNs, they differ most simply in the input image size requirement. ResNet50, VGG19, and DenseNet201 all require the input image size to not exceed 224 x 224, whereas Xception and InceptionV3 have a slightly larger input image size of 299 x 299. However, looking more analytically at the architecture of these pre-trained CNNs, they vary by number of layers, depth, types of layers, and other architectural characteristics. After identifying 1 or 2 potential pre-trained models with high accuracy, we will explore different modifications on top of those models that may further improve the total accuracy of our prediction. We will also closely monitor the difference between training accuracy and testing accuracy and consider adding different regularization techniques to the existing model accordingly.

3 Experiments

In our project, we used the total accuracy metric in predicting the Make-Model-Year class of the images in our test set as the success metric to compare different models and hyperparameter settings. All the experiments were conducted in the default Google Colab Notebook environment with 12.72 GB RAM, 64.40 GB disk space and GPU acceleration. In addition, we used the Tensorflow package in Python to run all experiments. Our notebooks can be found in this GitHub Repository.

3.1 Pre-trained Model and Batch Size

The first sets of experiments we ran focused on different pre-trained models and the parameter, "batch size". Batch size is the total number of data that will be simultaneously trained in the CNN. In our example, it is the total number of images that were randomly selected as training data each time. Current research suggests that best batch size will depend on the environment and the dataset. Generally, practitioners will choose a multiple of 2 as a starting place [6]. We tested batch size 32, 64 and 128 for each pre-trained model listed in Table 1.

3.2 Data Augmentation

Before beginning hyperparameter tuning, our team carried out experiments on data augmentation in order to improve model accuracy and to prevent overfitting. Our team focused on the pre-trained CNN with highest accuracy among different batch size settings to move forward with data augmentation experiments. As the images are taken from different angles and in different environments, our hypothesis was that we can continue to improve on our benchmark accuracy by implementing data augmentation techniques. In the interest of time, our team will look closely into image rotation, random contrast, central crop, vertical flip and horizontal flip.

3.3 Hyperparameter Tuning

After experimenting with data augmentation, our team next moved onto model tuning. Since we are mainly applying transfer learning in this project, we did not made any changes to the overall structure of the pre-trained models. Therefore, we did not update the number of the neurons in each layer or the number of layers in the network. Our team first separated out the learning rate and the number of epochs from other hyperparameters. By implementing a learning rate scheduler and early stop mechanism, we can automatically tune those two parameters while testing other hyperparameters. Based on the previous experience of running different pre-trained models, we set the learning rates to be 0.001 as the starting point, 0.0005 after 10 epochs and 0.0001 after 20 epochs. The early stop mechanism is watching the sparse categorical cross-entropy, defined as the validation loss metric, and the model is going to wait for 10 epochs (patience) to early stop if the validation loss does not improve. We also chose not to restore the best weight when early stop is activated and we set the number of epoch to run to be 100. In order to find the optimal values for the rest of hyperparameters, our team set up a grid search of hyperparameters, which included testing different momentum settings and pooling types using the Adam optimizer. We then ran that grid search on our top three baseline models thus far. They are Xception model with batch size 32, DenseNet201 model with batch size 32, and InceptionV3 model with batch size 64 with horizontal flip. Horizontal flip was our most beneficial data augmentation technique, which will be explained further in the next section.

4 Results

4.1 Pre-trained Model and Batch Size

We compared these pre-trained CNNs across batch sizes and our results are shown in Table 2. Our findings after running these five pre-trained CNNs across batch sizes suggest that increasing batch size improves performance when

using ResNet50 and InceptionV3. However, an accuracy similar to that achieved through a large batch size (128) using ResNet50 and InceptionV3 (75-90%) can be achieved in a much smaller batch size (32) using Xception and DenseNet201 (80-85%). Research into the similarities and differences across these pre-trained CNNs carried out in the Model Design stage indicates that ResNet50 and InceptionV3 may perform similarly due to both CNNs incorporating 50 layers. Conversely, Xception is made up of less layers at 36 layers while DenseNet201 has a relatively greater number of layers at 201, which does not immediately indicate why these two networks were able to achieve accuracy similar to each other and to ResNet50 and InceptionV3 at batch size 128. Though, notably, the worst performing model on our dataset, VGG19, also happens to be the model with the least amount of layers. This may indicate a significant finding regarding what the architecture of the CNN should be in order to perform well on this specific dataset. The issue of DenseNet being out of memory is due to the usage of concatenation operations in DenseNet architecture. Furthermore, Xception is known to have better performance than InceptionV3 and our results demonstrate that.

Pre-trained CNN	**Batch Size**		
	32	**64**	**128**
ResNet50	51.87% / 1 hour	63.36% / 1 hour	* 75.81% / 0.9 hour
Xception	* 82.85% / 2.5 hours	Out of Memory	Out of Memory
VGG19	2.54% / 2.3 hours	Out of Memory	Not Improving
Inception	42.32% / 1.2 hours	61.81% / 1.9 hours	* 87.63% / 2.8 hours
DenseNet201	* 85.20% / 1.5 hours	Out of Memory	Out of Memory

Table 2: Comparison of Pre-trained CNN Performance on Car Dataset

Ultimately, as four separate model specifications and parameter combinations were relatively close in accuracy, as indicated by asterisks in Table 2, runtime may be a consideration in determining which pre-trained CNNs to continue with in tuning our model. The runtimes are included in Table 2 and show that ResNet50 (batch size 128) and DenseNet201 (batch size 32) had the lowest runtime among the top four accurate models.

4.2 Data Augmentation

Contrary to our hypothesis, the majority of the data augmentation techniques did not improve the total accuracy regardless of how parameters were changed inside each augmentation function. The built-in Tensorflow data augmentation, `tf.image.rot90`, which rotates the image counter-clockwise by 90 degrees, was tested. This data augmentation proved unsuccessful on ResNet50 and InceptionV3 and had an accuracy much lower than the benchmark. Additionally, the built-in Tensorflow data augmentation, `tf.image.random_contrast`, which adjusts the contrast of an image by a random factor, was tried on our dataset. This function requires an interval of random factors to be input. Out of the intervals tested, the relatively smaller interval, (0.2, 0.5), was most successful in training an accurate model. Though this data augmentation method was relatively successful and came within percentage points of the benchmark accuracy, it still fell short of the benchmark accuracy. Next, our team experimented with another the built-in Tensorflow data augmentation, `tf.image.central_crop`, which crops the central region of the image. Based on our experiments with various cropping ratios, cropping the car images was not able to improve on our benchmark. Furthermore, we flipped images vertically and horizontally by using the built-in Tensorflow data augmentation `tf.image.flip_up_down` and `tf.image.flip_left_right`, which flips the image vertically or horizontally. Performing vertical flip did not improve our model. However, applying one of the data augmentation techniques, horizontal flip, did improve on our benchmark accuracy, but only on the InceptionV3 model. Considering our scenario with car model classification, we found that horizontal flip will generate a mirror image of our cars and it make sense that the mirror image of a car can potentially provide additional information on the car features. Since horizontal flip was able to improve the total accuracy on the InceptionV3 model, we selected this model as one of the candidate models and applied hyperparameter tuning on it.

4.3 Hyperparameter Tuning

Different combinations of hyperparameters were compared using the total accuracy metric as displayed in Table 3.

Pre-trained CNN	Pooling	Momentum	Validation Accuracy
Xception	Average	0	87.81%
		0.3	84.03%
		0.6	**89.49%**
		0.9	87.64%
	Max	0	84.03%
		0.3	71.47%
		0.6	79.58%
		0.9	0.85%
DenseNet201	Average	0	75.76%
		0.3	**78.26%**
		0.6	73.56%
		0.9	58.24%
	Max	0	7.55%
		0.3	1.14%
		0.6	10.45%
		0.9	23.33%
InceptionV3	Average	0	84.16%
		0.3	**87.02%**
		0.6	86.59%
		0.9	84.69%
	Max	0	9.86%
		0.3	1.18%
		0.6	1.13%
		0.9	12.24%

Table 3: Total Accuracy for Hyperparameter Combinations with Adam Optimizer

After running grid search with learning rate scheduler and early stop mechanism, the best Xception model provides a 89.49% total accuracy with batch size 32. The hyperparameters used are average pooling with Adam optimizer and momentum decay equal to 0.6. The model early stopped at 30th epoch. One thing to note is that the max pooling with 0.9 momentum decay did not make any improvements. The total accuracy stuck at 0.85%. We believe we encountered the vanishing gradient issue at this point.

The DenseNet model is a network of dense blocks where layers are connected within each block. In this model, feature maps of all preceding layers are used as inputs [5]. The Tensorflow implementation of the DenseNet model has 201 layers. Running the grid search on this model with a learning rate schedule and early stop mechanism resulted in average pooling and 0.3 momentum decay as the best hyperparameters. The total accuracy of this model was 78.26% with batch size 32 and the Adam optimizer. The model stopped at 34th epoch.

InceptionV3 model which is a third version of Inception by Google with 48 convolution layers was grid searched with learning rate scheduler and early stop mechanism [4]. We obtained the best accuracy of 87.02% with average pooling and Adam optimizer with 0.3 momentum decay. The model stopped training at 27th epoch. On the other hand, using the max pooling with Adam optimizer performed very poorly in general.

Based on the results in Table 3, we can see that the best model is the Xception Model with batch size 32 and no data augmentation using average pooling and momentum 0.6. It provided a 89.49% total accuracy, which means that out of the 8,041 test images, the model was able to correctly identify 7,196 images belonging to 196 class (Model-Make-Year). We found this is very impressive given the large number of classes and subtle difference for some of cars.

5 Conclusion

Through trial of various pre-trained CNNs and hyperparameter tuning, our team was able to achieve a relatively high total accuracy of 89.49%. This project in identifying the classes (Make-Model-Year) of cars has potential extensions to be integrated into existing systems to assist in traffic control, identifying stolen vehicles, and enforcing traffic violations. Ultimately, our model could reduce manual tasks and potentially minimize human error along with improving accuracy. If deployed on mobile devices, this model could also have applications in helping individuals identify cars seen on the street by inputting an image of the car of interest. This application could be helpful for car retailers by directing individuals to the identified car's manufacturer website.

We learned that increasing the batch size may lead to better performance, but this is also constrained by the operating environment. Our hypothesis that data augmentation would contribute to the model's total accuracy was not confirmed. We also understand the effect of average pooling method versus max pooling method and discovered that average pooling method greatly outperformed max pooling method in our case. With more time and resources, a wider range of hyperparameters, including different kinds of data augmentation techniques, could be tested. In addition, we would like to verify our findings on more similar datasets.

6 Contribution

Our team met twice a week: once to determine objectives for the upcoming project deliverable and again to review each individual's work towards that deliverable. We collaborated on code using Google Colaboratory notebooks. For each programming and coding assignment, every member carried a fair share of the work. Written deliverables were split up equally among group members and each member would review the completed draft before submission.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[2] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[4] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[5] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.