# Image Classification of Car Models

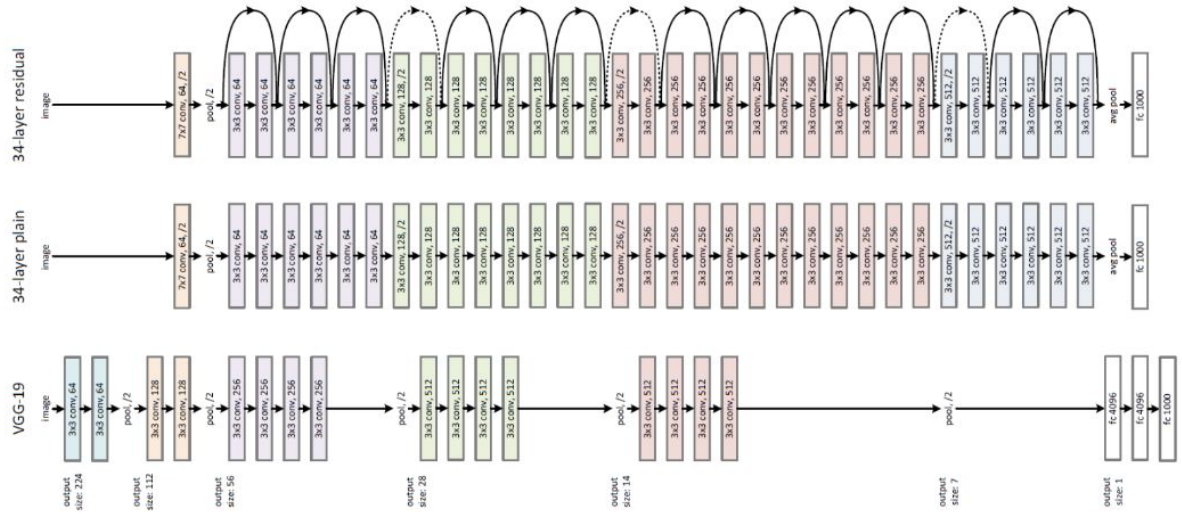
Group 21

Members:
Elizabeth Thorpe 74634224
Jason Ren 17766616
Babak Mehroziad 59772652
Tien Nguyen 68035743
Omkar Kawade 93566569

**Problem Statement**
If someone takes an image of a car but has no idea what model it is, they can use our application to find the car model and year. Using image classification, it will provide the results along with how confident it is that the match is accurate.

**Citations of relevant work**
The main application we are using to classify images is TensorFlow, a neural network trained with Keras to a data set, to classify specific images[1]. It's used for large-scale machine learning in Python by creating dataflow graphs. We downloaded a large dataset of different car model images from Stanford online. It contains 16000+ images and 196 classes split into training and testing images, including all the labels (devkit) [2].

For training with Keras, we used Resnet-152, a residual neural network created by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun at Microsoft research. One of ResNet's features is that it implements skipping over layers in order to get rid of the problem of vanishing gradients. This means that it will simplify the network by using fewer layers during the initial training stages. Thus there will be fewer layers to propagate through and will speed the learning. We discovered Resnet-152 from an article from medium.com where the research team won multiple image classification competitions[3]. It is known to have a very deep network:



34-layer ResNet with Skip / Shortcut Connection (Top), 34-layer Plain Network (Middle), 19-layer VGG-19 (Bottom)

In the article, it says that since the network is so complex, they used a bottleneck design to help reduce the time complexity. Flyyufelix provides a useful Resnet-152 Keras file we are using that loads in a pre-trained model on image recognition from a GithubGist post (resnet-152.py)[4]. He provides pre-trained weights and multiple implementations - we used the one for TensorFlow.

**Decomposition of our project**
The first step was making sure everyone has the correct dependencies installed for the project - installing TensorFlow and OpenCV. The project was developed in Python. Before running the program, we created a run.sh that will download and extract all files and dependencies needed to

run the program if they aren't already in the directory. This will be useful for anyone that wants to use the program that might not previously had the items we downloaded on our own.

Breakdown of files:
Preprocess.py:
This file takes in the devkit from the dataset of car images and labels we downloaded online. The file will open the MATLAB file which carries the car annotations in the "devkit" file. Afterwards it will also collect the bounding box information and uses that to create a margin of 16 pixels. The image is resized to a width and height of 224 pixels and then written to the file path using OpenCV. It also splits the images and labels into training (8144 images) and testing (8041) datasets, with 196 evenly split classes. They are then sorted into their respective folders. The folder which contains the data is in the data folder. The three directories within that folder is "train, test and valid" which have their respective images.

Train.py:
This file focuses on image augmentation with an image data generator class using Keras and Resnet-152 and training our model. After every epoch, a model with its accuracy and loss are saved.
On the training data, the code builds a resnet model by passing in the number of channels, classes, and image width and height. The data augmentation class, a generator, is prepared by passing in the rotation range of 20, width and shift range of 0.1, and zoom range of 0.2 (necessary arguments for the class).
On the test data, the code again builds a renet model but passes no parameters. This model uses callbacks with Keras - early stopping, reduce learning rate on plateau, and model checkpoints. Early stopping uses a patience of 50, which determines when training will be stopped when there is no improvement between epochs (number of iterations). Reduce learning rate on plateau determines when a metric has stopped improvement. Model checkpoints are used to save the model after every epoch and creates a callback list of all callbacks used. Like the training data, a generator is created with the same parameters.
After creating the training and test models and generators, each generator is fit to their respective models. The parameters passed into the final model are a batch size of 4, 30 epoch iterations, 6515 training samples, 1629 validation samples, and both generators. The steps per epoch is determined by the number of training samples divided by the batch size.

Run.sh
A basic script file to download the Stanford car dataset. The tar files include cars_test.tgz, cars_train.tgz and a devkit for car annotations and bounding box information.

Requirements.txt
Dependencies needed for this project.

Test.py:
This file loads the final resnet model created in the training file and all of the testing data. It then predicts and classifies each image to a class that represents a car model. The results are outputted to Results.txt.

Analyze.py:
This file loads the resnet model and test data. It then makes predictions on the test data using the model. To find the accuracy that the predicted label is correct, it compares the prediction to the actual test data label. The outputs are printed to the console.

Resnet_152.py:
This is the file taken from Flyyufelix. It contains a pretrained model on image classification. Everytime new data is trained on the model, the knowledge will add onto the pretrained model instead of retraining from scratch every time.

Demo.py:
This file takes in an image argument and converts their color and size to the same parameters as the images we trained in the previous files. For each of the images, the code predicts the class of the car model and prints out the file name, class name, and accuracy percentage to the console. An example demo run includes "demo.py carimage.jpg".

models/resnet152_weights_tf.h5:
Weights for the model. Link:
https://drive.google.com/file/d/1bXyHXAsKFBT9YTFZXIdQYZEv3X9snrQt/view?usp=sharing

models/models.23-00.89.hdf5
Model generated at the 23rd epoch. Link:
https://drive.google.com/file/d/1QzGhmRdn3yRJIJswnY9dbwNHN8fvKCtW/view?usp=sharing

Checkpoints:

1. Decided on project idea and researched technologies we could use for it. We ended up finding the stanford data set of car images and labels.
2. Distributed roles for the project and shared the research we learned.
3. Decided to use Resnet-152 and read over the implementation.
4. Coded files together in chunks as a team and debugged (preprocess, train, test).
5. Started running the files to get outputs for the report.
6. Finished up writing the report and creating the final submission.

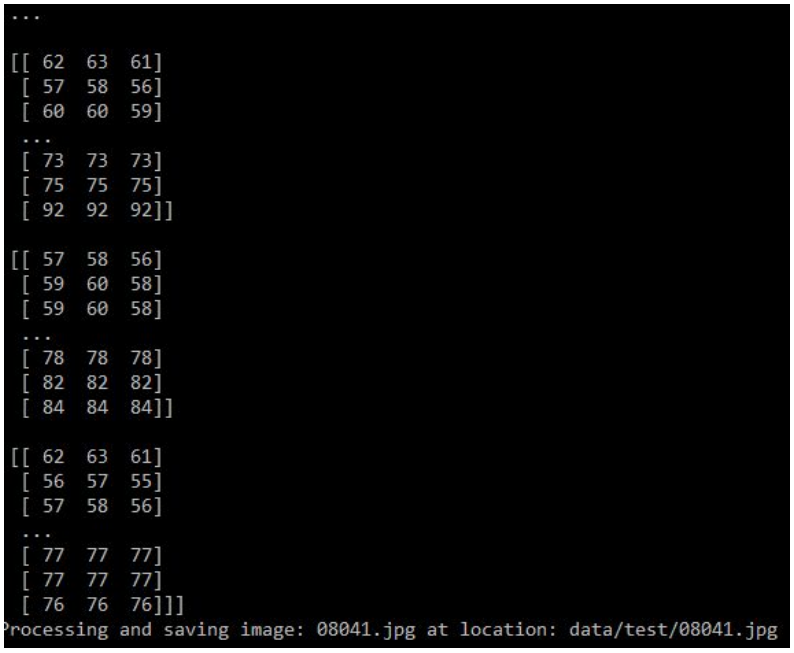
Who is working on what:
Omkar lead the project and acted as the facilitator for the whole group. Jason worked on writing the train file and Babak worked on writing and running the test and analyze files. We used Omkar's computer to run the train file for 10 epochs. Tien worked on writing and running the demo file. Elizabeth acted as the scribe and wrote up the report and created the poster. After everything was tested, Omkar gathered up all the files into the final document.

**Our experience coding it**
Since we are using such a complex model and including the resnet-152 implementation, we didn't realize that it would take a lot of time to train the model. Originally the train file was

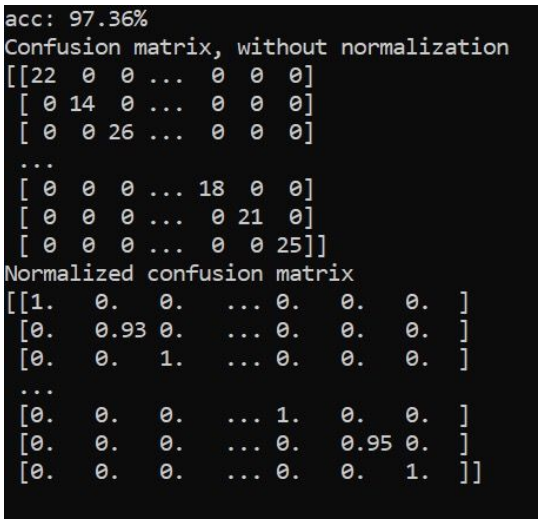
supposed to run for 50k epochs, but it turned out that each epoch would have taken around 1 hour to run. We also ended up using TensorFlow GPU with a Nvidia Geforce GTX 780 graphics card to gain more computation power and try to decrease the run time. It uses the video graphics card's memory in the computer to do the work alongside the cpu and does most of the heavy lifting when it comes to the computations. We ended up designating one of our team member's computers to running the application so we could run as many epochs as possible before the due date. Since we don't have enough time to completely train the model (since it's training from scratch) we are only running 28 epochs. However we were able to get up to a 89% validation accuracy.

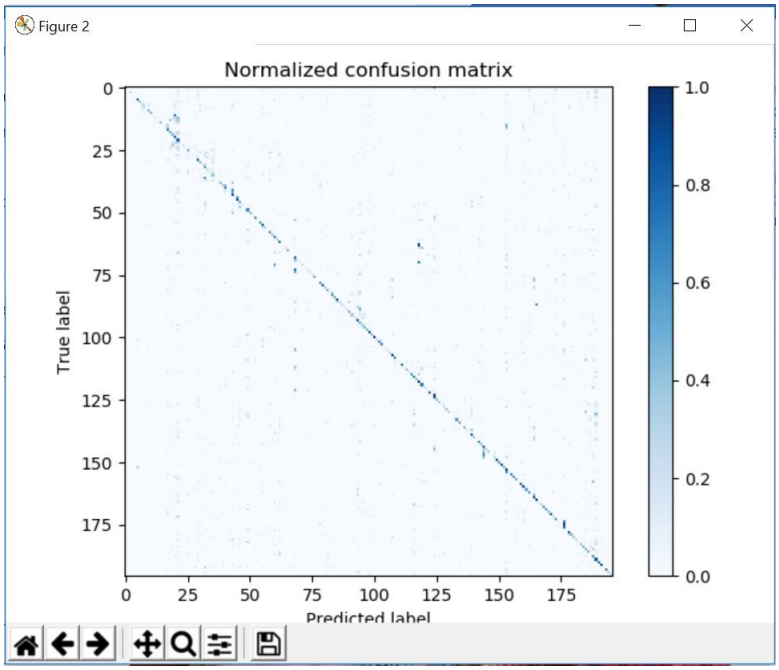
**Quantitative results/ plots**

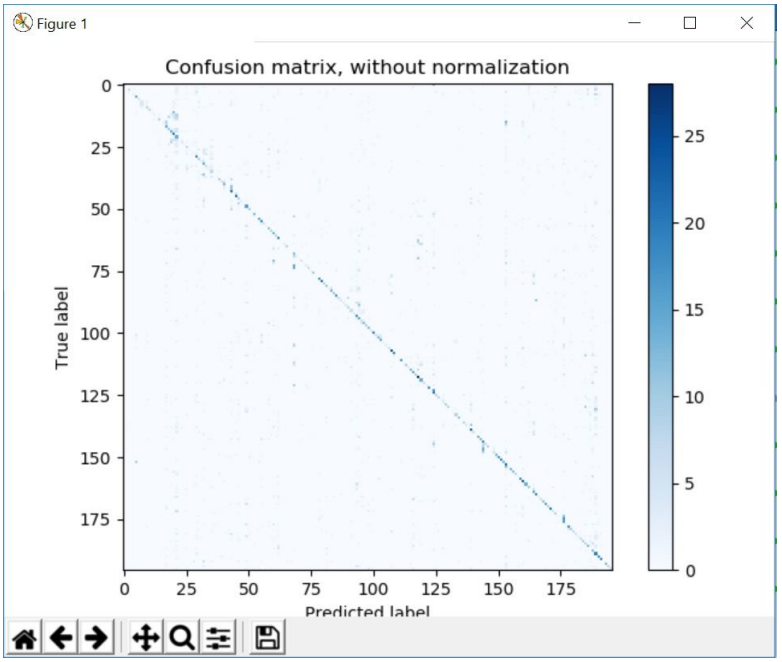
1. Preprocessing results:




2. Confusion matrices (analyze):





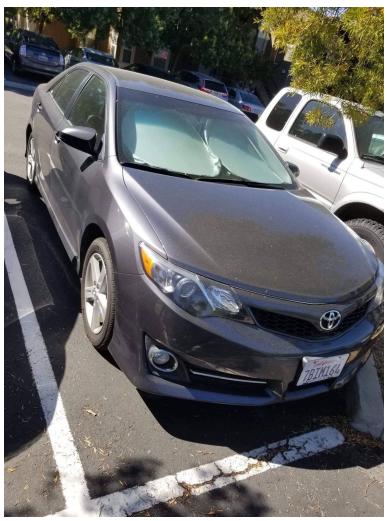
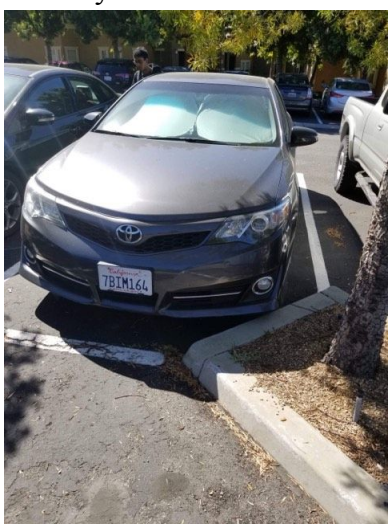

3. Results from train.py (first 10 epochs)

| Epoch # | Accuracy | Loss | Validation accuracy | Validation loss |
|---|---|---|---|---|
| 0 | 0.10425889177224851 | 4.38470643070013 | 0.2951127819548872 | 3.1331043436861874 |
| 1 | 0.4298580369454743 | 2.23683459144424 | 0.5492780916509730 | 1.7135328665873820 |
| 2 | 0.6493665089390329 | 1.2762968717317802 | 0.6792215944758317 | 1.2132548764786986 |
| 3 | 0.7717905663348503 | 0.8138348795740846 | 0.7476459510357816 | 0.9872565462498950 |
| 4 | 0.8412456113661433 | 0.5600741530877045 | 0.7809165097300691 | 0.8060362807308994 |
| 5 | 0.8787971302182269 | 0.4085119994244343495 | 0.7897049591964846 | 0.8290600710431826 |
| 6 | 0.9099374141352465 | 0.3261491023309709 4 | 0.8185812931575643 | 0.7263130123301849 |

| | | | | |
|---|---|---|---|---|
| 7 | 0.9337505724316898 | 0.24460914073644313 | 0.8267419962335216 | 0.7098756344236299 |
| 8 | 0.9421462372156922 | 0.20450626351520018 | 0.8298807281858129 | 0.7050566493810866 |
| 9 | 0.9549687070676233 | 0.16684265887016655 | 0.8480853735091023 | 0.729136396342807 |
| 10 | 0.9603114028392612 | 0.15002106563289072 | 0.8443188951663528 | 0.6691400153347372 |

Validation accuracy and validation loss are the accuracy and loss for the training data. When validation accuracy is greater than the training accuracy, it implies underfitting - we need to run more epochs to improve the validation accuracy[7]. At around 4 epochs, this is not the issue as training accuracy increases and passes validation accuracy. Due to power and time restraints, we were only able to run 28 epochs. The table above depicts only the first ten epoch runs, but at around the 13th epoch it was already getting 97% accuracy. By the 30th epoch, we were reaching a 99% accuracy. We believe the model generated from the 30th epoch might be susceptible to overfitting, so we used th model from the 28th epoch for the rest of the application.

4. Testing with Jason Ren's car:
We took pictures of Jason's car to use as testing images and the program successfully identified the first and last images but incorrectly identified the middle image.

JasonCar1 JasonCar2 JasonCar3

```
C:\Users\Jason\Desktop\Car-Recognition-master>python demo.py
Using TensorFlow backend.
WARNING:tensorflow:From C:\Users\Jason\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from tenso
rflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
2019-03-20 14:47:26.191420: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
Start processing image: data/test/JasonCar1.jpg
Start processing image: data/test/JasonCar2.jpg
Start processing image: data/test/JasonCar3.jpg
[{'label': 'Toyota Camry Sedan 2012', 'prob': '0.6377'}, {'label': 'Ford Edge SUV 2012', 'prob': '0.1793'}, {'label': 'Toyota Camry Sedan 2012', 'prob': '0.801'}]
```

**Conclusion & discussion**

Tensorflow/Keras - For a neural network library, Keras was much easier to implement than expected, and made developing this project fairly straightforward. There were built in functions for training and validation. So most of the scripts ended up being fewer than 50 lines. Powerful and easy to learn, but incredibly slow runtime, probably due to being implemented in python.

ResNet - ResNet 152 is an amazing neural network for image classification. It takes a lot a data and multiple training iterations ("epochs") to improve its accuracy to a good percentage. It has many features such as skipping over layers to improve efficiency. The model builds on constructs of pyramid cells in the cerebral cortex. These training sessions took an absurd amount of time and required a dedicated computer. After running one or two epochs, the accuracy was around 60%, and predictions were incredibly off, but after 4 epochs, the accuracy reached 73%, the predictions became decently accurate. But the model still was underfitting. ResNet is obviously intended to be run after many, many epochs, which we did not have the time or computing power to execute.

Future work:

We noticed when testing Jason's car, the middle image was misclassified. This could be caused by the types of photos in the dataset - most are images of the side view or angled view of the car. To improve the accuracy of correctly classifying car images, the dataset should be expanded to front views of the cars. Since there is a limited number of front views, the program thought the Toyota Camry was a Ford.

**References**

[1] "Train Your First Neural Network: Basic Classification | Tensorflow | Tensorflow." TensorFlow. N. p., 2019. Web. 17 Mar. 2019.

[2] Jonathan Krause, Michael Stark, Jia Deng, Li Fei-Fei, 4th IEEE Workshop on 3D Representation and Recognition, at ICCV 2013 (3dRR-13). Sydney, Australia. Dec. 8, 2013.

[3] Tsang, Sik-Ho. "Review: ResNet - Winner of ILSVRC 2015 (Image Classification, Localization, Detection)." Towards Data Science, Towards Data Science, 15 Sept. 2018, towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8.

[4] Yu, Felix. "Resnet-152 Pre-Trained Model in Keras." Gist, gist.github.com/flyyufelix/7e2eafb149f72f4d38dd661882c554a6.

[5] Prakash Jay. Understanding and Implementing Architectures of ResNet and ResNeXt for State-of-the-Art Image Classification: From Microsoft to Facebook [Part 1]. 7 Feb. 2018, medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624.

[6] Chablani, Manish. " RNN Training Tips and Tricks:" Towards Data Science, 13 June 2017, towardsdatascience.com/rnn-training-tips-and-tricks-2bf687e67527.

[7] "Validation Vs Training Accuracy." Deep Learning Course Forums. N. p., 2017. Web. 21 Mar. 2019.