Fine-Grained Categorization Using a Mixture of Transfer Learning Networks

Except where reference is made to the work of others, the work described in this project is my own or was done in collaboration with my advisory committee. Further, the content of this project is truthful in regards to my own work and the portrayal of others' work. This project does not include proprietary or classified information.

Justin Firsching

Certificate of Approval:

Sherif Hashem
Visiting Professor
Department of Computer and Information Sciences

Michael J. Reale, Chair
Associate Professor
Department of Computer and Information Sciences

Bruno Andriamanalimanana
Associate Professor
Department of Computer and Information Sciences

Michael A. Carpenter
Interim Dean
College of Engineering

Fine-Grained Categorization Using a Mixture of Transfer Learning Networks

Justin Firsching

A Project

Submitted to the
Graduate Faculty of the
State University of New York Polytechnic Institute
in Partial Fulfillment of the
Requirements for the
Degree of

Master of Science

Utica, New York
May 15, 2021

Fine-Grained Categorization Using a Mixture of Transfer Learning Networks

Justin Firsching

May 15, 2021
Signature of Author

Justin Firsching
Date of Graduation

Vita

Justin Firsching sparked his interest in the field of Computer Science young, being exposed to the Java programming language at the age of 12. Throughout Justin's high school education, there was no doubts among his family, friends, or classmates that he would one day work in the field of Computer Science. Prior to Justin's high school graduation, he began working a Software Engineering internship, further solidifying his love for the field of Computer Science. Upon his acceptance into the Computer and Information Science undergraduate program at SUNY Polytechnic Institute, Justin had immediately decided to obtain a minor in Mathematics as well as his master's degree through SUNY Polytechnic Institute prior to starting his career. Once able to take graduate courses, Justin began to hone in on his particular interest in Artificial Intelligence and Machine Learning.

Project Abstract

Fine-Grained Categorization Using a Mixture of Transfer Learning Networks

Justin Firsching

Master of Science, May 15, 2021
(B.S., State University of New York Polytechnic Institute, 2020)

30 Typed Pages

Directed by Sherif Hashem

In this paper, we apply a mixture of experts approach to further enhance the accuracy of transfer learning networks on a fine-grained categorization problem, expanding on the work of Firsching and Hashem [4]. Mixture of experts approaches may help to improve accuracy on categorization problems. Likewise, transfer learning is a highly effective technique for solving problems in machine learning of varying complexities. We here illustrate that mixtures of trained transfer learning networks, when applied properly, may further improve categorization accuracy.

Acknowledgments

This work is dedicated to my parents, John Firsching and Wendy Firsching for their endless love, support, and encouragement throughout every step in my life.

I would like to thank each of my committee members, especially Professor Sherif Hashem, for their support over the years. I have learned a great deal from each of them and they have all made a significant impact on my career trajectory.

I would like to thank my father, John Firsching, for all of the assistance he has provided to me throughout my life. My father was my very first mathematics teacher, and still helps me check for grammatical and some mathematical errors to this day. I would also like to thank my mother, Wendy Firsching, for her never-ending love, and for allowing me to play with her computer when I was a young child, sparking my interest in the area.

An additional round of thanks goes to my brother, Sean Firsching. Sean's support and seemingly infinite curiosity has allowed me to deepen my understanding of numerous mathematical and technical topics, many of which were needed in this work.

I would also like to thank my grandfather, Fred G. Firsching. My grandfather has never doubted my abilities in any field. His excitement to hear my accomplishments made each next big step that much easier to take.

Finally, I would like to thank my girlfriend, Jenna Smith. Jenna has always pushed me to be the absolute best I can be, and to always look at things in a positive light. Her support made the long nights of challenging, and at times aggravating, projects and assignments far more bearable.

Style manual or journal used Journal of Approximation Theory (together with the style known as "sunypolyms"). Bibliography follows van Leunen's *A Handbook for Scholars*.

Computer software used The document preparation package TEX (specifically LATEX2e) together with the style-file `sunypolyms.sty`.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Previous research shows multiple ways of combining trained neural networks through techniques such as optimal linear combinations and majority voting [5, 1, 13, 16]. More recently, Aggarwal highlights that ensemble based tricks provide a consistent improvement of between 2% and 3% with most architectures [1]. Aggarwal further explains that in the case of the 2012 ILSVRC competition, the winning error-rate was improved by 2.8% using an ensemble of seven models [1].

On the other hand, Firsching and Hashem (FH) [4] illustrated the effectiveness of applying transfer learning [12, 15] to numerous winners of the ImageNet Challenges [14] on a fine-grained categorization dataset of car images, provided by Krause, et al. [10]. The best performing network as reported by FH [4] scored an accuracy of 88.14%, an 11.04% improvement when compared to the results of Krause, et al. [10]. The aforementioned finely grained car dataset consists of 196 classes of vehicles classified on the basis of year, make, and model. Many of these classes consist of different years of the same and similar make and model of vehicle, appearing highly similar with few distinct features.

Chapter 2

Related Work

Firsching and Hashem [4] trained nine deep convolutional neural networks using transfer learning [12, 15] on the fine-grained car dataset [10]. These networks included various residual networks such as ResNet34, ResNet50, ResNet101, ResNet152 [6], AlexNet [11], SqueezeNet 1.1 [9, 3], VGG-16[17], DenseNet-201 [8], and an iteration of DenseNet-201 trained using Smith's One-Cycle policy [18]. Each network had 196 output nodes within the final classification layer group, where the predicted class is the index of the node with the greatest value. Prior to training, FH [4] performed a split on the training data so that 80% of the images remained for training and the other 20% of images were to be used for validation. The experimental results in FH [4] show that nearly every network had achieved a near perfect score when tested on the 6,442 training images [also shown in Table 4.1]. The best performing among the nine networks scored an accuracy of 88.14% on the 8,065 test images [4]. We can investigate further improvements in classification accuracy by utilizing combinations of transfer learning networks.

The 1994 Audi 100 Sedan was confused with the 1994 Audi V8 Sedan 172 times in total, being confused by all nine networks. The 2007 Chevrolet Express Cargo Van was confused with the 2007 Chevrolet Express Van 170 times, also being confused by all nine networks. Finally, the 2007 Dodge Caliber Wagon was confused with the 2012 Dodge Caliber Wagon 218 times between all nine networks. As displayed in Figure 2.1, these confusions are quite reasonable as these class pairs appear near identical, exemplifying the challenge of fine-grained categorization. With the high likelihood of the same images being confused by each of the nine networks, completely resolving the confusions through various combinations of FH's [4] trained networks would likely not be possible. However, through this test analysis, it was found that 4,192 confusions occurred in less than half of the networks. The pairs

Figure 2.1: Top 3 confused classes from the car dataset [10] among all networks by FH [4]. Vehicles pictured: 1994 Audi 100 Sedan (top-left), 2007 Chevrolet Express Van (top-center), 2007 Dodge Caliber Wagon (top-right), 1994 Audi V8 Sedan (bottom-left), 2007 Chevrolet Express Van (bottom-center), 2012 Dodge Caliber Wagon (bottom-right).

most confused by fewer than half of the nine networks can be seen in Appendix A, with the most frequently confused pair being confused 19 times across four networks.

Chapter 3

Mixtures of Transfer Learning Networks

We here discuss the effectiveness of some techniques for mixtures of experts.

3.1 Classification Summation

The first attempt at combining the FH [4] trained neural networks was summing the output of each network. The goal of this configuration was to allow the trained networks to apply their own weights to a selected class through the magnitude of the value in the output nodes. As with the nine individual networks, this combined network would select a class based on the node in the output layer with the greatest value. However, with no normalization of the output layer or any consistencies in the final layer outputs, aside from the number of classes, this was not how the network actually performed.

3.2 Majority Voting

A common method of combining neural networks trained on the same problem is through a majority voting convention [1, 13, 16]. When using majority voting to combine classification neural networks, each network is afforded a single vote as to which class the input data best aligns with. Majority voting is a popular technique due to its simplicity and effectiveness at ruling out incorrect classifications made by the minority of networks. Unfortunately, this is not a perfect combination technique. A potential weakness of majority voting is that the network votes are not weighted, thus the component networks are treated equally. This means that if multiple networks perform similarly, but one network performs significantly better than the other networks, the stronger performing network will still only get the same say in the final classification as the weaker performing networks. A lack of

weighting also means that if any network performs particularly well at correctly classifying a small section of classes, or even a singular class, majority voting has no intelligent method of selecting that expert when desirable.

3.3 Generation of Diverse Learners

Generating diverse learners may increase the potential benefits of the mixture of experts techniques [2]. We utilize a cross-validation approach to create diverse learners by training multiple couples of deep convolutional neural networks on different data subsets. The split of training and validation data as done by FH [4] results in a network that has not been exposed to any of the data in the validation set. By using a different subset of the training data as provided by Krause, et al. [10] for validation, it is possible to find a very different network to be the best fit for the data. Therefore, by training multiple networks on different variations of the Krause, et al. [10] dataset, we could have a few differing but still strong performing models using the same network architecture.

As previously stated, FH [4] initially split the training dataset as provided by Krause, et al. [10] into two subsets; one for training, and one for validation. FH's original training dataset consisted of 6,442 images, 80% of the original training data, to be used for training and 1,702 images, 20% of the original training data, to be used for validation. We created three additional sets of training and validation splits to perform three additional training sessions of leave-one-out cross-validation. The data was split evenly and randomly, the sizes of which can be seen in Table 3.1.

CV Number	Training Set Size (images)	Validation Set Size (images)
1*	6,442	1,702
2	5,866	2,278
3	6,062	2,082
4	6,062	2,082

Table 3.1: Cross-validation data splits.
*This is the same set used by FH for the original training.

Chapter 4

Experimental Results

We elected to use DenseNet-201 [8] with Smith's One-Cycle policy [18], ResNet101 [6], and VGG-16 [17] for further training using the CV sets due to their relative accuracies reported by FH [4]. Transfer learning [12, 15] was applied to each of these three networks using fastai [7] under the same preprocessing and hyperparameters used by FH [4]. The training for each network was halted when the loss on the validation dataset reached a minimum. Resulting accuracies on the training, validation, and test datasets for each of the CV networks can be found in Table 4.1. The original FH [4] dataset contained more training data and less validation data than each of the three additional data splits, as displayed in Table 3.1. This change in data allocation could explain the considerably better performance on the test data experienced by FH's trained networks [4].

Network	Cross-validation Set	Training Accuracy	Validation Accuracy	Test Accuracy
DenseNet-201*	1	**99.83%**	88.13%	**88.14%**
DenseNet-201*	2	99.57%	86.39%	86.69%
DenseNet-201*	3	99.72%	**88.52%**	87.45%
DenseNet-201*	4	99.82%	88.28%	87.03%
ResNet101	1	99.13%	82.54%	81.79%
ResNet101	2	99.23%	79.54%	79.62%
ResNet101	3	98.90%	82.90%	79.83%
ResNet101	4	99.11%	81.60%	80.60%
VGG-16 (D)	1	99.22%	83.31%	82.96%
VGG-16 (D)	2	99.68%	82.57%	82.84%
VGG-16 (D)	3	99.39%	84.10%	82.86%
VGG-16 (D)	4	98.68%	81.22%	80.71%

Table 4.1: Cross-validation Accuracy Scores. The bolded values indicate the highest score on the associated subset of each networks respective dataset.
*All training iterations of DenseNet-201 [8] used Smith's One-Cycle policy [18].

4.1 Majority Voting Among CV Networks

The three additional networks for each architecture trained using transfer learning could now be used in a majority voting system. This combination of networks should maximize performance as all of the original training data provided by Krause, et al. [10] is now being used in three of the four variations of each network architecture. This network performed significantly better, scoring an accuracy of 90.88%, a 2.74% improvement over the best accuracy scored by FH [4]. This is an overall improvement of 13.78% when compared to the performance of Krause, et al. [10].

Three additional majority voting variations were implemented for DenseNet201 [8], ResNet101 [6], and VGG-16 [17] as well. The DenseNet-201-based voting network scored an accuracy of 89.69%, a 1.55% improvement in accuracy over the equivalent network trained by FH [4]. The ResNet101-based voting network scored an accuracy of 86.05%, a 4.26% improvement over the results of FH [4]. Finally, the VGG-16-based network scored an accuracy of 87.39%, a 4.43% improvement over the results of FH [4]. All voting network results can be seen in Table 4.2.

4.2 Majority Voting Among Top Four FH Validation Networks

Majority voting was applied to the four strongest performing networks, in terms of accuracy on the validation dataset, as trained by FH [4]. These top performing networks include DenseNet-201 [8], both with and without the One-Cycle policy [18] applied, ResNet50[6], and VGG-16 [17]. This combined network ultimately scored an 88.05% accuracy on the test dataset. This is an improvement over most of FH's networks, but it is 0.09% worse than the accuracy scored by DenseNet-201 [8] with Smith's One-Cycle policy [18] on its own, as reported by FH [4]. Therefore, the majority voting actually hurt the overall performance on the test data due to the incorrect classifications being selected more frequently among the four individual networks than the correct classification.

4.3 Majority Voting Among All FH Networks

Another instance of majority voting was applied to all nine transfer learning networks trained by FH [4]. Again using fastai [7] for scoring, the combination of all previously mentioned networks scored a 88.41% accuracy on the test data. This is a 0.27% improvement in test accuracy compared to the best network trained by FH [4].

Experts	Accuracy on Test Data
Voting among nine FH models	88.41%
Top four FH models	88.15%
Voting among twelve CV models between DenseNet-201, ResNet101, and VGG-16	**90.88%**
Voting among four DenseNet-201 CV models	89.69%
Voting among four ResNet101 CV models	86.05%
Voting among four VGG-16 CV models	87.39%
FH Best Network	88.14%

Table 4.2: Results of Network Combinations

The best network achieved through majority voting scored an accuracy of 90.88% when tested on the data provided by [10]. This is an improvement in accuracy of 2.74% when compared to the results of FH [4], and an improvement in accuracy of 13.78% when compared to the results of Krause, et al. [10]. Additionally, each of the three sets of majority voting using just one architecture saw an improvement of between 1.55% and 4.43%. This improvement is exactly in line with the expected improvements reported by Aggarwal [1] of between 2% and 3% for most architectures.

4.4 Summation Among All FH Networks

A simple summation of outputs was applied to the nine FH trained networks. Through summing the output of each network, it was possible to score a classification accuracy of 85.84%. This score is better than the majority of the FH networks. However it is still a loss in classification accuracy from their best reported network of 2.3% [4].

Chapter 5

Conclusion

Our experimental results demonstrate that using a combination of trained neural networks can effectively improve categorization accuracy. Transfer learning [12, 15] is an effective method for solving classification problems, even when the data is finely-grained, as reported by FH [4]. We have demonstrated here that this capability is strengthened by using a mixture of transfer learning networks, using majority voting. We illustrated that through combining variations of networks with similar architectures but with differing training datasets, we can improve overall categorization accuracy. Through the use of cross-validation and transfer learning, DenseNet201, ResNet101, and VGG-16 resulted in an average increase in categorization accuracy of 3.4% over their FH [4] counterparts and 10.61% over the results reported by Krause, et al. [10].

Chapter 6

Future Works

Future expansions of this work primarily include the implementation and application of more advanced voting conventions. In this work we focused on the majority vote, where each component network has an equal weight in the final class selection. Based on the results seen with combining the FH [4] networks, it is possible that the introduction of weighting on a network basis may be advantageous. There is also potential that weighting based on the class selected from each component network is advantageous. Both weighting conventions may be investigated in the future.

Additionally, investigating the effect of ranked voting would be interesting. Especially in the case of frequently confused classes, a ranked voting scheme where each component network gets to select it's top-k predictions, may yield promising results.

Finally, it would be interesting to see how cross-validation batches of each of the nine networks trained by Firsching and Hashem perform under even the majority voting convention.

Bibliography

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer, 1 edition, September 2018.

[2] E. Alpaydin. *Introduction to Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2020.

[3] Wanghua Deng and Ruoxue Wu. Real-Time Driver-Drowsiness Detection System Using Facial Features. *IEEE Access*, PP:1–1, 2019.

[4] Justin Firsching and Sherif Hashem. Transfer Learning on Fine-Grained Categorization. In Kohei Arai, Supriya Kapoor, and Rahul Bhatia, editors, *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 1*, pages 561–567, Cham, 2021. Springer International Publishing.

[5] Sherif Hashem. Optimal Linear Combinations of Neural Networks. *Neural networks : the official journal of the International Neural Network Society*, 10:599–614, 1997.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016. IEEE.

[7] Jeremy Howard and Sylvain Gugger. fastai: A Layered API for Deep Learning. *arXiv:2002.04688 [cs, stat]*, February 2020. arXiv: 2002.04688.

[8] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[9] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360 [cs]*, November 2016. arXiv: 1602.07360.

[10] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D Object Representations for Fine-Grained Categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017.

[12] Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.

[13] Lior Rokach. *Ensemble Learning: Pattern Classification Using Ensemble Methods*. World Scientific, second edition, 2019.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[15] Ling Shao, Fan Zhu, and Xuelong Li. Transfer Learning for Visual Categorization: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 26(5):1019–1034, 2015.

[16] Amanda J. C. Sharkey. *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*. Springer Verlag, 1999.

[17] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, April 2015. arXiv: 1409.1556.

[18] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. *arXiv:1803.09820 [cs, stat]*, April 2018. arXiv: 1803.09820.

Appendices

Appendix A
Most confused classes by less than half of the Firsching and Hashem [4] networks

Class 1	Class 2	Network Count	Confusion Count
Aston Martin Virage Coupe 2012	McLaren MP4-12C Coupe 2012	4	19
Hyundai Accent Sedan 2012	Hyundai Sonata Hybrid Sedan 2012	4	10
Chevrolet Traverse SUV 2012	Hyundai Veracruz SUV 2012	4	10
Audi 100 Wagon 1994	Bentley Arnage Sedan 2009	4	9
Audi 100 Sedan 1994	Mercedes-Benz 300-Class Convertible 1993	4	9
Hyundai Azera Sedan 2012	Mercedes-Benz E-Class Sedan 2012	4	9
Nissan 240SX Coupe 1998	Plymouth Neon Coupe 1999	4	9
Ford Fiesta Sedan 2012	Hyundai Accent Sedan 2012	3	9
Daewoo Nubira Wagon 2002	Plymouth Neon Coupe 1999	3	9
BMW 1 Series Coupe 2012	BMW Z4 Convertible 2012	4	8
Audi 100 Wagon 1994	Chevrolet Malibu Sedan 2007	4	8
BMW Z4 Convertible 2012	Fisker Karma Sedan 2012	4	8
Hyundai Accent Sedan 2012	Hyundai Sonata Sedan 2012	4	8
Acura TL Type-S 2008	Mitsubishi Lancer Sedan 2012	4	8
Hyundai Tucson SUV 2012	Nissan Juke Hatchback 2012	4	8
Buick Verano Sedan 2012	Tesla Model S Sedan 2012	4	8
Audi 100 Wagon 1994	Volvo 240 Sedan 1993	4	8
Hyundai Veracruz SUV 2012	Volvo C30 Hatchback 2012	3	8

This table displays the classes of cars that are most frequently confused among less than half of the networks. The number in the confusion count column does not correlate to the number of unique images confused for another class and some overlap may exist between the multiple networks.

Appendix B
Data Splitting

```
#!/usr/bin/env python
# coding: utf-8

from splitfolders import ratio
import os
from random import randint
from sys import maxsize

seed_file = "split_seed_2021.txt"
seed = None

if os.path.isfile(seed_file) and seed is None:
    with open(seed_file, "r") as f:
        lines = f.readlines()
        seed = lines[0]

if seed is None:
    seed = randint(0, maxsize)
    with open("split_seed.txt", "w") as f:
        f.write(str(seed))

current_training_dir = "/workspace/School/CS498/car_data/train"
output = "/workspace/School/car_images/split"
train_pct = 1/3.
val_pct = 1/3.
test_pct = 1/3.

print(f"Data Directory is {current_training_dir}")
print(f"Output Directory is {output}")
print(f"Train Percent is {train_pct}")
print(f"Validation Percent is {val_pct}")
print(f"Test Percent is {test_pct}")
print(f"Seed is {seed}")

ratio_tuple = (train_pct, val_pct) if test_pct == 0.0 \
              else (train_pct, val_pct, test_pct)

ratio(current_training_dir, output, seed=seed, ratio=ratio_tuple)
```

Appendix C
Cross-validation Scoring

```
#!/usr/bin/env python
# coding: utf-8

import fastai
from fastai.vision import *
from fastai.vision.learner import *
from fastai.vision.data import ImageDataBunch

cv_set = 2
data = '/workspace/School/car_images'
net_path = "/workspace/School/CS598/MoreTraining/models/VGG-16_2"
databunch = ImageDataBunch.from_folder(Path(data),
                    size=256,
                    bs=1,
                    train=f"Training_Group_{cv_set}",
                    valid=f"test"
                ).normalize(imagenet_stats)
net = cnn_learner(databunch, models.vgg16_bn)\
          .load(net_poath).model

correct, total = 0, 0
net.eval()
for image, target in databunch.valid_dl:
    image, target = image.cuda(), target.cuda()
    out = net(image)
    predicted = torch.max(out.data, 1).indices
    total += image.size(0)
    correct += (predicted == target).sum().item()
    print(f"Network correctly identified {correct} of {total} "
          "images, {correct / total * 100:0.02f}%", end="\r")
```

Appendix D
Majority Voting Network Module

```
from torch import nn

class VotingNetwork(nn.Module):
    def __init__(self, class_count, *networks):
        super().__init__()
        self._class_count = class_count
        self._networks = networks
        for _network in networks:
            for parameter in _network.parameters():
                parameter.requires_grad = False
            _network.eval()
            setattr(self, str(_network), _network.cuda())

    def forward(self, inputs):
        votes = torch.zeros(
            [
                inputs.shape[0],
                len(self._networks),
                self._class_count
            ],
            dtype=torch.int32
        )
        for i, network in enumerate(self._networks):
            output = network(inputs)
            max_idx = torch.max(output, 1).indices
            votes[torch.arange(inputs.shape[0]), i, max_idx] = 1
        return votes.sum(dim=1)
```

Appendix E
Majority Voting Network Scoring

```
import fastai
from fastai.vision import *
from fastai.vision.learner import *
from fastai.vision.data import ImageDataBunch
from torch import nn

vgg16_1_path = "models/VGG-16_1"
vgg16_2_path = "models/VGG-16_2"
vgg16_3_path = "models/VGG-16_3"
vgg16_4_path = "models/VGG-16_4"

data = '/workspace/School/CS498/car_data'
databunch = ImageDataBunch.from_folder(
    Path(data),
    size=256,
    bs=16,
    train="train",
    valid="test"
).normalize(imagenet_stats)

vgg16_1 = cnn_learner(databunch, models.vgg16_bn)\
            .load(vgg16_1_path)
vgg16_2 = cnn_learner(databunch, models.vgg16_bn)\
            .load(vgg16_2_path)
vgg16_3 = cnn_learner(databunch, models.vgg16_bn)\
            .load(vgg16_3_path)
vgg16_4 = cnn_learner(databunch, models.vgg16_bn)\
            .load(vgg16_4_path)

net = nn.Sequential(VotingNetwork(
    vgg16_1,
    vgg16_2,
    vgg16_3,
    vgg16_4
))

correct, total = 0, 0
net.eval()
```

```
for image, target in databunch.valid_dl:
    image, target = image.cuda(), target.cuda()
    out = net(image)
    predicted = torch.argmax(out.data, -1).cuda()
    total += image.size(0)
    correct += (predicted == target).sum().item()
    print(f"Network correctly identified {correct} of {total} "
          "images, {correct / total * 100:0.02f}%", end="\r")
```

Appendix F
Summation Network Module

```
import torch
from torch import nn

class SummingNetwork(nn.Module):
    def __init__(self, *networks):
        super().__init__()
        self._networks = networks
        for _network in networks:
            for parameter in _network.parameters():
                parameter.requires_grad = False
            _network.eval()
            setattr(self, str(_network), _network.cuda())

    def forward(self, inputs):
        sums = torch.zeros(
            [
                inputs.shape[0],
                len(self._networks),
                196
            ],
            dtype=torch.float32
        ).cuda()
        for i, network in enumerate(self._networks):
            sums[torch.arange(inputs.shape[0]), i, :] = \
                network(inputs)
        return sums.sum(dim=1)
```