Final Project Report for CS 175, Spring 2018

Project Title: Classifying Distracted Driving with VGG16
Project Number: 19

Student Name(s)
Longinus Pun, 90955222, punl@uci.edu

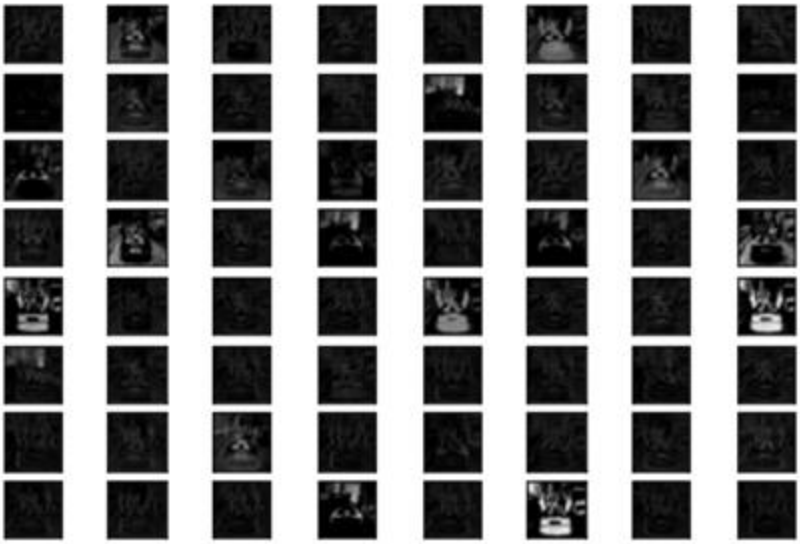
Introduction and Problem Statement

As provided by the problem in State Farm Distracted Driver Detection by State Farm on Kaggle from which I retrieved the data set, the goal is to predict the likelihood of what a driver is doing while driving based on images taken. With the huge number of submissions on Kaggle, there were a significant number that trained using VGG16 architecture. Thus, the object of this project is to identify some differences in the performance of VGG16 given certain changes in the architecture and training.

The primary subject among the different changes is to observe and display the change in validation accuracy as the number of epochs increases. Secondarily, I focus on how vgg16 performs compared to a simple CNN derived from another project in this course, as well as how a different classifier may affect accuracy.

Related Work

The extraction of features and visualization of filters in VGG16 is explored in many articles and research papers, including Hewage's [1] where he explores how VGG16 extracts features with Keras. He created visualizations of some of the filters that detect filters with edges and how it filters an image like the one below.



[1] Figure 1. VGG16 filters on an image

There is further extrapolated to highlight features of interest and display the detail abstracted from the model as the filters are deeper in the model.

Along with this article pertaining to extraction of features, there is research that delves into classification of flower species making use of deep learning. [2] Cıbuk tests AlexNet and VGG16 models to find features, implementing support vector machine classifiers and other techniques and experimental datasets. As presented in other research, VGG16 excels at feature extractions, leading me to research its success on distracted driving.

Data sets

My primary dataset for this research is the test data provided on Kaggle by State Farm [3]. It contains images of drivers in different states, along with their classifications. It also provided test data and csv of driver ids, both of which are not utilized in this research. The training data set provided consists of about 20000 training images classified into 1 of 10 classes based on the distraction, including not distracted, phone use, radio use, and talking with passengers.

While I initially planned to use the hand pose data set to recognize the hand postures/poses, it quickly became clear that it would be simpler and similarly efficient to make use of pre-trained models and weights. Thus, A secondary data set used is the ImageNet used to originally train the feature extraction of the pretrained VGG16.

[3] Figure 2. Drivers in different distracted poses.

Technical Approach

Convolutional Neural Networks (CNN) are a type of neural network that makes use of convolutional layers, activation functions, pooling layers, dropout layers, among others for the purpose of image recognition/classification and other uses. The VGG16 architecture I make use of in this project is an influential CNN developed by the Visual Geometry Group from Oxford that makes use of simple layers, primarily small convolutional layers and max pooling layers, and ReLU activation. I use Cross Entropy Loss and RMSprop optimization function for the training. In perform a simple preprocessing on the images that consists of normalization and reshaping to be easily processed.

First I extracted images from the Kaggle data set and tried to manually load in images using CV2 and the list of images provided by the csv file. However, they would have to be manually loaded into a PyTorch loader for the model. Thus I changed over to using PyTorch's ImageFolder function to transform/normalize the images and import them into a dataset. I then split the training dataset into a training and validation set with an arbitrary ratio of 20% and load it into PyTorch's DataLoader for use in the models.

I imported training and accuracy functions from a previous project to assist in the testing of a few CNNs and checking of validation accuracy. The training function runs the images through the model and performs the passes and optimizations while calculating loss. To check accuracy, the model is fed the images from the validation loader set and the accuracy is computed.

The very first CNN I make use of is a classifier similar to VGG16 that I used previously for training on CIFAR-10, created with PyTorch's Sequential function. This function also uses relatively small convolutional layers, Max pooling, Batch Normalization, and ReLU Activation functions with dropouts. The purpose of this CNN was to be used as a baseline for comparison with VGG16. Following up on the CNN, I start to use PyTorch's VGG16 architecture. The first test was running it from scratch on the data and training all the weights. Afterwards, I imported the pretrained weights from PyTorch and made small changes.

The first change was simply removing the last linear layer and replacing it with a new one that output for 10 classes instead of the 1000 classes as it was previously trained on ImageNet. The weights on the feature extraction were frozen and only the weights for the classifier at the end of the architecture were trainable. Following a similar methodology, I input a sample classifier that consisted of a few linear layers and a log softmax to replace the ending linear layer with 1000 output features.

At the end, I created a script to train the model with only the last linear layer replaced over multiple epochs to save the changes and plot the validation accuracy with Matplotlib to see the change in accuracy and how much the accuracy changes/improves as more epochs are run.

Software

Written By Me (Highlighted functions were not used in the end)

File Extraction
- In: Zipped Dataset
- Out: Folders of Images in classification
- Purpose: Unzip Dataset

CV2 Image loading
- In: Path to image
- Out: Image processed with CV2
- Purpose: Prepare images to load in

Manual Image Loading
- In: CSV file
- Out:
- Purpose: Find images referenced in csv file and load them with CV2 above

Random Split
- In: Data Set loaded manually
- Out: Data set split by a ratio
- Purpose: Split data for training/validation

PyTorch image loading
- In: Path to unzipped data set
- Out: Processed training image data set
- Purpose: Take images in classification folders, normalize/preprocess and prepare for loading

SubsetRandomSampler
- In: PyTorch loaded data set
- Out: Randomly split subset data
- Purpose: split images for training/validation for loading

Size/Space functions
- Purpose: Debugging purposes for my own CNN

Training models
- Explanation: Assorted versions of functions that take models, train them, then print out validation accuracy

Epoch_train
- Explanation: Wrapper for training function that trains model, checks validation accuracy, and saves it along with the respective epoch.

Saving Weights
- In: Takes models, reads weights/epoch number/validation accuracy from earlier runs
- Out: Saves weights/epoch number/validation accuracy to file
- Purpose: Takes a checkpoint of weights, runs it a number of times, then saves to a file for future use.

Plot_validations
- In: Takes validation dictionary mapping epoch to validation accuracy
- Out: Plot of validation
- Purpose: Visualize how accuracy changes with more epochs.

Not Written By Me

Model Training

Explanation: Taken from PyTorch.ipynb provided earlier but modified a little. Takes a model and trains it from training data (in PyTorch Loader)

Check Accuracy

Explanation: Taken from PyTorch.ipynb provided earlier in class. Takes model and validation set and computes accuracy according to validation set.

VGG16 modifications

Explanation: Taken from assorted sources. Freezes feature extraction weights trained from ImageNet, then modifies the layer in classification section.

Experiments and Evaluation

For my experimentation, I set up scripts to test the models and ran it on a training data set of about 18000 images. The primary metric for evaluation is validation accuracy after training the models. To retrieve this, a function inputs pre-classified images into a model, retrieves the classification, and compares it to the correct classification. From this, a percentage can be obtained of approximately how accurate the model is.

Epoch	Accuracy
1	83.9206066
2	89.45138269
3	88.49241748
4	92.03835861
5	90.9455843
6	90.65566459
7	94.04549509
8	92.573595
9	92.32827832
10	93.46565566
11	94.58073149
12	93.82247993
13	92.77430865
14	94.09009813
15	94.55842997

The primary experiment compared validation accuracies of VGG16 originally trained on ImageNet. I froze the feature extraction weights and input a new linear layer to output 10 classification outputs to make use of transfer learning. This way, the model did not need to be trained from scratch. I experimented with running a single epoch versus multiple epochs up to 15. The experiments found a general upward trend of accuracies as additional epochs were trained. The validation accuracy plateaus around 94% at 10 epochs. However, looking more closely at the validation accuracies shows a that the actual accuracies waver around despite the general trend.

Other experiments I performed included the evaluation of a different CNN I created originally for CIFAR-10 classification, as well as different versions and uses of VGG16. My original CNN obtained a validation accuracy

Figure 3. Plot of Epochs vs Validation Accuracy

of about 64% from 1 epoch. This was used as a baseline for reference after running VGG16 for a single epoch. My experimentation with VGG16 included running it bare without pretrained weights, which resulted in a very low initial accuracy of about 40%, and changing the last linear layer to a Sequential that included 2 linear layers and a log SoftMax at the end. Changing the classifier from a single linear layer dropped the 1 epoch accuracy to about 77%. The table below gives a comparison of validation accuracies of different models running a single epoch.

Model	Accuracy
VGG16	42
VGG16 with modified custom classifier	77
VGG16 with linear layer changed for output features	84
Custom CNN	64

Discussion and Conclusion

From this project, I learned about the difficulties of preprocessing to match a model's input, as well as the complexity of CNN architectures. Initially, I was under the belief that the more complicated and deeper architectures would always run more accurately than simpler ones. Primarily, I believed that VGG16, a influential and universally recognized architecture would run better than a simpler architecture I created even without previous training. However, I learned that this was not the case. My custom CNN had a higher validation accuracy after a single epoch than VGG16 trained only on the training data set. However, I was correct in my assumption that VGG16 with pretrained weights would be better at classifying after utilizing transfer learning.

Another expectation I had going into this project was regarding the accuracies after multiple epochs. With my experimentation on the model that utilized transfer learning, I had expected that the accuracy would spike up with early epochs, then plateau and start wavering once it approached a high accuracy. While the results show that the beginning epochs had the largest affect on validation accuracy, the way that there were small peaks and valleys all the way up was against my expectations. I realized afterwards that it was only the general trend of accuracies that would go up, and the accuracy after the model began to plateau would waver more than I had expected.

A major limitation to the approach that I took to this problem is computing power. Training on the entire training dataset for distracted driving took a lot of computing power and a lot of time initially. In order to make future progress, I would recommend that smaller subsets of data are used such that multiple models can be trained simultaneously so there are more samples for validation accuracy after a single epoch or multiple epochs. In my running of the project, a singular epoch took upwards of 3 to 4 hours, which scaled up to a statistically relevant sample size would take much longer. Other directions that could be taken for this project include a more thorough investigation of changing the classifier at the end of the architecture, and checking accuracy with a more logarithmic scale such that probabilistic accuracies are taken into account since forms of distraction can be different or merges of classes.

References

[1] CiBuk, Musa, et al. *Efficient Deep Features Selections and Classification for Flower Species Recognition*, ScienceDirect, Apr. 2019, www.sciencedirect.com/science/article/pii/S0263224119300284?via%3Dihub.

[2] Hewage, Roland. "Extract Features, Visualize Filters and Feature Maps in VGG16 and VGG19 CNN Models." *Medium*, Towards Data Science, 15 May 2020, towardsdatascience.com/extract-features-visualize-filters-and-feature-maps-in-vgg16-and-vgg19-cnn-models-d2da6333edd0.

[3] "State Farm Distracted Driver Detection." *Kaggle*, www.kaggle.com/c/state-farm-distracted-driver-detection/data.