A pair of deep pre-trained convolutional networks for distracted driver classification

Hlynur Davíð Hlynsson
Lisa Schmitz

KTH Royal Institute of Technology

Abstract. In this paper we are going to introduce our solution to a Kaggle competition. The aim of this competition was to build a program that can detect distracted drivers and classify the source of distraction based on given image data. Considering the tremendous amount of fatal accidents caused by distracted drivers this is an important task. Solving this problem might help to develop software that can warn the driver and bring his attention back to the traffic. In order to do so we use Google's TensorFlow to build a convolutional network that can classify the image data of drivers. An analysis of different approaches, network architectures, parameter settings and image distortions and their effect on the classification accuracy is provided.

Keywords: Image classification, Convolutional network, TensorFlow

1 Introduction

Kaggle is an online platform that offers everyone the possibility to participate in competitions and try to solve machine learning problems. Very often these solutions are valuable to companies and apply to real world problems. After submitting a solution to a competition the results are evaluated and feedback is immediately provided by assigning a rank to the solution. This reflects the accuracy of the solution compared to the ones from other competitors.

One of the latest competitions is an image based recognition and classification task to detect distracted drivers. The image data provided for this competition was taken by a dashboard camera that has been installed in a car. The images that have been taken show drivers displaying different behaviours. The training image data is split into ten different classes, one with pictures of non-distracted drivers and nine classes containing drivers being distracted in different ways. The task is to provide probabilities for unlabeled test data for each of the ten classes.

These ten classes are:

- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting - left

- c4: talking on the phone - left
- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: talking to passenger

On average, each class has 2200 training images. The images are 640 x 640 pixels. Note that the images were taken while a truck drove around dragging their cars on the street so the images are from a controlled environment.

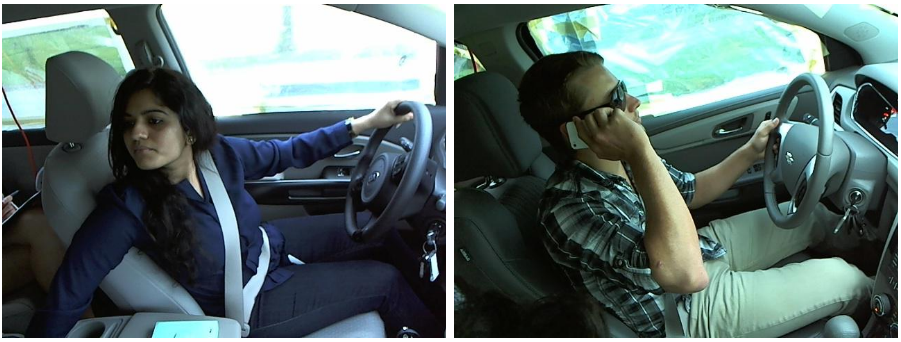
Fig. 1. Test data from classes c7 (right) and c2 (left)

The evaluation is done on a test set with approximately 80 thousand images, provided by the competition host. The competitors run their modle on this data set and upload their predicted class probabilities for each image. The website then evalutes the solution with the multi-class logarithmic loss

$$\text{log loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

Finding a satisfying solution to this problem could result in a distraction detection system in cars. Installing a dashboard camera connected to a trained network that can detect distractions and warn the driver could help to improve the horrendous car accident statistics and save lives. The host of this particular Kaggle competition is the insurance company State Farm Insurance. Drivers that have such a system and allow notifications when the driver is likely distracted could likely enjoy a lower premium on their insurance.

Nevertheless, working on this kaggle competition can also lead to results that might contribute to a better understanding of convolutional neural networks in general. Challenges of a bigger scope like ImageNet [1] but also smaller image recognition tasks like CIFAR10 are used to test, evaluate and improve state of the art network architectures. Researchers are still looking for the whole image of how and why convolutional networks are that well suited for image and object recognition. This is why solving those challenges by applying different methods and pretrained networks and evaluating their performance will maybe add a tiny piece to the big unsolved puzzle called deep learning.

For this task we will use transfer learning to classify the images with a deep convolutional neural network. By using a network that has been trained on much larger databases we can use them as a feature extractor and hope to use the final layer before the classification and retrain our own classification layer. Since our images seems to be of a similar scale (i.e. not microscopic or telescopic) of many large databases, we can hope that successful networks on popular, large databases can perform similarly well for ours.

2 Background

Though of a completely different scale one cannot help but think of the ImageNet challenge [1] as a similar problem. Instead of just ten classes and thousand of images, researchers were challenged to classify millions of images into a thousand different categories. Entire network architectures have been built to solve this problem and are now often provided as state-of-the-art networks that are retrained in order to solve other image recognition tasks. Therefore this is most relevant work for our project. Since training a convolutional network from scratch can take weeks we want to take one of those already trained models and add more layers to customize them for the specific task of detecting driver detection.

The network that performed best in 2012 on the ImageNet challenge was the AlexNet [2]. Except from a deep architecture with 60 million parameters they achieved an outstanding accuracy on the image data by applying new techniques to avoid overfitting [2]. By using data augmentation they could increase the amount of training data by a factor of 2048 and therefore use a large network with 650,000 neurons without a risk of overfitting. Furthermore, they randomly set the output of a node to zero during training which has a positive effect on overfitting. This strategy is known as dropout [2]. The AlexNet has become state-of-the-art and such a success cannot be ignored when confronted with a similar task. Thus those techniques should also be applied to the training of a network for the distracted driver competition also because the amount of training data which is available for us is rather small and therefore we have a higher risk of overfitting. But one should not forget that with 15 million labeled training images with 22,000 labels are of a completely different dimension then what we are confronted with in this challenge.

However, since this is a kaggle competition it would be foolish to only get inspiration by academic research work. Image classification tasks can be found regularly on kaggle and very often approaches of highly ranked solutions are described in detail by their creators. One famous competition took place in 2013 and was about classifying pictures of cats and dogs, known as the "Dogs vs. Cats" challenge. The most successful solutions made use of convolutional networks that have been trained with the ImageNet challenge data [3]. Since cat and dog classes are already included in this data set a high accuracy of 88% can be achieved with such a network [3]. However, for better results only the first layers of the pretrained network have been used for feature extraction and additional layers have been trained with the custom image data set [3]. This is an interesting approach for the distracted driver detection task and using a pretrained network is indeed how we tackled the distracted driver classification problem.

Another inspiring solution description is the one from the winner of the Galaxy Zoo challenge [4] simply because it is so detailed. It provides a great insight into common methods to approach image recognition challenges with convolutional networks. As pointed out this is not a classification task and a pretrained network will not be of a great help [4]. Despite from this a few practical applicable strategies can be taken out of this solution approach. It is mentioned that a combination of classifiers usually leads to a higher ranking [4] and should therefore be part of the experiments we are going to conduct.

3 Approach

TensorFlow as Google's newest machine learning library with the focus on image recognition seemed like an interesting choice to use. It provides pertained networks that can be enhanced and thereby adapted to a special task. The highest scoring solutions for image classification tasks are usually comprised of an ensemble of networks. We chose to experiment with the famous AlexNet network [2] as well and compare how they perform separately with the same training methods. We expect an average of their predicted probabilities to perform better than they do separately. In the following we will first introduce the general structure of TensorFlow and after that describe two different pretrained network architectures we used and different parameter settings and distortion methods we applied to them during training.

3.1 Transfer Learning

Since our data set is of a similar nature to large and well-analyzed data sets we thought that transfer learning would be a good approach for this task. It's predictable that a convolutional neural networks will be the most successful solutions in this competition but due to our time and hardware limitations we don't think it's feasible for us to design and train our whole network from scratch. We will take two well known networks and remove their top layer that is responsible for classification and replace it with our own, fully connected softmax layer.

AlexNet is arguably the network that started the deep learning phase. Published in 2012, it displayed groundbreaking results on the ImageNet dataset. It popularized rectified linear units as the transfer function between convolutional layers, a dropout training which randomly sets outputs of nodes to 0 and stacks convolutional layers without necessarily having a pooling layer between them.

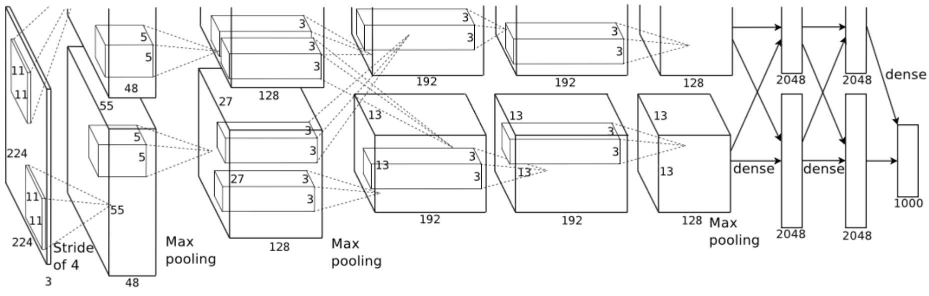

Fig. 2. Architecture of the AlexNet. From *ImageNet Classification with Deep Convolutional Neural Networks* by Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012.

Inception v3 is a version of the GoogLeNet which won the 2014 ImageNet competition. It has a better performance than AlexNet and only has 4 million parameters whereas AlexNet as 60 million. It shows that gains can be made through ingenious design rather than just stacking more and wider layers. The most salient feature of the network are its inception modules, which introduce a network-within-a-network, performing multiple operations in parallel. For example, their dimensionality-reducing inception module has one 2x2 max pooling layer and another 3x3 convolutional layer, and then concatenates all the resulting filters into one layer.

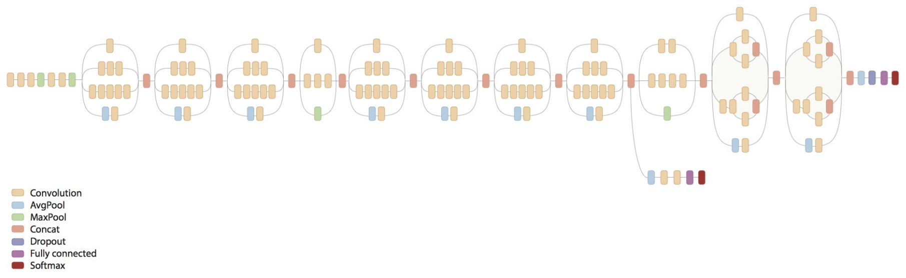

Fig. 3. Architecture of the TensorFlow model Inception v3. From *Inception in TensorFlow*, 2016. Retrieved from *https://github.com/tensorflow/models/blob/ master/inception/g3doc/inception_ v3_ architecture.png.*

3.2 TensorFlow

TensorFlow has a slightly different structure and uses different terms for concepts of a common convolutional network. Layers are represented by so called operations (`ops`) that can be added as nodes to the TensorFlow graph. Our model uses for example a gradient based optimization implemented in TensorFlow.

3.3 Generalization

To help us reduce overfitting, we will employ two methods suggested by Alex Krizhevsky to help generalize our model [2].

Dropout In our fully connected layer we give each node a 50% chance to returning 0, excluding it from the training process. This has been shown to help generalization and intuitively turns our model into an ensemble of all possible classification networks containing the features extracted from the pre-trained networks.

Data augmentation can help us effectively multiplying the size of our data. If our data is invariant to under a large family of transformations then we can hope to apply them to the image for "new" data, for example by rotating the image or flipping it vertically or horizontally. It doesn't appear that our data allows us to flip it but we might hope to rotate it by a few degrees which could be equivalent to the different ways the camera could be placed in the car. Other methods are changing the brightness or color ratios in the images, cropping the images and resizing which similarly account to the variability in the application environment.

4 Experiments

Our starting point of the experiments was a transfer learning tutorial for tensorflow [1], which we added on and customized to our needs as the project went on. In the following experiments we tested the performance of an Inception v3 and an Alexnet know to perform well on fairly similar image recognition tasks.

Most part of the experiments were spend on optimizing the hyperparameters of the network, trying to discover feasible ways of augmenting the data and finding good ways to train the model for lowest test loss in the timeframe of the project.

[1] https://www.tensorflow.org/versions/r0.8/how_tos/image_retraining/index.html

Mini-batch. We made used mini-batches with the size of 100 during the training. Since this mostly influences the training time no further experiments to optimize this parameter have been done.

Training and validation set. The labeled image data has been divided into two sets: the training and the validation set. We used eighty percent of the data as the training set and the other twenty percent for the validation set. This was done in order to do cross validation and early stopping.

Iterations. The amount of iterations has been determined with the help of early stopping. If the best validation accuracy has not been improved during the last ten evaluation steps then the training is stopped. This prevents us from overfitting the network.

Learning rate. We changed the learning rate manually. Starting with a learning rate of 0.01 we decreased it to a tenth after early stopping has occured.

Ensemble learning. A common practice to slightly improve the accuracy of the network is to instead of using only one model we let an ensemble of models predict the labels. This will mostly likely even out some errors specifically made in one network and therefore decrease the average loss. Moreover, it enables us to use the entire image data for the training and not only the random eighty percent that have been chosen as the training set for each model.

Data augmentation. One setting of distortions with a change of brightness of 10% and a crop of 10% was tried. We found that the training took way too long while we were applying the distortions so we chose to focus on faster learning with the next method.

Caching. Since we're using the pre-trained network only as a feature extractor without retraining anything but the classification layer, a great speedup can be achieved by only running the pretrained network once on each image and extract the features from the second-to-last layer. By doing this we are essentially changing the problem to one of only training a single fully connected softmax layer.

We did not only use an ensemble of ten models with the best training accuracy we could get but we additionally tested an ensemble of the best Inception net model and the best Alexnet model.

5 Results

In the following section we will present the results of our experiments. The evaluation of their performance will only depend on the accuracy that we achieved on the test set. Unfortunately the exact test accuracy was not accessable to us which is why we will use the score on the Kaggle Leaderbord instead. The score is the value of a loss function and therefore a low score is what is aimed for. In general we could achieve better results with the Inception v3 net than the Alexnet by applying the different experimental setup above.

5.1 Training of the Inception v3

To begin with we tried training the network with 40 thousand iterations. Our training prediction accuracy was 100% and we tried submitting it to the evaluation where we scored a whopping 6.40434 so we went back to the internet and discovered early stopping

The best training result of the Inception net gave a training with 5000 epochs and a constant learning rate of 0.01. This point has been found with early stopping. Further investigation of different manually set amounts of iterations with the same mini-batch size supported this first result. A such trained model could achieve a score of 1.08846.

Since building an ensemble of models seemed an attractive way to even out errors and get a better score. However, we tested an ensemble of six and ten of those models and only worsened our score by around 0.02 %. A reason for this could be that we got lucky with the training of the model that gave us our best score for the inception net. Other models that went through a training with the same hyperparameters do not necessarily predict as well as our best Inception model. On average the ensemble can still be better than a single model in the ensemble. But due to a restriction of only five submissions per day we were not able to prove this assumption.

5.2 Training of the Alexnet

We first tried training this network with 40 thousand iterations. Our loss for this was 2.36294 implying that it was heavily overfitted. Training of the Alexnet got stopped early after only 800 iterations. It took almost twice as long to extract the features with the net compared to the Inception one which is sensible considering it almost has quadruple the amount of parameters. The criteria for the stopping was the same as applied to the Inception v3 net: training stops if the best validation accuracy had not been improved during the last ten evaluation steps. The so trained model achieved a score of 1.46685. Other manually chosen amounts of iterations between 0 and 5000 gave no better result. However, further improvement could be achieved with an ensemble of seven different models that have been trained. By scoring 1.43171 we got exactly the slightly better result we expected.

5.3 A combined model of convolutional networks

We considered it an interesting experiment to create an ensemble of our two best results from both models - the Inception model that scored 1.08846 and the Alexnet model with a score of 1.43171. And as a matter of fact we achieved our final best score with this ensemble. With an improvement of 0.05959 we got a score of 1.02886. If we give Inception twice the weight of Alexnet, our score is 1.00412. And finally if Inception has three times the weight of Alexnet we get 1.00296. It seems like the Inception net is on the whole better for the task but having the Alexnet as well helps out. This could be because of the way that the loss function punishes incorrectly classified images. If our Inception net is doing some overfitting, than this can be remedied by smoothing out the incorrect predictions with the Alexnet.

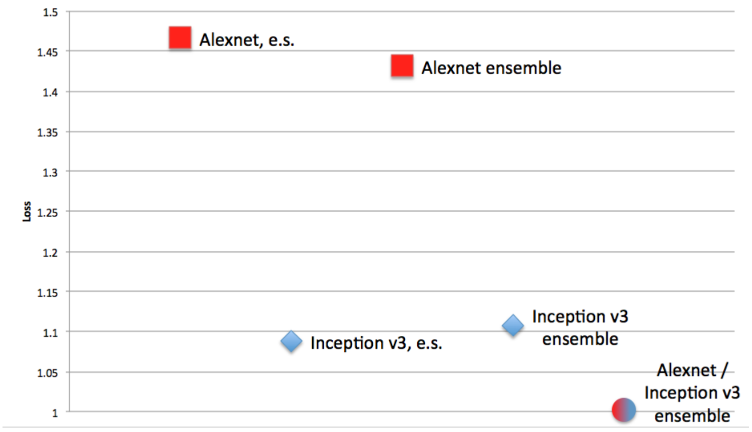

Fig. 4. Loss on the test set with early stopping (e.s.).

6 Conclusion

With our best classifier we could at the time of writing achieve a rank in the top 27% of the Kaggle competitors. This result is a promising start and there are many possible ways to improve this score even more. We were not able to implement the following improvements to the networks because of the limited amount of time that were available and the restriction of five submissions to Kaggle per day. But on of the first steps we will further take to train the network is an artificial enhancement of the image data. One obvious option is to add distortions like rotations, cropping and manipulation of the brightness. It would also be worth it to try more famous architectures that have ranked highly in similar competitions as feature extractors. Moreover, we have not manipulated the ensemble size yet. More or less than ten models might result into a better score. Another hyperparameter we should vary is the learning rate. Lowering this in a methodical way can help reducing the loss even more. Nevertheless, the way to go from this point is to enlarge the training set and hopefully train the network for more iterations while avoiding overfitting.

References

1. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision **115**(3) (2015) 211–252
2. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105
3. Z., Z.: Classifying images with a pre-trained deep network. http://fastml.com/classifying-images-with-a-pre-trained-deep-network/ (2014) [Online; accessed 20-May-2016].
4. Dieleman, S.: My solution for the Galaxy Zoo challenge. http://benanne.github.io/2014/04/05/galaxy-zoo.html (2014) [Online; accessed 20-May-2016].