State Farm Distracted Driver Detection

Mateusz Buda and Jokull Johannsson

KTH, Royal Institute of Technology
{buda,jokull}@kth.se

Paper ID 42

Abstract. According to the CDC motor vehicle safety division, one in five car accidents is caused by a distracted driver. Given a dataset of 2D dashboard camera images we classify driver's behavior. We first set the baseline with 4096 features extracted from the last but one layer of 16 layers VGG CNN and training simple MLP classifier with 10 output units and one hidden layer. Then we take the same VGG16 network with the last layer replaced to match ten classes in our task that we fine tune using the same training and validation sets. We test different learning settings like learning rate values for convolutional layers, fully connected and output layer and different optimization algorithms. The best accuracy of a fine tuned model is 150% better that we were able to achieve for simple MLP classifier.

Keywords: kaggle, CNN, deep learning, ImageNet, VGG16, fine tuning

1 Introduction

According to the CDC motor vehicle safety division, one in five car accidents is caused by a distracted driver. Sadly, this translates to 425,000 people injured and 3,000 people killed by distracted driving every year.

Given a dataset of 2D dashboard camera images, State Farm is challenging Kagglers [1] to classify each driver's behavior. Are they driving attentively, wearing their seatbelt, or taking a selfie with their friends in the backseat?

Images given are taken in a car with a driver doing something in a car (e.g. texting, eating, talking on the phone, makeup, reaching behind, etc). Our goal is to classify each image to one of tan classes.

The 10 classes to predict 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 this particular task we first set the baseline accuracy achieved with classification of 4096 dimensional representation of images extracted from the last but one layer of 16 layer VGG convolutional neural network (VGG16) [2] that proved to give very good image representation for various computer vision tasks [3] [4].

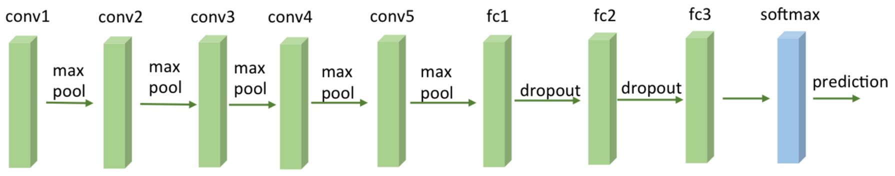

Fig. 1. VGG16 network architecture

Our goal is to achieve the best accuracy of predicted class on validation set that we extracted from given labeled training set. We split it based on drivers so that each driver is only present in one set.

We test different learning settings to fine tune the VGG16 CNN pre-trained on ImageNet with the last layer replaced to have 10 output units to select learning strategy that gives the best model. We try SGD and Adam as optimization algorithms but we focus on choosing the best learning rate weights

2 Background

Training the network from scratch with randomly initialized weights can be challenging due to high sensitivity of the learning result to many parameters and good initialization [5]. To train a good deep convolutional neural network we also need a lot of data that is not always available for a particular problem. Another constraint can come from limited computational resources since it takes a lot of time to train deep CNN even on the most powerful GPUs available.

Solution to this can be transfer learning. There are two major methods withing transfer learning. The first one is CNN as fixed feature extractor and the second is fine tuning the CNN [6]. Taking CNN as a feature extractor is very straightforward approach and can perform exceptionally well [7] [4]. Fine tuning is more tricky but can result in much better results [8].

The recipe for CNN as feature extractor method is to take CNN pre-trained on ImageNet with removed the last layer that output class probabilities. The rest of the CNN is then treated as a feature extractor for the new dataset. Then for all images we extract vector representation of size depending on the network architecture and chosen layer. Finally we train a simple classifier (e.g. SVM or MLP).

In fine-tuning we not only replace and retrain the classifier in the last layer of CNN on the new dataset, but we also change the weights of the pre-trained model. Some of the layers can be frozen (i.e. not updated at all) and some may be updated only by a small fraction. It is common to freeze (or update less) the layers closer to the input as they contain information about more generic features (e.g. edge detectors or color blob detectors) that should be useful regardless of tasks. Later layers of the CNN are more specific to the details of the classes contained in the original dataset.

3 Approach

3.1 MLP trainer on representations from VGG16

To set the baseline we used CNN as feature extractor approach. We took representation of the images by doing a forward pass through the VGG16 network pre-trained on ImageNet and skipping the last softmax layer.

We then used those feature vectors, of 4096 features, to train a simple multi layered perceptron network with one hidden layer. Different parameters were tested, with 256 unit giving the best result. The best accuracy score achieved was 32%.

3.2 Fine tuning VGG16

The same notwork was used for fine tuning. It is just a learning of the network that starts not with random weights but with the ones that are taken from the chosen pre-trained model. Another crucial difference is that learning rate is usually much smaller from the beginning as we already have almost fixed weights and we are somewhere close to the minimum and want to take small steps (updates). But the most challenging part in fine tuning is to the layers that will be updated and then to set how much they should be updated.

4 Experiments

4.1 Dataset

The dataset contains ∼22000 labeled images that we divided into train (80%) and validation set (20%). Images were split on drivers so that each driver is only in one set. VGG16 input is 224 by 224 color image with three channels (BGR). The input images are zero-centered by mean pixel subtraction. The StateFarm dataset provides images of size 640 by 480 so we also scale them to 224224.

Some of images in the dataset are reported to be mislabeled but in general the dataset is considered clean. Images do not vary in terms of rotation and scale. For all of them drivers are in the central part. Sometimes they only difference is in nuances (e.g. slightly rotated head for class c9 i.e. talking to a passenger) that is hard to spot even for humans that reached 94% accuracy on this task.

We are using the pre-trained model VGG16 that is trained on images from Imagenet. We add a custom layer with 10 outputs to the end of the pre-trained network to represent the 10 classes along with a softmax layer to calculate the probabilities of each class.

4.2 Setting the baseline

First we set the baseline accuracy with 4096 feature vector representation of images taken from the last but one VGG16 CNN. The we train MLP with one hidden layer and tanh activation function. For learning we used Adam optimizer with learning rate 0.0005 and momentum 0.9. Batch size was set to 256. We used L2 regularization with $\alpha = 0.001$ and also applied 10% of dropout in the input layer and 20% in the hidden one. We tested different sizes of hidden layers. Results of our test are summarized in the table 4.2.

hidden layer size:	128	256	512	**1024**	2048
train accuracy (%):	97	95	95	**100**	99
validation accuracy (%):	27	27	28	**32**	29

Table 1. Baseline classifier experiments with train and validation sets split on drivers.

The best model achieved 32% on validation set and that is the value we set for further comparison. We must mention that

In addition to prove that the split of training and validation set on drivers is crucial we conducted the same experiment but with a random validation set of the same size. Results of this experiments are shown in the table 4.2

hidden layer size:	**128**	256	512	1024	2048
train accuracy (%):	**99.9**	99.9	99.9	99.8	99.8
validation accuracy (%):	**98.9**	98.8	98.5	98.7	98.5

Table 2. Baseline classifier experiments with random validation set split.

4.3 Tested configurations

Table 3 show different experiments that were tested. It shows how different layers were fine tuned and the max accuracy it achieved on the validation set. Conv shows how much of the base_lr was used to update the convolutional layers in the pre-trained network. FC shows how much of the base_lr was used to update the fully connected layers in the pre-trained network. Custom shows how much of the base_lr was used to update the custom layer we added. Optimiser states which optimiser was used during training. Base_lr is the base learning rate that was used across all layers. Batch is the size of the batch used during training. Acc is the best accuracy the network achieved on the validation set.

Parameters that are constant across experiments is weight decay equal to 0.0005. SGD learning rate update policy is set to inverse decay:

$$\alpha_n = \alpha_{n-1}(1 + \gamma * iter)^{-p}$$

with $\gamma = 0.01$ and $p = 0.75$. Momentum is set to 0.9. Adam [9] is used with fixed learning rate policy and $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Experiments							
	Conv	FC	Custom	Optimiser	Base_LR	Batch	Acc
Experiment A	0%	0%	100%	SGD	0.001	64	33%
Experiment B	0%	0%	100%	Adam	0.001	64	35%
Experiment C	0%	50%	100%	SGD	0.001	64	44%
Experiment D	0%	5%	100%	Adam	0.002	64	55%
Experiment E	10%	10%	100%	SGD	0.001	32	55%
Experiment F	1%	1%	100%	SGD	0.001	32	64%
Experiment G	0%	50%	100%	Adam	0.001	64	47%
Experiment H	0%	100%	100%	SGD	0.001	64	48%
Experiment I	1%	1%	100%	Adam	0.001	32	**82%**

Table 3. Tested parameters used for fine tuning the pre-trained model together with their validation accuracy.

5 Results

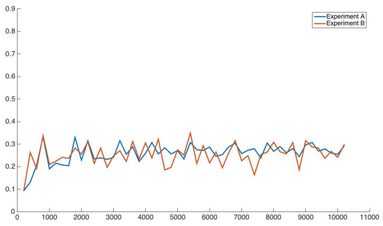

Fig. 2. Accuracy over 10000 iterations for experiments A(blue) and B(red)

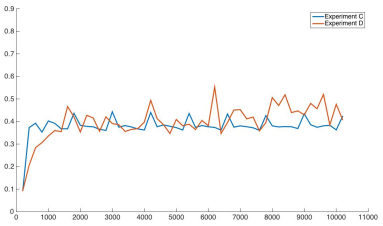

Fig. 3. Accuracy over 10000 iterations for experiments C(blue) and D(red)

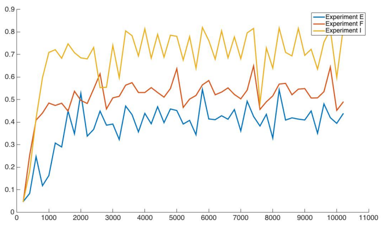

Fig. 4. Accuracy over 10000 iterations for experiments E(blue), F(red) and I(yellow)

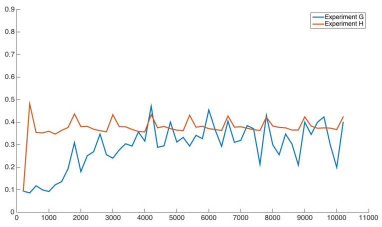

Fig. 5. Accuracy over 10000 iterations for experiments G(blue) and H(red)

Due to memory limitation batch size had to be relatively low, only 32 for the networks that trained all the layers. In most networks accuracy score grows fast in the beginning but stabilise at 3000 iterations but due to the batch size accuracy fluctuate.

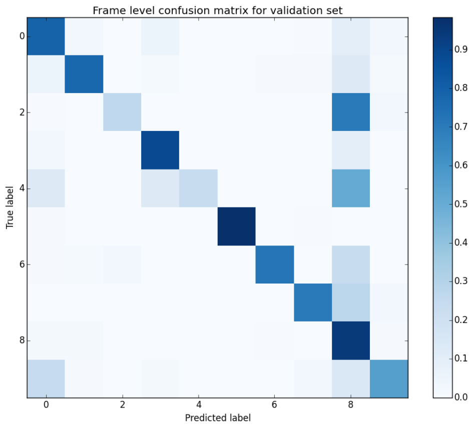

Fig. 6. Confusion matrix for the validation set

Figure 6 shows the confusion matrix for the validation set. On the x axis are the predicted labels and on the y axis are the true labels. It can be seen that the network is having troubles classifying label 8 correctly. This was expected because classifying label 8, which is the class "hair and makeup" can be hard to detect. While "operating the radio" gets 100% because the driver is reaching is hand across the image towards the camera and that is hard to confuse with other classes.

6 Conclusions

Like can be seen from Table 3 the accuracy improves by fine tuning the pre trained model. From the baseline of 32% we were able to get to 82% by fine tuning the parameters. In experiment I the parameters of the trained layers where changed 1% of the base learning rate, which is learning rate 0.000001 for those layers. Using the Adam optimiser outperformed the SGD optimiser by 18%.

Using a small learning rate proved to be better for fine tuning the network. Changing the pre trained layers with 100% of the base learning rate decreased the accuracy.

According to Kaggle, the human accuracy benchmark is 94% so there is still room for improvement before reaching the accuracy of a human. Also the best reported result outperform humans in this task reaching lest loss that translates to ∼98%.

7 Discussion

Although fine tuning a pre-trained network has proven to work for image classification it can sometimes be better to train the model from scratch. Unfortunately we did not have

On the other hand, we believe that for this specific task, results obtained with fine tuning can be better that for models trained from scratch. We thinks so because we found that this task is very prone to overfitting and using pre-trained model can serve as an effective form of regularization as most of the weights come from training done on general domain set of ImageNet and were barely updated during fine tuning.

Fin tuning is often done then there is not much enough data to train a deep network from random initialization. Then it may be beneficial to generate additional images by taking ones from original dataset and applying some transformation e.g. translation, mirroring, rotation, scaling, etc. However for this particular problem some of them are irrelevant and others probably would not improve anything as the variance in rotation, translation or scale is small. Nevertheless we did not try this.

8 Future work

Although the result presented in this paper have shown the fine tuning a pre-trained model to be able classify a drivers behaviour there are number of ways to improve. Since the release of VGG16 there have been numerous other network released that have been proven to perform better than VGG16 for some image based recognition tasks. Future work might include trying to fine tune other networks and comparing the results and see if they perform better.

Due to the limitation of GPU memory we had to decrease the batch size down to 32. By running the training phase on a GPU with larger memory we could increase the batch size which should lead to more accurate gradient computation, more stable loss and faster convergence.

The proposed training strategy allows the system to learn all layers end-to-end from data, without involving techniques that are specific to the task at hand. We believe that involving more domain knowledge and visualising the output of each convolution layer we could fine tune the model to improve the accuracy. It can be done by setting higher learning rate update for the hidden convolutional layers that are specialized in detecting objects and structures relevant to the task e.g. hand, mobile phone, cup, face, etc.

References

1. Kaggle: State farm distracted driver detection (2016)
2. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014)
3. Rothe, R., Timofte, R., van Gool, L.: DLDR: Deep Linear Discriminative Retrieval for Cultural Event Classification from a Single Image. ETH-Zürich (2015)
4. Hu, F., Xia, G.S., Hu, J., Zhang, L.: Transferring deep convolutional neural networks for the scene classification of high-resolution remote sensing imagery. Remote Sensing **7** (2015)
5. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th international conference on machine learning (ICML-13). (2013) 1139–1147
6. Karpathy, A.: Convolutional neural networks for visual recognition (2016)
7. Razavian, A.S., Azizpour, H., Sullivan, J., Carlsson, S.: CNN features off-the-shelf: an astounding baseline for recognition. CoRR **abs/1403.6382** (2014)
8. Castelluccio, M., Poggi, G., Sansone, C., Verdoliva, L.: Land use classification in remote sensing images by convolutional neural networks. CoRR **abs/1508.00092** (2015)
9. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)