Apply and Compare Different Classical Image Classification Method: Detect Distracted Driver

Ben(Yundong) Zhang Email: yundong@stanford.edu

***Abstract*—This project aims to build a computer vision system to detect and alarm the distracted driver. Using the dataset provided by Kaggle, we are interested in using machine learning algorithm to solve the problem. Several methods have been tested and the Convolution Neural Net proves its state-of-the-art performance, which is able to get log loss of 0.22.**

***Key*: SVM, CNN**

I. Introduction

Drivers are supposed to be focusing on driving by law. However, it is very common to see drivers doing something else while driving: texting, drinking, operating the radio, talking on the phone and etc. This distracted behaviors easily cause crash incidents. According to the report from National Center for Statistics and Analysis [1], each day there are over 8 people killed and 1,161 injured in crashes due to a distracted driver in US, which translates to 423,765 people injured and 2920 people killed each year. To alarm the distracted driver and better insure their clients, State Farm Insurance hopes to design an alarm system that can detect the distracted behavior of car drivers by using a dashboard camera. As a result, they held a online Kaggle competition [2] to encourage Kaggler to build a robust computer vision system. Specifically, in this task we were provided with an image dataset which consists of 10 classes of driver behaviors. For each test image, we were required to output the probability that the image belong to each of the ten classes. Two algorithms have been tried here and compared for the performance: Support Vector Machine (SVM) and Convolution Neural Network (CNN). To supplement the training set, pseudo-label semi-supervised technique [3] is used. We also implemented a recently-developed CNN structure called VGG-GAP [4] for visualizing what the neural network is looking for in the task, so as to better analyze the learned pattern and search for improvements. This task is very meaningful for improving the drivers' safety and can be easily applied to other applications such as triggering autonomous driving and etc.

II. Related Work

Machine learning has been proved to have powerful ability in performing image classification and object detection task. For SVM, the raw pixel feature usually performs poorly due to the high correlation between adjacent pixels. Thus the key is to choose a good feature so as to separate different classes well. The Histogram of Oriented Gradients (HOG) is a typical candidate. In [5], Kernel SVM trained with HOG feature is able to detect pedestrian with miss rate less than 10% at 10^{-4} false positive per window. In another well-known classification task MNIST hand-written digit classier, it is reported in [6] that the linear SVM with HOG feature can achieve accuracy of 97.2%, which is very close to the highest accuracy result (99.7%) achieved by CNN [7].

The state-of-art performance in almost every image classification task is achieved by using CNN. Utilizing convolution layers and deep architecture, the network is able to learn complex feature (e.g. visual pattern like hand, phone and etc) directly. By far the state-of-art model is the Inception-v4 model, achieving a 3.08% top-5 error on the test set of ImageNet [8]. Another famous model that we are particularly interested is the runner-up in ILSVRC 2014—VGGNet [9]. VGG has a very homogeneous architecture and is easy to follow. Also, the pre-trained model is available in many platforms and ready to plug, which makes the work of transfer learning much simpler.

Although CNN is known to be able to achieve a remarkably good performance, understanding the internal representation learned by CNN is hard. Recent works mainly focus on deconvolutional network [10] and inverting deep feature at different layers [11]. These methods are complex and cannot highlight the importance of the features. We found a very easy-to-implement VGG-compatible model called VGG-GAP [4], which is able to show the important image region for discrimination by using global average pooling layer. This visualization capability is able to tell us the quality of the trained model.

There is a previous year project working on the same topic using CNN approach [12]. However, the author is only able to achieve an accuracy of 21.7% and log loss of 11.421, which is significantly wore than our best performance (log loss 0.22).

III. Dataset

In this task, we are provided with 10 classes of 640x480 pixels RGB image data taken in a car for training, each corresponding to one type of behavior of a driver. The ten classes are: (c0) safe driving; (c1/c3) texting with right/left hand; (c2/c4) talking on the phone with right/left hand; (c5) operating the radio; (c6) drinking; (c7) reaching behind; (c8) hair and makeup; (c9) taking to passenger, where each class contains around 2,300 images (so the training set is balanced). Additionally, a list is provided, giving the image id, the corresponding driver id and the class. For test data, there are about 80,000 unlabeled images. The train and test data are split on the drivers, such that one driver can only appear on either train or test set. We split the train data into a train set and validation set, mainly using the K-folds cross-validation

technique. One thing we would like to highlight is that, in order to correctly reflect the validation score, the sets should be divided according to driver list. Recall that in the test set, the drivers are all unseen. If we validate our model by randomly splitting, there is a high possibility that we overfit the model to focus on the driver, instead of the actions. By splitting according to driver, we make sure the test set and validation set have same property, so the optimizer can work as desired. Given this dataset, our ultimate goals is to predict the likelihood of what the driver is doing in each image, so that the alarm system can make a proper remind to the driver. Some sample images are shown below:

Fig. 1. Sample drivers image: from top left to bottom right is (1) safe driving; (2, 3) texting (left & right); (4) operating video; (5) drinking; (6) reaching behind; (7) hair and makeup; (8) talking; (9) phone talking (left)

We do different data pre-processing for different methods. So the image pre-processing details will be discussed within the corresponding method section.

IV. Method

A. SVC

The first model we tried is SVC. SVC is nothing but a SVM using one-vs-one scheme to handle the multi-class classification. SVC tries to find $M(M-1)$ hyper-planes that best separate each pairwise classes, recall that M is the number of classes. Mathematically, the objective function of SVC for each pairwise classes is:

$$\begin{aligned} \min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{1}^{n} \zeta_i \\ \text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i \\ \forall i, \zeta_i \geq 0 \end{aligned} \tag{1}$$

where y_i is the label and $\phi(x_i)$ is the training vector. There are mainly three hyper-parameters that affect the performance of SVC in this case: the penalty term C, the kernel shape (linear or Gaussian) and the learning rate γ. The penalty term C typically specify how we want to optimize the hyper-plane: a large value of C penalizes more for outlier and will try to correctly classify all points; while a small value of C will encourage the optimizer find a hyper-plane with largest margin. Unluckily, in general there isn't a thumb of rule to determine the best C without knowing the shape of the data. The other two parameters are intuitive.

We perform two-step image pre-processing before feeding the data into the SVC classifier:

1) Bounding Box on Human Body: As we can see in Fig. 1(9), some of the training images consist of excessive components that might confuse the classifier (in this case the legs on the backseat). Hence we use the HOG feature to train a bounding box on human body in order to filter out this part of image and focus on the indicative part. We first manually annotate the human body of 500 images using MATLAB Image Labeler [13], then we train a human body detector using the HOG features on a cascade algorithm which consists of several stages and each stage is a boosting classifier [14]. The HOG feature basically calculates the distribution of intensity gradient (edge directions), which outlines the overall shape of the object. An example image is shown in fig. 2.

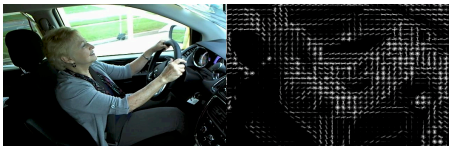

Fig. 2. HOG feature

The boosting algorithm ensembles many weak classifiers (decision stumps) and take a weighted average of them to perform classification. Specifically, in each stage a moving window is sliding across the image region and the corresponding classifiers label the region as positive or negative. Only the positive regions are passed to the second stage which focus on other features of the image. The benefit of doing this 'cascade thing' is that image region that is out of interest can be discarded as fast as possible [15]. We then crop the image by the bounding box, as in fig. 3. The cropped images are then resized to be 224x224x3 for further processing.

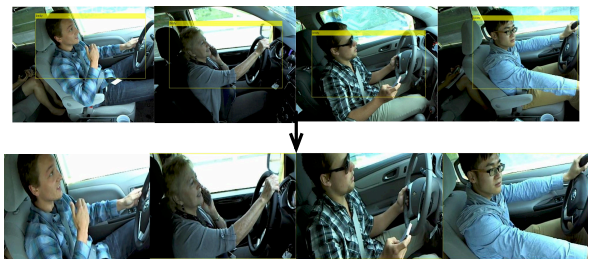

Fig. 3. Cropped Image

2) Using PCA: Principal Component Analysis (PCA) is known to be used for dimensionality reduction–but it also can be used to decorrelate the data. Clearly, pixel values are highly correlated in raw pictures. The idea is that if we can decrease the number of highly correlated pixels, the small objects can contribute more to the decision. After normalization, We perform the randomized PCA algorithm [16] to the training images for each color space, and apply the same transform to the validation set to make predictions. The number of principal components (PCs) is determined by maintaining above 95% variance of the original data. The output of PCA are then fed to the SVC classifier for training and testing.

B. Convolution Neural Network

As stated in section II, CNN has outstanding performance in computer vision task, thus there is no reason we don't try this method. The advantage of using CNN is that it can automatically learn complex feature utilizing massive simple neurons and back-propagation. CNN has many types of layers. For our interest, some key layers are: Convolution (Conv) layer (multiple convolution filters that obtain the visual pattern), Pooling layer (down-sampling by taking max or average and control over-fitting), Dropout layer (randomly drop some units, control over-fitting), and Fully-connected (FC) layer (preserve full information or make prediction). We build three models using CNN methods.

1) Simple CNN: The first model is built based on the entry task of CNN: MNIST hand-written digit classification [17]. The layer is cascaded as in Fig. 4. We use this to serve as a baseline for the performance of CNN in our own problem. The input to this model is the mean normalized images.

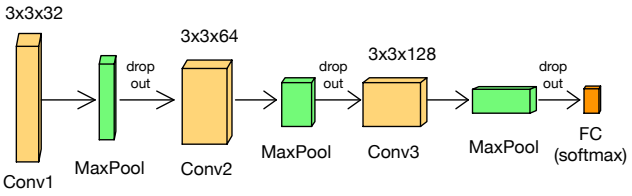

Fig. 4. Structure of scratched CNN

2) Transfer Learning on VGG-16: Training a CNN from scratch is very difficult, as tuning neural network is a highly non-convex problem. Thus, transfer learning has become a very popular way in practice. Our second model is built based on pre-trained VGG-16, since the model is easy to build as stated in section II. The network architecture is shown below. Compared to the previous model, VGG-16 has much larger depths, which is proved to be a critical component for good performance [9]. The relu rectifier is a threshold operator, providing non-linearity to the network. To do transfer learning, we remove the last softmax layer of VGG16, and insert a new 10-class softmax classifier. We also set the first five layers to be non-trainable to mitigate over-fitting motivated by [18].

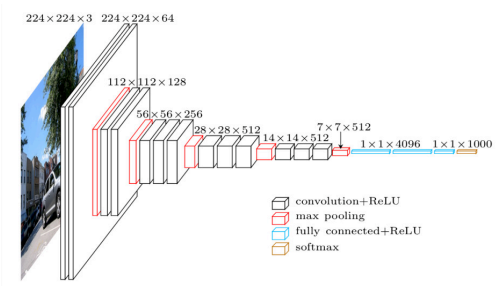

Fig. 5. Structure of VGG16 [19]

3) VGG-GAP: The last model we tried is called VGG-GAP [4]. This model removes the fully-connected layers after conv5 of VGG, resulting in a resolution of 14x14; then add an additional conv layers with 1024 filters of 3x3, stride 1 pad 1 and a global average pooling (GAP) layers (14x14) just before softmax. This special architecture (figure 6) enables us to visualize the informative region used by the network. Mathematically, consider the output of conv6 after batch normalization $f_k(x, y)$, where k is the index of convolution filter, the result after GAP F_k is $\sum_{x,y} f_k(x, y)$. The values are then fed into softmax classifier to compute score $S_c = \sum_k w_k^c F_k$ for each class c, where w_k^c is the weight of filter k for class c. By mapping back the weights directly to $f_k(x, y)$ and compute $M_c(x, y) = \sum_k w_k^c f_k(x, y)$, we then know how informative (x, y) is in predicting class c. Intuitively, we can think about the output of conv6 is a set of activation units which tell us the presence of some visual patterns. The GAP and softmax then give us weights for each pattern in predicting specific class. Then for each spatial location we compute the weighted linear sum regards to the unit and we can get a class activation map (CAM) showing the important region for class c.

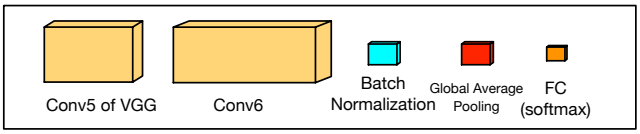

Fig. 6. Structure of VGG-GAP; Batch normalization is added to speed up convergence and improve performance

V. Experiments and Results

A. Evaluation Metric

We introduce the log loss evaluation, given by:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \log p_{ij} \quad (2)$$

where N is the number of predictions, M is the number of classes, y_{ij} is a binary number indicating whether class j is the correct label of i, and p_{ij} is the predicted probability. The log loss extremely punish the predictions saying definitely true (or false) when the actual label is false (or true). Note that for a random guessing classifier, the log loss is $\log M$, which is about 2.3 given $M = 10$ in our task. The log loss is used by Kaggle to evaluate the submission. Besides log loss, we will also use accuracy (correct prediction/total prediction) and confusion matrix to give us more information on the performance, since they are easier to interpret.

B. SVC

For the pre-processing step, we choose 3-stage cascade. The limitation comes from the fact that it is hard to collect negative samples that has similar background as the training samples. At the end, we are only able to find around 50 car inside images that film the driver seat from the copilot seat. Together

with some other car images, the cascade classifier are able to achieve a false positive rate of 0.1 at each stage. For tuning the SVC classifier, we do a grid-search [20] with 5-Fold cross validation (each fold is separated as described in section III) to optimize the log loss. We limit our candidates by choosing $C \in \{1, 10, 100, 1000\}, \gamma \in \{10^{-3}, 10^{-2}\}$ and $Kernel \in$ {RBF, Linear}. During our experiments, we found that RBF kernel shape always performed better in this task, while C and γ vary according to different model. Our intuition is that image data is highly-correlated (many parts of the image is symmetric) and it is almost linearly inseparable. We test SVC with several settings with optimized hyper-parameters given by grid-search, and the results are shown below. In all cases, processed images are resized to be 224x224x3.

TABLE I
SVC GRID SEARCH WITH 5-FOLDS

Model	Accuracy	Log Loss	Kaggle Score
Pixel SVC	18.3%	1.94	1.98
SVC + HOG	28.2%	1.82	1.85
SVC + PCA	34.8%	1.73	1.69
SVC + Bbox + PCA	40.7%	1.47	1.53

The first observation was that using SVC with HOG feature was not so powerful in this task. When we looked at the HOG map, we found that the HOG map is very sensitive to the image size. For image of 224x224, the resulting HOG map shows very little difference for small object such as water cup and phone, not to mention the facial elements. As a result, the classifier has difficulty in recognizing the object. Another surprising result is that a simple PCA can improve the result significantly, while also reduce the data space and increase the computation efficiency. The improvement of using bounding box to crop the image is also satisfactory, though not very large. We show confusion matrix for one of the five folds validation results of the Bbox-PCA-SVC model below. Clearly, we can see that the classifier has difficulty in

```
[[200   1   0  40   8  17   0   0  63  59]
 [  7  80   3   0   0  56   0 131  40  66]
 [123   0  33   0   0  72   0  89  44   3]
 [ 85   0   0  40   0  45   0   0  32 155]
 [116   0   0   1 172   9   0   0  45   8]
 [  1   0   1   0   0 309   2   0   0  18]
 [ 50  13   2   0   0  64 110   0  45  71]
 [ 61   0   0   0   1  27   1 154   2  18]
 [  7  10  28   0   0  10  38  24 167   3]
 [ 52   0   0  38   0  37   0   0 115  97]]
```

Fig. 7. Confusion Matrix for SVC

differentiating small object class (1-4, all about using phones). It also make many mistakes in classifying the last class (c9) talking as well.

C. Convolution Neural Network

The required computation power of CNN is huge. Therefore, we launched an ECS AWS instance to speed up the prototyping. The instance is g2.8xlarge with GPU of GRID K520. The GPU only has memory of 4G, which places strict limitations on the batch size. We use Keras library [21] with Theano backend [22] for the implementation.

In all three CNN models, we chose Adam [23] to be the gradient optimizer, which is considered preferably to SGD since it adapt the learning rate to the weights of features. We also use early-stopping technique [21], which monitors the validation loss and will stop the training process if results are not improved in 5 successive epochs. For pre-processing, we only do mean normalization with random rotate +/-10 degree. The reason we do not use Bounding box is that we are not confident enough with the false positive rate of 0.1 here. In fact, in some test image we did see some examples that the bounding box would wrongly discard the key components such as water cup. We also do not use PCA, since CNN needs to learn high-level visual pattern features and PCA will completely destroy the visual connections. For scratched CNN, the mean normalization is respect to the training data, while for transfer learning, we need to do exactly the same normalization as the original model. Besides the network architecture mentioned above, other hyper-parameters are initialized as follows:
For scratched CNN:

- learning rate: $\gamma = 0.001$
- Weights: Normal distributed
- training epoch: 20
- batch size: 1

For transfer learning:

- learning rate: $\gamma = 10^{-5}$
- Weights: Pre-trained VGG-16 model weights
- training epoch: 25
- batch size: 16

Learning rate for transfer learning is typically much smaller since we are fine-tuning the model. We also find that larger batch size is better. Recall that batch size indicates how frequently the network update its parameters. Thus if batch size is 1, then the network risks to overfit on a particular image. However, 16 is the largest we can use due to the hardware limitation. Finally, for each of the three CNN models, we use 5-fold cross-validation to optimize the use of training set and obtain local score. It took me around one hour per epoch for the transfer learning models. In total the three models took me about 7 days for training and testing. The results are as follows:

Surprisingly, we see that a simple CNN network is already outperforming the previous best SVC. When we goes to deeper and using VGG network, the results are very intriguing—we are able to achieve a local accuracy more than 90% and very small log loss. Specifically, after mean ensemble VGG-16 and VGG-GAP, the log loss goes down to 0.24, and this score gives us a Kaggle placement of 122/1440, which is top 8%. For reference, the championship achieves a log loss of 0.08.
To better understand the behavior of VGG model, we show

TABLE II
PERFORMANCE OF CNN

Model	Accuracy	Log Loss	Kaggle Score
Scratched CNN	63.3%	0.98	1.02
VGG-16	90.2%	0.28	0.34
VGG-GAP	91.3%	0.27	0.32
Ensemble VGG-16 & VGG-GAP	**92.6%**	**0.23**	**0.24**

the training curves and confusion matrix as below. We can see

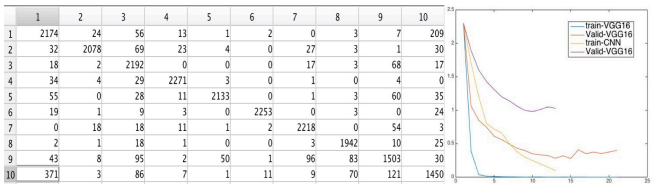

Fig. 8. Confusion Matrix and Learning curve for VGG-16

that the biggest confusion for the network now is to identify (c0) safe driving and (c9) talking. However, by looking back to the training set, we found that this two classes are intrinsically very similar, some of them cannot even be classified correctly by human. We show some mis-leading images as in figure 9. Also, from the learning curve we can see that after about 10 epochs, overfitting occurs for VGG-16.

Fig. 9. Confusing training samples: the top two are labeled as talking, and the bottom two are labeled as safe driving, while they do look the same

We also use the VGG-GAP model to generate class activation map for even better understanding the model behaviors (see figure 10). From the top middle image, we are happy to see that the network has figured out that the legs on the backseat is uninformative. We can also see that when the network find some unseen objects, for example, the clothes pattern and the paper in backseat of the top rightmost figure, and black curly hair in bottom rightmost, it feels confused or makes mistakes. This tells us that when there is presence of object, instead of classifying based on drivers action, the network is somehow more like a object detector. Although it may be able to do well in this test set, ideally we want the network to focus on action instead of objects. One way to mitigate this issue is to have some training samples in the safe driving class where there is presence of those small objects but the driver is doing safe driving. This can be achieved by using data argumentation technique (e.g. copy those objects and add to the safe driving images). Unluckily, we don't have enough time to do this.

Another issue is that the training data size is very small

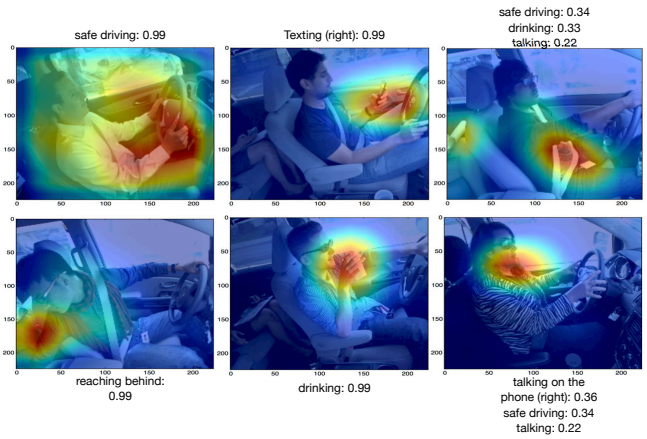

Fig. 10. CAM: from left to right, the first two column is correctly classified samples. The last column is some of confusing/mis-classified samples. The last top show the CAM for drinking, the bottom show CAM for phone talking

compared to typical image classification task. As we can see in the learning curve, the training loss quickly goes down to very low and hence little learning has been obtained after around 10 epochs. In fact, this is exactly the reason that the network made mistakes in Figure 10. Our attempt to address this issue is using semi-supervised technique called pseudo label [3] to utilize the test set (there is 80,000 images!). The procedure is simple: we pick the class which has the largest predicted probability of each test image as the 'true' label, and fine-tune the model with these additional data. For a learning rate of 10^{-6} and 2-epoch full pass, the new model is able to achieve Kaggle score of **0.22**, which is **top 6%**.

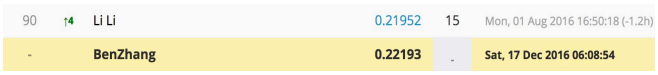

Fig. 11. final Kaggle score

VI. CONCLUSION AND FUTURE WORK

In this project, we used two methods to classify drivers' behaviors based on the input images: SVC and CNN. For SVC, we trained a bounding box and used PCA to distinguish the small objects, which is able to get a Kaggle score of 1.53. For CNN, transfer learning on VGG shows its power and with mean ensemble and pseudo label, the model is able to get a Kaggle score of 0.22, which is significantly better than SVC. However, overfitting exists and surely there is further improvements we can do. On the model side, it will be worth trying use the state-of-art model inception v4; on the data side, more data argumentation is highly needed. For example, as mentioned in the previous section, copying the small objects to safe driving classes would be a good way to prevent overfitting on object. Also, more advanced semi-supervised technique such as Denoising Auto-Encoder or dropout [3] is a good way to infuse the training set as well.

VII. Reference

[1] National Center for Statistics and Analysis, Distracted Driving: 2013 Data, in Traffic Safety Research Notes. DOT HS 812 132. April 2015, National Highway Traffic Safety Administration: Washington, D.C.

[2] Kaggle Competition Site: https://www.kaggle.com/c/state-farm-distracted-driver-detection

[3] Lee, D. H. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In Workshop on Challenges in Representation Learning, ICML (Vol. 3, p. 2).

[4] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2015). Learning Deep Features for Discriminative Localization. arXiv preprint arXiv:1512.04150.

[5] Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) (Vol. 1, pp. 886-893). IEEE.

[6] Ebrahimzadeh, R., & Jampour, M. (2014). Efficient handwritten digit recognition based on histogram of oriented gradients and svm. International Journal of Computer Applications, 104(9).

[7] Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., & Fergus, R. (2013). Regularization of neural networks using dropconnect. In Proceedings of the 30th International Conference on Machine Learning (ICML-13) (pp. 1058-1066).

[8] Szegedy, C., Ioffe, S., & Vanhoucke, V. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv preprint arXiv:1602.07261.

[9] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[10] Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In European Conference on Computer Vision (pp. 818-833). Springer International Publishing.

[11] Mahendran, A., & Vedaldi, A. (2015, June). Understanding deep image representations by inverting them. In 2015 IEEE conference on computer vision and pattern recognition (CVPR) (pp. 5188-5196). IEEE.

[12] Singh, D. (2016). Using Convolutional Neural Networks to Perform Classification on State Farm Insurance Driver Images. [online] Available at: http://cs229.stanford.edu/proj2016spr/report/004.pdf [Accessed 24 Nov. 2016]

[13] Matlab Image Labeler. (2013). https://www.mathworks.com/help/vision/ug/label-images-for-classification-model-training.html

[14] Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on (Vol. 1, pp. I-511). IEEE.

[15] Matlab Train Cascade Object Detector. (2013). https://www.mathworks.com/help/vision/ref/traincascadeobjectdetector.html

[16] Halko, N., Martinsson, P. G., & Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM review, 53(2), 217-288.

[17] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[18] Cs231n.github.io. (2016). CS231n Convolutional Neural Networks for Visual Recognition. [online] Available at: http://cs231n.github.io/convolutional-networks/ [Accessed 17 Dec. 2016].

[19] heuritech - le blog. (2016). A brief report of the Heuritech Deep Learning Meetup #5. [online] Available at: https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/ [Accessed 17 Dec. 2016].

[20] Scikit-learn.org. (2016). 3.2. Tuning the hyper-parameters of an estimator scikit-learn 0.18.1 documentation. [online] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html [Accessed 17 Dec. 2016].

[21] Keras.io. (2016). Keras Documentation. [online] Available at: https://keras.io/ [Accessed 17 Dec. 2016].

[22] Team, T. T. D., Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., ... & Belopolsky, A. (2016). Theano: A Python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688.

[23] Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.