Object classification for autonomous vehicle navigation of Stanford campus

Heather Blundell and Sarah M. Thornton
Email: {hrblun,smthorn}@stanford.edu

***Abstract*—With automated vehicle technologies rapidly advancing, fully automated vehicles may not be far behind. For an autonomous vehicle to properly navigate the environment, it will need information about objects it is likely to encounter. An affordable and popular source for this information is taken from camera images. We apply and compare different supervised learning techniques to images for object classification. Particularly, we investigate softmax regression, support vector machines and convolutional neural networks. We generate baseline results using the CIFAR database, and then move to a larger dataset from ImageNet to improve model accuracy. The models trained and validated on ImageNet are then tested against our own test data generated from GoPro footage of driving around the Stanford campus.**

I. Introduction

As automotive manufacturers incorporate more and more automation into passenger vehicles, it seems inevitable that fully automated vehicles will soon be upon us. Many autonomous vehicle platforms [1] are outfitted with cameras, which are low cost devices that capture the surrounding environment similarly to human eyes. Camera images capture rich information about the environment an autonomous vehicle traverses.

We propose an object classification problem using camera images of objects an autonomous vehicle is likely to encounter. In particular, we bound the driving scenario to navigating Stanford campus around other vehicles and students. To accomplish this task, we will compare different supervised learning techniques with varying features for image recognition. One supervised learning technique we investigate is a support vector machine (SVM), which allows for high-dimensional feature kernels in case data is not linearly separable. Another supervised learning technique we propose to use is softmax regression because it is a generalized logistic regression algorithm and the output is a probability of the label. In addition to comparing these two learning techniques, we will discuss the impact of varying features to the learning algorithms. In particular, we will look at comparing the features of RGB values and gray-scale values. Lastly, we will discuss an alternative supervised learning approach using a Convolutional Neural Network, which does not require manual specification of features.

In order to focus the problem on the goal of assisting autonomous vehicle navigation on campus, the project will be limited to classification of items most likely to be found around the Stanford campus. We include baseline results from training and validating on the CIFAR dataset for nine classes: bus, car, bicyclist, motorcycle, pickup truck, construction truck, CalTrain, pedestrian and tree. We also form results from training and validating on an ImageNet dataset for five superclasses: bicycle, people, sign, tree and vehicle. The ImageNet models are tested against images we gathered from GoPro cameras while driving around the Stanford campus.

II. Related Work

While the work we present in this paper looks at object classification for a broad range of objects, there are those that focused on an even smaller subset of computer vision algorithms for autonomous vehicles. For example, traffic signs are part of the rich information captured by a camera mounted in a vehicle. Autonomous vehicles can take advantage of the current infrastructure if they are able to properly identify and read these signs. Through the use of a genetic algorithm and neural network, de la Escalera et al. [2] demonstrate the ability to not only recognize and classify traffic signs but also the condition of the sign at various times of day.

Camera images also capture other vehicles as well as vulnerable road users such as pedestrians and bicyclists. When cameras are mounted to the infrastructure, Messelodi et al. [3] leverage minimal changes to the background and road plane in order to create a real-time vision system capable of detecting and tracking vehicles on the road and estimate the respective speed. They classify seven categories of bicycle, motorcycle, car, van, truck, urban bus and extra-urban bus, and use a combination of model-based and feature-based techniques to help distinguish between similar models such as motorcycles and bicycles. Since autonomous vehicles may require traveling through non-urban intersections, Gavrila [4] uses a vehicle mounted camera to detect pedestrians by implementing a real-time shape-based object detection system which extends the Chamfer System and filters false-positives using a radial basis function based verification method.

More recently, standards for benchmarking object classification have surfaced through machine learning competitions such as Kaggle [5] and the Pascal Visual Object Classes competition [6]. Although these competitions require a larger number of classes to identify, they showcase advances in machine learning to this greater variability in the dataset. For ImageNet, both Litayem et al. [7] and Maji et al. [8] use support vector machines. Krizhevsky et al. [9] show convolutional neural networks perform well on ImageNet. In the following sections, we show results of both support vector machines and convolutional neural networks on ImageNet.

III. Datasets

A. CIFAR

For our baseline object classification dataset, we use select classes from CIFAR-10 and CIFAR-100 [10]. CIFAR-10 includes 32×32 color images of 10 classes with 6000 images per class, while CIFAR-100 includes 100 classes but with only 600 images per class. Since we are using *both* datasets, we chose to select the minimum size of images per class to prevent a skewed dataset. Thus, we only sample 600 images per class out of the available 6000 from CIFAR-10.

Since we are focusing on the application to autonomous vehicles, we only use classes representative of likely objects the vehicle may encounter on the road. From CIFAR-10, we use classes 'automobile' and 'truck'. From CIFAR-100, we use nine classes: 'people', 'trees', 'bicycle', 'bus', 'motorcycle', 'pickup_truck' and 'train'. For each of these classes, we partition our 600 available images for that class into 500 images for training and 100 images for validation testing.

B. ImageNet

For a larger number of examples per class, we gather images from ImageNet [11]. In our project milestone, we found the number of classes and number of examples per class spread the data too thin for proper object classification. Thus, for the ImageNet data, we collapse the number of classes to five superclasses: bicycle, people, sign, tree and vehicle. Even though ImageNet provides us with many more examples, they are not consistently processed like CIFAR. We address this by scaling the images to 128 pixels and then cropping them into 128×128 images. For our training set, we gather 4,500 images per class. The validation set contains 1,000 images per class.

C. GoPro Footage

Both the CIFAR and ImageNet datasets are used for training and validation. We created our own dataset for testing. To do this, we mounted two GoPros to the interior of a vehicle. We drove the vehicle around the Stanford campus with the cameras recording at 60 frames/second for about one hour. Providing us with a total of approximately two hours of footage. In processing of the footage, we grabbed one out of every 10 frames for analysis. We selected images that captured objects in the five superclasses used from ImageNet: bicycle, people, sign, tree and vehicle. Each of these images were then passed through a segmentation algorithm [12] to further parse them into the desired classes. The segmentation process outputs the images in various resolutsions and sizes, so similarly to the ImageNet dataset, we scaled and cropped them to 128×128 pixel images. We successfully assembled 600 images per superclass. The number of examples from each dataset are summarized in Table I.

TABLE I: Dataset sizes per class

Dataset	Train	Validate	Test
CIFAR	500	100	-
ImageNet	4,500	1,000	-
GoPro	-	-	600

IV. Softmax Regression and SVM

A. Features

In this report, we only look at using RGB values and gray-scale values as features for softmax regression and an SVM.

1) RGB: The RGB values are the default feature provided by CIFAR and ImageNet, and are provided as three 8-bit values for each pixel. For the rest of the paper, we focus on images gathered from ImageNet and will refer to CIFAR for comparison with our milestone report. Since the images are 128×128 RGB images, each feature $x^{(i)}$ in the training set $\in \mathbb{R}^{49,152}$.

2) Gray-scale: To obtain the gray-scale information, we transform the RGB values using a weighted sum of the R, G and B components [13]:

$$x^{(i)} = 0.2989R^{(i)} + 0.5870G^{(i)} + 0.1140B^{(i)}$$

where $R^{(i)}$, $G^{(i)}$ and $B^{(i)}$ are the $\in \mathbb{R}^{16,384}$ vectors corresponding to red, green and blue values of the pixels for feature $x^{(i)}$. Thus, each gray-scale feature $x^{(i)}$ in the training set is now $\in \mathbb{R}^{16,384}$.

B. Softmax Regression

Object classification of five superclasses is a multi-class classification problem, so we turn to an inherently multi-class algorithm known as softmax regression. Softmax regression is logistic regression for multinomial data. We use the Python library scikit-learn [14] to train a softmax regression model and use it to predict on our validation set. Since we are using `LogisticRegression()` from scikit-learn with the multinomial setting, it uses solvers that only support L2 regularization with primal formulation. Our softmax regression problem has the following form (with weight decay):

$$J(\theta) = -\frac{1}{m}\left[\sum_{i,j=1}^{m,k} 1\left\{y^{(i)} = j\right\} \log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right] + \frac{\lambda}{2}\sum_{i,j=1}^{k,n} \theta_{ij}^2.$$

1) RGB: Using 4,500 training examples from each class, we trained a softmax regression model and obtained an accuracy of 54.92%. When the model is tested on the 1,000 example validation set, we see a degradation in performance by almost 8% with an accuracy of 47.23%. On the CIFAR dataset, we had a degradation of about 18% between training and validation.

2) Gray-scale: In comparison, training a softmax regression model on gray-scale features obtained an accuracy of 40.31%. When tested on the validation set, the accuracy again decreased to 31.10%. Overall, using gray-scale features seems to lose information about the image compared to using RGB features.

C. SVM

Another approach to multi-class classification is "one-vs-rest," where one class is positive and all other classes are "negative." Thus, creating k number of models; one for each class. Using scikit-learn, multi-class functionality is built-in to their several SVM implementations. For the baseline, we

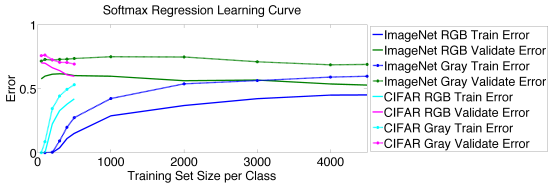


Fig. 1: Both training and validation errors start to converge to a large error for both CIFAR and ImageNet. Gray-scale valued features always perform worse than RGB valued features.

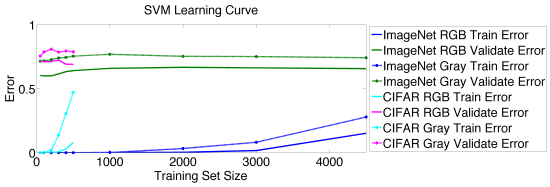
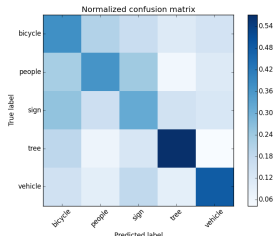
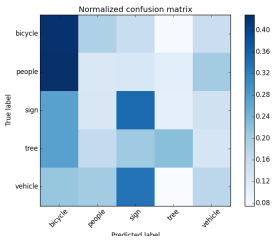


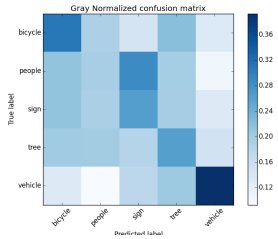
Fig. 3: Validation errors for both CIFAR and ImageNet remain high despite increase in training set size. Similarly to softmax, gray-scale features perform worse than RGB features.



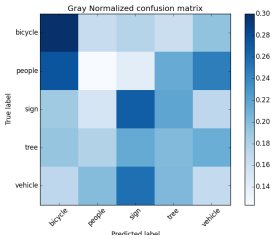
(a) RGB feature on validate set.



(b) RGB feature on test set.

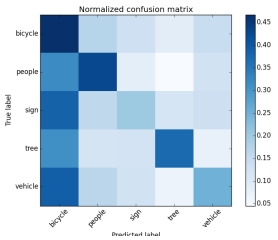


(c) Grayscale feature on validate set.

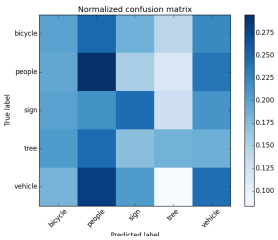


(d) Grayscale feature on test set.

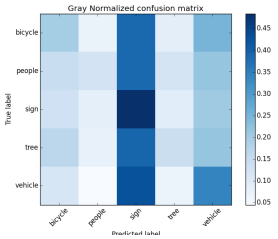
Fig. 2: Normalized confusion matrices for Softmax.



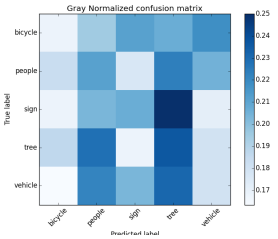
(a) RGB feature on validate set.



(b) RGB feature on test set.



(c) Grayscale feature on validate set.



(d) Grayscale feature on test set.

Fig. 4: Normalized confusion matrices for SVM.

use `LinearSVC()`, which defaults to using the linear kernel. The other default settings use L2 regularization with dual formulation and $C = 1$, so we have the following problem formulation:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0. \end{aligned}$$

1) RGB: We train this linear kernel SVM using the 4,500 training examples from each class and achieve an accuracy of 84.88%. Unfortunately, the validation set produces an accuracy of 34.37%, which indicates our SVM is over-fitting when using RGB features.

2) Gray-scale: Comparing with gray-scale features, we achieve an accuracy of 72.17% using the training examples. The validation set obtained an accuracy of 25.87%. The results of gray-scale features to RGB features, again, shows better performance with using RGB values.

V. Convolutional Neural Networks

Besides Support Vector Machines implicitly representing high-dimensional features, we can also automatically learn the image features by training a Convolutional Neural Network (CNN). Given that naturalistic stimuli in images are multiparametric, CNN models can learn many parameters and dynamics not captured by simpler models. Moreover, overlapping receptive fields in the convolutional layer allow for sensitivity to small sub-regions of the i. Note that in this image classification task, our oracle would be for the self-driving car's visual classification system to match the precision of a human's manual image classification.

We implemented an eight-layer CNN architecture to classify our image dataset using the Keras Theano-based Deep Learning Library [15], which achieves nearly 100% accuracy on our larger ImageNet training set and also generalizes well on our held-out validation set data. A detailed description of our model architecture (layer-by-layer) is given as follows:

A. Architecture

Many competing factors influenced our model decisions, of which the most important consideration was avoiding overfitting to the training data. It was necessary to use a model with complexity on par with the complexity of the training data. Hence, as we increased the size of our dataset with more ImageNet images and higher resolution (128×128), we found our models to be less prone to overfitting than with our small CIFAR dataset. We based our model on a VGG-style

CNN, but ran many experiments to tune the parameters to fit our 5-class ImageNet dataset. The VGG-style architecture of Simonyan and Zisserman [16] found that the depth of a CNN is important to its accuracy in the large-scale image recognition setting, and they also used very small (3×3) convolution filters. Therefore, our architecture has four 2D convolutional layers with small (3×3) filters. The first such layer consists of 32 (3×3) convolutional filters. Next, our model applies a ReLU nonlinearity, defined by $\text{ReLU}(x) = \max\{0, x\}$, which helps to ensure sparse activations. This layer is immediately followed by another 2D convolutional layer and a ReLU nonlinearity, followed by a max pooling layer which down-samples the input by 2 along both the width and height dimensions. The function of the max pooling layer is to reduce the spatial size of our input, which reduces the number of parameters, and hence controls overfitting. We next use a 2D convolutional layer but with more filters (64 3×3 filters) followed by a ReLU, followed by another 2D convolutional layer of this size and a ReLU, followed by a max pooling layer. Finally, we flatten the output of the last max pooling layer into a vector and pass it through two fully connected layers, the first of which has 356 neurons and the last fully connected layer has 5 neurons (the number of classes) in order to output the CNN's predictions $\hat{y}$. In total, our CNN architecture is relatively deep, consisting of 8 layers, as described above. Our objective function was categorical cross-entropy, defined for the truth $y \in \mathbb{R}^n$ and the CNN's prediction $\hat{y} \in \mathbb{R}^n$, where n is the number of classes, as

$$\text{CCE}(y, \hat{y}) = -\sum_{j=1}^{n} y_j \log(\hat{y}_j).$$

which is a good loss function for multi-class classification problems and softmax output units.

B. Weight Initialization and Optimizer

As these are both critical for the CNN's performance (especially considering that the objective function is non-convex), we experimented with several choices of weight initialization and optimizer. Initially for our choice of weight initialization, we used Glorot uniform [17], a commonly used initialization. If we let n_{in} and n_{out} be the number of neurons feeding into and feeding out of the weights of a layer, respectively, then Glorot uniform initializes these weights randomly according to a Uniform distribution scaled by $1/(n_{in}+n_{out})$. However, we found that what worked best was a He uniform initialization, recently introduced by He et al. [18], which initializes the weights according to a Uniform distribution scaled by $\dfrac{1}{n_{in}}$. The authors report that this type of initialization works better for ReLU nonlinearities (which our model uses), whereas Glorot uniform is more appropriate for sigmoid and tanh nonlinearities. Without this uniformly random initialization of weights in the first convolutional layer, we found that our model would classify nearly all training (and validation) images into one or two of the five classes, which we believe was a problem with a local optimum. Also, to further address this problem, we used SGD with momentum as our optimizer. The results of Sutskever et. al [19] suggest that first-order Nesterov momentum methods are key to avoiding SGD getting stuck in poor *local* optima. Momentum serves as an additional term added to the SGD gradient updates to help speed up convergence and make SGD more stable by avoiding the problem of vanishing gradients. With momentum μ, SGD updates the parameters θ of an objective function $J(\theta)$ as follows:

$$v_{t+1} = \mu v_t - \eta \nabla J(\theta_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

We found that the Nesterov momentum parameter μ that gave our model the best generalization error was 0.95 when our SGD learning rate was 0.01 with decay 10^{-6}, and using a mini-batch size of 200. We initially experimented with mini-batch sizes of 32 and 400, but found this intermediate mini-batch size to work best since it is small enough to help avoid poor local optima, but also large enough to avoid difficulty converging to a good optima due to gradient noise.

C. Decisions To Control Overfitting and Biases

To prevent overfitting, we used Dropout, recently introduced by [20], after two of our 2D Convolutional layers during training. More importantly, we used Dropout after our large 356-neuron fully-connected layer, as this layer is responsible for most of the parameters in our CNN. Here, we set 0.5 of input units randomly to 0 at each update during training time, which helped to balance the complexity of our parameters with the nature of our ImageNet dataset. We also applied dropout after two of our 2D Convolutional layers, but at a probability of only 0.25 instead of 0.5 because the parameters introduced by these layers are fewer in number. However, the CNN still overfit slightly to the training data, which we hypothesize was due to its nevertheless large number of parameters.

Moreover, we used an L2-weight regularization penalty of 0.01 on both of our dense layers. As mentioned in the previous section, our CNN was more biased to sparse representations in its image classification. From our observations, adding L2-regularization discouraged this imbalanced classification, giving us confusion matrices with few images in our datasets predicted off of the diagonal (see Fig. 7).

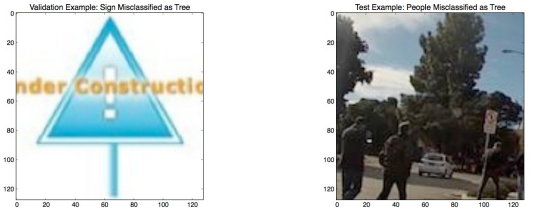


Fig. 5: Example CNN Misclassifications. On the left, a validation example of a sign misclassified as a tree. On the right, a test example of people misclassified as a tree.

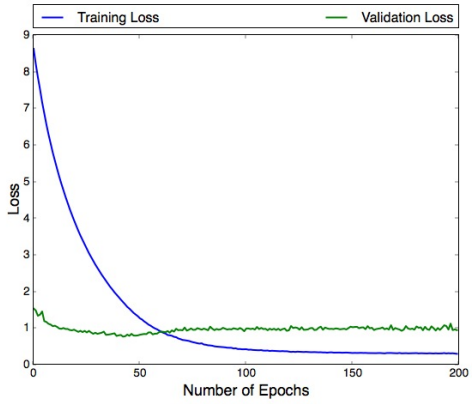
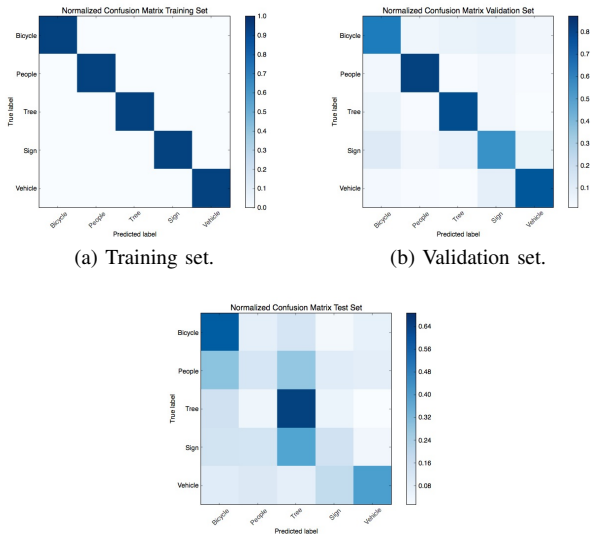


Fig. 6: CNN Training Loss decreases over 200 iterations, especially during the first 50 iterations. The CNN's Validation Loss also decreases in the first 50 iterations, and then increases only slightly due to overfitting.



(a) Training set.

(b) Validation set.

(c) Test set.

Fig. 7: Normalized confusion matrices for CNN.

VI. Discussion

For softmax regression and SVM, we notice a trend when comparing using RGB values and gray-scale values for features in that RGB values overall outperform using gray-scale values as seen in Figs. 1 and 3. When looking at training vs. validation error for softmax regression compared to SVM, we see softmax regression has less of a discrepancy between the two errors. This suggests our softmax regression model suffers from high bias. We can see the SVM has high variance because the training error is significantly lower than the test error. Even though we increased the set size from CIFAR to ImageNet, there is no improvement in the validation error. This suggests the need to explore other feature representations of the data; most likely one independent of scaling or rotation of objects in the image patch such as SIFT [21] or SURF [22].

As for our CNN, although it started to overfit slightly around 50 iterations, the decreasing training loss of our model demonstrates that it can most effectively learn from our training set given its hyperparameters. However, when tested on our Stanford GoPro footage, the CNN had very low accuracy on people and signs (see Fig. 7c), predicting most people as bicycles or trees and predicting most signs as trees. This behavior is likely because, given the nature of Stanford's campus, many of our test images featured people riding bicycles or walking by trees and most signs are by trees. So, this lack of mutual exclusivity of our classes with respect to our test set made choosing a single class difficult.

TABLE II: Accuracy for Different Models on ImageNet

Model	Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
Softmax RGB	84.9	34.4	26.0
Softmax Gray	72.2	25.9	21.2
SVM RGB	54.9	47.2	23.4
SVM Gray	40.3	31.1	20.1
CNN	99.9	76.2	39.0

A comparison of the CNN and all the models applied in the paper is summarized in Table II. We see CNN performed the best with an almost perfect training accuracy and almost double the validation accuracy of the other models. Since the test accuracy of the CNN and the other models was not particularly high, this suggests more post-processing of the GoPro footage and our datasets would improve the scores. With more computational resources, our future work would consist of performing ZCA-whitening on the datasets which is shown to reduce high correlation between adjacent pixels [23]. As can be seen in Fig. 5, even our CNN had difficulty classifying images that contained multiple classes in one image. Even though we labeled the right image in Fig. 5 as people, we could have instead labeled it as tree. The fact that the CNN labeled it as tree gives insight into how its predictions depend on what are the most prominent features of the image (here, the tree is largest). For future work, we could help to resolve this issue by further cropping the images or using bounding boxes.

VII. Conclusion

We produced results for object classification of five super-classes of objects an autonomous vehicle may encounter while driving along Stanford campus by training and validating on ImageNet and testing on our set gathered from GoPro footage. We analyzed three supervised learning techniques: softmax regression, support vector machines and convolutional neural networks. Our 8-layer CNN exhibited the best classification accuracy of our three methods, in comparison to RGB and grayscale features for softmax and SVM. For our desired application, it is critical that an autonomous vehicle is able to classify unseen examples with high accuracy, and our CNN model is able to to do that on the validation set. To put this onboard an actual vehicle, we would need to further improve processing of real-time images to pass to the CNN for proper classification. To be safe on the road, an autonomous vehicle should be able to recognize multiple objects in a single image.

REFERENCES

[1] J. M. Gitlin, "Face to face with fords self-driving fusion hybrid research vehicles," August 6, 2015. [Online]. Available: http://arstechnica.com/cars/2015/08/face-to-face-with-fords-self-driving-fusion-hybrid-research-vehicles/

[2] A. De la Escalera, J. M. Armingol, and M. Mata, "Traffic sign recognition and analysis for intelligent vehicles," *Image and vision computing*, vol. 21, no. 3, pp. 247–258, 2003.

[3] S. Messelodi, C. M. Modena, and M. Zanin, "A computer vision system for the detection and classification of vehicles at urban road intersections," *Pattern analysis and applications*, vol. 8, no. 1-2, pp. 17–31, 2005.

[4] D. M. Gavrila, "Pedestrian detection from a moving vehicle," in *Computer VisionECCV 2000*. Springer, 2000, pp. 37–49.

[5] A. Goldbloom, "Data prediction competitions–far more than just a bit of fun," in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1385–1386.

[6] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[7] S. Litayem, A. Joly, and N. Boujemaa, "Hash-based support vector machines approximation for large scale prediction." in *BMVC*, 2012, pp. 1–11.

[8] S. Maji, A. C. Berg, and J. Malik, "Efficient classification for additive kernel svms," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 1, pp. 66–77, 2013.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[10] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[12] K. E. Van de Sande, J. R. Uijlings, T. Gevers, and A. W. Smeulders, "Segmentation as selective search for object recognition," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1879–1886.

[13] G. Bradski, *Dr. Dobb's Journal of Software Tools*.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[15] F. Chollet, "Keras: Theano-based deep learning library," *Code: https://github. com/fchollet. Documentation: http://keras. io*.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *arXiv preprint arXiv:1502.01852*, 2015.

[19] I. Sutskever, J. Martens, G. E. Dahl, , and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, vol. 28. ICML, 2013, pp. 1139–1147.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] Y. Ke and R. Sukthankar, "Pca-sift: A more distinctive representation for local image descriptors," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–506.

[22] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[23] A. Coates and A. Y. Ng, "Selecting receptive fields in deep networks," in *Advances in Neural Information Processing Systems*, 2011, pp. 2528–2536.