ImageNet Classification with SIFT + FV Representation and Baseline Classifiers

T1-09: ImageNet Classification with Deep Convolutional Neural Networks

Shalaby, Mohammed - 260607172 - mohammed.shalaby@mail.mcgill.ca
Hishon-Rezaizadeh, Ismael - 260627946 - ismael.hishon-rezaizadeh@mail.mcgill.ca
Manalad, James - 260743793 - james.manalad@mail.mcgill.ca

I. INTRODUCTION

Image classification is an area of considerable importance in the field of machine learning, and has been widely studied due to its relevance in various applications. The research within this field used to be inherently limited due to limited processing power and the need for easily available large datasets of millions of images. However, with recent improvements in processing power and the emergence of datasets such as ImageNet [1] and LabelMe [2], various researchers have been tackling image classification using different approaches for reasons such as computer vision and natural language processing applications.

The main goal of this project is to try and beat the performance of a highly complex and computationally expensive deep convolutional neural network [3] by fine-tuning their baselines and assessing other simple algorithms that have lesser computational complexity. This is to address a common issue of researchers spending more time fine-tuning their proposed new models, consequently neglecting the fact that the already present simple baselines may actually outperform their suggested model with proper fine-tuning. Therefore, this paper is essentially acting as an adversary to their highly complex model.

In this report, we begin with a literature review of the paper we are reproducing and trying to beat, as well as other papers of relevant work in Section II. Literature review was a particularly important part of this project due to the large dataset at hand and limited computational resources and therefore, literature review allowed narrowing down the methods to consider. We continue with an in depth discussion on the methodology for approaching the classification task; in particular, the preprocessing and representation learning steps, machine learning models used and the hyperparameter tuning approach are discussed in Section III. We present the results of the different models in Section IV, followed by a detailed discussion of the results and related decisions in Section V. As expected, we were able to match the performance of their highly complex deep convolutional neural network (CNN) at a fraction of the time by using a SIFT (Scale-Invariant Feature Transform) + FV (Fisher Vector) representation with logistic regression as our baseline classifier.

II. RELATED WORK

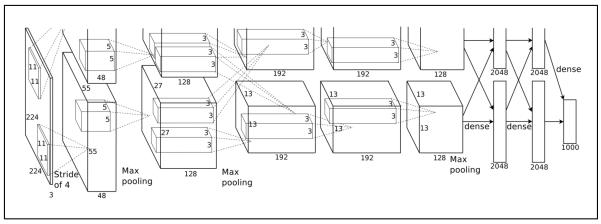

Fig. 1. An illustration of the architecture of Hinton *et al.*'s deep convolutional neural network [3].

We begin with a review of the research paper we are addressing for this project. Hinton *et al.* [3] trained a large, deep convolutional neural network, shown in Fig. 1, for the ImageNet LSVRC-2010 contest [4]. They were able to achieve a top-1 and top-5 error rates of 37.5% and 17.0% respectively, which are considerably better than the winner of the competition at the time. Their model consisted of 60 million parameters and 650,000 neurons, and consisted of five convolutional layers and three fully-connected layers with a final 1000-way softmax. Such a large neural network is highly susceptible to overfitting and has a very long learning time, therefore they implemented various different techniques to allow a more efficient network. They used Rectified Linear Units (ReLUs) to introduce a non-saturating linearity, which is a way of modelling the neuron's output f as a function of its input x with

$$f(x) = max(0,x)$$

which allows faster learning. They also applied local response normalization after applying ReLU non-linearity in certain layers, which they found helpful for generalization. In addition, other techniques used in their model include overlapped pooling to summarize the outputs of neighboring groups of neurons in the same kernel map, data augmentation using label-preserving transformations, inclusion of image translations and reflections, as well as variation of the intensities of the RGB channels using PCA (Principal

Component Analysis). They also used the dropout technique, which sets the output of each hidden neuron to 0 with a probability of 0.5, thus providing a very efficient version of model combination to reduce test errors. Despite all this, they reported that their model takes between five and six days to train on two GTX 580 3GB GPUs, which reflects the high complexity of this model. Our objective is to find a way to match their results within a limited period of time and without the need for a GPU.

TABLE I
COMPARISON OF RESULTS ON ILSVRC-2010 TEST SET

Model	Top-1	Top-5
SIFT + FVs + SGD	47.1%	28.2%
Proposed CNN	37.5%	17.0%

In this paper, the results achieved by their neural network is compared to the work of Sànchez *et al*. [5], as shown in Table I. The main focus of the latter's work is to address large scale classification accuracy as a function of the image signature dimensionality and the training set size, as well as to tackle the problem of data compression on very large signatures. They then report the results on the same dataset (ILSVRC-2010) as the original paper. However, the paper does not go in too much detail on the process of how the results were obtained. It rather places emphasis on how performance changes when the hyperparameters associated with the signature dimensionality and compression are varied. A more thorough discussion on the representation learning and classification steps can be found in another paper by Sànchez *et al*. [6]. There are two major steps for image classification: representation learning and classification model learning. By extracting a set of local patch descriptors from the images, encoding them into a high dimensional vector and pooling them into an image-level signature, we negate the need for a highly complex classification model even in the presence of large amounts of high dimensional data. The representation learning approach used by Sànchez *et al*. can be summarized in the following five steps:

1) Extraction of a set of local patch descriptors using SIFT
2) Dimensionality reduction of the SIFT descriptors using PCA
3) Learning of the parameters of a Gaussian Mixture Model (GMM) by optimizing a Maximum Likelihood criterion to represent a universal generative model
4) Encoding the descriptors into a high dimensional vector by using the FV patch encoding strategy
5) Compression of the dense Fisher Vectors by using Product Quantization

III. METHODOLOGY

We will start by discussing the dataset to be used in our model in Section III-A, followed in Section III-B by an overview of the representation learning techniques utilized. Section III-C then discusses the approach towards the selection of classifiers while Section III-D discusses the hyperparameters to be tuned.

A. Dataset

We will be using the same dataset as the one used by Hinton *et al*. for two reasons: it is the only ImageNet LSVRC contest with available testing data, and to allow a fair comparison with the results of their proposed model. The dataset is the one that was used in the ImageNet ILSVRC-2010 contest [4], and consists of approximately 1.2 million training images, 50,000 validation images, and 150,000 testing images split roughly equally among 1000 classes. This translates into 144GB of data: 124GB, 5GB and 15GB for training, validation and testing respectively. The images consist of RGB variable-resolution images, which might necessitate preprocessing to allow more efficient representation learning.

B. Representation Learning

The main preprocessing step performed was reducing the depth of the images from 3 to 1 by converting the coloured images to a grayscale format, which allows for more efficient learning. After that, local patch descriptors were extracted using SIFT followed by patch encoding using FV (discussed further in the following sections). In addition, no dimensionality reduction nor FV compression was used as the learned vectors were of 6400 dimensions per image, which is not outrageous even for baseline classifiers. It is also worth mentioning that due to limited time and storage available, and due to the size of the dataset, two scripts were run in parallel: one downloads one class with all its images while the other iterates through the downloaded class, learns the representation and deletes the folder from storage. Extracting SIFT descriptors, learning the GMM and the FV representations was all done with the help of the VLFeat [7] open source library implemented in MATLAB.

1) SIFT: SIFT stands for Scale-Invariant Feature Transform, which is a feature detection algorithm used to detect and describe local features in images, developed by Lowe at the University of British Columbia [8]. This technique was inspired by the responses of neurons in inferior temporal (IT) cortex in primate vision, which gives it its characteristic invariance to image translation, scaling, and rotation, while also being partially invariant to illumination changes and affine or 3D projection, thus deeming it a very efficient method for descriptor extraction.

The first stage of the SIFT process involves identifying the key locations in scale space by looking for locations that are maxima or minima of a difference-of-Gaussian function. The SIFT keys are created by generating a feature vector that describes the local image region sampled relative to its scale-space coordinate frame, and by burring image gradient locations, we achieve partial invariance to local variations. Consequently, the SIFT keys that are derived can then be used to identify candidate object models. This is done

by using a nearest-neighbour approach to indexing, where an object is recognized in a new image by individually comparing each feature from the new image to the identified SIFT keys, and therefore finding candidate matching features based on Euclidean distance of their feature vectors. Fig. 2 shows an example where by identifying SIFT keys from the top row, the nearest-neighbour approach allows us to efficiently extract those objects in a picture with various other objects.

Fig. 2. Top row shows model images for 3D objects with outlines found by background segmentation. Bottom image shows recognition results for 3D objects with model outlines and image keys used for matching [8].

2) Fisher Vectors: Fisher Vectors, also known as FV for short, is a patch encoding strategy technique that encodes the descriptors extracted using SIFT or any other feature extraction method into dense high dimensional vectors, by describing patches by their deviation from a "universal" generative Gaussian mixture model [9]. It is essentially an extension of the more common Bag-of-Visual (BoV) representation by encoding statistics up to the second order about the distribution of descriptors assigned to each visual word rather than using BoV's 0-order statistics.

As mentioned, the first step in learning FV representations is to learn a GMM model, which is a parametric probability density function represented as a weighted sum of Gaussian component densities. By specifying the number of Gaussians, the GMM model can be learned, which is a problem of unsupervised learning. The GMM u_λ is then a combination of the different Gaussians

$$u_\lambda = \sum_{i=1}^{N} w_i u_i$$

where

$$\lambda = \{w_i, \mu_i, \sigma_i, i = 1...N\}$$

and w_i, μ_i, and σ_i are the mixture weight, mean vector and standard deviation vector of Gaussian u_i.

The FV G_λ^X characterizes a sample X by its deviation from the distribution u_λ as

$$G_\lambda^X = L_\lambda G_\lambda^x$$

where G_λ^X is the gradient of the log-likelihood with respect to λ where

$$G_\lambda^X = \frac{1}{T} \bigtriangledown_\lambda log u_\lambda(X)$$

and L_λ is the Cholesky decomposition of the inverse of the Fisher information matrix F_λ of u_λ. In our case, $X = \{x_t, t = 1...T\}$ is the set of T local descriptors extracted using SIFT. Letting $\gamma_t(i)$ be the soft assignment of descriptor x_t to Gaussian i, and assuming that the descriptors x_t are independent and identically distributed, we obtain the following formulas for the gradients with respect to μ_i and σ_i:

$$G_{\mu,i}^x = \frac{1}{T\sqrt{w_i}} \sum_{t=1}^{T} \gamma_t(i) \frac{x_t - \mu_i}{\sigma_i},$$

$$G_{\sigma,i}^x = \frac{1}{T\sqrt{2w_i}} \sum_{t=1}^{T} \gamma_t(i) [\frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1].$$

The FV G_λ^X is the concatenation of the $G_{\mu,i}^x$ and $G_{\sigma,i}^x$ vectors.

C. Algorithms

After converting the input images into a more compact and uniform representation, the task of finding out which classifiers to fine-tune follows. To classify the ILSVRC-2010 dataset, the following baseline models were considered:

1) Logistic Regression
2) Linear SVM (Support Vector Machine) trained with SGD (Stochastic Gradient Descent) (**baseline reported by Hinton *et al.***)
3) Gaussian SVC (Support Vector Classifier)
4) Sigmoid SVC
5) Random Forest

All models used were trained using their corresponding scikit-learn [10] implementation. Given the size of the ImageNet dataset, the considered classifiers were first run on a subset of the data consisting of images from 100 of the 1000 classes. The best performing classifiers were then run on the complete dataset for hyperparameter tuning. The following are descriptions of the algorithms used by the models:

1) Logistic Regression: Logistic regression is a probabilistic discriminative model commonly used for classification tasks. To negate the need for learning the class conditional distributions $p(\mathbf{x}|\mathrm{C_k})$, the class priors $p(\mathrm{C_k})$ and the need for implementing the Bayes' theorem, discriminative models rather use the functional form of the generalized linear model, which is defined through the conditional distribution $p(\mathrm{C_k}|\mathbf{x})$. The parameters $\mathbf{w}$ are then determined by using maximum likelihood. This approach has two advantages over the generalized approach: there are fewer adaptive parameters to be determined (linear dependence as compared to a quadratic growth) and the predictive performance will not suffer if the class-conditional density assumptions give a poor approximation of the true distributions.

In multiclass classification, the posterior probabilities for a large class of distributions is given by a softmax transformation of linear functions of the feature variables

$$p(C_\mathrm{k}|\phi) = y_\mathrm{k} = \frac{exp(a_\mathrm{k})}{\sum_j exp(a_\mathrm{j})} \tag{1}$$

where the activation functions a_k are given by

$$a_\mathrm{k} = \mathbf{w_k}^\mathrm{T}\phi. \tag{2}$$

Due to the non-linearity of the activation functions, the maximum likelihood solution is no longer in closed-form, thus it is necessary to use an iterative optimization technique to find the unique minimum of the concave error function E($\mathbf{w}$). By using the Newton-Raphson method, the update takes the form

$$\mathbf{w}^{(\mathrm{new})} = \mathbf{w}^{(\mathrm{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w}) \tag{3}$$

where $\mathbf{H}$ is the Hessian matrix whose elements comprise the second derivatives of E($\mathbf{w}$) with respect to the components of $\mathbf{w}$.

By using the maximum likelihood and the negative logarithm of the likelihood, we can obtain the cross-entropy error function for the multiclass classification problem, which is the default error function used in the scikit-learn implementation.

$$\nabla_{\mathbf{w_j}} E(\mathbf{w}_1, ..., \mathbf{w}_\mathrm{K}) = \sum_{n=1}^{N} (y_\mathrm{nj} - t_\mathrm{nj})\phi_\mathrm{n} \tag{4}$$

The $M \times M$ Hessian matrix can be obtained by applying the Newton-Raphson update to the cross-entropy error function for the logistic regression model

$$\mathbf{H} = \nabla_{\mathbf{w_k}}\nabla_{\mathbf{w_j}} E(\mathbf{w}_1, ..., \mathbf{w}_\mathrm{K}) = -\sum_{n=1}^{N} y_\mathrm{nk}(\mathbf{I}_\mathrm{kj} - y_\mathrm{nj})\phi_\mathrm{n}{\phi_\mathrm{n}}^\mathrm{T}. \tag{5}$$

Finally, the iterative re-weighted least squares equation shown in Eq. 3 can be applied repeatedly until the parameters converge to the minimized error solution, and thus learning a maximum likelihood multiclass logistic regression classification model.

2) Linear SVM: The Support Vector Machine, commonly referred to as just SVM, is another classifier that was implemented as a baseline algorithm. SVM is a "decision machine" which means it does not provide posterior probabilities. The method involves learning a hyperplane H (called the maximum margin hyperplane) that maximizes the distance between the decision boundary and any of the samples. Consequently, any unclassified samples can be projected into the same space as the training data, and then a class can be assigned according to the learned decision boundary.

Multiclass classification using SVM can be of various different forms. However, scikit-learn's implementation utilizes the widely used one-against-one approach which trains $\frac{K(K-1)}{2}$ different 2-class SVMs on all possible pairs of classes, where K is the total number of classes in the modelled dataset. The test samples can then be classified according to which class has the highest number of "votes".

Learning the maximum margin hyperplane is based on learning two parameters: a set of weights $\mathbf{w}$ denoting the normal vector of the hyperplane as well as some bias b describing the position of the hyperplane. The training task comprises of learning the hyperplane $H = \mathbf{w} + b$ by learning the parameters that maximize the margin for the given training data, which is equivalent to the optimization task

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N} \xi_n \tag{6}$$

where the first term is the loss function, $C > 0$ controls the trade-off between the slack variable penalty and the margin, and ξ_n is the classification error known as the slack variable. The purpose of the second term in Eq. 6 is to softly penalize points that lie on the wrong side of the margin boundary. This comes up in models dealing with training data that is not linearly separable, and thus, the model will allow overlapping class distributions at a penalty. Therefore, Eq. 6 should be solved using the following constraints

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n, \quad n = 1, ..., N \tag{7}$$

$$\xi_n \geq 0 \tag{8}$$

where t_n is the true class and $y(\mathbf{x}_n)$ is the predicted class.

The linear SVM model we used for this task was trained using SGD as implemented by scikit-learn's SGD Classifier. This is also the baseline model reported in the original paper. The Gaussian and Sigmoid SVC follow a similar procedure as linear SVM, but with an added step of implicitly mapping their inputs into high-dimensional input spaces using a kernel function. This step allows kernelized SVMs to have non-linear boundaries.

3) Random Forest: A random forest classifier is a meta estimator that trains multiple decision tree classifiers on varying sub-samples of the dataset and uses averaging to control overfitting and improve accuracy. A decision tree for a given subset of data is constructed by regrouping data points recursively according to the input feature that best separates the classes. For each feature, either the cost in accuracy or the Gini impurity is calculated. The feature giving the least value is then selected. A random forest classifies an input data by evaluating the input on multiple decision trees and doing a "majority voting".

D. Hyperparameter Tuning

The best-performing models for the 100-class classification are the logistic regression and SGD-trained linear SVM models (more on Section IV).

1) Logistic Regression: The scikit-learn implementation of the logistic regression model has a hyperparameter C which denotes the inverse of regularization strength. The regularization term controls the values of the model parameters from reaching large values that cause overfitting. The regularization term is multiplied to the sum of the square of the weights, and this product is added to the cost function. Thus, if we are to minimize a cost function with a high regularization term, the weights end up having small values.

2) SGD-trained Linear SVM: For this model, the scikit-learn implementation has a hyperparameter α which functions as a regularization multiplier and is also used to calculate the learning rate of the SGD. The learning rate η is calculated via the following formula:

$$\eta = \frac{1}{\alpha(t+t_0)}$$

where t is the current step number of the SGD and t_0 is chosen by a heuristic proposed by Leon Bottou [10]. The learning rate decreases over time to simulate annealing, where favor shifts from exploration to exploitation over the course of the training. A higher α value means larger regularization and lower learning rate.

IV. RESULTS

Using different baseline models to classify a subset of the data into 100 classes, the following results on Table II were obtained. The accuracy of each algorithm was assessed using the validation set.

TABLE II
RESULTS ON 100 CLASS ILSVRC-2010 VALIDATION SET

Baseline Model	Accuracy
Logistic Regression	95.89%
SGD-trained Linear SVM	95.35%
Random Forest	82.67%
Gaussian SVC	12.67%
Sigmoid SVC	9.28%

The algorithms with the best performance on the 100 class subset were logistic regression and SGD classifier. The strong performance of linear models is consistent with the results of Hinton *et al*. who used an SGD-trained classifier as their baseline. Thorough hyperparameter tuning was then conducted on both models. The value of the inverse of the regularization strength parameter C was tuned for the logistic regression model and the value of the regularization term α was tuned for the SGD-trained linear SVM classifier. Training each model took 2-3 hours on the full ILSVRC-2010 training set. The accuracy of the logistic regression models and SGD-trained linear SVM classifiers with varying hyperparameter values can be found on Tables III and IV respectively.

TABLE III
COMPARISON OF RESULTS OF LOGISTIC REGRESSION CLASSIFICATION ON ILSVRC-2010 VALIDATION SET

C Value	Accuracy
0.01	60.77%
0.1	86.03%
1	82.67%
10	89.38%
100	88.98%
200	88.13%

TABLE IV
COMPARISON OF RESULTS OF SGD-TRAINED LINEAR SVM CLASSIFICATION ON ILSVRC-2010 VALIDATION SET

α Value	Accuracy
1E-6	75.19%
1E-5	80.89%
1E-4	79.39%
1E-3	67.36%

The optimal value of C was found to be 10 for logistic regression while the optimal value of α was found to be 1E-5 for SGD classification. The accuracy of both algorithms were then assessed using the test set. The accuracy of the optimized algorithms on the test set can be seen on Table V.

TABLE V
COMPARISON OF RESULTS ON ILSVRC-2010 TEST SET

Baseline Model	Accuracy
Logistic Regression	89.16%
SGD-trained Linear SVM	80.84%

The performance of the SGD classification was very similar to the results reported by Hinton *et al*. Our SGD classifier slightly outperformed the SGD classifier used in the original paper likely due to our hyperparameter tuning. The logistic regression algorithm we implemented significantly outperformed both the SGD and CNN classifiers used in the original paper.

V. DISCUSSION

Running different baseline models on 100 out of 1000 classes allowed us to pick the models with the most potential given our data representation. We were expecting that the performance of each model in a smaller class set will carry over and scale well enough with the full class set that is 10 times bigger. Thus, we did not bother fine tuning the hyperparameters of the models that performed poorly at this stage. Despite the decent performance of the random forest classifier, we decided to fine tune only the top 2 models due to time constraints. We also considered a k-NN (Nearest Neighbours) model for the task, however, due to it being a lazy classifier, it would loop through each training data for each testing input which took too much time to accomplish.

The scikit-learn implementation of the logistic regression model has a default value of 1 for the hyperparameter C. We decided to test C values in the scales of 10 because this will allow us to cover a wide range of values for the hyperparameter while enabling us to establish a rough trend in accuracy. An additional value of $C = 200$ was tested to see whether the accuracy has already peaked or will still oscillate. Due to the decrease in accuracy from $C = 100$ to $C = 200$, we assumed that larger values of C will just lead to lower accuracy. Since C is the inverse of regularization strength, a high C value means low regularization which could lead to large variations in model parameter values. Upon finding the best value for C, we decided not to test values near 10 due to the lengthy training time and also because this best-performing logistic regression model already outperformed the baseline model and CNN of the original paper on the test set as seen on Table V. The solver used in training the logistic regression models is the Limited-memory BFGS algorithm implemented by scikit-learn which is ideal for multiclass problems with large datasets.

As for the SGD-trained linear SVM classifier, the default value of α in the scikit-learn implementation is 1E-4. Again, we used logarithmic increments to cover the most range of values with the least total training time. SGD was favored over a full gradient descent training because the training time for the latter proved to be much longer. Also, the baseline model used in the paper used SGD in training. The SGD classifier implementation on scikit-learn offers a variety of options for the loss function, penalty and learning rate schedule, but since we wanted a linear SVM classifier, we kept the hinge loss function, L2 penalty, and learning rate as discussed in Section III.

VI. CONCLUSION

To conclude, by fine-tuning the hyperparameters of a logistic regression baseline model, we were able to beat the accuracy score of the SGD-trained linear SVM and deep CNN classifiers proposed by Hinton *et al*. on the ILSVRC-2010 dataset by a significant margin. This was accomplished using the same SIFT + FV representation, but with much less training time and without the use of a GPU.

VII. STATEMENT OF CONTRIBUTIONS

A. Mohammed Shalaby

Mohammed worked on generating the MATLAB code for the preprocessing and SIFT and FV representation learning. He also carried out tests for a wide range of classifiers on those representations to shortlist the final classifiers to be used. As for the report, he set up the structure of the report, wrote the Introduction, the Related Work section and parts A, B and some of C of the Methodology.

B. Ismael Hishon-Rezaizadeh

Ismael assisted in developing the preprocessing, the logistic regression, SGD classification and the SIFT and Fisher Vector extraction code. He also conducted the hyperparameter tuning and wrote the Results section of the report.

C. James Manalad

James worked on preparing the dataset for the 100-class classification, wrote parts of Methodology, Discussion and Conclusion. He also did an overall review and consistency check of the report, as well as the managing of the references.

We hereby state that all the work presented in this report is that of the authors.

REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[2] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "Labelme: A database and web-based tool for image annotation," *International Journal of Computer Vision*, vol. 77, no. 1, pp. 157–173, May 2008. [Online]. Available: https://doi.org/10.1007/s11263-007-0090-8

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386

[4] A. Berg, J. Deng, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," 2010.

[5] J. Sanchez and F. Perronnin, "High-dimensional signature compression for large-scale image classification," in *CVPR 2011*, June 2011, pp. 1665–1672.

[6] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the fisher vector: Theory and practice," *International Journal of Computer Vision*, vol. 105, no. 3, pp. 222–245, Dec 2013. [Online]. Available: https://doi.org/10.1007/s11263-013-0636-x

[7] A. Vedaldi and B. Fulkerson, "VLFeat - an open and portable library of computer vision algorithms," in *ACM International Conference on Multimedia*, 2010.

[8] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ser. ICCV '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 1150–. [Online]. Available: http://dl.acm.org/citation.cfm?id=850924.851523

[9] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ser. ECCV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 143–156. [Online]. Available: http://dl.acm.org/citation.cfm?id=1888089.1888101

[10] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.