Assignment 2
Tutor: Wang Jue

Group Members: Michael Muniappan (mmun6233, SID:450441905), Graham Herdman (gher4675, SID:440151889), Liming Ge (lige0519, SID:460124751)

Abstract

There are myriads of different techniques which are designed and developed to solve the problem of image classification. Given this multiplicity of classifiers, it is only natural to ask which classifier performs best on a given dataset. To this end, we employ the CIFAR-100 dataset to analyse and compare the performance of three different classifiers: random forests (RF), convolutional neural networks (CNNs) and support vector machines (SVMs). Our results indicate that CNNs performed best overall, followed by SVMs, with RFs coming in last. Instructions to run the code are found in Appendix 5 of the report.

1. Introduction

Given the prolific abundance of algorithms devoted to the problem of classifying images [1],[2],[3],[4],[5], it is becoming increasingly difficult to determine which techniques work best and which do not. As image classification has a diverse range of applications, from remote sensing [6],[7],[8] to face recognition [9],[10], this is problematic as the inability to filter out less effective classification methods could lead to grave, real-world miscategorizations. For example, if a brain tumour that was actually malignant was misclassified as benign, the consequences are literally life-or-death. Therefore, the question of which classifier performs best on a given dataset is one of extreme benefit, as it allows researchers and scientists alike to weed out less successful methods and focus their attention to those which exhibit the highest levels of accuracy and reliability. To contribute meaningfully to this discussion, we compare and contrast the performance of three classifiers which are commonly used in the machine learning literature, namely random forests (RF), convolutional neural networks (CNN) and support vector machines (SVM), on the CIFAR-100 dataset.

CIFAR-100 Dataset

The CIFAR-100 dataset [11], a labelled subset of the 80 Million Tiny Images collection [12] , is composed of 60,000 images. These images are divided into six batches of equal size, where five batches are for training whilst the remaining batch is for testing. All images are of size 32 x 32 in 3 color channels, resulting in each image being an input vector with 3072 dimensions [13]. Furthermore, each image belongs to exactly one of 100 categories as well as one of 20 super-categories, with an equal number of members (600) in each class. For a complete description of categories and super-categories, refer to Appendix 1.

The primary reason for selecting the CIFAR-100 is because of its relevance and widespread use within the machine learning literature, a fact evidenced by the plethora of studies which employ this collection of images [14], [15], [16], [17]. Moreover, as this dataset has already been classified by a number of CNNs [14], [15], [16], [17], [18] , it allows us to develop a comparative benchmark for the performance of our own convolutional neural network. On this note, as the current state-of-the-art performance on CIFAR-100 sees an error rate of 19.25% (accuracy of 80.75%) [16], these comparative benchmarks can be extended to our random forest and support vector machine models.

2. Previous Work

As aforementioned, the CIFAR-100 dataset has been used to test the efficacy of various different classifiers, with convolutional neural networks being the most common technique being examined. The approach and subsequent results of these studies have been mixed. Zeiler and Fergus' attempt to replace deterministic pooling operations with stochastic ones yielded a success rate of ~57% with the CIFAR-100 dataset [14]. Lin, Chen and Yan developed a novel structure referred to as 'Network in Network' which replaces the conventional convolutional layers with multilayer perceptrons. This approach fared significantly better than Zeiler and Fergus' with an accuracy of ~65% with the CIFAR-100 dataset [15].

Huang, Liu & Weinberger's algorithm currently exhibits a state-of-the-art performance on the CIFAR-100 dataset with an accuracy rate of 80.75%, a tremendous increase over other methods. Their key to success was grounded in making the CNN denser, so that "*each layer is directly connected to every other layer in a feed-forward fashion*" [16]. Lastly, Romero et al. employed a student-teacher approach whereby intermediate teacher layers output 'hints' to improve the training and overall final performance of the outer student layers [18]. On the CIFAR-100 dataset, this resulted in an overall accuracy of ~65%.

To our knowledge, no existing studies discuss the performance of either random forests or support vector machines on the CIFAR-100 datasets. However, these algorithms have been applied to other image collections. Bosch, Zisserman & Munoz apply random forests on the Caltech-101 and Caltech-265 datasets, achieving accuracy rates of 80% and 45.3% respectively [3]. These datasets are interesting as, much like the CIFAR-100 set, they exhibit a large inter-class variability with 101 and 256 object categories apiece. Nevertheless, they do differ in both image resolution (300 x 300 pixels), and in the case of Caltech-256, the degree of intra-class variability (80 - 827 images per category) [3].

Yao, Khosla & Fei-Fei combined random forests with discriminative decision trees and evaluated the performance of the combined classifier on the PASCAL VOC2010 dataset [19]. This collection describes nine human activities: “Phoning”, “Playing a musical instrument”, “Reading”, “Riding a bicycle or motorcycle”, “Riding a horse”, “Running”, “Taking a photograph”, “Using a computer”, and “Walking” [20]. This is a rather challenging dataset to work with, and Yao’s algorithm yielded an overall precision of 64.6% [19]. While PASCAL VOC2010 may not be too akin to the CIFAR-100 dataset, it’s still an interesting result which puts the performance of random forests with respect to image classification into context.

SVMs have also been applied to other datasets. Lin et. al train support vector machines on a rather difficult dataset, namely the ImageNet-1000 [21]. The ImageNet-1000 collection is comprised of 1,461,406 images of varying resolutions, with each image falling into one of 1000 classes [22]. When their paper was published, Lin et. al’s SVM achieved a state-of-the-art performance with a classification accuracy of 52.9% [21].

Two studies apply SVMs to the problem of classifying 3D objects, making use of the COIL-100 dataset [23], [24]. The COIL-100 dataset consists of 7200 128 x 128 images which, in sum, represent a total 100 objects (72 views per object) [23]. Pontil & Verri used linear SVMs on the image collection, and found that if the training sets were of size 36 per object, the system reached a perfect score. They also corrupted the images with noise to test the robustness of the classifier and found that if the noise units were within a certain quantity ($\pm$ 100 gray levels), the SVM performed equally well [23]. Roobaert & Van Hulle used the same dataset as Pontil & Verri, but made it more similar to the CIFAR-100 by reducing the size of the images to 32 x 32. They then trained the support vector machine with different numbers of views per object for different subsets of the database. They obtained a classification accuracy of ~87% when they restricted the number of views per object to 4 for the entire COIL database [24].

Finally, Bosch, Zimmerman & Munoz contrasted the performance of their random forest classifier on the Caltech-101 dataset with that of a multiple kernel SVM (M-SVM). When they used a training set of 30 images, the M-SVM outperformed the random forest learner by a marginal 1.3% (81.3% as opposed to RF's 80.0%) [3]. The authors note that this gain in performance is very much offset by the computational expensiveness of the former as compared to the latter [3].

3. Methods

Three different classification techniques were used for the purposes of this study: random forests, convolutional neural networks and support vector machines.

Random Forests

Random forests, as conceptualized by Breiman, are generalizations of decision trees [25], whereupon rather than classifying an object based on a single tree, the decision or verdict of multiple trees are pooled together to reach a singular, aggregate decision [25]. This ensembling results in a "*substantial performance improvement over single base learners*" [26] , hence its growing usage in the problem of image classification [27].

The high-level description of how a random forest classifies images is fairly intuitive. The image in question is sent down all trees in the forest, with each tree outputting a class label. The forest then simply classifies the image as the majority class label across all trees [3]. Randomness is introduced into the forest through training via two mechanisms: a) subsampling the training data so each tree is grown using a different subset and b) generating the decision tests for the internal nodes in each tree [3].

Unique Design Choices for our Random Forest Implementation

Our Random Forest implementation used 30 trees and enforced a max depth of 5 levels. These hyper-parameters were chosen because they provided the best mix of accuracy to model running time. As well as that, we chose to balance the class weighting for each subsample chosen to make a decision tree. This proved to provide a ~1% increase in accuracy over not balancing the subsets.

Convolutional Neural Networks

Convolutional neural networks are a particular form of deep learning generally applied to grid-like data, such as images, to "*automatically and adaptively learn spatial hierarchies of features, from low to high-level patterns*" [28]. They are of much interest and importance to the image classification problem, as their performance often exceeds that of any other classifier, a fact which holds true for a variety of datasets [28], [29].

While CNNs can and often do have different architectures, they generally consist of three layers: convolutional, pooling and fully connected [28]. Convolutional layers serve as the feature extractors, or more specifically, derive feature maps from the input images [29], which in turn serve as input for some non-linear activation function. Pooling layers reduce the dimensionality of these feature maps, whilst simultaneously making them more invariant to input distortions and translations [28]. Finally, in the case of image classification, the fully connected layers are responsible for performing the classification (i.e. assigning the image a class label) [28].

Unique Design Choices for our Convolutional Neural Network Implementation

For our CNN implementation we used an architecture with 6 convolutional layers (each followed by a max pooling layer) followed by 2 dense layers, making for a total of 11 layers. This architecture was chosen because it proved to give us the best accuracy in a feasible time frame. As well as the choice in layers we also made use of ELU activation in all layers except the final dense layer. This choice was made because it was found that the ELU activation function significantly decreased training time over the standard ReLu activation function without any loss in accuracy [20].

Further improvements were made to the architecture by adding batch normalization and dropouts. Batch Normalization normalizes the data on each epoch instead of in the preprocessing step. This significantly improved our models accuracy as when we tested the model without batch normalization vs with batch normalization over 5 epochs, accuracy improved by ~3%. Our use of dropouts also allows our model to generalize much better to the testing set, further increasing accuracy by around ~1%.

Support Vector Machines

Support vector machines are a supervised classification method which "*generate input-output mapping functions from a set of labelled training data*" [30]. Grounded in statistical learning theory, they have garnered widespread attention in the machine learning community due to the fact they offer highly competitive performance across different fields, including bioinformatics, text processing and, last but not least, image classification [30].

The basic principle underlying SVMs is to separate a dataset into two (or, through extension, more) distinct classes via an optimal hyperplane, one which '*adopts the maximal distance from any of the [members of the dataset]*' [31]. In the case of linearly separable data, this hyperplane perfectly separates the dataset. However, in most cases, the data is nonlinearly separable, in which case a 'soft margin' is used to allow '*some data points to push their way through the margin of the separating hyperplane without affecting the final result*' [31] . The 'support vectors' refer to those points which lie closest to the hyperplane (otherwise known as a decision surface) [32].

Unique Design Choices for our Support Vector Machine Implementation

For our SVM implementation we used sklearns SVC model with an rbf kernel. We decided on using the rbf kernel because out of all the kernel options; linear, poly and sigmoid, the rbf kernel performed the quickest (running 10 seconds quicker than the next fastest kernel option - linear).

Data Preprocessing

Data preprocessing was handled differently for all three models based on the performance measurements we gleaned from our experiments. For our Random Forest model we used standard normalization (dividing each pixel entry by 255) because other methods such as SkLearn's standardscaler did not provide better performance. As well as using normalization we used a subset of our dataset for our Random Forest model. Taking a 10,000 : 1,000 split for images training data to testing data.

For our CNN model we initially tried full dataset normalization, however we then compared the results from this technique to batch normalization and decided to use batch normalization since it increased the accuracy of our model by ~3%. We also initially trained the model with the same subset of the data as with our Random Forest model. However, we discovered that the accuracy of our model was significantly higher (~10%) when we used the entire dataset than when we used a subset of the data over the same time period (20 minutes - 85 epochs: subset vs 20 epochs: full set).

Finally, for our SVM model we made use of standard full dataset normalization, grey scaling and Histogram of oriented gradients (HOG) preprocessing. We found that converting the images from full colour to grey scale significantly decreased our models running time. Further the use of HOG preprocessing decreased the training time of our model even further. In fact in an experiment we conducted using 5000 images from the training set and 5000 from the use of HOG preprocessing allowed the model to train 3 times faster than without it.

4. Experiments and Discussion

The results obtained in the table below are averages of the scores obtained in 10-fold cross validation and each model was run according to the parameters listed below:

CNN:

1. Epochs = 15 (63 seconds per epoch)
2. Batch Size = 64
3. Data set = Full data set
4. Run time = ~150 minutes

SVM:

1. kernel = rbf
2. Data set = Full data set
3. Run time per model = ~14:30 minutes
4. Run time = ~140 minutes

Random Forest:

1. Number of trees = 30
2. Data set = 10000 : 1000 training : test split
3. Run time per model = ~25 seconds

4. Run time = ~4 minutes

Hardware and Software Specifications

The models were run on Google's Google Colaboratory software.

GPU: 1 x Tesla K80 - 12GB VRAM

CPU: 1 x single core hyper threaded Xeon Processor - 2.3Ghz

RAM: 12.6 GB

DISK: 33 GB

Table 1: Summary of Results

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
Random Forest	10.27	7.43	10.26	6.11
SVM	17.54	17.00	18.00	16.00
CNN	55.93	59.00	56.00	56.00

The first thing to note about the results displayed above is that the CNN model achieved the highest accuracy (55.93%), followed by the SVM model (17.54%) and then finally the Random Forest (10.27%). This was to be expected as it was mentioned that this trend was seen in the literature. The most surprising finding in our results is the significantly poor performance of the Random Forest model since Random Forest models are increasingly being used to analyse satellite imagery.

As well as this, it appears that the Random Forest Model suffered from the fact that a majority vote of multiple decision trees is used to build decisive decision rules. We can see this in the difference between the model's Precision and Recall values. Precision is significantly lower than recall, thus, suggesting that the model came up with a decision rule that aggressively assigned images to a subset of a few classes (i.e. causing the portion of positive identifications to be accurate whilst increasing the proportion of the datasets class members that are correctly identified).

Another interesting finding from the results is that both the SVM and the CNN managed to balance precision and recall (i.e. SVM: 17 vs 18 and CNN: 59 vs 56). These measures are usually competing measures and do not end up very similar. A possible reason for this is that our dataset is very balanced and each class has a relatively small amount of examples (500 per class) which means there is not much difference between the proportion of images correctly identified and the proportion of images per class in the actual dataset.

It is important to note that even though the results seem relatively low in comparison to other classification accuracies on other datasets (i.e. models on the MNIST dataset routinely achieve above 98% accuracy) it should be noted that our dataset has 100 categories (vs 10 for MNIST) and therefore, the baseline for our models (as defined by the accuracy of random selection) is only 1%. Thus, even the poorest performing model (Random Forest) performed 10 times better than our baseline.

As well as this, a significant issue that we were met with in this experiment was the trade-off between resources and accuracy. The most powerful models on this data set used a dense neural network architecture. This is the architecture that was used to achieve the state of the art score of 80.25%. A issue with this architecture is that it requires a lot of computing power in order to implement in a feasible amount of time. In fact, using our resources (google colab) it would take 40 days to train such a model. Therefore, for future study on this dataset a significant investment in computing resources would allow for more creative and accurate models.

5. Conclusion & Future Work

In this paper, we have compared and evaluated the performance of three different classification schemes with respect to the CIFAR-100 dataset. In accordance with the literature, the CNN model was found to be the highest performing with an accuracy of 55.93, while the SVM and RF fared worse with accuracies of 17.54% and 10.27% respectively[*]. These models could have attained higher rates of accuracy, for example our CNN model achieved an accuracy of 66% when run for 200 epochs (3 hours and 20 minutes). however, the performance gain would come at the expense of taking significantly more time to train. This brings to light one of the toughest facets involved in constructing a well-rounded classifier, which is balancing the tradeoff between accuracy and runtime. It also highlights the fact that not all classification schemes are rendered equal, with an appropriate model selection having the potential to result in considerable performance gains.

There are several directions this research could be taken for future work. For one, this analysis can be extended to include other image classification schemes, such as logistic regression and nearest-neighbor based approaches. On this note, other neural network (NN) structures, for example, residual or stochastic neural networks, could be incorporated into the experiment to better gauge performance differences across varying types of NN. Moreover, it would be beneficial to replicate the results of this study across a number of datasets, such as the Caltech-256 or even the ImageNet collection, to shed light on the role the choice of images has on classifier performance. The latter set would be particularly interesting, because '*extending methods for medium-scale datasets to large-scale datasets is not easy*' [21] and hence could lead to a wildly different set of outcomes.

[*]These accuracy results were obtained through averaging the 10 fold cross validation results.

References

1. Boiman O, Shechtman E, Irani M. In defense of nearest-neighbor based image classification. Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on 2008 Jun 23 (pp. 1-8). IEEE.
2. Chapelle O, Haffner P, Vapnik VN. Support vector machines for histogram-based image classification. IEEE transactions on Neural Networks. 1999 Sep;10(5):1055-64.
3. Bosch A, Zisserman A, Munoz X. Image classification using random forests and ferns. Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on 2007 Oct 14 (pp. 1-8). IEEE.
4. Ciresan DC, Meier U, Masci J, Maria Gambardella L, Schmidhuber J. Flexible, high performance convolutional neural networks for image classification. IJCAI Proceedings-International Joint Conference on Artificial Intelligence 2011 Jul 16 (Vol. 22, No. 1, p. 1237).
5. Li J, Najmi A, Gray RM. Image classification by a two-dimensional hidden Markov model. IEEE transactions on signal processing. 2000 Feb;48(2):517-33.
6. Wang F. Fuzzy supervised classification of remote sensing images. IEEE Transactions on geoscience and remote sensing. 1990 Mar;28(2):194-201.
7. Pal M, Mather PM. Support vector machines for classification in remote sensing. International Journal of Remote Sensing. 2005 Mar 1;26(5):1007-11.
8. Blaschke T. Object based image analysis for remote sensing. ISPRS journal of photogrammetry and remote sensing. 2010 Jan 1;65(1):2-16.
9. Gao S, Tsang IW, Chia LT. Kernel sparse representation for image classification and face recognition. Computer Vision–ECCV 2010 Sep 5 (pp. 1-14). Springer, Berlin, Heidelberg.
10. Tan X, Chen S, Zhou ZH, Zhang F. Face recognition from a single image per person: A survey. Pattern recognition. 2006 Sep 1;39(9):1725-45.
11. Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images. Technical report, University of Toronto; 2009 Apr 8.
12. Massachusetts Institute of Technology & New York University (US). 80 Million Tiny Images [Internet]. 2017 [cited 2018 Oct 25]. Available from: http://groups.csail.mit.edu/vision/TinyImages/
13. Ba J, Caruana R. Do deep nets really need to be deep?. Advances in neural information processing systems 2014 (pp. 2654-2662).
14. Zeiler MD, Fergus R. Stochastic pooling for regularization of deep convolutional neural networks. arXiv preprint arXiv:1301.3557. 2013 Jan 16.
15. Lin M, Chen Q, Yan S. Network in network. arXiv preprint arXiv:1312.4400. 2013 Dec 16.

16. Huang G, Liu Z, van der Maaten L, Weinberger KQ. Densely Connected Convolutional Networks. arXiv preprint arXiv:1608.06993. 2016 Aug 25.
17. He K, Zhang X, Ren S, Sun J. Identity mappings in deep residual networks. In European conference on computer vision 2016 Oct 8 (pp. 630-645). Springer, Cham.
18. Romero A, Ballas N, Kahou SE, Chassang A, Gatta C, Bengio Y. Fitnets: Hints for thin deep nets. arXiv preprint arXiv:1412.6550. 2014 Dec 19.
19. Yao B, Khosla A, Fei-Fei L. Combining randomization and discrimination for fine-grained image categorization. In Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on 2011 Jun 20 (pp. 1577-1584). IEEE.
20. Bourdev L, Malik J, Maji S. Action recognition from a distributed representation of pose and appearance. In Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on 2011 Jun 25 (pp. 3177-3184). IEEE.
21. Lin Y, Lv F, Zhu S, Yang M, Cour T, Yu K, Cao L, Huang T. Large-scale image classification: fast feature extraction and svm training. In Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on 2011 Jun 20 (pp. 1689-1696). IEEE.
22. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC. Imagenet large scale visual recognition challenge. International Journal of Computer Vision. 2015 Dec 1;115(3):211-52.
23. Pontil M, Verri A. Support vector machines for 3D object recognition. IEEE transactions on pattern analysis and machine intelligence. 1998 Jun;20(6):637-46.
24. Roobaert D, Van Hulle MM. View-based 3D object recognition with support vector machines. In Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop. 1999 Aug (pp. 77-84). IEEE.
25. Breiman L. Random forests. Machine learning. 2001 Oct 1;45(1):5-32
26. Dahinden C, ETHZ M. An improved Random Forests approach with application to the performance prediction challenge datasets. Hands-on Pattern Recognition, Challenges in Machine Learning. 2011;1:223-30.
27. Horning N. Random Forests: An algorithm for image classification and generation of continuous fields data sets. In Proceedings of the International Conference on Geoinformatics for Spatial Infrastructure Development in Earth and Allied Sciences, Osaka, Japan 2010 Dec 9 (Vol. 911)
28. Yamashita R, Nishio M, Do RK, Togashi K. Convolutional neural networks: an overview and application in radiology. Insights into Imaging. 2018:1-9.

29. Rawat W, Wang Z. Deep convolutional neural networks for image classification: A comprehensive review. Neural computation. 2017 Sep;29(9):2352-449.
30. Wang L, editor. Support vector machines: theory and applications. Springer Science & Business Media; 2005 Jun 21.
31. Noble WS. What is a support vector machine?. Nature biotechnology. 2006 Dec;24(12):1565.
32. Berwick R. An Idiot's guide to Support vector machines (SVMs) [lecture notes]. Massachusetts Institute of Technology. Retrieved on October. 2003;21:2011

Appendix 1: CIFAR-100 Categories and Super-Categories

Superclass	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo

medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Appendix 2: F1 Scores for RF model

	precision	recall	f1-score	support
0	0.13	0.37	0.19	100
1	0.07	0.06	0.07	100
2	0.20	0.01	0.02	100
3	0.10	0.13	0.12	100
4	0.05	0.03	0.04	100
5	0.00	0.00	0.00	100
6	0.12	0.05	0.07	100
7	0.00	0.00	0.00	100
8	0.00	0.00	0.00	100
9	0.24	0.07	0.11	100
10	0.00	0.00	0.00	100
11	0.00	0.00	0.00	100
12	0.00	0.00	0.00	100
13	0.33	0.01	0.02	100
14	0.02	0.06	0.03	100
15	0.00	0.00	0.00	100
16	0.07	0.01	0.02	100
17	0.07	0.18	0.10	100
18	0.00	0.00	0.00	100
19	0.00	0.00	0.00	100
20	0.15	0.53	0.24	100
21	0.09	0.11	0.10	100
22	0.00	0.00	0.00	100
23	0.08	0.24	0.12	100
24	0.18	0.41	0.25	100
25	0.00	0.00	0.00	100
26	0.00	0.00	0.00	100
27	0.05	0.03	0.04	100
28	0.29	0.05	0.09	100
29	0.00	0.00	0.00	100
30	0.14	0.05	0.07	100
31	0.00	0.00	0.00	100
32	0.00	0.00	0.00	100
33	0.09	0.05	0.06	100
34	0.05	0.01	0.02	100
35	0.07	0.02	0.03	100
36	0.06	0.27	0.10	100
37	0.06	0.01	0.02	100
38	0.05	0.13	0.07	100
39	0.50	0.01	0.02	100
40	0.00	0.00	0.00	100
41	0.34	0.43	0.38	100
42	0.00	0.00	0.00	100
43	0.11	0.07	0.08	100
44	0.00	0.00	0.00	100
45	0.00	0.00	0.00	100
46	0.00	0.00	0.00	100
47	0.10	0.28	0.15	100
48	0.00	0.00	0.00	100
49	0.00	0.00	0.00	100
50	0.00	0.00	0.00	100
51	0.06	0.10	0.07	100
52	0.10	0.83	0.18	100
53	0.21	0.46	0.28	100
54	0.06	0.06	0.06	100
55	0.00	0.00	0.00	100
56	0.05	0.06	0.06	100
57	0.00	0.00	0.00	100
58	0.00	0.00	0.00	100
59	0.00	0.00	0.00	100
60	0.11	0.63	0.19	100
61	0.13	0.16	0.14	100
62	0.14	0.40	0.21	100
63	0.06	0.39	0.10	100
64	0.04	0.03	0.03	100
65	0.05	0.02	0.03	100
66	0.00	0.00	0.00	100
67	0.00	0.00	0.00	100
68	0.18	0.29	0.22	100
69	0.10	0.01	0.02	100
70	0.12	0.28	0.17	100
71	0.08	0.26	0.13	100
72	0.00	0.00	0.00	100
73	0.10	0.48	0.17	100
74	0.06	0.13	0.08	100
75	0.10	0.29	0.15	100
76	0.07	0.03	0.04	100
77	0.00	0.00	0.00	100
78	0.00	0.00	0.00	100
79	0.00	0.00	0.00	100
80	0.01	0.01	0.01	100
81	0.25	0.01	0.02	100
82	0.13	0.40	0.19	100
83	0.00	0.00	0.00	100
84	0.00	0.00	0.00	100
85	0.14	0.03	0.05	100
86	0.09	0.19	0.12	100
87	0.00	0.00	0.00	100
88	0.04	0.01	0.02	100
89	0.00	0.00	0.00	100
90	0.00	0.00	0.00	100
91	0.39	0.22	0.28	100
92	0.03	0.04	0.04	100
93	0.00	0.00	0.00	100
94	0.20	0.02	0.04	100
95	0.13	0.33	0.19	100
96	0.04	0.01	0.02	100
97	0.07	0.21	0.10	100
98	0.00	0.00	0.00	100
99	0.31	0.04	0.07	100
avg / total	0.07	0.10	0.06	10000

Appendix 3: F1 Scores for CNN Model

	precision	recall	f1-score	support
0	0.80	0.76	0.78	88
1	0.66	0.80	0.72	89
2	0.47	0.52	0.50	88
3	0.33	0.39	0.35	93
4	0.33	0.46	0.38	92
5	0.43	0.54	0.48	96
6	0.55	0.55	0.55	85
7	0.65	0.71	0.68	87
8	0.76	0.66	0.71	88
9	0.76	0.59	0.66	90
10	0.35	0.46	0.40	84
11	0.48	0.36	0.41	89
12	0.68	0.61	0.64	90
13	0.58	0.50	0.54	94
14	0.69	0.41	0.51	90
15	0.65	0.34	0.45	87
16	0.49	0.52	0.51	90
17	0.86	0.68	0.76	94
18	0.47	0.59	0.52	88
19	0.57	0.52	0.55	90
20	0.84	0.79	0.82	92
21	0.78	0.71	0.75	87
22	0.67	0.51	0.58	86
23	0.64	0.78	0.71	92
24	0.59	0.77	0.67	88
25	0.68	0.38	0.49	90
26	0.71	0.46	0.55	90
27	0.44	0.41	0.42	88
28	0.72	0.75	0.73	91
29	0.78	0.43	0.56	92
30	0.52	0.62	0.57	93
31	0.79	0.51	0.62	90
32	0.54	0.50	0.52	86
33	0.57	0.62	0.59	86
34	0.31	0.79	0.45	94
35	0.47	0.22	0.30	90
36	0.63	0.58	0.60	84
37	0.61	0.67	0.64	87
38	0.43	0.34	0.38	92
39	0.82	0.66	0.73	90
40	0.49	0.43	0.45	87
41	0.87	0.77	0.82	90
42	0.31	0.74	0.43	89
43	0.42	0.70	0.52	90
44	0.47	0.29	0.36	95
45	0.40	0.50	0.45	90
46	0.33	0.39	0.36	92
47	0.48	0.68	0.56	90
48	0.75	0.81	0.78	94
49	0.81	0.66	0.73	92
50	0.37	0.29	0.33	95
51	0.58	0.49	0.53	92
52	0.57	0.62	0.59	91
53	0.72	0.88	0.79	88
54	0.65	0.73	0.69	89
55	0.26	0.34	0.29	92
56	0.86	0.70	0.77	90
57	0.80	0.57	0.67	91
58	0.68	0.75	0.71	87
59	0.73	0.34	0.47	96
60	0.80	0.79	0.80	91
61	0.64	0.50	0.56	94
62	0.57	0.69	0.63	90
63	0.54	0.57	0.55	90
64	0.53	0.23	0.32	91
65	0.37	0.42	0.39	91
66	0.74	0.50	0.60	86
67	0.54	0.38	0.44	88
68	0.74	0.92	0.82	88
69	0.80	0.70	0.74	92
70	0.63	0.51	0.56	92
71	0.67	0.74	0.70	87
72	0.27	0.27	0.27	92
73	0.46	0.66	0.54	92
74	0.38	0.47	0.42	92
75	0.77	0.76	0.77	92
76	0.74	0.77	0.76	88
77	0.37	0.54	0.44	84
78	0.53	0.30	0.38	90
79	0.64	0.60	0.62	94
80	0.42	0.31	0.35	91
81	0.46	0.73	0.57	89
82	0.83	0.82	0.82	88
83	0.49	0.43	0.46	86
84	0.71	0.38	0.50	94
85	0.72	0.73	0.73	90
86	0.81	0.57	0.67	92
87	0.45	0.76	0.56	92
88	0.38	0.76	0.51	92
89	0.66	0.65	0.66	88
90	0.73	0.48	0.58	89
91	0.64	0.69	0.66	90
92	0.57	0.45	0.50	92
93	0.34	0.30	0.32	90
94	0.80	0.79	0.80	91
95	0.64	0.51	0.56	91
96	0.52	0.48	0.50	86
97	0.44	0.64	0.52	88
98	0.41	0.18	0.25	89
99	0.73	0.31	0.43	88
avg / total	0.59	0.56	0.56	9000

Appendix 4: F1 Scores for SVM Model

	precision	recall	f1-score	support
0	0.40	0.29	0.34	100
1	0.05	0.11	0.07	100
2	0.00	0.00	0.00	100
3	0.00	0.00	0.00	100
4	0.03	0.03	0.03	100
5	0.12	0.06	0.08	100
6	0.17	0.04	0.06	100
7	0.09	0.01	0.02	100
8	0.13	0.14	0.13	100
9	0.12	0.22	0.16	100
10	0.05	0.04	0.05	100
11	0.00	0.00	0.00	100
12	0.04	0.03	0.04	100
13	0.02	0.03	0.02	100
14	0.04	0.14	0.06	100
15	0.13	0.02	0.03	100
16	0.15	0.22	0.18	100
17	0.06	0.25	0.09	100
18	0.05	0.02	0.03	100
19	0.02	0.02	0.02	100
20	0.13	0.29	0.18	100
21	0.05	0.07	0.06	100
22	0.00	0.00	0.00	100
23	0.67	0.02	0.04	100
24	0.18	0.06	0.09	100
25	0.06	0.03	0.04	100
26	0.05	0.05	0.05	100
27	0.03	0.04	0.04	100
28	0.24	0.08	0.12	100
29	0.08	0.04	0.05	100
30	0.06	0.13	0.08	100
31	0.00	0.00	0.00	100
32	0.06	0.08	0.07	100
33	0.04	0.01	0.02	100
34	0.00	0.00	0.00	100
35	0.00	0.00	0.00	100
36	0.07	0.06	0.06	100
37	0.04	0.02	0.03	100
38	0.07	0.01	0.02	100
39	0.10	0.05	0.07	100
40	0.18	0.02	0.04	100
41	0.33	0.24	0.28	100
42	0.11	0.02	0.03	100
43	0.04	0.02	0.03	100
44	0.06	0.01	0.02	100
45	0.00	0.00	0.00	100
46	0.07	0.01	0.02	100
47	0.20	0.03	0.05	100
48	0.06	0.09	0.08	100
49	0.09	0.39	0.14	100
50	0.10	0.01	0.02	100
50	0.10	0.01	0.02	100
51	0.03	0.07	0.04	100
52	0.20	0.35	0.25	100
53	0.27	0.04	0.07	100
54	0.02	0.03	0.03	100
55	0.00	0.00	0.00	100
56	0.11	0.04	0.06	100
57	0.12	0.08	0.10	100
58	0.05	0.26	0.09	100
59	0.05	0.02	0.03	100
60	0.25	0.79	0.38	100
61	0.21	0.35	0.27	100
62	0.03	0.02	0.02	100
63	0.06	0.10	0.08	100
64	0.04	0.01	0.02	100
65	0.00	0.00	0.00	100
66	0.02	0.01	0.01	100
67	0.14	0.05	0.07	100
68	0.00	0.00	0.00	100
69	0.21	0.06	0.09	100
70	0.05	0.20	0.07	100
71	0.22	0.22	0.22	100
72	0.00	0.00	0.00	100
73	0.11	0.09	0.10	100
74	0.04	0.02	0.03	100
75	0.04	0.07	0.05	100
76	0.16	0.21	0.18	100
77	0.00	0.00	0.00	100
78	0.03	0.18	0.05	100
79	0.33	0.01	0.02	100
80	0.07	0.02	0.03	100
81	0.08	0.04	0.05	100
82	0.20	0.08	0.11	100
83	0.04	0.01	0.02	100
84	0.00	0.00	0.00	100
85	0.07	0.09	0.08	100
86	0.07	0.15	0.09	100
87	0.18	0.25	0.21	100
88	0.00	0.00	0.00	100
89	0.00	0.00	0.00	100
90	0.11	0.03	0.05	100
91	0.11	0.20	0.14	100
92	0.04	0.02	0.03	100
93	0.00	0.00	0.00	100
94	0.24	0.54	0.33	100
95	0.14	0.09	0.11	100
96	0.04	0.01	0.02	100
97	0.00	0.00	0.00	100
98	0.04	0.20	0.06	100
99	0.00	0.00	0.00	100
avg / total	0.09	0.09	0.07	10000

Appendix 5: Instructions to run the code

__**Code Files**__

1. CNN.ipynb
2. SVM with HOG.ipynb
3. Random_Forest.ipynb

CNN.ipynb

1. Open the folder entitled "Code"
2. Open the Folder entitled "CNN"
3. Download the dataset from http://www.cs.toronto.edu/~kriz/cifar.html and place the train and test files in the "CNN" folder
4. Open the CNN.ipynb file and follow the comments. Sequentially execute every cell in the notebook.

SVM with HOG.ipynb

1. Open the folder entitled "Code"
2. Open the Folder entitled "SVM with HOG"
3. Download the dataset from http://www.cs.toronto.edu/~kriz/cifar.html and place the train and test files in the "SVM with HOG" folder
4. Open the SVM with HOG.ipynb file and follow the comments. Sequentially execute every cell in the notebook.

Random_Forest.ipynb

1. Open the folder entitled "Code"
2. Open the Folder entitled "Random_Forest"
3. Download the dataset from http://www.cs.toronto.edu/~kriz/cifar.html and place the train and test files in the "Random_Forest" folder
4. Open the Random_Forest.ipynb file and follow the comments. Sequentially execute every cell in the notebook.