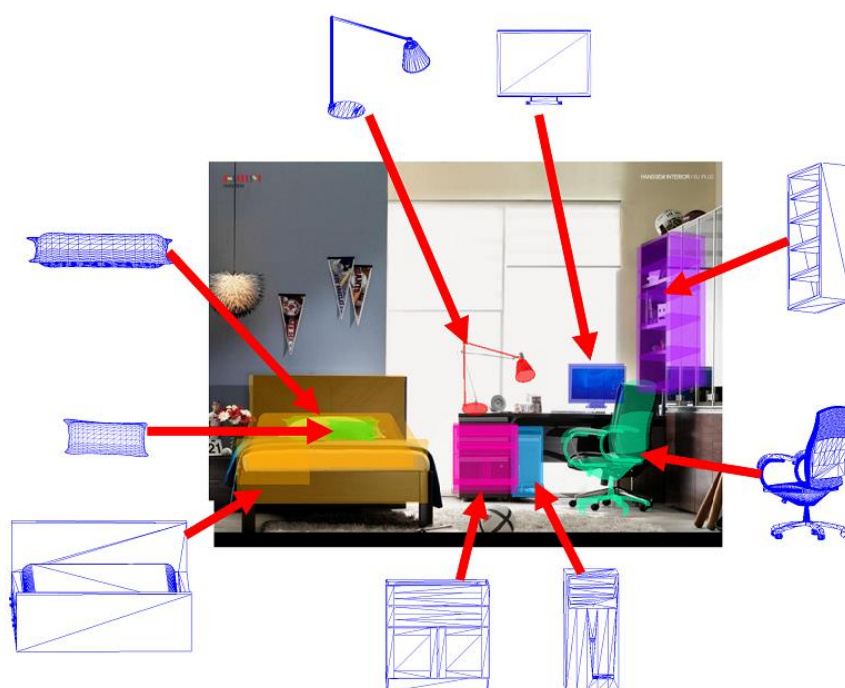

-Task 1-
2-Pipe-Based Ensemble Classifier for Image Recognition



Lecturer: Dr. Ahron Bar-Hillel

Teaching Assistant: Mrs. Liel Cohen-Lavi

1. Introduction

In this report, we propose an ensemble classifier for the task of image recognition. The ensemble model composed of two independent image recognition pipes:

1. HOG-SVM Pipe

2. SIFT-NBNN Pipe

Each one of the abovementioned pipes was optimized separately on Caltech 101 dataset which contains 101 classes, with 31-800 images each. (for more details: http://www.vision.caltech.edu/Image_Datasets/Caltech101)

In addition, the abovementioned pipes were optimized on the first 10 classes only (ascending lexicographic order), referred to as *Fold-1* (see classes below). Moreover, in order to find the best configuration during each pipe hyperparameters optimization process, we used 5-Fold Cross Validation on 20 first images per class (ascending order) by shuffling and dividing the data by a 80%-20% ratio, which referred to as *Train-set* and tested on the rest of the images per class referred to as *Validation-set* respectively. Further, after finding the best configuration using the minimal averaged error, each pipe was tested on the next 20 images per class within Fold-1. Finally, in order to test the abovementioned pipes generalization capability, we configured each pipe to the best configuration found during the hyperparameters tuning step and then we trained and tested each pipe on the next 10 classes within the dataset referred to as *Fold-2* (see classes below).

The report is organized as follows: Sec. 2 discusses the optimization process of the 1st pipe, known as the HOG-SVM Pipe. Sec. 3 discusses the optimization process of the 2nd pipe, known as the SIFT-NBNN Pipe. In Sec. 4 we present how pipes are combined in order to build an ensemble model based on maximum likelihood approach and provide an empirical evaluation of the ensemble model while compared to each pipe separately. Finally, in Sec. 5 we provide our final results.

- Fold-1

['accordion', 'airplanes', 'ant', 'barrel', 'bass', 'beaver', 'binocular', 'bonsai', 'brain']

- Fold-2

['brontosaurus', 'buddha', 'butterfly', 'camera', 'cannon', 'car_side', 'ceiling_fan', 'cellphone', 'chair', 'chandelier']

2. HOG-SVM Pipe Optimization

This section describes the optimization process of the 1st pipe, known as the HOG-SVM Pipe. It was built as a class in order to naturally merge with the 2nd pipe. First, in subsection 2.1 we briefly describe the *Histogram of Oriented Gradients* (HOG) algorithm which is used to extract features from an image in the form of image descriptors. Second, in subsection 2.2 we briefly describe the Support Vector Machine (SVM) with a polynomial kernel algorithm that was used for the task of image recognition. Third, in subsection 2.3 we describe the hyperparameter tuning process for obtaining the best model configuration found to fit our task. Finally, in order to test the model's generalization capability, we present a performance evaluation based on Fold-2 classes using the best selected HOG-SVM model in subsection 2.4.

2.1 Histogram of Oriented Gradients (HOG) algorithm

HOG is a feature descriptor mainly used in computer vision and image processing for the purpose of object detection. Introduced using a different term in 1986, first widespread usage emerged in 2005 with the work of Navneet Dalal and Bill Triggs [3], *Histograms of Oriented Gradients for Human Detection*. HOG descriptor is computed by calculating image gradients and orientation in localized portions thus capturing contour and silhouette information into a histogram. The computation of a HOG descriptor is performed as follows; first, the image gradient is computed row and column-wise, while the image is divided into small spatial regions called "cells". Second, for each cell, a 1D histogram is accumulated of gradient orientations over all of the pixels within a cell. This combined histogram forms the basic "orientation histogram" representation. Third, each orientation histogram divides the gradient angle range into a fixed number of bins. Then, the gradient magnitudes of the pixels within the cell are summed into the orientation histogram. The next stage regards the normalization of local groups called "blocks". Usually, each cell is shared between several blocks, but its normalizations are blocked dependent and thus different. As such, the normalized cell will appear several times in the final output vector. Further, each normalized block descriptor is referred to as a HOG descriptor. Finally, HOG descriptors from all blocks are flattened into a combined feature vector for the use of the classifier. Furthermore, in the original paper, smoothing within cells with *Gaussian Spatial Window* i.e. low-pass Gaussian filter was used. This stage is not performed in the original implementation; therefore, we have implemented it in the stage of the image preprocessing after the resizing phase, and for the HOG-SVM pipe only.

2.2 Support Vector Machine (SVM) with polynomial kernel

SVM is a binary classifier that aims to find a linear hyperplane that separates a given set of instances into two given classes. In addition, SVM is known for its capability to handle a large number of features and robustness to overfitting as the VC-dimension of an SVM linear separator in R^d is d . The SVM attempts to specify a linear hyperplane that with the maximal margin, defined by the maximal (perpendicular) distance between the examples of the two classes. Figure 1 below illustrates a two-dimensional space, in which the instances are located according to their explanatory features; the hyperplane splits them according to their label. The instances closest to the hyperplane are the *Supporting Vectors*. W , the normal of the hyperplane, is a linear combination of the supporting vectors, multiplied by Lagrange multipliers (alphas).

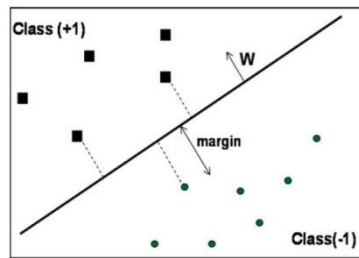


Figure 1 - SVM margin

In most real-life problems, the original space cannot be separated linearly. Therefore, a kernel function K is used. By using a kernel function, the SVM can project the input instances into a higher dimensional space which hopefully creates a linear separation at this higher dimension, moreover, this step is often referred to as the kernel trick. In our assignment, we have decided to use a Polynomial kernel. The Polynomial kernel was chosen after it has shown better performance than the Radial basis function (RBF) kernel over the training set using the hyperparameters in subsection 2.3. The polynomial kernel creates values of degree p , where the output depends on the direction of two vectors $[X_1, X_2]$ which exist at in the original problem space, as shown in the equation below:

$$K(x_1, x_2) = (x_1 \cdot x_2 + 1)^p$$

The kernel is used to evaluate the inner product of X_1, X_2 vectors in the original dimensional space without actually computing it at the higher dimensional space, thus searching for the optimal hyperplane at the higher dimension with relatively low cost in terms of complexity. Obviously, it can result in numerous separating hyperplanes for a specific projection of data; therefore, the hyperplane which maximizes the margin is selected in an attempt to achieve better generalization capabilities with a mean of increasing the expected accuracy.

2.3 HOG-SVM Hyperparameter Tuning

In order to select the best hyperparameters configuration for our HOG-SVM pipe, we had to consider three types of hyperparameters for three continues steps: image processing, HOG creation, and the polynomial SVM tuning. Each combination of hyperparameters is built on top of the previous one and the integration of all three effects the validation error directly. In order to obtain the three best configurations group, referred to as *Ultra-Set*, we have implemented a grid search over the hyperparameters space using a 5-Fold Cross-Validation (CV) on Fold-1 classes first 20 images of each class (classes were sorted by ascending order). Although computationally expensive, by using CV the training error is being averaged and therefore, a more reliable result for deciding which hyperparameter configuration should be used, is given. Further, instead of tuning each hyperparameter separately, which can result in poor performance, we tuned all hyperparameters simultaneously in order to consider the interaction between them into account as well.

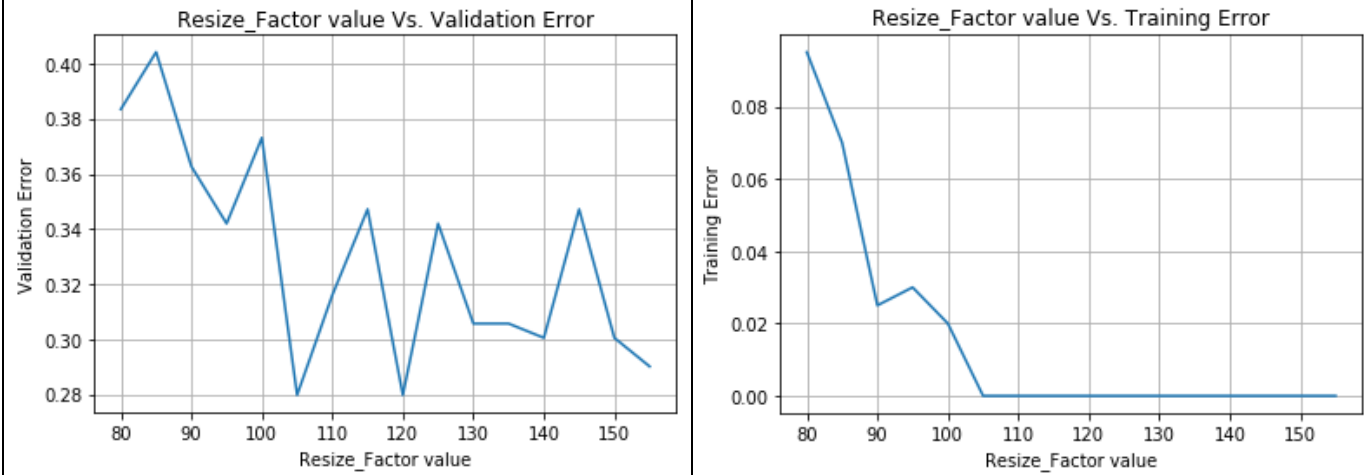
The Ultra-Set found:

Parameter	Value	Step
Resize Factor	120x120	Image Processing
Resize Interpolation Algorithm	Cubic	Image Processing
Low-Pass Filter Size	(5,5)	Image Processing
Orientation Bins	11	HOG Creation
Pixels Per Cell	(15,15)	HOG Creation
Cells Per Block	(2,2)	HOG Creation
Block Norm	L2	HOG Creation
C	0.1	Polynomial SVM Tuning
Gamma	0.1	Polynomial SVM Tuning
Degree	2	Polynomial SVM Tuning

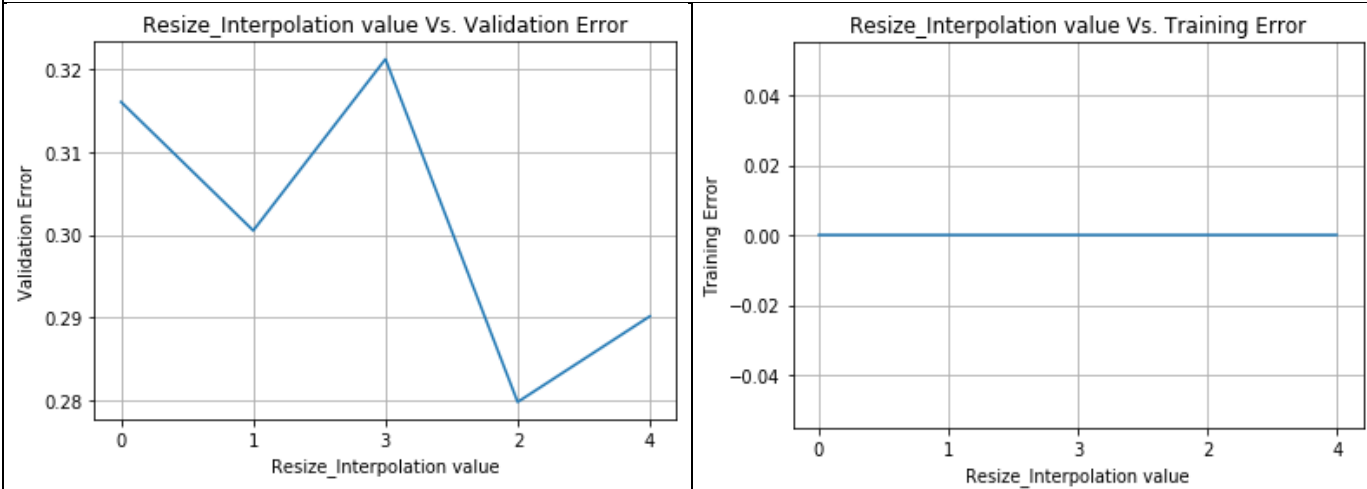
The table below shows the validation and training errors vs. each hyperparameter tuning implemented on Fold-1. In order to create the below graphs, first, the best Ultra-Set had to be obtained by the above-mentioned CV, then the classifier had to be trained and tested again across each hyperparameter span according to the following technique; except for the measured hyperparameter, all other hyperparameters were configured to the best values found i.e. the Ultra-Set, then the classifier was trained and tested across a single hyperparameter space while the rest hyperparameters were “frozen”, resulting with a validation and training errors across the measured hyperparameter span. In addition, the span was sampled by logical intervals configured for each hyperparameter.

Validation and Training Errors vs. Hyperparameter Tuning

Resize Factor: The resize Factor is the resolution set for each image that is processed into the dataset. This was done for unformal reasons and later on, proved to be high inflectional on the training and validation errors. Two different points were examined as potential values for the Ultra-Set [105,120], though eventually, 120 had a slightly lower validation error.

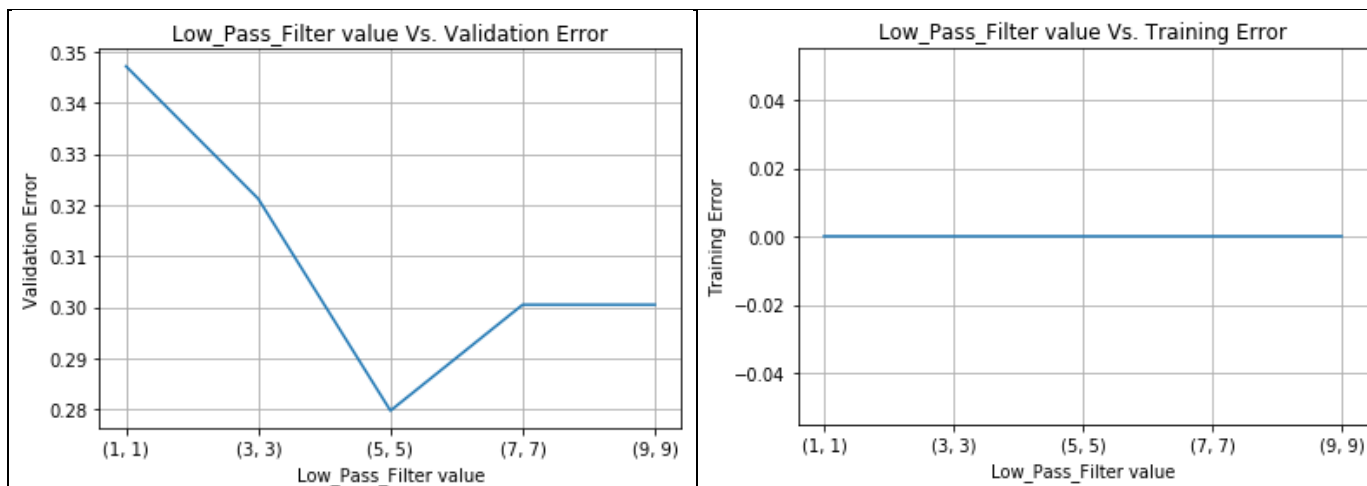


Resize Interpolation: Image interpolation was used due to the resizing done to the image thereby distorting it from one pixel's grid to another. Interpolation works by using known data to estimate values at unknown points. Image interpolation works in two directions and tries to achieve the best approximation of a pixel's intensity based on the values at surrounding pixels. Common interpolation algorithms can be grouped into two categories; adaptive and non-adaptive. In our case, all four interpolation algorithms; Nearest, Linear, Area, Bicubic and Lanczos4 are non-adaptive. As seen in the validation error graph, choosing the right algorithm has a tremendous effect on the accuracy of our model.

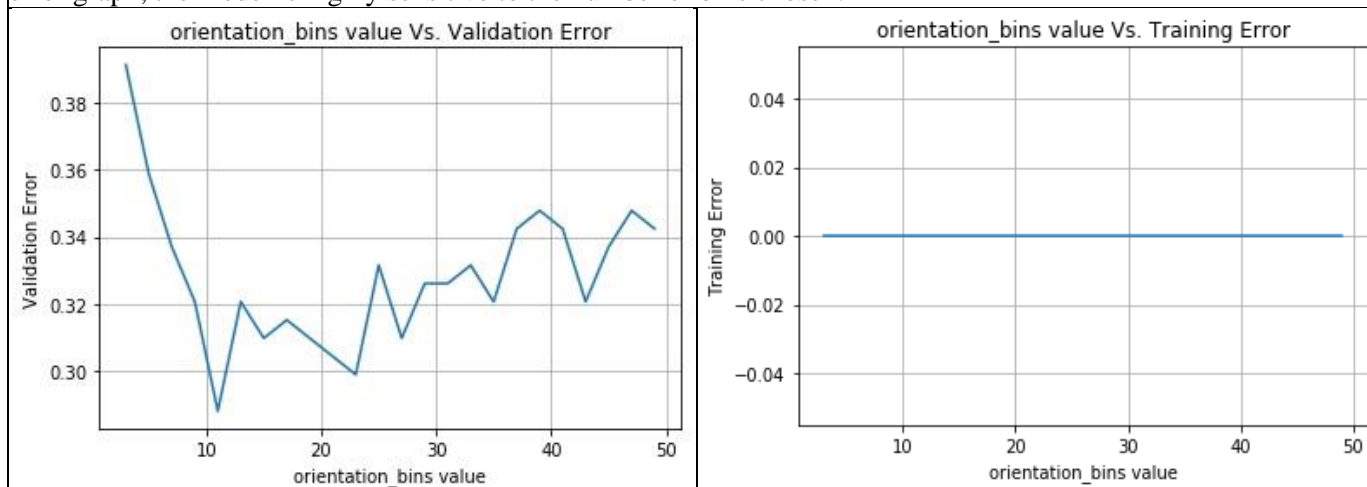


1: Nearest, 2: Linear, 2: AREA, 3: Bicubic, 4: Lanczos4

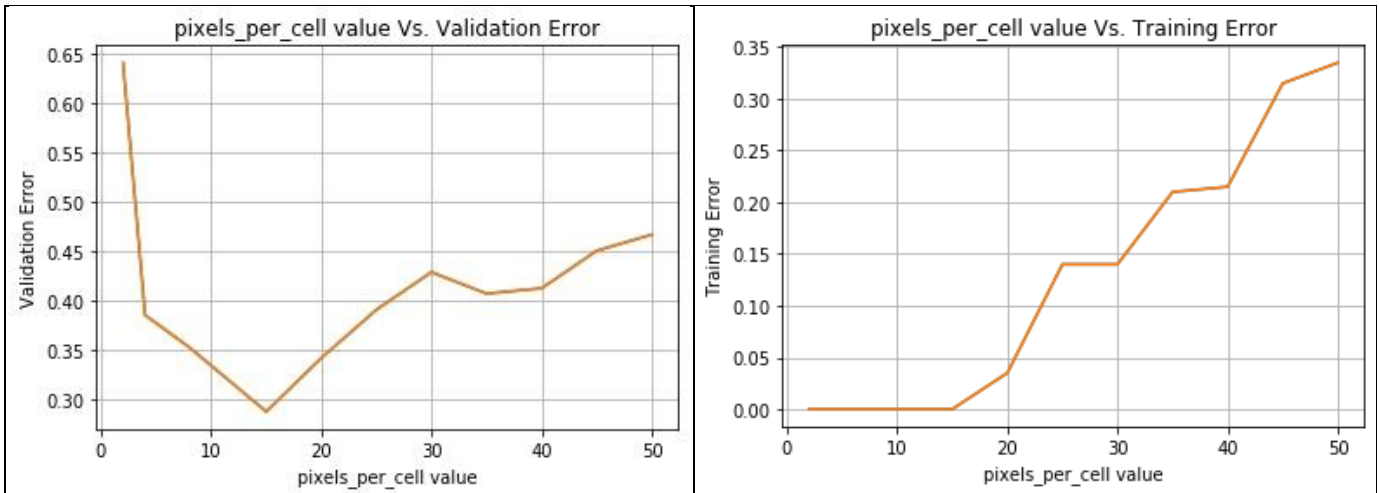
Low-Pass Filter size: A low pass filter, also know as smoothing or blurring filter, is intended to average high changes in intensity e.g gradients in our case, across the image. As such it is also used for noise reduction. In the original paper by Navneet Dalal, in order to down weighting the pixels near the edges of the block, a Gaussian spatial window was applied to each pixel before accumulating orientation votes into bins. This has shown to improve performance. As seen in the validation error graph, choosing the right filter size has a tremendous effect on the accuracy of our HOG-SVM pipe.



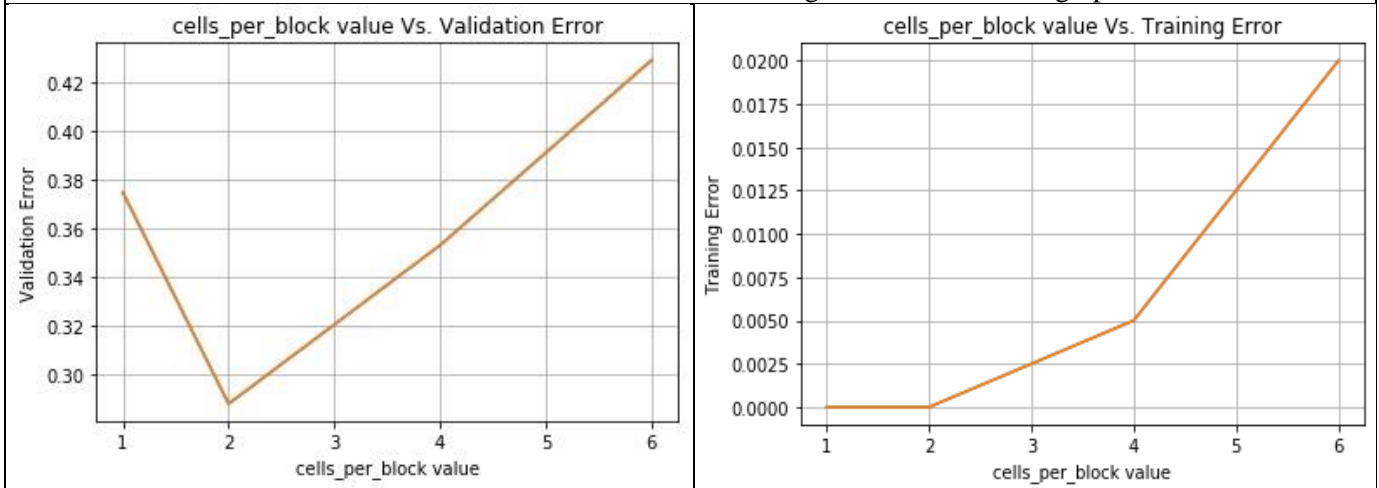
Orientation Bins: Each image is divided into small spatial regions called "cells". For each cell, we have computed a local 1-D histogram of gradient over all the pixels in the cell. This forms the basic "orientation histogram" representation. Each orientation histogram divides the gradient angle range into a fixed number of predetermined bins, i.e. the hyperparameter "Orientation Bins". The gradient magnitudes of the pixels in the cell are distributed between orientation histogram bins according to the pixel's orientation and bins width. As seen in the validation error graph, the model is highly sensitive to the number of bins chosen.



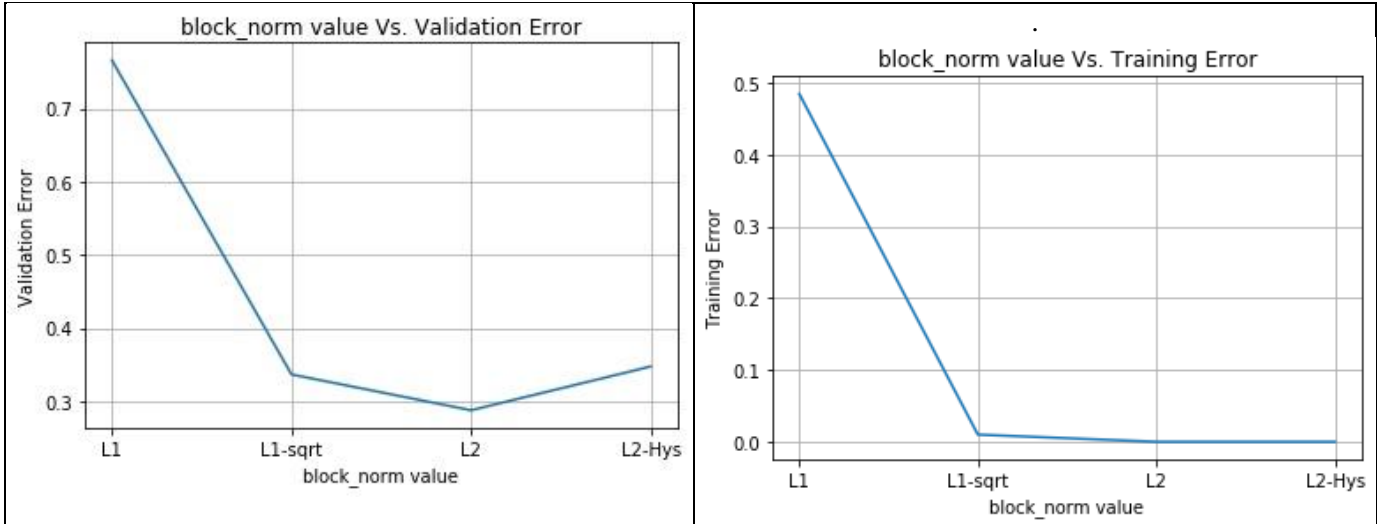
Pixels Per Cell: This hyperparameter denotes the number of pixels per cell, as in HOG the image is divided into spatial cells according to the number of pixels in each cell. This hyperparameter has shown to have a tremendous effect both on the validation error and the training error as seen by the instance error movements in the graphs below.



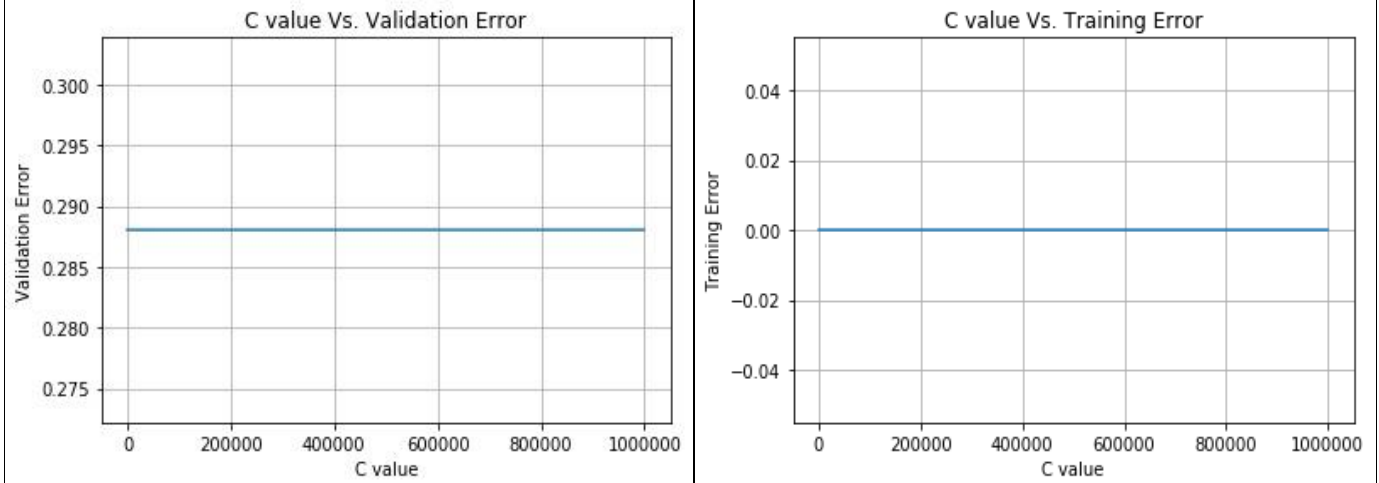
Cells Per Block: This hyperparameter denotes the number of cells per block, which correspond to a crucial step in HOG feature creation, referred to as Block Normalization. Rather than normalize each histogram individually, the cells are grouped into blocks and normalized based on all histograms in the block. Normalization introduces better invariance to illumination, shadowing, and edge contrast. In our case, the cells per block were (2,2) which means that most pixels were normalized several times. In Scikit-Image (skimage) implementation, each cell thus appears several times in the final output vector according to the different normalizations. This hyperparameter has shown to have a tremendous effect both on the validation error and the training error as seen in the graphs below.



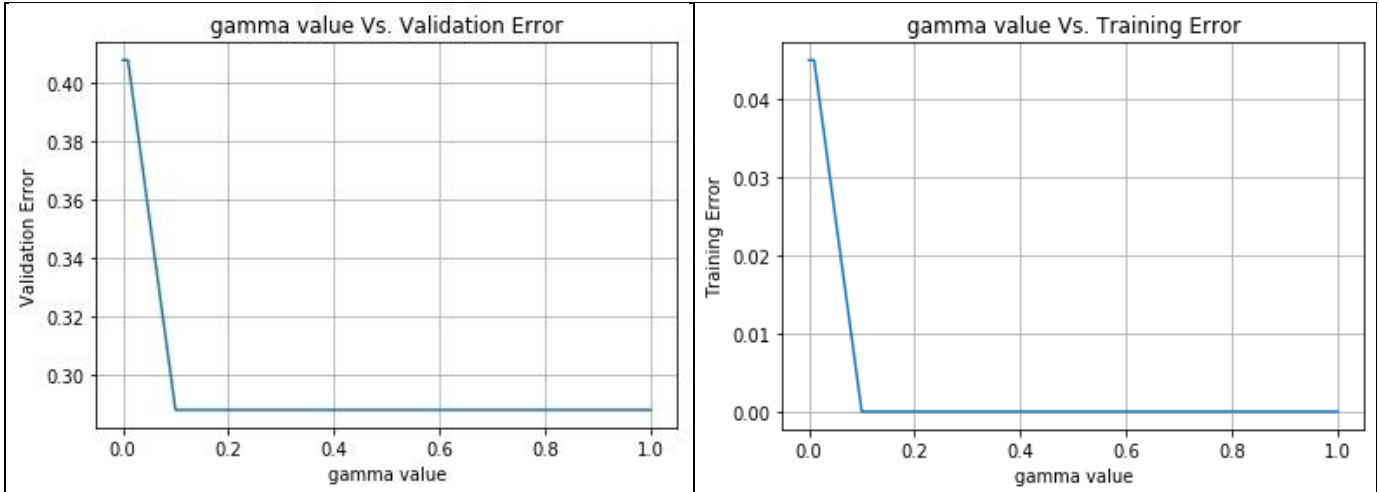
Block Norm: This hyperparameter denotes the different options of block normalization methods. In our case, 'L2' was chosen, as seen in the graphs below.



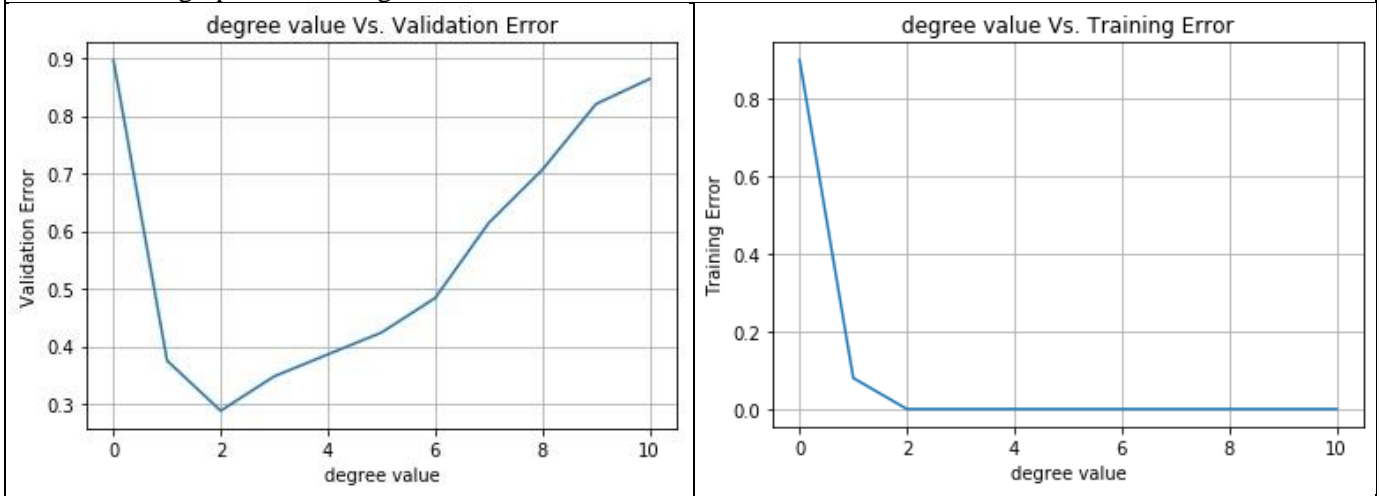
C: Parameter C (i.e. Cost) controls the tradeoff between hard SVM to soft SVM. As seen by the graphs below, in our case the effect of the cost parameter was marginal across the wide span tested.



Gamma: The Gamma parameter in the context of polynomial SVM serves as a scaling parameter in the form of $K'(x,y) := (\gamma \langle x,y \rangle + r)^d$ as $\gamma = \gamma$, (it has another meaning in RBF SVM). Here gamma is a parameter that ranges from 0 to 1. A higher value of gamma will push the hyperplane to perfectly fit the training dataset, which may cause over-fitting. As seen in the graphs below, after the point of 0.1 the error does not change. Thus, we have decided to choose $\text{Gamma} = 0.1$.



Degree: Denotes the degree of the polynomial kernel function, e.g. degree = 1 is a special case i.e. linear SVM. As seen in the graphs below, degree = 2 was chosen for our Ultra-Set.



2.4 HOG-SVM Model Generalization Evaluation using Fold-2

In order to test our HOG-SVM generalization capability, in this section, we present a performance evaluation of the best-selected model found according to the previous section, conducted using *Fold-2* classes.

2.4.1 Metric Summary

Metric	Value
Validation Error	~22.9%
Training Error	0%
Accuracy	77%

2.4.2 Confusion Matrix

Confusion matrix breaks down the validation error, thus enabling us to observe which classes had a high True Positive Rate (TPR) and which had a high False Positive Rate (FPR). As seen in the figure below, the SVM-HOG classifier had high TPR for the accordion & airplane classes, but it had high FPR for the bass & binocular classes.

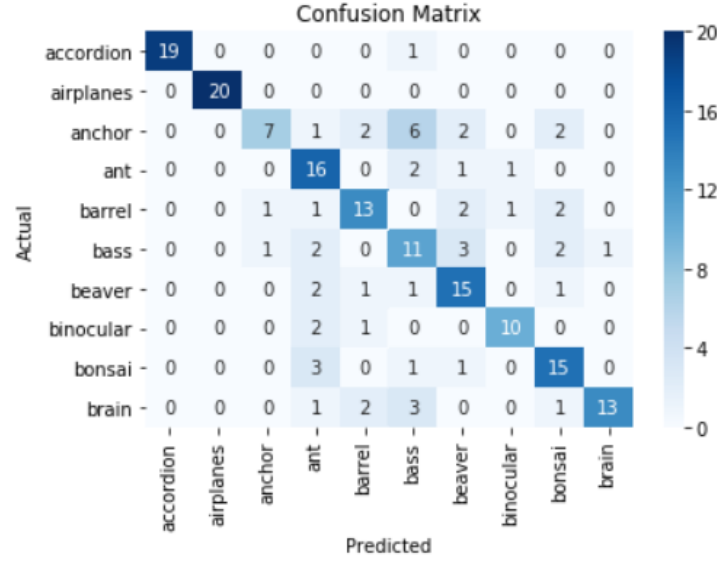


Figure 2 - HOG-SVM Pipe Confusion Matrix using Fold-2 classes

3. SIFT-NBNN Pipe Optimization

This section describes the optimization process of the 2nd pipe, known as the SIFT-NBNN Pipe. First, in subsection 3.1 we describe the *SIFT* algorithm which is used to extract features from an image as a form of image descriptors. Second, in subsection 3.2 we describe the *NBNN* algorithm which is used here for the task of image recognition. Third, in subsection 3.3 we describe the hyperparameter tuning process for finding the best NBNN model configuration by optimizing both hyperparameters Size (S) and Stride. Finally, in order to test the model's generalization capability, in subsection 3.4 we present a performance evaluation conducted on Fold-2 classes using the best selected NBNN model.

3.1 SIFT Descriptors

In 2004, Lowe [4] proposed a new algorithm called *Scale Invariant Feature Transform* referred to as *SIFT*. The suggested algorithm is used to extract distinctive image features from Scale-Invariant Keypoints, by extracting key points and compute a descriptor for each key point. When a keypoint descriptor is created, a 16x16 neighborhood around the key point is taken. Then, each neighborhood is divided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histograms are created. As a result, a total of 128 bin values (see below calculation) is created and represented as a vector to form a keypoint descriptor. In addition, note that SIFTs considered to be robust against illumination changes, rotation, etc.

$$SIFT \text{ Descriptor Bins Histogram} = N_{sub-blocks} \cdot N_{orientation} = 16 \cdot 8 = 128 \text{ bins}$$

Furthermore, since SIFT descriptors are computed at points on a regular grid with a stride of M pixels, we needed to choose a value for M . Moreover, based on Bosch et al. [2] work of classifying images on Caltech-101 & Caltech-256 datasets which showed reasonable performance results, we tested different stride values around 10 pixels for M . An Example for Raw image before and after SIFT extraction using fixed key points is described in Fig.3 below.

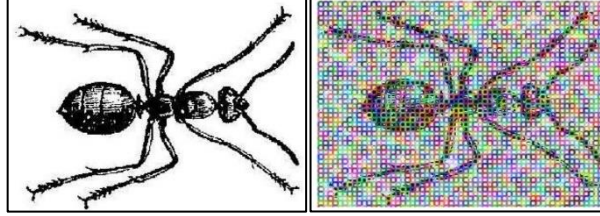


Figure 3 - Raw image before SIFT (Left); image after SIFT (Right);

3.2 NBNN Classifier

A simple nonparametric NN-based classifier called *Naïve-Bayes Nearest-Neighbor* (NBNN) was proposed in 2008 by Boiman et al. [1]. NBNN requires no descriptor quantization and employs a direct "Image-to-Class" distance. In addition, the authors showed that under the assumption of independence among predictors based on Bayes' Theorem, the theoretically optimal image classifier can be accurately approximated by the NBNN algorithm. In addition, since the input of the NBNN classifier composed of a batch of images' descriptors with a dimension of $N_{descriptors} \times 128_{bins} \times N_{images}$ it results in relatively high dimensions. Moreover, a nearest neighbor's search is being performed to find the 1-nearest-neighbor for each image descriptor using a distance metric. As a result, for each class within the train set, all of the descriptors are grouped and inserted into a *KD-Tree* data structure, which is used for the efficiency of NN-search. Further, for multi-class classification, a set of KD Trees needs to be implemented, referred to as *KD-Forest*. Furthermore, note that the preprocessing step has a low complexity of $O(N \cdot \log(N))$. In addition, it has a low runtime of ~ 3 seconds for constructing the KD-Forest for all classes. Furthermore, the time complexity for one image query search within a KD-Forest is: $O(N_{class} \cdot N_{descriptors} \cdot \log(N_{descriptors}))$. Although there is no training involved with the NBNN model, and from terminology aspects, we defined **m_classes_NBNN_train(..)** function "training" process to be in charge of the SIFTs KD-Forest creation. Further, we defined **m_classes_NBNN_test(..)** to be in charge of classifying unseen images by querying each class' KD-Tree within the KD-Forest for nearest neighbors search which is then used for image-to-class dissimilarity calculation using **NBNN_class_Score(..)**. The lowest image-to-class dissimilarity score will then be picked and result in classification.

3.3 NBNN Hyperparameter Tuning

In order to select the best hyperparameters configuration for our NBNN model which are Image Size (S) & Stride, we used **KfoldCrossValidation(..)** function to conduct a 5-Fold Cross-Validation (CV) on Fold-1 classes in general and in particular, on the 20 first images (by ascending order) with a ratio of 80\20 between both *Train* and *Validation* sets respectively. Although computationally expensive, by using CV the training error is being averaged and therefore, provides more reliable results in order to determine which hyperparameter configuration outperformed. Further, instead of tuning each hyperparameter separately, which can result in much poor performance, we tuned both hyperparameters simultaneously which takes the interaction between into account. The experiment configuration is described below.

3.3.1 Hyperparameters Optimization - Experiment 1 Configuration

In this experiment, using **ExecuteExperiment_1_NBNN()** procedure, we used different hyperparameters values in order to find the best candidates for optimizing our NBNN model. The hyperparameters to be optimized are the Image Size (S) and Stride. Further, we used a fixed split ratio of 80\20 between both *Train* and *Validation* sets respectively. In order to see the interaction between both hyperparameters, and since only two hyperparameters needed to be tuned within our NBNN model, we used a grid search of 49 different combinations using a 5-Fold CV ($7 \times 7 = 49$) procedure which describes below. In addition, because of time and performance constraints, we picked a range of relatively small image sizes, and stride values around 10 following Bosch et al. [2] article, to be tested.

- 7 Image Size Candidates = [90, 95, 100, 105, 110, 115, 120]
- 7 Stride Candidates = [8, 9, 10, 11, 12, 13, 14]

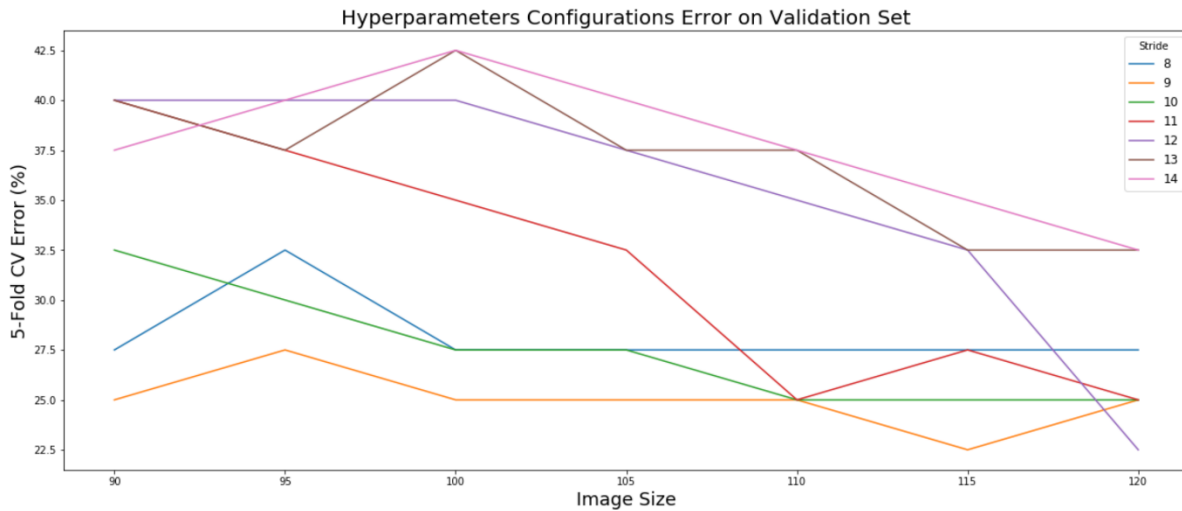


Figure 4 - 5-Fold Cross Validation results using a Grid Search

By viewing Fig. 4 graph, generated using our **plot3DErrorPerformanceChart()** function, we can infer that there are two hyperparameters candidates that outperformed the rest and achieved the same averaged error score of 22.5%. In addition, unlike the SVM, the NBNN is a non-parametric method, meaning that there are no learned parameters within this model such as weights, support vectors, etc. Further, note that the variability of the descriptors within the KD-Forest directly depends on the *Train* set used to build it. Therefore, we didn't present a training error graph which is 0% constant. Furthermore, in order to determine the best hyperparameter candidate pair among those who got the lowest average error, we decided to conduct another experiment.

3.3.2 Hyperparameters Optimization - Experiment 2 Configuration

In the 2nd experiment, in order to determine which candidates pair derived from the 1st experiment outperformed the rest, we tested each pair robustness by using a 20-fold Cross-Validation with a fixed split ratio of 85\15 between both *Train* and *Validation* sets respectively and moreover, we used a Gaussian Blur filter which causes image edges to be blurred a bit, from which this operation can result in a more robust classifier.

The final candidates' pairs configurations are:

1. (Size: 115, Stride: 9)
2. (Size: 120, Stride: 12)

Experiment results showed that the 2nd candidate pair (120,12) achieved an averaged error of 35% while the 1st candidate pair (115,9) outperformed with an averaged error of 30%. As a result, we chose an image size of 115 and a stride of 9, referred to as the best NBNN hyperparameters combination. Furthermore, after setting the best-derived configuration, we trained the model using the Fold-1 train set and tested it using the test set which results in an accuracy of $\sim 69.95\%$.

3.4 NBNN Model Generalization Evaluation using Fold-2

In order to test our NBNN generalization capability, in this subsection, we present a performance evaluation of the best selected NBNN model found in the previous subsection, using *Fold-2* classes. Results showed that an accuracy score of 69.5% achieved on Fold-2 classes. Moreover, in Fig. 5 a confusion matrix is described below.

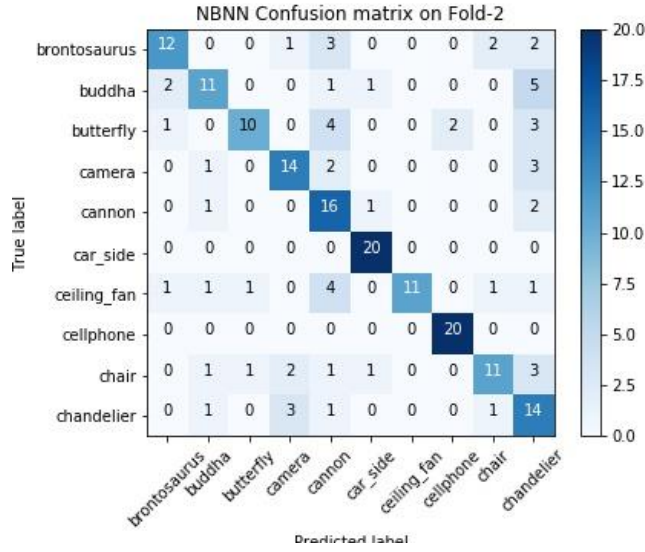


Figure 5 – Confusion Matrix of best NBNN model configuration on Fold-2 classes

4. Ensemble Model Optimization and Evaluation

In the previous sections, we presented how each pipe was optimized independently on Fold-1, which resulted in finding the best hyperparameters configuration for each pipe. In this section, we present our ensemble model which combines both HOG-SVM and SIFT-NBNN pipes to a unified model. First, in order to combine both pipes' scores, we needed to calculate a unified scoring function for our proposed ensemble model referred to as **ensembleScoreCalculation()**. Although each pipe returns a vector of distances between a test sample to each class classifier, note that in case of the HOG-SVM pipe, we prefer to achieve high distance between the observed test sample to the SVM margin which makes the classification much more significant, while for the SIFT-NBNN pipe, high distance implies on higher dissimilarity which we want to minimize. Therefore, in order to treat both pipes' scores as a maximization problem, the SIFT-NBNN pipe's distances were multiplied by -1. Further, in order to transform pipes' distances to probabilities, we used our softmax function **calculateSoftmax()** on each pipe distance vector. Furthermore, in order to classify an image, we used a weighted average, in order to aggregate both pipes' probabilities to a unified probabilities vector. Second, in order to determine the appropriate pipes' weights, we conducted another experiment using 5-Fold cross-validation with a 90\10 ratio between train and validation sets respectively using Fold-1 classes and our **EnsembleKfoldCrossValidation()** function. During the last experiment,

we tested different weights combinations in a range between [0,1] with a step size of 0.05 between, in order to see which weights' combination minimizes the error of our ensemble model. In addition, Fig. 6 presents a performance comparison in order to determine if our ensemble model outperformed each pipe independently while comparing each pipe's independent score to the score achieved using our suggested ensemble model. Note that when the NBNN weight is set to 0%, our ensemble model takes into account only the SVM output, and when the NBNN weight is set to 100%, our ensemble model will use the NBNN classification only.

Experiment results showed that our suggested ensemble model outperformed each independent pipe with an error of 15.6% (accuracy of 84.4%) when the SIFT-NBNN pipe's weight defined in a range between 55%-65%. Further, when the NBNN weight was set to 0% an averaged error of 17.66% was achieved, and alternatively, when the NBNN weight was set to 100% an averaged error of 62.21% was achieved.

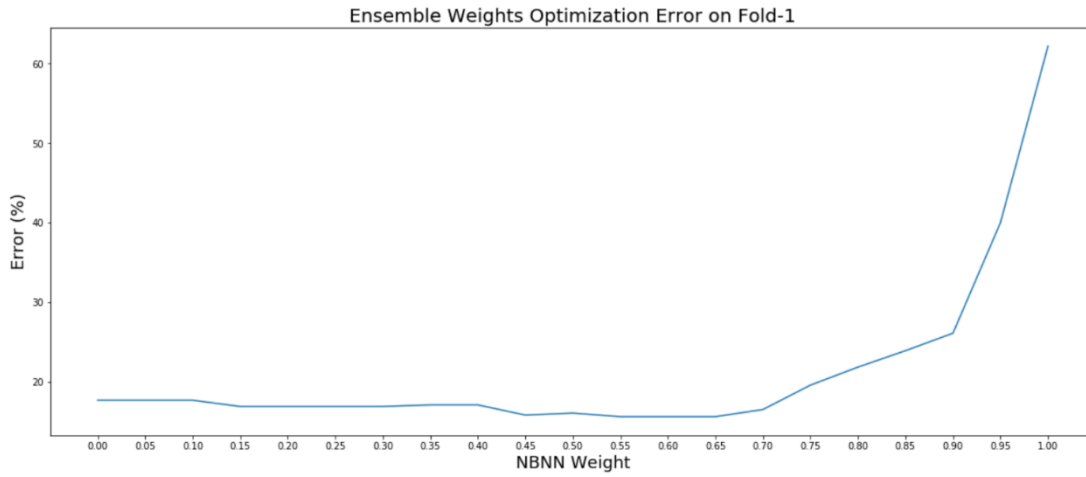


Figure 6 - 5-Fold Cross Validation results for finding best ensemble weights

Furthermore, since there were few consecutive points that achieved the same averaged error, similar to the previously described experiment, we conducted an experiment using 5-Fold CV with a ratio of 90\10 in order to explore the NBNN weight area between [0.55,0.74] only with a lower granularity step size of 0.01, that is in order to avoid from arbitrary choice and to empirically determine which weight set outperformed. Figure 7 presents the last experiment results showed that the knee cap point located where our NBNN weight is set to 68%, which resulted in a mean error of 15.6%. Therefore, we picked the 68% weight for our SIFT-NBNN pipe and 32% for the HOG-SVM pipe as our best combination.

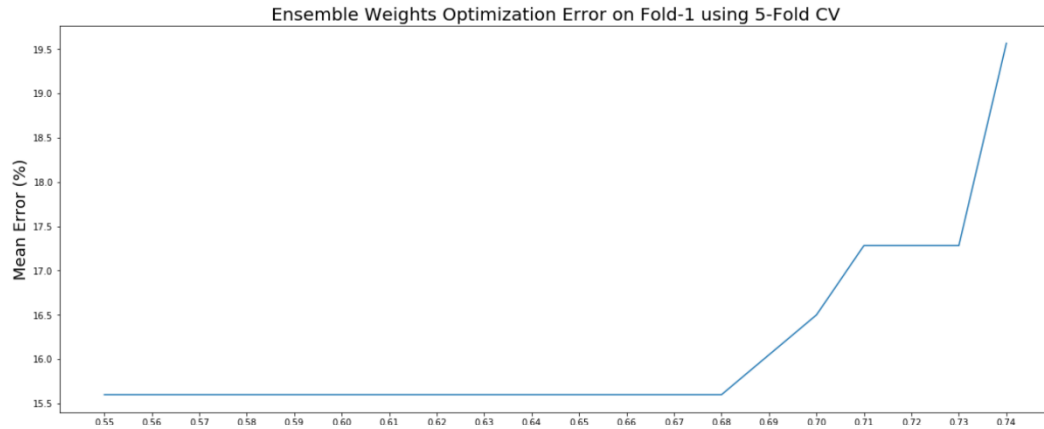


Figure 7 - A Drilled Down 5-Fold Cross Validation results for finding best ensemble weights

5. Results

After specifying how the ensemble weights were optimized and set up in the previous section, we evaluated our tuned ensemble model on Fold-2 classes which result in an accuracy score of **78%**. In addition, the final confusion matrix achieved is described in Figure 8 below.

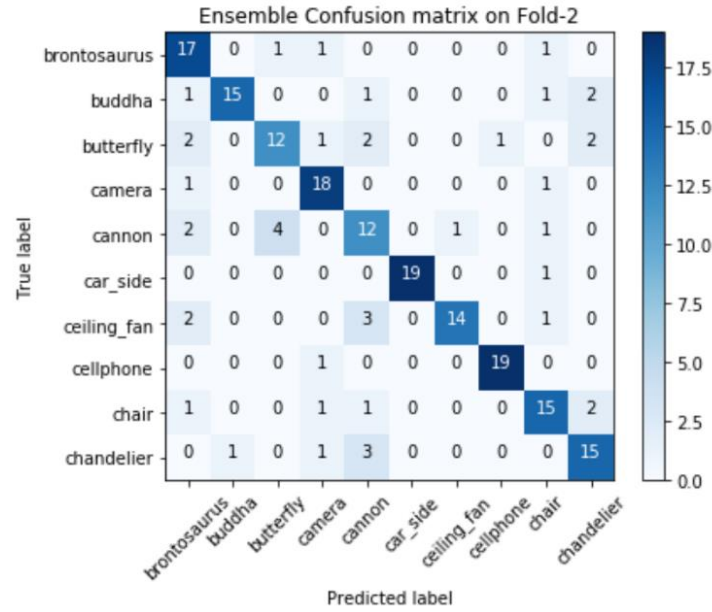

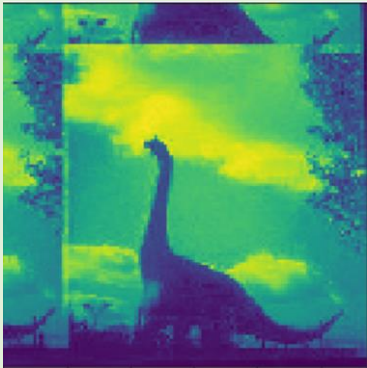
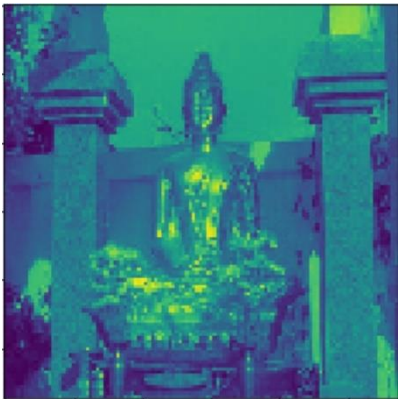


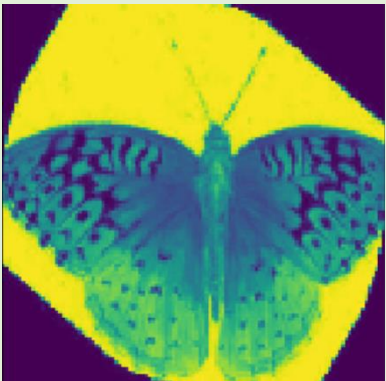
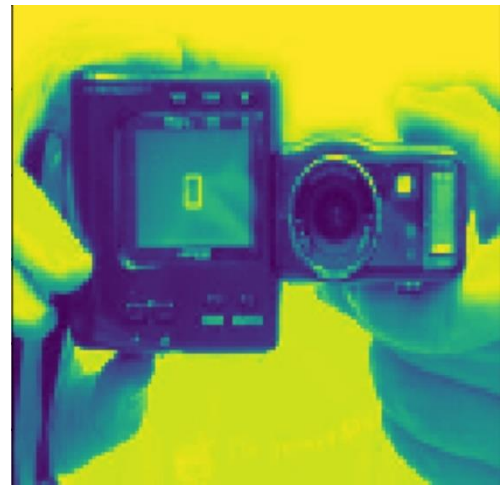


Figure 8 - Final Ensemble model confusion matrix

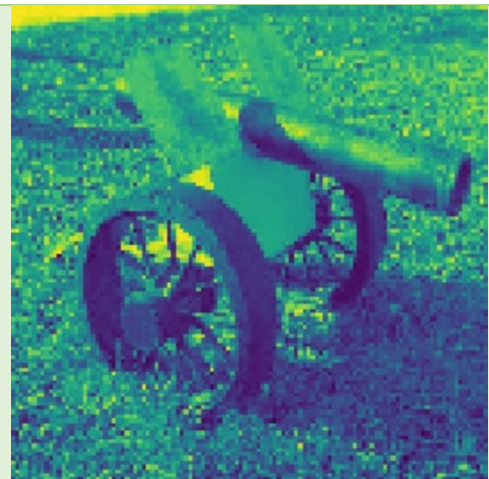
Further, by using our **ErrorVisualization()** and **CreateErrorGrid()** functions we were able to present an error analysis for the top 2 (or less in case of one misclassified image or none) largest errors of misclassified images per each class. Furthermore, note that we calculated the largest errors by the following procedure: First, we extracted the misclassified images for each class. Second, since both NBNN and SVM distances were converted to probabilities vectors, we extracted the True class probability vector. Third, we extracted the maximum probabilities achieved per each misclassified sample within the classified class in order to calculate the error between the true class probability to the predicted class probability. Finally, we selected the top 2 highest errors and their images respectively. The table below presents the error analysis for each class within Fold-2 classes.

Class	Highest Error	2 nd Highest Error
Brontosaurus		
Buddha		
Butterfly		

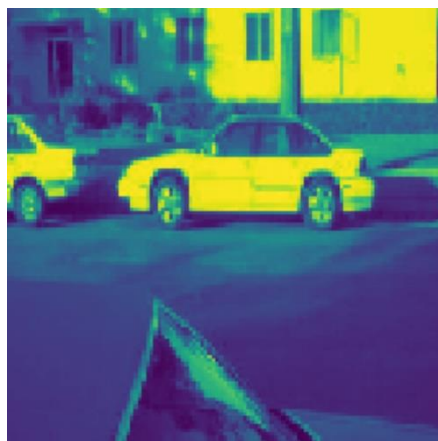
Camera



Cannon



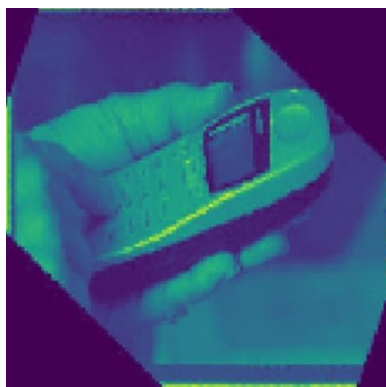
**Car
side**



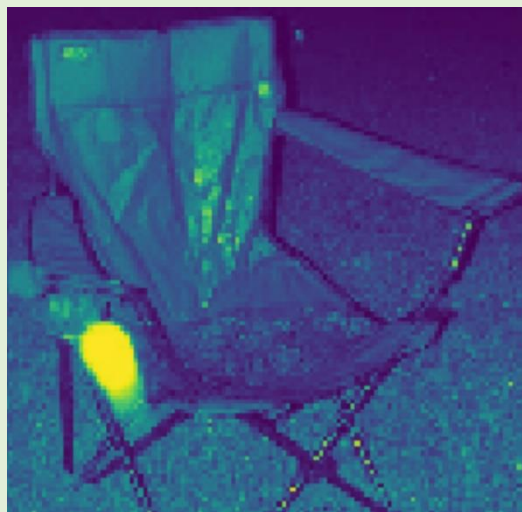
**Ceiling
Fan**


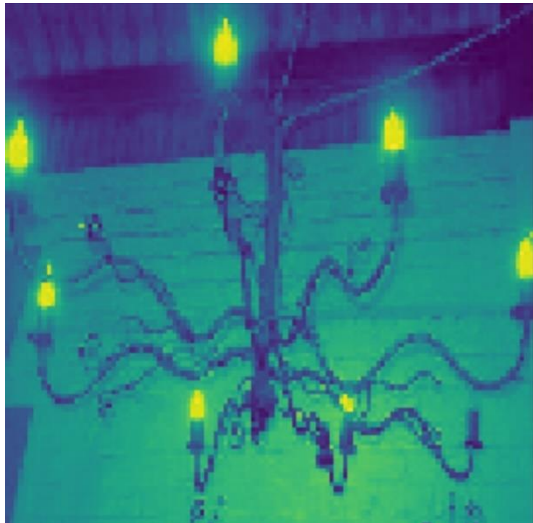


Cellphone



Chair



<p>Chandelier</p>		
--------------------------	---	--

6. Reference

- [1] Boiman, O., Shechtman, E., & Irani, M. (2008, June). In defense of nearest-neighbor based image classification. In 2008 IEEE Conference on Computer Vision and Pattern Recognition (pp. 1-8). IEEE.
- [2] Bosch, A., Zisserman, A., & Munoz, X. (2007, October). Image classification using random forests and ferns. In 2007 IEEE 11th international conference on computer vision (pp. 1-8). IEEE.
- [3] Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection.
- [4] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2), 91-110.