Capstone Project

Udacity Machine Learning Engineer Nanodegree

Git repo:
https://github.com/joshnewnham/udacity_machine_learning_engineer_nanodegree_capstone

Define

This project is based on the paper **How Do Humans Sketch Objects?** published in journal ACM Transactions on Graphics (Proc. SIGGRAPH 2012) by **Mathias Eitz, James Hays and Marc Alexa**, details available http://cybertron.cg.tu-berlin.de/eitz/projects/classifysketch/

The paper compares the performance of their sketch recognition system with people, citing people accurately identified 73% of the object category from sketches while in comparison, their system achieved 56% accuracy (across 250 categories). The authors argue that sketch is a more universal than writing and reading, therefore better computational understanding of sketches will lead to better computer accessibility.

The complete crowd-sourced dataset of sketches to the community was released under Creative Commons and the dataset used here.

This project follows their agenda (and general approach) of trying to **accurately classify a (trained) category given a sketch from the user** (*using a subset of their data*). The motivation for choosing this was due to its close relationship to my work (Design and Machine Learning/Intelligent Interfaces).* with focus on opportunities for user augmentation.

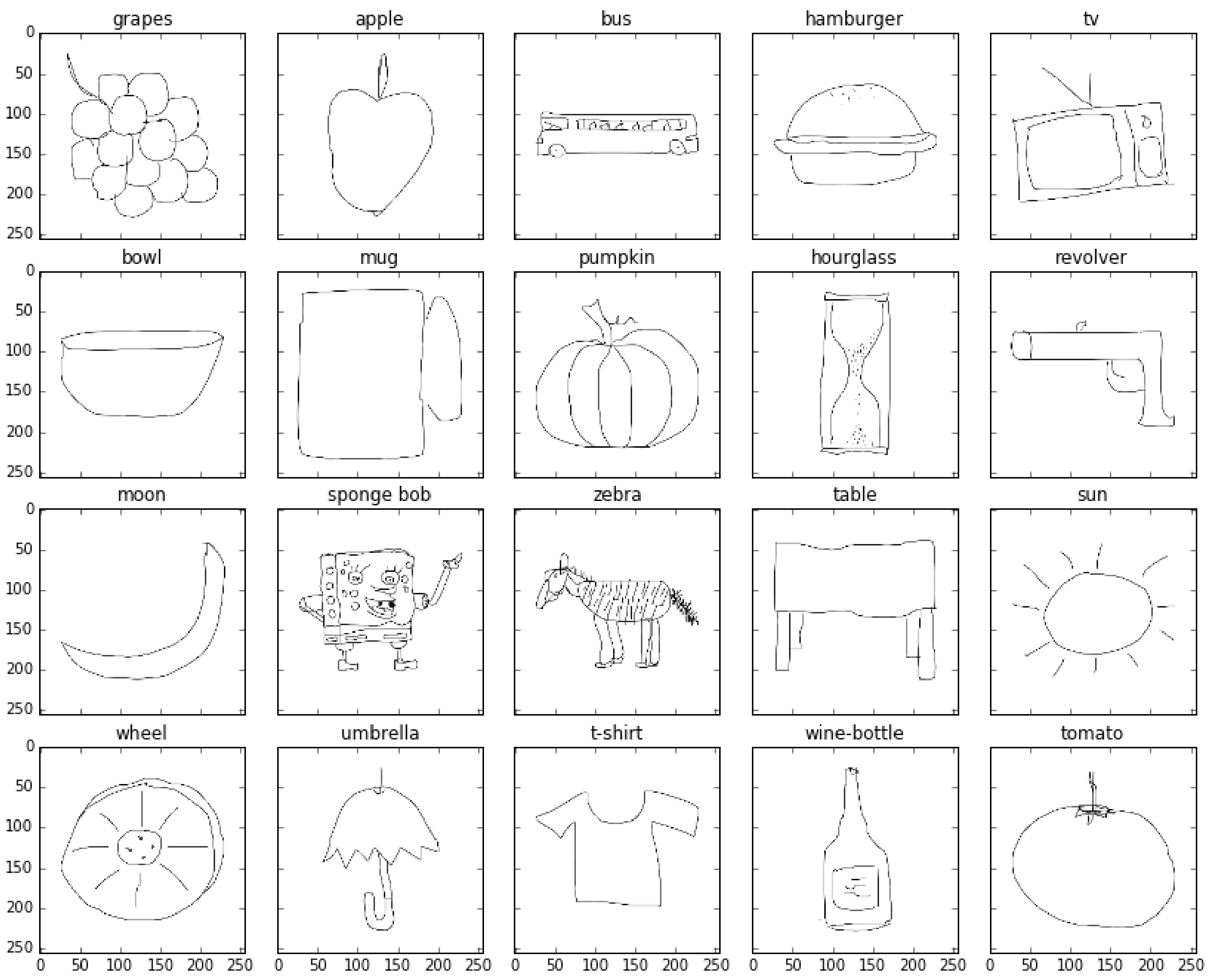
Analyze

Dataset

The dataset is available http://cybertron.cg.tu-berlin.de/eitz/projects/classifysketch/sketches_png.zip and released publicly under the Creative Commons license.

The **full dataset consist of 250 categories** each containing 80 samples (20000). SVG and PNG versions are made available, where each sample (PNG) consists of a 1111x1111 sketch of the relevant category object, files organised into their categories within sub-directories.

Because of computing demands a **subset of 113 categories** were selected, this list can be found in the accompanying subset_labels.csv file. For illustrative purposes, below shows 2 images from categories.



Test Set

A randomly generated test set was created to be used for evaluation throughout the process. This test set is defined in the accompanied file test_set.json, containing a dictionary structure for each category and corresponding file names. 8 (10%) of the images were selected from each category to form this test set.

Evaluation

Performance of the features and model are based on:

Log Loss (Logarithmic Loss):
Log Loss quantifies the accuracy of a classifier by penalising false classifications i.e. minimising the Log Loss is basically equivalent to maximising the accuracy.
Log Loss heavily penalises classifiers that are confident about an incorrect classification e.g. if for a particular observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the Log Loss will be very large.
More details can be found: https://www.kaggle.com/wiki/LogarithmicLoss

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

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}(y_i\log(p_i) + (1-y_i)\log(1-p_i))$$

Accuracy score:
The fraction of correct predictions.

Accuracy of the top 10:
For each k, how accurate is the model i.e. when k is set to 2, then consider a correct prediction if the predicted probability includes the correct prediction within the top 2 predictions.

Benchmark

In this project I'm basing the benchmark on the performance achieved by the reference paper (http://cybertron.cg.tu-berlin.de/eitz/projects/classifysketch/) where they were able to identify unknown sketches with 56% accuracy.

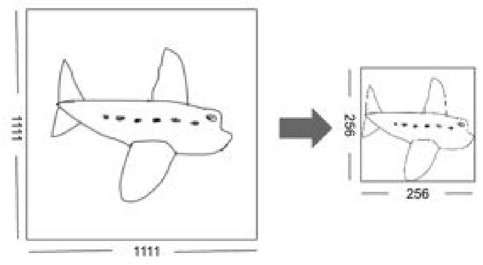
Implement

As described above, the goal of this project is to be able to correctly assign a learn't category from a given user's sketch, the training set includes **113 categories**. From each category I'm using **72 images for training** and leaving out **8 images for testing**.

In this section I introduce these features and the extraction process along with some exploration of (what I have called) feature tuning i.e. tuning the features to increase performance of classification.

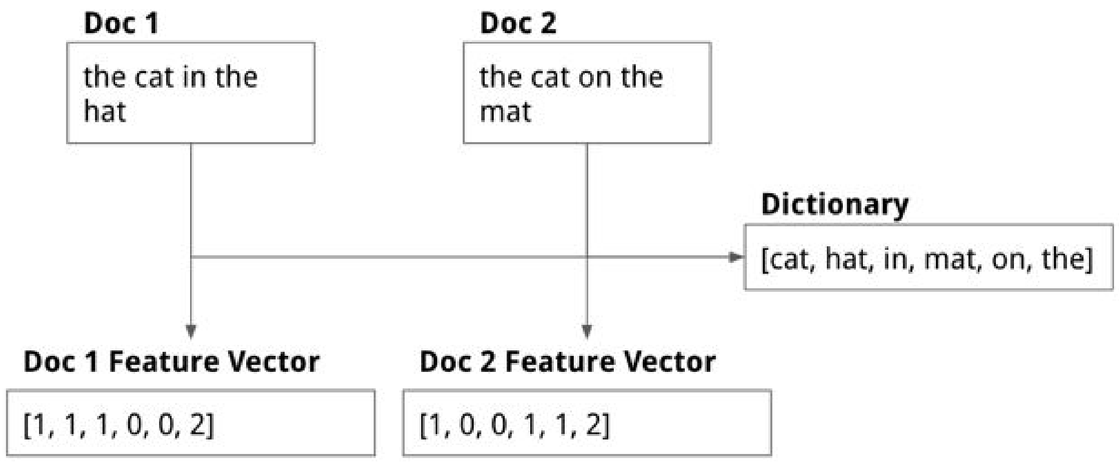
Image pre-processing

Prior to extracting features from the images, images were first pre-processed - the original images were 1111x1111 PNG's, with varying scales. To mitigate variance in scale, each images padding was cropped and the image uniformly rescaled to 256x256.



Describing images with Bag of Visual Words

In text analysis, an effective technique for classifying documents is by using the concept Bag of Words (BoWs), this is essentially counting words in each document, and using this ‘bag of words’ to measure the similarity (distance) between other documents. This ‘bag of words’ defines the feature vector for each document, this vector is defined by a global vocabulary made up of all document words in all the documents. The figure below illustrates this concept.

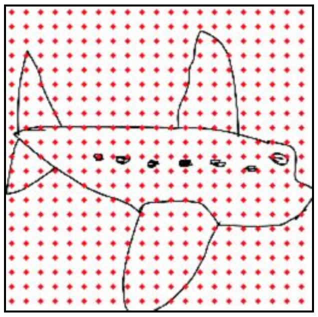


In this section I introduce how this technique can be applied to sketch classification, starting off by describing how we extract the equivalent of ‘words’ in images.

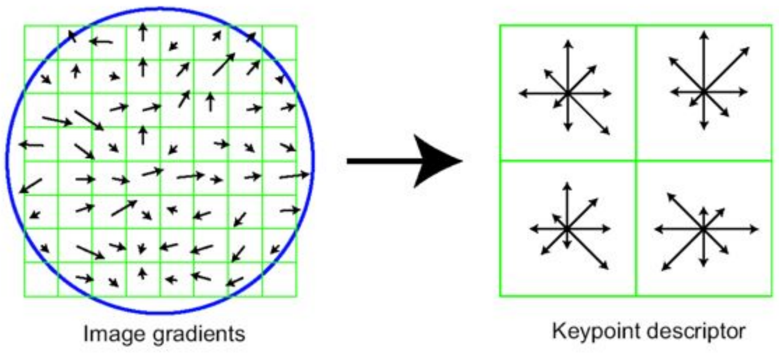
Visual Words

Visual words (or, as commonly referred to in Computer Vision, Descriptors) are encodings from a patch of an image used to describe some characteristic of that patch (and collectively the image) i.e. you could describe a patch of a image from its average RGB (Colour) channels or raw pixels.

Because sketches are generally sparse, in terms of detail, a dense grid of **keypoints** is distributed across the image, these keypoints act as the centre point for each of the patches, where each patch is 12.5% of the image size and overlaps each other 2.5 times. The final feature extractor uses **441** patches per image. The following figure shows the distribution of keypoints (red dots) for a given image.



An algorithm called SIFT (Scale-invariant feature transform) was then applied to create a descriptor (visual word) from each patch. The descriptor is built by using a 16x16 patch (centered around each keypoint), this patch is decomposed into 4x4 pixel tiles (x16 tiles for each patch) where for each tile is described as a gradient orientations and magnitude histogram (comprised of 8 orientation bins). The final result is a **128 bit feature vector** for each keypoint (16x8). The following figure illustrates the feature vector for a single patch.

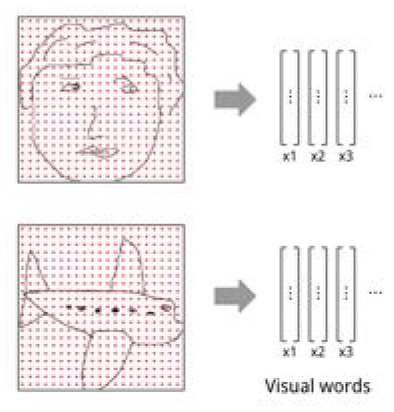


[1]

[1] Source: http://www.cs.toronto.edu/~kyros/courses/2503/Handouts/features.pdf

Further details about can be found it the original paper by David Lowe, http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf.

After this process, we have 441x128 feature vectors (or 441 visual words) for each image.



NB; Other feature descriptors were explored during development, including HOG (https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients) and BREIF (http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_brief/py_brief.html) but settled with SIFT as it performed significantly better than the rest. The details have been excluded from this report to put more emphasis on Machine Learning and evaluation rather than Computer Vision.

Building a Visual Codebook (Dictionary)

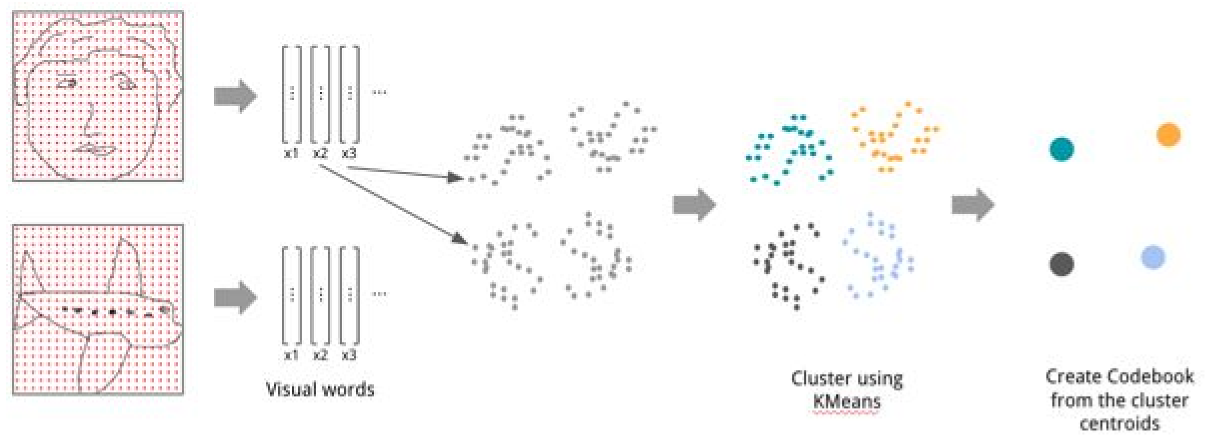
Our next task is to build a Codebook from the visual words extracted from all our training images. A Codebook is equivalent to a dictionary, and will be used to describe each of our images.

Our visual words can be compared with each by computing the distance between them i.e. Euclidean distance. We can use this property to find clusters within our data, that will then used as our Codebook.

To find clusters, we use Mini Batch KMeans (batch variant of KMeans, using a batch of 100 and using the 'kmeans++' method for initilising the centroids); the algorithm works by iteratively computing the distance between all Visual Words, updating the clusters centres (from the mean) and repeating until converged (no/small change in clusters).

KMeans algorithm:

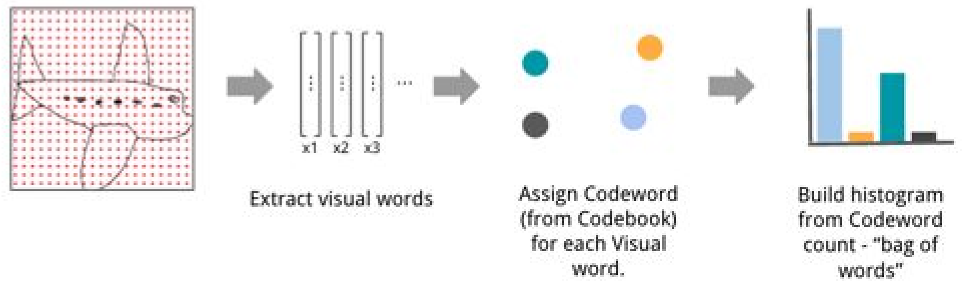
1. Compute a cluster centre for each cluster by computing the mean from the cluster members.
2. Re-assign each data point to the closest cluster.



Histogram of Visual Words

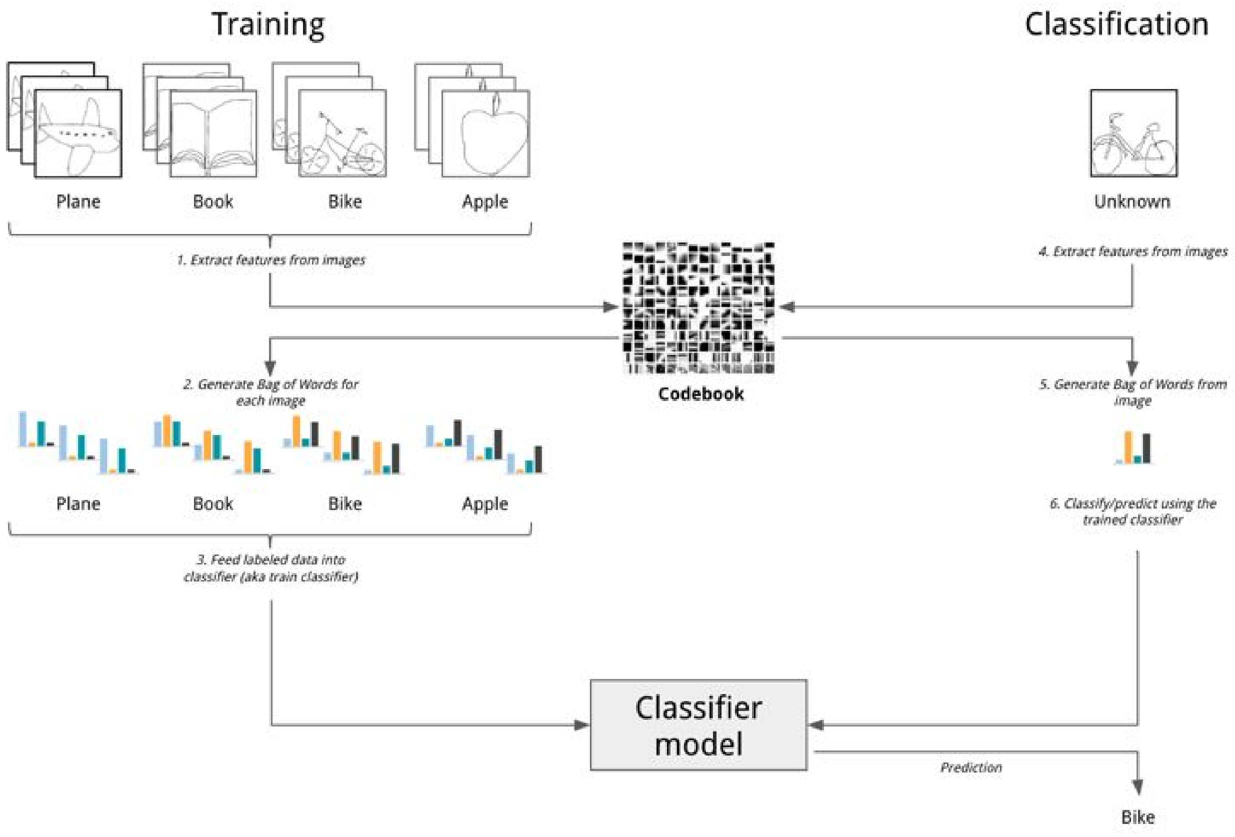
The Codebook gives us our dictionary of Codewords (cluster centres) which we now use to describe our images. We do this by first extracting the Visual Words from the image (as described above), every Visual Word is compared with the Codewords (from our Codebook) and assigned the nearest Codeword.

We do this for each Visual Word and build up a histogram, where the bins are the Codewords, which is then used as the 'signature' for the image (this process is known as **vector quantization**).


x1 x2 x3
Extract visual words
Assign Codeword (from Codebook) for each Visual word.
Build histogram from Codeword count - "bag of words"

Training

As described in the introduction, our goal is to accurately predict a category for a given sketch - a classical classification problem. Above we have described the process of describing images using 'bag of words', which is essentially a histogram of codewords from our codebook.
We can now use these features (bags of words) to train a classifier model to classify an image. The following figure illustrates this process (at a high-level).



Training
Classification
Plane
Book
Bike
Apple
Unknown
1. Extract features from images
4. Extract features from images
Codebook
2. Generate Bag of Words for each image
5. Generate Bag of Words from image
Plane
Book
Bike
Apple
6. Classify/predict using the trained classifier
3. Feed labeled data into classifier (aka train classifier)
Classifier model
Prediction
Bike

Feature engineering (exploration)

In this section I experiment by varying properties used to describe the image - the classification model[2] is held constant (along with the other properties) to observe the influence adjusting each property has on the performance.

Admittedly properties are interdependent but given the constraints of time, computing power and patience, this factor was ignored (to some extend).

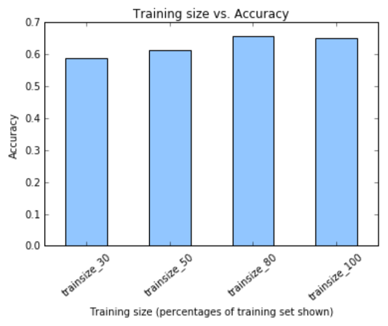
Properties

- Influence of training size i.e. is there a point where no more information is gained through samples.
- Number of clusters (aka visual words) i.e. how many visual words are required to effectively capture all the required information.
- Window resolution and Window Overlay both have to do with the density of the grid of keyboards i.e. how dense does this grid need to be to capture all necessary information.

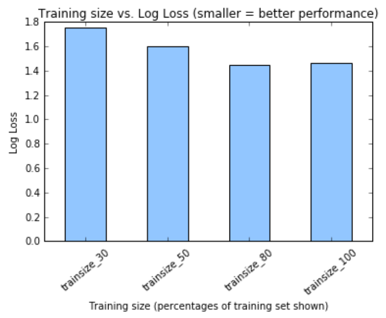
[2] Linear SVM; selected as it provided the better performance during initial training and evaluation on.

Training set size

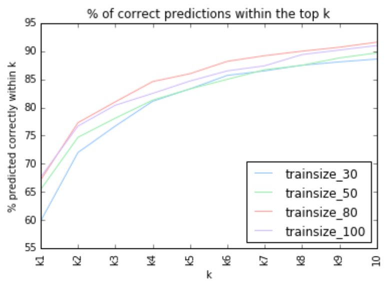
Exploring what influence the training size has on the performance of our model.



Accuracy



Log Loss

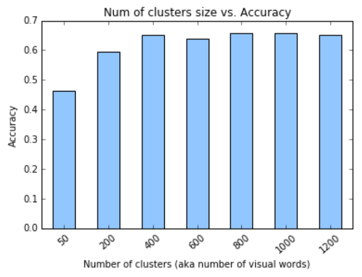


Accuracy for the top predictions

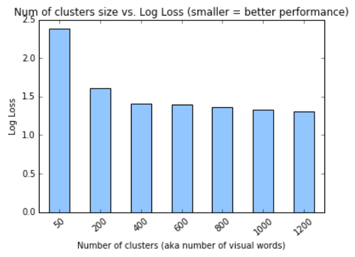
NB: all subsequent tests are performed on 50% of the training set to increase responsiveness whilst experimenting with the properties.

Number of clusters

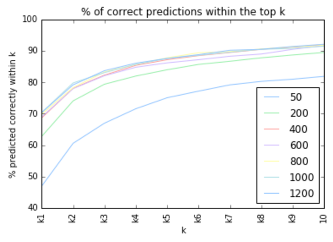
Clusters in this context is the number of visual words created for each image feature vector to be assigned to. Similar to compression, the lower the number of clusters the more detail we lose, but possibly gain in generalisation/pattern identification. In this section we vary the number of clusters and compare the performance.



Accuracy



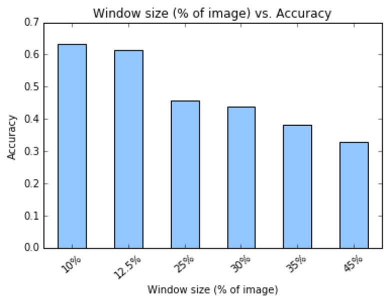
Log Loss



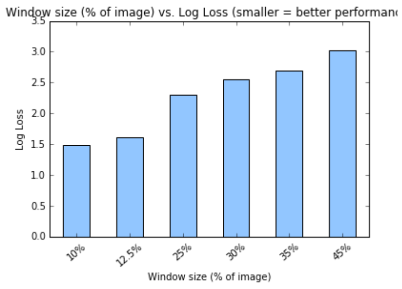
Accuracy for the top predictions

Window Resolution

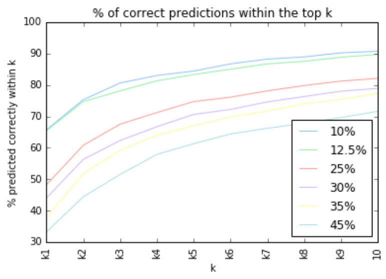
SIFT (and all considered feature extractors) can be confined to a window such to describe segments of the sketches. Here we vary the window size to determine what it's influence has on the performance of our model.



Accuracy



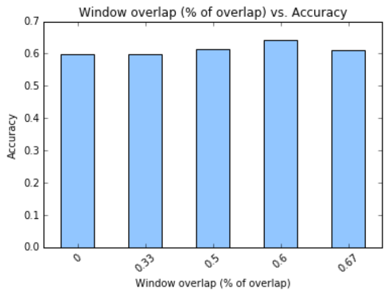
Log Loss



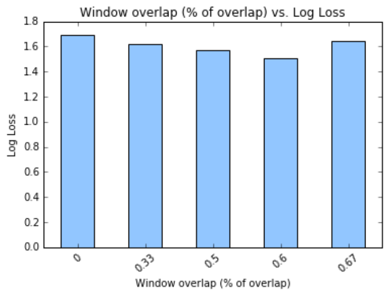
Accuracy for the top predictions

Window overlap

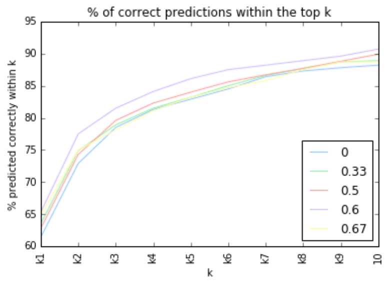
This parameter determines how much overlap there is for each window (or step size 1-1/N where N is the overlap) i.e. N=2 would mean there is a 1/2 overlap between all windows as the window is slide along.



Accuracy



Log Loss



Accuracy for the top predictions

Summary

In regards to **training size**, as expected, the more training data available increases the performance, up to a point - appears to plateau around 80% - this could mean the additional extracted features don't add any more information or that the other parameters of our training are limiting the model's ability to learn these additional features.

The number of **clusters** (visual words) beyond 400 resulted in marginal performance gains, for this reason and to keep things as simple as possible, a visual word dictionary of approx. 500 will be chosen.

Increasing **window size** reduced performance significantly, possibly loosing to much detail of the sketches. A window size of around 10-12.5% appears to give optimal results but obviously incurs additional computational cost.

Our last property we explored for extracting features was **window overlap** i.e. what stride was taken for each iteration. The result have shown that an overlap of approx. 60% provide optimal results, similar to **window size**, this adds to the computing overhead.

Based on the above results, the feature properties moving forwards will be:

- Use the full training dataset during training
- 450 visual words when creating the codebook
- Window size of 12.5%
- Overlap of 60%

Results

In section above the focus was on feature selection and tuning, in this section I compare various Machine Learning Algorithms and parameters in respect to the performance metrics described below and finally conclude with the performance of the most performant algorithm.

Performance of the *features* and *model* are based on (also described above):

Log Loss (Logarithmic Loss):
Log Loss quantifies the accuracy of a classifier by penalising false classifications i.e. minimising the Log Loss is basically equivalent to maximising the accuracy.
Log Loss heavily penalises classifiers that are confident about an incorrect classification e.g. if for a particular observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the Log Loss will be very large.
More details can be found: https://www.kaggle.com/wiki/LogarithmicLoss

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

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}(y_i\log(p_i) + (1-y_i)\log(1-p_i))$$

Accuracy score:
The fraction of correct predictions.

Accuracy of the top 10:
For each k, how accurate is the model i.e. when k is set to 2, then consider a correct prediction if the predicted probability includes the correct prediction within the top 2 predictions.

Models

The following is a list of algorithms and parameters used for sketch recognition.

Naive Bayes Classification

"Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features." - http://scikit-learn.org/stable/modules/naive_bayes.html

1. Multinomial Naive Bayes Classifier

"The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work." - http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB

2. Gaussian Naive Bayes Classifier

"GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian" - http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

K Nearest Neighbors Classification

"Classifier implementing the k-nearest neighbors vote." - http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

3. K Nearest Neighbors Classifier

A GridSearch was performed on the parameters (with ranges/values):

- algorithm = auto (attempt to decide the most appropriate algorithm based on the values passed during training)
- k nearest neighbors (n_neighbors) = [5, 10, 15, 20, 25]
- Distance metrics = ['minkowski', 'euclidean', 'manhattan']
- weights = ['uniform', 'distance'] (uniform; all points in each neighborhood are weighted equally, distance; weight points by the inverse of their distance)

Best performed parameters (based on the GridSearch):

- n_neighbors = 5
- Distance metric = 'manhattan'
- weights = 'distance'

Support Vector Machines (SVMs)

SVMs find the "maximum-margin" line that separates the classes (line "straight in the middle"). If the data cannot be linearly separable, the SVM will project the datums into higher dimensions (hyper planes). This can be done effectively by using kernals (known as the 'kernal trick') - http://scikit-learn.org/stable/modules/svm.html

4. SVM

http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

A GridSearch was performed on the parameters (with ranges/values):

- kernals = ['linear', 'rbf', 'poly']
- C_range = np.logspace(-2, 10, 13)
- gamma_range = np.logspace(-9, 3, 13)

Best performed parameters (based on the GridSearch):

- kernal = RBF
- C = 10.0
- gamma = 10.0

5. LinearSVM

http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

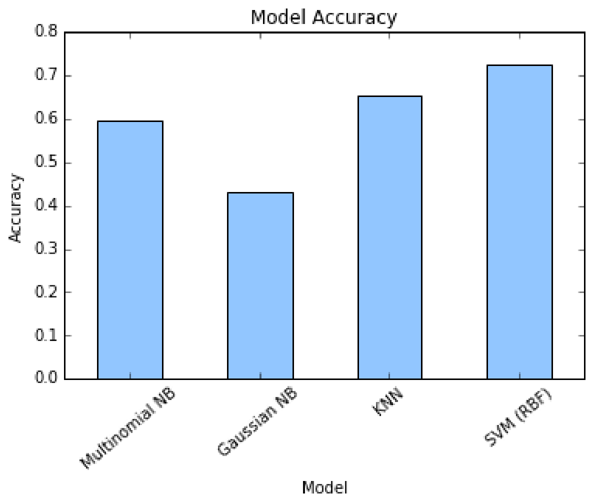
A GridSearch was performed on the parameters (with ranges/values):

- C_range = np.logspace(-2, 10, 13)
- loss_functions = ['squared_hinge', 'hinge'] (squared_hinge; square of the hinge loss, hinge; standard SVM loss)

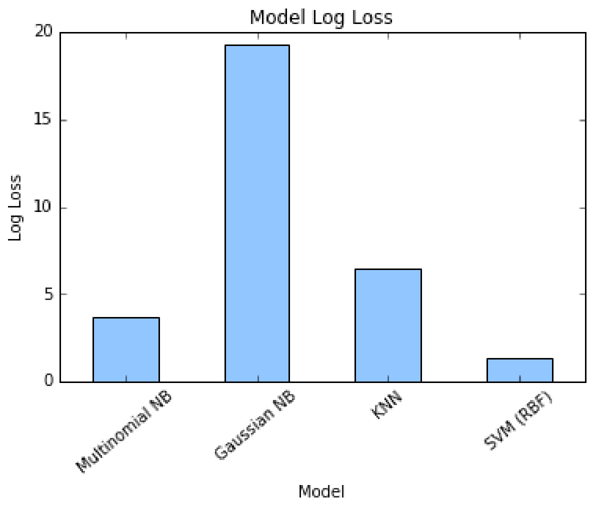
Best performed parameters (based on the GridSearch):

- C = 18
- loss = 'squared_hinge'

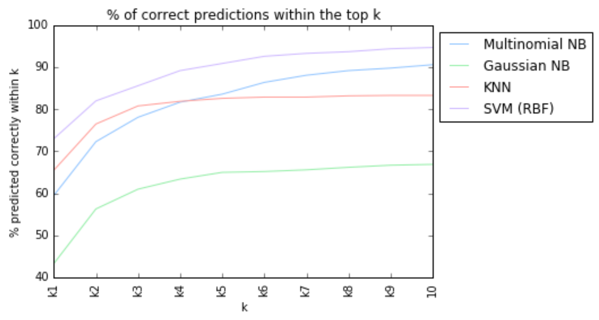
Model Evaluation



Accuracy



Log Loss



Accuracy for the top predictions

Model Selection

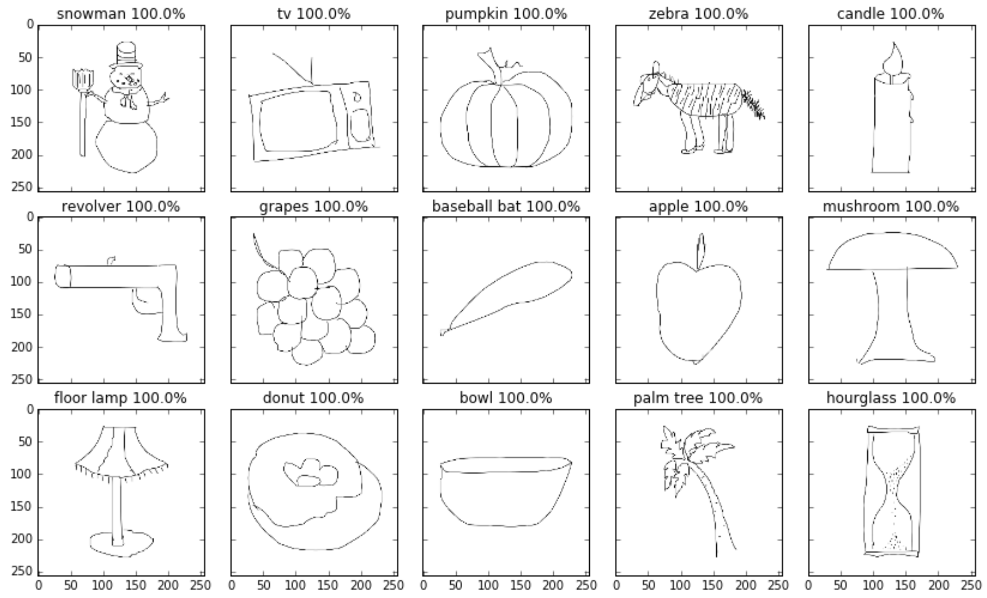
With respect to Accuracy, Log Loss, and 'K Accuracy'; SVM (using RBF) outperformed the other algorithms:

- Accuracy achieved was 0.73 (with an accuracy of 94.69 with k = 10)
- Log Loss achieved was 1.33

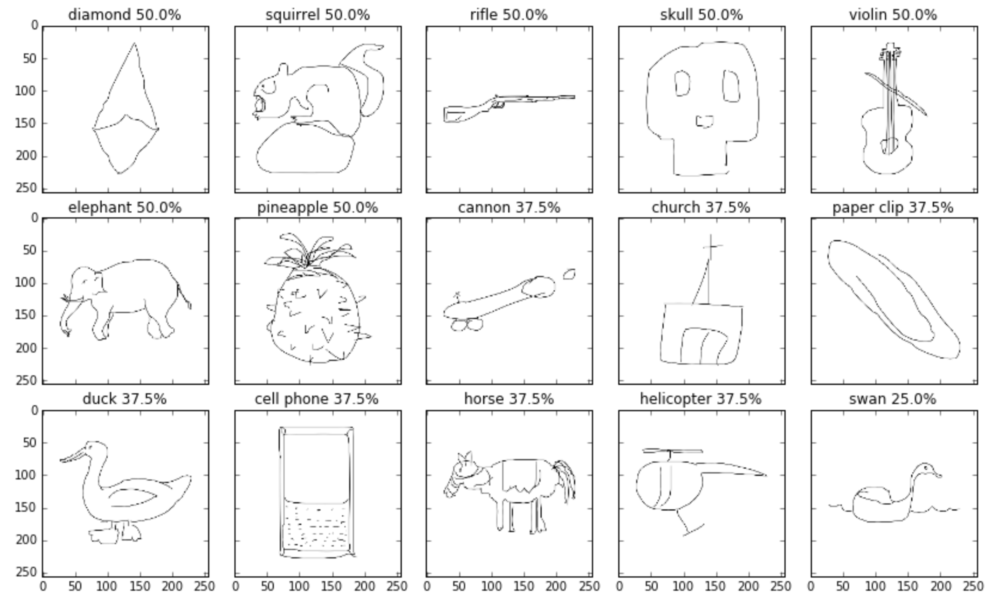
Visual investigation of the model

In this section we inspect the most accuracy and least accurate categories, and attempt to better understand the limitations of the approach.

15 most accurately classified sketches

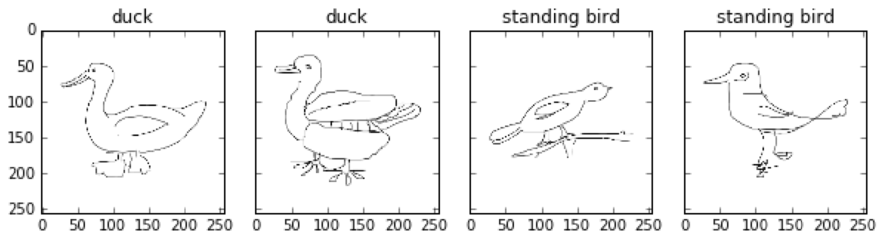


15 least accurate classified sketches

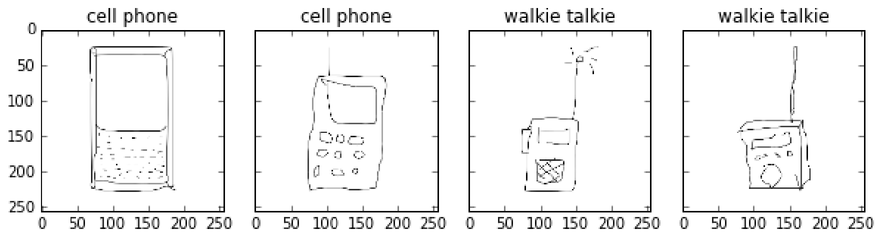


Most confused with

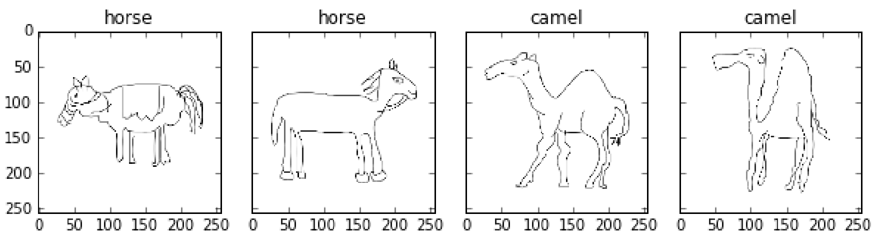
Each row of the grid below present a prediction, the first 2 columns display thumbnails of the true label and the last 2 columns display the incorrectly predicted category. Despite only seeing a small sample of the data you can an appreciation for its mistakes, especially for the categories duck, cell phone and camel.



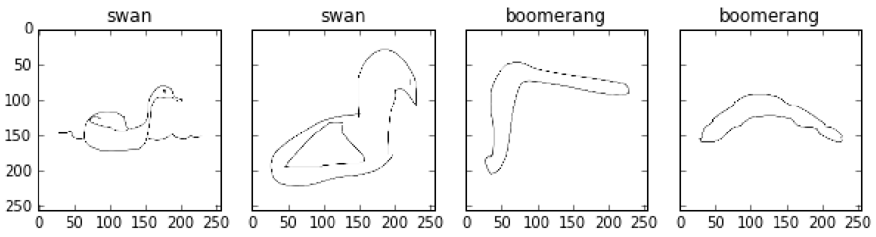
'duck' was most confused with 'standing bird' (by 25.0%)



'cell phone' was most confused with 'walkie talkie' (by 37.5%)



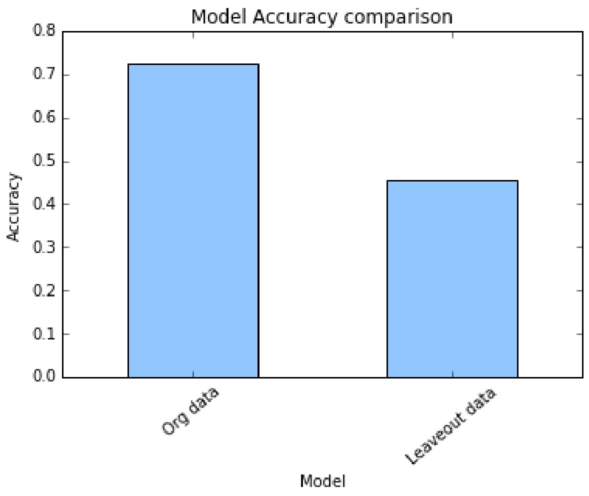
'horse' was most confused with 'camel' (by 25.0%)



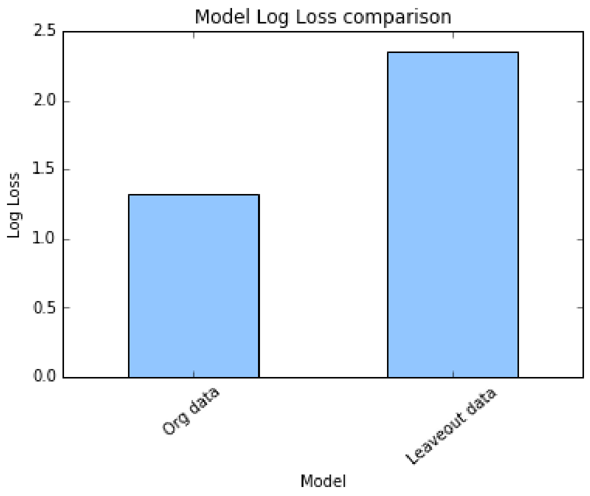
'swan' was most confused with 'boomerang' (by 12.5%)

Testing the robustness of the model parameters

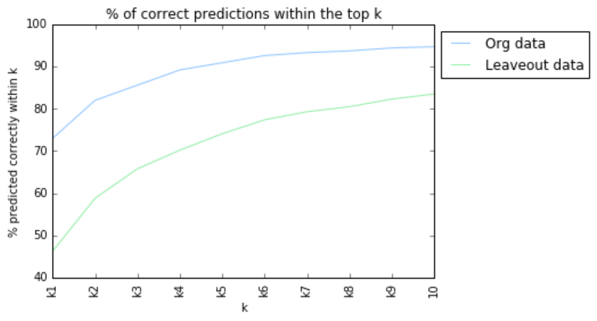
In this section we explore the robustness of the selected model parameters by evaluating the performance by training and testing against the data that was left out from the original set. Similar to above, we extract 113 categories - removing 8 images from each category for testing and training with the remaining 72 images from each category. Below are the results:



Accuracy

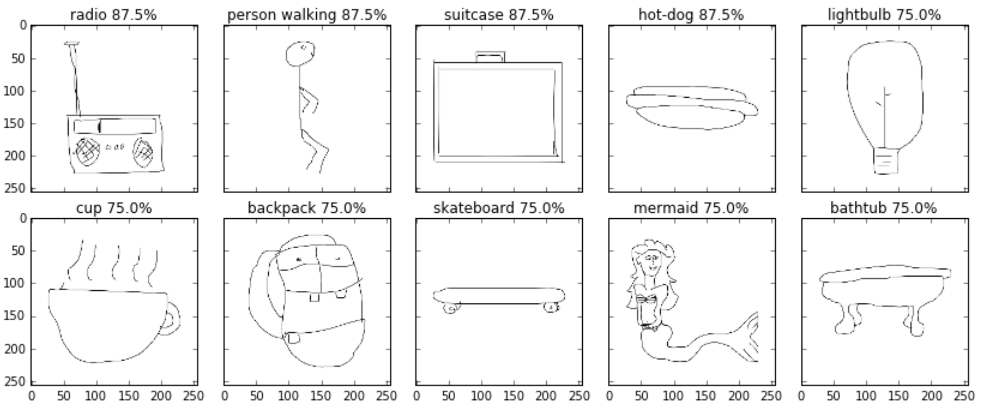


Log Loss

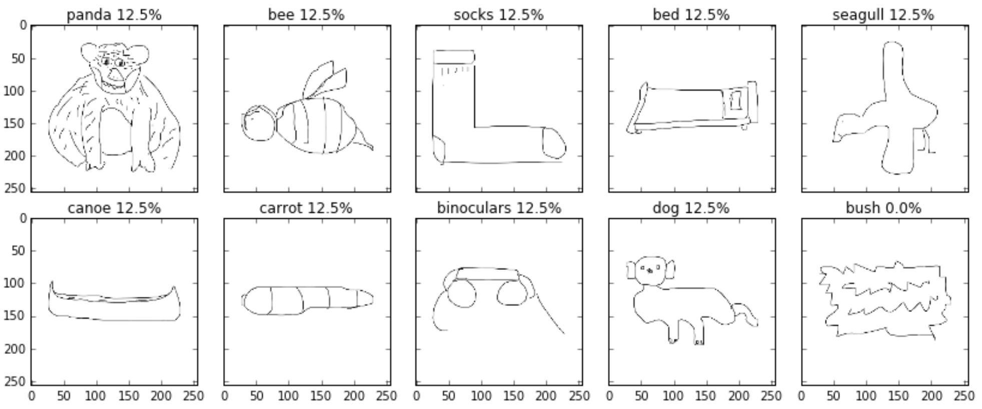


Accuracy for the top predictions

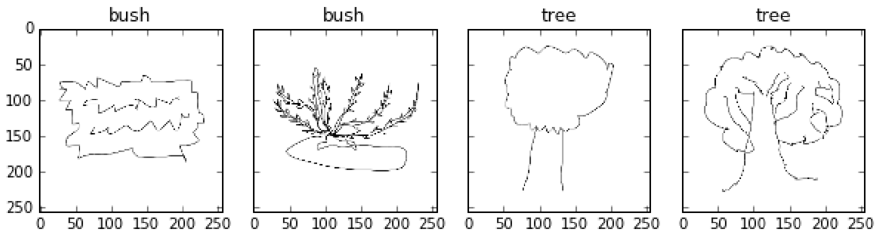
10 most accurately classified sketches



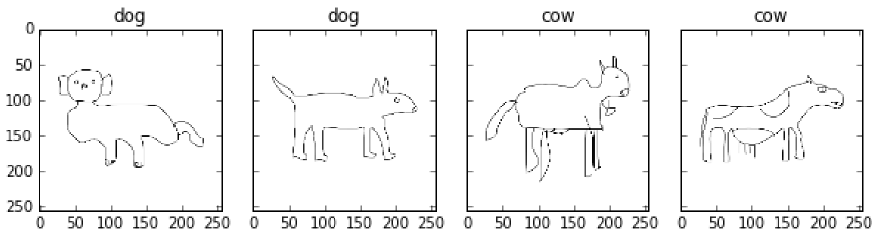
10 least accurate classified sketches



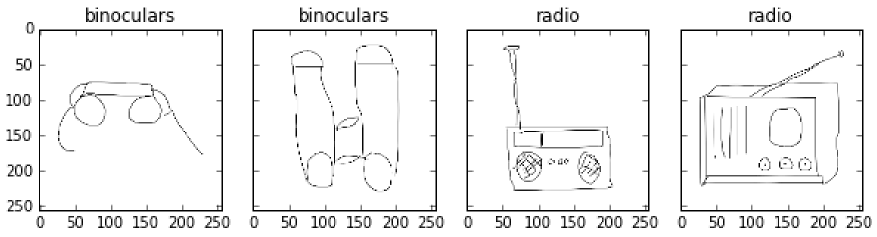
Most confused with



'bush' was most confused with 'tree' (by 37.5%)



'dog' was most confused with 'cow' (by 50.0%)

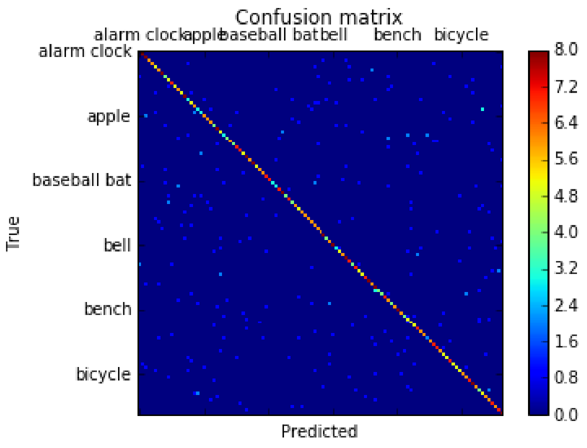


'binoculars' was most confused with 'radio' (by 25.0%)

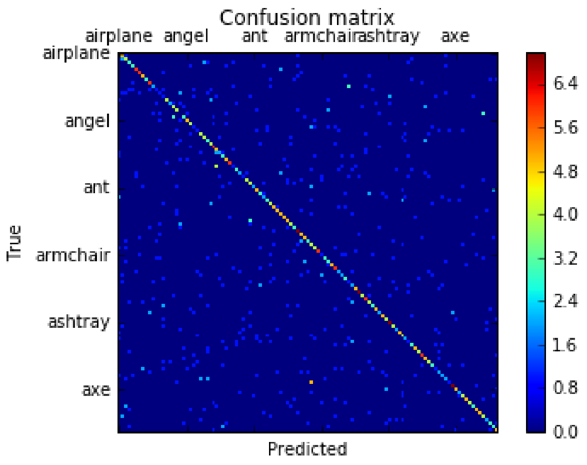
Results

The final solution was able to achieve an **accuracy of 73%** and **log loss of 1.33** on the dataset that was used in the process feature engineering and tuning, this fell to an **accuracy of 45%** and **log loss of 2.35** when trained and tested on the remaining sketches.

This difference in performance could be a result of the consequence of tuning features for a particular dataset and/or the result of outliers of *difficult* (to recognise) *sketches* in the second dataset. The high log loss and visual inspection of sketches (showing confident false positives) hint that this could been one of the influencers. This is further highlighted in the confusion matrices presented below, the confusion matrix from the original dataset has a clean diagonal line while the confusion matrix for the leave-out data shows a high degree of confident false positives, further suggesting the reduced performance may be caused by the representative similarities between categories.



Confusion matrix for from the original dataset



Confusion matrix for from the leave-out dataset

But further analysis is required to better understand what is affecting performance.

Whether this is sufficiently accurate enough or not is highly dependent on the application and user interface but does offer opportunity in assisting the user for input and guidance, some example applications include (with respective to the accuracy level achieved):

- Using sketch for input e.g. https://www.microsoft.com/en-us/research/project/mindfinder-finding-images-by-sketching/ (where-by n results are shown, allowing the system to correctly classify the input)
- Assisting in drawing e.g. http://vision.cs.utexas.edu/projects/shadowdraw/shadowdraw.html

The final parameters of the solution are:

- SIFT feature extractor, using a dense grid spaced 12.5% apart, and overlapping by 2.5%
- Codebook created using 400 visual words
- SVM with a RBF kernel for classification

Conclusion

The goal of the project was to accurately classify a user's sketch. This was achieved by first creating a codebook which would be used to describe each image, and collectively the category. Using this codebook, a image was described by building a histogram of the code labels (from the codebook) which was then used to classify the sketch.

Conceptually the project was intriguing and introduced me to a lot of new domains and challenges, but also interesting how solutions can be applied to different problems (bag of words). One big challenge was due to the abstract nature of 'features', this abstraction meant a lot of time was spent exploring different way of describing images. Consequently it also gave me appreciation of computational limitations i.e. having a more responsive workflow would allow for more exploration and end to end iterations in feature extraction and learning.

Future work

As mentioned above, further analysis is required to better understand what is affecting differences in performance between the different datasets, in addition, training and testing against the whole dataset is desirable.

Also, given the advancements and success in Neural Networks (Deep Learning), it would make for an interesting project to compare the performance of this with such model (and one I hope to do in the near future). Another interesting area to explore is how to improve the system with some component of reinforcement learning (or online learning)

Appendix

References

How Do Humans Sketch Objects?
http://cybertron.cg.tu-berlin.de/eitz/projects/classifysketch/

Free-hand sketch recognition by multi-kernel feature learning
https://www.eecs.qmul.ac.uk/~sgg/papers/LiEtAl_CVIU2015.pdf

Sketch Recognition by Ensemble Matching of Structured Features
https://www.eecs.qmul.ac.uk/~sgg/papers/LiEtAl_BMVC2013.pdf

Bag of visual words model: recognizing object categories
http://www.robots.ox.ac.uk/~az/icvss08_az_bow.pdf

Histogram of Oriented Gradients and Object Detection
http://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/

Bag-of-words model in computer vision
https://en.wikipedia.org/wiki/Bag-of-words_model_in_computer_vision#Codebook_generation

HOGgles: Visualizing Object Detection Features
http://web.mit.edu/vondrick/ihog/

Evaluation metrics
http://scikit-learn.org/stable/modules/model_evaluation.html

Visual Codebook Tae Kyun Kim Sidney Sussex College
http://mi.eng.cam.ac.uk/~cipolla/lectures/PartIB/old/IB-visualcodebook.pdf

Scale Invariant Feature Transform (SIFT) - http://www.vlfeat.org
http://www.vlfeat.org/api/sift.html

Feature Descriptors, Detection and Matching
http://www.cs.toronto.edu/~kyros/courses/2503/Handouts/features.pdf

Distinctive Image Features from Scale-Invariant Keypoints
http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

Libraries

http://opencv.org

http://scikit-learn.org