# CMP-6035B - Image categorisation I

Eden Attenborough 100301654
Cameron Bateman xxxxxxxxx

April 14, 2022

# 1 Literature Review

## 1.1 Feature Detection & Description

Feature detection in computer vision is the process of extracting pieces of information from within an image, typically whether a certain region of the image contains unique properties. These properties can vary but common ones are corners, edges or curves and are commonly referred to as local features.

Feature description in computer vision is a method to describe local properties of an image at pixel coordinates marked via a feature detection algorithm. These methods extract valuable information from the image data and are then commonly stored in a vector, termed a feature vector.

Features in image processing fall into two categories; global and local features. These two categories vary in their application, with global features typically being used in image retrieval (CBIR [**?** ]), object detection and classification, while local features are commonly used for recognition and identification.

- Global features are those which describe the image as a whole, such as gradient histograms, colour histograms, contour descriptors and image texture, these are typically used image retrieval, object detection and classification. These features are effective as they produce compact descriptions of images in which each image corresponds to a location in a high dimensional feature space. But global features are sensitive to occlusion and clutter. This results in prior assumptions to image data sets when working in the global space, in that the image contains only a single object or that the object can be easily segmented from the background. [**?** ]

- Local features are those extracted from analysing sub regions of an image, typically a singular or grouping of pixels and its relation to its surrounding pixels. This relation can be something simple such as calculating changes in values between the pixels, or more sophisticated such as a histogram of gradients. Sub-regions are commonly referred to as interest points or keypoints and their relation referred to as a descriptor.

#### 1.1.1 HOG

#### 1.1.2 Keypoints

#### 1.1.3 SIFT

#### 1.1.4 BRIEF / ORB

#### 1.1.5 CNNs

## 1.2 Classifiers

### 1.2.1 The perceptron and Linear Classifiers

The perceptron [1] is one of the most simple and oldest machine learning algorithms. It is provably able to separate linearly separable problems in a finite number of steps [2], but it may fail to converge in problems that aren't linearly separable. It was originally made to model the behaviour of a brain neuron.

As shown in Figure 1, linear classifiers take the form

$$f(\mathbf{x}) = \mathbf{w} * \mathbf{x} + b \tag{1}$$

Where $\mathbf{w} \times \mathbf{x}$ is the dot product $\sum_{i=1}^{m} w_i x_i$ where $m$ is the number of inputs and $b$ is the bias, and $w$ is a weight for each input $x$. Then it can be used to make binary classifications:

$$f(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{w} * \mathbf{x} + b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Given a set of input attributes $x_1, x_2, ..., x_m$, response variable $t$ which can be 1 or -1,

$$y = \psi(w_0 + w_1 x_1 + ... + w_m x_m) \tag{3}$$

and

$$\psi(z) = \begin{cases} +1, & \text{if } z \geq 0 \\ -1, & \text{if } \ < 0 \end{cases} \tag{4}$$

a dataset $\mathbf{X}$ and a *learning rate* $\eta$, a perceptron training algorithm can be defined formally in Algorithm 1 as:

**Algorithm 1** Trainperceptron(Dataset $X$, Learning Rate $\eta$)

```
Set w to random weights
while StoppingFunction(t, y) == False do
    for i = 0 to n do
        y_i = ψ(w, x_i)
        for j = 1 to m do
            Δw_j ← (1/2)η(t_i − y_i)x_ij
            w_j ← w_j + Δw_j
```

Informally, it works by iterating through training data, making a new classification equation each time. If it makes a misclassification, update the equation by a factor of the value. The learning rate $\eta$ defines by how much the equation is changed with each iteration.

An important part to consider is the stopping condition function. We need to make sure that it stops eventually,

even if an exact separation cannot be achieved. Simple stopping conditions could be to stop when there is no change in $y$ or when a linear separation has been accomplished.

perceptrons are not useful in the context of image classification, it is not advanced enough for that. However, it provides us with a useful place to start understanding classifiers, and they are crucially important for CNNs, which are extremely relevant to image classification.

### 1.2.2 kNNs

Nearest neighbour classifiers are actually one of the oldest machine learning algorithms, first proposed in 1951 [3]. It is one of the easiest algorithms to understand: it considers the *distance* between training points and test points.

1-NN selects the single nearest point. kNN selects the $k$ nearest points and uses some voting algorithm to select a point (this is discussed later) this can allow a measure of confidence to be generated.

**Algorithm 2** knnclassify($x$, $(z, c) \in Z$, $k$, $D(p, q)$)

**Require:** Test point $x$, training points $z$ and label $c$ in $Z$, $k$ nearest neighbours, a distance function $D$

1: **for** each train data point $x \in Z$ **do**
2: Compute the distance $D(x, z)$ between the train data point and the test data point
3: Select $Z(k, c) \subseteq Z$, the set of the $k$ nearest neighbours
4: The label $c$ of test $x$ can be given by $c(x) = \max_{v \in \{1,\ldots,C\}} \sum_{(z,c) \in Z(k,x)} \delta(v, c)$

There are a number of different distance metrics that can be used some of the most popular are as follows: euclidean distance:

$$D(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{5}$$

Where $x$ and $y$ are the data points for which we are trying to calculate the distance between. Chebyshev distance:

$$D(x, y) = \max_i(|x_i - y_i|) \tag{6}$$

Minkowsky distance:

$$D(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^r)^{r^{-1}} \tag{7}$$

My implementation uses the default matlab value of $r$ of 2. There are many more options, matlab has 12 functions available. The above are the three my implementation was tested on.

What is the best value of $k$ to use? It makes sense to use an odd number to simplify voting. Random tie breaking with $2k$NN has been shown to give the same results as $(2k-1)$NN [4]. The value of $k$ defines the *generalisation*: too small leads to over-fitting and too large leads to under-fitting. It is common to use a validation set to test on to choose the best value of $k$ for a given problem. This is what we will do.

The simplest way of selecting a class from $k$ neighbours is a simple majority vote. A full algorithm pseudo-code implementation is shown in Algorithm 2 However, this can lead to bad results since therefore the distance of a given point is not considered directly. [5] proposes giving a weight to votes depending on their distance, since the nearest the neighbour, the more relevant that neighbour is. The two weightings he proposes are:

$$w_x = \frac{1}{D(x, y)} \tag{8}$$

Where $w_x$ is a corresponding weight for $x$ and $D(x, y)$ is a distance function; and

$$w_x = k - j + 1 \tag{9}$$

Where $j$ is the *rank* of the distance metric compared to others.

By default when looking for nearest neighbours, all points are used. This can lead to poor performance if there is a large amount of training data. A number of optimisation techniques have been proposed. For example by precomputing reference distances to rule out far away points straight away [6] or by keeping track of similar distances using a sum of squares metric [7]. In addition to these heuristic methods, you can also remove points for optimisation. However in the context of image classification this is not likely to be easy due to the high dimensionality. Too much dimensionality is in fact a common issue with using kNNs for image classification, due to the fact that the classifier is sensitive to irrelevant features. [8] proposes a solution to this problem by implementing kNN classifiers in conjunction with a CNN. They used it for image classification with a high degree of accuracy.

[9] uses a kNN classifier with a set of local interest features generated by SIFT [10] and SURF [11]. They did full image classification with a weighted kNN classifier, and achieved an accuracy of around 0.9 for their test set.

[12] showed that quantization and informative feature selection lead to bad accuracy with kNNs due to the information loss, proposes a direct 'Image-to-Class' distance, without descriptor quantization.

### 1.2.3 Support Vector Machines

To understand support vector machines (SVMs) [13] we will first consider linear classifiers.

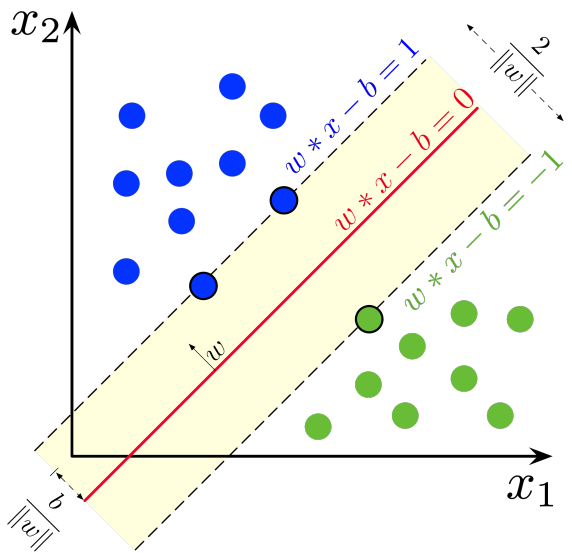


Figure 1: Example graph showing a linear SVM classifier. Image licensed under the Creative Commons Attribution-Share Alike 4.0 International [14]

Figure 1 shows a linear SVM classifier. Linearly separable problems can be separated using the function:

$$f(\mathbf{x}) = \mathbf{w} * \mathbf{x} + b \tag{10}$$

Assuming a two-class problem, adding more classes simply adds more dimensions. However, there will be many possible lines that fulfil this separation. We must select the best line.

$\gamma_i$ is the distance between a point $x_i$ and the hyperplane $\mathbf{w} * \mathbf{x}_i + b = 0$:

$$\gamma_i = \frac{t_i(\mathbf{w} * \mathbf{x}_i + b)}{||\mathbf{w}||} \tag{11}$$

$\gamma$ is the distance between the hyperplane and the *nearest point* in each class. Let:

$$t_i(\mathbf{w} * \mathbf{x}_i + b) \geq 1$$

Yields:

$$\gamma = \min_i \gamma_i = \frac{1}{||\mathbf{w}||} \tag{12}$$

The data points that define the hyperplane, the ones that lie on the margins, are called the *support vectors*. These points have black edges in Figure 1. Thus our *optimisation problem*, to find the maximum $\gamma$, is defined as:

$$\max_{\mathbf{w},b} \gamma = \max_{\mathbf{w},b} \frac{1}{||\mathbf{w}||} \Rightarrow \min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w} \tag{13}$$

Subject to:

$$t_i(\mathbf{w} * \mathbf{x}_i + b) \geq 1, \forall i$$

There are a number of solving the optimisation problem. A Bayesian method would be to simply take the average of all possible functions. Another, more formal, way of doing this is to use the method of *Lagrange multipliers*. Our Lagrangian function is:

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} = \sum_{i=1}^{n} \alpha_i(t_i(\mathbf{w} * \mathbf{x}_i + b) - 1) \tag{14}$$

Where $\alpha_i \geq 0 (\forall i \in 1, 2, ..., n)$ are the Lagrange multipliers. Now we can use the differential to find the optimization:

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i t_i \mathbf{x}_i = 0 \tag{15}$$

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \alpha)}{\partial b} = -\sum_{i=1}^{n} \alpha_i t_i = 0 \tag{16}$$

Substituting gives us the *dual optimization problem*- maximize:

$$\mathcal{L}(\alpha) = \sum_{i=i}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=i}^{n} \alpha_i\alpha_j t_i t_j (\mathbf{x}_i * \mathbf{x}_j) \tag{17}$$

Subject to $\alpha_i \geq 0 \forall i$ and $\sum_{i=1}^{n} \alpha_i t_i = 0$

This is a quadratic function, and is therefore easily solvable.

Very few problems are completely separable, however. We can use a variable $\zeta$ to allow for misclassification errors this turns the optimization problem into:

$$\min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \zeta_i \tag{18}$$

Subject to:

$$t_i(\mathbf{w} * \mathbf{x}_i + b) \geq 1 - \zeta_i, \forall i \in 1, 2, 3, ..., n$$

And:

$$\zeta_i \geq 0, \forall i \in 1, 2, 3, ..., n$$

This is called a 'soft margin', since $\zeta$ allows points past the margin. $C$ is a regularisation argument- a small $C$ allows more points past the margin. A small $C$ reduces the number of points past the margin.

The above method is the traditional method for solving support vector classifiers. There are more modern algorithms now available. For example sub-gradient descent algorithms [15], [16] have also been used in the context of SVMs, and have shown to be more efficient [17]. A coordinate descent approach has been used too [18].

In the context of image classification, the original SVM paper authors used their SVM for handwriting classifiers [19]. This paper is especially useful for comparing SVMs against other classifiers, for example kNNs and CNNs, despite the simple training dataset.

#### 1.2.4 CNNs

Convolutional Neural Networks (CNNs) are now the main classification system used for image classification. A the core of the CNN there is the perceptron, which has already been discussed here. [20] is a famous book that stalled the use and development of perceptrons for many years; but what came next is using multiple layers of perceptrons, through which data travels and is transformed with each level. To train a CNN we must have some measure of the accuracy of the fit, for example by using the sum of squares error:

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}(y_k(\mathbf{x}^n;\mathbf{w}) - t_k^n)^2 \tag{19}$$

Where $y_k$ is the $k^{th}$ output and $t_k^n$ is the target output for the $n^{th}$ training data of the $\mathbf{x}^n$ corrisponding input data.

We iteratively update the value of $\mathbf{w}$ which minimizes the error $E$ with the *gradient descent*- the derivative of $E$. This is called the *back-propagation algorithm.*

In the context of computer vision, CNNs can be used to form position independent filters to extract features.

# 2 Evaluation

## 2.1 Tiny Images

Our implementation of tinyimages had three tunable parameters: the number of vertical pixels to crop to, the colour space to use, and the cropping algorithm. The colour space options were `rgb` and `grayscale`. The cropping algorithms were `crop` and `distort`. `crop` cuts out the center of the image so that it is always a square. `distort` distorts aspect ratio to a square.

Evaluation was done using the same training and test datasets, and on the same classifier (the default one) for fairness:

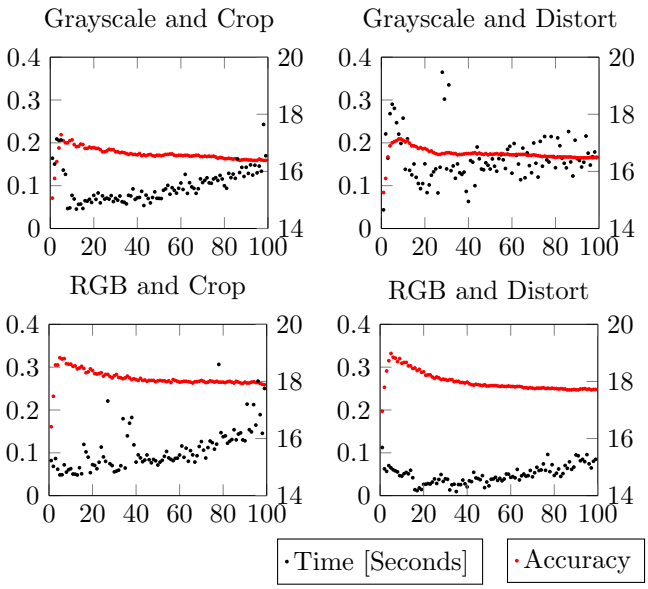


Figure 2: Graphs showing classifier accuracy (left axis) and time taken (right axis) for resizing to a given number of pixels (tinyimages), in different colour spaces, and using different cropping algorithms

#### 2.1.1 Resizing Images

Figure 2 shows that distorting the images lead to better accuracy than cropping them. A hypothesis for this could be that less pixel information is lost this way.

#### 2.1.2 Colours

Figure 2 also shows that using an RGB colour space leads to significantly better accuracy, with no speed penalty. Indeed, the *most accurate* parameters were also the *fastest*!

#### 2.1.3 Image Sizes

These tests show that adding more pixels does not mean more accuracy. For all tests, the best accuracy is from 4-16 vertical pixels. Accuracy tapers off after that.

The best accuracy was achieved resizing to five vertical pixels, distorting the image, using the RGB colour space, with an accuracy using the default classifier of 0.332.

## 2.2 Colour Histograms

Our implementation of colour histograms had two tunable parameters. The number of bins for the histogram to use and the colour space to use. Tests and their conclusions are shown below:

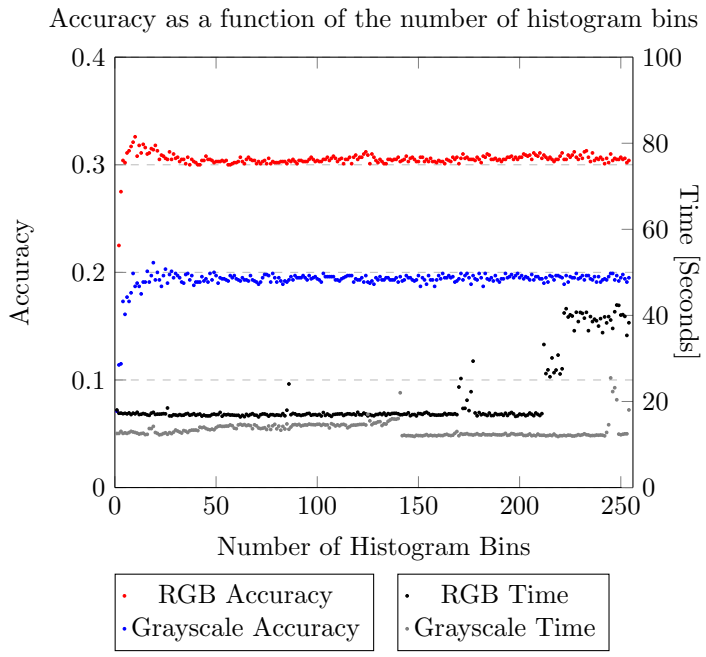


Figure 3: Graph showing resulting classifier accuracy, and the time taken, from the number of histogram bins and colour space used

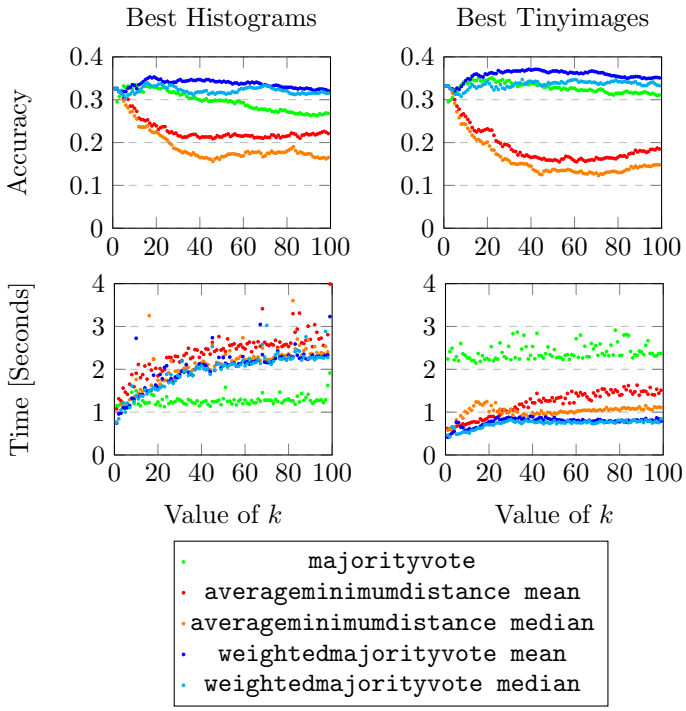


Figure 4: Graphs showing resulting accuracy, and the time taken, from the value of $k$ with different averaging functions, using the **euclidean** distance function

#### 2.2.1 Quantisation

Image quantisation defines into how many bins there are in the histogram there are. My testing showed that the ideal number of bins was 10, which gave me an accuracy of 0.326, using the default classifier. There was no noticeable time penalty for using more bins.

#### 2.2.2 Colour Spaces

We can either convert to grayscale and make just one histogram, or we can make three histograms, one for each colour channel, and them vectorize them. My testing showed that doing the latter led to much better accuracy, which a negligible time increase.

### 2.3 kNN classifier

#### 2.3.1 K value

Our implementation of a kNN classifier has four tunable parameters:

- The value of $k$ (the number of nearest points to consider)
- The distance function to use
- The voting method (how a result is produced from $k$ different results)
- The average function to use (two voting methods require averaging)

#### 2.3.2 Value of $k$

The value of $K$ selects from how many nearest neighbours are considered. For the majority vote and weighted majority vote, the peak accuracy was a $k$ of around 20. The time taken seemed to increase proportionally, but not linear-ally, as $k$ increases. Here the relative increase is important, since tests were done on a number of different computers.

#### 2.3.3 Voting Algorithm

When we consider $k$ nearest neighbours, we must decide from which of them we make a prediction. The simplest possible way to do this is just a majority vote, which is implemented in our code using the `majorityvote` parameter. [5] describes how this is not ideal, since it does not consider the individual distances of the $k$ nearest neighbours. It suggests two improvements: weighing the votes by a factor of the distance, and weighing the votes by the rank of the distance. We only implemented the former of these, using the `weightedmajorityvote` argument. We want the *smallest* distance to have the *largest* multiplier- [5] achieves this by multiplying by the inverse of the distance:

$$w_x = \frac{1}{D(x, y)}$$

We achieve this by subtracting each distance by the largest distance and taking the magnitude:

$$\mathbf{d}_2 = |\mathbf{d}_1 - \max(\mathbf{d}_1)|$$

The final algorithm selects the label with the lowest average distance from the point.

The latter two algorithms need a definition for 'average' so we here have two additional options: to use the median or the mean.

Figure 4 shows that for all values of $k$, using a weighted majority vote was slightly more accurate than using a simple majority vote. Using the average minimum distance, the accuracy decreased as $k$ increased. This leads me to believe that it is not a good algorithm.

For all tests, using the mean was more accurate than using the median. Generally for statistics we prefer to use the median since it is less distorted by extreme results, but in this context this is something we want.

These conclusions were the same for both colour histograms and tiny images, this is also shown in Figure 4.

### 2.3.4 Distance Measurement Algorithm

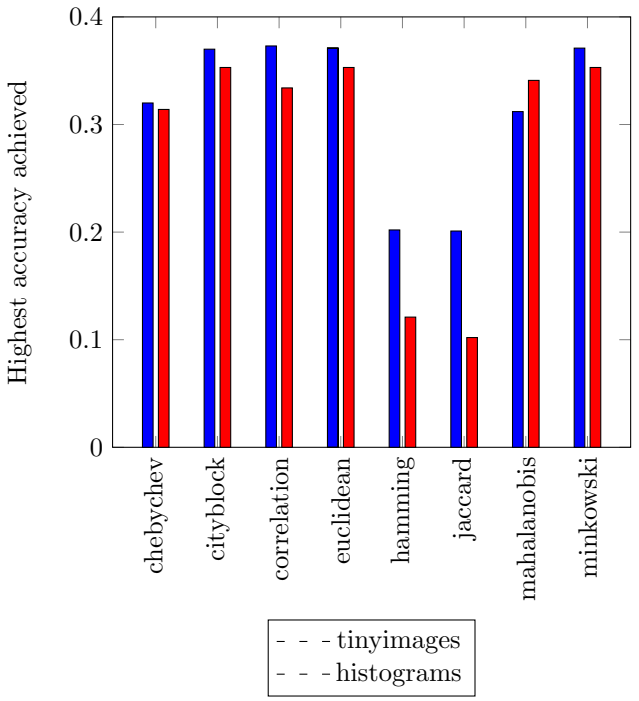


Figure 5: Chart showing maximum accuracies achieved with different distance metrics

There are a number of different metrics we can use to define distance between points. Figure 5 shows the different performances of these algorithms. For histograms, `cityblock`, `euclidean` and `minkowski` are tied at having the best preformance, but `euclidean` and `minkowski` are considered superior since they achieved this with a smaller value of $k$: (20 vs. 17). For tinyimages, the `correlation` metric was the best, with a value of 37 for $k$.

## 2.4 Other metrics

So far we have only considered the accuracy as a metric of performance. This can potentially be dangerous. A confusion matrix can be used to show us more detail, the true positives, false positives, false negatives, and true negatives:

| | | Actual | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted | Positive | True Positive | False Positive |
| | Negative | False Negative | True Negative |

From this data we can calculate additional metrics.

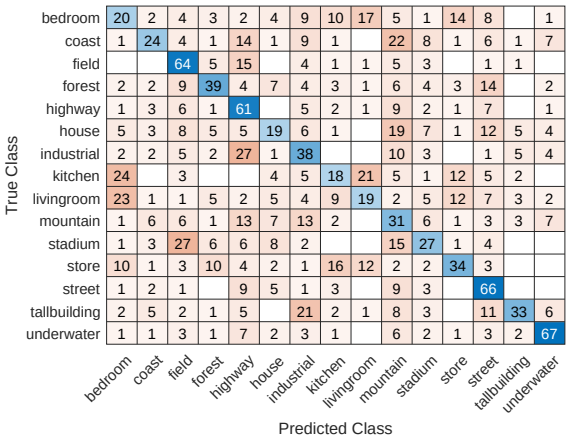


Figure 6: Confusion Matrix, generated from best **tinyimages** parameters.

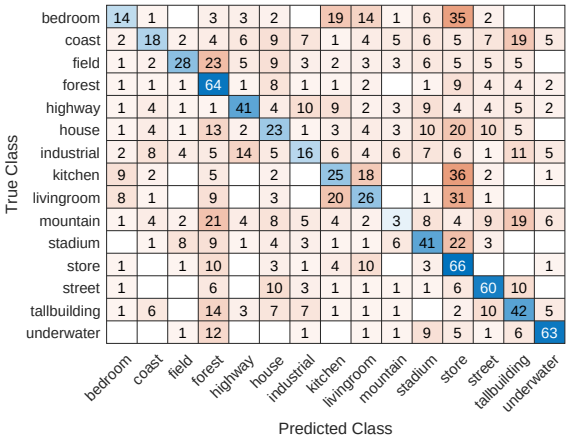


Figure 7: Confusion Matrix, generated from best **histograms** parameters.

Since our classifier can predict multiple outcomes besides true and false, we add columns to show incorrect predictions. Figure 6 shows the confusion matrix for tinyimages. The diagonal line is correct predictions. The *true positive rate*, also called the *sensitivity* or *recall*, is the proportion of predictions correctly classified; it is given by:

$$\text{True Positive Rate} = \frac{\text{True Positives}}{\text{True Positives} \times \text{False Negatives}}$$

Conversely, the *true negative rate*, also called the *specificity* is the proportion of negative cases that were correctly classified. It is given by:

$$\text{True Negative Rate} = \frac{\text{True Negatives}}{\text{False Positives} \times \text{True Negatives}}$$

We also have the *false positive rate*, also called the *type I error*:

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} \times \text{True Negatives}}$$

and the *false negative rate*, also called the *type II error*:

$$\text{False Negative Rate} = \frac{\text{False Negatives}}{\text{True Positives} \times \text{False Negatives}}$$

Plotting the False Positive Rate against the True Positive Rate gives us the *Receiver Operator Curve* (ROC). The area under the ROC can be an important metric.

The *balanced accuracy* is an important metric if many more predictions should be from one class than another; e.g. medical testing. It is given by:

$$\text{Balanced Accuracy} = \frac{\text{True Positive Rate} + \text{True Negative Rate}}{2}$$

Table 1 shows metrics for different classifier labels and their averages.

## 2.5 Conclusion

In conclusion, we have created an image classification system using non-CNN methods and have ran experiments to find the best parameters for this setup. However, the maximum accuracy we have achieved is 0.4. All modern research in the area of image classification uses CNNs so we propose that using a CNN, in conjuncton with more advanced feature detection algorithms e.g. SIFT could achieve a much higher accuracy.

| | sensitivity | | specificity | | precision | | F | | Balanced Accuracy | |
|---|---|---|---|---|---|---|---|---|---|---|
| | tinyimages | histograms | tinyimages | histograms | tinyimages | histograms | tinyimages | histograms | tinyimages | histograms |
| bedroom | 0.18 | 0.25 | 0.9636 | 0.9486 | 0.2609 | 0.2577 | 0.213 | 0.2538 | 0.5718 | 0.5993 |
| coast | 0.34 | 0.66 | 0.9664 | 0.8643 | 0.4198 | 0.2578 | 0.3757 | 0.3708 | 0.6532 | 0.76215 |
| field | 0.2 | 0.14 | 0.9471 | 0.9793 | 0.2128 | 0.3256 | 0.2062 | 0.1958 | 0.57355 | 0.55965 |
| forest | 0.19 | 0.26 | 0.9614 | 0.9521 | 0.2603 | 0.2796 | 0.2197 | 0.2694 | 0.5757 | 0.60605 |
| highway | 0.19 | 0.23 | 0.9671 | 0.9471 | 0.2923 | 0.2371 | 0.2303 | 0.2335 | 0.57855 | 0.58855 |
| house | 0.38 | 0.16 | 0.9379 | 0.97 | 0.304 | 0.2759 | 0.3378 | 0.2025 | 0.65895 | 0.565 |
| industrial | 0.27 | 0.41 | 0.9643 | 0.9521 | 0.3506 | 0.3796 | 0.3051 | 0.3942 | 0.61715 | 0.68105 |
| kitchen | 0.67 | 0.63 | 0.9757 | 0.9807 | 0.6634 | 0.7 | 0.6667 | 0.6632 | 0.82285 | 0.80535 |
| livingroom | 0.33 | 0.42 | 0.9843 | 0.94 | 0.6 | 0.3333 | 0.4258 | 0.3717 | 0.65715 | 0.68 |
| mountain | 0.66 | 0.6 | 0.9393 | 0.9579 | 0.4371 | 0.5042 | 0.5259 | 0.5479 | 0.79965 | 0.77895 |
| stadium | 0.61 | 0.41 | 0.9193 | 0.9721 | 0.3506 | 0.5125 | 0.4453 | 0.4556 | 0.76465 | 0.69105 |
| store | 0.64 | 0.28 | 0.9414 | 0.985 | 0.4384 | 0.5714 | 0.5203 | 0.3758 | 0.7907 | 0.6325 |
| street | 0.24 | 0.18 | 0.9779 | 0.9757 | 0.4364 | 0.3462 | 0.3097 | 0.2368 | 0.60895 | 0.57785 |
| tallbuilding | 0.31 | 0.03 | 0.9121 | 0.9786 | 0.2013 | 0.0909 | 0.2441 | 0.0451 | 0.61105 | 0.5043 |
| underwater | 0.39 | 0.64 | 0.9707 | 0.9036 | 0.4875 | 0.3216 | 0.4333 | 0.4281 | 0.68035 | 0.7718 |
| median | 0.33 | 0.28 | 0.9636 | 0.9579 | 0.3506 | 0.3256 | 0.3378 | 0.3708 | 0.6532 | 0.6325 |
| mean | 0.3733 | 0.3533 | 0.9552 | 0.9538 | 0.3810 | 0.3595 | 0.3639 | 0.3362 | 0.6642 | 0.6535 |

Table 1: Table showing metrics for different classifier labels

# References


[1] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386 – 408, 1958.

[2] A. Charnes. The geometry of convergence of simple perceptrons. *Journal of Mathematical Analysis and Applications*, 7(3):475–481, 1963.

[3] Randolph Field, Evelyn Fix, and Joseph Hodges. Discriminatory analysis, non-parametric discrimination: consistency properties. *USAF School of Aviation Medicine*, 1951.

[4] Pierre A.Devijver. A note on ties in voting with the k-nn rule. *Pattern Recognition*, 10, 1978.

[5] Sahibsingh A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6, 1976.

[6] M.T. Orchard. A fast nearest-neighbor search algorithm. In *ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, pages 2297–2300 vol.4, 1991.

[7] Chang-Da Bei and R. Gray. An improvement of the minimum distortion encoding algorithm for vector quantization. *IEEE Transactions on Communications*, 33(10):1132–1133, 1985.

[8] Aditya D. Bhat, Harshith R. Acharya, and Srikanth H.R. A novel solution to the curse of dimensionality in using knns for image classification. In *2019 2nd International Conference on Intelligent Autonomous Systems (ICoIAS)*, pages 32–36, 2019.

[9] Giuseppe Amato and Fabrizio Falchi. Knn based image classification relying on local feature similarity. In *3rd International Conference on Similarity Search and Applications SISAP 2010*, pages 101–108, 2010.

[10] David G. Lowe. istinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(10):91–110, 2004.

[11] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[12] Oren Boiman, Eli Shechtman, and Michal Irani. In defense of nearest-neighbor based image classification. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[13] Vladimir Vapnik and Corinna Cortes. Support-vector networks. 20(3):273–297, 1995.

[14] Larhmam. Maximum-margin hyperplane and margin for an svm trained on two classes. samples on margins are called support vectors. image licensed under the creative commons attribution-share alike 4.0 international. `https://en.wikipedia.org/wiki/File:SVM_margin.png`, 2018. Accessed: 2022-03-10.

[15] Dimitri P. Bertsekas, Angelia Nedic, and Asuman Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, 2 edition.

[16] Naum Z. Shor. *Minimization Methods for Non-differentiable Functions*. Springer-Verlag, 1985.

[17] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: primal estimated sub-gradient solver for svm. *Mathematical Programming*, 127:3–30, 2009.

[18] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th International Conference on Machine Learning*, page 408–415. Association for Computing Machinery, 2008.

[19] L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, volume 2, pages 77–82, 1994.

[20] Marvin Minsky and Seymour Papert. *Perceptrons : an introduction to computational geometry /.* MIT Press, 1988.