CMP304 – Computer Vision and Machine Learning Report
-
Marc Philippe Beaujean (1502932)

Introduction and Data Selection:

Machine learning has seen a recent surge in popularity due to overall technological advances and is finding increasingly more common use in everyday applications, as even games companies are starting to explore the field *(Vincent and Savov)*. The task at hand, is to create a machine learning application, that uses computer vision techniques to recognise six different emotions on an image of a face. There were two main contenders in regards to what software would be used to develop the application, *MatLab* (developed by *MathWorks*) and *OpenCV* (an open source image detection, processing and machine learning library) for Python. The benefits of *MatLab* were that it provided a lot of hands-on documentation and a very useful classification tool, that allows the user to try different techniques on their data very easily. The downsides, however, were that the data structures and confined environment of *MatLab* meant that exploration was limited in regards to pre-processing and feature extraction. *OpenCV* required a lot more research and code, but ultimately generated far better results, while also allowing for more room to show knowledge of the techniques used and proceedings in machine learning. Additionally, the machine learning module *sklearn* was used to generate a confusion matrix and split the data set into testing and training.

Illustration 1: Whole-body picture with point (SURF) feature detection, shown in green

Choosing a high-quality data set is an important part of supervised machine learning and to identify a suitable set, one needs to understand how image feature detection works. In order to determine what type of image would be ideal for the task at hand, a basic feature detection method was applied to a recorded image with a SURF (Speeded-Up Robust Features) function from *MatLab*. SURF, an improvement on the SIFT detection algorithm, uses the Laplacian of Gaussian and Hessian matrix to detect blob-like features in a grayscale image (*Bay, Tuytelaars and Van Gool, 2006*). While this might not be the detection method used in the final application, it seemed reasonable to get an initial indication of how feature extraction works and what features stood out immediately, when applied to a newly created image. The first recorded image included the shirt of the model (*illustration 1*), which had extreme colour contrasts. Since SURF uses a grayscale of the original image to detect features, the shirt distracted from the main target of the feature extraction, which was the face. Thus, it became clear that an image which focused on the face as much as possible would be required for accurate detection. It also becomes evident that while greyscale is an excellent pre-processing method for feature detection, unwanted elements in the picture with extreme grayscale values can easily derivate from the features of interest.

Illustration 2: A subject from the Cohn Kanade image set being Angry (left) and Disgusted

Given that the task at hand requires the identification of six different emotions, it can be identified as a non-binary classification problem. To solve this problem using a supervised machine learning approach, a large data set of images needs to be accumulated, from which features are extracted, so that they can be used to identify our classified element in any given image *(Machine Learning Challenges: Choosing the Best Model and Avoiding Overfitting, 2016)*. The importance of having an abundance of data cannot be understated, as having too little will usually lead to the algorithm overfitting the data. This is because due to the lack of data, the classifier becomes too familiar with the training data and is unable to recognise anything that does not resemble those exact images *(EliteDataScience, 2017)*. Before even the pre-processing stage, a large amount of accuracy can already be lost, if the data provided is not consistent or the amount is simply too small. For example, if the images in two categories within the data set are too similar (a common offender in the given task are images labelled under “Anger” and “Disgust”, as seen in *illustration 2*), recognition accuracy will decrease as the algorithm will fail to identify distinct features to separate the two. To fight the chance of overfitting, a large training set (composed of various smaller sets from blackboard and the internet – see references for full list) was created and labelled for classification. Furthermore, the images were carefully reselected and some images, where the labelled emotion was not immediately apparent, removed from the data set. This process of elimination increased accuracy by 8%, when applied to the data set for the finished application.

Pre-Processing:

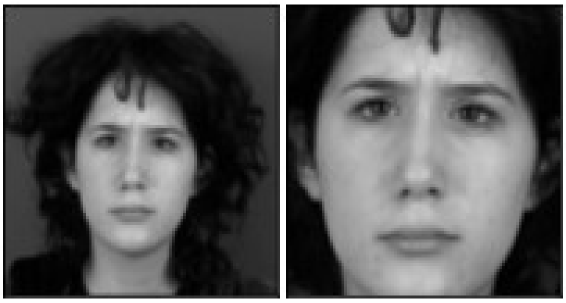


Illustration 3: Result of the OpenCV Haar Cascade function applied to an image

To increase the efficiency of the algorithms ability to detect features in the provided data, image pre-processing is required. When trying to extract facial features, especially when the task is to extract something as complex as an emotion, it is important to make sure that only the face is being analysed and no background noise gets picked up (as mentioned in the introduction). One method that can be used to isolate faces from images, is via the Viola Jones algorithm. The Viola Jones algorithm iterates over a grayscale image (this is known as “cascading”), until it finds the Haar-like features that it is trying to detect, which for the given task would be the eyes, nose, mouth, etc. *(Viola and Jones, 2001)*. These features are represented by rectangles that are formed around areas of the image, determined by the contrast in grayscale values and weights at a given area (*vocal.com, 2017*). *OpenCV* offers a multitude of cascade classifiers, which allows us to find a face in a sampled image and returns the coordinates of where the face begins, as well as the dimensions. Given this information, the face can be extracted from the image as an area of interest, making it less likely that background noise interferes when extracting features. Images on which no face was detected are discarded and not used for training. Four different face classifiers are used with this method, to increase accuracy *(Van Gent, 2016)*.

Cascades	1	2	3	4
Failed Detections	33	22	22	19

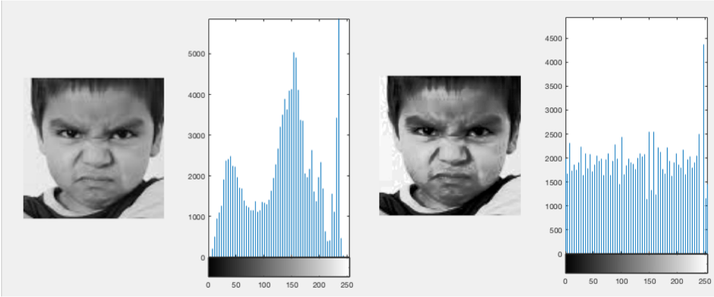


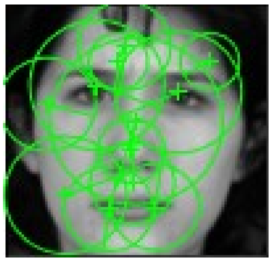
Illustration 4: A grayscale image before (left) and after histogram equalisation was applied to it.

One way to enhance grayscales for computer vision, is to use histogram equalization. This technique focuses on increasing the range of contrasts in the image, by equalising the grayscale values on an image to make use of the entire scale. This makes it easier to distinguish features, as the difference between the darkest and lightest points within the image are much greater. Equalisation also exposes more subtle features, by giving them distinct grayscale values that improve the effectiveness of most computer vision techniques applied to them *(docs.opencv.org, 2014)*. Given that the viola jones cascade uses grayscale to identify faces, it is expected that more faces are identified in the data set if the images are equalised using this technique, which is supported by the data collected in the table below (three cascades were used).

Images Equalised	Yes	No
Failed Detections	22	25

While there are many other image enhancement techniques, a lot of them are hard to apply to such a large set of data, because they usually require specific preparation for a given type of image. For example, if an image is blurry, a Wiener Filter technique can be applied to an image *(De.mathworks.com, n.d.)*. If a particular image is suffering from low light or haze, a Dehazing Algorithm can help normalize the features that require more brightness for effective detection *(Hsieh, Chen and Lin, 2014)*. For this application, however, this would require picking out specific images where these problems are apparent. From the sets used, no image was identified that was particularly blurry or had other extreme noise distortion, that could interfere with detection. Likewise, only very few images seemed to suffer from bad exposure and lighting, so these methods were not applied for this application.

Feature Extraction:



llustration 5: SURF on an extracted face

For computer vision, the feature extraction methods are plentiful. The main task for this application, was to identify an extraction method that would allow the classifier to differentiate between various facial emotions, when handed a brand new image. As mentioned previously, one of the ways that such features can be extracted, is by using SURF. While SURF features can be applied to an image of any size or orientation dynamically, as well as different scales (*Beyeler, 2017*), the raw output of these features is not very effective for machine learning. This is because the number of features detected varies for each image and can lead to inconsistencies in the data, which are hard for the supervised classifiers to interpret. To use these features for object recognition, a technique called “Bag of Words” is used, which uses the extracted points and turns them into descriptors. These descriptors, also known as the “Vocabulary”, can then be used for classification, as the amount of descriptors found in an image will be more consistent and easier to interpret (*Kundrac, n.d.*). *MatLab* has a function that creates a Bag Of Words using a large set of images, *OpenCV* however does not have this functionality.

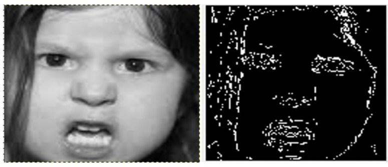


Illustration 6: Image before and after Roberts edge operator was applied to it

Another method that can be used to extract features in images, is by using a Histogram of Gradients (HOG simplified). The first step to retrieving the HOG features, is to calculate the x and y gradients of the image, which can be done with an edge detection operator like Sobel or Roberts (*Price, 1996*). Using these values, it is possible to obtain the magnitude and direction of each gradient, using the formulas $g=\sqrt{(g_x^2+g_y^2)}$ and $\theta=\arctan(\frac{g_y}{g_x})$ *(Mallick, 2016).* Doing this will highlight the edges of an image, which can be used to identify features for classification. For more compact representation, the image is further divided into x by x cells, with a histogram of gradients being created for each cell, to reduce the impact that noise has on the features. The biggest benefit of using HOG over SURF features, is that HOG creates outputs descriptor matrices that are equal for all images and thus, can be used for classification without any further processing. Because HOG requires a fixed image size, the area of interest generated by the Viola Jones face classifier is resized to a specified scale, in this case a resolution of 100 by 100 (*Beyeler, 2017*). With this resolution, 1980 features can be extracted from each image. It also means that an arbitrary data base of images can be used in conjunction with HOG, even if these aren’t all of the same resolution.

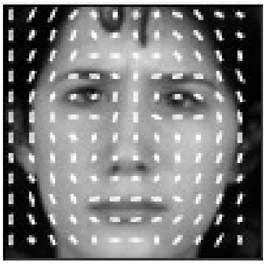


Illustration 7: HOG features on an extracted face

Testing and Validation:

Before discussing some of the classifiers that were tested for this application and the recorded results, it seems important to outline how measurements were made and the accuracy of the classifier was scored. The application includes two measuring functions, one for a regular test-training set split and one for two-fold cross-validation. For the testing and training set split, 20% of the full data set are randomly separated from the rest, with equal amounts of data for each label. This is done, so that after the classifier memorises the training set, data is available that can be used to assess how well the classifier scores when trying to label unseen information. The data is divided using a function from the python module *sklearn*, called *train_test_split* (*Beyeler, 2017*). The random seed parameter in the function means that when iterating through different sets of training and testing data, it is possible to maintain consistency over how the data is split, thus allowing for accurate and non-biased or chance-based measurements. Another popular technique used in machine learning to measure performance and tune hyperparameters, is k-fold cross-validation. The data is split *k* times (these segmentations of the data are called *folds*), then each fold is trained. Finally, each fold is used as a training set and compared to the other (*k - 1)* folds. This results in *k* accuracy scores, so the mean of these scores is the final accuracy score used to evaluate the classifier. The upside of this, is that all of the data is used for training and testing, leading to a more stable accuracy measurement. A function for k-fold cross-validation was implemented using *sklearn (Bronshtein, 2017*). Finally, the program automatically creates a spreadsheet, that keeps track of the cross-validation and testing scores for each trial (*McNamara, n.d.*).

Classification:

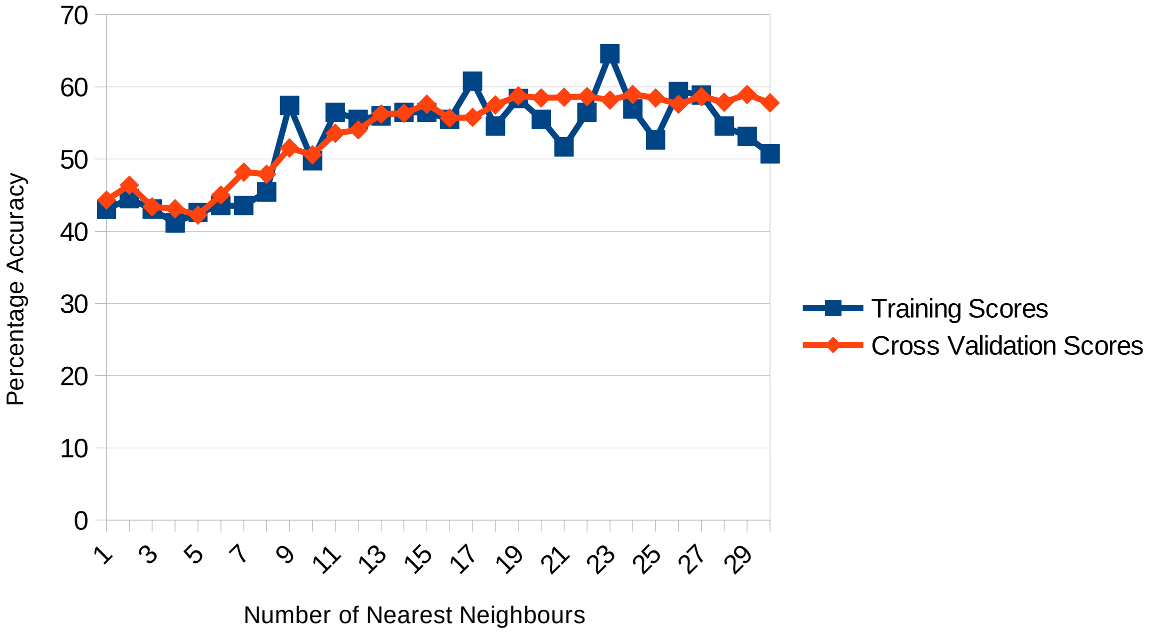
The most simplistic machine learner, can arguably be considered the k-nearest neighbour (KNN for short) algorithm. KNN uses similarity to classify data and thus, does not actually require a lot of training when compared to other machine learning algorithms (though it does store a lot of data, which it transfers to the testing stage to make predictions). In simplified terms, the KNN learner will try to label a feature vector based on it’s similarities to other feature vectors in the training set. Once a majority of features with a specific label are matched with the new feature, it will classify that feature as belonging to that label. In a regression problem, where data is used to make a numerical prediction, the benefit of KNN comes in that form that it does not make any presumptions about that data it is fed, which can be beneficial in cases where data is non-linear (*Bronshtein, 2017*). As mentioned previously, the predictions are made based on the nearest match in features space, but sometimes more data needs to be taken into consideration. This is why the algorithm is called *k* nearest neighbour, as *k* is a parameter that determines how many features the algorithm considers before making a prediction (*Beyeler, 2017*).

Another classification learner, is the decision tree. It inherits its name, because the way it classifies information and makes decisions, can be represented in a tree diagram. The tree has multiple nodes, which are split into new branches whenever the classifier undergoes a new calculation. This means that until all data is classified, the tree will continue to create new branches, which impact that decisions that the tree makes when dealing with new data. For a tree to stop splitting the data, a minimum score threshold needs to be determined. Once the classification score does not increment by an amount that exceeds the given threshold after a split, the tree is complete. For the following tests, a threshold of 0.1 was used (increasing this value will lead to a premature termination of the classification algorithm). Random trees (also known as random forest) is a technique that uses multiple decision trees to classify data. It takes into account the results of each tree and decides which to use via a voting system. The reason why this method is often considered superior to using a single decision tree, is because the use of multiple trees can compensate for overfitting (*Beyeler, 2017*).

For the application, a Support Vector Machine was chosen as the classifier, due to the fact that since it was introduced in 1990, SVM has always been a well-established supervised learner. An SVM uses decision boundaries, which dynamically change during the training process as more data gets added to the learner's memory (*Beyeler, 2017*). The decision boundary is formed by comparing the differences between sets of feature vectors that have already been classified and trying to separate them using a hyperplane(s). To adjust the boundaries, the biggest difference between feature vectors of separate labels is used (the margin). Feature vectors that are near the centre (those similar to feature vectors that have been classified differently) are called support vectors and ignored when trying to determine the decision boundary. This makes SVM more robust towards outliers and misclassification *(Ray, 2017)*. One of the biggest benefits that SVM classifiers have over others, is the use of kernels functions, which can be specified to define how the classifier learns and adjusts the decision boundaries. Kernels supported in *OpenCV* include linear, sigmoid, RBF, polynomial and histogram intersection.

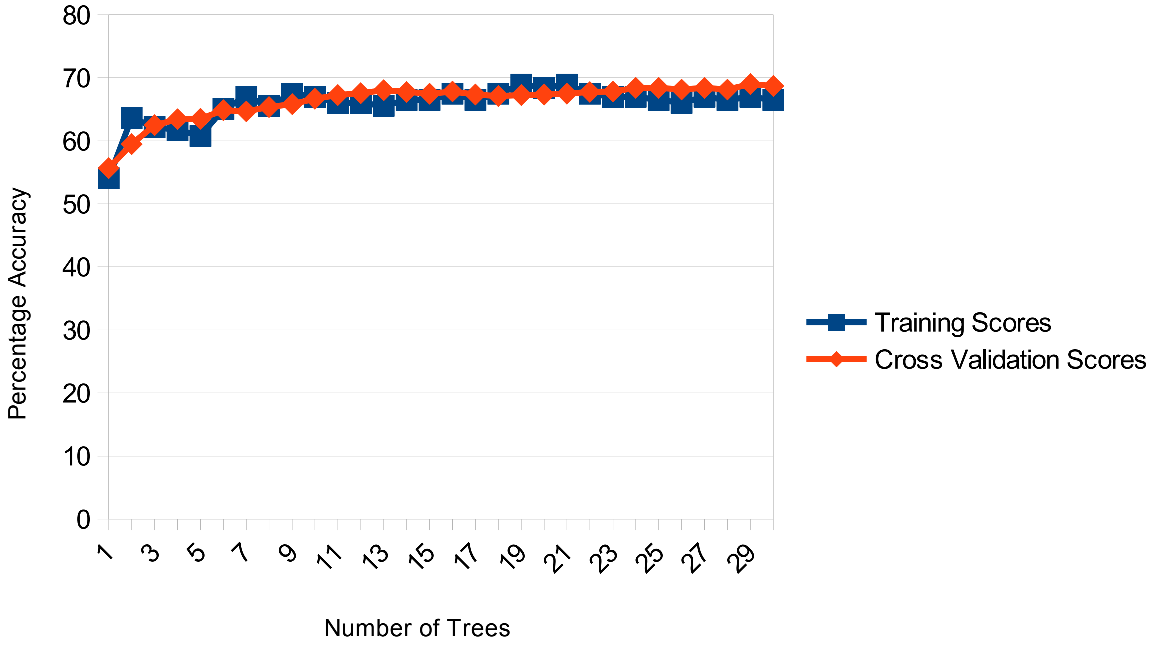
Results:

To find the best classifier for the provided task, it is important to tune the hyperparameters for each and find out when it best performs. For the k-Nearest Neighbour classifier, that would be the number of nearest neighbours taken into account. For the following data, the amount of neighbours taken into account for classification was gradually incremented from one to 30.



As can be determined by the provided results, a value of *k* that is below 10, will yield worse results than a value that is greater, however performance will stagnate and stop incrementing after a certain point. A low value for *k* will make the system very suspect to noise in the data, while a value that is too high will defeat the purpose of the algorithm. This is because the pool of data (also called the "neighbourhood") becomes so large, that recognising patterns or significant differences between the features begins to get increasingly difficult. Of course, there are also computational problems that can be considered when incrementing the value for *k*.

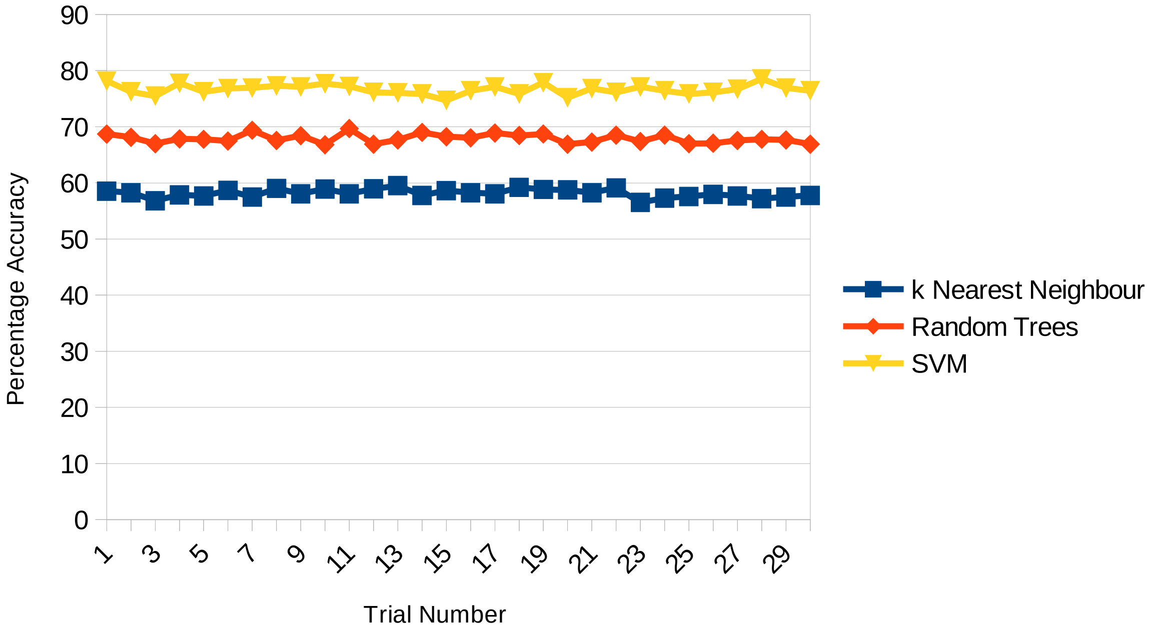
Theoretically, there should be no downside to adding trees to a random decision tree, when trying to make the best possible classifier. Each tree represents an additional source of information and can increase data accuracy for validation. However, in many practical situations, the computational expenses of having a very high amount of trees can take its toll on the efficiency and usability of the application.



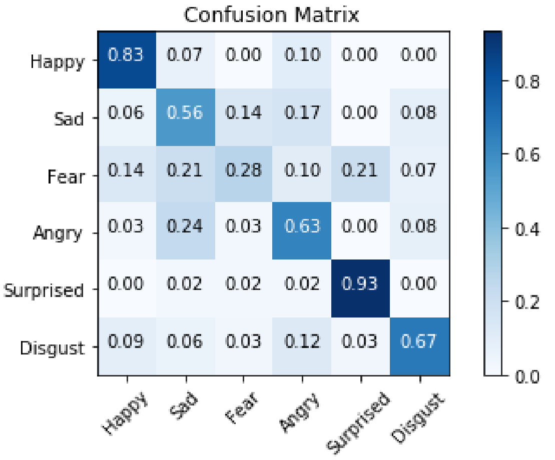
A similar pattern can be seen as when incrementing the value of *k* for the nearest neighbour classifier (with the deciding difference, that the amount of trees was incremented by 10 each time). While the hypothesis, that more trees lead to better results, is supported by the results, it is possible to see that the increment in accuracy begins to stagnate over time.

Finally, all three implemented classifiers are compared directly. As can be seen from previous graphs, the cross-validation gives a much more consistent measurement of a classifiers performance, hence it was the only measurement used for the following graph. For the KNN learner, 50 neighbours were taken into account. The number of trees used in random trees, was 300. A linear kernel was used in SVM.

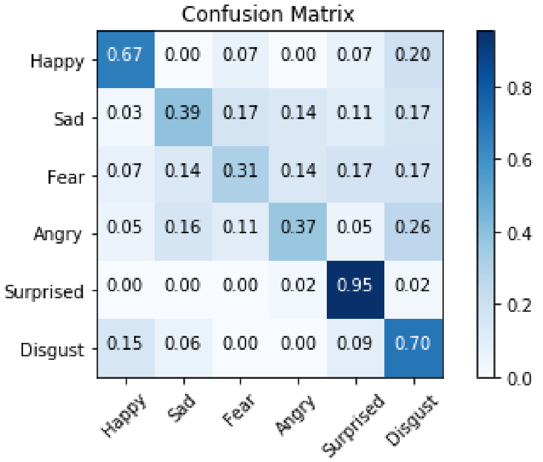
As can be seen from the graph, the different classifiers performed in almost perfect 10 percent intervals, with the support vector machine performing the best. Possible reasons for this include the fact that nearest neighbour algorithms are often too simplistic for complex classification tasks and decision trees can have trouble with non-binary classification problems.



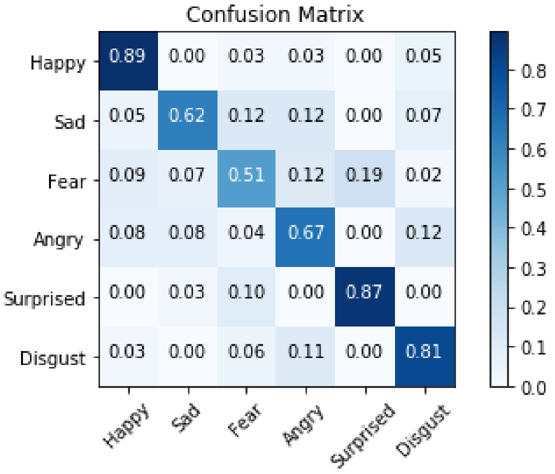
It is also possible to identify which emotions were the most difficult for the classifiers to identify during testing, by using a classification matrix.



Confusion matrix for k nearest neighbour



Confusion matrix for random trees



Confusion matrix for support vector machine

Finally, a table can be used to compare the mean accuracy and standard deviation of the classifiers.

Classifier	KNN	Random Trees	SVM
Mean Accuracy	58.14	67.9	76.64
Standard Deviation	0.73	0.79	0.87

Conclusion and Critical Discussion:

From the tests made, it is possible to conclude that for this particular application, the support vector machine was the best performing classifier and thus should be the one used for this application. However, even with this method, peek performance was only at around 80% accuracy on the test set. There can be multiple reasons for this, the main one being a lack of well-distributed data. More specifically, there are inconsistencies in the amount of images available for each of the six categories. This is reflected in the confusion matrices, that can be used to see for what labels the classifier achieved the worst percentage accuracy. If the amount of samples was increased and the data samples evenly distributed amongst each category, the percentage accuracy score could be raised significantly.

Of course another aspect that made classification less effective, is the nature of the task. A binary classification task, for example, is much simpler to handle than a task with a lot of categories. In general, machine learner's will be able to classify features better, if there are less possible solutions (*Van Gent, 2016*). As mentioned previously, some of the features in each of these categories have significant overlap, further increasing the difficulty to differentiate between the two. The ideal way to compensate for this, would be by hand selecting each image or creating a personal data set, that would allow for more control over how each feature should be recognised.

In regards to feature extraction, there are many methods that have not been explored. It is possible that HOG features might not be that suitable for recognising emotions, as it was initially used to determine if an image contained a pedestrian or not (*Malick, 2016*). Many other methods, like eigenfaces, SURF features, ORB features, etc. could have resulted in higher accuracy, because they might have offered a better descriptor for each of the different emotions. To see how the data performs on completely unseen images, a webcam class in *OpenCV* was used, that returns the image of each frame. This image is then pre-processed with all the steps mentioned earlier and the features extracted. The prediction of the classifier is then displayed at the top left corner, in integer form.

References:

Available at: http://opencvpythontutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html [Accessed 29 Mar. 2018].

- Bay, H., Tuytelaars, T. and Van Gool, L. (2006). *SURF: Speeded Up Robust Features*. [ebook] Zurich: Springer, Berlin, Heidelberg, pp.2-7. Available at: http://www.vision.ee.ethz.ch/~surf/eccv06.pdf [Accessed 5 Apr. 2018].
- Beyeler, M. (2017). *Machine learning for OpenCV*. Birmingham: Packt Publishing, pp.154-161, 300-307.
- Bronshtein, A. (2017). *A Quick Introduction to K-Nearest Neighbors Algorithm*. [online] Medium. Available at: https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7 [Accessed 7 Apr. 2018].
- Bronshtein, A. (2017). *Train/Test Split and Cross Validation in Python – Towards Data Science*. [online] Towards Data Science. Available at: https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6 [Accessed 6 Apr. 2018].
- de.mathworks.com. (n.d.). *Deblurring Images Using a Wiener Filter- MATLAB & Simulink Example- MathWorks United Kingdom*. [online] Available at: https://de.mathworks.com/help/images/examples/deblurring-images-using-a-wiener-filter.html [Accessed 31 Mar. 2018].
- Docs.opencv.org. (2014). *Histogram Equalization — OpenCV 2.4.13.6 documentation*. [online] Available at: https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html [Accessed 30 Mar. 2018].
- EliteDataScience. (2017). *Overfitting in Machine Learning: What It Is and How to Prevent It*. [online] Available at: https://elitedatascience.com/overfitting-in-machine-learning [Accessed 5 Apr. 2018].
- Hsieh, C., Chen, C. and Lin, Y. (2014). *Adaptive fast image dehazing algorithm*. [online] Orlando: IEEE, p.1. Available at: http://ieeexplore.ieee.org/document/7011755/ [Accessed 31 Mar. 2018].
- Kanade, T., Cohn, J. F., & Tian, Y. (2000). Comprehensive database for facial expression analysis. Paper presented at the Fourth IEEE International Conference on Automatic Face and Gesture Recognition.

- Kundrac, J. (n.d.). *Bag of visual words in OpenCV*. [online] Vision and Graphics Group. Available at: http://vgg.fiit.stuba.sk/2015-02/bag-of-visual-words-in-opencv/ [Accessed 5 Apr. 2018].
- Learned-Miller, E., B. Huang, G., Roy Chowdhury, A., Li, H. and Hua, G. (2007). *Labelled Faces in the Wild*. [image] Available at: http://vis-www.cs.umass.edu/lfw/ [Accessed 8 Apr. 2018].
- Machine Learning Challenges: Choosing the Best Model and Avoiding Overfitting. (2016). 1st ed. [ebook] MathWorks, pp.6-8. Available at: https://www.mathworks.com/tagteam/87371_92974v00_Machine_Learning_Whitepaper.pdf [Accessed 30 Mar. 2018].
- Mallick, S. (2016). *Histogram of Oriented Gradients | Learn OpenCV*. [online] Learnopencv.com. Available at: https://www.learnopencv.com/histogram-of-oriented-gradients/ [Accessed 5 Apr. 2018].
- McNamara, J. (n.d.). *Tutorial 1: Create a simple XLSX file — XlsxWriter Documentation*. [online] XlsxWriter. Available at: http://xlsxwriter.readthedocs.io/tutorial01.html [Accessed 6 Apr. 2018].
- Michael J. Lyons, Shigeru Akemastu, Miyuki Kamachi, Jiro Gyoba. Coding Facial Expressions with Gabor Wavelets, 3rd IEEE International Conference on Automatic Face and Gesture Recognition, pp. 200-205 (1998).
- Mordvintsev, A. and K., A. (2018). *Introduction to SURF (Speeded-Up Robust Features) — OpenCV-Python Tutorials 1 documentation*. [online] OpenCV Python Tutorials.
- N. Aifanti, C. Papachristou and A. Delopoulos, ”The MUG Facial Expression Database,” in Proc. 11th Int. Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS), Desenzano, Italy, April 12-14 2010.
- Pingel, J. (2018). *Edge Detection with MatLab*. [video] Available at: https://de.mathworks.com/videos/edge-detection-with-matlab-119353.html [Accessed 29 Mar. 2018].
- Price, S. (1996). *Edges: Gradient Edge Detection*. [online] University of Edinburgh School of Informatics. Available at: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/edges/gradient.htm [Accessed 5 Apr. 2018].
- Ray, S. (2017). *Understanding Support Vector Machine algorithm from examples (along with code)*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/ [Accessed 5 Apr. 2018].
- Van Gent, P. (2016). Emotion Recognition With Python, OpenCV and a Face Dataset. [Blog] *PAULVANGENT.COM*. Available at: http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/ [Accessed 5 Apr. 2018].
- Vincent, J. and Savov, V. (2018). *EA has started training AI players in Battlefield 1*. [online] The Verge. Available at: https://www.theverge.com/2018/3/22/17150918/ea-dice-seed-battlefield-1-ai-shooter [Accessed 29 Mar. 2018].
- Viola, P. and Jones, M. (2001). *Rapid Object Detection using a Boosted Cascade of Simple Features*.
- vocal.com. (2017). *Face Detection using Viola-Jones Algorithm*. [online] Available at: https://www.vocal.com/video/face-detection-using-viola-jones-algorithm/ [Accessed 6 Apr. 2018].