Computer Vision: Facial and Optical Character Recognition

Stephanie Jury

1. Introduction

The problem of visual perception is a trivial one for human adults and toddlers alike and over the past decade we have come to expect similar capabilities from our digital devices. We assume that a digital camera will autofocus on faces detected in its viewfinder and that, later on, a freely downloaded application will predict which friends we want to tag in the photo taken. However, such progress made in the field of computer vision is extremely impressive, given the problem of visual perception in the natural world is hugely complex and inherently unconstrained. It is only the combination of improvements in processing power, progression in the field of neural computing and an explosion in the number of accessible digital images accessible for training, which has hastened the advancements in the field of computer vision which make this "human-like" perception a reality.

In this project I attempt to recreate elements of the detection and classification capabilities described above by writing two functions, a face recognition and classification function, "RecogniseFace" and a numeric character recognition function "detectNum". Both functions were created using MATLAB R2018b. RecogniseFace detects faces of my classmates in individual or group images (.jpg) and returns the ground truth label ("class") originally assigned to each of them. This is achieved in one of two ways, either by extracting features from the passed image, by generating speeded up robust features ("SURF") or creating histogram of oriented gradients ("HOG") features, followed by classification by trained support vector machine ("SVM") or multilayer perceptron ("MLP"), or by passing the original image to the pre-trained convolutional neural network ("CNN"), "AlexNet". detectNum combines image pre-processing and optical character recognition techniques to return the digits printed on an A4 sheet of paper held any number of classmates in either an image (.jpg) or video (.mov) file.

2. Data Collection and Pre-Processing

A full class of computer vision students was photographed and filmed to create a database of group images and videos, with some files not provided to the students and constituting an unseen "test set". Additionally, students were photographed and filmed individually, from varying angles and holding an A4 sheet of paper with printed digits assigning their class. These constituted a training set of 69 individual students, generating 69 folders with class labels ranging non-sequentially from "01" to "81".

2.1. Face Extraction

With this training set I used MATLAB's built-in, Viola-Jones cascade object detector to extract faces from every picture of each student. Through trial and error, I determined that the properties resulting in the most consistently correct extraction were a "MergeThreshold" of 10 and "MinSize" of [20, 20]. This merge threshold, higher than the default, helped suppress false detections by requiring that the target object be detected multiple times during the detection phase, and was aided by the minimum detectable region being 20 x 20 pixels. Below these values the detector returned false positives including clothing logos, and background shadows. The coordinates of each detected face were used to create a bounding box, which was used to crop each face from the picture. Each cropped face was then resized to 227 x 227 pixels (the size required to be supplied to the AlexNet classifier) and saved back into the class folder from which the image was sourced. Any non-faces detected were deleted.

2.2. Training Set Augmentation

I chose to augment this initial training set of faces in two ways. Firstly, I extracted faces from every fifth frame of the video files of the individuals, resulting in between 80 and 150 new labelled images per class, and from the whole group, resulting in approximately 130,000 unlabelled images (number of faces x number of extracted frames). Secondly, I used this small number of labelled images to train a preliminary SVM classifier using SURF features and applied it to the huge folder of unlabelled faces. I prepended the predicted class onto each image file, sorted the files by their new name, and manually identified which faces were correctly classified so that I could move them in batches into the correct training folders. By inspection, the classification accuracy on this first pass was approximately 50%, making the manual task surprisingly time efficient. I also identified two individuals who had not been assigned a class, so I created classes “82” and “83”, bringing the total number of classes to 71. I repeated the model training on the augmented training set, and again carried out the prediction, renaming and moving process. This time, classification accuracy improved to around 80%. The final pass left me with approximately 1,000 images per class which I randomly reduced to a consistent 360 images per class. This provided a highly diverse set of training images for each class, whilst remaining small enough to be computationally manageable.

3. Face Recognition and Classification

3.1. Function Overview

RecogniseFace takes three string arguments, image path (“I”), feature type (“featureType”) and classifier name (“classifierName”), and returns “P”, an N x 3 matrix, where “N” is the number of faces detected. Table 1 outlines the combinations of arguments which can be provided to the function. For each face detected, i.e. for each row, column one contains that person’s predicted class, column two the x-pixel coordinate of the central position of the detected face, and column three the y-pixel coordinate of the central position of the detected face. The function returns a label and coordinates of 0 for any faces found in the picture that did not match the faces used to train the classifiers.

I	featureType	classifierName	Trained Classifier
"image path"	"SURF"	"SVM"	SVM_SURF.mat
"image path"	"SURF"	"MLP"	MLP_SURF.mat
"image path"	"HOG"	"SVM"	SVM_HOG.mat
"image path"	"HOG"	"MLP"	MLP_HOG.mat
"image path"	"None"	"AlexNet"	AlexNet.mat

Table 1: Valid combinations of arguments for RecogniseFace.

3.2. Approach

I trained each classifier according to the process diagram illustrated in Figure 1. For every image in the training set, image pre-processing comprised face detection as described in section 2.1, and class label extraction. I stored these outputs in 69 separate MATLAB “ImageDatastore” objects, one per class. I extracted SURF and HOG features from each image class and used these alongside the known class labels to train SVM and MLP classifiers. The AlexNet classifier was trained on the 227 x 227 images directly. I saved five trained classifiers produced for use in validation and testing, whereby I passed group images without class labels. Depending on the feature extraction and classifier type selected, the function applies the appropriate saved classifier, as shown in Table 1, and a list of class labels of all recognised faces are returned.

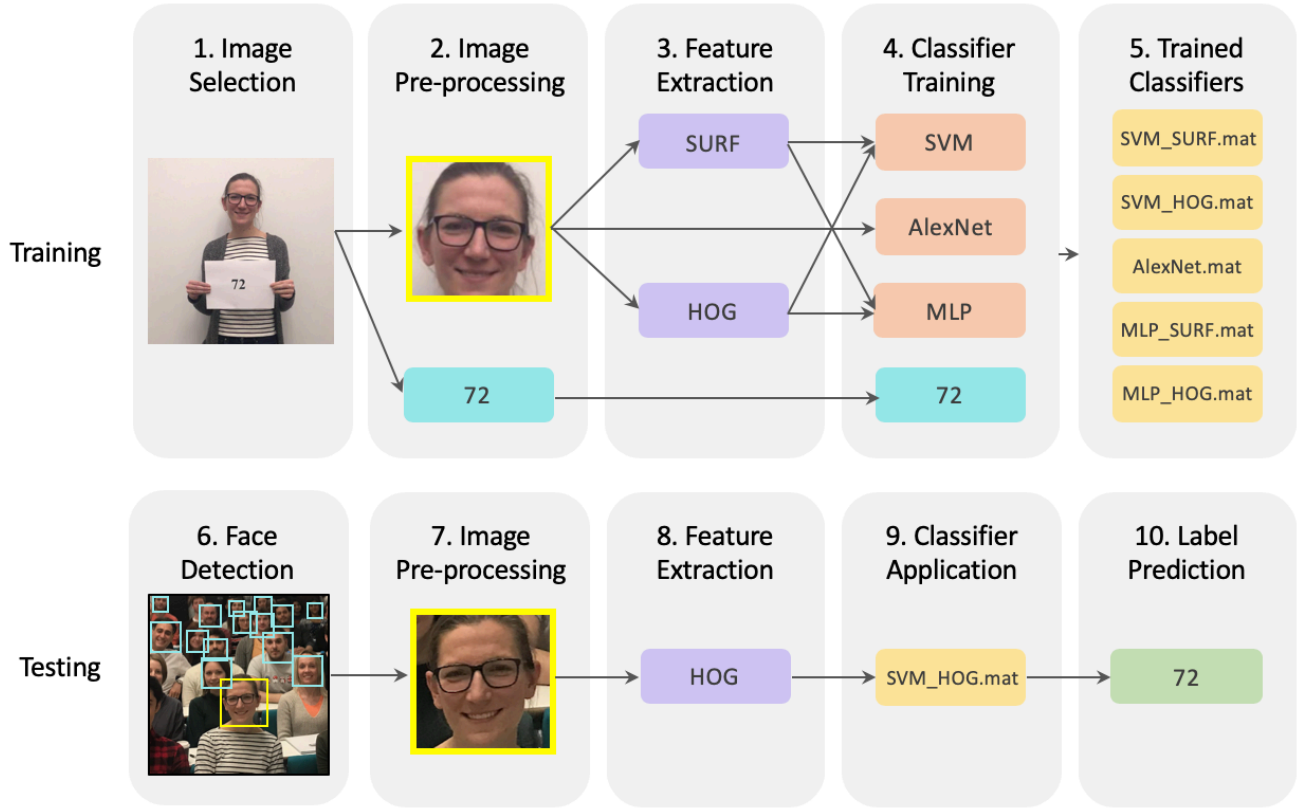

Figure 1: Graphical representation of the model training and label prediction processes.

3.2. Feature Extraction

3.2.1 Speeded Up Robust Features (SURF)

In their 2006 paper "SURF: Speeded Up Robust Features", Bay, H., Tuytelaars, T. and Van Gool, L, introduced a novel feature extraction technique which, like its harbinger scale-invariant feature transform ("SIFT"), was invariant to image scale and rotation but was faster and more robust. SURF uses integral images for image convolutions, an integer approximation of the proven Hessian, matrix-based blob detector, and a descriptor based on the sum of Haar wavelet responses [1].

Advantages of this kind of approximation is that convolutions can be calculated quickly and can be done in parallel for different scales. Although SURF feature extraction is invariant to image scale, blurring and rotation, it does not perform well when handling viewpoint and illumination change [2]. Figure 2 shows blob features detected by the SURF algorithm overlaid on the input image.

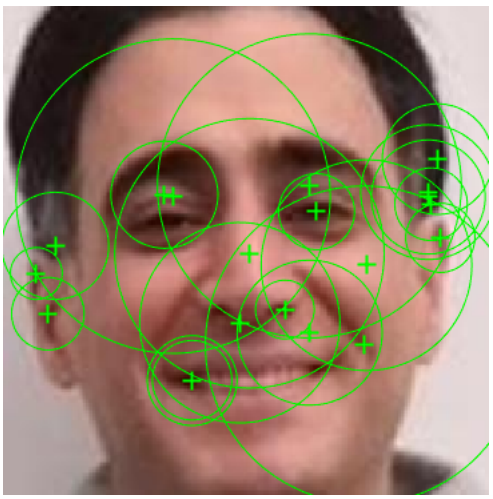

Figure 2: Blob features detected by the SURF algorithm, overlaid on the original image.

To generate SURF features for the training data, I used the "bagOfFeatures" function with default parameters. In one call, this extracts SURF features from all images in all image classes and subsequently constructs a visual vocabulary, reducing the number of features through quantisation of feature space using *K*-means clustering [3]. This is analogous to the bag-of-words model used in document classification which creates a dictionary of features, but instead returns a bagOfFeatures object, with a default size of 500 features per input picture.

3.2.2 Histogram of Oriented Gradients (HOG)

In their 2005 paper "Histograms of Oriented Gradients for Human Detection", Dalal, N. and Tiggs, B. introduced the now widely used HOG feature extraction technique. HOG features are counts of the occurrences of particular orientations of changes in image density or colour (gradients) in uniformly spaced cells, which capture local intensity gradients and edge directions [3].

To calculate these HOG features, the image is divided into radial or rectangular cells, then each pixel casts a weighted vote for an orientation-based histogram channel based on the values found in the gradient computation. To account for changes in illumination and contrast, the gradient strengths must be locally normalized by grouping cells together into larger, connected blocks [3]. The HOG descriptor therefore comprises a vector of components of the normalised histograms from all of the blocks. Figure 3 shows these gradients overlaid on the input image.

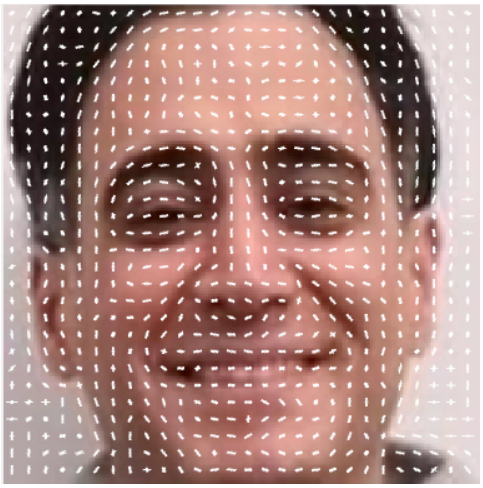

Figure 3: Example of HOG features, overlaid on the original image.

To generate HOG features for the training data, I used the "extractHOGFeatures" function with a small "CellSize" of [8 8] in order to retain the fine detail of the facial images.

3.4. Classifiers

3.4.1. Classifier Training

After completing the data collection and pre-processing steps outlined in section 2, I carried out random splitting of all the images in the "ImageDatastore" objects, placing 70% of the images in each class into a "classifier training" set and the remaining 30% into a "classifier validation" set. This left 252 images per class in the training set and 108 images per class in the validation set.

3.4.2. Support Vector Machine (SVM)

SVMs are a family of non-probabilistic, linear classifiers. Applied to a binary classification problem, the algorithm searches for an optimal hyperplane which maximises the geometric margin of separation between positive and negative observations, as shown in Figure 4. In the case that observations are not linearly separable in the finite-dimensional space of the original problem, the space may be mapped onto a much higher dimensional space in which separation is achievable. In this case the "kernel" can be applied, which avoids the explicit mapping of observations into the higher-dimensional space, instead defining them in terms of a kernel function that computes only the inner (dot) product of pairs of observations in that space.

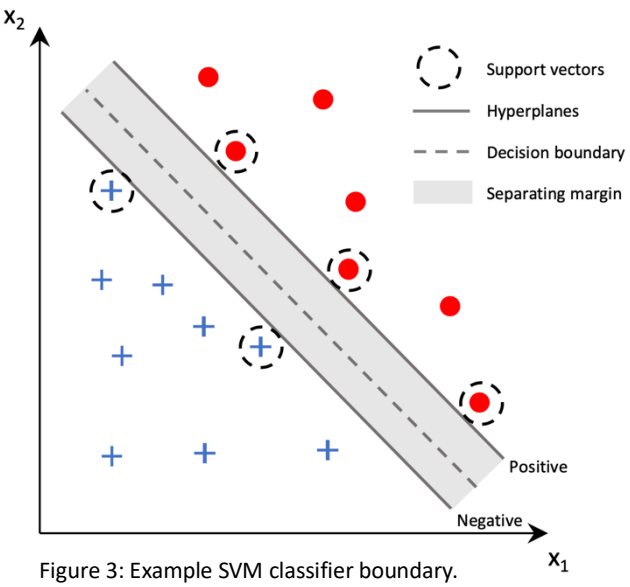

Figure 3: Example SVM classifier boundary.

SVM models exhibit good generalization performance without prior domain knowledge and, due to the convex nature of the optimisation problem, are guaranteed to find the global minimum error. However, because SVM models are only directly applicable to binary classification problems, they require multi-class tasks to be framed as a sequence of binary tasks, in a "one versus rest" or "one versus one" approach. They can also be

memory intensive in both training and testing when the feature space is large, as is often the case in image classification tasks.

I used the in-built function “fitcecoc” to train both the SVM with SURF features (SVM_SURF.mat) and SVM with HOG features (SVM_HOG.mat) classifiers. Both SVM models performed well in validation, with SVM with SURF features achieving a classification accuracy of 98.5%, and SVM with HOG features achieving the highest classification accuracy of all the trained models, at 98.9%. The confusion matrix associated with this best model is shown in Figure 4.

Figure 4: Confusion matrix illustrating classification accuracy of the SVM model trained using image HOG features when tested on the validation image set.

3.4.3. Multilayer Perception (MLP)

MLPs are a class of feedforward neural network model, structures that resemble the biological networks in the brain and map a set of input data onto a set of appropriate outputs. A network's architecture comprises multiple layers of parallel neurons, including an input layer, one or more hidden layers, and an output layer, with a system of synaptic weights and biases creating a fully-connected, directed graph between them, an example of which is shown in Figure 5. Multiple layers and non-linear activation functions enable MLPs to distinguish data that are not linearly separable in the problem space. One of the most popular methods of training an MLP model is backpropagation, a supervised learning technique that adjusts the weight of neurons in the direction that minimises the gradient of a loss function (gradient descent). Neural networks are universal approximators which can approximate virtually any function and exhibit good fault tolerance due to their distributed, brain-like structure.

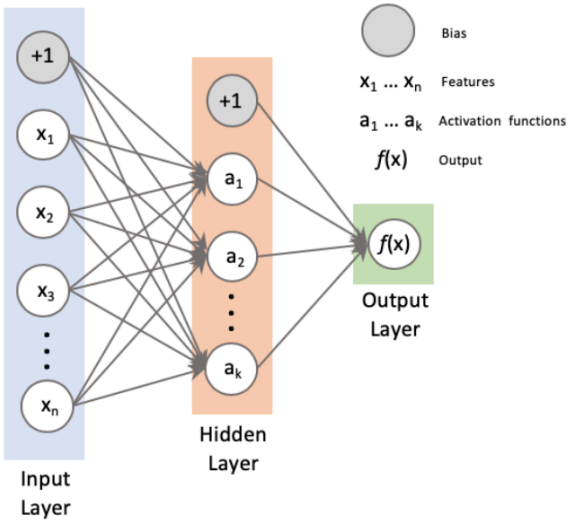

Figure 5: Example feedforward neural network structure.

I used the in-built "Deep Learning Toolkit" to train MLP models with both the SVM with SURF features (MLP_SURF.mat) and with HOG features (MLP_HOG.mat) classifiers. I tested a range of hidden neurons in the hidden layer from 1 to 100, selecting 50 for the final model, being the value that best reduced the degree of under and overfitting. I used the default training function, default scaled conjugate gradient backpropagation, which adjusted neuron weights with every iteration, and cross entropy as the performance function. The structure of this MLP is shown in Figure 6.

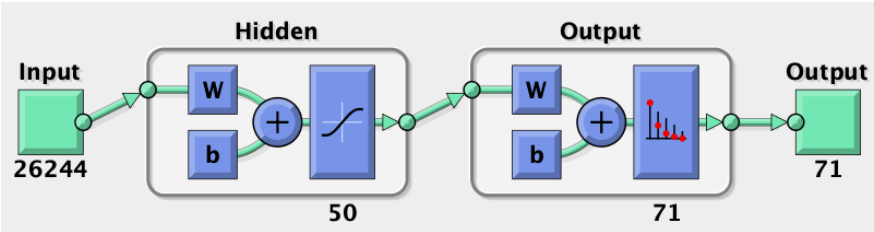

Figure 6: MLP model architecture diagram generated by MATLAB's Deep Learning Toolkit during model training.

Both MLP models performed worse than the SVM models, with MLP with SURF features achieving a classification accuracy of 86.2%, and MLP with HOG features achieving a classification accuracy of 86.1%.

3.4.4. AlexNet

AlexNet is a CNN designed by Alex Krizhevsky and presented in the 2012 paper “ImageNet Classification with Deep Convolutional Neural Networks” [6]. It was trained on the “ImageNet” dataset, containing over one million images belonging to more than one thousand classes that were manually labelled by humans in Amazon’s Mechanical Turk program [7]. The structure of the AlexNet CNN is shown in Figure 7, and comprises five convolutional layers (C1 to C5) and three fully-connected layers (F1, F2, Output). A rectified linear unit (“ReLU”) activation function is applied after each convolutional and fully-connected layer, improving speed without losing classification accuracy. Neuron dropout (random exclusion) is applied before the first and second fully-connected layer to minimise overfitting, and max pooling image discretisation is performed during normalisation to reduce computational cost and the total size of the network. Although 224 x 224 is implied in Figure 7, AlexNet takes an RGB input image of size 227 x 227 pixels and does not require any feature extraction method to be applied as image processing is built-in to the structure of the network.

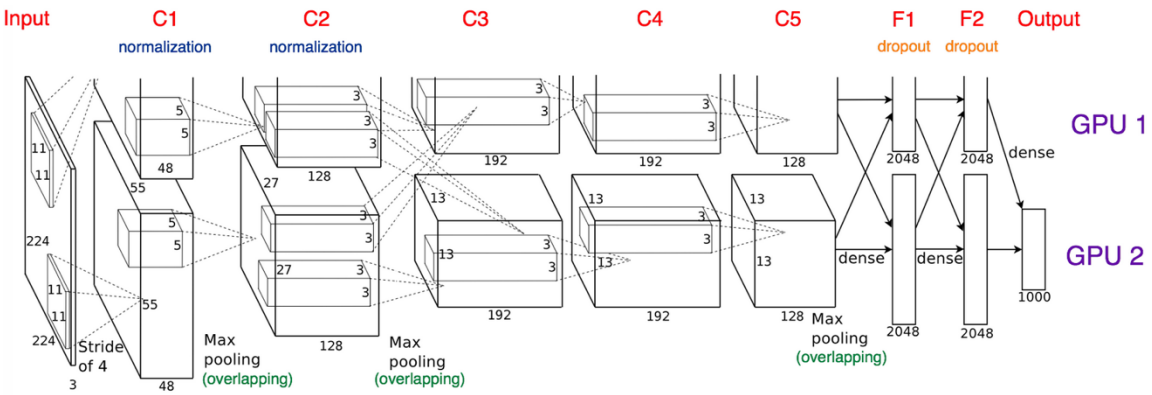

Figure 7: AlexNet CNN architecture [8].

Although AlexNet is pre-trained with images, classification of unseen classes is possible via transfer learning, replacing the last three layers of the AlexNet architecture with a custom fully connected layer, softmax layer, and output layer [9]. Fine tuning a pre-trained network is usually much faster and simpler to implement than training a classifier from scratch and I used the MATLAB function called “trainNetwork” to apply the AlexNet to the training data. In the replaced layers I used an initial learning rate of 0.0001, which I decreased by a factor of 10% after every epoch and the stochastic gradient descent with momentum (“SGDM”) optimisation function. This combination of parameters achieved the best classification accuracy during my hyperparameter search, achieving classification accuracy of 97.9% after two training epochs, as shown in Figure 8.

Figure 8: Accuracy (top) and loss (bottom) achieved by AlexNet during training.

4. Optical Character Recognition

4.1. Function Overview

detectNum takes the image (.jpg) or film (.mov) path string ("FileName") as an argument and returns an N x 1 matrix, containing a recognised number for each "N" face detected in the image. For example, if a video of two people is supplied, and one person is holding an A4 sheet with printed "04" printed on it and another with "37", detectNum will return [4, 37]. The numbers returned from input video files reflect the mode of the class predictions made from every fifth extracted frame.

4.2. Approach

Before attempting character recognition, I carried out a number of image processing, "digit extraction" steps, including normalising image brightness and colour, removing digital noise, detecting the white paper region of the image and centring and cropping the digit area. I then tested two methods of digit classification, first using MATLAB's optical character recognition function "ocr" and second by applying an SVM classifier trained on the HOG features of characters in the "English Typed Alphabet and Numbers Dataset" available at kaggle.com [10]. These approaches are discussed in detail in section 4.4.

4.3. Digit Extraction

When the function is applied it firstly pre-processes the supplied image then produces a mask over areas where digits are most likely to be found. When implementing this I attempted a number of different techniques and found the following image pre-processing method to be most effective. I converted images to greyscale then carried out thresholding, where pixel intensities were between 10 and 180, to selected regions of interest. To remove noise, I supplied a flat structuring element with a neighbourhood of five pixels, to the "imclose" function which performs a dilation followed by an erosion. The mask generated is shown in Figure 9 and shows that the likely digit area (white) is well defined, despite additional false-positive areas.

Secondly, after the mask is created, the function supplies the corresponding areas to the "detectMSERFeatures" function which returns all maximally stable external regions ("MSERs"), which are potential digits. By manually inspecting the properties of 50 correctly identified digit areas and 50 non-digit areas, I developed the following inclusive filtering conditions for detected MSER regions: a perimeter of 100 or greater, a maximum intensity less than one, a convex area 400 or greater, and an absolute orientation of between 75 and 900 inclusive. I also ignored all regions under a pixel size of 50. I made small amendments to these values for when video files are supplied to the function as the extracted frames are slightly smaller than supplied image files. Finally, the function uses another dilation and erosion to clean any remaining noise from the digit area and resizes it 227 x 227 pixels. An example of a clean, extracted digit is shown in Figure 9.

Figure 9: Left: raw image supplied to detectNum function. Centre: mask generated by image pre-processing. Left: cleaned, extracted digit image to be passed to the classifier.

In some cases, most commonly when a shadow was cast on the white paper, the thresholding stage of image pre-processing resulted in incorrect application of the mask, as seen in Figure 10. Although I made many attempts to rectify this, each successive failed case was slightly different from the last and most changes to the pre-processing method affected the classification of previously correctly classified instances. The final method proposed in this section is the one that facilitated correct digit recognition in the greatest number of sample images.

Figure 10: Left: raw image supplied to detectNum function, with low illumination on the digit area. Right: mask generated by image pre-processing which incorrectly conflates the printed digits with the paper background.

4.4. Digit Classification

The detectNum function uses MATLAB's in-built optical character recognition function to return characters from the supplied digit image. During final processing I use a regular expression to ensure only alphanumeric characters are returned. In some cases, numeric characters were being misclassified as letters, for example "0" as "O", "1" as "l" or "I" and "5" as "S" and I convert these if detected.

In addition to optical character recognition, I also trained an SVM classifier on the HOG features of characters in the "English Typed Alphabet and Numbers Dataset". This dataset contains approximately 1,000 black and white, synthetic images of numbers from 0 to 9, similar to those printed on the A4 sheets of paper. I chose not to use this trained classifier in the function as, by inspection, classification accuracy was considerably poorer and would require considerable model tuning to improve.

5. Reflection and Further Work

In this report I have discussed a number of techniques and approaches which are popular in the field of computer vision and produced successful, if highly domain-specific, solutions to the problems of face recognition and numeric character extraction. I was pleased to learn that two simple classifiers that I trained and implemented myself, SVM with SURF features and SVM with HOG features, could outperform the sophisticated AlexNet CNN, and that I was able to overcome a number of challenges in image pre-processing which are relevant to real-world scenarios. I have learned that it is much simpler to standardise images during photography, for example with constant illumination, than trying to correct this during pre-processing, however, good results are still achievable if creative image cleaning techniques are applied and extensive model tuning and validation is performed. In future work I would like to explore a single shot "all-in-one" approach, where subject detection and recognition are combined into one task. Example architectures have been suggested by Redmond, J., et al. with their "you only look once" approach [11] and Liu, W., et al. with their "single shot multibox detector" [12]. Additionally, I would be keen to employ other CNN architectures such as ResNet and GoogleNet, as well as testing more sophisticated ways to detect the white paper digit regions, such as Suzuki's "border following" algorithms [12], and Laurentini's visual-hull technique [13].

6. References

[1] Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features." European conference on computer vision. Springer, Berlin, Heidelberg, 2006.

[2] Mordvintsev, Alexander, and Abid K. “Introduction to SURF (Speeded-Up Robust Features).” *OpenCV*, 2013, opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html (Accessed 22/04/19).

[3] “bagOfFeatures.” *Bag of Visual Words Object – MATLAB – MathWorks United Kingdom*, uk.mathworks.com/help/vision/ref/bagoffeatures-class.html (Accessed 22/04/19).

[4] Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." international Conference on computer vision & Pattern Recognition (CVPR'05). Vol. 1. IEEE Computer Society, 2005.

[5] “ExtractHOGFeatures.” *Extract Histogram of Oriented Gradients (HOG) Features - MATLAB ExtractHOGFeatures - MathWorks United Kingdom*, uk.mathworks.com/help/vision/ref/extracthogfeatures.html (Accessed 22/04/19).

[6] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[7] Gao, Hao. “A Walk-through of AlexNet.” *Medium*, Medium, 2017, medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637. (Accessed 23/04/19].

[8] Pechyonkin, Max. “Key Deep Learning Architectures: AlexNet.” *Medium*, Medium, 2018, medium.com/@pechyonkin/key-deep-learning-architectures-alexnet-30bf607595f1 (Accessed 23/04/19).

[9] “Deep Network Designer.” *Pretrained AlexNet Convolutional Neural Network - MATLAB AlexNet - MathWorks United Kingdom*, uk.mathworks.com/help/nnet/ref/alexnet.html (Accessed 23/04/19).

[10] Swarna, Esther. “English Typed Alphabet and Numbers Dataset.” *Kaggle*, 2018, www.kaggle.com/passionoflife/english-typed-alphabets-and-numbers-dataset#EnglishFnt.tgz (Accessed 13/04/19).

[11] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[12] Liu, Wei, et al. "Ssd: Single shot multibox detector." European conference on computer vision. Springer, Cham, 2016.

[13] Suzuki, Satoshi. "Topological structural analysis of digitized binary images by border following." Computer vision, graphics, and image processing 30.1 (1985): 32-46.

[14] Laurentini, Aldo. "The visual hull concept for silhouette-based image understanding." IEEE Transactions on pattern analysis and machine intelligence 16.2 (1994): 150-162.