TDT4173: Machine Learning and Case-Based Reasoning

Assignment 5

Amar Jaiswal

Department of Computer Science
Norwegian University of Science and Technology (NTNU)

March 24, 2019

- **Delivery deadline: May 03, 2019** by 23:59.
- **This assignment counts towards 6 % of your final grade.**
- You can work on your own or in groups (max up to three person). Even if you are working in a group, make sure to deliver separately. Don't forget to mention your and your group member's **name and username** on the top of the report.
- Deliver your solution on *Blackboard* before the deadline.
- Please upload your report as a **separate PDF file**, and package your code into an archive (e.g. zip, rar, tar).
- The programming tasks may be completed in the programming language of your choice.
- Your code is part of your delivery, thus make sure your code should be well-documented and as readable as possible. The first section of the report *must* contain an explanation on how to run your code (don't expect evaluators of your code to configure any environment variables. All the paths (file path, resource path, etc.) should be relative to your code.

Objective: Gain experience with how to set up a pipeline for a non-trivial machine learning problem.

1 Optical Character Recognition

Optical character recognition, or OCR, is an old field of research in computer science that concerns the electronic, or mechanical, process of transforming images of characters into digital text. In this assignment, you will get the opportunity to implement such a system using machine learning. An explicit listing of the deliverables can be found in 2. What follows is a brief explanation of the dataset and the three main components of a run-of-the-mill OCR system.

Dataset: Chars74K Lite

We will be using a modified version of the Chars74K dataset by de Campos et al. from 2009 [1, 2]. The complete dataset includes a range of symbols in both English and Kannada, however, the modified version we will be using only encompasses majuscule and minuscule letters from the English alphabet, i.e. *A-Za-z*. Please consider visiting the dataset website if you want to know more about the Chars74K dataset.

Below is a summary of the modified dataset:

- 26 classes (*a-z*).
- 7112 images of characters stored as 8-bit greyscale images with JPEG compression.

- The size of each image is 20×20, which means that each image, or feature vector, contains 400 features (or pixels).
- The dataset does not differentiate between upper- and lower-case letters, e.g., both $\boldsymbol{f}$ and $\boldsymbol{F}$ belong to the same class.
- Classes are not explicitly labelled anywhere other than by the containing directory. For example, all images of the $\boldsymbol{a}$ or $\boldsymbol{A}$ symbol reside in the directory named "*a*".
- The dataset has not been split up into a training and test set. As such, this is something you will have to do by yourself.

Please refer back to this overview if you are unsure that you have loaded the data correctly. A good set of sanity checks you can do is: (i) count up the number of images, (ii) print out one of the images to see that the intensity values lie in the 8-bit, or normalised[1], range, and (iii) visualise a couple of images using a plotting library. 16 images randomly sampled from the modified Chars74K dataset can be seen in Figure 1.

Figure 1: 16 random images sampled from the modified version of the Chars74K dataset. Each image is a 20×20 8-bit greyscale image.

Feature Engineering

In many machine learning problems we start with a set of data, often gathered by a sensor, that lie in a high dimensional space. Rarely, is every dimension of a set of data equally useful, thus the first thing we have to do is to engineer *good* features.

For classification problems, we are first and foremost interested in engineering features that, when varied, allow us to discriminate between classes. For example, when trying to recognise whether an image *contains* a person, we are not interested in the rotation or scale of that person. In this case, a *good* feature vector will contain features that do *not* vary (much) with respect to the distance or angle of a person; however, the features should vary a lot depending on whether a person is present in the image.

Given Occam's razor we can assume that there exists a low dimensional space that allow us to discriminate between our data perfectly. The features that make up this space are the ones we are most interested in, alas, these can be very difficult to find. The modified Chars74K dataset contains exactly 400 greyscale intensity values, or features, per image. This is a large space of possible configurations of pixel values, and while we will probably not find the *ideal* set of features, we may be able to improve the ones we have and/or reduce their number.

Features are often engineered by domain experts, however, automatic procedures, such as unsupervised learning, do exist and are heavily used. Below are a series of topics and techniques related to feature engineering that may prove useful for the assignment:

- Manually cropping, or otherwise removing parts of the image that are uninteresting.
- Feature scaling, such as ensuring that features (i) reside in the 0 to 1 range, have (ii) zero mean, and (iii) unit variance.

[1]It is common to transform intensity values in a set of images to values between 0 and 1 before any processing takes place. Assuming we have an 8-bit image, with intensity values between 0 and 255, all we have to do is change the data type to `float` and divide by 255.

- Dimensionality reduction, such as projecting high dimensional data to the top n principal components, using principal component analysis (PCA) [3, 4]. Other alternatives include, for example, summarisation with clustering (e.g. K-means) and learnable data encodings with autoencoders [5, 6, 7].
- Digital image processing techniques, such as noise removal or edge detection filters.
- Feature descriptors, such as the scale-invariant feature transform (SIFT) [8, 9], histogram of oriented gradients (HOG) [10], and local binary patterns (LBP) [11]. These are algorithms that create feature vectors with *good* properties.

Classification

The main goal of the assignment will be to train a model that assigns a class (a-z) according to a 20×20 distribution of pixel intensities. Seeing as the classes are discrete, it follows that we can do a process called classification. A popular approach to create a classifier is to model the mapping from the inputs $\mathbf{x}$ to the output class y. This can be done by fitting a model of the form $\Pr(y|\mathbf{x})$ and is called a *discriminative* classifier. The relationships defining the possible mappings between the inputs and output are determined by the parameters of the model. The symbol θ is usually used as a catch-all term for these parameters[2]. Intuitively, this means that calculating the probability of a new datum is done by computing the posterior $\Pr(y|\mathbf{x}, \theta)$.

In this assignment there are 26 classes, meaning that we have a multiclass classification problem. The classes are the 26 letters of the English alphabet. Instead of using alphabetical letters (a-z) it is helpful to let our target labels be numerical ($0 - 25$), e.g., $\boldsymbol{a}$ corresponds to 0, while $\boldsymbol{z}$ corresponds to 25. A number of machine learning models can be used for this assignment, such as logistic regression, support vector machines, k-nearest neighbour, decision tree learning, naive Bayes, artificial neural networks, and various ensemble approaches. Several of these models do not care about the spatial structure in images. If you are using a model such as this, e.g. logistic regression, it is often assumed that all images are reshaped, flattened, or unrolled to regular 1-d feature vectors[3]. Please investigate how this can be achieved in the programming language you have elected to use.

Detection

Object detection or object localisation, is a difficult problem where we attempt to find objects within a scene. In this assignment we will be doing a special case of object detection called character detection. The concept of finding the exact position of a character in an image seems like a daunting task, however, with the help of a character classifier (see above) and a technique called a *sliding window detector* we may be able to overcome it. The basic idea is to divide the incoming image, e.g., a document of handwritten or printed text, into a slew of overlapping patches at different locations, and then classify each patch based on what kind of character it may contain. With each patch being associated with a probability estimate, per class, produced by a classifier, we can use a threshold to localise patches that are likely to contain a letter.

In practice, a sliding window detector can be implemented by looping over an image[4] with a certain stride, and for each (x, y) pixel coordinate extract a window of nearby pixels, using a particular window size. In our case it is natural to let the window size be 20×20 pixels, however, in order to detect characters at multiple scales it is important to have different window sizes. Keep in mind that with a classifier with a fixed-sized input constraint, the window of pixels must be resampled to 20×20 pixels before applying the classifier.

As previously mentioned, character detection is a difficult task and there are many issues that must be considered when creating an OCR system. Common issues include overlapping patches that are difficult to prune away using a simple threshold, multiple character classes with high probability for a single patch, scale and orientation issues, and how to interpret background patches that should not be assigned to any class.

[2] Some models like artificial neural networks and several types of linear models use *weights* to indicate trainable parameters.

[3] Using the raw input images, a flattened image from the assignment dataset will have 400 elements.

[4] By looping over an image we mean setting up a nested loop that iterate over each pixel in scanline.

Machine Learning Libraries

Implementing most of the machine learning techniques mentioned above from scratch is outside the scope of this assignment. Rather, you should use a general-purpose machine learning library. In other words, take advantage of what is already out there instead of reinventing the wheel. Getting more experience with reading documentation is a core part of this assignment.

Below is a selection of four common general-purpose machine learning libraries using four different programming languages. Click on the bolded name of the library to visit the library webpage.

- **JuliaStats**[5] A family of statistics and machine learning packages in *Julia*.
- **Machine Learning Toolbox**[6] A suit of machine learning tools in the *MATLAB* environment. Can be combined with the Computer Vision System Toolbox to build systems using, for example, feature descriptors.
- **scikit-learn**[7] [12] The most popular general-purpose library for machine learning in *Python*. Has tools for just about anything related to machine learning. However, it does not support artificial neural networks very well. We strongly recommend this library to those of you that are unsure about which library to use.
- **Weka**[8] [13] A popular collection of machine learning algorithms for data mining tasks in *Java*. Contains tools for data pre-processing, classification, regression, clustering, association rules, and visualisation.

In addition to the libraries listed above, there are an abundance of specialised libraries for all sorts of machine learning techniques. Some notable examples for deep learning are Theano, MXNet, Torch, PyTorch, Caffe, and Tensorflow [14, 15, 16, 17, 18, 19, 20, 21]. We do not recommend using deep learning unless you are either already well-versed in one of these libraries or are heavily invested in the field.

2 Deliverables [6 points]

Your deliverables must include the following:

- Source code for all aspects of your OCR system.
- Report of approximately 4 pages (not including references).
- Any additional material, e.g. detection images.

The assignment is split up into five main parts: (i) introduction, (ii) feature engineering, (iii) character classification, (iv) character detection, and (v) conclusion. The parts are ordered by how they should appear answered in your report. The following few sections outline what you must do for each part. Please read all of the tasks before starting on the assignment.

Introduction [0.5 points]

a) A brief overview of the whole system. This includes a short explanation on how to run it as well as what libraries your system is using.

Feature Engineering [1 point]

This part requires you to try at least *two* different ways of pre-processing the data, i.e. *feature engineering*. Which techniques to use is completely up to you, however, you must be able to both justify and explain the techniques you selected in the report. Take a look at the list provided earlier in this document for inspiration.

[5] https://juliastats.github.io/
[6] http://mathworks.com/solutions/machine-learning/
[7] http://scikit-learn.org/stable/
[8] http://www.cs.waikato.ac.nz/ml/weka/

b) Explain how each of the feature engineering techniques you selected work. Why did you select these techniques? Justify your answer.

c) Were there any techniques that you wanted to try, but for some reason were not able to? Explain.

Character Classification [2.5 points]

Character classification is the main part of your OCR system. You are required to try at least *two* different machine learning models for classifying the dataset according to the 26 labels. You can use any model as long as it can be used for classification, you are able to explain it in the report, and can justify your model selection. Have a look at the paper by de Campos et al. to see what the baseline performance might look like on the Chars74K dataset [2].

As mentioned, the two models you elect to use must be *different*. This means that both models cannot be of the same type with different hyperparameters. For example, both models can not be artificial neural networks (ANNs). One can, for example, be an ANN, while another might be a support vector machine, random forest, or a k-nearest neighbour classifier etc.

With 80 % of the dataset in the training set and 20 % in the test set, your *best* performing model *must* have 80 % or higher top-1 accuracy on the *test* set, i.e. the most probable letter is the correct one around 80 % of the time over the test set.

The use of data augmentation[9] is encouraged, however, using pre-trained model parameters is **not** allowed.

Tip: Generating numerical labels for your dataset can easily be done during loading because each directory of images is associated with only one class. For example, when loading in all n_b images of the letter $\boldsymbol{b}$, create an array of size n_b containing all ones, where 1 is the class associated with the letter $\boldsymbol{b}$.

d) After having looked at the dataset, did you have any initial idea about what kind of model that might work well? Explain your reasoning.

e) Give a description of the *two* models you elected to use[10]. The description must include a brief explanation on how they work. Why did you select these models?

f) A critical evaluation of your two models. How are you measuring their performance? How did they do? Which model gave the best results? Include at least five predictions in the report (both good and bad).

g) Were there any additional models that you would have liked to try, but for some reason were not able to? Explain.

Character Detection [1.5 points]

In this part you are to use the *best* classifier you were able to learn in the previous part to do character detection, i.e. the final part of your OCR system. While there are different ways to implement detectors, we strongly recommend that you use the sliding window approach outlined in a previous section.

After you have looped through an image using your sliding window detector, you should have a list of probabilities, one for each letter, for every patch that the detector has looked at. The final output of your character detector should be an image of the input image with coloured boxes around the patches with the highest probability (selected by way of a threshold). This can be done quite easily by keeping track of the (x, y) pixel coordinate of the current window (when looping over the image) and also the pixel coordinate you get by adding the window size, i.e. $(x + size_x, y + size_y)$, where $size_x$ and $size_y$ are the horizontal and vertical window size, respectively.

You may want to experiment with simple extensions to your detector, e.g. different scales, but this is *not* required.

h) Test your character detector on `detection-1.jpg` and `detection-2.jpg` and show the result in the report. Feel free to find or create additional images to test your detector, if you are so inclined.

[9] Data augmentation is a set of techniques used to artifically increase the size of the dataset. This includes, for example, flipping images, adding small amounts of noise, or affine transformations.

[10] For example, say you have elected to use support vector machine (SVM) as one of your two models. This means you will have to explain what kind of model SVM is, what kernel you are using, and how it is trained.

i) Give an evaluation of your detection system. How does it perform?

j) Describe any improvements you made to your detector. Discuss how you can improve your system further.

Conclusion [0.5 points]

k) What is the weakest and strongest component of your OCR system (feature engineering, character classification, and character detection)? Explain your answer.

l) What went good/bad with the project? Any lessons learned?

Acknowledgment

This assignment is based on the original work of Aleksander Rognhaugen and Daniel Groos.

References

[1] The chars74k dataset - character recognition in natural images, 2012.

[2] Teo de Campos, Bodla Rakesh Babu, and Manik Varma. Character recognition in natural images. In *VISAPP (2)*, pages 273–280, 2009.

[3] Karl Pearson. On lines and planes of closest fit to system of points in space. philiosophical magazine, 2, 559-572, 1901.

[4] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[5] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[6] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 833–840, 2011.

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *stat*, 1050:10, 2014.

[8] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[9] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[10] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[11] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):971–987, 2002.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard
Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[14] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James
Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas
Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[15] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX, 2010.

[16] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[17] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[18] PyTorch core team. Pytorch, 2016.

[19] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey
Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[20] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur,
Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore,
Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[21] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.