Assignment 5 - OCR

By Johannes Kvamme (johannkv), Pål Larsen (paaledwl), Peter Rydberg (hmrydber)

Introduction

Language and tools

The programming language chosen was Python. Python was selected due to the fact that it is one of, if not the most, utilized programming language used in machine learning. The language contains many good libraries to help support implementations of these algorithms. Scikit-learn [*1*] (sklearn) was then chosen as the main library as it is one of the most known Python machine learning libraries for implementing the different classification algorithms (and more).

Used packages:

- **PIL**: Used for reading, loading, and writing to images.
- **Numpy**: Machine learning requires a lot of matrix calculations which numpy is incredibly good at.
- **Pickle**: Serializes the training models which lets us skip training the model every time the program is ran.
- **Scikit-image**: Another package by Scikit which is dedicated to pre-processing images. From Skimage it is for example simple to extract Histogram of Oriented Gradients (HOG) with the method `hog`.
- **pyttsx3**: Text-to-speech synthesizer. Which was added for fun.

Installation guide

Create a virtual environment using the command `python -m venv env`, then activate the environment. The packages themselves can be installed using the command `pip install -r requirements.txt` inside the `code`-folder.

Usage

When these are installed, run `python ocr.py`. The args and flags that can be sent to the program are can be found by running `-h` or check the `readme.md` in the project. The default runs ANN.

When running the program, it will first fetch a classifier, either by training a new one or fetching a cached, pre-trained classifier from Pickle. Then the Sliding Window technique is applied to a given image in order to detect present objects. The classifier is then used to detect which characters each object relates to, and prints the string (and reads it aloud if `--use-tts` is set). When finished, a `dump` folder is created with an output image which has drawn a square around the letters it found.

Feature Engineering

As mentioned in the assignment, there exists many ways of getting features from an image. The group decided that cropping the data by hand would lead to human error and unevenly shaped input data, and quickly found out empirically that the feature descriptor HOG worked well on the given dataset.

Histograms of Oriented Gradients (HOG)

The way that HOGs extract features from an image is similar to other methods, but HOGs create overlays of dense grids of cells and uses local contrast normalization (LCN), which normalizes the contrast of an image non-linearly, to increase accuracy.

In an article on HOGs [*2*], the authors explain how this feature descriptor method works better than SIFT as well as other methods, which is one reason the group decided on using HOG. After applying it by easily implementing it with `skimage`, there was an increase of approximately 10% accuracy of the F1-score (harmonic mean) on training the data. The group quickly decided that HOG was a feature descriptor the group would keep on using.

Inverted images

When training character recognition software, a lot of the training data contains a black background with white text and vice versa. By inverting the image colors, making black backgrounds white and white text black, then adding them to the training samples, the OCR can better recognize characters regardless of color. The given dataset for this assignment contains a lot of different images that consists of either black background with white text, or white background with black text. The group's hypothesis were that by inverting all images, it would be better at ignoring irrelevant features (the background) and only draw features from what is important (the edges of the text).

By adding inverted images to the training sample, the accuracy of the F1-score was increased by around 10%, the same as HOG, together adding about 20% on the training data for all of the models.

Other pre-processing methods

A couple of other methods were considered, but decided against in favor of HOGs and inverted images.
Scale-invariant feature transform (SIFT) was considered at first, but was dropped when reading the article about HOGs [*2*] and how HOGs were more effective than SIFTs.
Manually going through the images and cropping them were considered briefly, but was dropped due to the large size of the dataset, as well as worries about human error and input data size.
Local Binary Pattern (LBP) was also considered to be combined with the other processes, but was also dropped. This is due to the fact that after combining HOG and inverted images the group were satisfied with an F1-score between 96-98% and felt that adding LBP to the mix was not necessary.

Character Classification

Due to previous experience with assignment 3 in this course, the group's first thoughts on which models could be applicable were the algorithms Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), and K nearest neighbor (K-NN). The idea of using these models were then solidified after reading the article 'Character Recognition in Natural Images' [*3*], where the authors implement K-Nearest Neighbor and Support Vector Machines to recognize characters.

5 predictions were randomly created shown below. The model used was ANN. Bad results were rarely generated due to the fact that each model had an F1-score between 96-98%.

```
    Classifying: n and got ['n']
    Classifying: i and got ['i']
    Classifying: p and got ['p']
    Classifying: p and got ['p']
    Classifying: b and got ['b']
```

Chosen models

Image detection is a fundamental and exciting part of machine learning and as an exercise, fun to implement and try out. In this assignment, the group decided to implement three different algorithms. The algorithms chosen were K-NN, SVM, and ANN which `sklearn` has methods to help implement.

The model that performed best overall was ANN which is further explained later in a later section. To measure the performance of each trained model a report is generated. The report is generated with the help of the package `sklearn.metrics`'s `classification_report` which calculates the precision, recall, and F1-score of the model overall as well as for each letter (which is not included in the report, due to space limitation).

K-NN

K-NN is an algorithm that is quite simple to implement, understand. The fact that it often performs quite well compared to many other algorithms [*4*], makes it a quite good candidate for this task.

K-NN is an instance based learning algorithm, as it stores the features and labels from the training data. It does not generalize the data before it starts classifying. The model's tuning input is the amount of *k* neighbors one should search for. When classifying an image, K-NN starts assigning labels to the data based on the frequency of the training samples.

```
          precision    recall  f1-score   support
accuracy                           0.97      2845
macro avg      0.97      0.96      0.97      2845
weighted avg   0.97      0.97      0.97      2845
```

SVM / SVC

SVM is an old, tried and true, algorithm. When training the model, each example is separated into either of two categories. The data can be separated with a hyperplane and by using the kernel trick (adding another dimension) the data is made linearly separable. When classifying, the image is viewed and its features are used to categorize the location of the new input.

Since SVM does not generally work well with unsupervised learning, the group decided to use SVC (Support Vector Clustering) which works well with it and is also easily implemented with sklearn.

```
          precision    recall  f1-score   support
accuracy                           0.97      2845
macro avg      0.96      0.95      0.96      2845
weighted avg   0.97      0.97      0.97      2845
```

ANN

Artificial neural networks are usually more complex than both K-NN and SVMs and are often referenced as 'black box' machine learning algorithms. In the early 2010s, ANNs became incredibly good at different tasks, like image and pattern recognition [*5*]. ANNs, in short, are inspired by biological neural networks and uses sets (layers) of nodes, which are based on biological neurons. Each node from one layer is connected to each node of the next. The nodes are activated by a summation of the connection weights into it, and an addition of a bias, then passed into an activation function. This is propagated through the network, which eventually reaches the output layer (classification).

The model implemented in this task uses all training examples available. The amount of neurons in each layer is automatically calculated as a formula of `n_layers - 2`. The algorithm used is a stochastic gradient-based optimizer called "Adam", which works well on large datasets such as the characters in the provided dataset [*6*]. Other hyperparameters were attempted, but were generally set to their defaults.

```
          precision    recall  f1-score   support
accuracy                           0.98      2845
macro avg      0.97      0.97      0.97      2845
weighted avg   0.98      0.98      0.98      2845
```

Additional models

Other models were briefly considered, but for this task the group was satisfied with using three different models.

Character Detection

As a basis for finding the string in the supplied images, the best performing model was used (ANN). The results are described below.

Detection image 1 and 2

The output result for image one was:

```
Most probable single solution: stte
```

Due to the characters being unaligned vertically, reading the letters in the correct order is difficult using the Sliding Window method as it does not correct for such large variations in y-placement. That being said, the the technique found all letters in their respective boxes and classified them correctly. Further tweaks can be made to the string concatenation system by checking the coordinate for each chosen box and adding them based on vertical overlap based on box size. This might work for many examples, but in the case of the given example, 's' and 'e' do not actually overlap and might still end up as different lines. Too high of an overlap threshold might have unintended consequences for line breaks.

The output result for image two was:

```
Most probable single solution: machinelearningandcasebasedreasoning
```

In general, this was the result of using the detection algorithm with ANN classification on the given example. Some models would classify a *very* vague 'e' or 'g' wrongly, but the detection algorithm was mostly correct.

Additional images

Using a ransom-letter generator [*7*], a few more test images were produced for testing. These did not give good results, however, as the letter spacing were generally too close and the background boxes were shaped too differently (the sliding window would detect content where the main letter did not reside even if it found the boxes). These images are included in the dataset folder. In order to classify these letters correctly, tweaks could be made on the Sliding Window method when encountering new non-whitespace areas. For example, the window could move towards center mass and expand, allowing for more of the object to come into view and predict for each expansion. The window would then decide on which prediction it was most sure of, preventing cases where it would only predict the top of the object which only partially contained the letter.

Conclusion

The OCR system

The weakest part of the OCR system is that it is unable to locate letters with specially large variance in background sizes. The character detection is based on locating sections of non-whitespace areas, but might not actually find the letter within the object itself. It is therefore not as versatile for all types of images. The character classification, however, is superb, especially with models KNN and ANN where the accuracy generally is in the range of 97-98 percent.

Final words

We very much enjoyed working on this project. Although it was a lot of work, working with something more tangible and practical than just programming models by hand was a lot of fun. We learned how to implement and evaluate different kinds of models, as well as combine different Python packages to solve multiple problems (image classification *and* image detection). Conclusively, we made steady progress throughout the project and are very satisfied with our results.

References

[1]: https://scikit-learn.org/stable/index.html "Scikit Learn" - "Scikit"
[2]: https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf "Histograms of Oriented Gradients for Human Detection" - "Navneet Dalal and Bill Triggs"
[3]: http://personal.ee.surrey.ac.uk/Personal/T.Decampos/papers/decampos_etal_visapp2009.pdf "Character Recognition in Natural Images" - "Teofilo E. de Campos, Bodla Rakesh Babu, Manik Varma"
[4]: http://www.rroij.com/open-access/object-recognition-using-knearest-neighborsupported-by-eigen-value-generated-fromthe-features-of-an-image.php?aid=46808 "Object Recognition Using K-Nearest Neighbor Supported By Eigen Value Generated From the Features of an Image" - "Dr. R.Muralidharan"
[5]: https://web.archive.org/web/20180831075249/http://www.kurzweilai.net/how-bio-inspired-deep-learning-keeps-winning-competitions "How bio-inspired deep learning keeps winning competitions", - "Amara D. Angelica, Jürgen Schmidhuber"

[6]: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier "MLPClassifier", - "Scikit"
[7]: http://www.ransomizer.com/ "Ransomizer" - "Melvix"