TASK

Capstone Project I: Image Processing

Visit our website

Introduction

WELCOME TO THE IMAGE PROCESSING TASK!

For the first Capstone Project, we'll be building an image recognition classifier that accurately determines the house number displayed in images from Google Street

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with a code reviewer. You can also schedule a call or get support via email.

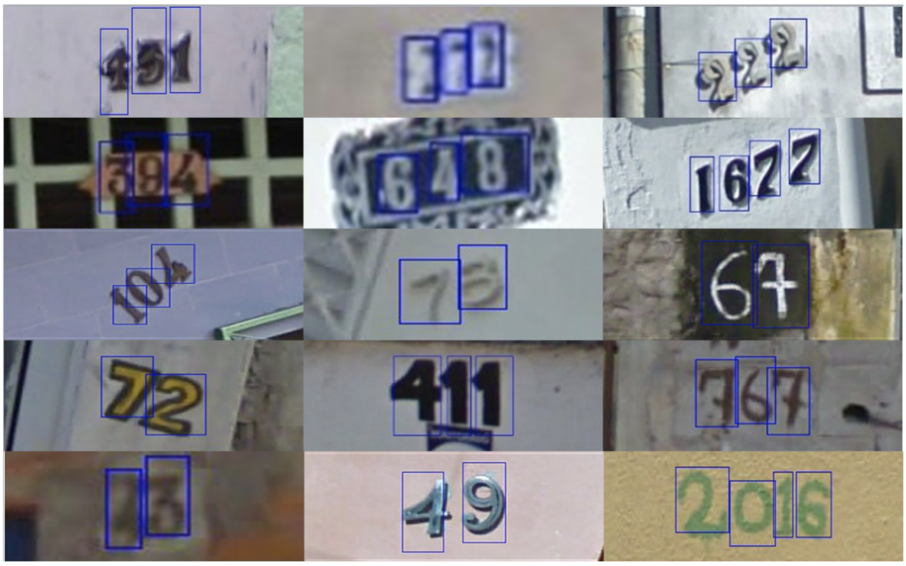
Our expert code reviewers are happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

IMAGE PROCESSING

Previously, if someone wanted to build a program to distinguish between an image of the number 1 and an image of the number 2, they would have set up a plethora of rules looking for e.g. straight lines vs curly lines, or a horizontal base vs a diagonal tip, and so forth. What machine learning allows us to do instead is feed an algorithm with many examples of images that have been labelled with the correct number. The algorithm then learns for itself which features of the image are distinguishing, and can make a prediction when faced with a new image.

The dataset

For this Capstone Project, we'll be using the House numbers dataset from Stanford University (**http://ufldl.stanford.edu/housenumbers**). It contains images of house numbers taken from Google Street View. Each one has been cropped to 32×32 pixels in size, focused on just the number.



Source: http://ufldl.stanford.edu/housenumbers

There are a total of 531131 images in this dataset, and we will load them in as one 4D-matrix of shape 32 x 32 x 3 x 531131. This represents each 32×32 image in RGB format (so the 3 red, green, blue colour channels) for each of our 531131 images. We'll be predicting the number shown in the image, from one of ten classes (0-9). Note that in this dataset the number 0 is represented by the label 10. The labels are stored in a 1D-matrix of shape 531131 x 1. You can check the dimensions of a matrix X at any time in your program using `X.shape`.

Take note:

Although this task focuses on just house numbers, the process we will be using can be applied to any kind of classification problem. Autonomous vehicles are a huge area of application for research in computer vision at the moment, and the self-driving cars being built will need to be able to interpret their camera feeds to determine traffic light colours, road signs, lane markings, and much more. With this in mind, at the end of the task, you can think about how to expand upon what you've developed here.

The setup

We need to download our dataset from **here** and save it in our working directory. This is a large dataset (1.3 GB in size). If you don't have enough space on your computer, try this **one** (182MB), but expect worse results due to the reduced amount of data. Create a new Jupyter notebook and name it `image_processing`.

Feature processing

Now let's begin! To understand the data we're using, we can start by loading and viewing the image files. First, we need to import three libraries:

```
import scipy.io

import numpy as np

import matplotlib.pyplot as plt
```

Then we can load the training dataset into a temporary variable `train_data`. The dictionary contains two variables X and y. X is our 4D-matrix of images, and y a 1D-matrix of the corresponding labels. So to access the i-th image in our dataset we would be looking for X[:,:,:,i], and its label would be y[i]. Let's do this for image 25.

```
# load our dataset
train_data = scipy.io.loadmat('extra_32x32.mat')

# extract the images and labels from the dictionary object
X = train_data['X']
y = train_data['y']
```

```
# view an image (e.g. 25) and print its corresponding label
img_index = 25
plt.imshow(X[:,:,:,img_index])
plt.show()
print(y[img_index])
```

Note that if you're working in a Jupyter notebook, you don't need to call `plt.show()`*. Instead, use the inline function (*`%matplotlib inline`*) just once when you import matplotlib.*

As you can see, we load up an image showing house number 3, and the console output from our printed label is also 3. You can change the index of the image (to any number between 0 and 531130) and check out different images and their labels if you like.

However, to use these images with a machine learning algorithm, we first need to vectorise them. This essentially involves stacking up the 3 dimensions of each image (the width x height x colour channels) to transform it into a 1D-matrix. This gives us our feature vector, although it's worth noting that this is not really a feature vector in the usual sense. Features usually refer to some kind of quantification of a specific trait of the image, not just the raw pixels. Raw pixels can be used successfully in machine learning algorithms, but this is typical with more complex models such as convolutional neural networks, which can learn specific features themselves within their network of layers.

We will be using a Random Forest approach with default hyperparameters. First, we import the necessary library and then define our classifier:

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier()
print(clf)
```

We can also print the classifier to the console to see the parameter settings used. Although we haven't changed any from their default settings, it's interesting to take a look at the options and you can experiment with tuning them.

```
RandomForestClassifier(bootstrap=True,class_weight=None,
criterion='gini',max_depth=None,max_features='auto',
max_leaf_nodes=None,min_impurity_split=1e-07, min_samples_leaf=1,
min_samples_split=2,min_weight_fraction_leaf=0.0,n_estimators=10,
```

```
n_jobs=1,oob_score=False,random_state=None,verbose=0, warm_start=False)
```

Training the model

We're now ready to train and test our data. But before we do that, we need to split our total collection of images into two sets – one for training and one for testing. Keeping the testing set completely separate from the training set is important because we need to be sure that the model will perform well in the real world. Once trained, it will have seen many example images of house numbers. We want to be sure that when presented with new images of numbers it hasn't seen before, that it has actually learnt something from the training and can generalise that knowledge – not just remember the exact images it has already seen.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state = 42 )

clf.fit(X_train, y_train)
```

Because of our large dataset, and depending on your machine, this will likely take a little while to run. If you want to speed things up, you can train on less data by reducing the size of the dataset. The fewer images you use, the faster the process will train, but it will also reduce the accuracy of the model.

Test results

Now we're ready to use our trained model to make predictions on new data:

```
from sklearn.metrics import accuracy_score

preds = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test,preds))
```

So our model has learnt how to classify house numbers from Google Street View with 76% accuracy simply by showing it a few hundred thousand examples. Given a baseline measure of 10% accuracy for random guessing, we've made significant progress. There's still a lot of room for improvement here, but it's a great result from a simple untuned learning algorithm on a real-world problem. You can even try going outside and creating a 32×32 image of your own house number to test on.

Compulsory Task 1

In this task, we will use the MNIST database, available from **this page**.

As stated by the creators of the dataset, "The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalised and centred in a fixed-size image."

Follow these steps:

- Load the MNIST dataset.
- Split the data into a training, development and test set.
- Choose two machine learning algorithms among the ones discussed in the previous Tasks, and explain why you chose them.
- For each model, pick one parameter to tune, and explain why you chose this parameter.
- Choose which value for the parameter to set for testing on the test data and explain why.
- Print confusion matrices for your two competitor models' predictions on the test set.
- Report which classes the models struggle with the most.
- Report the accuracy, precision, recall and f1-score.
- Comment on the differences in performance and report which model you believe did the best job.

Completed the task(s)?

Ask an expert to review your work!

Review work

Things to look out for:

1. Make sure that you have installed and set up all programs correctly. You have set up **Dropbox** correctly if you are reading this, but **Python or Notepad++** may not be installed correctly.
2. If you are not using Windows, please ask your mentor for alternative instructions.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.

Reference:

LeCun, Y., Cortes, C. and Burges, C.J.C. (2010). The MNIST Database of Handwritten Digits, Available from **http://yann.lecun.com/exdb/mnist/**