Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 10, Issue, 07, pp.71786-71791, July, 2018

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

ISSN: 0975-833X

RESEARCH ARTICLE

SPAM DETECTION IN IM IMAGES USING CONVOLUTIONAL NEURAL NETWORKS

***Ashutosh Mahesh Pednekar**

VIT University, Vellore – TN

ARTICLE INFO

Article History:
Received 28th April, 2018
Received in revised form
23rd May, 2018
Accepted 09th June, 2018
Published online 31st July, 2018

Key Words:
Conv-nets, Multi Class Classification,
Tensor Flow,
CUDA.

ABSTRACT

Spam detection in emails has been a standard classification problem since decades. But nowadays, emails are only a small part of our digital conversations. Instant messengers, for example, are a huge source for spam messages and images these days. This paper proposes an application of deep CNN's to help classify these images into three main classes: Important, Acceptable and Spam. Acceptable are those images which might be irritating, but are not necessarily Spam, e.g. Good Morning messages.

Citation: Ashutosh Mahesh Pednekar. 2018. "Spam Detection in IM Images using Convolutional Neural Networks*.", International Journal of Current Research, 10, (07),* 71786-71791

INTRODUCTION

Every morning, when we check our WhatsApp, we are usually bombarded with many messages. Some of them are from our loved ones, while others are of professional importance. But unfortunately, among these messages are many unwanted messages, like fake news, product promotions, etc. These messages disrupt our IM experience and also clog our phone's internal storage. I've often seen people spending a lot of time deleting these messages and sorting out the good ones. Wouldn't it be great if this tedious task be automated?. That is precisely what this paper intends tom illustrate. As mentioned earlier in this paper, classifying emails has been a standard classification problem for ears. But we are no longer limited to emails anymore. Spammers have many more ways to target us. The most popular being WhatsApp, or other instant messaging services. Also, the messages we receive today, are not just text messages, where we could use word count, or other such means to identify spams. These messages are often in the form of images, which makes the classification task even more challenging. I seek to solve this problem by leveraging the power of deep learning techniques, namely, convolutional neural networks.

These neural nets have to capacity to learn the hundreds of thousands of features, and almost intuitively make an intelligent guess of whether a given image is Spam or not. But some messages that might get misclassified as spam, may not necessarily be that bad. That's why I've considered a third class label, "Acceptable". Some examples of these acceptable messages may be the numerous "Good morning" messages that we may receive. We don't necessarily want to lose these. I plan to use optical character recognition to extract text from these messages, and show the users just this text, so that the users are not clogged with the images, yet they receive their messages. The one after explains how I went about implementing this Idea.

Objective: The main objective of this paper is to employ convolutional neural networks to classify IM images into three main classes, Important, Acceptable and Spam. This is an example of a multi class classification problem. I could I used traditional machine learning algorithms to do the job, but the sheer amount of data seems to show better promise in using deep learning approaches

Literature Survey: In their paper, Zhao proposed a very simple deep learning method for classifying images, based on cascaded PCA, binary hashing, and block-wise histogram decomposition. Laiming He and Jian gave a novel solution to mitigating the difficulty faced in training neural networks.

Corresponding author: **Ashutosh Mahesh Pednekar,**
VIT University, Vellore – TN.
DOI: https://doi.org/10.24941/ijcr.31649.07.2018

The presented a residual learning framework to ease the training networks that are way deeper than those preceding them. Multi-column DNN's have been a great inspiration for this application as it eased the process of training the network over image data, as it better leverages the multiple columns available in 2D data such as images. Dan Ciresanm Meier and Schmidhuber were the first ones to achieve near human level benchmarks. I referred their paper from the Cornell University Library to gain better insight on how Neural networks actually work. These insights were very useful. Alex, Ilya and Geoffrey trained large, Deep Conv-nets to classify millions of high-resolution images in the LSVRC ImageNet training set into thousands of classes. This was on a whole different scale than the scope of this paper. Their paper contained a new regularization method that greatly improved the efficiency of the network. I referred many other such journals, and I've sedulously cited all of them in the references section below.

Technologies involved - explanation

Compute Unified Device Architecture (CUDA): In today's world, parallel processing and scalable code is no longer a head turner, but it has evolved to become an absolute necessity. CUDA, or compute unified device architecture is a parallel programming platform from Nvidea. It allows users to parallelize programs, and run them on a GPU. Graphical processing units have very large potential. CUDA is a parallel programming platform and programming model developed by Nvidea corporation. It allows software developers and researchers to use a CUDA-enabled Graphic processors for general purpose computing – an approach termed as GPGPU. (General purpose GPU).

Tensor Flow: Tensor Flow is a system for Large-Scale Machine learning developed by the Google Brain team. Pioneered by Martin Abadi, and Paul Barham, it is a machine learning system that operates at large scale and in heterogeneous environments. It uses Computation Graphs to represent the computation. Tensor Flow can be run on CPU's, GPU's, Clusters and even on smartphones. It can sedulously leverage the power of Nvidea's CUDA framework. Unlike core python, Tensor Flow uses a lazy evaluation mode, where it first just creates the Directed Acyclic graph of the required calculation. Only when we instruct it to evaluate this graph, it does the actual computation. This helps to facilitate implicit parallelism, as different sub-graphs in the DAG can be run on independent processors. But for developmental purposes, this can be disables by using the 'eager' execution mode.

MATERIALS AND METHODS

In the context of this paper, I'm scraping sample images from the internet, and indexing them iteratively, at first, to create my desired dataset. Once the dataset was ready, then I wrote some Tensor-flow functions to create Conv-nets that I later used to create my Image Classification models. I split my data-set into test and train splits. That followed training this model over the dataset I had put together earlier. I used Nvidia's CUDA architecture for optimized performance. I then tested the results. Each of these steps are elaborated below: -

Data Acquisition: Every machine learning problem requires data. For this paper, I needed a labelled dataset of images that usually circulate on WhatsApp. Because of the immense fastidiousness of the data, I had to make my own datasets. For that, I needed to scrape some images from the internet. In the Implementation section that follows, I've illustrated this process. The next stem after data acquisition was to label this data.

Labelling: -Since I had collected the data manually, it was very simple to label them in batches. E.g., go to 'https://www.google.com/ images?/good morning messages' and scrape the first few images, and label all of those as 'Acceptable'. Likewise, I scraped the Spam and Important, respectively. For the important section, I scraped some images from my own Google Photos, as those would be the kind of images I would consider important. Also, since I used my own data, there are no privacy or copyright concerns in any way.

Model Creation and Training: The implementation section illustrates my code to create the
conv-nets and using them to train images. Then I trained these models over my dataset. As I mentioned earlier, I used Nvidia's CUDA platform and the awesome Tensor Flow Framework for the above mentioned tasks.

Implementation: This sections consists of the screen captures and illustrations of the actual implementation. The detailed workings are explained above.

Data Acquisition: The first phase was to acquire the necessary data. I scraped the necessary data using the urllib and Beautiful Soup libraries in python. Here's the code snippet for the same:

In order to scrape the training data, I created the following directory structure:

The below snapshot shows some 'good-morning' images being scraped into the "acceptable" folder. The program automatically renames the images for convenience. I scraped these images from www.sendscraps.com. These images were used for academic research only. The website contained multiple pages, so I ran the spider, or the crawler on each of these pages one after another. The resultant 'acceptable' dataset looked as follows: -

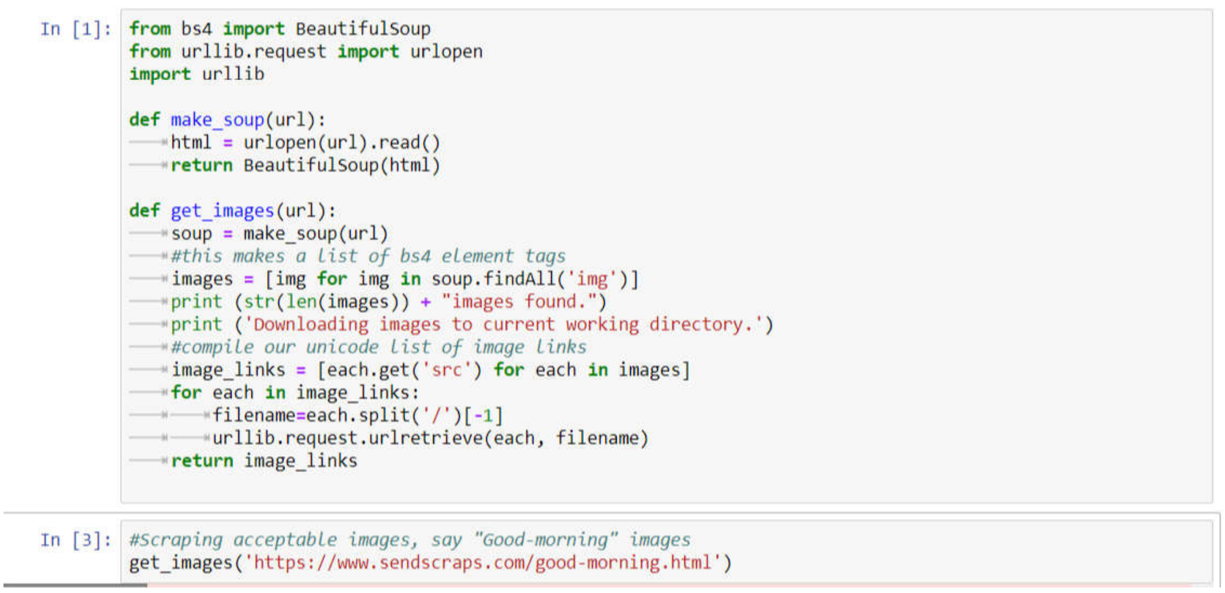
In a similar fashion, data for the other two classes was also acquired. The next step was to actually code the CNNs. This task, being more computationally expensive, needed to be carried out on a better platform. I had to switch to the cloud for this purpose. As mentioned earlier, I utilized GPU processing using the CUDA framework for faster processing. I transferred my dataset to Google Drive, as can be seen in the picture below: -

The following snaps picturise the configuration and hyper-parameter tuning: -

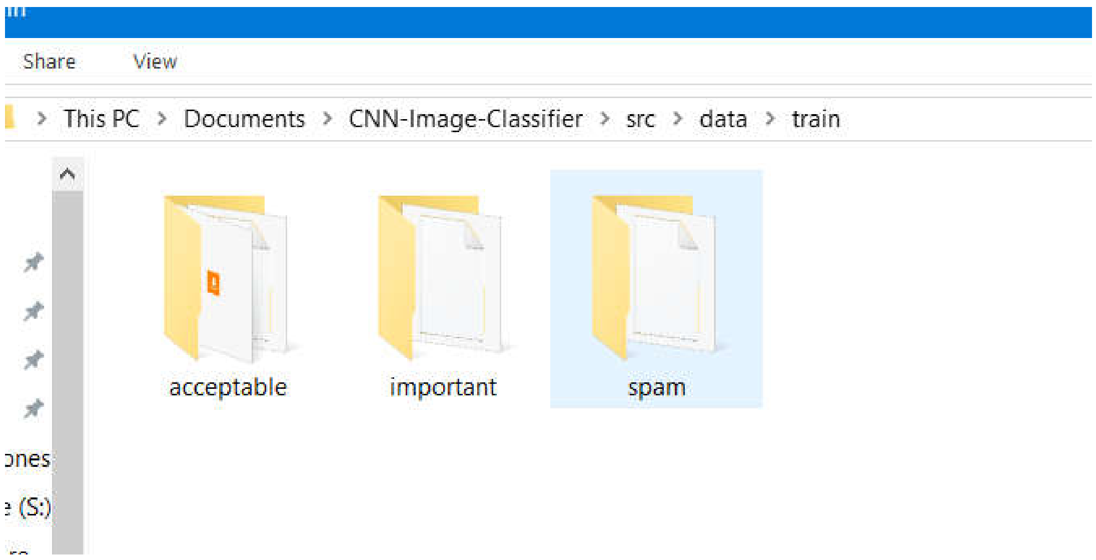
Loading the data to the notebook. This followed some helper functions. I've avoided adding their snaps as that would unnecessarily make the paper too long.

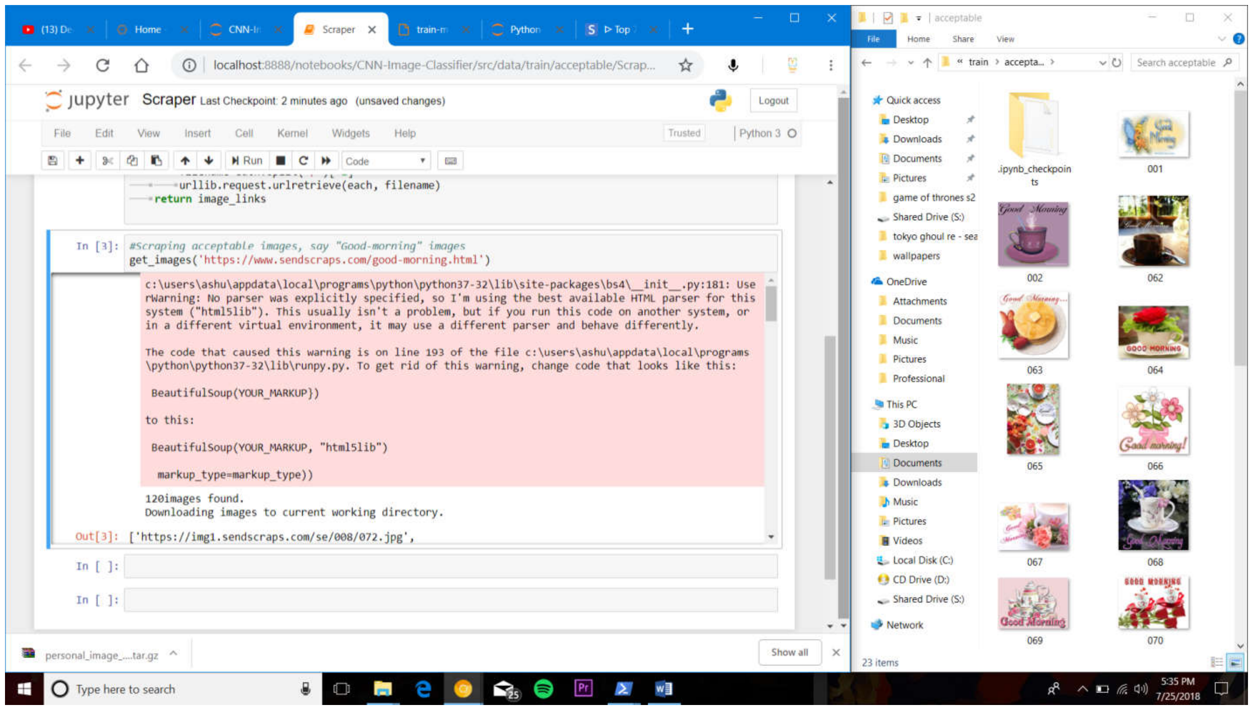
After training the network, and testing it with a sample image, I obtained the following results: -

We are now done using Tensor Flow, so we close the session to release its resources.

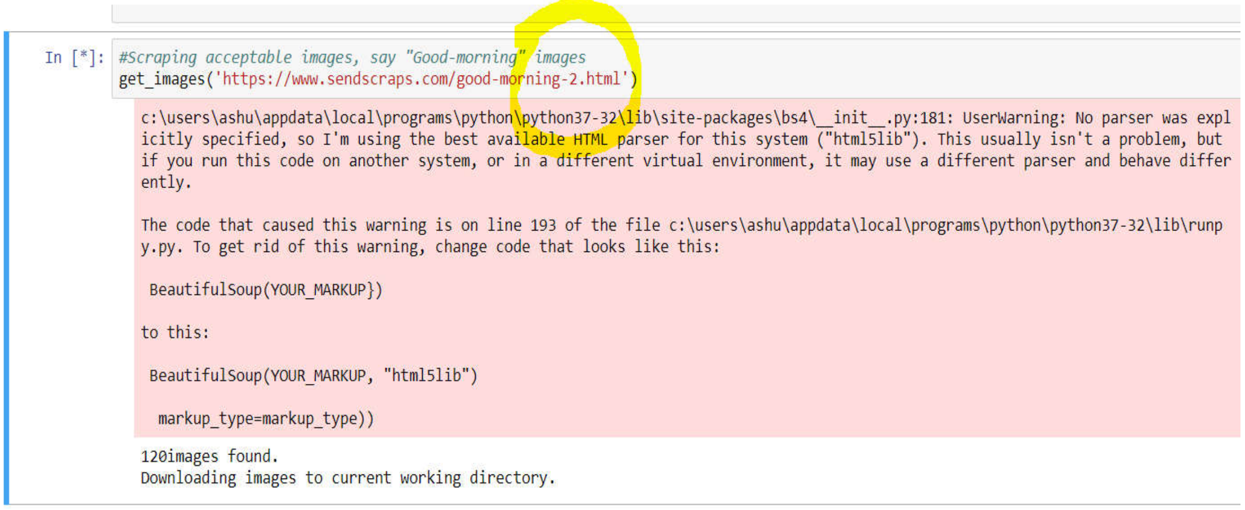


```
In [1]: from bs4 import BeautifulSoup
        from urllib.request import urlopen
        import urllib

        def make_soup(url):
            html = urlopen(url).read()
            return BeautifulSoup(html)

        def get_images(url):
            soup = make_soup(url)
            #this makes a list of bs4 element tags
            images = [img for img in soup.findAll('img')]
            print (str(len(images)) + "images found.")
            print ('Downloading images to current working directory.')
            #compile our unicode list of image links
            image_links = [each.get('src') for each in images]
            for each in image_links:
                filename=each.split('/')[-1]
                urllib.request.urlretrieve(each, filename)
            return image_links

In [3]: #Scraping acceptable images, say "Good-morning" images
        get_images('https://www.sendscraps.com/good-morning.html')
```

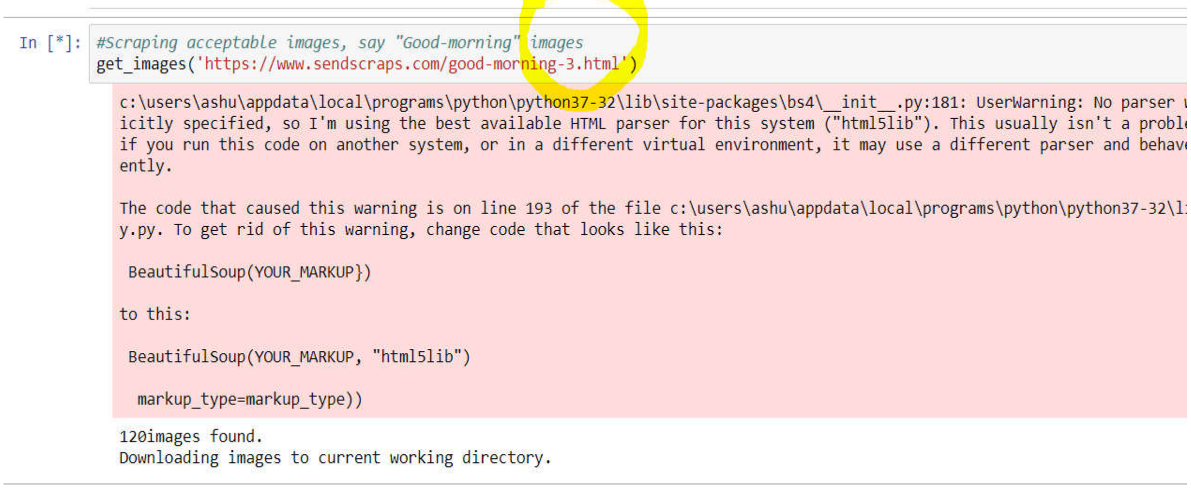

Share
View
This PC > Documents > CNN-Image-Classifier > src > data > train
acceptable
important
spam



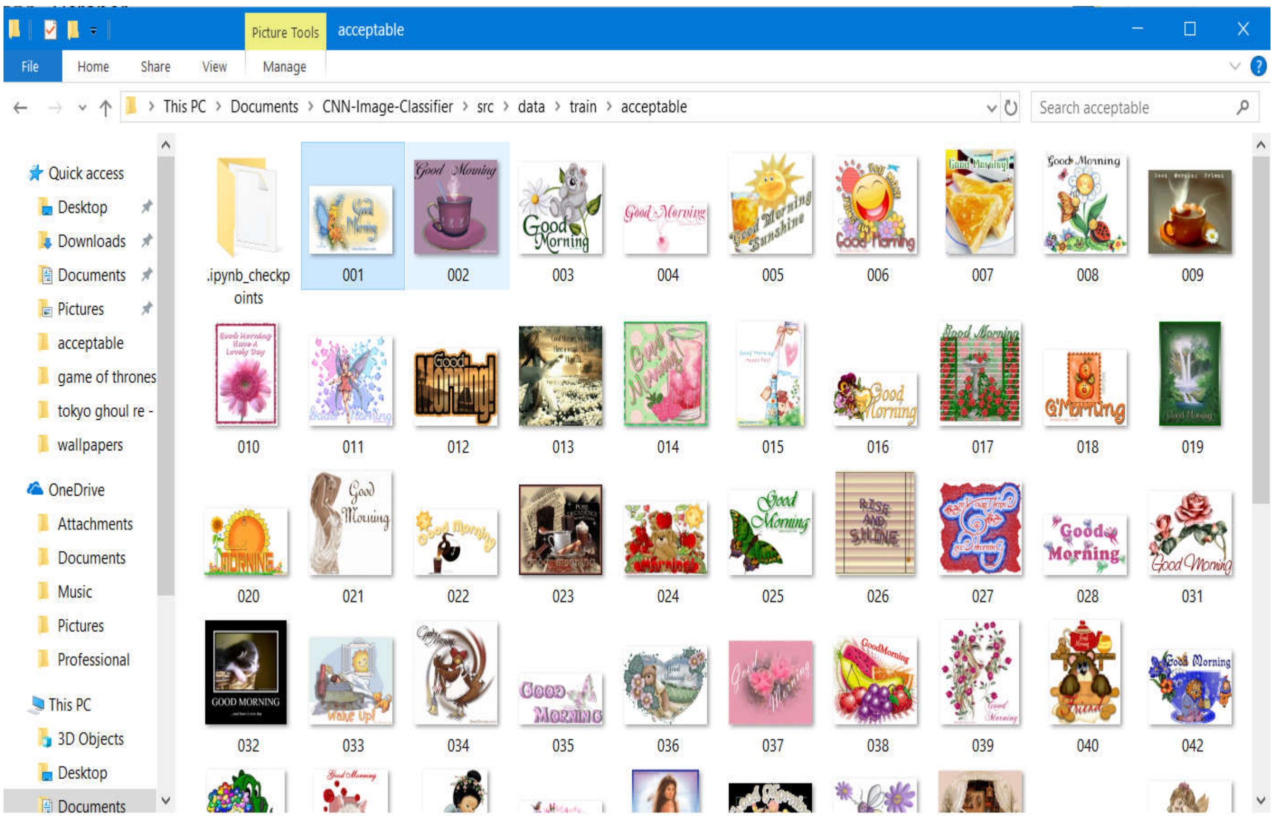
```
        ─────urllib.request.urlretrieve(each, filename)
        return image_links

In [3]: #Scraping acceptable images, say "Good-morning" images
        get_images('https://www.sendscraps.com/good-morning.html')

        c:\users\ashu\appdata\local\programs\python\python37-32\lib\site-packages\bs4\__init__.py:181: Use
        rWarning: No parser was explicitly specified, so I'm using the best available HTML parser for this
        system ("html5lib"). This usually isn't a problem, but if you run this code on another system, or
        in a different virtual environment, it may use a different parser and behave differently.

        The code that caused this warning is on line 193 of the file c:\users\ashu\appdata\local\programs
        \python\python37-32\lib\runpy.py. To get rid of this warning, change code that looks like this:

         BeautifulSoup(YOUR_MARKUP})

        to this:

         BeautifulSoup(YOUR_MARKUP, "html5lib")

          markup_type=markup_type))

        120images found.
        Downloading images to current working directory.

Out[3]: ['https://img1.sendscraps.com/se/008/072.jpg',

In [ ]:

In [ ]:
```

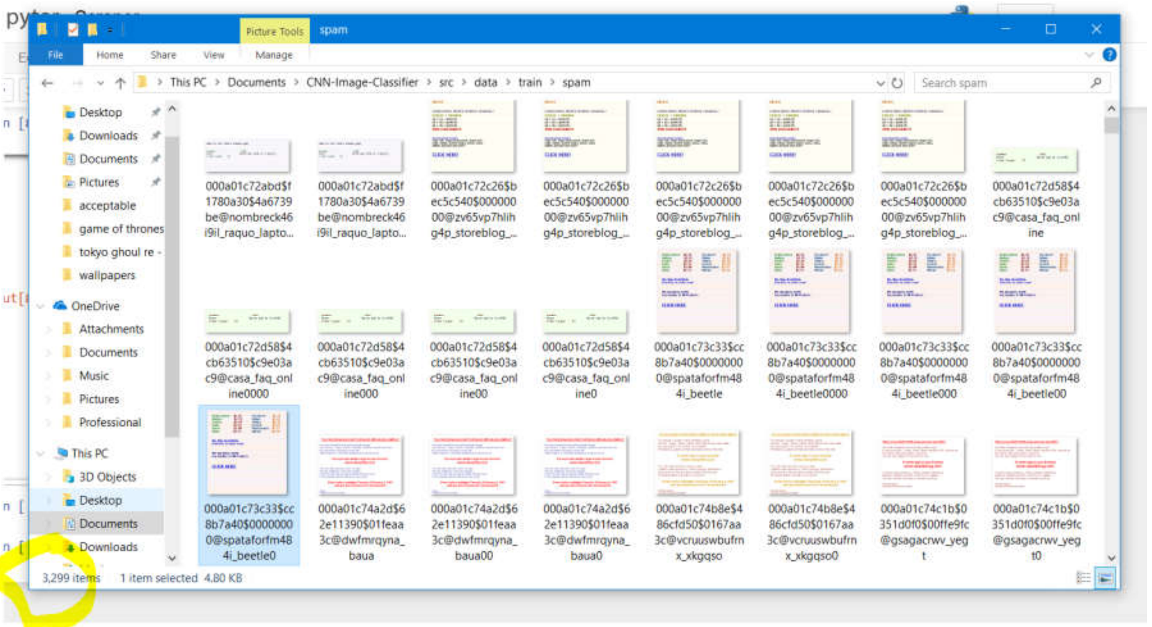


```
In [*]: #Scraping acceptable images, say "Good-morning" images
        get_images('https://www.sendscraps.com/good-morning-2.html')

   c:\users\ashu\appdata\local\programs\python\python37-32\lib\site-packages\bs4\__init__.py:181: UserWarning: No parser was expl
   icitly specified, so I'm using the best available HTML parser for this system ("html5lib"). This usually isn't a problem, but
   if you run this code on another system, or in a different virtual environment, it may use a different parser and behave differ
   ently.

   The code that caused this warning is on line 193 of the file c:\users\ashu\appdata\local\programs\python\python37-32\lib\runp
   y.py. To get rid of this warning, change code that looks like this:

    BeautifulSoup(YOUR_MARKUP})

   to this:

    BeautifulSoup(YOUR_MARKUP, "html5lib")

     markup_type=markup_type))

   120images found.
   Downloading images to current working directory.
```



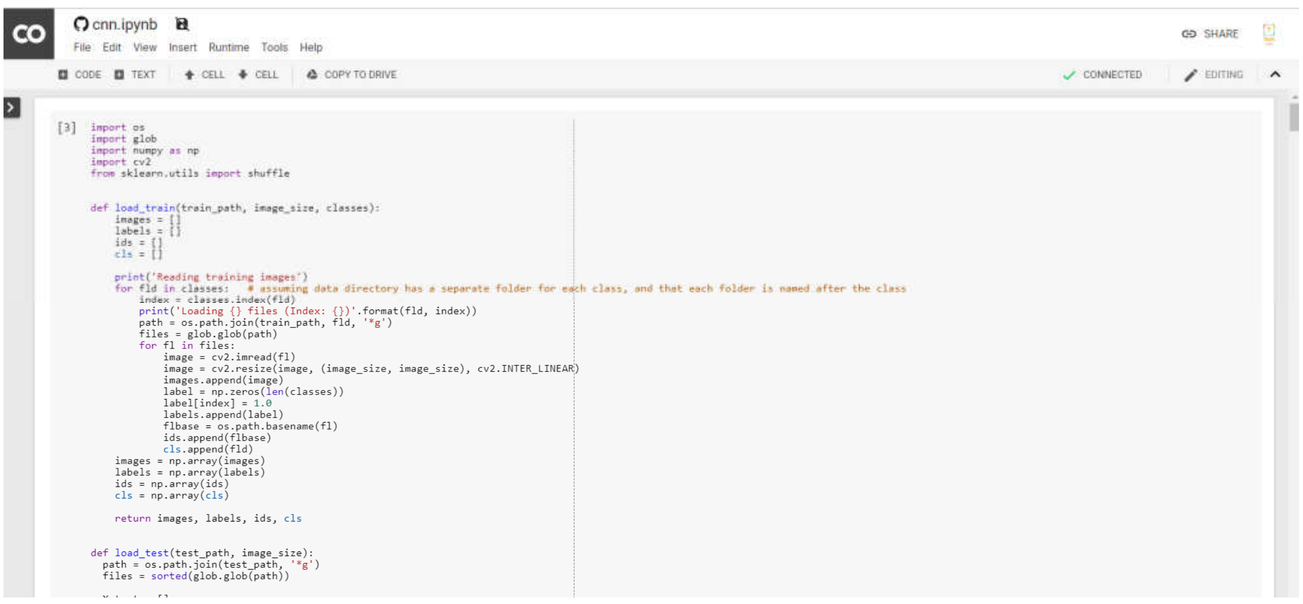
```
In [*]: #Scraping acceptable images, say "Good-morning" images
        get_images('https://www.sendscraps.com/good-morning-3.html')

   c:\users\ashu\appdata\local\programs\python\python37-32\lib\site-packages\bs4\__init__.py:181: UserWarning: No parser 
   icitly specified, so I'm using the best available HTML parser for this system ("html5lib"). This usually isn't a probl
   if you run this code on another system, or in a different virtual environment, it may use a different parser and behav
   ently.

   The code that caused this warning is on line 193 of the file c:\users\ashu\appdata\local\programs\python\python37-32\l
   y.py. To get rid of this warning, change code that looks like this:

    BeautifulSoup(YOUR_MARKUP})

   to this:

    BeautifulSoup(YOUR_MARKUP, "html5lib")

     markup_type=markup_type))

   120images found.
   Downloading images to current working directory.
```



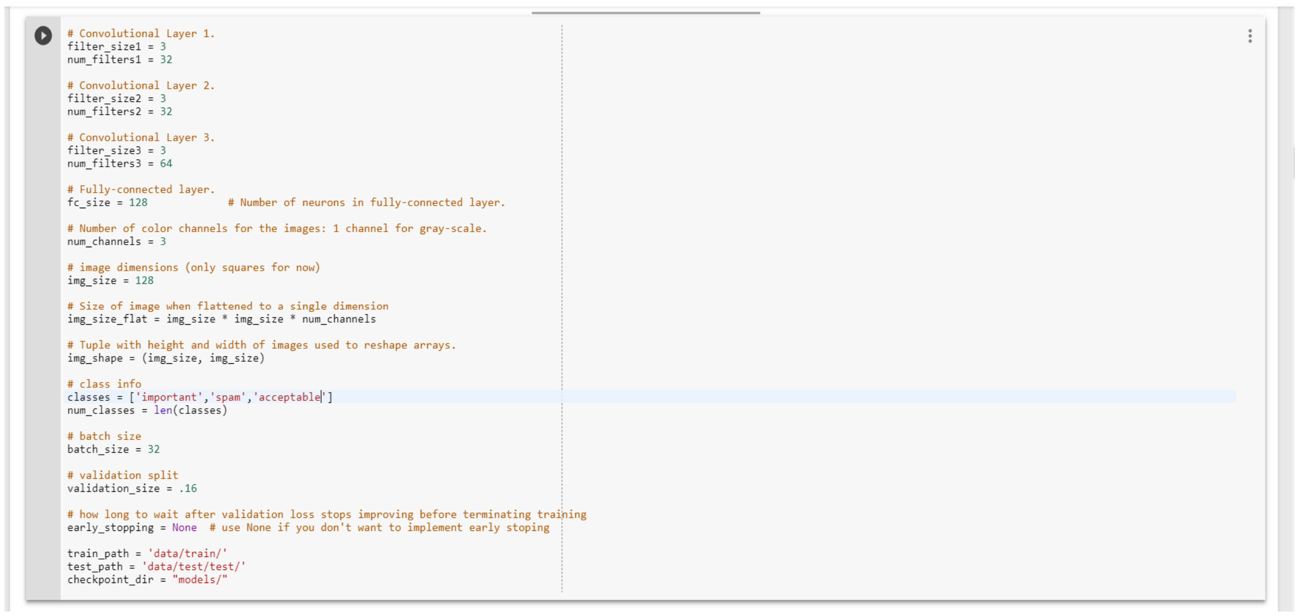


000a01c72abd$f1780a30$4a6739be@nombreck46i9il_raquo_lapto...
000a01c72abd$f1780a30$4a6739be@nombreck46i9il_raquo_lapto...
000a01c72c26$bec5c540$00000000@zv65vp7hlihg4p_storeblog_...
000a01c72c26$bec5c540$00000000@zv65vp7hlihg4p_storeblog_...
000a01c72c26$bec5c540$00000000@zv65vp7hlihg4p_storeblog_...
000a01c72c26$bec5c540$00000000@zv65vp7hlihg4p_storeblog_...
000a01c72c26$bec5c540$00000000@zv65vp7hlihg4p_storeblog_...
000a01c72d58$4cb63510$c9e03ac9@casa_faq_online
000a01c72d58$4cb63510$c9e03ac9@casa_faq_online0000
000a01c72d58$4cb63510$c9e03ac9@casa_faq_online000
000a01c72d58$4cb63510$c9e03ac9@casa_faq_online00
000a01c72d58$4cb63510$c9e03ac9@casa_faq_online0
000a01c73c33$cc8b7a40$00000000@spataforfm484i_beetle
000a01c73c33$cc8b7a40$00000000@spataforfm484i_beetle0000
000a01c73c33$cc8b7a40$00000000@spataforfm484i_beetle000
000a01c73c33$cc8b7a40$00000000@spataforfm484i_beetle00
000a01c73c33$cc8b7a40$00000000@spataforfm484i_beetle0
000a01c74a2d$62e11390$01feaa3c@dwfmrqyna_baua
000a01c74a2d$62e11390$01feaa3c@dwfmrqyna_baua00
000a01c74a2d$62e11390$01feaa3c@dwfmrqyna_baua0
000a01c74b8e$486cfd50$0167aa3c@vcruuswbufrnx_xkgqso
000a01c74b8e$486cfd50$0167aa3c@vcruuswbufrnx_xkgqso0
000a01c74c1b$0351d0f0$00ffe9fc@gsagacrwv_yegt
000a01c74c1b$0351d0f0$00ffe9fc@gsagacrwv_yegt0

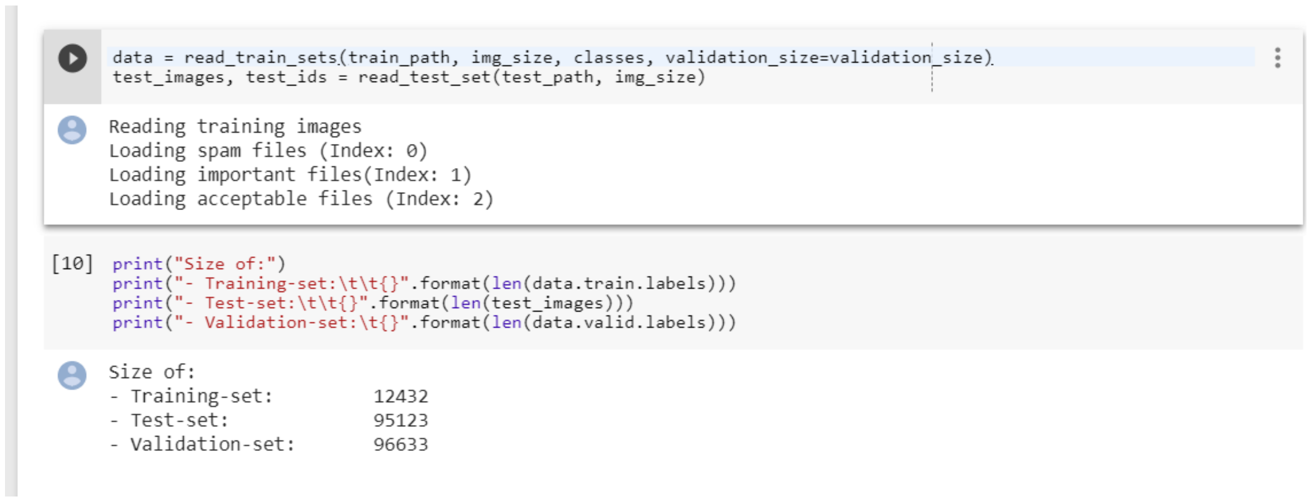
3,299 items 1 item selected 4.80 KB



```
[3]  import os
     import glob
     import numpy as np
     import cv2
     from sklearn.utils import shuffle


     def load_train(train_path, image_size, classes):
         images = []
         labels = []
         ids = []
         cls = []

         print('Reading training images')
         for fld in classes:   # assuming data directory has a separate folder for each class, and that each folder is named after the class
             index = classes.index(fld)
             print('Loading {} files (Index: {})'.format(fld, index))
             path = os.path.join(train_path, fld, '*g')
             files = glob.glob(path)
             for fl in files:
                 image = cv2.imread(fl)
                 image = cv2.resize(image, (image_size, image_size), cv2.INTER_LINEAR)
                 images.append(image)
                 label = np.zeros(len(classes))
                 label[index] = 1.0
                 labels.append(label)
                 flbase = os.path.basename(fl)
                 ids.append(flbase)
                 cls.append(fld)
         images = np.array(images)
         labels = np.array(labels)
         ids = np.array(ids)
         cls = np.array(cls)

         return images, labels, ids, cls


     def load_test(test_path, image_size):
       path = os.path.join(test_path, '*g')
       files = sorted(glob.glob(path))
```



```
# Convolutional Layer 1.
filter_size1 = 3
num_filters1 = 32

# Convolutional Layer 2.
filter_size2 = 3
num_filters2 = 32

# Convolutional Layer 3.
filter_size3 = 3
num_filters3 = 64

# Fully-connected layer.
fc_size = 128             # Number of neurons in fully-connected layer.

# Number of color channels for the images: 1 channel for gray-scale.
num_channels = 3

# image dimensions (only squares for now)
img_size = 128

# Size of image when flattened to a single dimension
img_size_flat = img_size * img_size * num_channels

# Tuple with height and width of images used to reshape arrays.
img_shape = (img_size, img_size)

# class info
classes = ['important','spam','acceptable']
num_classes = len(classes)

# batch size
batch_size = 32

# validation split
validation_size = .16

# how long to wait after validation loss stops improving before terminating training
early_stopping = None  # use None if you don't want to implement early stoping

train_path = 'data/train/'
test_path = 'data/test/test/'
checkpoint_dir = "models/"
```

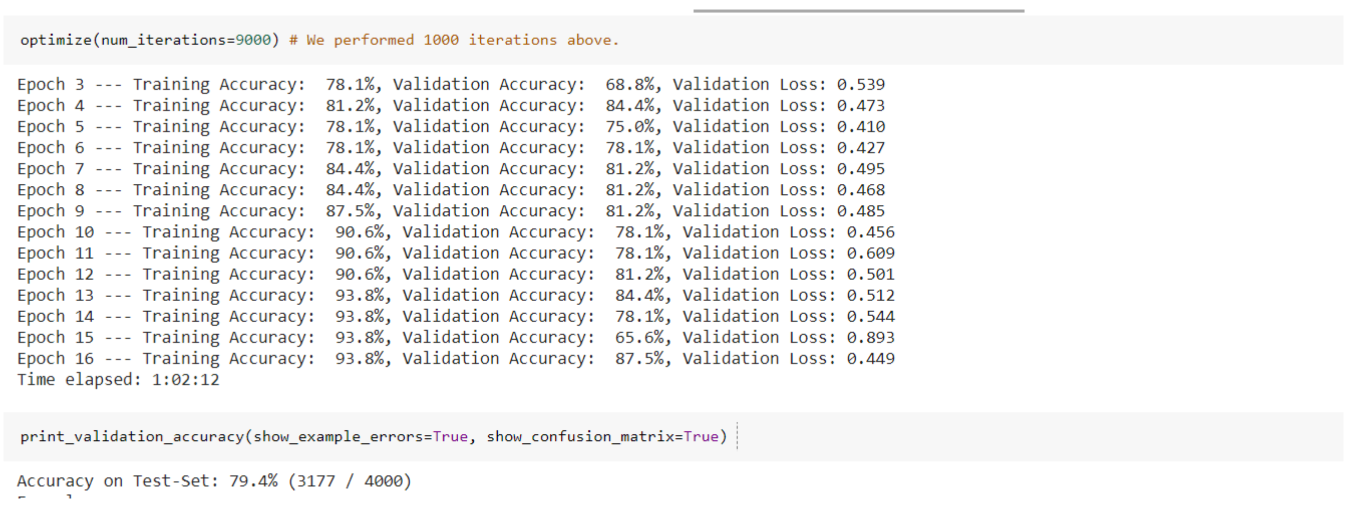


```
data = read_train_sets(train_path, img_size, classes, validation_size=validation_size)
test_images, test_ids = read_test_set(test_path, img_size)
```

```
Reading training images
Loading spam files (Index: 0)
Loading important files(Index: 1)
Loading acceptable files (Index: 2)
```

```
[10] print("Size of:")
     print("- Training-set:\t\t{}".format(len(data.train.labels)))
     print("- Test-set:\t\t{}".format(len(test_images)))
     print("- Validation-set:\t{}".format(len(data.valid.labels)))
```

```
Size of:
- Training-set:        12432
- Test-set:            95123
- Validation-set:      96633
```

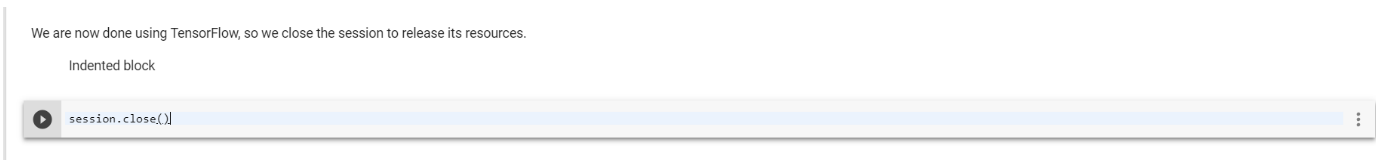


```
optimize(num_iterations=9000) # We performed 1000 iterations above.
```

```
Epoch 3 --- Training Accuracy:  78.1%, Validation Accuracy:  68.8%, Validation Loss: 0.539
Epoch 4 --- Training Accuracy:  81.2%, Validation Accuracy:  84.4%, Validation Loss: 0.473
Epoch 5 --- Training Accuracy:  78.1%, Validation Accuracy:  75.0%, Validation Loss: 0.410
Epoch 6 --- Training Accuracy:  78.1%, Validation Accuracy:  78.1%, Validation Loss: 0.427
Epoch 7 --- Training Accuracy:  84.4%, Validation Accuracy:  81.2%, Validation Loss: 0.495
Epoch 8 --- Training Accuracy:  84.4%, Validation Accuracy:  81.2%, Validation Loss: 0.468
Epoch 9 --- Training Accuracy:  87.5%, Validation Accuracy:  81.2%, Validation Loss: 0.485
Epoch 10 --- Training Accuracy:  90.6%, Validation Accuracy:  78.1%, Validation Loss: 0.456
Epoch 11 --- Training Accuracy:  90.6%, Validation Accuracy:  78.1%, Validation Loss: 0.609
Epoch 12 --- Training Accuracy:  90.6%, Validation Accuracy:  81.2%, Validation Loss: 0.501
Epoch 13 --- Training Accuracy:  93.8%, Validation Accuracy:  84.4%, Validation Loss: 0.512
Epoch 14 --- Training Accuracy:  93.8%, Validation Accuracy:  78.1%, Validation Loss: 0.544
Epoch 15 --- Training Accuracy:  93.8%, Validation Accuracy:  65.6%, Validation Loss: 0.893
Epoch 16 --- Training Accuracy:  93.8%, Validation Accuracy:  87.5%, Validation Loss: 0.449
Time elapsed: 1:02:12
```

```
print_validation_accuracy(show_example_errors=True, show_confusion_matrix=True)
```

```
Accuracy on Test-Set: 79.4% (3177 / 4000)
```



We are now done using TensorFlow, so we close the session to release its resources.

Indented block

```
session.close()
```

Observation: It can be clearly seen that the accuracy score varies from data point to data point. For some inputs the accuracy rose up to even 90%, while others were trammeled to only 70%. Computing 16 Epochs took around 1 minute. The overall accuracy was seen to be 79.4%. The results were indeed promising. After certain refinements, it can definitely be used in production.

Conclusion

This paper concluded that convolutional neural networks can be successfully deployed for image spam detection. IM and other social networks are nowadays plagued with many spam messages. It would be of paramount significance to the society

REFERENCES

Abadi, Martín, et al. 2016. "Tensorflow: a system for large-scale machine learning." *OSDI*. Vol. 16.

Chan, Tsung-Han, et al. 2015. "PCANet: A simple deep learning baseline for image classification?." *IEEE Transactions on Image Processing* 24.12 : 5017-5032.

Cireşan, Dan, Ueli Meier, and Jürgen Schmidhuber. 2012. "Multi-column deep neural networks for image classification." *arXiv preprint arXiv:1202.2745*.

He, Kaiming, et al. 2016. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*.

http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks, by Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, NIPS Proceedings.

Metev, S. M. and Veiko, V. P. 1998. *Laser Assisted Micro-Technology*, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag.

Nvidia, C. U. D. A. 2011. "Nvidiacuda c programming guide." *Nvidia Corporation* 120.18: 8.

Qiao Liu, Zhiguang Qin, Hongrong Cheng, Mingcheng Wan - Efficient Modeling of Spam Images.

Zhao, Wenzhi, and Shihong Du. "Spectral–spatial feature extraction for hyperspectral image classification: A dimension reduction and deep learning approach." *IEEE Transactions on Geoscience and Remote Sensing* 54.8 (2016): 4544-4554.

****