eYSIP-2018

Machine Learning and its Applications

Swapnil Masurekar
Abhishek Sharma

Mentors:
Rutuja Ekatpure
Suprabha Jadhav

Duration of Internship: 21/05/2018 – 06/07/2018

2018, e-Yantra Publication

Machine Learning & Applications

Abstract

This project comprises of three Machine Learning applications, viz. Optical Character Recognition (OCR), Automated Replies for Piazza, Image Captioning with face recognition. OCR involves a convolutional neural network approach, the network correctly identifies English handwritten characters with 96.03% accuracy, Automated reply system for Piazza determines the most similar existing query and sends the corresponding reply using Gmail-API, Image Captioning with Face Recognition involves VGG16 CNN model for image summarization, RNN model to map the vectors with transfer-values from the image-recognition model into sequences of integer-tokens that is converted into text, the face recognition model which identifies faces in the image and modifies the captions generated by recurrent units.

Completion status

1. **Optical Character Recognition:** The CNN model has been successfully trained over 100 epochs with 5975 samples with 98.3% of training accuracy and 97.6% of cross-validation accuracy. Correct recognition for dotted, blurred characters, handwritten characters on ruled pages, handwritten thick/thin characters with contrast in background.

2. **Automated Replies for Piazza Queries:** Correct category classification amongst two classes with 100% cross validation accuracy. Automatic labeling of mail as per the predicted category. Reply corresponding to closest query in the Piazza Dataset is sent automatically.

3. **Image Captioning with Face recognition:** Any random image is correctly captioned, if image consists of known face the corresponding image is captioned along with the name of the person whose face is predominant in the picture.

1.1 Software used

1.1.1 List of software used:

- Anaconda a scientific Python distribution for Data Science
- Anaconda includes Python 3.6, Spyder IDE & Jupyter Notebook

The list of libraries required for each application are:

1. Optical Character Recognition:
    - Scikit-Learn v0.19.1
    - Keras v2.1.5
    - Tensorflow-CPU v1.8.0
    - Pandas v0.22.0
    - OpenCV v3.4.1
2. Automated Replies for Piazza Queries:
    Extra Libraries Required are:
    - Natural Language Toolkit v3.2.5
    - *Dependencies for using Gmail-API:* apiclient, httplib2, outh2client
3. Image Captioning with Face recognition:
    Hardware used: *Graphical Processing unit used to train model is NVIDIA GeForce GTX1050Ti*
    Extra Libraries Required are:
    - Tensorflow-GPU v1.1.0
    - PIL v5.1.0

1.1.2 Installation steps:

For Installation Instructions refer this page: -

1.2 Software and Code

Github-link for the repository of code. The software description is explained below:

1.2.1 Optical Character Recognition:

Image Preprocessing: [1]

1. Acquiring the image bitmap in gray-scale colors.
2. Applying a Gaussian filter to the bitmap. (blur)
3. Applying morphological transformations for further noise removal
4. Segmenting the bitmap using thresholding to get a binary bitmap.
5. Finding the bounding boxes of external contours in the bitmap.
6. Extracting sub-bitmaps from the bounding boxes.
7. Resizing the sub-bitmaps to 20X30 or 32x32 pixels.
8. Unrolling the sub-bitmap matrices to feature vectors per 600 or 1024 elements.

Caching Characters Dataset:

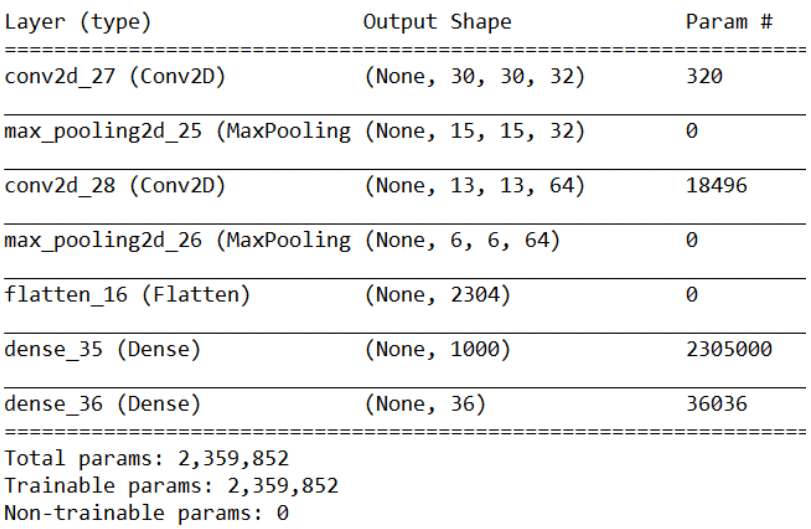
The Dataset with 5975 samples and "0-9,A-Z" label categories are preprocessed by above steps and cached into the text file which makes it computationally inexpensive while running code each time.

Logistic Regression approach:

This approach involves multinomial class classification using saga optimizer. Grid Search results of the optimizer are as shown below:

Solver	Validation Accuracy(%)
newton-cg	84.8516%
lbfgs	85.1402%
liblinear	83.6721%
sag	85.2901%
saga	85.5486%

As seen the saga optimizer has the highest cross-validation accuracy. Hence, this approach involves multinomial class classification using saga optimizer.



Layer (type)	Output Shape	Param #
conv2d_27 (Conv2D)	(None, 30, 30, 32)	320
max_pooling2d_25 (MaxPooling	(None, 15, 15, 32)	0
conv2d_28 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_26 (MaxPooling	(None, 6, 6, 64)	0
flatten_16 (Flatten)	(None, 2304)	0
dense_35 (Dense)	(None, 1000)	2305000
dense_36 (Dense)	(None, 36)	36036

Total params: 2,359,852
Trainable params: 2,359,852
Non-trainable params: 0

Figure 1.1: CNN Model Summary

Convolutional Neural Network Approach:

In *Logistic Regression* the classification is solely based on 600 independent feature pixels, which led to k-Fold cross validation accuracy of 85.54% with standard deviation of 0.04166. The accuracy is further increased by using CNN, which also enables Real-Time Data Augmentation to generalize the model. The model summary is shown in figure 1.2.1. The convolutional and fully connected layer's perceptrons are associated with Rectifier activation function while as the output layer is associated with softmax activation which gives probabilistic distribution in output layer.

1.2.2 Automated Replies for Piazza Queries:

During e-YRC help-desk and piazza receive lot of queries regarding certificate, battery, etc. Many queries are similar and are subject to same replies, so our code finds the most similar queries amongst the queries already present and send the corresponding reply. Even if the similar query is not present in the data, the mail is labeled correctly with the category to which the query belongs to, so its efficient for the user to reply to these remaining queries. Entire system flow is explained briefly in Figure: Automated-Reply Flowchart

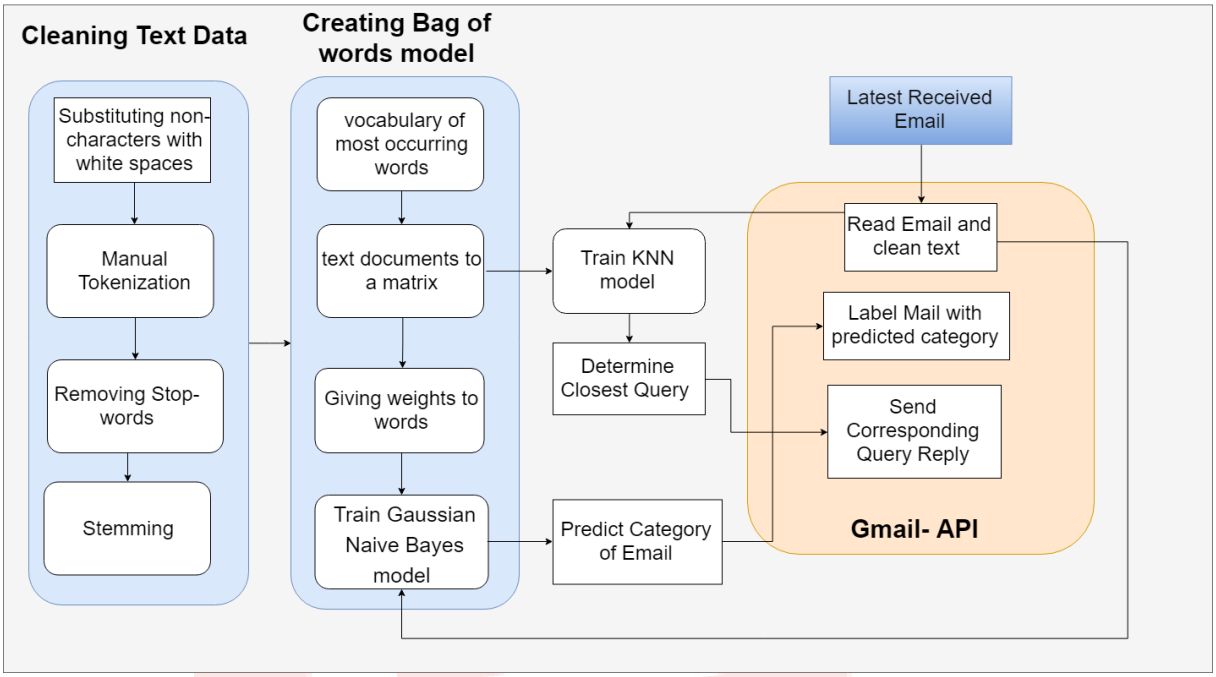


Figure 1.2: Automated-Reply Flowchart

Creating Bag of words model:

This involves two stage, first stage involves cleaning the data and in second stage we create the Bag of words model. The steps involved are:

1. Substituting non-characters with white spaces
2. Manual Tokenization: Split into words by white space
3. Removing Stop-words
4. Text Normalization - Stemming
5. Creating vocabulary of 50 most occurring words in Dataset
6. Convert a collection of text documents to a matrix of token counts
7. Giving weights to words which belong to category

Category Classification:

After the creation of bag of words model the each query with every word in vocabulary as independent vector and labeled categories are fitted to Gaussian Naive Bayes model for category prediction of latest received mail. This mail is given a category label using Gmail API.

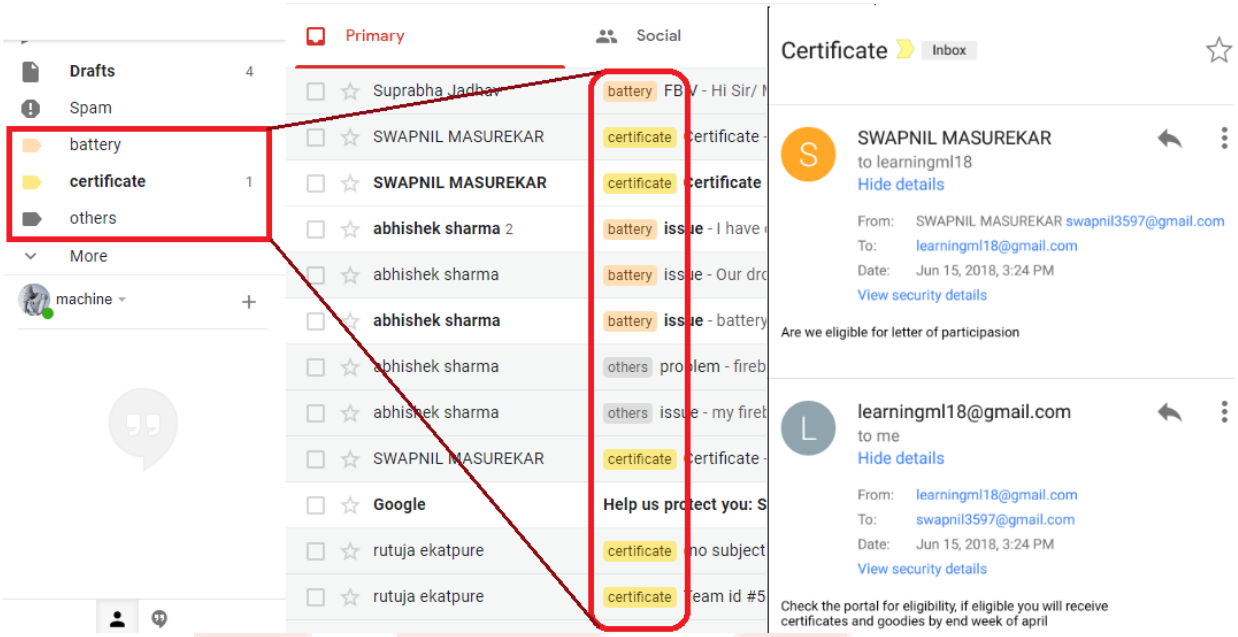


Figure 1.3: Gmail Labeling and Reply

Sentence Similarity based on Semantics nets approach:[2]

This method for semantic analysis has better accuracy over the KNN approach, but is computationally more expensive.

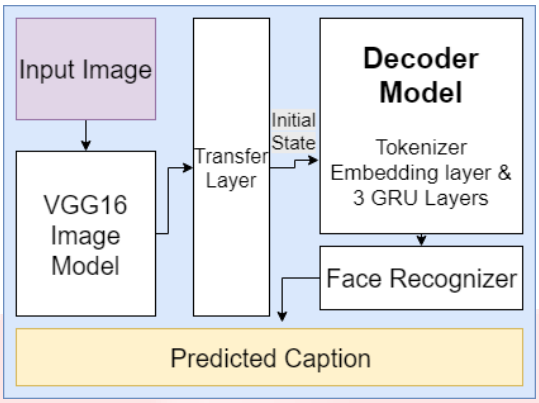
KNN Approach:

This approach involves finding closest similar query considering top words in vocabulary as independent vectors in N-Dimensions. This method is computationally less expensive and gives quite reliable predictions.

Gmail-API setup for Python Project:

Gmail API is used to modify email labels, reading mails and sending replies. To access Gmail-API using python project to need to create project on Google API. In order to create project, get credentials and client secrets refer this video. The quick-start documentation and installation instructions are provided here: Google-Documentation. In order to use pip install with anaconda you need to add *"../Anaconda3/Scripts"* path to environment variables.The example of labeling and reply is shown in Figure: Gmail Labeling and Reply

1.2.3 Image Captioning with Face recognition:



The network consist of a image classifier, Face recognition model, decoder transfer layer and a decoder network. VGG16 a pre-trained model is used as an image summarizer, the output vector which is vector of 4096 elements are mapped to 512 elements using decoder transfer layer which is taken as the initials state for the GRU layers is the decoder model. The decoder model is then trained on the caption(which is sequence of integer tokens) of the summarized image such that it predicts the next integer token based on the decoder input and the initial-state of the GRU Layers.[4]

Caching the transfer values from VGG16 model:

Since the GRU states are initialized by the output state of Image summarizer (VGG16), we cache these transfer value for all images in dataset in *".pkl"* file. These makes is computationally efficient to train the decoder model by initializing its GRU states from the cache file.

Creating the batch Generator:

Each image in the training-set has at least 5 captions describing the contents of the image. The neural network will be trained with batches of transfer-values for the images and sequences of integer-tokens for the captions. If we were to have matching numpy arrays for the training-set, we would either have to only use a single caption for each image and ignore the rest of this valuable data, or we would have to repeat the image transfer-values for each of the captions, which would waste a lot of memory. Hence we create a custom data-generator for Keras that will create a batch of data with randomly selected transfer-values and token-sequences.

Building Decoder Model:

A decoder model is build with Embedding layer, 3 GRU layers and output dense layer with linear activation function. A decoder transfer map is created to initialize the GRU's 512 states. For more details you may refer Image_Captioning_with_Face_Recognition.pdf tutorial. The loss function used while compiling the model is RMSprop. This model is trained on COCO dataset with image transfer values as initial states and caption integer tokens as input and time shifted tokens as output.

Predicting Caption:

For caption prediction we initialize the 3 GRU layer's states with by the transfer values initially and input the start-marker integer token. As the decoder model is trained on the original image captions and the time shifted output captions. When start marker is inputted the next token based on the initialized values is expected to be available at the output. Now,at this next time-step the 2nd predicted token is passed through this recurrent neural network(decoder model), and 3rd output token is expected at the output. Now at next time-step this 3rd token is passed and so on. When end-marker integer token is detected as the output of the RNN, we take the list of all predicted tokens as the output caption. This process is carried out at the most 30 time-steps, if end-marker is not detected in this 30 time steps then all predicted 30 tokens is taken as the output caption.

Face Recognizer:

The face recognizer module includes the following steps for identifying faces and substituting the human-like words detected in captions by name of the predominant person recognized in the image. [3]

- **Detecting Faces:** As face needs to be detected from a image and not video, we use cascade classifier in this case for detecting faces. Cascade classifier seemed to work efficiently. The detected faces are then cropped for further processing.
- **Normalize intensity:** This is done by equalizing histogram of a gray-scaled image.
- **Resizing image:** Image is resized to standard size of 50x50 numpy array.
- **Building local Dataset:** Preprocessing is done on all the images from live video feed and 200 normalized images of the subject are saved in the directory.

- **Train Eigen face recognizer:**Eigen face recognizer is based on the principle component analysis of the data. The model is trained on local dataset of people.
- **Prediction:** After the prediction of correct face the human-like words detected in captions are substituted by name of the predominant person recognized in the image.

Output:

The examples of the output are:

1.3 Future Work

- **Optical Character Recognition:**
  This can be further expanded over paragraph reading and recognition of characters with different colors of varied contrast.

- **Automated Reply System:**
  Due to ever increasing dataset, later word Embedding can be done and RNN model can be trained to improve the system's semantic accuracy.

- **Image Captioning with Face Recognition:**
  Computation speed can be improved by modifying the decoder so it also returns the states of the GRU-unit and make changes in caption generation such that it only inputs and outputs 1 int-token in each iteration

1.4 Challenges Faced:

1. **Optical Character Recognition:**
   - Finding the best possible dataset to train the character recognition CNN model. So we used English Handwritten dataset and expanded that dataset with morphological operations and Real-time data augmentation using Keras Image-Generator.
   - As dataset consisted of non-preprocessed images, caching the pre-processed image was necessary. Finding the best possible and most generalized preprocessing step was a challenge. The entire preprocessed dataset was cached in the text file.
2. **Automated Reply for piazza queries:**
   - As no dataset was available for Automated reply, we had to create our own dataset of piazza queries and answers. As less amount of dataset was available word embeddings was not possible.
   - Due to high computational requirements of Sentence Similarity based on Semantic nets and corpus statistics, the most similar query was determined on the basis of every word as a feature using Nearest Neighbor algorithm.
3. **Image Captioning with Face Recognition:**
   - Building a customized batch generator such that most of the information was covered from the COCO dataset on training model over several epochs.
   - Generating batches of appropriate size such that Graphical Processing Unit is occupied completely at the same time not exhausting the heap memory.

Bibliography

[1] Uhliarik, I. (2013). Handwritten Character Recognition using Machine Learning Methods [online] Available at: *http://davinci.fmph.uniba.sk/ uhliarik4/recognition/files/thesis.pdf* [Accessed 29 May 2018].

[2] Li, Y., McLean, D., Bandar, Z., OShea, J. and Crockett, K. (2006). Sentence Similarity Based on Semantic Nets and Corpus Statistics. [online] Available at: *https://ieeexplore.ieee.org/document/1644735/* [Accessed 14 Jun. 2018].

[3] Razim, R. (2017). Face Recognition using OpenCV and Python. [online] Available at: *https://www.superdatascience.com/opencv-face-recognition/* [Accessed 24 Jun. 2018].

[4] Radhakrishnan, P. (2017). Image Captioning in Deep Learning. [online] Available at: *https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2* [Accessed 25 Jun. 2018].