Machine Learning Engineer Nanodegree

Capstone Project

Harshvardhan Aggarwal
January 24th, 2018

I. Definition

Project Overview

This project focuses on analyzing and predicting various types of labels a Restaurant may have. Many of the problems today lie in correctly classifying the images, and restaurant classification is one of them. With growing number of restaurants, it becomes difficult to filter out restaurants based on specific category.

With users wanting a good recommendation for trying out new restaurants, it becomes increasingly necessary to develop a robust machine learning algorithm that can correctly classify restaurants based on various parameters like the quality of food, average wait times, kid friendly etc. This problem is just a stepping stone in classifying different types of business based on user uploaded pictures of restaurants and their qualities in order to provide better services than manual task are able to achieve.

A similar research is done to use image-recognition algorithms to classify the type of food being eaten by a person in restaurant for the purpose of automatic food journaling. [9]

Problem Statement

The main goal of this project is to look at various pictures of a restaurant business, identify its core features from the images and classify them based on their features. There are thousands of photos uploaded by users everyday in different lighting conditions, shapes, angle that it becomes difficult to properly tag them without the help of automated system.

The problem can be considered as a classification problem, where images can have multiple output class labels assigned to them. For example, a scenery image may contain roads, mountains or plains, clouds etc. In this case the output labels represent whether an image contains cloud in it or whether there is home in the image etc. Similarly for restaurants, a multiple output class label can be described as whether a image contains food or drinks, or the type of food in the plate. A user searching for a particular restaurant, say kid friendly, will help deliver results which are useful to them and in the end saving time.

For this problem, I plan to use different methods involving scaling the input images to a consistent size, changing the RBG color images to different format for processing. I also plan to use multi-label classification algorithm called OneVsRestClassifier to predict multiple labels for a single data input while using SVM classifier as the base. This is explained in detail in next few sections.

Metrics

The following evaluation metrics can be used in this case:

1. F(beta) score (from Kaggle evaluation method)
   The F(beta) score can be calculated as a measure of precision and recall.

   Precision = Sum (True Positives) / Sum (True Positives + False Positives)
   Recall = Sum (True Positives) / Sum (True Positives + False Negatives)

   F(beta) =

   ```
           (1 + beta^^2) * (precision * recall) /
           (beta^^2 * precision) + recall
   ```

   The benchmark model for this problem has a F-beta score of 0.64597 with random guessing algorithm [3]. A solution model having F-beta score *more* than the benchmark model can be considered as good model.

The F-1 score was chosen for this model as this score gives more accurate outcome than in comparison to other metrics like accuracy metric. Accuracy only measures the correctness of predicted labels whereas F-1 score will measure both the precision (Number of samples correct out of total predicted to be positive) and recall (number of samples correct out of total correct samples) of the classifier and produce results that accurately maps test's correctness.

II. Analysis

Data Exploration

The dataset for this problem is obtained from Kaggle Competition "Yelp Restaurant Photo

Classification" (Refer [1] and [2]). This dataset contains thousands of user uploaded images for restaurants. These images contain images of food, drinks of various cuisines, restaurant interiors, exteriors etc belonging to a particular restaurant. Similarly for all other restaurants, several types of images are present.

Following are some specifications of the dataset

Feature	Total count
Total number of training images	234,842
Total number of testing images	237,152
Total number of businesses in training set	1,996
Total number of businesses in testing set	10,000

The dataset contains mapping of photos to a particular business in both train and test set. The images contained in the dataset are of different sizes. Examples of some of the images from training
dataset is shown below:

An image with lower quality and dimensions of 500x375 pixels (width x height)

An image with better quality than previous one and dimensions of 375x500 pixels (width x height)

The below image shows how the training photos id's are mapped to each business id in the dataset.

```
print(train_photos_to_business_id[:5])
```

```
   photo_id  business_id
0    204149         3034
1     52779         2805
2    278973          485
3    195284          485
4     19992          485
```

As shown above, each photo is mapped to a particular business (restaurant). Each business owner can have multiple photos which can then be assigned multiple labels depending on the image.

Similarly, for testing dataset also there is a mapping between each test photo and businesses. These businesses are different from the training data.

```
print(test_photos_to_business_id[:5])
```

```
   photo_id business_id
0    317818       003sg
1     30679       003sg
2    455084       003sg
3    371381       003sg
4     86224       003sg
```

The dataset also has 9 categories of labels to be predicted upon. They are:

Category Label	Category Name
0	good_for_lunch
1	good_for_dinner

2	takes_reservations
3	outdoor_seating
4	restaurant_is_expensive
5	has_alcohol
6	has_table_service
7	ambience_is_classy
8	good_for_kids

Hence, for each business there is a clearly defined multiple output labels which identifies the type or category of restaurant.

So, for example, below is the mapping of training business with their labels.

```
y_training_labels = pd.read_csv(dataset_root + 'train.csv')
print(y_training_labels[:5])
```

```
   business_id          labels
0         1000   1 2 3 4 5 6 7
1         1001         0 1 6 8
2          100     1 2 4 5 6 7
3         1006       1 2 4 5 6
4         1010           0 6 8
```

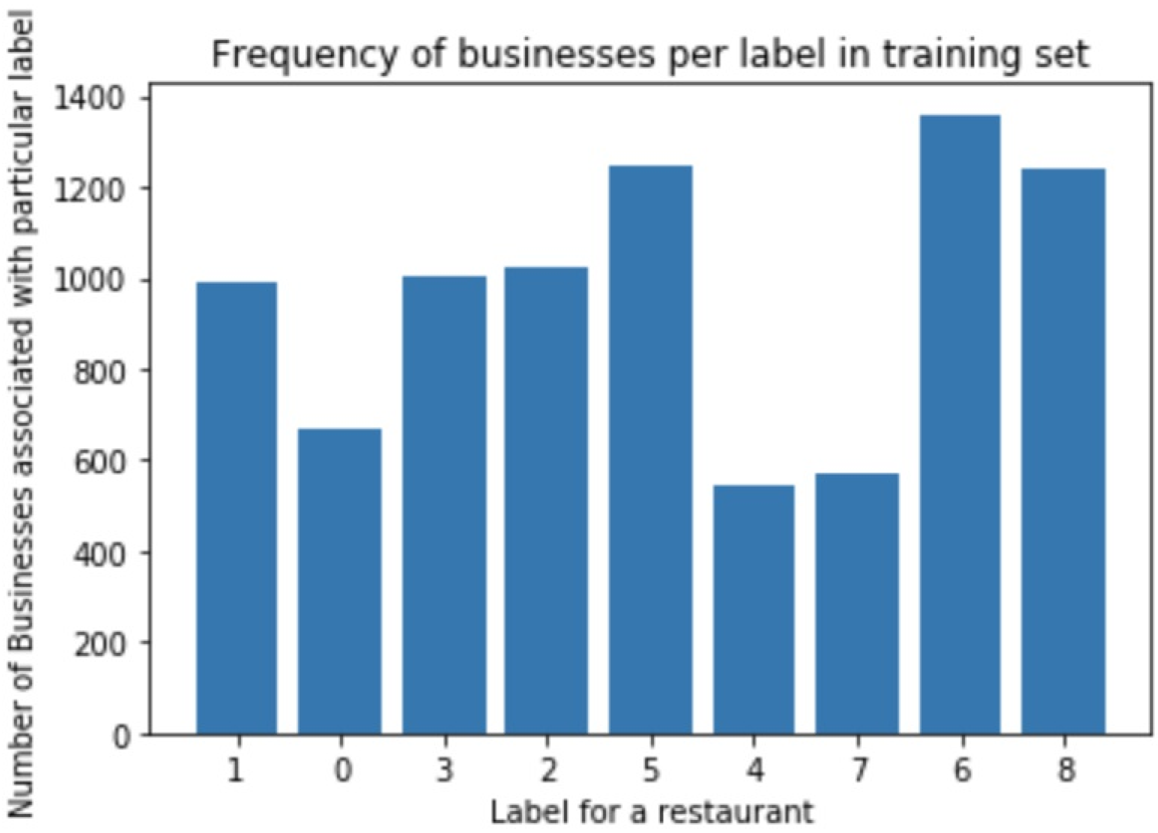
The numbers corresponding in label are mapped to the label name in table above. So business id `1001` has labels `(0, 1, 6, 8)`. This means business 1001 is identified as `good_for_lunch`, `good_for_dinner`, `has_table_service` and `good_for_kids`.

The goal is to find the label for testing businesses.

The training dataset did have some abnormalities, which were identified while working with the data. Four of the businesses in the dataset don't have any labels associated with them. Out of total 2000 businesses present in the dataset, only 1996 were considered for training.

Exploratory Visualization

The plot below shows the number of businesses that represent for each class of label for restaurant.



From the above plot, it is evident that there is some bias towards the output labels of (5, 6, 8) while the classes (0, 4, 7) are under-represented in this dataset. I did some data augmentation like rotation and gamma correction to some of the images represented by output labels (0, 4, 7) to increase their number in dataset and prevent bias from happening. The augmentation step is explained in the data pre-processing section below.

Also, the below images show the distribution of images between different businesses.

For training set, out of total 234,842, the average number of photos per business is 117. Also, the minimum number of photos for a restaurant is at least 2 images.

```
Maximum photos in a business for training set: 2974
Minimum photos in a business for training set: 2
Average photos in a business for training set: 117
```

Similarly, the number of images per business for testing set are in line with the training images as shown below:

```
Maximum photos in a business for testing set: 2825
Minimum photos in a business for testing set: 1
Average photos in a business for testing set: 119
```

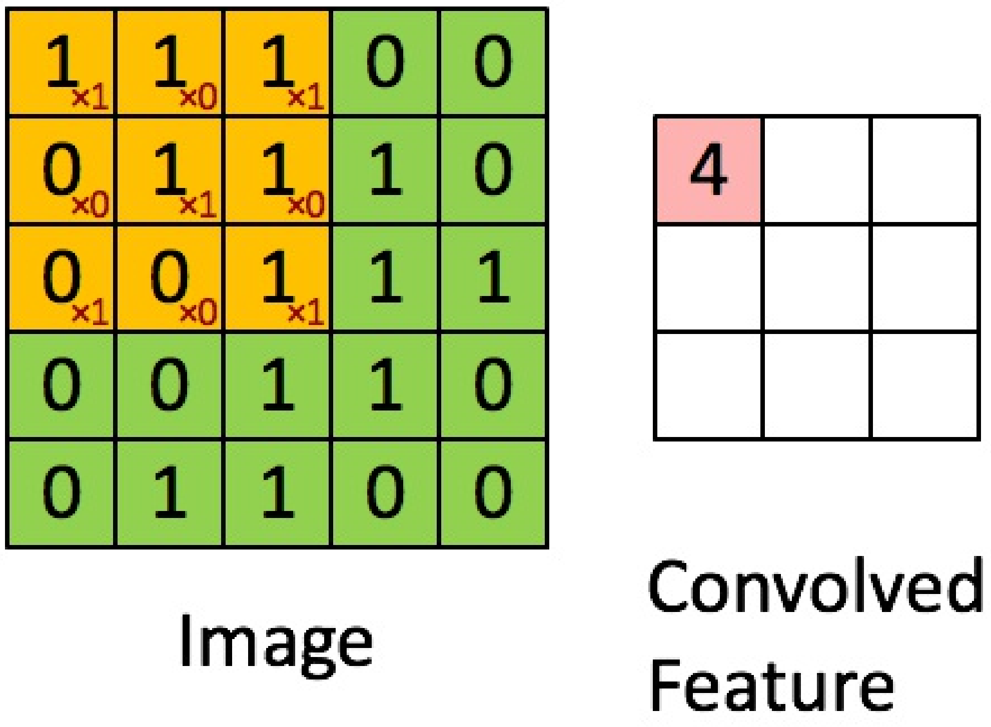
Algorithms and Techniques

The algorithm and techniques used in this project includes use of `convolution neural networks` to extract the features from images and then using these features as an attribute in a supervised learning algorithm like linear regression or support vector machines.

In a CNN, there are mainly 5 layers which forms the overall Architecture to produce an output. These are as follows:

1. **Input Layer** - As the name implies, this is the layer where input is fed to the network. An input can be an image with (width x height x number of color channels).

2. **Convolution layer** - This layer is the building block of CNN. This layer consists of set of learnable filters which are small in width and height but extends through the full depth of the input (the color channels). For example lets consider the image below:

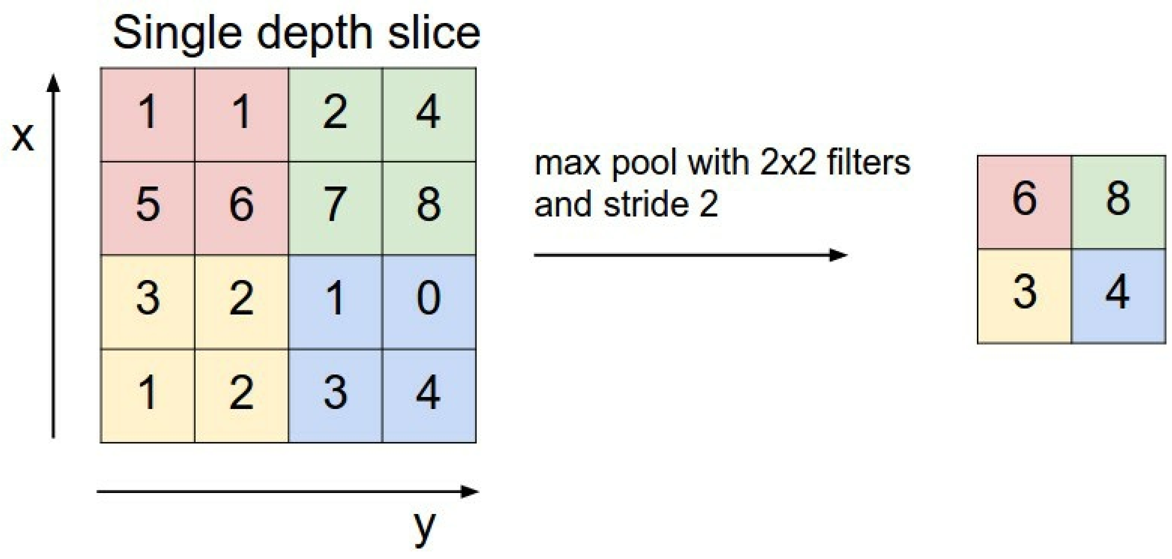


In the above image, the input image (marked in green) is of size (5x5) and filter size is

(3x3). At each pass, we slide the filter (marked in yellow) from top left corner to across the full width and height of image. At each position of filter, we compute the dot product between the pixels covered by filter and the filter itself to give a convolved feature (marked with pink) of size (3x3). If the filter depth was suppose 12 filters, each filter will slide over the input image to produce an output convolved feature set of size (3x3x12).

3. **Activation layer** - This layer will apply an activation function to capture only elements which are above some thresholds. Examples of some activation functions are sigmoid, Relu etc. The output of relu layer has size same as input layer size.

4. **Pooling layer** - The pooling layer downsamples the input into smaller dimensions mainly in height and width. This is done to reduce the size of parameters and computations in the network and prevent overfitting. The pooling layer operates independently on every depth slice of the input and resizes it spatially. An example of how maxpool layer operates is shown below:



In this layer a filter of an appropriate size is chosen. In this example the, the output from Convolution layer produced (4x4) output. If the maxpool layer size is (2x2) and stride (amount by which to slide the filter) is 2, then the max pool filter is slided from top left corner. At position (0,0) the maxmimum of of values covered by (2x2) filter is chosen (here 6). The filter is slided by stride of 2 pixels. In next step at position (0,2), the maxmimum value of area covered by filter is chosen (this time its 8) and so on.

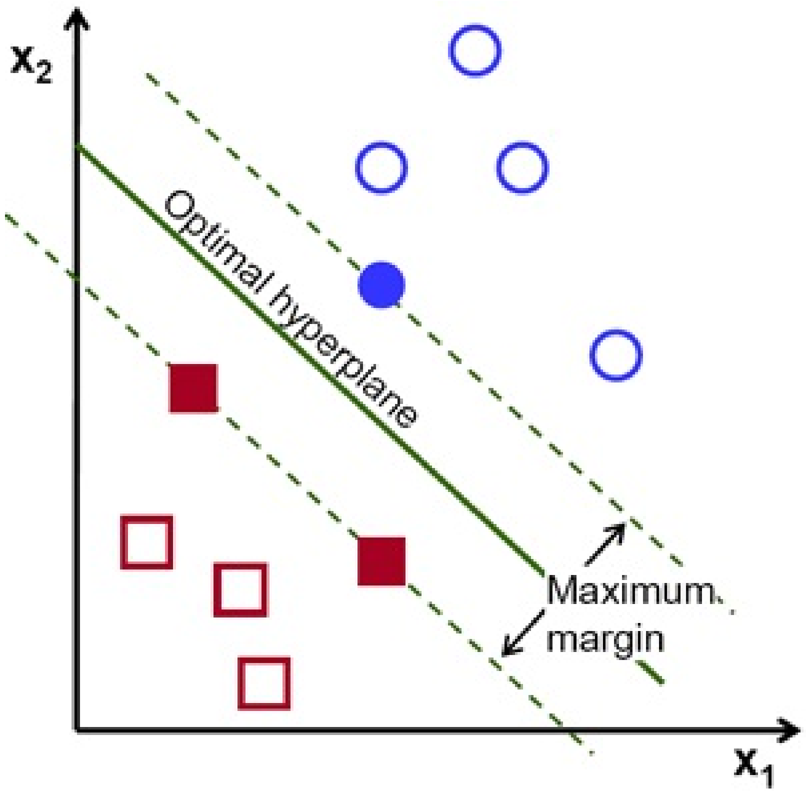
5. **Fully connected layer** - The fully connected layer is also called dense layer and can transform 3-D input from previous layer into a flattened output. The nodes in this layer are fully connected each and every node in the input. The final dense layer may have output nodes with size of the number of classes to predict upon.

During training process, a CNN with many layers is used to extract the high-level and low-level features from the images. I incorporated Caffe to make use of multiple core of GPU's at a single time to reduce time extracting features. Caffe is built to support parallel processing on multiple cores of GPU. The tunable parameters for this stage can range from specifying the shape of image to be used to the number of filters and size of each filter at each layer of CNN.

The CNN was chosen to represent an image in terms of numerical features along with business (restaurant) and its labels. The combination of multiple image features can be used as input to a supervised learning classifier while labels will act as the output to be predicted upon.

The next algorithm I used in this project was `Support Vector Machines (SVM)` to classify the output received from the convolution layers. The support vector machines are used as supervised learning classifier to assign a label for each business.

A linear SVM classifier works by drawing a hyperplane (a line segment) to separate the different classes so as to isolate maximum points belonging to one class from another. It does so by choosing a hyperplane that maximizes the margin between the classes as shown below:



SVM classifier can also be trained to perform a non-linear separation of training inputs using what is called a kernel trick. Kernels are functions which convert low dimensional input space to a higher dimensional space. A kernel can be either a linear algebra function or a polynomial function of varying degree that best separates the various input classes.

The tunable parameters for SVM can be the type of kernel used, the initial random state to shuffle data and the gamma variable to determine the variance.

The classification of restaurants problem falls into the multi-label classification problems. The multi-label classification problems are the ones where each instance or row of data can be assigned to multiple classes (also called labels) at once. They differ from binary classifiers or multi-class classifiers in way that binary or multi-class classifiers have all classes as mutually exclusive. This means an instance of data cannot be assigned to more than one label.

This project uses OneVsRestClassifier algorithm, which is an implementation of multi-label classification provided by sklearn using a base classifier of user's choice (e.g. Decision Tree, SVM, Linear Regression). This classifier fits a classifier for each class of label to be predicted upon. In this problem, it will generate 9 types of classifier for each label to output a confidence measure if the particular instance can belong to one of the classes. In the end it outputs all the labels which have a high confidence measure as compared to others.

Benchmark

The benchmark metrics for this project is obtained from Kaggle leaderboard page. The project has a F-beta score of `0.64597` with random guessing algorithm as shown in Kaggle leaderboard page[3]. This will be considered as the benchmark model for this problem.

A solution model having F-beta score *more* than the benchmark model can be considered as good model.

III. Methodology

Data Preprocessing

The dataset for this project was given in form of csv files. The csv files contains the mapping of each image (image name) with a particular restaurant id.

For pre-processing the images, I used Caffe framework to modify the input images in the form Caffe expects. Caffe is a deep-learning framework designed for speed and works well for image classification.

The steps performed in pre-processing the data are as follows:

1. The pandas library is used to load all the image file names from the csv file.
2. Once the file name of images are present in an array, each of the image is read using Caffe library.
3. Caffe framework provides Caffe transformer module, which can transform the input image in the form Caffe can process.
4. The first step in Caffe transformation is to convert input image size channel from `(Height x Width x Number of color channels)` to `(Number of color channels x Height x Width)`.
5. Next step is to calculate the mean and subtract the mean BGR pixel values from image. This is done to even out the image properties (like contrast etc).
6. The input image is now scaled to [0, 255] pixel values as Caffe operates on images in this range.
7. Next, the RGB color channel is swapped into BGR format so that Caffe can process the image correctly.

At the end of these steps, all the images were of same size with same order of color channels to provide consistent results.

The next step in pre-processing was augmenting the dataset with images that under-represented output class labels (0, 4, 7). I specifically added images for these classes to prevent any bias from happening and the classifier actually learns the features and not memorize them.

To add data to existing training dataset, I used the scikit learn transform library to transform the images by performing following steps:

1. Rotating the images by either 90, 180 or 270 degress randomly.
2. Adjusting the brightness (gamma) of the image by decreasing the brightness by quarter or half or doubling the brightness randomly.

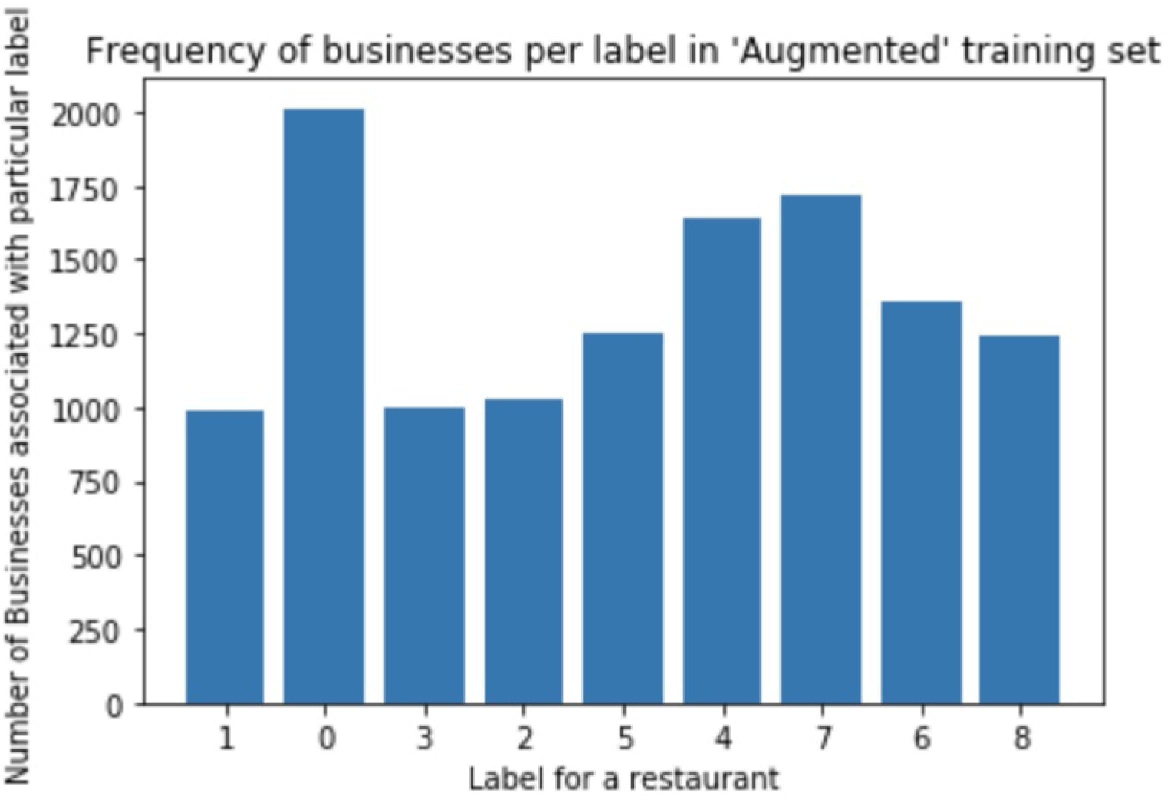
Example of an image that went through pre-processing steps is shown below:

The original image in the training dataset:

The rotated image (image rotated randomly by 90 degrees):

The brightness adjusted image (brightness reduced by half):

The updated plot showing the frequency of businesses that represent each class of label is shown below:



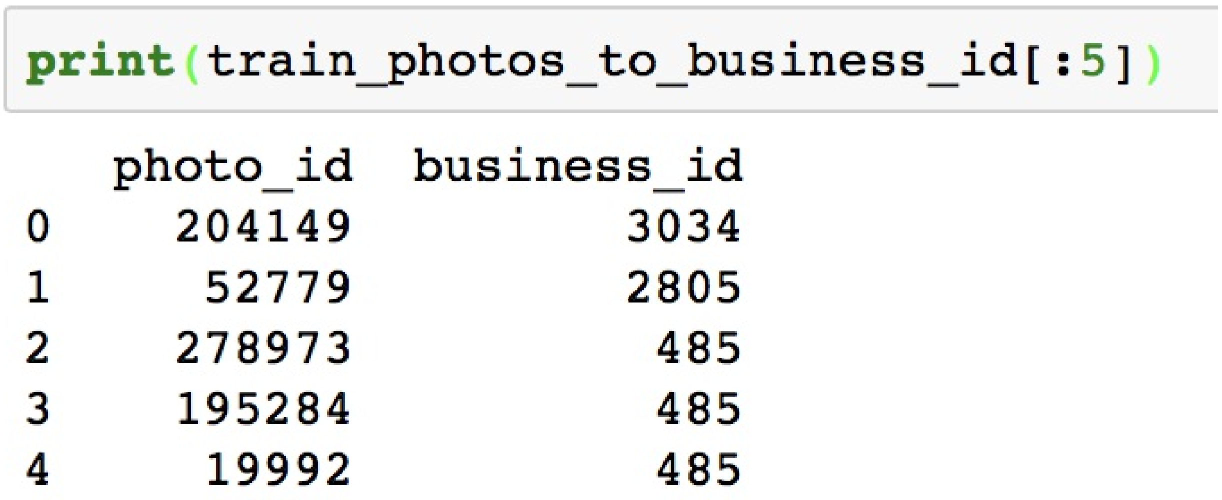
Implementation

The implementation step consisted of below main steps:

1. Load the training images from CSV file.
2. Compute features from these training images and store them.
3. Load the businesses and their labels from CSV file.
4. Compute mean of all image features associated with one business and add it as a feature along with label.
5. Use the business and features as training data and labels as training output to train a multi-label classifier.
6. Extract features from testing images.
7. Compute mean of all images associated with a testing business.
8. Predict the labels for testing set using the classifier trained earlier.

Now lets take a detailed look at each of the steps described above.

1. Firstly I used pandas DataFrame to load `train_photo_to_biz_ids.csv` file to create a list of all the images being used in training dataset. The training dataset looked like below:



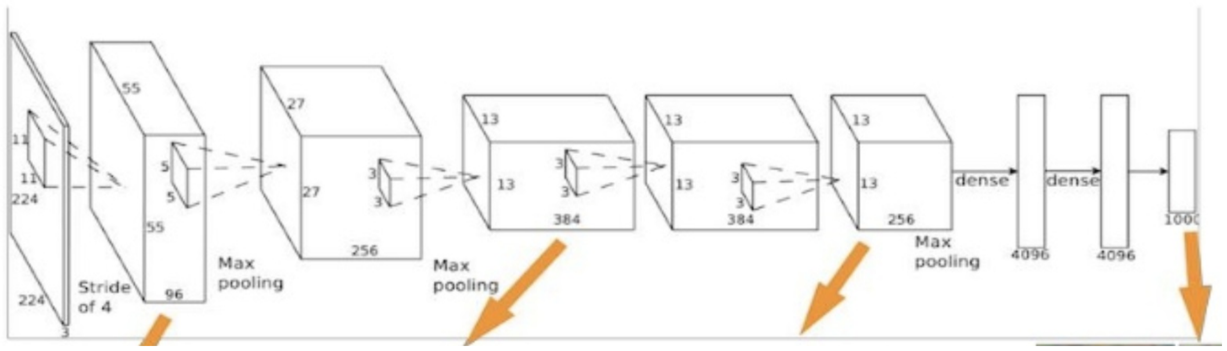
```
print(train_photos_to_business_id[:5])
```

```
   photo_id  business_id
0    204149         3034
1     52779         2805
2    278973          485
3    195284          485
4     19992          485
```

2. Next, to extract features from each image I thought of using a custom CNN to extract low and high-level features with multiple layers using different filter size and strides and padding. Since, the training image dataset consisted of 234,842 images with similar number for testing set, it was essential that all important features were extracted within a reasonable amount of time. A custom-built CNN turned out to be very time consuming taking hours to extract features from all the images. I then started to explore to use Caffe within my project to extract image features. As it turns out, Caffe proved helpful in extracting the features as it reduced the time to couple of hours as compared earlier.

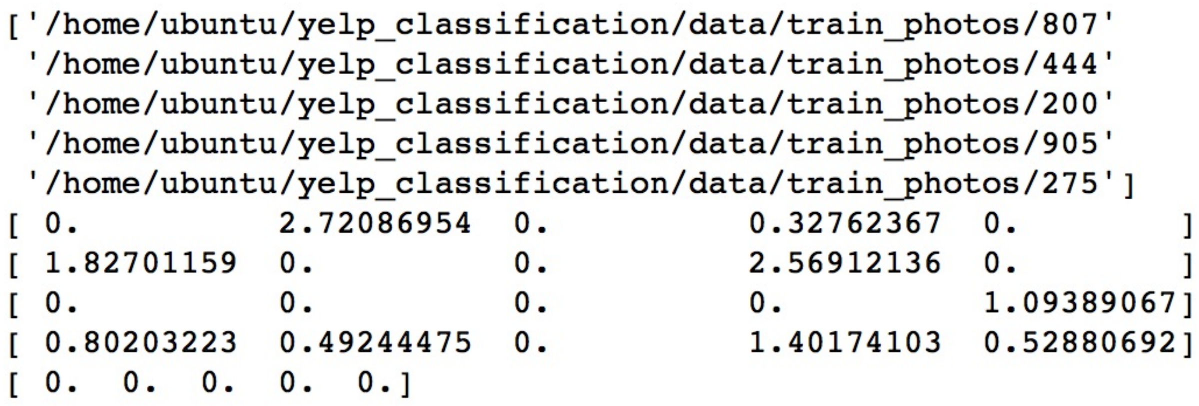
However, using Caffe was no small feat as setting up of Caffe took a long time on AWS and made sure all the dependencies were up to date. Once the Caffe was up and running, I used the Caffe provided model called `BVLC Reference CaffeNet` which is Caffe's implementation of ImageNet optimized for Caffe framework. The BVLC reference model has 8 layers as shown below:

The image has been taken from the link[6]



I used the BVLC CaffeNet to provide image as input and perform computation on the images for all layers. The layer-8 is very specific to CaffeNet as it outputs features based on the number of classes for this model. I used the output of layer-7 as the image features that can be used to classify each business later. The layer-7, also called fc7, provides good representation of all the features for image. The output of layer 7 has 4096 nodes, so all the image features consists of 4096 features.

Below is the representation of features for images 5-10 and their first 5 features from 4096 features:



```
['/home/ubuntu/yelp_classification/data/train_photos/807'
 '/home/ubuntu/yelp_classification/data/train_photos/444'
 '/home/ubuntu/yelp_classification/data/train_photos/200'
 '/home/ubuntu/yelp_classification/data/train_photos/905'
 '/home/ubuntu/yelp_classification/data/train_photos/275']
[ 0.          2.72086954  0.          0.32762367  0.        ]
[ 1.82701159  0.          0.          2.56912136  0.        ]
[ 0.          0.          0.          0.          1.09389067]
[ 0.80203223  0.49244475  0.          1.40174103  0.52880692]
[ 0.  0.  0.  0.  0.]
```

3. Next, I loaded the business and their output labels from CSV file. This looked like below:

```
y_training_labels = pd.read_csv(dataset_root + 'train.csv')
print(y_training_labels[:5])
```

```
   business_id          labels
0         1000   1 2 3 4 5 6 7
1         1001         0 1 6 8
2          100     1 2 4 5 6 7
3         1006       1 2 4 5 6
4         1010           0 6 8
```

4. The earlier features calculated from images were then averaged out and added as an extra column to above businesses. This created a new attribute to be used by classifier to train and learn which labels are associated with image features. The combination of businesses, features and labels is shown as below:

```
First five features
   business_id                    label  \
0         1000  (1, 2, 3, 4, 5, 6, 7)
1         1001           (0, 1, 6, 8)
2          100     (1, 2, 4, 5, 6, 7)
3         1006        (1, 2, 4, 5, 6)
4         1010              (0, 6, 8)

                                            features
0  [0.19977085, 0.43287092, 0.22732987, 0.3551694...
1  [0.0, 0.58893245, 0.53906047, 0.17221628, 0.01...
2  [0.11155061, 0.034822084, 0.12025276, 0.520122...
3  [0.078059338, 0.054452635, 0.05638162, 0.69423...
4  [0.39657032, 0.27962369, 0.0, 0.17205141, 0.36...
```

5. The next step was to train the classifier. Before classifying, I used the image features as input to the classifier and labels as the expected output (labels). This is a multi-label classification problem so it required to one hot encode the labels from (0, 1, 2...) to something that can be represented in binary form. I used `sklearn.MultiLabelBinarizer()` to transform the training labels into a 2-D matrix where point in (row x column) having value 1 represents that this data instance (image feature) can be represented by this label column. The output of MultiLabelBinarizer looks like below:

```
Original Labels
[list([1, 2, 3, 4, 5, 6, 7]) list([0, 1, 6, 8]) list([1, 2, 4, 5, 6, 7])
 list([1, 2, 4, 5, 6]) list([0, 6, 8])]
One hot encoded labels
[[0 1 1 1 1 1 1 1 0]
 [1 1 0 0 0 0 1 0 1]
 [0 1 1 0 1 1 1 1 0]
 [0 1 1 0 1 1 1 0 0]
 [1 0 0 0 0 0 1 0 1]]
```

I then used `sklearn.train_test_split()` method to split the data into training and validation set and then used the `sklearn.OneVsRestClassifier()` with SVM as base classifier as explained earlier to be used as classifier algorithm.

6. I then used the same Caffe model `BVLC Reference CaffeNet` to calculate the image features for testing dataset. I again used the second-to-last 'fc7' layer to get the same level of features as in training set.

7. As described in step-4 above, the mean feature was calculated from the 'fc7' layer features.

8. Using these mean testing features, I used the classifier which was trained in step-5 to predict multiple labels for each image feature.

At the end of the above step-8, we get the predicted labels for all the testing business which can be scored again the Kaggle submission test suite.

Refinement

The first step in refinement was to choose between a custom Convolution Neural network built to extract image features versus using transfer learning by using an existing algorithm and reduce training time.

The main tradeoff between choosing the two was the training time for computing features for 234,842 training images and 237,152 testing images. Using a custom built CNN was taking long time to process all the images (sometimes more than 6 hours). The accuracy of finding the correct features was also a concern as if enough different features are not present, the classifier may not produce good results. Due to processing time and accuracy, I used the existing Caffe model to extract the features.

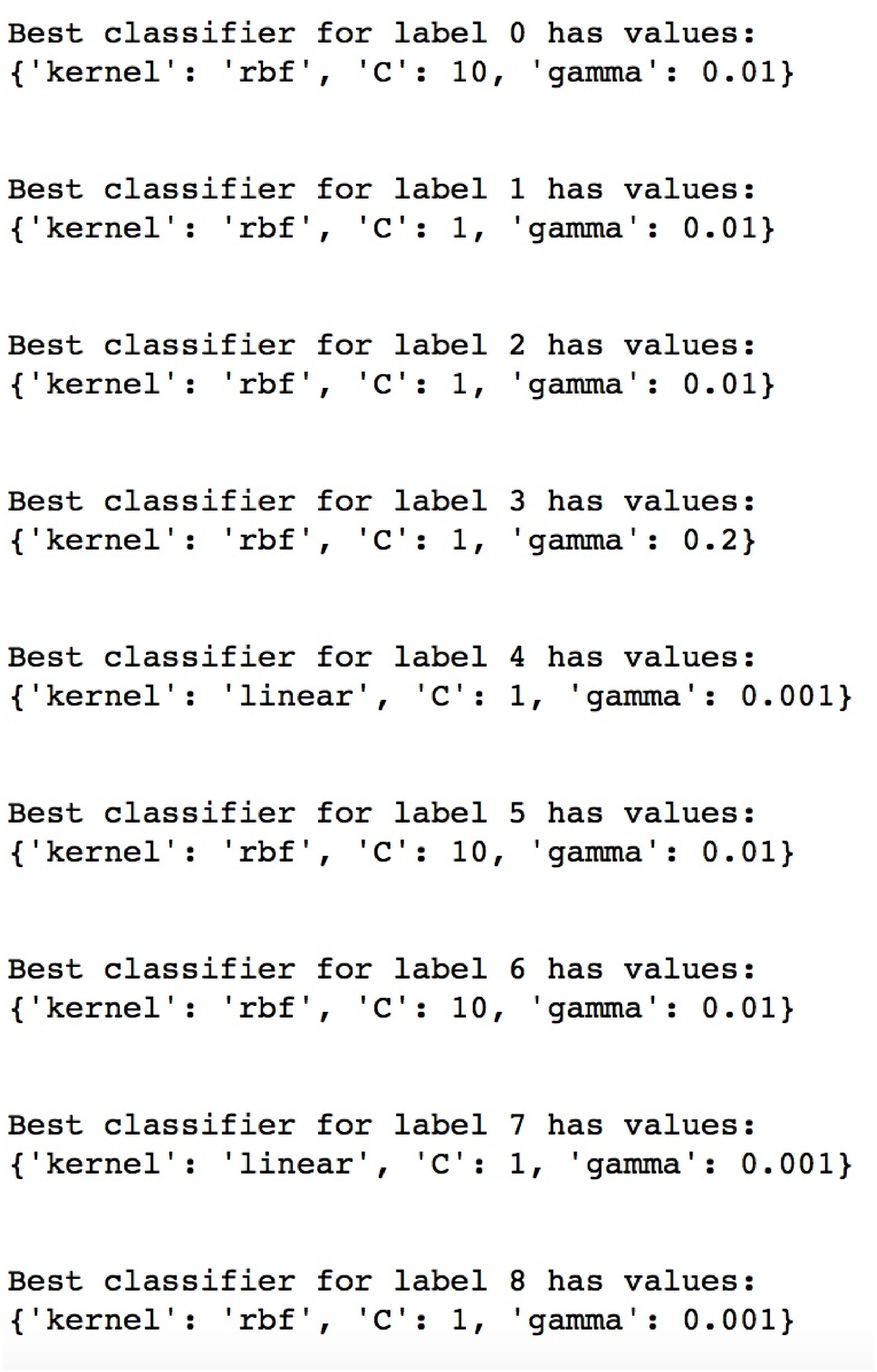
The second refinement I made was adjusting the parameters of Support Vector Machine used in the OneVsRestClassifier algorithm. I first used the linear kernel as the base classifier. I then switched to an RBF kernel with a random state as my next classifier.

The results for the linear kernel SVM on testing dataset had an F-1 score of 0.75. Given the dataset (images) and the features extracted from it, it seems that this data is not linearly separable as many image features were conflicting with the labeled outputs. So, I used the 'Radial basis function (rbf)' kernel to create hyperplanes so as to differentiate between different features. The results for using rbf kernel on testing dataset resulted in improved F-1 score of 0.79.

The third refinement I made was making use of sklearn's `GridSearchCV` algorithm with SVM as the estimator to find the best SVM parameters that fits the data. The various parameters used for doing an exhaustive search by GridSearchCV were:

1. Kernel type for SVM: either linear or rbf.
2. The 'C' parameter: Values were [1, 10, 50]
3. The 'Gamma' parameter: Values were [1e-3, 1e-2, 0.2]

The results for the GridSearchCV with maximum results have the following parameters for each output label:



```
Best classifier for label 0 has values:
{'kernel': 'rbf', 'C': 10, 'gamma': 0.01}


Best classifier for label 1 has values:
{'kernel': 'rbf', 'C': 1, 'gamma': 0.01}


Best classifier for label 2 has values:
{'kernel': 'rbf', 'C': 1, 'gamma': 0.01}


Best classifier for label 3 has values:
{'kernel': 'rbf', 'C': 1, 'gamma': 0.2}


Best classifier for label 4 has values:
{'kernel': 'linear', 'C': 1, 'gamma': 0.001}


Best classifier for label 5 has values:
{'kernel': 'rbf', 'C': 10, 'gamma': 0.01}


Best classifier for label 6 has values:
{'kernel': 'rbf', 'C': 10, 'gamma': 0.01}


Best classifier for label 7 has values:
{'kernel': 'linear', 'C': 1, 'gamma': 0.001}


Best classifier for label 8 has values:
{'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
```

However, the actual F-1 score obtained by the GridSearchCV classifier on testing dataset on Kaggle website was lower than SVM classifier with rbf kernel and no tuning of other hyperparameters. The F-1 score for GridSearchCV is 0.713.

IV. Results

Model Evaluation and Validation

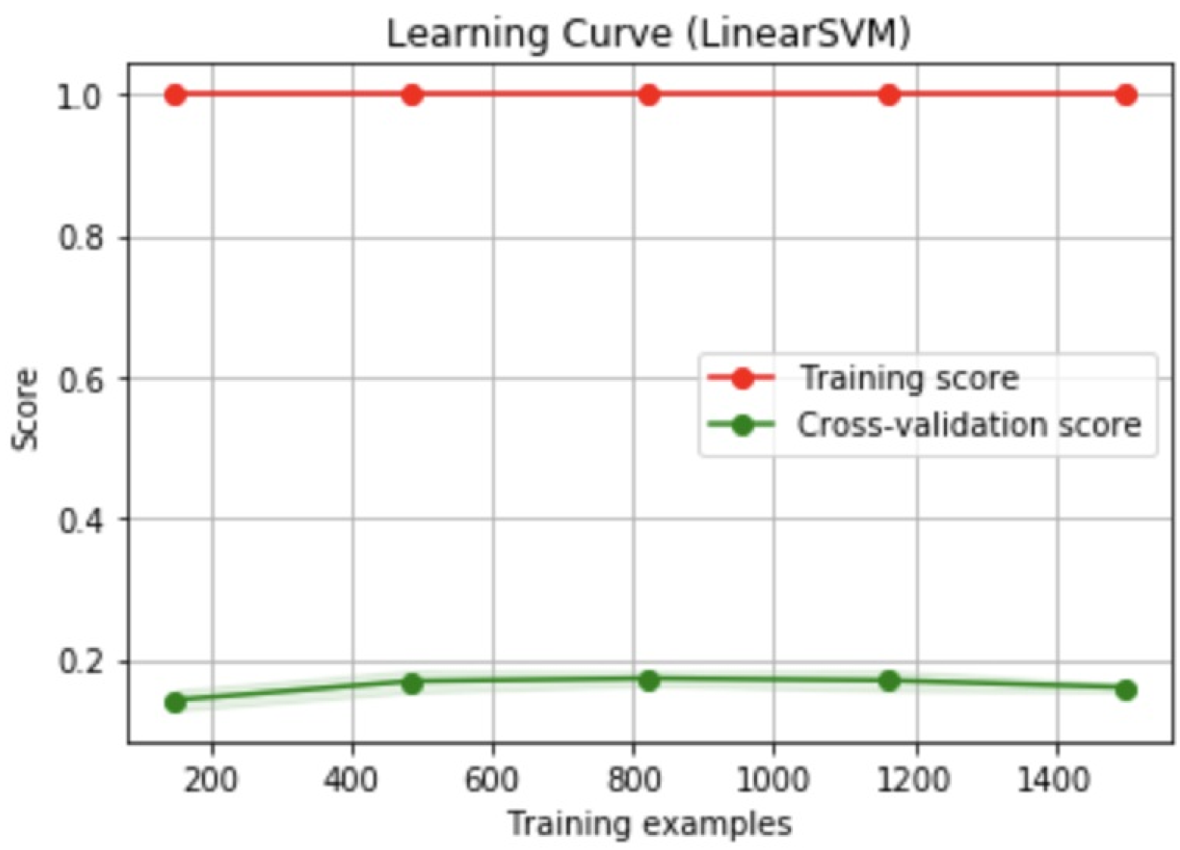
The final model consisted of the following algorithms and techniques:

1. A CaffeNet model called `BVLC Reference CaffeNet` was used to extract image features from both training and testing set. The output of the second last 'fc7' layer was used as input features to classifier.
2. The output labels in training set was one hot encoded using sklearn.MultiLabelBinarizer().
3. The image features extracted above for both training and testing were used as input to OneVsRestClassifier().
4. The OneVsRestClassifier() used SVM as base classifier with `rbf` kernel to classify training images and predict on testing images.

The choice of OneVsRestClassifier() seems appropriate for this multi-label classifying problem as this algorithm will try to fit a classifier for each class of labels in order to predict the correct labels for testing set.

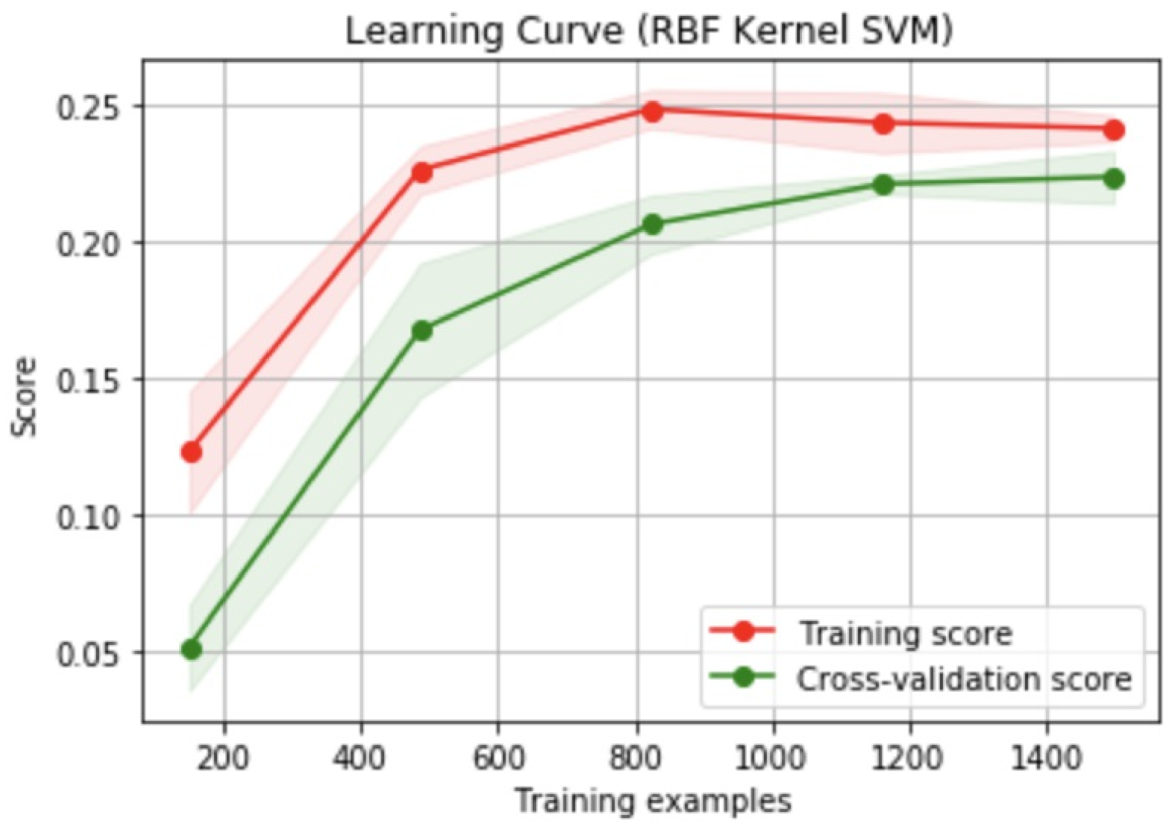
I used sklearn's `learning_curve` function to plot the training and validation set scores (validation set consisted of 25% of original dataset) average over 3-5 cycles.

The learning curve for Linear SVM classifier is shown below:



Surprisingly, the linear SVM performs too badly on the validation scores since this score never reaches beyond 0.2 even after learning over many samples of data.

The learning curve for an SVM classifier using 'rbf' kernel is shown below:



The above image captures a real world scenario where both training score and validation score have very less values in the beginning as the classifier is still learning to predict correctly. At the end both training and validation score flattens out which indicates that classifier has learnt successfully.

The SVM classifier with rbf kernel model is thus a good choice for this problem and is confirmed by the fact that it was able to achieve a high F-1 score of 0.79 on 237,152 testing images alone.

Also, the model achieved a high F-1 score of 0.829 on validation dataset (25% of training dataset was reserved to be used as validation data).

Justification

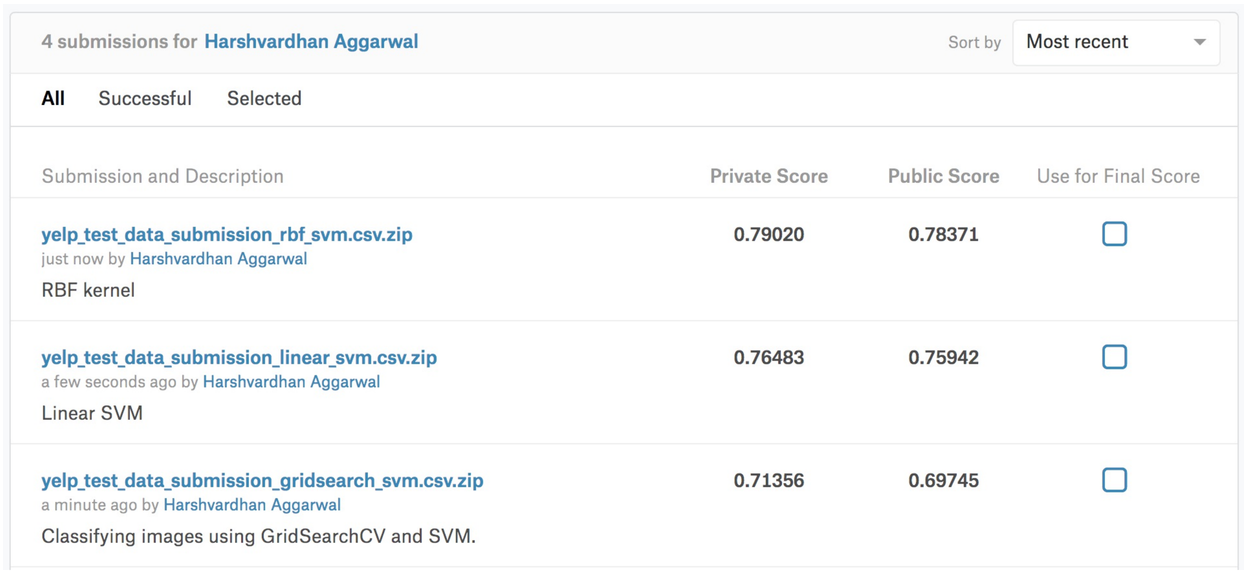
The benchmark model earlier mentioned had an F-1 score of 0.64597 with random guessing as the algorithm chosen.

The model implemented above with an SVM classifier using `Linear` kernel had an F-1 score of 0.75.

The same model with SVM classifier using `RBF` kernel has an even higher F-1 score of 0.79.

The F-1 score obtained by using GridSearchCV with best parameters as rbf kernel in an SVM classifier having C and gamma values as 10 and 1e-2, respectively, is 0.713.

Hence, the F-1 score obtained by rbf kernel SVM (0.79), which is obtained from the Kaggle website, affirms that this model is stronger than the benchmark model and does a good job in classifying of nearly 80% of images into correct categories.



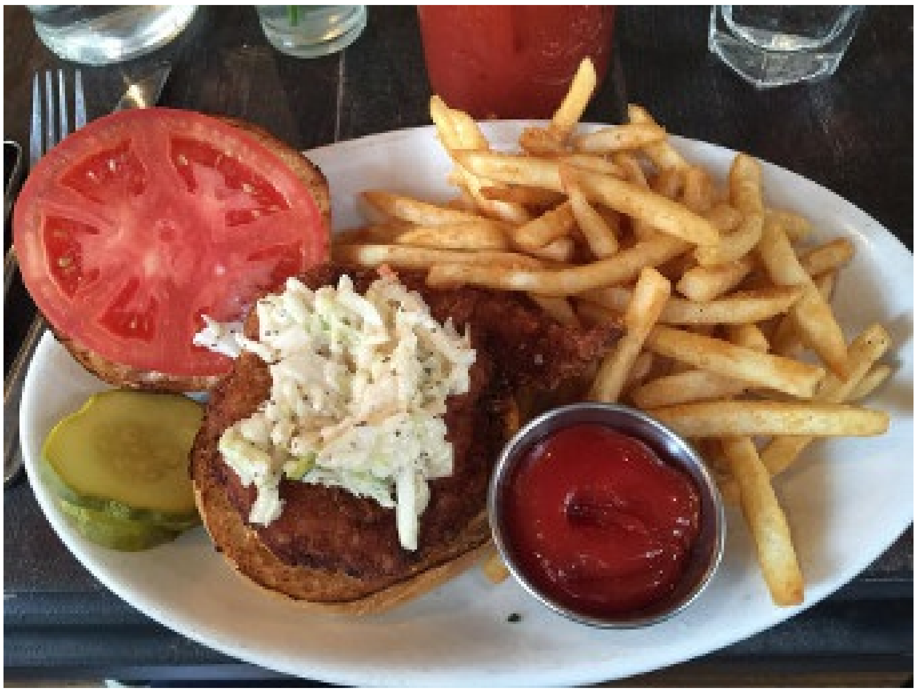
Submission and Description	Private Score	Public Score	Use for Final Score
yelp_test_data_submission_rbf_svm.csv.zip just now by Harshvardhan Aggarwal RBF kernel	0.79020	0.78371	☐
yelp_test_data_submission_linear_svm.csv.zip a few seconds ago by Harshvardhan Aggarwal Linear SVM	0.76483	0.75942	☐
yelp_test_data_submission_gridsearch_svm.csv.zip a minute ago by Harshvardhan Aggarwal Classifying images using GridSearchCV and SVM.	0.71356	0.69745	☐

V. Conclusion

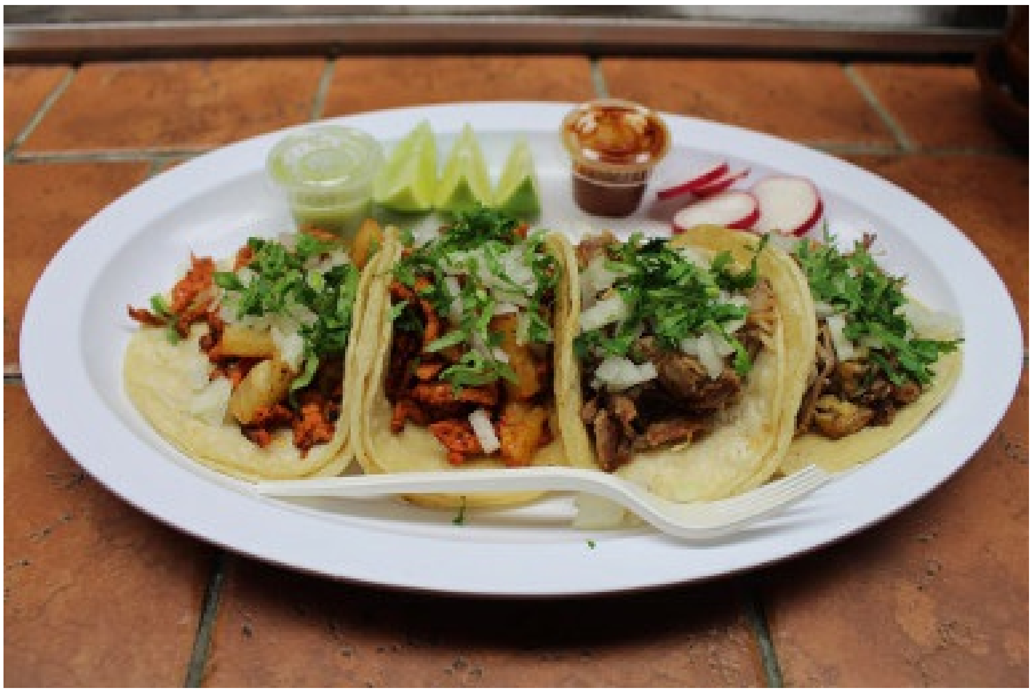
Free-Form Visualization

To visualize the restaurant image classification, I am using below two images, which are not part of the original Yelp dataset, to see how well they perform.

The first image is shown below:



And the second image looks like below:



Running the algorithm through these two images produces the following output labels:

```
Image-1: /home/ubuntu/yelp_classification/data/yelp_rest_test_image_1.jpg
Predicted Labels
(6, 8)
Image-2: /home/ubuntu/yelp_classification/data/yelp_rest_test_image_2.jpg
Predicted Labels
(0, 6, 8)
```

The predicted label for the first image based on the table earlier defined is (6 -

has_table_service) and (8 - good_for_kids).

Similarly for second image, the algorithm predicts the output labels as (0 - good_for_lunch), (6 - has_table_service) and (8 - good_for_kids)

From the yelp pages description of restaurant, the algorithm is quite close to some of the results. For example, both the restaurants are good for kids and algorithm determined correctly based on the inference from training images.

The algorithm is still not perfect as it clearly misses some of the expected labels like 'takes_reservations' and 'restaurant_is_expensive'.

Reflection

The process used in developing and coming up with a solution in this project is summarized as:

1. The Kaggle dataset was loaded onto a Amazon AWS Deep Learning Machine Image to take advantage of Caffe.
2. Caffe framework was setup on the AWS instance.
3. The training and testing image dataset were fed through a A CaffeNet model called `BVLC Reference CaffeNet` to extract the features.
4. The reference model's layer 7, called 'fc7', was used as input features to classifier.
5. The training data set was augmented with the image features along with business id and the output labels.
6. The output labels in training set was one hot encoded using sklearn.MultiLabelBinarizer().
7. The sklearn.OneVsRestClassifier() was used for multi-label classification with base classifier for each label being a SVM classifier with rbf kernel.
8. The training dataset was fitted to the classifier.
9. The testing dataset consisting of image features was passed to the classifier to predict its output labels.
10. The multi-labels generated by classifier were submitted to Kaggle for final results.

The difficult part of the project has to be working with the Caffe framework. Although the Caffe framework is documented, it was still not enough to get started as errors while setting up the framework keep cropping up if I updated any dependent libraries. I also got many errors initializing Caffe in IPython notebook as the input images were not in the format Caffe expected which took a while to figure it out. The next difficult thing was to storing the features of images. As extracting features took a very long time to complete, it was not feasible to run every time I start the AWS instance and hence required me to store those features.

The interesting aspect of the project was to find out that there are problems and then there are solution algorithms for multi-label classification. This class of problem was entirely new as compared to the coursework. These type of multi-label algorithms are very useful in real-world scenarios like self-driving car where you can classify multiple object in just a single frame.

This project is just a stepping stone in restaurant image classification and can be useful for future implementation to see what other techniques can be augmented to make it more robust.

Improvement

The first improvement that can be made is in using a different mechanism altogether like TensorFlow to take advantage of parallel processing and speed things up. This will likely result in reduction in time to extract features from the thousands of images. Also, the classifier itself can be run inside TensorFlow framework utilizing its speed to shorten the time more.

The second improvement that I researched was using AdaBoost or ensemble method for classification in a multi-label classification setting. I think if we use any of these two methods, they would improve upon the accuracy achieved by rbf kernel in SVM classifier.

Based on the Kaggle leaderboard, it appears there exists a better solution that can increase the accuracy from 0.79 to 0.83 probably by using better classifier that supports multi-label classification problems.

VI. References

All the references that I used that made me realized this project are as follows:

1. Kaggle Competition Page - https://www.kaggle.com/c/yelp-restaurant-photo-classification
2. Dataset for Project - https://www.kaggle.com/c/yelp-restaurant-photo-classification/data
3. Caffe Framework and Integration with Yelp DB - https://engineeringblog.yelp.com/2015/10/how-we-use-deep-learning-to-classify-business-photos-at-yelp.html
4. Kaggle Benchmark Model value - https://www.kaggle.com/c/yelp-restaurant-photo-classification/leaderboard
5. AlexNet Image classification on ImageNet - http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

6. CaffeNet - https://www.slideshare.net/ducha/visual-object-tracking-review

7. Kaggle blog posts inspiration:

    i. https://www.kaggle.com/c/yelp-restaurant-photo-classification/discussion/20127
    ii. http://blog.kaggle.com/2016/04/28/yelp-restaurant-photo-classification-winners-interview-1st-place-dmitrii-tsybulevskii/
    iii. https://www.kaggle.com/c/yelp-restaurant-photo-classification/discussion/18762

8. Getting started with Caffe http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

9. V. Bettadapura, E. Thomaz, A. Parnami, G. D. Abowd and I. Essa, "Leveraging Context to Support Automated Food Recognition in Restaurants," 2015 IEEE Winter Conference on Applications of Computer Vision, Waikoloa, HI, 2015, pp. 580-587.
   doi: 10.1109/WACV.2015.83
   http://ieeexplore.ieee.org/abstract/document/7045937/