Capstone Project

I-Chun Liu
October 28th, 2018

I. Definition

Project Overview

The ability to understand the context of an image is extremely useful. For instance, it could be used to reduce and optimize search results for pictures, adding tags automatically to crowdsourced photos on various social platforms, and allowing images to have "words." However, to understand the context of an image, the program must understand the nuances or the "depth" of the image. In Simonyan and Zisserman's paper, they used Convolutional Neural Networks to perform large-scale, multi-label image classifications and concluded that deep CNNs have superior performance than the ones that had fewer convolutional layers [1]. I think it is an exciting problem with many applicable uses for a system that could perform well in multi-label image classification tasks in the real world.

Problem Statement

Yelp held a photo classification competition on Kaggle two years ago. It asked Kagglers to build a model that automatically tags user-uploaded photos with multiple labels, nine labels to be exact. In this capstone project, I will be working on designing and building a Convolutional Neural Network to try to achieve or better the highest benchmark score. The goal of this project is to assign a set of labels to each photo correctly.

Metrics

In 2015, Yelp published an introductory blog post on this challenge, where it went over the metric that is used to assess the performance of a model and the performances of two benchmark models. The evaluation metric for this challenge is the mean F1 score.

The F1 score measures the accuracy by using precision p and recall r. Precision is the ratio of the number of true positives (tp) to the total number of elements being classified as positive (tp + fp). Recall is the ratio of the number of true positives to the number of elements that should be classified as positive (tp + fn). Since this task has multi-label, I will compute the mean of F1 score, which is the weighted average of the F1 score of each class label. The following is the formula for the F1-score:

$$F1 = 2\frac{p \times r}{p + r} \; where\, p = \frac{tp}{tp + fp} \; and\, r = \frac{tp}{tp + fn}$$

The F1 score gives equal weights to recall and precision. To have a high F1 score, one must have both high recall and precision scores. Having a low recall or a low precision score would result in a low F1 score [2].

II. Analysis

Data Exploration

The target labels for this challenge are the following:

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

The datasets can be found at https://www.kaggle.com/c/yelp-restaurant-photo-classification/data

The datasets consist of the following files:

sample_submission.csv.tgz

test_photo_to_biz.csv.tgz

test_photos.tgz

train.csv.tgz

train_photo_to_biz_ids.csv.tgz

train_photos.tgz

sample_submission.csv.tgz shows how the submission file is supposed to look like. There are two columns in this CSV file. The first column contains all the business IDs, and the second column contains each business ID's corresponding labels in digits from 0 to 8. If there are multiple labels in a business ID, they are separated by spaces.

test_photo_to_biz.csv.tgz and **train_photo_to_biz_ids.csv.tgz** contain the mapping from each photo to its associated business ID. In other words, this mapping allows a business to have more than one pictures, and we're able to tell which photos belong to which store.

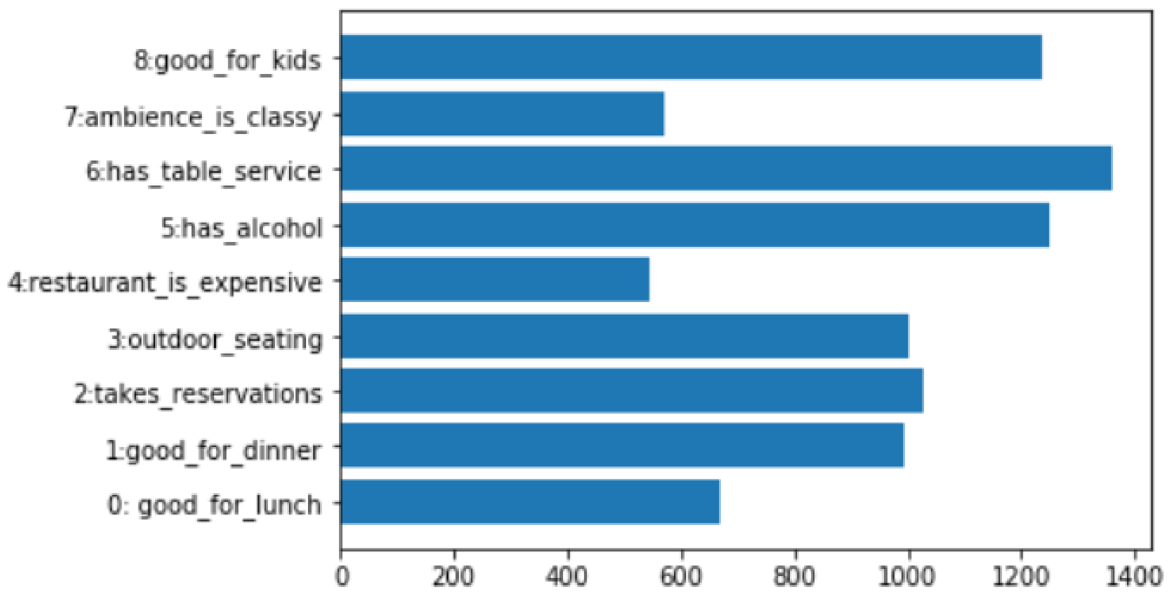
test_photos.tgz and **train_photos.tgz** contain the actual photos. Photo ID is in each photo's file name. All the images are user-uploaded, meaning they are non-uniform in sizes and color images. There is a total of 469683 training images and 474304 test images. Some of them are unusable, which contain an underscore in front of their filenames. As a result, we're left with 234840 training images and 237152 test images.

train.csv.tgz contains each business ID with its associated correct/truth labels. There are 2000 distinct businesses.

To allow hyperparameters tuning and assessing training performances, I randomly divided 20% of the training data into the validation dataset and 80% into the training dataset. The test dataset does not have each business' truth labels. To obtain the test accuracy for the test dataset, I submitted my test prediction results to Yelp's challenge page on Kaggle.

In addition to processing photos' filenames with an underscore, I performed data manipulations on the training target labels. Originally, if a target label has numbers 0, 1, 6, 8, it would be in the form of a string with each label number separated by a space: "0 1 6 8". After data manipulations, it would be [1 1 0 0 0 0 1 0 1]. On the other hand, to obtain the final prediction results in the format of sample_submission.csv, I reversed the above process to transform arrays of target labels to strings of target labels separated by spaces. For instance, if the prediction result is [1 1 0 0 0 0 1 0 1], the transformed array would be “0 1 6 8”.

The following bar chart is the distribution of the training dataset’s target labels:

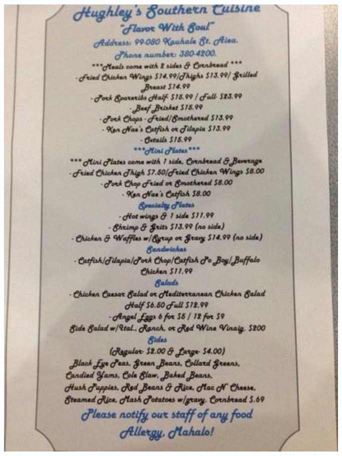


Labels 0, 4, and 7 appear less frequently than the other labels. We could use this distribution of labels as a tuning parameter for determining whether a label should be classified as true.

Exploratory Visualization

Because this challenge has real-world, user-uploaded photos of restaurants from Yelp, we are dealing with a very complex set of photos. They range from food, storefronts, selfies, menus, restaurant interior, company logos, etc. These are some of the training photos:

UFood Grill
feel great. eat smart.



Hughley's Southern Cuisine
"Flavor With Soul"
Address: 99-080 Kauhale St. Aiea.
Phone number: 380-4200.
***Meals come with 2 sides & Cornbread ***
- Fried Chicken Wings $14.99/Thighs $13.99/ Grilled Breast $14.99
- Pork Spareribs Half- $15.99 / Full- $23.99
- Beef Brisket $15.99
- Pork Chops - Fried/Smothered $13.99
- Ken Nae's Catfish or Tilapia $13.99
- Oxtails $15.99

Mini Plates
*** Mini Plates come with 1 side, Cornbread & Beverage
- Fried Chicken Thigh $7.50/Fried Chicken Wings $8.00
- Pork Chop Fried or Smothered $8.00
- Ken Nae's Catfish $8.00

Specialty Plates
- Hot wings & 1 side $11.99
- Shrimp & Grits $13.99 (no side)
- Chicken & Waffles w/Syrup or Gravy $14.99 (no side)

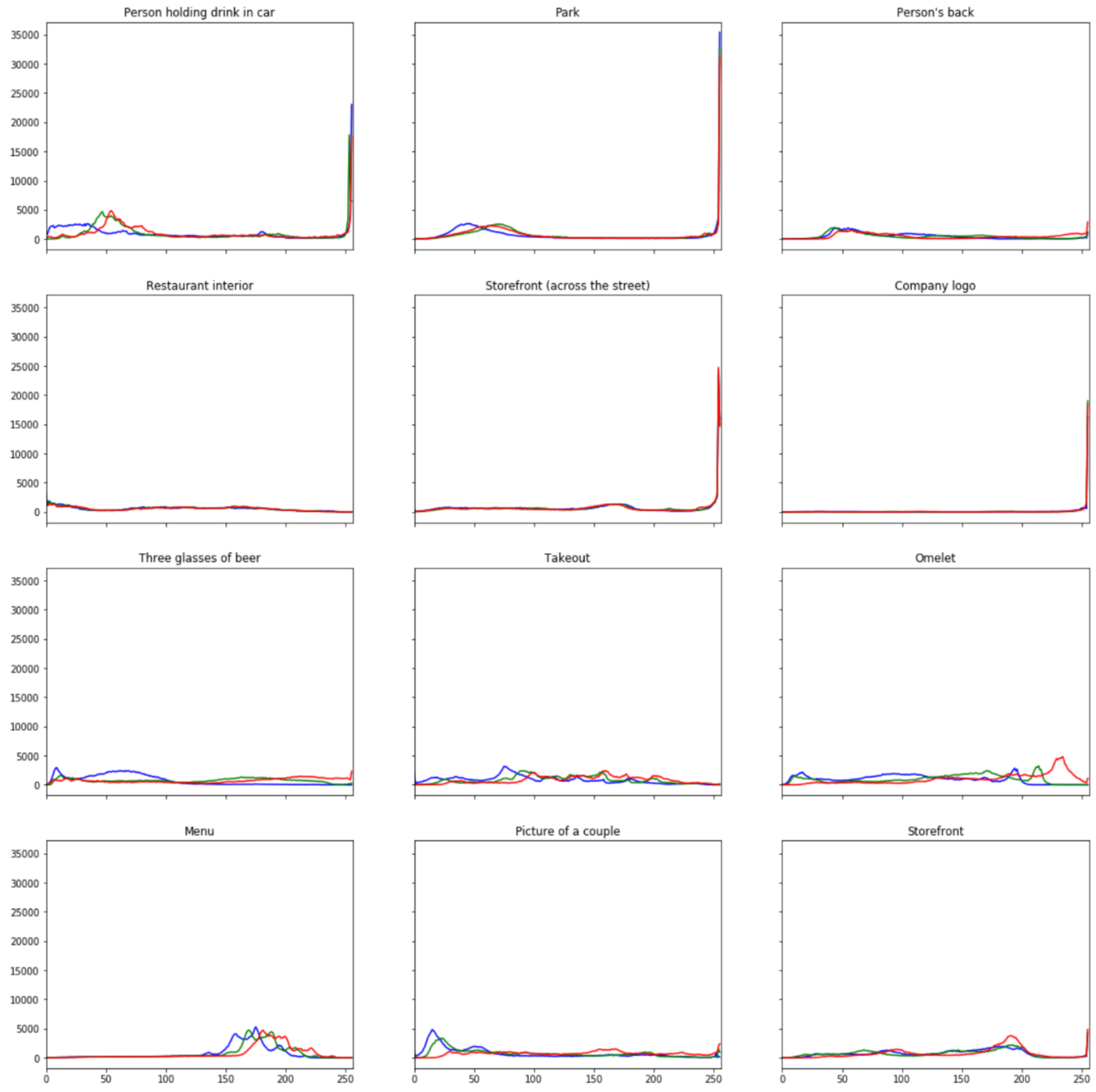
Sandwiches
- Catfish/Tilapia/Pork Chop/Catfish Po Boy/Buffalo Chicken $11.99

Salads
- Chicken Caesar Salad or Mediterranean Chicken Salad Half $6.50 Full $12.99
- Angel Eggs 6 for $5 / 12 for $9
Side Salad w/Ital., Ranch, or Red Wine Vinaig. $200

Sides
(Regular- $2.00 & Large- $4.00)
Black Eye Peas, Green Beans, Collard Greens, Candied Yams, Cole Slaw, Baked Beans, Hush Puppies, Red Beans & Rice, Mac N' Cheese, Steamed Rice, Mash Potatoes w/gravy, Cornbread $.69

Please notify our staff of any food Allergy, Mahalo!

The following plots contain histograms of those training images' pixel intensity values:

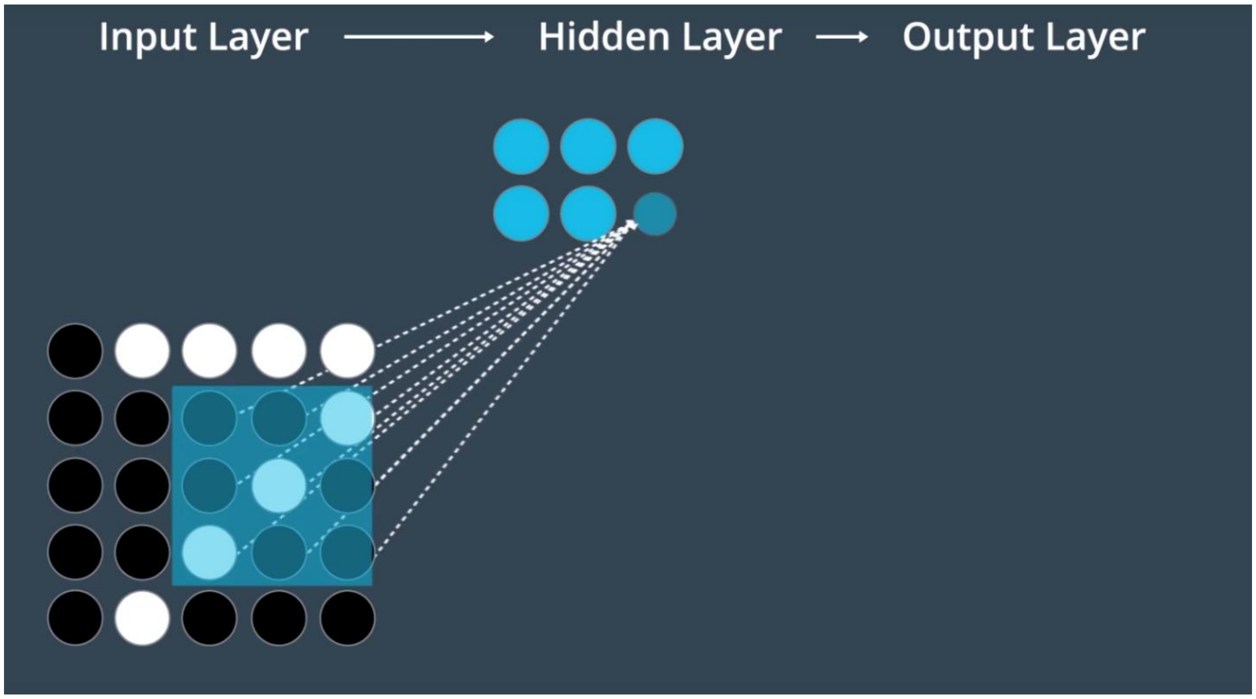


Each subplot contains three lines, which are colored coded in green, blue, and red. They are the color channels of an image. An image is made of RGB colors, and each color channel has values between 0 and 255. A histogram of the pixel intensity values allows us to see an image's pixel colors' distribution, which could be a useful feature in this project. Based on the plots, we can see different types of photos exhibiting different distributions [7].

Algorithms and Techniques

Since this task requires understanding the depth of images, a Convolutional Neural Network would be appropriate for this task. Besides, according to a Yelp blog post, Yelp also utilized Convolutional Neural Networks to classify business photos [3]. Because of the varieties of photos, pre-trained Convolutional Neural Networks may be a great candidate to use to extract features from the images.

What are Convolutional Neural Networks (CNN)? Compared to a Multi-Layer Perceptron (MLP), which uses fully connected layers, a CNN uses sparsely connected layers, meaning each node does not need to be responsible for understanding the entire image. Thus, it’s less prone to overfitting. A CNN contains one or more convolutional layer(s). A convolutional layer involves sliding a window, which is called a filter, horizontally and vertically through an image of a matrix of pixels. Each movement of the window creates a new node in the hidden layer, and this set of nodes is called the convolutional layer. For each node, we will multiply the input nodes with their corresponding weight and sum up their result, which is then fed into an activation function. The following figure shows a filter is being moved to the right to create a new node in the hidden layer:



When we have multiple convolutional layers, we could have multiple filters that detect different patterns in an image [8].

Why is a pre-trained Convolutional Neural Network preferable to building one from scratch for this project? Pre-trained Convolutional Neural Networks have already been trained to process lots of different categories of photos, which applies to our task that requires understanding different types of photos. Moreover, many images have similar underlying shapes and edges. Pre-trained networks have already learned this information, so building a Convolutional Neural Network from scratch would most likely be inefficient and have worse performance.

Once I extract the bottleneck features, I would use the training bottleneck features for feedforward and backpropagation and use the validation bottleneck features to check how the model performs in each epoch. During feedforward, the weights and the inputs are multiplied to get the hidden node values. Depending on which activation function we have, we will use the activation function with the hidden node values to compute the outputs. If we have more than one layers in our model, this procedure repeats for the number of layers in the model. Backpropagation happens after each feedforward operation. It compares the outputs of the model with the desired outputs and calculates the error. It then runs the feedforward operation backward to spread the error to each of the weights. Furthermore, the weights are updated accordingly to the error [8].

To sum up, I would be experimenting with different pre-trained Convolutional Neural Networks for features extraction and models with varying numbers of fully connected layers for training.

Benchmark

In 2015, Yelp published an introductory blog post on this challenge, where it went over the metric that is used to assess the performances of two benchmark models. The first benchmark model is based on a random guesser, where each label has equal probability. This model results in a score of 0.4347. The second benchmark model "calculates the color distribution of all the images of a test business, compares that to the average color distribution of businesses with positive attribute values and negative attribute values respectively, and assigns the value with a more similar color distribution to the test business." This model achieves a score of 0.6459 [4]. My goal for this project is to come up with a model that scores higher than the benchmark model that utilizes color distribution.

III. Methodology

Data Preprocessing

For training and validation datasets:

1. Get each photo's ID from their filename (excluding photo IDs with an underscore)

2. For each photo ID, I use train_photo_to_biz_ids.csv.tgz to obtain this photo ID's business ID and use train.csv.tgz with the business ID to retrieve this business/photo's target labels. Then, I perform data manipulations to transform a string of labels to an array of labels. For instance, business ID 1001 has labels "0 1 6 8." After data manipulations, it would have [1 1 0 0 0 0 1 0 1].

3. I use sklearn's train_test_split to divide 20% of the training data into the validation dataset and 80% into the training dataset.

For using Keras' pre-trained Convolutional Neural Networks:

1. Keras CNNs require a 4D array as an input with shape:

   (number of images, rows, columns, channels)

   Therefore, I load each image and resize it to size 244x244. Since an image is comprised of red, green, and blue (RGB) values, the number of channels is 3. For each image, it has a shape of (1, 244, 244, 3). We have a total of 234840 usable photos, so after this pre-processing step, we have a NumPy array of (234840, 244, 244, 3) [5].

2. I use Keras' ImageDataGenerator to perform image augmentations. I believe image augmentations are important especially for this challenge because user-uploaded photos come in various sizes and orientations. I use rotation_range=20, width_shift_range=0.2, height_shift_range=0.2, shear_range=0.05, zoom_range=0.2, and horizontal_flip=True. They are used to account for images that are slightly off-center, out of focus, or tilted.

3. Before using a pre-trained model to perform feature extraction, I use the corresponding pre-trained model's preprocess_input function to allow images to become fully compatible with the pre-trained model.

4. I performed the above steps on the training, validation, and test datasets. They are feed into a pre-trained model for feature extraction.

Implementation & Refinement

When I was transforming each training image into a 4D array, I encountered several errors, including kernel error, memory error, and application killed, in the Jupyter notebook on both AWS machine and my local machine. As a result, I decided to make the following changes to my code:

1. I changed this procedure into batch processing. Specifically, I used batch_size=10000 on the pre-processing steps for the pre-trained neural networks.

2. I moved everything from the Jupyter notebook into python files and ran them directly in the command line.

3. I had different feature extraction python files for training, validation, and testing datasets.

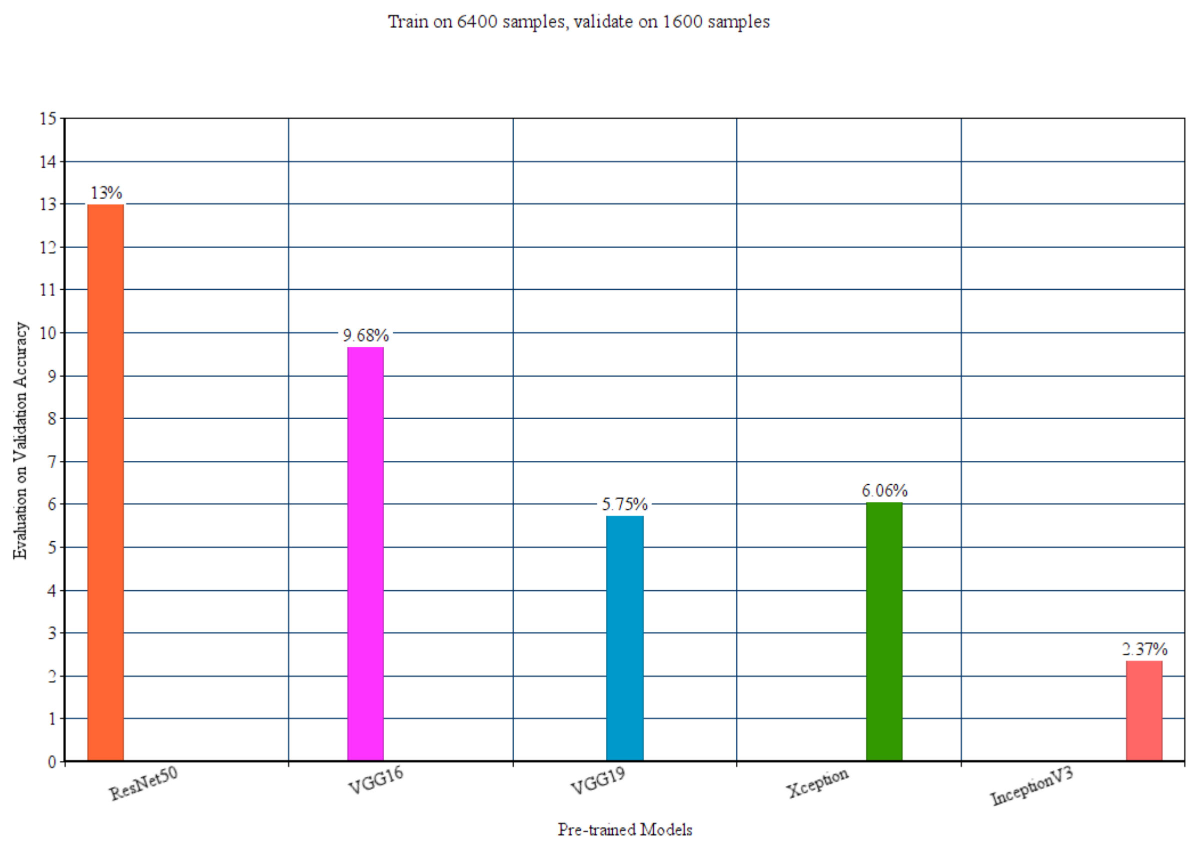
This has several benefits:

1. The program can run on almost any machines without having to worry about memory constraint.

2. If we run out of disk space during feature extraction, which actually happened to me while I was doing feature extraction, we can redo the feature extraction starting at the failed batch.

3. The bottleneck features generated from the pre-trained model for the training, validation, and test datasets take up 176 GB of storage, and since they're processed and exported in chunks, they can be moved around and processed much more easily.

Due to time and storage constraints on retrieving the bottleneck features for each available pre-trained model on Keras, I decided only to run a batch of images, where batch_size=8000, on the available pre-trained models to compare their performances and choose the best one for feature extraction. The following is a comparison chart between different models' training performances on the batch of images:



As a result, I used ResNet50 to extract the bottleneck features for the training, validation, and test datasets.

Once the bottleneck features are generated for the training and testing datasets, I implemented a function to combine those chunks of files into an array of training and validation tuples, which consists of each training image's data and target label and each validation image's data and target label. As for my model architecture, I used a GlobalAveragePooling2D layer and an output layer with sigmoid activation and output size of 9. For compiling the model, I used binary_crossentropy for the loss function and rmsprop for the optimizer. Binary_crossentropy is used because it's more effective when dealing with multi-labels and their values are 1 or 0.

I decided to use GlobalAveragePooling2D layer instead of a fully connected layer because of the success that I had in my previous dog breed project [5]. Moreover, according to an answer from StackExchange, "an advantage of global average pooling over the fully connected layers is that it is more native to the convolution structure by enforcing correspondences between feature maps and categories", and "another advantage is that there is no parameter to optimize in the global average pooling thus overfitting is avoided at this layer. [6]" Overfitting is a common problem in transfer learning; therefore, global average pooling seems to be a better choice.

During training, I save the best model weights so that future prediction and evaluation of the model does not require re-training, which normally take up an hour and a half for 25 epochs across the entire training datasets with validation.

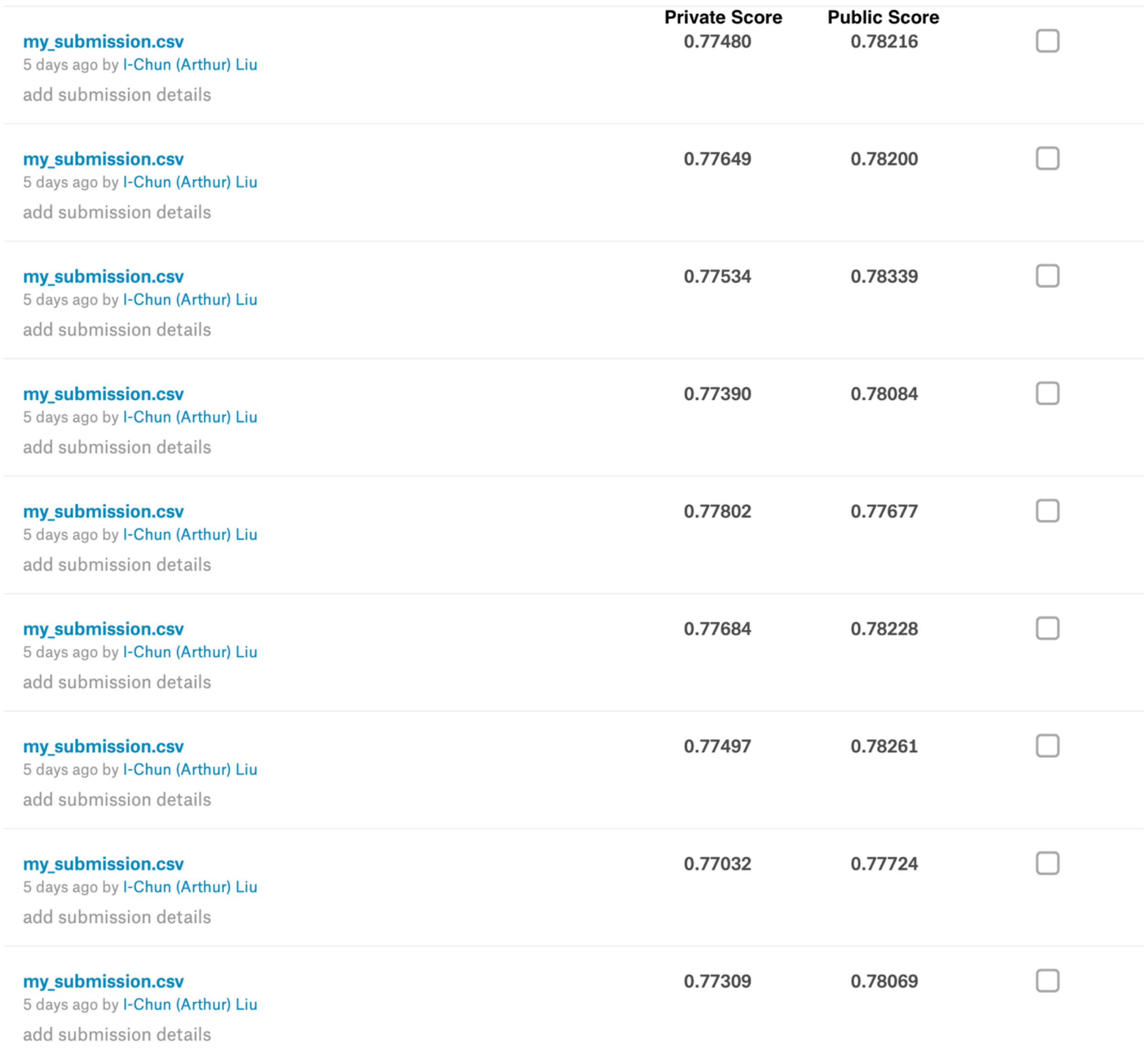
Since the bottleneck features for the training and validation datasets take up 87.8 GB of storage, it's not feasible to load the entire bottleneck features into memory for training. Thus, similar to feature extraction, I load a tuple of training data, training target labels, validation data, and validation target labels at a time for training the model. The model's weights for each consecutive training are preserved, meaning this approach should yield the same results as loading the entire bottleneck features and training the model.

IV. Results

Model Evaluation and Validation

Before I use the model to predict test dataset' labels, I use the validation bottleneck features to evaluate the validation dataset's mean F1 score. It would give us an idea of how well the model performs. With the same parameters in the previous section, this model can score a 0.723 on the validation dataset.

An interesting problem that I found when working on the prediction function for the test dataset was that the same image could appear under similar but different business establishments. It means our model must be robust enough to filter out the noise in the test photos. To overcome this problem, I take the mean of each business' prediction probabilities for the nine target labels and set a threshold to determine whether a label is true. The following is a sample of the mean F1 scores for the test dataset using different thresholds:



	Private Score	Public Score	
my_submission.csv 5 days ago by I-Chun (Arthur) Liu add submission details	0.77480	0.78216	☐
my_submission.csv 5 days ago by I-Chun (Arthur) Liu add submission details	0.77649	0.78200	☐
my_submission.csv 5 days ago by I-Chun (Arthur) Liu add submission details	0.77534	0.78339	☐
my_submission.csv 5 days ago by I-Chun (Arthur) Liu add submission details	0.77390	0.78084	☐
my_submission.csv 5 days ago by I-Chun (Arthur) Liu add submission details	0.77802	0.77677	☐
my_submission.csv 5 days ago by I-Chun (Arthur) Liu add submission details	0.77684	0.78228	☐
my_submission.csv 5 days ago by I-Chun (Arthur) Liu add submission details	0.77497	0.78261	☐
my_submission.csv 5 days ago by I-Chun (Arthur) Liu add submission details	0.77032	0.77724	☐
my_submission.csv 5 days ago by I-Chun (Arthur) Liu add submission details	0.77309	0.78069	☐

By tuning this threshold, I was able to gain a 0.00539 increase in the mean F1 score, which is a good improvement considering the best mean F1 score in this competition is 0.83177. With the threshold set to 0.488, I got a mean F1 score of 0.78339 on the test dataset.

With the amount of time that I have spent and have left on this project, I have attempted three different experiments to try to improve the test results. The first experiment that I tried was using SGD for my model's optimizer. The mean F1 score for the validation dataset scored higher than one above, which was 0.739; however, the test result with tuned threshold only had a score of 0.78319. The second experiment involved replacing the GlobalAveragePooling2D with a MaxPool2D layer. This one had a terrible mean F1 score, 0.60334, for the validation dataset, and its val_loss stopped improving after only processing the first few batches of files, so I did not bother running it on the test dataset.

The last experiment that I tried was replacing GlobalAveragePooling2D with a fully connected layer and adding a dropout layer. The model's training process went smoothly; however, when I tried to load the model's best weights from the file, there was a problem with the input shape size for the model. After googling the problem, I found the problem had to do with a bug in the current version of Keras. Because of my computer's setup, I am using CUDA 10 with self-compiled TensorFlow build, and downgrading Keras version could cause compatibility issues, so I did not continue with this experiment.

The following table contains the validation and test performances for each experiment:

Model	Validation Mean F1 Score	Threshold Used for Predicting Test Labels	Test Mean F1 Score
GlobalAveragePooling2D Dense (9, activation='sigmoid') 'Binary_crossentropy' for loss function and 'rmsprop' for optimizer	0.723	0.488	0.78339
GlobalAveragePooling2D Dense (9, activation='sigmoid') 'Binary_crossentropy' for loss function and 'SGD for optimizer	0.739	0.496	0.78319
MaxPooling2D Flatten () Dense (9, activation='sigmoid')	0.603	n/a due to bad mean F1 score	n/a due to bad mean F1 score

'Binary_crossentropy' for loss function and 'SGD for optimizer		from validation dataset	from validation dataset
Flatten () Dense (256, activation='relu') Dropout (0.2) Dense (9, activation='sigmoid') 'Binary_crossentropy' for loss function and 'SGD for optimizer	n/a due to a bug in Keras' load weight function in current version of Keras	n/a due to a bug in Keras' load weight function in current version of Keras	n/a due to a bug in Keras' load weight function in current version of Keras

Justification

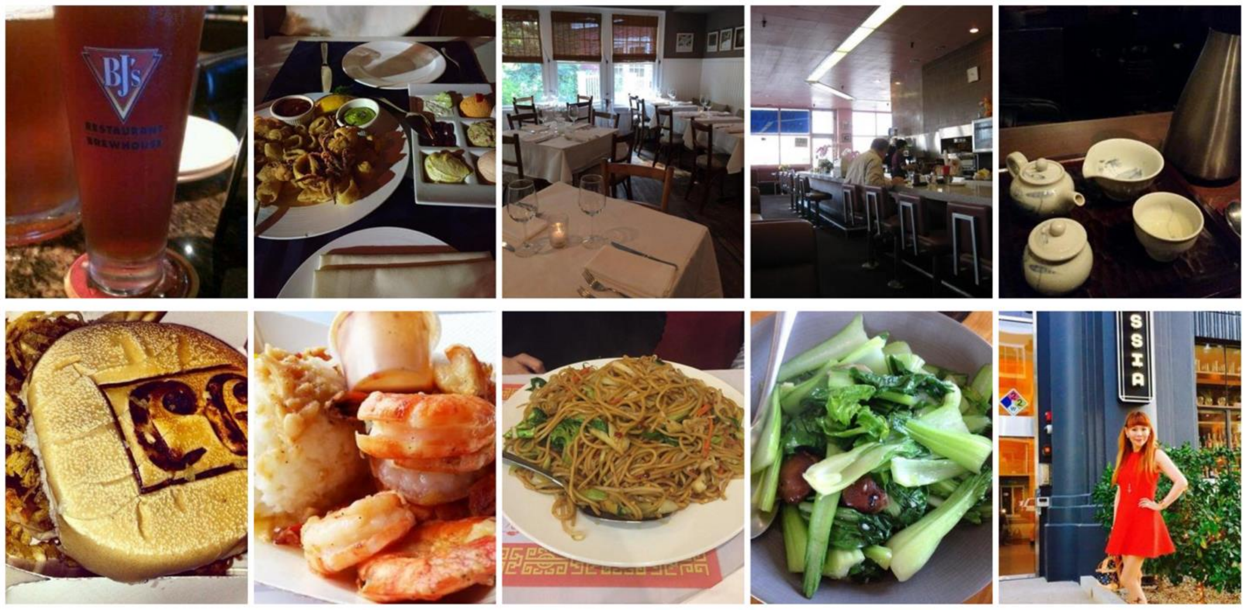
The best model from my experiments uses a GlobalAveragePooling2D layer, and an output layer with input size=9 and activation='sigmoid.' The model compiles with 'binary_crossentropy' for the loss function and 'rmsprop' for the optimizer. The model scores a mean F1 score of 0.723 on the validation dataset and a 0.78339 on the test dataset with a threshold value of 0.488. Compared to the best benchmark model from Yelp, which is 0.6459, this model's performance is good.

Since this competition had already ended two years ago, the first place's best mean F1 score is 0.83177. With my best score of 0.78339, it would place me in rank 122 out of 355 submissions on the leaderboard. The first place's solution utilizes a combination of different pre-trained models' bottleneck features and other business-level features. Although there is only a 0.04838 difference between my score and the first place's score, it would take considerably more time on model training, feature extraction, and experiments to achieve this improvement.

V. Conclusion

Free-Form Visualization

The following pictures are placed under business ID, 019fg, and it is one of the 10000 business IDs in the test dataset:



The model predicts this business to have the following tags: 2 (takes_reservations), 3 (outdoor_seating), 5 (has_alcohol), 6 (has_table_service), 8 (good for kids). It does contain tags: 0 (good_for_lunch), 1(good_for_dinner), 4 (restaurant_is_expensive), 7 (ambience_is_classy).

I would argue that this prediction result is quite similar to how a person would label this restaurant for the following reasons:

1. The restaurant most likely offers reservation service and table service based on the type of environment it has in the top middle picture.
2. Based on the photos, outdoor seating is probably not available in this restaurant. The model may be thrown off by the bottom right photo, which is taken in an outdoor environment and has a person in it.
3. Has_alcohol is predicted correctly based on the top left photo.
4. The tags that are not assigned to this photo are arguably more debatable as to whether they belong to the picture than the ones that are assigned. Therefore, the model seems to make a relatively conservative and good prediction.

Reflection

The following is a step-by-step process from data pre-processing to model predictions:

1. Load *train_photo_to_biz_ids.csv.tgz* and *train.csv.tgz* into DataFrames.
2. Add photos' filename, excluding ones with an underscore, to NumPy array.

3. For each photo's filename, I use *train_photo_to_biz_ids.csv.tgz* to find the corresponding business ID and use this ID with *train.csv.tgz* to find its target labels. The target labels are transformed into an array of labels with a value of 1 being true to a label and a value of 0 for being false to the label.
4. For every batch of 10000 image files, I convert every RGB image to a 4D tensor/array with shape (1, 244, 244, 3) and apply image augmentations with the parameters stated in the Data Preprocessing section.
5. I use ResNet50 with the outputs from step 4 to extract bottleneck features and save them to disk.
6. The above steps are performed for the training dataset and the validation dataset.
7. Load training's and validation's bottleneck features and target labels into an array of tuples. For each tuple, the training bottleneck features, the training target labels, the validation bottleneck features, and the validation target labels are loaded. Each tuple is trained five times, and the array of tuples is also trained five times. In other words, the number of epochs for training the entire dataset is 25. At the end of training for each tuple, a validation mean F1 score is computed for monitoring purposes. The best weights with the lowest val_loss are saved to disk to allow faster experiments with a model.
8. Load validation's bottleneck features and target labels. The same model architecture is defined and loaded with the best weights from step 7. The validation's mean F1 score is then computed and recorded.
9. The following steps are performed for predicting the test labels:
    a. Extract bottleneck features for the test dataset. It is similar to steps 2 to 5.
    b. Load test_photo_biz.csv.tgz into DataFrame.
    c. Load test photo IDs from filenames.
    d. For each bottleneck feature, I use the model with the best weights to predict each image's labels' probabilities. I add these probabilities to a dictionary of arrays. This dictionary contains each business ID's photos' labels' probabilities.
    e. Once all bottleneck features are processed, this dictionary is saved to disk.
    f. For every business ID in the dictionary, I average the corresponding arrays of probabilities and set a threshold to determine which labels should be a 1 or 0. Each business ID's array of labels is transformed into a string of labels. The output is then saved as a CSV file to disk.
    g. Upload the CSV file to Yelp's submission page on Kaggle to obtain the final mean F1 score for this prediction.

All in all, there were two particular challenges in this project. The first challenge was how to get the basic model working. For a while, I did not pre-process the data correctly for feature extraction, so no matter what I changed my model's architecture, the model's val_loss never changed, meaning it failed to learn. The second challenge was to overcome the memory and storage constraints for this challenge's datasets. Regarding memory constraint, it is not possible to load the entire training, or test dataset's bottleneck features into memory because each take up 70.2 GB and 88.6 GB, respectively. As for storage constraint, my main SSD ran out of disk space when processing the test dataset's bottleneck features; therefore, I moved all the datasets and bottleneck features into a new SSD with 256 GB of storage. As I extracted all the features for this challenge, the SSD only had 18.5 GB of free storage. It is crazy to think about the amount of storage we need if we extract bottleneck features for different pre-trained models, which is how the first place's solution is implemented.

Improvement

For my current solution, I would like to continue experimenting with different fully connected layers and see how each performance varies. In addition, I would like to combine different pre-trained models' bottleneck features and investigate business-level features and add an ensemble learning algorithm to improve my model further.

References

[1] Karen Simonyan, Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015.

[2] Yelp Restaurant Photo Classification. Kaggle. https://www.kaggle.com/c/yelp-restaurant-photo-classification#evaluation

[3] Wei-Hong C. How We Use Deep Learning to Classify Business Photos at Yelp. Yelp, October 2015. https://engineeringblog.yelp.com/2015/10/how-we-use-deep-learning-to-classify-business-photos-at-yelp.html

[4] Daniel Y. Introducing the Yelp Restaurant Photo Classification Challenge. Yelp, December 2015. https://engineeringblog.yelp.com/2015/12/yelp-restaurant-photo-classification-kaggle.html

[5] I-Chun Liu. Dog Breed Project. Udacity's Machine Learning Engineer Nanodegree project, August 2018. https://github.com/arthur801031/dog-breed-project/blob/master/dog_app.ipynb

[6] What is global max pooling layer and what is its advantage over maxpooling layer? StackExchange. https://stats.stackexchange.com/questions/257321/what-is-global-max-pooling-layer-and-what-is-its-advantage-over-maxpooling-layer

[7] R. Fisher, S. Perkins, A. Walker and E. Wolfart, "Intensity Histogram", HIPR2. https://homepages.inf.ed.ac.uk/rbf/HIPR2/histgram.htm

[8] Course materials from Udacity’s Machine Learning Engineer Nanodegree. Udacity. https://www.udacity.com/