Multilabel Classification of Restaurants Through User-Submitted Photos

Kaan Ertas, Neval Cam

Stanford University

March 20, 2019

Abstract

In this paper we examine the Yelp Photo Classification Challenge[9] on Kaggle, which presents a dataset of user submitted photos of restaurants and 9 possible labels for each business. The task is to predict, from several photos per business, what subset of labels apply to each business. We tackle this multi-instance, multi-label problem by utilizing convolutional neural networks with different approaches to handle data imbalance and the problem of weakly labeled data. Using these methods that can easily be transferred to other similar problems, we achieve an F1 score of 0.80 – close to the highest F1 score achieved on Kaggle which was 0.83.

1 Introduction

Through the popular online and mobile application Yelp, users can explore and review restaurants for other users to see. Restaurants can provide information about themselves by using labels that are presented under a section called "More Business Info" on Yelp's user interface. Labels like "outdoor seating" or "good for kids" help users find restaurants with desired qualities. Furthermore, users can also upload photos of the business, its food and its ambience. The uploaded photos may provide a useful basis for the business labeling described above. By analyzing the photos uploaded for a business, one may extract the labels assigned. Aside from being an intellectually challenging question, it can aid both restaurants and Yelp to establish truthful, useful and less human-intensive restaurant profiles.

Yelp released a dataset on Kaggle for a competition with these purposes. We analyze photos of restaurants submitted by users from this dataset that was released for Yelp's competition and assign predetermined labels to the restaurants. Different photos of the same restaurant provide different insights, and automated extraction of higher-level information about places from photos is useful for large-scale location-based software such as Yelp, Google Maps, Zomato, TripAdvisor, Airbnb etc.

We propose to use a deep convolutional neural network with a weighted loss function and custom thresholds for each label. The input to our algorithm is an image, and the output is a 9-dimensional vector of values in the range (0,1), each value corresponding to the prediction of a label. Predictions for businesses are then made by aggregating the output labels for the photos associated with the businesses through different approaches described in the paper.

2 Related Work

Multi-instance learning tasks are different in nature to strongly labeled supervised learning tasks. As mentioned by Carbonneau et al. [4], we have multiple bags (in our case, businesses) and instances associated to these bags (images). A straightforward to get labels for each instance, labels of the bags (businesses) are assigned to each instance (image) in that bag. This approach allows the use od conventional image recognition algorithms.

Zhou et al. have proposed[11] two supervised learning algorithms for multi-instance multi-label problem instances, an example which is the business classification problem at hand. The two algorithms MIMLBOOST and MIMLSVM decompose the problem in two different ways and then utilize already existing machine learning algorithms. MIMLBOOST first reduces the multi-instance multi-label classification problem into multiple independent multi-instance single-label learning problems. This is done by turning each learning example (multi-instance bag) $(X^{(i)}, y^{(i)})$, with k output labels and s examples in the bag, into k bags $[(X^{(i)}, y_1^{(i)})]...[(X^{(i)}, y_k^{(i)})]$ where $[(X^{(i)}, y_v^{(i)})] = \{(X^{(i)_1}, y_v^{(i)})...(X^{(i)_s}, y_v^{(i)})\}$. The algorithm then uses MIBOOSTING proposed by Xu and Frank[8], which is a multi-instance single-label learning algorithm that averages over all training examples for a bag and optimizes likelihood for the bags. When predictions are made, the predictions for each label can be collected to produce an overall labeling for the example. MIMLSVM, on the other hand, reduces the problem into several single-instance multilabel problems. The algorithm first performs k-medioids clustering on the multi-instance bags using Hausdorff distance, then uses the same distance metric to represent the bag as a k-dimensional vector of distances. The representation vectors are then used to perform SVM binary classification on each label, which reduces the problem to several single-instance single-label problems. The underlying assumption is that the input vectors for the algorithm can be reliably compared with a spatial distance metric. Predictions are made by aggregating the predictions of the SVM classifiers corresponding to each label.

MIMLBOOST makes two major assumptions: that labels are uncorrelated, and that a multi-instance prediction can be made by averaging all examples in the multi-instance bag. The assumption that labels are uncorrelated may not be warranted in our current problem; for example, in our training dataset, labels "good for dinner" and "takes reservations" have correlation 0.64, while labels "restaurant is expensive" and "good for kids" have correlation -0.57, and most correlation values are greater than 0.3 in magnitude (Fig. 2). MIMLSVM assumes that a spatial distance metric is valid for the examples; this assumption, however is not valid for the input format in our problem, which are images. This hurdle can be overcome by producing vector embeddings for images where a spatial distance metric would be valid, which may be used by the algorithm.

Yosinski et al. [10] demonstrate that earlier futures in a trained deep neural network are not task-specific and can be transferred to different problems, especially in computer vision tasks. Also demonstrated is that the transferability of features is better when the distance between the base and target tasks is small. Simonyan and Zisserman [6] argue that specific very deep and large convolutional neural network architectures perform well on large-scale image recognition tasks, as demonstrated by their success in ILSVRC 2014. Using a specific architecture proposed in the paper, known as VGG19, and using pre-trained weights on the ImageNet database can be a valid approach to our problem since the base task (classification of objects in the ImageNet database) is close to the target task of labeling images, which is largely dependent on the objects that appear in the image.

3 Dataset and Features

The dataset Yelp provided has images submitted by users for 2000 restaurants. The dataset consists of a map of business IDs to photo IDs, a map of business IDs to business labels, and images for training and test sets[9]. The restaurants (but not individual images) are each labeled with a subset of the following tags:

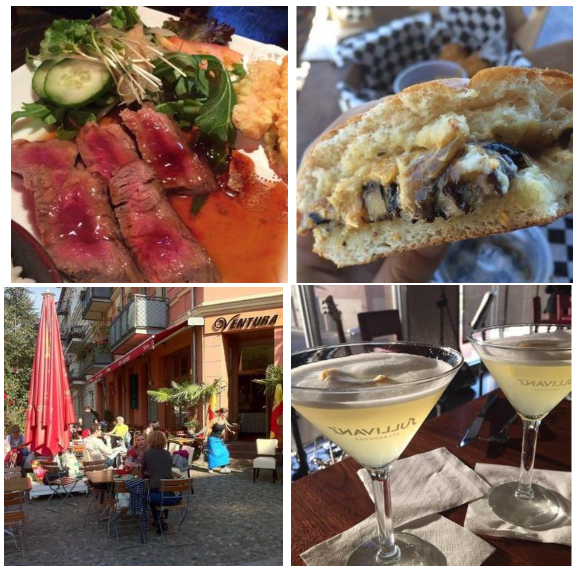
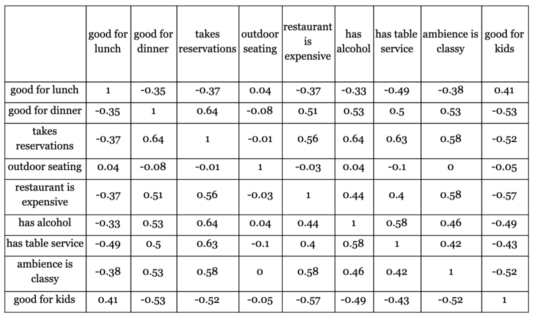


Figure 1: Sample images from our dataset, with business labels "good for dinner", "good for lunch", "has outdoor seating" and "has alcohol" respectively



	good for lunch	good for dinner	takes reservations	outdoor seating	restaurant is expensive	has alcohol	has table service	ambience is classy	good for kids
good for lunch	1	-0.35	-0.37	0.04	-0.37	-0.33	-0.49	-0.38	0.41
good for dinner	-0.35	1	0.64	-0.08	0.51	0.53	0.5	0.53	-0.53
takes reservations	-0.37	0.64	1	-0.01	0.56	0.64	0.63	0.58	-0.52
outdoor seating	0.04	-0.08	-0.01	1	-0.03	0.04	-0.1	0	-0.05
restaurant is expensive	-0.37	0.51	0.56	-0.03	1	0.44	0.4	0.58	-0.57
has alcohol	-0.33	0.53	0.64	0.04	0.44	1	0.58	0.46	-0.49
has table service	-0.49	0.5	0.63	-0.1	0.4	0.58	1	0.42	-0.43
ambience is classy	-0.38	0.53	0.58	0	0.58	0.46	0.42	1	-0.52
good for kids	0.41	-0.53	-0.52	-0.05	-0.57	-0.49	-0.43	-0.52	1

Figure 2: Correlation matrix for labels

1) good for lunch
2) good for dinner
3) takes reservations
4) outdoor seating
5) restaurant is expensive
6) has alcohol
7) has table service
8) ambience is classy
9) good for kids

The challenge is that photos do not have specific labels attached to them, but the businesses do. The data set also has different numbers of photos for restaurants, and there are duplicate photos. However, the duplicate photos will not affect the training process, and removing duplicates is a much more computationally difficult task than processing the duplicates. Both a blessing and a curse of this task is that different photos will contain information regarding different labels, eg. a photo of food gives us different information than a photo of the restaurant's entrance.

The dataset originally had 2000 businesses and ∼234,000 images for the training set. Due to lack of computational power and time, we are training our model on 1000, validating on 32 and testing on 32 businesses with 32 randomly selected images for each. This results in data sets of 32,000, 1024 and 1024 images for training, validation and testing respectively.

In order to prepare the data for our models, we preprocessed each image in our dataset. First, we reshape them to 224 by 224 pixels with 3 color channels giving an array of (224, 224, 3) for each image. We then normalize each channel to the interval (0,1). This helps the algorithm learn faster and allows for finer tuning of the learning rate hyperparameter.

4 Methods

4.1 Multi-Instance Learning Considerations

Our dataset is weakly labeled, as images do not have associated labels but the corresponding businesses do. Therefore, as mentioned by Carbonneau et al. [4], we assign all labels of the business to its images. The advantage of this approach is that it allows for more automated training and the use of conventional computer vision algorithms. The downside is that it introduces error and noise to the labeling. Consider a business that is labeled both "good for kids" and "has alcohol". An individual photo from this business may show an alcoholic drink and not have elements that suggest a "good for kids" environment. However, this photo gets both labels under the current approach.

In the final prediction step, we use two different approaches to aggregate the predictions associated with photos of the same business.

Mean: For each business, we take the arithmetic mean for each label across it associated photos. This approach has the underlying assumption that a business is represented equally by all its photos. This approach has a regularizing effect for mispredictions, but ignores the problem above. Since not every photo will highlight all features of a business, a sparse representation of a feature will be missed when taking the arithmetic mean.

Max: For each business, we use the maximum values of sigmoidal activations across all photos of the business. This approach takes better advantage of the fact that photos of the same business can highlight different aspects of the business. For example, consider a business that has alcohol, but only one image associated with it shows alcohol being served. While the previous approach will incorrectly predict that the label does not apply, this approach will predict correctly. A disadvantage of this approach is that it is more sensitive to individual misclassifications, and does not benefit from the regularizing effect of 32 images per business in the dataset.

4.2 Loss Function

This is a multilabel, multiclass classification task, since every restaurant has a subset of the 9 possible labels. Treating the problem as 9 separate classification problems, we can use a vectorized binary cross entropy loss

Label	Frequency
good_for_lunch	0.29
good_for_dinner	0.54
takes_reservations	0.56
outdoor_seating	0.51
restaurant_is_expensive	0.31
has_alcohol	0.68
has_table_service	0.73
ambience_is_classy	0.34
good_for_kids	0.57

Figure 3: Frequency of labels in training dataset

for each label:

$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N} \cdot [y^{(i)} log(\hat{y}^{(i)}) + (1-y^{(i)}) log(1-\hat{y}^{(i)})]$$

4.3 Weighted Loss Function

Some data exploration reveals that the dataset is imbalanced, and that some labels appear much less frequently than others.

Due to the imbalance, training algorithms with equal weighted loss for each label results in poor performance in the underrepresented labels. Therefore we use a weighted loss function:

$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N} w_i \cdot [y^{(i)} log(\hat{y}^{(i)}) + (1-y^{(i)}) log(1-\hat{y}^{(i)})]$$

Where w is an n-dimensional stochastic vector, with each entry corresponding to the weight of the label. In order to emphasize the learning of underrepresented labels, we assigned a higher value to weights whose corresponding labels appeared less frequently in the training dataset. After experimenting with different weights, we found the following to work best:

[0.17, 0.07, 0.09, 0.11, 0.18, 0.08, 0.08, 0.14, 0.08]

These weights reflect the imbalance of the data, with labels indexed 0, 5 and 7 being the most infrequent and hardest to classify.

4.4 Custom Thresholds

Another way to combat underrepresented data is to use different thresholds for each label when making the final predictions.

This problem is apparent when we look at the values output by the baseline-CNN for the first label, almost all of which are in the range (0,0.35). Using a threshold of 0.5 for this label causes all predictions for this label to be 0, which neglects the variations among activations that are squeezed into an interval but nonetheless vary greatly when normalized.

Since we are using a sigmoid activation at the last layer and using binary crossentropy loss, we can assume that the model will learn to output higher values for labels whose ground truth is 1 (label applies). Thus, if a label appears in x% of the training data, then we can expect the top x% of the training prediction activation values output by the model for that label to correspond to photos that possess that label. We propose the following algorithm to set thresholds:

```
Result: Returns array of custom threshold values
        for each label
Initialize empty array of thresholds;
for each label l do
    Determine the frequency f of l in the training
      data sets;
    Predict the training data set;
    Take predictions for l;
    Sort the predictions for l in ascending order;
    Determine the (1-f)*100 percentile of the sorted
      predictions ;
    Append the percentile to the thresholds array
end
```

Algorithm 1: Custom Thresholds

Although there will not be a one-to-one correspondence between the data points whose outputs are at the top x% and the data points whose true values are 1 for that label, this algorithm still provides a good heuristic for the threshold value that needs to be applied.

4.5 Evaluation Metric

For each label, we apply the F1 metric, which is the harmonic mean of precision and recall:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

As the overall evaluation metric, we use the mean F1 score computed over all 9 labels. This is the metric used by Kaggle for their challenge leaderboard.

4.6 Baseline Model

As a baseline, we created a simple convolutional neural network (b-CNN) with 2 convolutional layers and 3 fully connected layers, followed by a sigmoid output. The model was trained on the training set for 10 epochs, with an Adam optimizer and a minibatch size of 512. This minibatch size was preferred since stochastic gradient descent would not be taking advantage of vectorization, but we did not have sufficient memory and computational speed to run gradient descent on the entire batch.

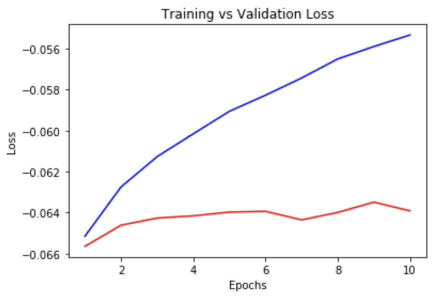


Figure 4: Training and validation losses of VGG19 with weighted loss (note the scale on the y-axis)

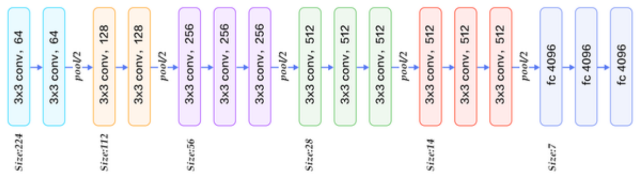


Figure 5: VGG19 architecture

On the test set, to represent the business predictions, we used the arithmetic mean of the outputs for the 32 images corresponding to the business, and used a threshold value of 0.5 for each label.

4.7 Transfer Learning with VGG19

Instead of training a large CNN from scratch, we employed a pre-trained state of the art model, VGG19. The weights were trained on the ImageNet dataset, which includes copious images of food and room settings. This makes the weights appropriate for our task. To use transfer learning, we removed the final softmax layer and inserted a fully connected 9-neuron sigmoid layer. We froze the training on all layers except the last 3 fully connected layers. This allows at once to use the feature extraction properties inherited by the pre-trained weights, and to fine-tune the last few layers to our specific dataset. We further trained the VGG19 network for 10 epochs, with and without the custom loss weighting. For each resulting network, we predicted the training dev and test sets using the mean and max aggregation techniques discussed above.

5 Results

In this section, we will go through our results for each experiment presented in section 4 and discuss the results.

5.1 Baseline Model

b-CNN resulted in a mean F1 score of 0.59. The F1 score was 0 for the label "good for lunch", due to the absence of positive predictions. The model also

	TRAIN		DEV		TEST	
	Mean	Max	Mean	Max	Mean	Max
b-CNN	0.48	**0.72**	0.40	**0.72**	0.35	0.62
VGG19	0.70	0.69	0.59	0.68	0.68	**0.65**
VGG19-CT	0.86	0.66	**0.74**	0.68	**0.80**	0.63
VGG19-CT-CL	**0.90**	0.67	0.72	0.68	**0.80**	0.63
VGG19-CL	0.75	0.70	0.58	0.70	0.63	**0.65**

Figure 6: Mean F1 scores for various architectures and methods

performed poorly on label "restaurant is expensive", with an F1 score of 0.22. This signals the possibility that those two labels are harder to train, and that a threshold of 0.5 might not be appropriate for all labels.

5.2 Transfer Learning with VGG19

For gradient descent, having higher learning rates causes our loss to converge to a certain high value after some number of epochs and having lower learning rates slows the learning process. After experimenting with several learning rates, we discovered 0.000001 works best for our case. With no weighted loss and a 0.5 threshold, we achieved a mean F1 score of 0.68 using the mean aggregation method.

5.3 Weighted Loss Function

Applying our weighted loss function with transfer learning with VGG19 (with the mean technique) resulted in training F1 score to increase by 7%. However, we also observed that F1 scores for our validation and test sets dropped when we used the weighted loss function (again with the mean technique). This shows us that the weighted loss function is prone to overfitting our training set when using the mean aggregation technique.

5.4 Custom Thresholds

Applying our custom threshold algorithm, we observed 18% increase in our mean F1 score for test set when we did not apply the weighted loss function. With our weighted loss function this increase went up to 27% since initial application of weighted loss decreased the F1 score. Since mean F1 scores for training, validation, and test sets all increased proportionally, we can conclude that we are not facing the overfitting problem we had when weighted loss function was applied. Thus, this method was drastically more successful than the weighted loss function method.

6 Conclusion/Future Work

We have tackled Kaggle's Yelp Photo Classification Challenge by employing transfer learning with a VGG19 convolutional neural network. Due to the multi-instance and multi-label nature of the problem, we used a weighted loss function that reflected the data imbalance, custom thresholds for each label and two different aggregation methods for weakly labeled data. Our algorithm that used a weighted loss function, custom thresholds and an arithmetic mean photo aggregation technique achieved a mean F1 score of 0.80, performing close to the winner of the Kaggle competition.

With more time and computational resources, possible areas of further research could be simply training the algorithm on the entire dataset, expanding our dataset using various data augmentation techniques(eg. flipping images), applying transfer learning with different architectures(eg. Inception-Resnet-V2), employing attention models, and modularizing our approach to recognize features in photos (eg. using machine learning to recognize features such as the presence of bottles, which may be a strong indicator of the label "has alcohol").

7 Contributions

Both authors took equal part in the formulation of the problem, data preprocessing, algorithm implementation and report writeup processes. Special thanks to Sarah Najmark for her guidance and help.

8 Code

GitHub Repository Link:
https://github.com/kaanertas/yelp-photo-classification

References

[1] arXiv:1802.04712

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

[3] Chollet, François. "Keras." (2015).

[4] Marc-André Carbonneau, Veronika Cheplygina, Eric Granger, Ghyslain Gagnon, Multiple instance learning: A survey of problem characteristics and applications, Pattern Recognition, Volume 77 (2018), Pages 329-353.

[5] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[6] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[7] Tsoumakas, Grigorios and Katakis, Ioannis. (2009). Multi-Label Classification: An Overview. International Journal of Data Warehousing and Mining. 3. 1-13. 10.4018/jdwm.2007070101.

[8] Xu, Xin, and Eibe Frank. "Logistic regression and boosting for labeled bags of instances." Pacific-Asia conference on knowledge discovery and data mining. Springer, Berlin, Heidelberg, 2004.

[9] Yelp Restaurant Photo Classification | Kaggle, www.kaggle.com/c/yelp-restaurant- photo-classification.

[10] Yosinski, Jason, et al. "How transferable are features in deep neural networks?." Advances in neural information processing systems. 2014.

[11] Zhou, Zhi-Hua, et al. "Multi-instance multi-label learning." Artificial Intelligence 176.1 (2012): 2291-2320.