Concept Learning on Yelp Restaurant Classification

Shenxiu Liu
Physics Department
Stanford University
shenxiu@stanford.edu

Haoming Li
ICME
Stanford University
haoming@stanford.edu

Abstract

We are using Yelp restaurant photos from Kaggle to classify restaurants into 9 different concepts. In this project, we tackled a multi-label classification problem for a "group of images". Also, labels here correspond to not the objects but some higher level concepts. We solved this problem by using transfer learning approach from pre-trained models. Performance of various models are studied and compared in our project.

1. Introduction

For commercial applications of image classification, it's not always a vanilla classification and identification of objects. Sometimes we want to do more complex tasks on images. One very common problem is to use images as an intermediate layer to understand concepts. Such as on yelp, we can see a lot of user-posted photos of a specific restaurant on customer side and as customers we can have pretty good ideas about this restaurant, such as if this is expensive or not, the seats are indoor or outdoor, *etc.*. It will be useful if we can extract these concepts out of photos automatically and directly label it even before customers digest it themselves. Furthermore it can be followed with applications like restaurant searching and/or ranking. Our problem is then from a Kaggle competition dealing with this multi-concept learning of restaurants based on their photos uploaded by Yelp users.

This application is particularly interesting because people like to share photos on Yelp. Using these photos human beings can easily get a lot of information of this restaurant. But the throughput for this process is extremely low. We have a large database here and it will be very useful if we can digest these photos to make a description automatically in a fast and high-throughput manner. The information then can be easily integrated into recommendation systems and search engines based on features. Our high level end-to-end pipeline is shown in Fig. 1.

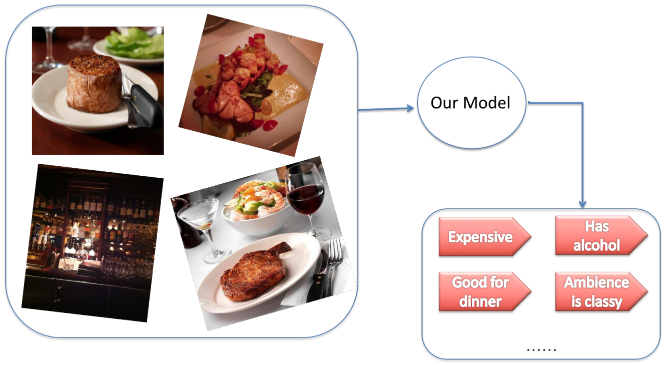


Figure 1: Our model pipeline in high level. We have a series of photos for a particular restaurant, then send them through our model, we'll end up with a list of labels for this restaurants indicating its features.

2. Problem Statement

We are given around 230k training photos associated with 2k restaurants. Each restaurant is labeled as one or more of these classes:

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

Our goal is to classify any restaurant into these 9 classes, with multi-label allowed, given the photos associated with the restaurant. The test set is unseen with around 240k photos associated with unknown number of restaurants. (There are 10k restaurants we are going to predict, but Yelp claims most of these "restaurants" are actually made up to avoid hand labeling.)

3. Evaluation Metric

The final test will be on an unlabeled test set for the competition. We also construct a validation test by splitting the labeled data into training part and developing part in order to fine tune our hyper parameters. Our evaluation metric is mean-F1 on whole dataset, which is the mean value of per-instance F1. Per-instance F1 score is defined as the harmonic mean of precision and recall on these 9 labels for this sample, i.e.,

$$F1 = \frac{2pr}{p+r}.$$

where

$$p = \frac{tp}{tp+fp}, r = \frac{tp}{tp+fn}$$

Here true positives tp is the number of labels that are both true and predicted for this instance. False positives fp and false negatives fn follow the same way.

4. Approach

Since the labels are on restaurants but data we want to utilize are photos, we separate the problem into two phases following an intuitive approach:

1. Firstly we propagate labels for restaurants to photo-level, as shown in Fig. 2. To classify these photos, we utilize convolutional neural networks to build a multi-label classifier. Because these labels are relatively harder to learn, we use transfer learning from pre-trained models with loss function as the cross entropy of logistic regression output and the label, defined as

$$\text{loss} = -\sum_{\text{data}} \sum_{i=0}^{8} (y_i \log(p_i) + (1-y_i)\log(1-p_i)) \tag{1}$$

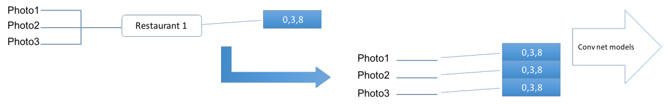


Figure 2: If we have a restaurant associated with photos shown in the picture with label 0,3,8, we will label these photos as the restaurant label and send them through convolutional neural network.

2. For the second phase from image to business, we use the following strategy:

   By collecting all probabilities of the belonging photos of each restaurant, we compute the mean for each restaurant to make it as the probability of this restaurant to be in this class, as shown in Fig. 3.

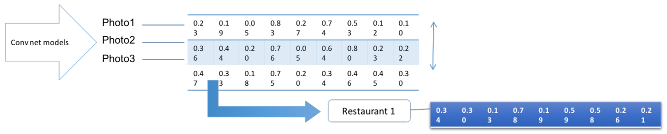


Figure 3: Collecting probabilities back to business level, as shown in the example. We compute the mean value of photo-level probabilities into business.

   We further make decisions based on these probabilities. Note that as our data may be imbalance in some classes, so the best cut for predicting as positive isn't necessarily 0.5. Especially as we are working on optimizing F1, which means that we would like to have a balanced precision/recall, it is clear that if we make thresholds to predict as positive being 0 we can have a low precision and 1.0 recall. However on the other hand if we make the threshold close to 1.0 the recall will be zero but precision is relatively high. Therefore we make 9 more parameters for the threshold to optimize F1 on training set. We find these values by iteratively using binary search between 0 and 1 on each class. The prediction process is visualized in Fig. 4.

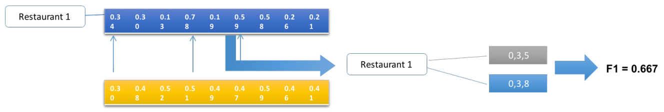


Figure 4: This is showing how a single prediction is made. We train the threshold values from training set, and then get probabilities from test set. Based on those training thresholds, we can set up this prediction: a class is labeled as positive if and only if the score for this class exceeds the threshold value of this class.

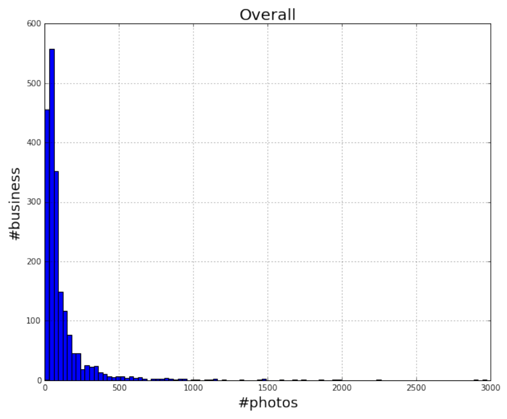
To train the photo-level classifier we reshape all photos into 256×256, then randomly pick a 224×224 crop with random mirror, as usual processes in ImageNet training. At test time we use 5 224×224 crops at corners and middle together to make the prediction.

5. Empirical Results

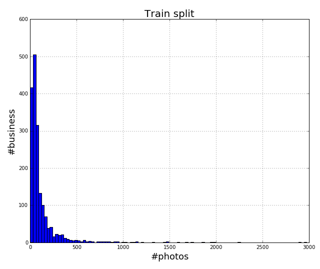
5.1. Data exploration

The first step is always to look at data. The histogram of photo number vs. business number looks like Fig. 5a. Although restaurants with less than 50 photos dominate, it's still long tail, and each restaurant has a relatively large number of photos.

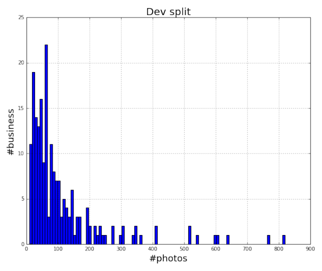
We make a train-dev split on the training data based on business id (not on photo), 90% on training set and 10% on dev set. We also show the distribution of train-dev split in Fig. 5b,5c. It has a quite similar shape.



(a) overall

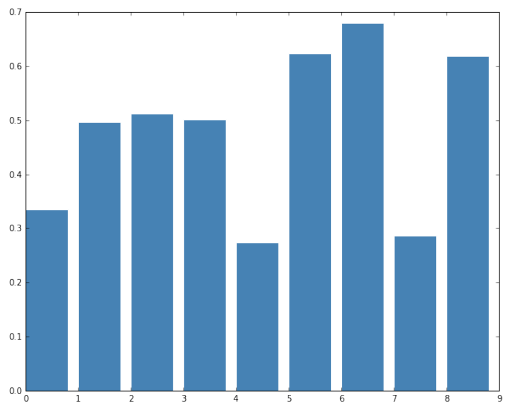


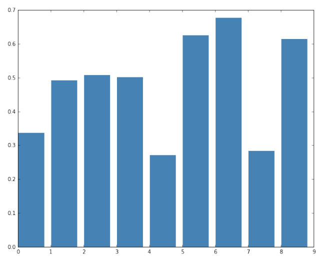
(b) train split



(c) dev split

Figure 5: The histogram of number of restaurants that have specific number of photos.


(a) overall


(b) train split

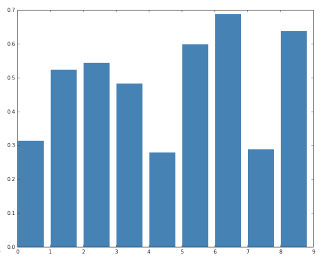

(c) dev split

Figure 6: The ratio of positive samples in different classes, business level.

For all 9 classes the positive ratio is shown in Fig. 6. It doesn't indicate too severe imbalance problem therefore we are not specifically worrying about that.

Correlation of different classes is shown in Fig. 7. We found that some classes are quite mixed, so we choose to use a single loss function rather than training 9 different classifiers.

5.2. Baselines

We are using two different trivial baselines as our starting point.

- Random baseline: We compute the positive portion of all nine classes in training set, and in evaluation phase we randomly assign labels for each class by the probability distribution collected in training set for each sample. This gives us a F1 at 0.485 on dev set.
- Same-class baseline: We still compute the positive portion of all nine classes in training set. But this time instead of assigning all into random, we compare accumulation of each class and find all classes with a positive portion larger than 0.5, then we assign every sample into those classes. This strategy gives us a F1 at 0.608 on dev set.

The same-class baseline is surprisingly better than random baseline and very robust to train-dev splitting. It might be related to the fact that classes have a high correlation and guessing constants makes a favorable precision and recall in a lot of cases, leading to a very high overall performance.

5.3. Learning Curve

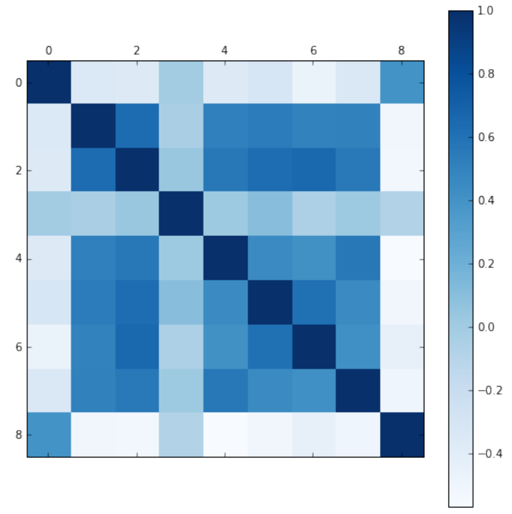
For the training process on photos since there's no accuracy metric that can directly represent the final F1, we only look at loss function in this case. And we look at both training and validation error, in the end we choose the training iteration that has best validation accuracy for the evaluation.

Specifically, as we are using pre-trained model, but the last layer is random initialized by xavier initializer, we use a 10 times larger learning rate in the last layer and use 1×10^{-4} as base learning rate for the whole network, such that we can update all parameters a bit, and won't affect too much to the pretrained parameters.

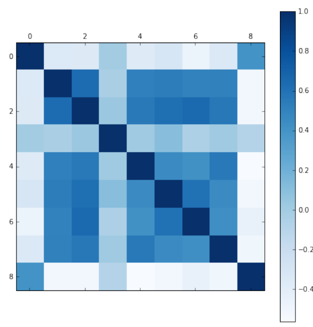
For the gradient solver we choose Adam after some experiments, the momentums are set to be

$$\beta_1 = 0.9, \beta_2 = 0.999.$$

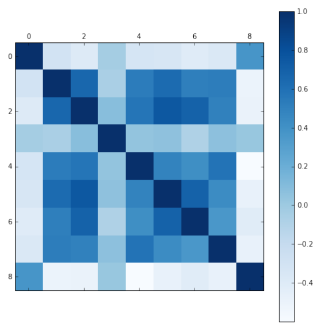
And the learning rate policy is always chosen to be 'inv' in Caffe solver.



(a) overall



(b) train split



(c) dev split

Figure 7: The correlation between different classes, business level.

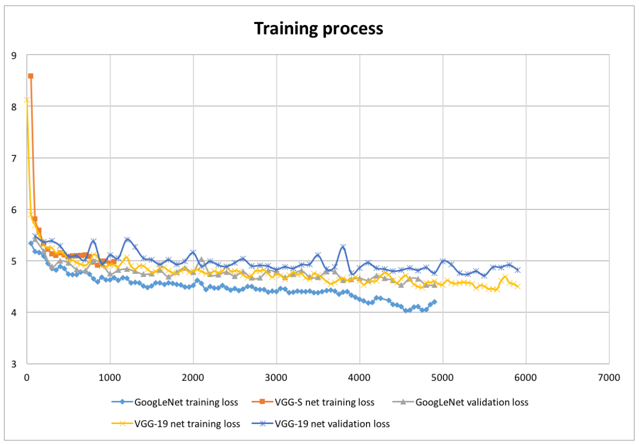


Figure 8: Losses in GoogLeNet and VGG-nets. In figure we can see that VGG net not only trains slower, but also saturates to a higher dev loss. Also, GoogLeNet shows a larger model capacity, which leads to a lower training loss, too.

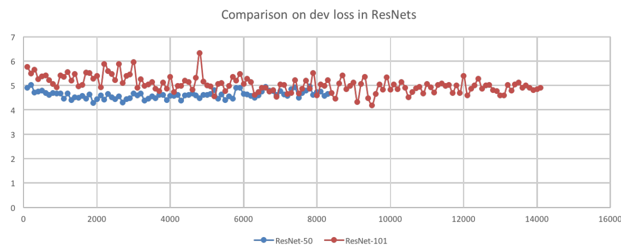


Figure 9: The dev loss in ResNet-50 and ResNet-101. Although ResNet-101 is reported to have a better performance in ImageNet, the model is harder to train and the limitation in graphics memory leads to an extremely small batch size (only 16 even on 8 Titan-Xs, thanks to Sherlock cluster, which makes our experiment possible). Training ResNet-101 is practically intractable. Also it's clear that ResNet-50 starts to overfit after less than 4k iterations from the figure.

5.3.1 comparison between GoogLeNet and VGG-net

We train GoogLeNet [1] and VGG-nets (both VGG-S and VGG-19) with pre-trained models from Caffe Model Zoo (GoogLeNet parameters are from BVLC implementation). We find GoogLeNet can give us a significantly better validation performance, see Fig. 8. Also GoogLeNet trains much faster because of having less parameters. So we abandoned VGG-nets and only use GoogLeNet in further investigation.

5.3.2 comparison between ResNet-50 and ResNet-101

We train ResNets using pre-trained parameters released recently, in both 50-layer version and 101-layer version. For the comparison, we show the dev loss in both model in Fig. 9.

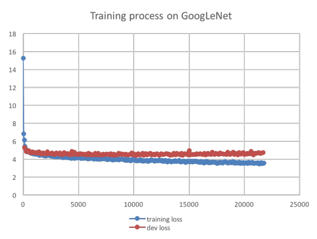
From the figure, we can see that ResNet-50 has a much faster convergence and needs significantly less iterations to reach optimal validation accuracy. And its optimal validation loss is about the same as GoogLeNet.

5.3.3 architecture choice

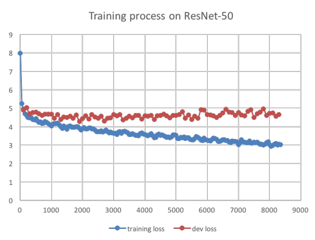
We choose both GoogLeNet and ResNet-50 as our architectures. The learning curve of these models are shown in Fig. 10.

5.4. Evaluation

In this part we report the training and validation F1 for different models, in both photo-level and business-level.



(a) GoogLeNet



(b) ResNet-50

Figure 10: The training and dev loss in architectures we choose for prediction and evaluation. GoogLeNet shows almost no overfitting during training, however ResNet-50 shows an overfitting after several thousands iterations. The optimal dev loss in these two different models are about the same.

	F1 on training set	F1 on dev set
Business level, GoogLeNet	0.8257	0.8092
Business level, ResNet 50	0.8115	0.8010
Photo level, GoogLeNet	0.7882	0.7203
Photo level, ResNet 50	0.7806	0.7266

Table 1: Training and dev performance: naive thresholds

5.4.1 Without threshold training

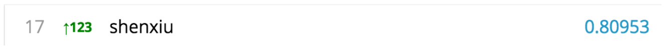
We first try using 0.5 as the threshold value for all classes, both for photo level and business level. At photo level, the F1 scores are computed as if the propagated labels of all photos are ground truth and each photo is an instance. For business level the approach we introduce in figure 3 and 4 is used. The performance in mean F1 score on both training set and test set is reported in table 1.

There are a couple of interesting observations. The business level performance is what we focus more on, and they turn out to be pretty good (compared with baselines) on both training and test set, indicating that overfitting here is not that severe. Two models give about the same performance.

The photo level performance, however, is less impressive. Our interpretation is that our process of propagating restaurant labels to all associated photos can introduce noise, e.g, some photos may just represent a subset of the restaurant labels. However this kind of noise is canceled out to some extent by our combining approach, leading to a significantly better performance on restaurant level. Furthermore we observe a more severe overfitting on photo level. The overfitting comes from the convolutional neural networks and our approach mitigates the effect of this kind of overfitting has on our restaurant level predictions.

	F1 on training set	F1 on dev set
Business level, GoogLeNet	0.8436	0.8151
Business level, ResNet 50	0.8474	0.8214
Photo level, GoogLeNet	0.8016	0.7334
Photo level, ResNet 50	0.7929	0.7364

Table 2: Training and dev performance: trained thresholds



17 ↑123 shenxiu 0.80953

Figure 11: Test performance and ranking of the challenge.

5.4.2 With threshold training

We would like to further justify our approach of optimize threshold values for making prediction. Following the approach introduced we report the same set of performance in table 2. We observe improvement in almost all performance scores: not only the training F1 is improved (which is for sure since that's how we optimize the thresholds) but the validation F1 is improved as well. Therefore we believe that this approach can indeed help us do a better job in labeling.

ResNet gets more benefit from this approach than GoogLeNet does and outperforms the latter. And other patterns and observations we made previously also hold here. For testing we will use threshold training and focus on ResNet-50 model. We expect to obtain a test F1 score close to our best performance on validation set, i.e, 0.82.

5.4.3 Model ensemble and test performance

We run 3 different ResNet-50s independently. We name them ResNet-2000, ResNet-2100 and ResNet-2700 after the number of iterations that hits the optimal validation lost. We use ensemble of the predictions of the 3 models to be our prediction on test set. Our submitted predictions got a F1 score of 0.8095 on the test set, making us ranking the 17^{th} place for this challenge among about 200 competitors. Note that this ensemble gives us a 0.1% improvement on test score compared to a single model prediction. This test performance is close to our expectation (0.82). Some of our prediction examples are shown in figure 12

6. Discussion and Conclusion

We designed a series of approaches to take the challenge of doing multi-class labeling of restaurants given their photos. Our simplification and transformation of the problem, including propagating labels, combining scores and optimizing thresholds perform well in practice. Our core models are based on transfer learning and we tried different architectures. Some of them indeed worked as expected and

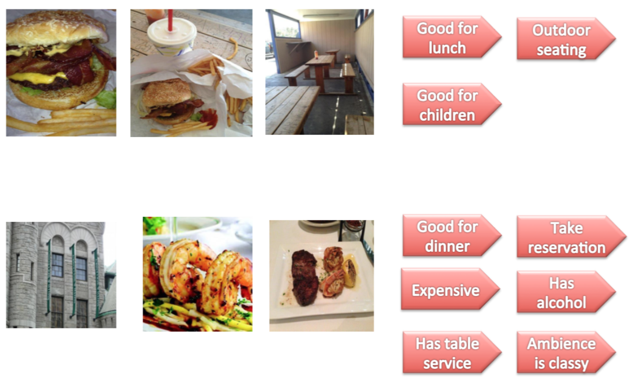


Figure 12: Some prediction examples: photos of restaurant and added labels.

became the engine of our model. The performance on test set we get significantly outperformed the baselines and is close to the best result ever reported.

There are some topics which we didn't dive into but can be very interesting. One is to visualize the neural networks and see what kind of features/patterns these networks learned, since the concepts represented by the training labels are more abstract than some classic image classification problems. Another thing is the possibility to refine the propagated photo labels with trained network and further improve the business level performance, which is an iterative process and can be computationally very expensive.

References

[1] Szegedy, Christian, et al. ”Going deeper with convolutions.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015. 4

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