Using Convolutional Neural Networks and Transfer Learning to Perform Yelp Restaurant Photo Classification

Diveesh Singh
Stanford University
450 Serra Mall, Stanford, CA 94305
diveesh@stanford.edu

Pedro Garzon
Stanford University
450 Serra Mall, Stanford, CA 94305
pgarzon@stanford.edu

Abstract

For the Yelp Image Classification Kaggle Challenge, we use a modified VGGNet to make predictions on 9 specific attributes of restaurants based on images by performing transfer learning. We explore two approaches to making these predictions: a naive model that assigns the attributes of the restaurant to each picture, and a more sophisticated method called multiple instance learning. With our naive model, we achieved a mean F1 score of 0.533*, while our MIL model achieved a mean F1 score of* 0.618*. We then explore and evaluate other past methods attempted for this challenge and note various methods of improvement on our current models.*

1. Introduction

We will be attempting to classify restaurants based on images that various Yelp users have posted. Yelp is always looking for ways to use the pictures people post effectively, and now based on these pictures, it is possible to assign attributes to various restaurants. Essentially, this means using business images to automatically capture metadata. This is useful because it allows Yelp to make better recommendations to users about which restaurants they might like by having less manual text input from the user. Our plan involves a Convolutional Neural Network (CNN) in the form of a modified VGGNet pretrained on ImageNet dataset. We then perform transfer learning using the dataset provided by Yelp of its various restaurants and introduce a custom use of Multiple-Instance Learning for the challenge.

We are trying to predict attribute labels for restaurants based on user-submitted photo; this is a variant of the image object detection problem since we'd like to take note of multiple objects that might indicate a high likelihood of having a certain attribute. The attributes consist of: Good for Lunch, Good for Dinner, Takes Reservations, Outdoor Seating, Restaurant is Expensive, Has Alcohol, Has Table Service, Ambiance is Classy, Good For Kids. Each restaurant has an arbitrary number of photos associated with it, and can be assigned multiple attributes. Our problem involves looking at the pictures associated with a restaurant and using them to assign attributes. Another added challenge is the subjectivity of the material since even human readers could differently label an image for the given attributes. We will present the approach of tackling the problem by training a convolutional neural network to do the full prediction from input to output. We'll be giving as input batches, or in our terminology "bags", that correspond to each business and attempt prediction and training on a business given a bag of photos. Our output is a 9-label binary classification.

2. Previous Work and Background

The state of the art in CNNs gets pushed ever so slightly with each passing year with more complicated and computationally demanding models. As of now, the leading model for the ImageNet Challenge is the Inception-v4 model architecture that was able to achieve 3.08% top error on the ImageNet challenge by using 75 trainable layers [12], beating out the ResNet and GoogleNet that were the previous reigning champs in image classification. However, the main drawback of latest state of the art is ever increasing train-time to get the models to convergence. The model that has been widely used in the past two years for application problems has been the VGGNet architecture, which won the ImageNet Challenge in 2014. It benefits from having less layers than the current state-of-the-art while still having significantly good with an error of 7.3% on accuracy [11]. VGGNet models are also widely distributed on the Internet for various deep learning frameworks, making it an ideal CNN model to start out with.

The original blog post announcing the Yelp Challenge gave some initial results as well. All proceeding measures of accuracy will be based on Mean F1 Score. A complete random guess giving each label equal probability gave an

accuracy of 43.5%. A better approach using color distribution and compares the color distribution for each image in comparison to the average of color distributions for images had the label marked as positive or negative. This gave an accuracy of 64.6% [14].

User uploaded submission scripts shared on the Kaggle site provided more insight and understanding to the challenge. For instance, the total average number of labels per of picture in the training set is 4, and predicting labels with means in proportion to each class can give a baseline result of 60.9% accuracy on the test set [5]. This simple approach without any machine learning will serve as our baseline comparison to improve on with our CNN model.

Evidence that CNNs could prove useful for the problem come from a Kaggle script that uses a CNN as a feature extractor. Here, a deep CaffeNet model was used as a feature extractor for the training images. An SVM was then later trained using the features extracted from the SVM to achieve an accuracy of 76% on the test set [4]. t-SNE feature visualizations from these CNN features show a 2D representation of the distance in features between multiple instances [6]. This SVM based submission passed each image of a business to get its features and then used SimpleMI Multiple Instance Learning on the average of the feature vectors for each business [4]. Although CNNs are generally used as a full pipe-line for classification, research has shown that using deep CNNs as feature extractors and SVMs as classifiers tends to give similar results to a well-trained classification layer within the CNN architecture itself. Thus, it seems that perhaps a well-trained neural net should be able to at least reach an accuracy of 76% on the Yelp dataset.

However, this SVM use of a CNN didn't involve directly training on a CNN so that it could be fine-tuned for the specific task of classifying the 9 labels desired in the challenge. Exploring this space gives us our trial approach that will attempt to have an end-to-end pipeline using a fine-tuned CNN trained on the Yelp data itself.

The most state of the art approach that could be applied to understanding several objects in the Yelp data is the use of faster-rCNN region proposals. This technique uses a neural net to learn region proposals and then detect objects accordingly to those regions. This proves exceptionally well for understanding multiple objects in the same image and keeping track of where they occur spatially. This method has yielded a mean average precision of 58.85% on the ImageNet object detection challenge [10]. Another technique like YOLO (You Only Look Once), which formulates object detection as a regression problem, is also capable of tackling object detection even though in practice it performs slightly worse than faster-rCNN [9]. However, training these architecture requires very specifically labeled datasets that have every object of interest labeled. Leveraging these models would require us to do additional separate labeling on the Yelp data to make these effective.

Fig 1. t-SNE Plot of Expensive Images [4]

Fig 2. t-SNE Plot of Lunch Images [4]

2.1. Multiple Instance Learning

Another significant aspect of solving this problem involves the concept of multiple instance learning. The details of multiple instance learning (MIL) are provided later in Section 5.4, but essentially it is a learning technique that is useful when individual training examples do not have labels themselves, but attributes are assigned to *bags* of labels

instead. MIL has proven very successful in the past. For example, in a paper by Babenko *et al.*[1], MIL is used for solving a Visual Tracking problem. Their main goal was to show that MIL resulted in more stable and accurate tracking models relative to the existing models in literature at the time. They presented a novel algorithm, in addition to a novel loss function, that they used to train their models to perform online MIL classification. The loss function itself was crafted artfully, as it took into account the gradient boosting framework detailed in another paper, Viola *et al.* [8], to maximize the log-likelihood function across several bags of instances.

Another notable paper that detailed MIL is Jia *et al.* [13], specifically in its formulation of the loss function. The loss function in this paper still calculates the loss across bags of instances rather than separately calculating the loss of each instance individually, but it uses a more intuitive method to take the true label of the bag into account, essentially formulating a different loss function for positive bags as opposed to negative bags. While these different loss functions lead to a discrepancy in the magnitudes between positive bag losses and negative bag losses, they cleverly weight them appropriately so that the final loss is still a within a reasonable range.

2.2. Transfer Learning

Within the field of Convolutional Neural Networks, there has been a lot of work done with Transfer Learning. Transfer learning is useful when one wants to train a CNN on their own dataset, but for various reasons, the dataset may not be adequate to train a full neural net on (i.e. it could be too small). While data augmentation is a viable option in a lot of cases, transfer learning has also proven effective. Transfer learning refers to the process of taking a pretrained CNN, replacing the fully-connected layers (and potentially the last convolutional layer), and training those layers on the pertinent dataset. By freezing the weights of the convolutional layers, the deep CNN can still extract general image features such as edges, while the fully connected layers can take this information and use it to classify the data in a way that is pertinent to the problem. In practice, when there is a moderate amount of data, transfer learning is performed on the last fully-connected layer and classification layer with a learning rate 1/100th of the value used to get the model to converge when first trained on ImageNet.

3. Approach

3.1. Dataset

The training dataset consists of about 234,842 images, all of which belong to one of about 2000 businesses. The dataset was specifically procured for this particular task and is based off a real life scrape of Yelp data. This means that the data is subject to noise and incorrect classification as mentioned in challenge page. The test set is similar and contains 237,152 images. CSV files denote which images correspond to which businesses and the

0- good-for-lunch
1- good-for-dinner
2- takes-reservations
3- outdoor-seating
5- has-alcohol
6- has-table-service
7- Ambiance-is-classy
8- good-for-kids

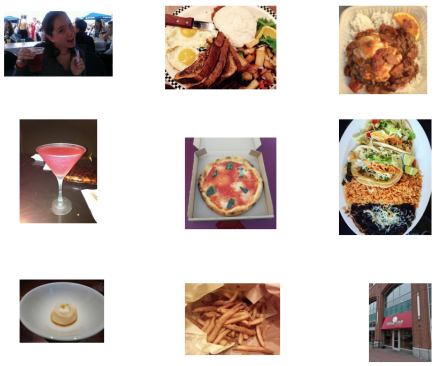

Fig. 3 A Representative Sample of the Data

Regarding dividing the dataset up into training/validation/test sets, we divide our dataset by businesses, rather than pictures. Therefore, the exact size of the training and validation sets are not well-defined, as a business could have an arbitrary number of pictures. For the purposes of this people, we keep a validation set of 100 businesses.

3.2. Preprocessing

Since we attempt to a VGGNet like architecture and have a decent amount of training data, we do the same preprocessing as performed in the VGGNet paper. This simply invovled applying normalization by subtracting the mean of all images from each image. Additionally, since photos came in varying sizes and resolutions, we used OpenCV to rescale each image to 224x224x3 so that its dimensions correspond to the original VGGNet input requirements.

3.3. Single Instance Graph Model(Baseline)

For initial experimentation, we created a graph model CNN based on the VGGNet architecture. In order to use the features of the pre-trained VGGNet, we remove the final softmax classifier and substitute that in for 9 independent, single-neuron fully connected layers and apply a sigmoid activation. These new layers receive the input of the last fully connected layer in the original VGGNet. The output

of these 9 layers gives a single scalar between 0 and 1 that we then merge together to make the prediction vector for a given image.
For a given single-neuron layer, the sigmoid activation is calculated as follows, where s is the score of the example prior to applying the sigmoid activation:

$$\sigma(s) = \frac{1}{1 + e^{-s}}$$

We clip at values of 0.95 to set a label as positive. For the rest of the experiments, we define a negative classification as 0, and a positive classification as 1. An example output vector for a given training example would look as follows:

$$[0, 1, 1, 0, 1, 0, 0, 1, 0]$$

As an initial start, we trained this model on single instances using cross-entropy loss on the output, Adam update function, an initial learning rate of $1e-9$, and 100k examples for 10 epochs. The learning rate was then reduced to $1e-7$ after it was noticed that little reduction in loss continued. We chose an initial learning rate of $1e-7$ since it matches the recommendation of 1/100th of the learning rate of the original VGGNet. It took approximately 14 hours to train. We call this general architecture "YelpNet".

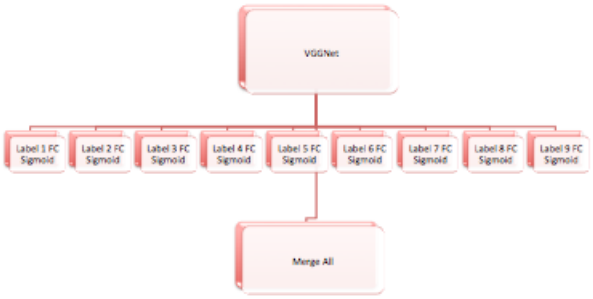

Fig 4. Our used CNN Architecture

3.4. Multiple Instance Learning Detail

A particular quirk about this problem is that we have several images per business, where each business is assigned a training label. However, this does not directly translate well to a standard convolutional neural network model, simply because we cannot directly assign training labels to each image. In our initial baseline approach with separate models and our first true attempt at using a single, multi-output model, each image was assigned all of the training labels of the business that it corresponded to. This does not necessarily train the model very accurately, because a business could be assigned the attribute "Classy", and have a picture of its bathroom; while the bathroom itself does not reflect the classiness of the restaurant, the picture would still train the model to think that pictures of bathrooms may correlate to the classiness of the restaurant.

To combat this issue, we look at the concept first referenced in Viola *et. al*[8] called Multiple Instance Learning. Effectively, Multiple Instance Learning is required where training examples themselves do not have training labels, but are put into bags where each bag has a training label. This is directly relevant to the problem we are trying to solve. Multiple Instance learning works by taking a "Noisy-OR" across all of the examples in a bag, and uses the result of that to classify the bag itself.

We index bags with i, where the i^{th} bag is denoted as X_i and the label associated with bag X_i is known as y_i. Each training example in bag X_i can be denoted as x_{ij}.

When x_{ij} is passed through the model, we take the output from the final fully connected layer as the score of the training example, $\hat{y}_{ij}$.

Our model that uses MIL is identical to our single instance model besides how the loss is calculated and the batch size of images we input into the CNN.

3.5. Loss Function

Formulating the loss function for a multiple instance learning problem is not as direct as that of a normal machine learning problem. This is mainly because the loss attributed to each example is not very clear, as a particular example in a bag need not reflect a particular attribute of the bag (as detailed earlier in the MIL section). Instead, we compute the loss function over an entire bag of training examples as opposed to each example. For semantic reasons, we must still provide a loss for each example in a bag (such is demanded by the Keras API); for simplicity, we attribute the loss of the entire bag to each example. Here, we denote a restaurant as a bag, and the pictures associated with restaurant as the individual training examples. Also, given that this is a multiple task classification problem, to calculate the total loss, we sum together all of the losses that are all computed with respect to each attribute. Before formally describing the loss function, we define some important terms.

- Let $L_{total,i}$ be the total loss due to bag i
- Let L_{ai} be the loss of a bag i with respect to attribute a
- Let training example j in bag i be denoted as x_{ij}
- Let the soft-label prediction of example x_{ij} with respect to attribute a be denoted as $\hat{y}_{ija}$

$$L_{total,i} = \sum_{a=1}^{class} L_{ai}$$

For a restaurant i whose label is negative for attribute a, we describe the loss with respect to bag i as follows:

$$L_{ai} = \sum_j (\hat{y}_{ija})^2$$

For a restaurant i whose label is positive for attribute a, we describe the loss with respect to bag i as follows:

$$L_{ai} = \sum_j (\hat{y}_{ija} - 1)^2$$

Intuitively, for positively classified bags, this loss function assigns a high loss to examples classified negatively, and low loss to examples classified positively; similarly, for negatively classified bags, this loss assigns a low loss to examples classified negatively, and a high loss to examples classified positively.

This could be extended to something more sophisticated as detailed in Jia *et al.*[13], which makes use of a Laplacian matrix to help regularize the loss function, in addition to modifying the basic approach above. For negatively classified bags, the approach is the same. However, for positive bags, instead of assigning a loss to all negatively classified examples, they only attribute a loss to the example with the highest classification. Intuitively, what that does is that if a bag is labeled negative for a particular attribute, none of the examples should reflect that attribute; however, if a bag is labeled positive, then only one of them needs to strongly reflect that attribute in order for the bag to be classified as such. Therefore, they should only assign a high loss to a positively classified bag if even the most "positive" example is still negative.
Inspiration for our custom loss function was derived from Jia *et al.*[13] and Babenko *et al.*[1].

3.6. Technical Limitations

Due to the nature of multiple instance learning and specifically formulating the loss function, it is necessary that all training examples of a particular bag (restaurant) be batched together. Therefore, a minibatch paradigm cannot be used. However, with this particular problem, some restaurants have on the order of 400 to 500 images; with the limited computation power that an AWS GPU instance provides, the maximum batch size possible is 64 images. This prevents looking at the whole bag of examples at a time while training, but for the purposes of this paper, we can somewhat approximate the full-batch technique. During training, for a bag X_i, we split its examples x_{ij} into bags of size 64, and treat the new bags' attributes as those of the old bag. This allows us to mimic looking at the full batch of examples as well as possible with limited computation power. We are also limited in adding more layers as part of the 9 branches since the VGG-19 model pushes us to use most of the memory. Since we want to use as many business photos per batch as possible, we did not experiment with adding more layers per branch.

3.7. Classifying Bags based on Examples

With multiple instance learning, we cannot evaluate the accuracy by looking at each example, but instead by looking at the classification of the entire bag of examples. First, we detail two distinct ways of obtaining the classification of a bag X_i given the examples x_{ij} and their corresponding soft-label predictions $\hat{y}_{ija}$. Let the prior definitions of variables apply, adding $\hat{Y}_{ai}$ as the prediction for bag X_i for attribute a.

3.7.1 Classification Method 1

The first method involves the following procedure:

$$\hat{Y}_{ai} = max_j(\hat{y}_{ija})$$

If $\hat{Y}_{ai}$ is greater than some constant γ, then predict 1 for attribute a. In this case, $\gamma = 0.9$

3.7.2 Classification Method 2

The second is as follows:

- First clip all predictions above a constant α to 1 and clip the rest to 0. Call the clipped version of $\hat{y}_{ija}$ as c_{ija}.
- α was set to 0.8

$$t_{ai} = \sum_j (c_{ija})$$

- Assign $Y_{ai} = 1$ if $t_{ai} > \frac{N}{\beta}$, where N is the total number of examples in bag X_i and β is an integer value that intuitively represents the proportion of examples in a bag that must classify as positive in order for the bag to be classified as positive with respect to that attribute

3.8. Performance Metric

Since we are deviating from standard single label object classification, we use a different measure of accuracy as opposed to the standard total labels correct over the amount of test examples. To evaluate performance, we use the Mean-F1 score. Mean-F1 functions as an effective metric since it takes into account both precision and recall. Precision in this case is all true positives over true positives plus false positives. Recall is a ration determining how many of the

right labels we caught. Formally, recall is the ratio of true positives to all actual positives. Below p is precision, r is recall, tp is true positives, fp is false positives, and fn is false negatives. Mean-F1 score poses as a valid accuracy metric since it entails maximizing both precision and recall as opposed to one individually [7]. Essentially, this allows metric measures consistently correct predictions and capturing as many true positives as possible. F1 scores are generally used in measuring accuracy on queries and classifications in the Natural Language Processing space.

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

3.9. Results

We evaluated both our naive model that does not perform Multiple Instance Learning and our model that does. From now on, we will refer to the prior model as the naive model, and the latter as the MIL model.

The following is a graph of the loss function while training our MIL model for 12 epochs, where each epoch looked at 80,000 training examples; the learning rate was $1 * 10^{-8}$ on an Adam update, the batch size was 64 (due to technical limitations), and Adam parameters were $beta_1 = 0.9$, $beta_2 = 0.999$, $epsilon = 1e-08$. For visualization purposes, we have only plotted 200 points for each epoch to provide a representative view of the loss function.

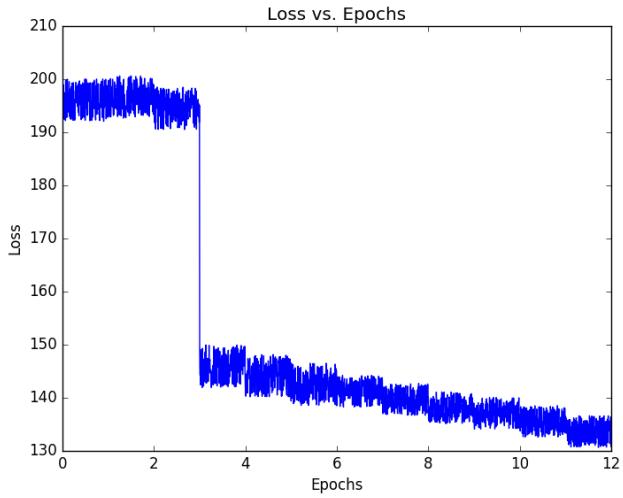

Fig. 6 Training Loss Over Iterations for the MIL Model

To evaluate our model, we calculated the Mean-F1 score across a validation set of 100 businesses, that achieved mean-F1 Score of:

$$F1 = 0.618$$

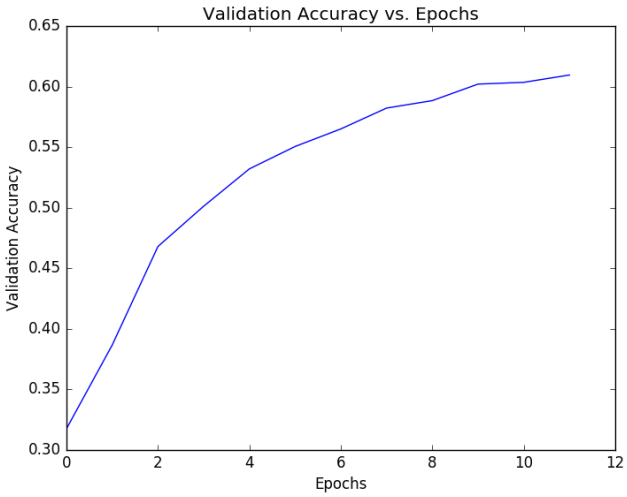

Fig. 7 Validation Accuracy vs. Epochs

Compared to the naive model, our MIL model performed better by a non-negligible amount, indicating the effectiveness of Multiple Instance Learning compared to naively approaching this problem as single instance classification.

Because this project was inspired by a Kaggle challenge, we acquired the results of other methods of classification and compared those with our results.

Model	Accuracy (F1 %) * Indicates train accuracy for 1k businesses
Random guessing	43.5
Color Distribution	64.6
Mean sampling	60.9
SVM on CaffeNet Features	76
Naïve YelpNet	53.3
MIL YelpNet	61.8

Fig. 5 Accuracies for Different Model Approaches

Experimentation was also necessary for the classification strategies. Earlier we detailed two classification strategies, both of which looked at the classification of all examples in the bag and took some combination of them to classify the bag. Experimentation with both of these methods (Section 5.7.1 and Section 5.7.2) led us to realize that the first gave us more accurate predictions. This intuitively makes sense, as the second classification strategy takes the prediction of all examples in a bag and checks whether a certain number of examples depict that attribute. However, in practice, only one picture that heavily depicts that attribute needs to exist in the batch in order for the bag to likely reflect that attribute. For example, if a restaurant served alcohol, and only one of the pictures had a glass of

beer in the frame, it would still provide adequate evidence that this restaurant served alcohol. We achieved the highest accuracy on our validation set with the former method. The only caveat with using the former method, however, was that by taking the maximum classification value $\hat{y}_{ij}$ across all instances in a bag, the resulting value would inherently be very large, regardless of the classification of the bag. This required adjusting the threshold value at which we clipped the prediction to the space $\{0, 1\}$.
Below is a table of different clipping thresholds and their corresponding Mean-F1 scores. These were evaluated on a randomly-sampled 100-business training set in addition to a 100-business validation set.

Clipping Threshold	Training Score	Validation Score
0.7	0.432	0.416
0.8	0.555	0.581
0.85	0.594	0.587
0.9	0.627	0.618
0.95	0.609	0.595

Fig 6. F1 scores for Different Threshold Clippings

4. Discussion

4.1. Algorithm

An instant challenge in using a VGGNet model for the Yelp problem was modifying its functional use for multi-label classification in batches. Predicting multiple labels for an image is a fairly used extension of single object recognition which can involve training so that the output gives the top k labels. What proved to be a significant part of experimentation for us is determining a way to adequately adopt a CNN to process images by batch. The added technical challenge stems from the fact that each business tends to have hundreds of examples that have the set of labels that correspond to that business. These pictures tend to have high amount of variation in their content. For instance, the same business could have several pictures of the same pasta dish, have a few of a takeout pizza box, or selfie of a couple on a date. Thus, the classification could not be easily done by just looking on single instances as separate cases. We have to make predictions accordingly as a batch by all photos for a given business, which was somewhat fixed by using Multiple Instance Learning

4.2. Dataset

Another significant challenge for our algorithm to overcome came from general noise in the dataset. A manual run through of the data shows that there are several noisy images that don't pertain to the food, drink, or ambiance of a business. There could be pictures of the storefront, the view outside the window, parking lots, or group photos. Thus, a technique that processes all photos of a business at once at training would also need to be very resistant to the variation in object that would help in classification along with resistance to the general noise of objects that can confuse in the classification task. To make the task even more difficult, the dataset contains several duplicates due to users uploading the same photo multiple times and chain business uploading the same photos to each location of the same business.

Another problem was that predicting one of the 9 labels is a classification problem that might fall more in the lines of object detection by regions to extract the multiple objects in a picture. For instance, a picture of place that's good for dinner should be highly activated if there's wine, fancy food, and fancy plates. It would be ideal to have an explicit understanding of all pertinent items such as these.

In the context of transfer learning, we also need to note the difference between the ImageNet dataset and the Yelp dataset. Although both pertain to real-world objects, the Yelp dataset tended to have a more narrow range of objects. For instance, we expect scenes of within a restaurant and it's surrounding area. Intuitively, it doesn't make sense to have an understanding of unrelated objects such as mountain lions. Instead, we want our net to be more sensitive to objects that pertain to the restaurant space and to be able to make fine distinctions between such objects. For instance, an upscale and expensive Italian restaurant might also offer pizza, but the attributes for this restaurant would vary from a take-out pizza chain. Thus, training a CNN from scratch for this task might be able to make fine-grain distinction.

4.3. Choice of Hyperparameters

The learning rate was calculated based on VGG-19's original learning rate and the fact that we were performing transfer learning. As mentioned earlier, in transfer learning, it is necessary to decrease the magnitude of your learning rate by a couple orders in order for it to actually fine tune the model. Though our original learning rate was $1 * 10^{-8}$ at the end, in order to see the effects of fine-tuning even more, we took our 12-epoch trained MIL model discussed in our results section and trained it for 6 more epochs, but this time with a learning rate of $1 * 10^{-9}$. With fine tuning a model, it is important to decay the learning rate as the loss function begins to converge; by decaying the learning rate, we can intuitively reach the "crevices" of the loss function that were unreachable with higher learning rates. Below is a graph of the loss function:

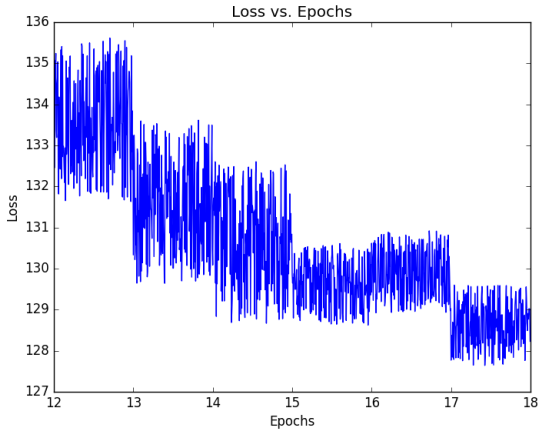

Fig. 7 Training Loss Over Extra Iterations

The choice of using Adam as our optimizer was simply because it has yielded some very positive results recently and has been known to converge faster than Vanilla-SGD and other popular optimizing techniques. The Adam-specific hyperparameters $\beta_1, \beta_2, \epsilon$ were chosen based on the recommended standard by the Keras API.
Another important hyperparameter is the minibatch size; while MIL requires a somewhat "full-batch" approach, the batch size is not very well defined. However, technical limitations prevented us from using a batch size of larger than 64.

5. Conclusion

The main takeaway of is that our results tend to fall close to the baseline result of making results by the distribution of data. This approach looks at how on average 5 labels are chosen and that those labels tend to be labels 2, 3, 5, 6, 8. A quick observation of our actual output show that our results tend match this sort of prediction scheme. The net tended to highly misclassify businesses that had fewer than 5 attributes. Thus, it appears that our CNN was not working well in terms of extracting a fine amount of detail from the VGG-extracted features. Instead, it seems that our branching layers we trained on were only getting an average result. In fact, just by arbitrarily predicting that a restaurant exhibited attributes 2, 3, 5, 6, and 8, we achieved an accuracy of about

$$F1 = 0.658$$

That shows that the dataset did not cover a broad enough range of examples, and due to this lack of diversity, it is possible that the model could easily overfit the training data. However, this is not necessarily a bad thing. If we take a closer look, attributes 2, 3, 5, 6, and 8 (refer to **Section 5.1** for exact mappings) intuitively would appear in most restaurants, and so even if our model was more likely to predict those by seeing more of those examples, it would still produce reasonable results in the context of the problem, as most of the restaurants in the test dataset would exhibit these same features.

Despite this flaw in the dataset, it is still to overcome being limited by only viewing the averages in the data, and so a model must be able to make finer-grain understanding about images and the objects they contain. An observation of the t-SNE figures in **Fig 1.** shows a decently wide amount of variation even within photos strongly signal the same attributes. There's a wide range of foods, lighting, plates, silverware, and other miscellaneous objects related to a restaurant setting. Thus, it appears that what is needed is not a generic image classifier but instead a model that more tightly fits the scenarios presented in the data. Achieving this would require a more robust dataset and adjusting the size of the model, as well as adjusting the learning rate and other update parameters.

5.1. Future Extensions

There are quite a few areas of further investigation that might prove worthwhile in improving the results of this architecture. More time could be spent in fine tuning the learning rate to see if that's a source of low accuracy. Additionally, one could train on a more powerful machine that would address the technical limitations we faced so that one could train by batch on all of a business' images. One could also transfer learn on another dataset, such as Food-101 [3], which labels 101 classes of food and contains Yelp-like user uploaded photographs of dishes. This could aid in teaching the net how to make finer distinctions between culinary items.

Outside of transfer learning all together, one could of course train a CNN from scratch only using the Yelp dataset. A motivation for this is that VGGNet extracts features for 1,000 different objects, many of which, are not particularly relevant to our problem. For instance, being able to distinguish several breeds of felines or dogs is frivolous to understanding restaurants. Instead, it would be ideal to have a net only focus on the features that really matter in a restaurant business setting.

A more complex could involve transferring the problem into an object detection problem as opposed to image classification. A faster rCNN-like approach could be applied so that the net has more regional understanding of multiple objects as opposed to trying to extract just one meaning from the photo. This would aid in being able to distinguish cups, plates, food, tables, chairs, and other objects at a more specific level by applying self learned region proposals [10].

References

[1] B. Babenko, M. H. Yuan, S. Belongie. Visual Tracking with Online Multiple Instance Learning.

[2] B. Baben, M. Yang,, & Belongie, S. (2009). Visual tracking with online Multiple Instance Learning. 2009 IEEE Conference on Computer Vision and Pattern Recognition.

[3] L. Bossard, M. Guillaumin Van Gool, and L. Van Gool (2014). Food-101 – Mining Discriminative Components with Random Forests. European Conference on Computer Vision.

[4] N. Chen,(2016, February 28). Deep learning starter code. Retrieved March 8, 2016, from https://www.kaggle.com/c/yelp-restaurant-photo-classification/forums/t/19206/deep-learning-starter-code

[5] JP_smasher. (2016, February 9). Naive Benchmark (0.61). Retrieved March 08, 2016, from https://www.kaggle.com/wongjingping/yelp-restaurant-photo-classification/naive-benchmark-0-61

[6] L.J.P. van der Maaten. Accelerating t-SNE using Tree-Based Algorithms. Journal of Machine Learning Research 15(Oct):3221-3245, 2014.

[7] "Mean F Score. (n.d.). Retrieved March 09, 2016, from https://www.kaggle.com/wiki/MeanFScore

[8] P. Viola, J. C. Platt, C. Zhang. Multiple Instance Boosting for Object Detection. In NIPS, 2005.

[9] "Redmon et al, You Only Look Once: Unified, Real-Time Object Detection, arXiv 2015"

[10] S. Ren, K.He, R. Girshick, and J. Sun,(2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. ArXiv.

[11] K. Simonyan and A. Zisserman (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR.

[12] C.Szegedy, S. Ioffe, and V. Vanhoucke (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. ArXiv.

[13] Y. Jia and C. Zhang. Instance-level Semi-supervised Multiple Instance Learning.

[14] Y., D. (2015, December 23). Introducing the Yelp Restaurant Photo Classification Challenge. Retrieved March 08, 2016, from http://engineeringblog.yelp.com/2015/12/yelp-restaurant-photo-classification-kaggle.html