Yelp Restaurant Photo Classification using Convolutional Neural Network: A transfer learning based approach

Deepak, Pulkit, Usha, Anahita

March 11, 2016

Introduction

In this age of food selfies and photo-centric social storytelling, it may be no surprise to hear that Yelp's users upload an enormous amount of photos every day alongside their written reviews. Every time a user uploads a photo of a restaurant, the person tags some attributes along with the image that closely describe the restaurant. For example, a user might upload a few photos of a restaurant and tag these images with labels such as 'restaurant is expensive", 'good for kids' etc. However till now, this tagging has to be done manually by the user. It would be useful to have a model that automatically recommends these tags for these restaurants based on the photos uploaded by the user. Our project is aimed towards this task of building a system that predicts the attribute labels of restaurants based on user-submitted photos.

Dataset

For this project, we are using the data set provided by Kaggle as part of the Yelp Restaurant Photo Classification Challenge (provide link in footnote).

The data set consists of around 230,000 user-submitted images. Each image is down sampled to around 30 Kb and the entire data set amounts to around 7.4 Gb. All the images are RGB images (three color channels: Red, Green and Blue) and can be of any dimension.

Each of these images is indicated by a unique ID called the Image ID, and every image is associated to a restaurant indicated by a Restaurant-ID. There are approximately around 2000 restaurant IDs to which these 230,000+ images are mapped to. This mapping information is available in the 'train-photos-to-biz-ids.csv' file provided along with the dataset.

Also, each of these restaurants contain nine binary attributes. For example, a restaurant with ID 1 may contain attributes such as 'restaurant is good for dinner', 'restaurant has a nice ambiance', 'restaurant is expensive' etc. Another restaurant might contain a different set of attributes like 'restaurant is good for kids' etc. Each of these nine attributes is indicated by labels as follows:

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: ambiance is classy
8: good for kids

This mapping information between each restaurant ID and the corresponding attributes is given in the 'train.csv' file.

Methodology

Why is this classification task hard?

Given these set of images (mapped to restaurants) along with the set of attributes for every restaurant, our aim is to build a model which predicts these attributes for a restaurant given a new set of test images related to this restaurant.

For example, assume that a user goes to a restaurant and uploads five images on Yelp. Then ideally our model should be able to predict the attributes for the restaurant based on the information extracted from these set of five images.

However, this classification is extremely hard due to the following reasons:

1. Most of the existing image classification datasets such as MNIST, CIFAR10, CIFAR100 and ImageNet consists of preprocessed images without much noise. However, the dataset for our task consists of photos uploaded by users and the quality of these images vary drastically. For example, by looking at some of the images we observed that some images were of a high quality whereas most of the other images in the data set were blurry. This kind of inherent noise in the data set makes the classification task even more harder.

2. Secondly, the attributes tagged by users for a specific restaurant is highly

subjective and depends on the user. For example, in image classification data sets such as ImageNet, images containing helicopter will be tagged to a specific class and images containing cats will be tagged to another class. For our data set though, images taken at the same restaurant might be given different attributes by different users. For example, one user might think that the restaurant is expensive and give this as an attribute. Whereas another user might not consider the restaurant to be expensive. Also, a restaurant which is good for dinner according to a particular user might not be good for another user. This kind of subjective opinion also makes this task harder.

3. Also, we have to keep mind that our problem is a classification task based purely on visual information alone. Hence, our model will work best for attributes for which relevant visual information can be extracted from images. For attributes such as 'restaurant is good for dinner', our model is able to capture the important semantic information that is common to images that have been tagged with this attribute. For example, most of the images which are tagged with this label usually have a dark lighting. This seems logical as the user is most likely to give 'restaurant is good for dinner' as a tag when he goes to that restaurant for dinner and hence, the photos he/she uploads usually have a dark background. For other attributes such as 'restaurant takes reservations', we believe that the visual information alone is not enough as these kind of attributes cannot be represented well in images.

Convolutional Neural Network: An overview

Having explained about the difficulty of this classification task, we felt that the shallow machine learning models might not be able to capture this semantic information embedded deep within the images. Hence we used a convolutional neural network based approach for this multi-label classification task. In recent times, deep neural networks have outperformed shallow machine learning models in classification tasks such as ImageNet etc. This is because the higher layers in a deep model are able to reuse primitives obtained from the lower layers and build more complex functions. For example, in vision tasks, the first layer learns Gabor filters capable of detecting edges which are then put together in the consequent layers to form partofobject shapes. As the number of layers increases, these part of object shapes are combined to obtain detectors for more complex shapes ([1] describes this behavior very well). However, on the other hand, a shallow classification model with a single layer is forced to generate detectors for target objects based on the features learnt from the first layer alone.

A vanilla CNN (displayed in Fig.1) consists of the following layers:

1. INPUT layer: which holds the raw pixels of the image.

2. CONV layer: which computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and the region they are connected to in the input volume.

3. RELU layer: which applies an element wise activation function such as max(0,x) thresholding at zero.

4. POOL layer: which performs a down sampling operation along the spatial dimensions.

5. FC layer: which computes the class scores.

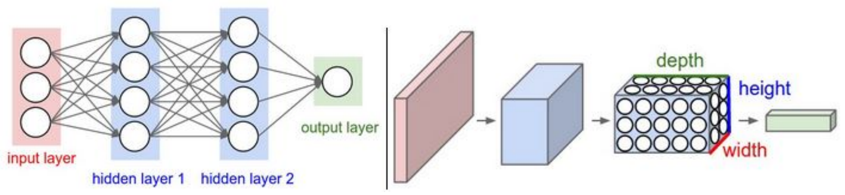


Figure 1: Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels)

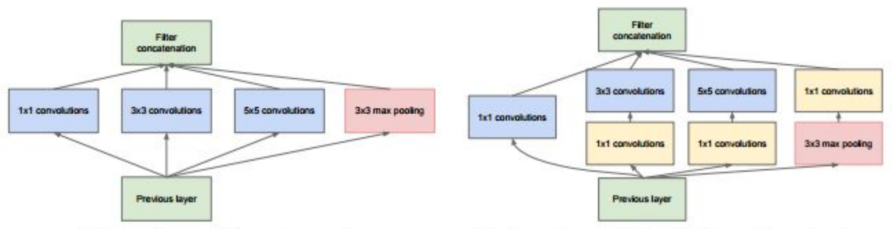
Inception

However for sophisticated classification tasks such as ours, we need more complex models with more layers in order to better represent the features. There are already a lot of pre-trained CNN models available such as the AlexNet, the VGGNet, GoogleNet architecture. These architecture have been trained on a data set of around one million images and have got state-of-the-art results in the 2012 ImageNet and 2014 ILVSRC competitions.

More recently, the Inception v3 architecture released by Google has achieved the best results so far. It has achieved a top-5 error of 3.5% and a top-1 error of 21% which is the best result till now. Naturally, in order to the achieve the best results for our task, we invoked the Inception v3 architecture and built our transfer learning based model on this. The Inception v3 architecture is displayed in Fig.

Inception architecture has shown great performance in terms of computational cost.This has made it suitable for big data scenarios where the data need to be processed at a reasonable cost. The main idea of the Inception architecture is to represent and approximate local sparse structure in a convolutional network by readily available dense components. Layer-by-layer we need to analyze the correlation statistics of the last layer and cluster them into groups of units with high correlation.In the Inception architecture, the filter sizes are restricted to sizes of 1x1, 3x3 and 5x5 primarily for convenience purposes.

The 'Inception block' is composed of convolutional layers with the output filter banks concatenated into a single output vector which serve as an input to the next stage as shown in Fig[add figure number].This is the naive version of the architecture. The problem with this version is that the merging of two layers (say, a max pooling layer with a convolutional layer) would increase the number of outputs at each stage. This might lead to computational blow-up in just a few stages. To handle this problem , the second idea for Inception architecture was proposed i.e. to judiciously apply dimension reduction whenever computations increase too much.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

Generally speaking, an Inception architecture is a network consisting of 'inception modules' stacked upon each other.For memory efficiency while training, Inception architecture usually start using inception modules only at higher layers keeping the convolutional layers at the lower layers.

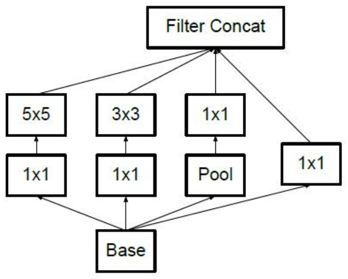
One main advantage of this architecture is that it allows increasing the number of units at each stage without any uncontrollable blow-up in the computational complexity of the model. The methods explaining the efficient approaches to increase the number of units is explained in [7]. One of these approaches is factorizing the convolutional layer, for example, factorizing one 5x5 convolutional layer into two 3x3 convolutional layers.

This architecture intuitively also suggests that the visual information should be captured at different scales and combined appropriately, so that the next layer can abstract features from different scales simultaneously.

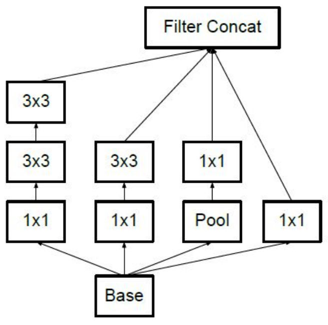
We believe that the Inception architecture fits best for our multi-label classification task, meeting the computational complexity and memory requirements, along with good performance results. The number and types of layers used in final inception-v3 architecture are depicted in the table below.

Type	patch size/stride	input size
conv	3x3/2	299x299x3
conv	3x3/1	149x149x32
conv padded	3x3/1	147x147x32
pool	3x3/2	147x147x64
conv	3x3/1	73x73x64
conv	3x3/2	71x71x80
conv	3x3/1	35x35x192
3xInception	As in figure [fig number]	35x35x288
5xInception	As in figure [fig number]	17x17x768
2xInception	As in figure [fig number]	8x8x1280
pool	8x8	8x8x2048
linear	logits	1x1x2048
softmax	classifier	1x1x2

Table 1: The outline of the inception network architecture



(a)



(b)

Figure 3: Inception modules where each 5x5 convolution is replaced by two 3x3 convolution

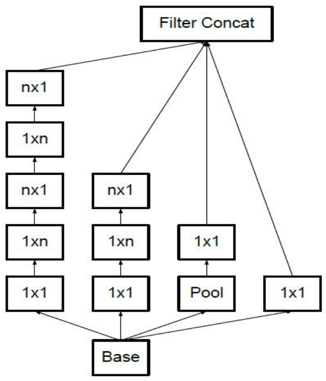


Figure 4: Inception modules after the factorization of the nxn convolutions. In our proposed architecture, we chose n = 7 for the 17x17 grid.

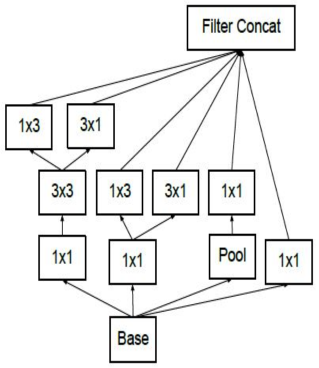


Figure 5: Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8x8) grids to promote high dimensional representations.

Transfer Learning

In computer vision, forming a representation based on sets of trained classifiers on related tasks has recently been shown to be effective in a variety of retrieval and classification settings, specifically using classifiers based on visual category detectors. In our approach, we use the pre-trained deep convolutional model based on the Inception v3 architecture (explained above) and then retrain the weights of the last layer specific to our classification task based on our own images from the Yelp Restaurant data set. The intuition behind this idea of transfer learning is that the neurons in the initial layers of the CNN architecture capture low-level information such as edges, color maps which are common to most of the images and only the neurons in the last feature layer contribute most to the selection of high level features which are crucial for classification. Hence, in order to create a new classification system for a different set of input images which was not previously seen while training the original model, we randomly initialize the weights of the last layer and retrain this layer with inputs from our new data as the new set of training images and the attributes specific to these images as ground truth labels. In this way, we are removing the thousand class softmax layer and replacing it with our own set of classes. The new weight matrix that we obtain after performing stochastic gradient descent is optimized for our specific classification task.

There are many advantages to performing transfer learning, some of which are as follows.

1. Prevents over-fitting of data: The number of images used to train the Inception architecture (the ImageNet database) is very high compared to the images we have in our dataset (nearly 4x). If we were to retrain this complex architecture which contains convolutional layers with our training data, then there is a high chance that this architecture might over-fit our data. Hence, for new test images, our model might not generalize well. By retaining the weights of the previous layers and fine-tuning only the top k layers, transfer learning ensures that we are able to preserve the characteristics of the model which are essential for capturing image-independent low level features but also brings in the ability to capture image-specific high level features relevant to our classification task.

2. Lesser computation time even on CPU: Training a convolutional neural network architecture from scratch costs a lot of time and memory on GPU. With transfer learning however, the computation time is reduced by a huge margin. Though it is not as good as a full training run, this is surprisingly effective for many applications, and we can easily observe from our results that it performs well for our task.

Pre-processing

As mentioned previously in section 1, the Yelp Restaurant Image data set consists of images which are of different dimensions. In order to send these images to the Convolutional neural network, all the images need to be re-sized to the same dimension. Typically in deep learning literature, for giving images as inputs to CNN, the images are augmented to a dimension of 224 x 224. In this pre-processing step, we augment images to dimensions of 224 x 224 x 3 (the third dimension is for the R,G,B color channels).

Also, it must be noted that our model is an image classification model. i.e. a set of attributes will be predicted for every input image. Then the attributes for all the images belonging to the same restaurant (this information can be traced back by the mapping between Image ID and Restaurant ID) can be pooled in together in order to predict the set of attributes for each restaurant. However, in the data set we are provided only with a mapping between the images and the restaurants and a mapping between the restaurants and the attributes. So as part of our pre-processing step, we scan through each of the images and assign the corresponding set of labels/attributes to each image.

Software Used and Implementation Details

We used Tensorflow (Python) for our implementation. One of the major advantages of using Tensorflow is its excellent documentation and easy use of code. Also, our model is based on the recently released Inception v3 architecture for image classification developed by Google.

The pre-processing of the 231,000+ images in the data set took about 2 hours and our training process took close to 26 hours on a 8-core 32GB RAM i7 CPU. Although the usage of a GPU could have potentially speeded up our training, since we are using transfer learning, we are effectively retraining the top layer of the pre-trained model specific to our task and this is achievable with a CPU. However, training the entire model from scratch for our entire data set would have been an arduous task s this would have taken close to two weeks even on a very powerful GPU. As a comparison, the training of the Inception v3 architecture (over which we build our model is based on) took more than three weeks even with using high capacity Tesla K80 GPUs in parallel.

Evaluation Metrics

The evaluation metric that we used for our study is the 'mean of F1 scores' measured over N data points in the test set. For this purpose, we will assign a 9bit binary number (one digit per label) to each sample in which, each bit shows if each data point belongs to the corresponding label or not ('1' for belonging and '0' otherwise). For example if our algorithm predicts attributes: 1,3,4,5,6 for an image 1, then the corresponding vector would be a one hot encoded version of these attributes i.e [0,1,0,1,1,1,1,0,0]. Then we will calculate the F1 score for each sample based on the number of correctly predicted '1' bits w.r.t the ground truth, and then will take mean of calculated measures over all the samples. The F1 score is calculated as follows:

P (precision) = $\frac{t_p}{t_p+f_p}$

R (recall) = $\frac{t_p}{t_p+f_n}$

F_1 =2 PR/(P + R)

$F1_{mean}$ =mean($\sum_{i=1}^{N} F1_i$)

where t_p is equal to number of '1' digits predicted correctly and f_p corresponds to number of digits falsely guessed to be '1'.

For the sake of completeness, we also provide training accuracy, testing accuracy and cross-validation accuracy as other evaluation metrics.

Results

In our approach we are performing a binary classification (0/1) for each attribute and then calculate the mean-f1 score based on the results obtained from this classification. In Fig. 6 we plotted a bar graph with the distribution of positive samples for each attribute. Intuitively speaking, this can be explained as the number of times a specific attribute (say 'good for kids') occurs in the data set. We calculated this distribution for both the training data set which consisted of 200,000 samples and the test data set which consisted of 31250 samples. From the distribution, we could observe that almost all the attributes were almost equally represented in the training/test examples except the first attribute 'good-for-lunch' which was under-represented. Unsurprisingly, we got the lowest F1 score for this attribute (refer to Fig. 6).

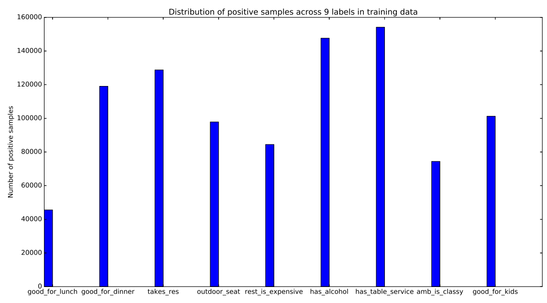


Figure 6: Distribution of positive samples across the nine different attributes for training data

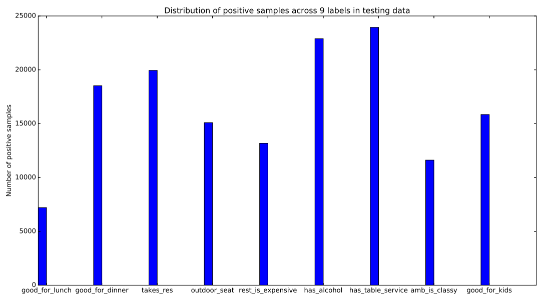
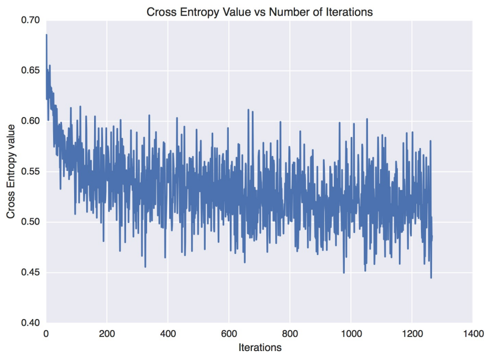
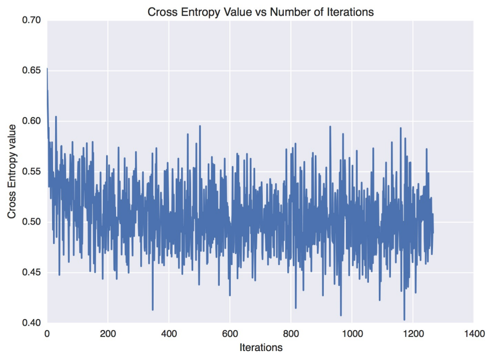


Figure 7: Distribution of positive samples across the nine different attributes for test data

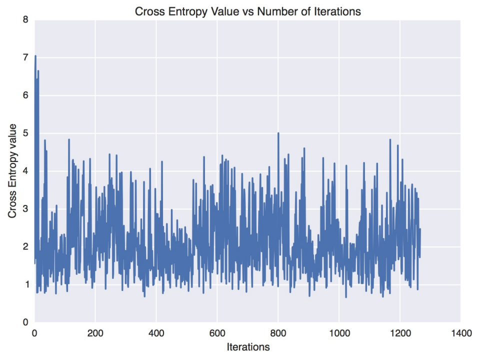
After obtaining the CNN codes (2048-feature vector representation) for the training images, we had to retrain the last feature layer in order to tune the model for our classification task. We evaluated the training accuracy, cross-validation accuracy and the cross entropy values for the different hyper-parameter settings (shown in Fig 8, Fig 9, Fig 10). Two important hyper-parameters that we considered were the learning rate and the batch size. We chose different learning rates from {0.001, 0.005, 0.01, 0.1, 1} and different batch size from {128, 256}. After analyzing the results, we found the best setting to be a learning rate of 0.001 and a batch size of 256.



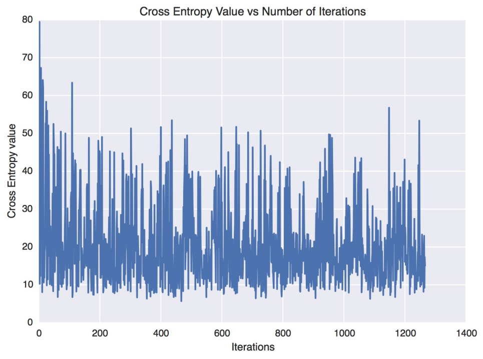
(a) LR=0.001



(b) LR=0.01

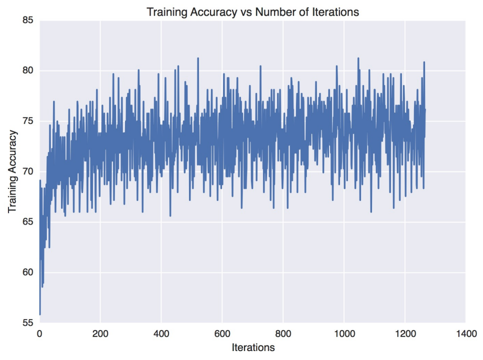


(c) LR=0.1

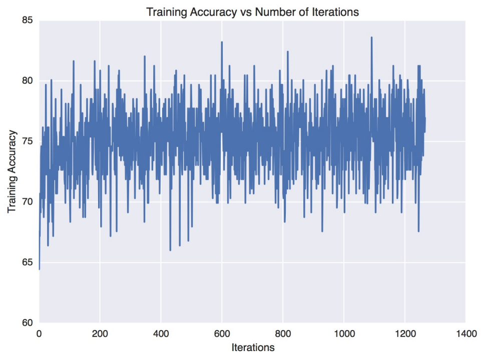


(d) LR=1

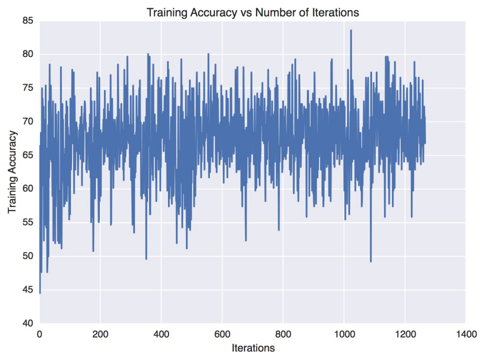
Figure 8: Cross Entropy Value with different Learning rates



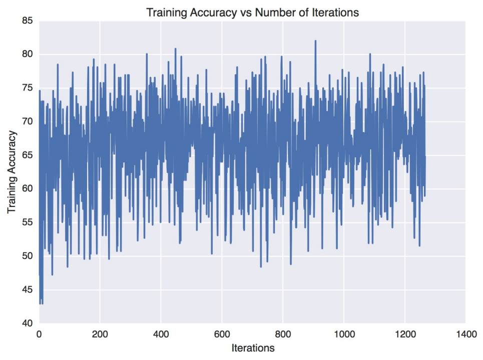
(a) LR=0.001



(b) LR=0.01

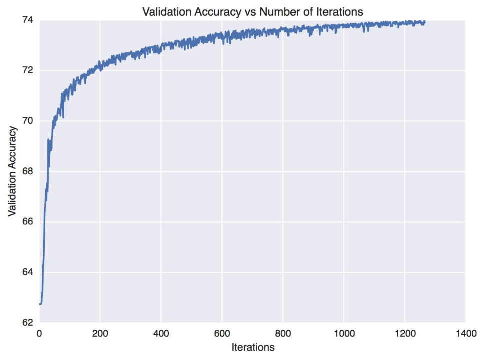


(c) LR=0.1

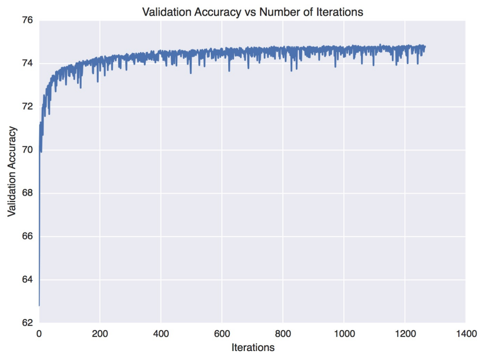


(d) LR=1

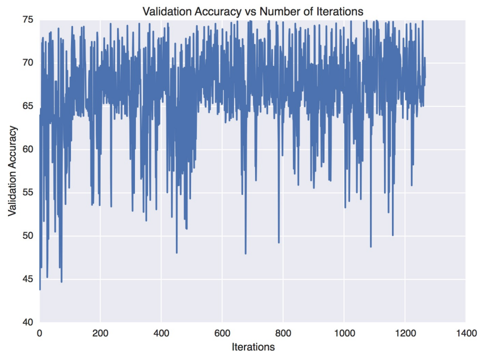
Figure 9: Training Accuracy with different Learning rates



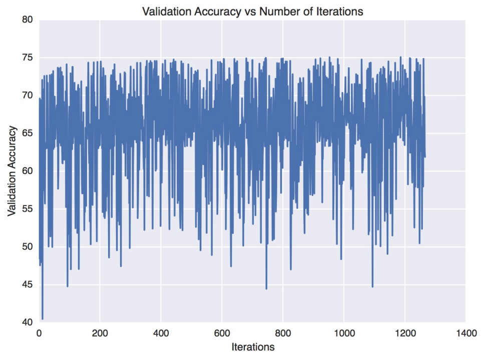
(a) LR=0.001



(b) LR=0.01



(c) LR=0.1



(d) LR=1

Figure 10: Cross Validation Accuracy with different Learning rates

After training our model on this hyper-parameter setting, we evaluate our performance on the test set. We calculate the F1, precision, recall scores for each attribute (based on the formula given in the previous section. These values are reported in Table 2. We get very high F1 scores for all of the attributes except the first attribute. Possibly reasons might be due to the fact that this attribute is under-represented in the training data (as shown in Fig. 6).

For the sake of completeness, and to go along similar terms of calculating F1 as mentioned in the Kaggle Challenge (https://www.kaggle.com/c/yelp-restaurant-photo-classification), we calculated the F1 score for each test example and averaged the F1 scores across the entire test set. For the entire test set of 31250 examples, we got an average F1 score of 0.727 which is amongst the top 30% in the on-going competition. The average F1, precision, recall score for our test set is reported in Table 3.

Table below shows the exact F1-score,precision and recall values.

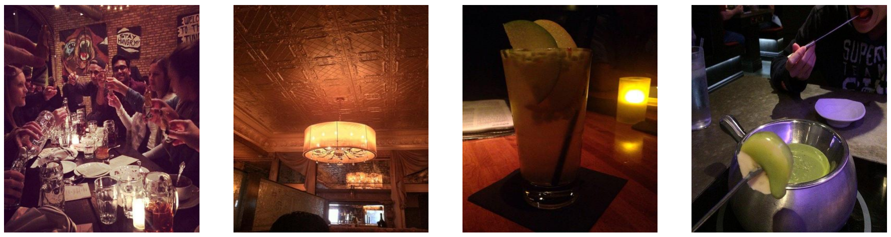
Attribute	F1-score	Precision	Recall
good_for_lunch	0.3389	0.6327	0.2314
good_for_dinner	0.7931	0.7661	0.8221
takes_reservations	0.8282	0.7953	0.8639
outdoor_seating	0.5592	0.5839	0.5365
restaurant_is_expensive	0.6816	0.7258	0.6425
has_alcohol	0.8649	0.8130	0.9239
has_table_service	0.8917	0.8491	0.9389
ambience_is_classy	0.6032	0.7002	0.5298
good_for_kids	0.7457	0.7384	0.7531

Table 2: F1-score,precision,recall for 9 attributes

F1-score	Precision	Recall
0.7279	0.7533	0.7529

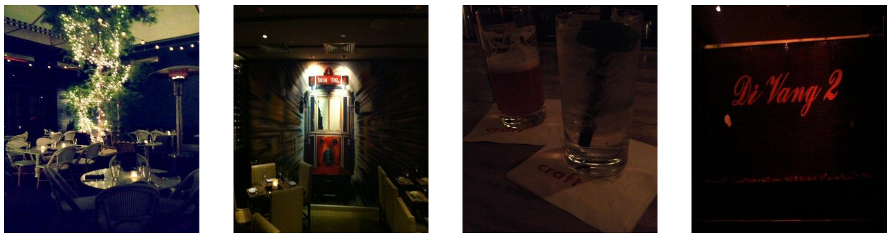
Table 3: Average F1-score,precision,recall across testing data samples

Sample ground truth images and predicted images for "good_for_dinner" attribute.



(a) (b) (c) (d)

Figure 11: Sample ground truth images for good_for_dinner attribute



(a) (b) (c) (d)

Figure 12: Sample predicted images for good_for_dinner attribute

Sample ground truth images and predicted images for "expensive" attribute.

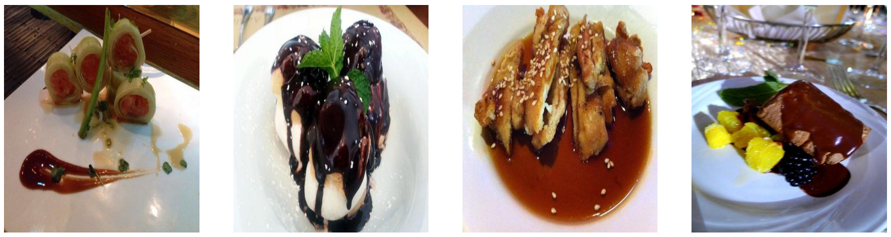

(a) (b) (c) (d)

Figure 13: Sample ground truth images for expensive attribute

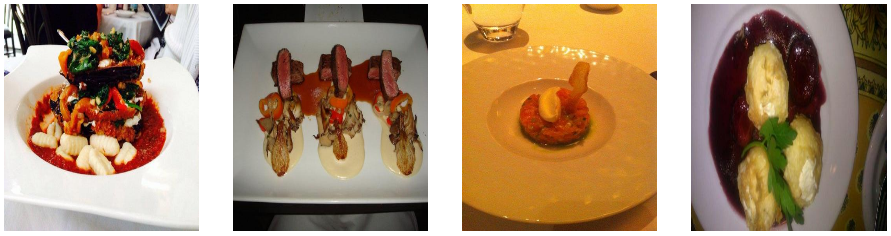

(a) (b) (c) (d)

Figure 14: Sample predicted images for expensive attribute

Conclusion and Future Work

In this project, we utilized the concept of transfer learning to use a set of features derived from one model for another classification task.

Our model produced a good mean F1 score of approx. 0.72 on the cross-validation data set and this is amongst the top 25 percent amongst the current entries on Kaggle. We believe that this describes the effectiveness of transfer learning as we were able to achieve this F1 score by retraining the last layer alone. For our future work, we would like to observe how the F1 score improves if we fine-tune top-k layers instead of the top layer alone. However, it has to be noted that with limited CPU computation, the training time exponentially increases as the number of layers required to train increases.

Also for this project, we treated each attribute individually and trained our model for each label using the cross-entropy loss as our loss measure. Although, this model is effective, one of the limitations is that the training time is longer as the model has to be trained for each attribute separately. For our future research, we would be focused on using a different loss measure such as the Hanning Loss which is better suited to multi-label classification. Also, we would like to use an unweighted soft-max in the last layer with the number of classes as the number of attributes we have. It would be interesting to see if a model with the Hanning loss function is as effective as our current model, as the single model has to be robust enough to classify all the nine attributes with confidence.

References

1. M.D.Zeiler and R.Fergus.Visualizing and understanding convolutional networks. Technical report, arXiv:1311.2901, 2013.

2. Krizhevsky,A., Sutskever,I., Hinton,G.:Imagenet classification withdeep convolutional neural networks. In: NIPS (2012)

3. Simonyan,K., Vedaldi,A., Zisserman,A.: Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv 1312.6034v1 (2013)

4. Girshick,R., J.,Darrell, T.,Malik, :Region based convolutional neural networks for accurate object detection and segmentation. IEEE Transactions on Pattern Analysis and Machine Learning (2014).

5. Boutell, MatthewR., et al. ”Learning multilabel scene classification.”Pattern recognition 37.9 (2004): 175717.

6. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. arXiv 1512.00567v3 (2015)

7.Szegedy, Christian, et al. ”Rethinking the Inception Architecture for Computer Vision.” arXiv preprint arXiv:1512.00567 (2015).