Capstone 2: Yelp Dataset Image Classification
Final Report

Nils Madsen

Introduction

Goal and Utility

When photos are uploaded to Yelp, not all information will necessarily be filled in by the user. Certain fields of information for the photo, such as the label, will likely be left empty for many pictures. Yet, having missing data in the database can make it difficult to run analyses, so there is a motivation to fill in data holes wherever possible. Having labels for all images also helps visitors to the website find information they want about a business.

The sheer number of pictures uploaded to Yelp on a daily basis would make it uneconomical to hire people to classify photos manually. However, modern machine learning algorithms, especially convolutional neural networks (CNN), are powerful, accurate, high-throughput tools for image classification, and therefore are well suited to solving this problem.

The goal of this project will be to develop a CNN model that can accurately classify the photographs in the Yelp Dataset into broad categories.

Data

Yelp has published the Yelp Dataset for academic uses. The larger dataset includes a variety of fields for each rated business, including location, name, type, and star rating, as well as full-text reviews. For each user, there is information on join date, number of reviews written, and number of compliments received. Each photo entry has a corresponding label, caption, and business it is tied to.

This project will focus entirely on the ‘label’ field for each photograph. This is a broad classification, with only five possible entries (‘food’, ‘drink’, ‘inside’, ‘outside’, and ‘menu’). There are a total of 280,992 photographs.

Round 12 of the Yelp Dataset was collected from the Yelp website:
https://www.yelp.com/dataset/challenge

Data Cleaning, Inspection and Management

Data Format

The Yelp Dataset comes in the form of a set of json files, accompanied by a set of images in .jpg format. To be more usable in Python, the json files were loaded into pandas dataframes before manipulation.

Missing Values

There are no missing values in the label field, either in the form of NaN or as empty strings. All values are members of the set of five categories noted above: 'food', 'drink', 'inside', 'outside', and 'menu'.

Class Imbalance

There is a dramatic degree of class imbalance in the data, with the majority of images being classified as food, and very few being classified as menus. This will likely lead to the model learning to classify high-representation classes well, but performing poorly on the lower-representation classes.

Class	Image Count	Abundance (%)
food	184456	65.64
inside	61620	21.93
outside	23214	8.26
drink	10350	3.68
menu	1352	0.48

Table 1: Class imbalance in the Yelp image set

Duplicate Images

There are a small number of duplicated images in this dataset, which might hinder the evaluation of model performance by contaminating the validation and test sets with images that are also in the training set. However, almost all the duplicate images appear to be stock images, for example a promo image for McDonald's McCaffe. Since these images will likely show up again in future unseen Yelp images, it would be more relevant to this task to keep the duplicates in the dataset, as long as this dataset is, in fact, a random sample of Yelp images. Therefore I have elected not to remove any duplicate images.

Image Sorting

The ImageDataGenerator used to feed images to Keras models expects image files to be separated into different directories based on their classification. Since all the Yelp Dataset images start in one directory when downloaded, Python's system utilities were used to automatically sort them into appropriate directories based on their label.

Mislabeling

A cursory inspection of the images and labels indicates that there is a small degree of mislabeling in the dataset. For example, some images which are labeled as outside are clearly

of indoor areas, and a small amount of images that are labeled as ‘inside’ or ‘drink’ are actually of food. In the case of food being misclassified as drinks, it is usually soft food such as yogurt or ice cream, served in drink-shaped containers such as mugs or deep bowls. In the case of food being misclassified as an indoor area, there are instances where the picture of the food has been taken at such an angle that the indoor area can be seen in the background.

Without manual human inspection of all 280,000 photo/label pairs it is not clear how this issue could be addressed, and for the most part the labels appear to be accurate enough to be useful for this task. The mislabeling present will, however, lower the model’s max possible performance to some degree.

Approach to Model Design and Evaluation

Overview of Approach

CNN architectures are very time consuming to design, and I had limited computing hardware. For this reason, the approach taken for this project was to fine-tune a premade CNN model for transfer learning rather than attempt to design and train an architecture from scratch. The VGG neural network architecture designed by the Visual Geometry Group at Oxford scored second in the 2014 ImageNet classification challenge, and is available as a template in the Keras API for Tensorflow, with or without pretrained weights. This made it a convenient model to use for this task.

CNNs are also notorious for taking enormous computational resources to train. Using the entire set of images for fine-tuning of the model design would be impractical, as training would take a very long time and lead to large gaps between design iterations. Given the class imbalance in the image set, downsampling was used to both equalize the representation of classes, and create a smaller image set on which rapid design iteration could be achieved.

Splitting into training, validation, and test image sets

The 280,992 images were randomly divided into training (230,992), validation (20,000), and test (30,000) image sets. The sampling was not stratified by class; however, inspection of class counts shows nearly identical relative abundance of each class in each set.

Class	Train	Validation	Test
food	65.62	66.21	65.42
inside	21.94	21.32	22.24
outside	8.29	8.20	8.12
drink	3.67	3.76	3.71
menu	0.47	0.53	0.50

Table 2: Class abundances (%) in each image set

Downsampling
Downsampling is an established and effective method for addressing class imbalance. This technique involves randomly sampling the higher-representation classes without replacement to create a dataset where all classes have the same number of instances as the least-represented class. However, this solution comes at the cost of producing an image set that is very small in size. Since there are only 1096 menu images in the train set, and 105 menu images in the validation set, the sizes of these sets after downsampling are 5480 and 525, respectively. This is two orders of magnitude smaller than the 230,992 and 20,000 images in the full training and validation sets, and will likely confer a penalty on overall performance.

Tools
Pandas was used as the main data management and manipulation tool, while Keras API for Tensorflow was used for building and training the CNN. Python's built-in system utilities were used for automated image file management. Sci-kit learn was utilized for various model evaluation metrics. Version numbers for these software tools can be found in the appendix.

The model was trained on an Nvidia GTX 1060 graphics card with 6GB VRAM. The graphics card was fed by an Intel i7-3770k CPU with access to 16GB system memory.

Choice of performance metric
Due to the large degree of class imbalance in this dataset, overall accuracy is not the optimal performance metric. The loss metric will also suffer from a tendency to favor performance on the high-representation classes over performance on the low-representation classes.

Therefore, three additional metrics have been included. The first of these is the macro-averaged recall (MacR, macro-recall), which is the arithmetic mean of the per-class recalls (true positives for that class divided by the class count). Macro-recall thus weighs all classes equally, and is completely insensitive to class imbalance. Conveniently, in the case where all classes have equal representation, such as in the downsampled image set, macro-recall is equivalent to accuracy.

Macro-averaged precision is a less valuable metric in this case due to its sensitivity to class imbalance. In a class-imbalanced dataset, improving the recall of the major classes will improve macro-precision more than improving the recall of the minor classes, so macro-precision has a tendency to favor performance on the major classes over performance on the minor classes. That being said, macro-averaged F1 score (MacF1, macro-F1) and AUC score (MacAUC, macro-AUC) have been included to integrate the precision performance into the model evaluation.

Model architecture
The VGG-16 architecture was imported from the Keras database, and the convolutional layers were kept, since they are the aspect of the model that extracts features from the images. The fully connected layers at the end of the VGG model were discarded, as they are specific to the ImageNet task. The output of the convolutional layers was flattened, and two newly-initialized,

fully connected (FC) layers were added after the flatten layer. Finally, a new output layer with 5 nodes was added to the end of the model.

ReLu was used as the activation function for the FC layers, while the output layer was transformed with softmax. Categorical cross-entropy was used as the loss function, and Adam was used as the optimizer.

Keras layer	Shape	Param #
InputLayer	(224, 224, 3)	0
Conv2D	(224, 224, 64)	1,792
Conv2D	(224, 224, 64)	36,928
MaxPooling2D	(112, 112, 64)	0
Conv2D	(112, 112, 128)	73,856
Conv2D	(112, 112, 128)	147,584
MaxPooling2D	(56, 56, 128)	0
Conv2D	(56, 56, 256)	295,168
Conv2D	(56, 56, 256)	590,080
Conv2D	(56, 56, 256)	590,080
MaxPooling2D	(28, 28, 256)	0
Conv2D	(28, 28, 512)	1,180,160
Conv2D	(28, 28, 512)	2,359,808
Conv2D	(28, 28, 512)	2,359,808
MaxPooling2D	(14, 14, 512)	0
Conv2D	(14, 14, 512)	2,359,808
Conv2D	(14, 14, 512)	2,359,808
Conv2D	(14, 14, 512)	2,359,808
MaxPooling2D	(7, 7, 512)	0
Flatten	(25088)	0
Dense	(500)	12,544,500
Dropout	(500)	0
Dense	(500)	250,500
Dropout	(500)	0
Output	(5)	2,505

Table 3: Layers of the adapted VGG-16 architecture

Model Hyperparameters

Approach to tuning

Tuning hyperparameters in neural networks is a very random process, as the ultimate performance of a model depends on the random initialization state of the weights, which is different for each new model instance trained. For this reason, each condition in the tests below was run in triplicate. Any model that did not converge to >70% accuracy was discarded. The mean performance among the models that did converge within each condition is reported below. Any condition which is reported as 'did not converge' (DNC) had no models converge out of the three replicates. Macro-recall is not reported in the performance tables in this section due to its being equivalent to accuracy in the downsampled image set.

While having more replicates within each condition is preferred to get a more accurate picture of the expected performance across different hyperparameter values, this requires more computing time.

ImageNet weights vs new weights

The first aspect of the model to investigate is whether the convolutional weights pretrained on the ImageNet dataset are useful for this dataset. To evaluate this, three model conditions were compared. In the first condition, the ImageNet weights were discarded and all weights were trained from scratch. In the second condition, the ImageNet weights were kept, but the model was not able to alter the weights of the convolutional layers during training. In the final condition, the ImageNet weights were kept, and the model was able to further train these weights to fine-tune the feature extraction for this dataset.

Keeping and further training the ImageNet weights performed the best, while the baseline ImageNet weights outperformed new weights trained from scratch (Table 4). Therefore, the ImageNet weights are useful for this task, and can be made to perform even better by fine-tuning them to this dataset.

Image augmentation

The next aspect of the model that needs to be implemented is image augmentation. Image augmentation is the act of randomly altering images to increase, in a way, the amount of data the model has available to learn from. Since image augmentation has a regularizing effect (i.e. it makes the model less likely to overfit) it is important to establish image augmentation before tuning parameters such as model capacity.

Image augmentation, however, needs to be executed in a way that is sensible for the task. Flipping images upside down, for example, is not a sensible transformation because people do not usually upload upside-down images to Yelp. Flipping horizontally was included because food is food regardless of whether the meat is to the left or right of the potatoes, and an indoor space is an indoor space regardless of whether the door is to the left or right of the bar. One aspect of this task that could be affected by a horizontal flip is the text on menus, however there are other features of the images that the model can use to identify this class.

Random horizontal and vertical shifting were also performed (up to 20% of total width/height), as users may choose to upload off-center images for stylistic reasons. Finally, a random amount of zoom (up to 20%) was applied because users may take images at differing distances from the object of interest. Any gaps in the image caused by shifts were filled by reflection of the image across the border.

The model trained using image augmentation had similar performance on the validation set as the model without image augmentation. However, the performance on the training set decreased to come more into line with the performance on the validation set. This suggests that the model trained without image augmentation was overfitting to some degree. The lack of improvement in validation performance could indicate that the performance is limited by some other factor, such as a model capacity that is too small.

Convolution Weights	Mean Acc	Mean Loss	Mean F1	Mean AUC
ImageNet Baseline	0.877	0.904	0.878	0.979
ImageNet Trainable	0.872	0.375	0.872	0.981
New Weights	0.707	0.763	0.708	0.924
ImageNet with Augmentation	0.854	0.381	0.853	0.980

Table 4: Performance on downsampled validation set across different weight conditions

Model capacity

The number of nodes in the two FC layers is an important hyperparameter to tune, as it affects the tendency of the model to overfit vs generalize. Having FC layers with too many nodes will lead to a model which will overfit to the training data, and have worse performance on the validation and test data. Having FC layers with too few nodes will lead to poor performance on both the training and validation images, as there will not be enough model capacity to capture all the important patterns in the dataset.

To find the optimal amount of nodes, the model was trained with a FC width ranging from 400 to 1600, in 200 node increments. The model improved in accuracy and loss up to a width of 800-1000 nodes. At around 800 nodes and greater, the model's performance was essentially the same. This indicates that 800-1000 nodes is the minimum width required for optimal performance on the downsampled image set. The model might be able to make use of more capacity on the full image set, but to optimize this parameter on the full set would require too much computation time to be feasible for this project. 1000 nodes was chosen as the model width because it is the safe choice for having at least the minimum capacity needed.

Width	Mean Acc	Mean Loss	Mean F1	Mean AUC
400	0.857	0.403	0.857	0.977
600	0.878	0.327	0.879	0.984
800	0.884	0.306	0.884	0.985
1000	0.886	0.305	0.885	0.985
1200	0.888	0.311	0.888	0.983
1400	0.882	0.330	0.882	0.983
1600	0.887	0.284	0.887	0.987

Table 5: Performance on downsampled validation set across different widths

Learning rate

The learning rate of the Adam optimizer is also an important hyperparameter to tune. A learning rate that is too low or too high could lead to the model finding a less optimal solution, or failing to learn at all.

The model was tested on learning rates of 0.0005, 0.0002, 0.0001, 0.00005, and 0.00002. A learning rate of 0.0001 was optimal, with 0.00005 performing similarly, 0.0002 performing slightly worse, and 0.00002 producing a much worse solution, according to the validation loss. A learning rate of 0.0005 did not converge to a solution.

Learning Rate	Mean Acc	Mean Loss	Mean F1	Mean AUC
0.0005	DNC	DNC	DNC	DNC
0.0002	0.815	0.608	0.814	0.959
0.0001	0.876	0.325	0.875	0.984
0.00005	0.879	0.330	0.880	0.984
0.00002	0.863	0.370	0.861	0.982

Table 6: Performance on downsampled validation set across different learning rates

Dropout

Dropout is a powerful regularization technique used to boost validation set performance. In this technique, a proportion of the nodes in a given layer are randomly selected to be dropped for each training batch. The remaining nodes are then trained on that batch. This forces the model to store a more robust, redundant understanding of the dataset across its nodes. Dropout thus allows for a larger model capacity without overfitting the training data.

Dropout was tested for each FC layer individually. FC layer width was increased in proportion to the degree of dropout. For example, at a dropout of 0.2, width was increased to 1250 so that 1000 nodes were active after 20% were randomly dropped. Dropout on the first layer was tested first, as earlier dropout usually has a greater impact on the model.

Dropout proportions ranging from 0.1 to 0.4 in the first FC layer did not improve the performance of the model over a baseline of 0 dropout. Testing dropout in the second FC layer revealed that a dropout of 0.2 performed best.

Dropout 1	Mean Acc	Mean Loss	Mean F1	Mean AUC
0	0.887	0.302	0.886	0.986
0.1	0.885	0.320	0.885	0.985
0.2	0.868	0.349	0.867	0.982
0.3	0.848	0.382	0.847	0.979
0.4	0.863	0.372	0.862	0.979

Dropout 2	Mean Acc	Mean Loss	Mean F1	Mean AUC
0	0.872	0.353	0.872	0.982
0.1	0.874	0.330	0.874	0.982
0.2	0.896	0.294	0.896	0.986
0.3	0.865	0.343	0.865	0.982
0.4	0.872	0.353	0.872	0.982

Table 7: Performance on downsampled validation set across different dropouts

Final model design

In summary, the final model design that was adopted for this task includes:

1. VGG-16 convolutional layers with pretrained weights from the ImageNet competition, with the capability of further tuning these weights for this dataset
2. Two fully connected layers with ReLU activations. The first layer is composed of 1000 nodes, while the second is composed of 1250 nodes (to account for dropout)
3. An output layer composed of 5 nodes, with softmax activation and categorical cross-entropy as the loss function
4. Image augmentation in the form of horizontal flip, horizontal and vertical shift, and zoom, with fill by reflection
5. A learning rate of 0.0001 for the Adam optimizer
6. No dropout regularization in the first FC layer, and a dropout of 0.2 in the second FC layer

Solving Class Imbalance

Performance of downsampled model

The model trained on the downsampled image set achieved an accuracy of 88.9%, macro-recall of 89.2%, and macro-F1 of 0.747 when predicting the full validation set (Table 9). The class recalls (the diagonal values in the normalized confusion matrix below) are all relatively high, with the model having the most difficulty distinguishing inside from outside. This performance will serve as a baseline, and any models which use the full image set for training will need to exceed this performance in order to be adopted in favor of the downsampled model.

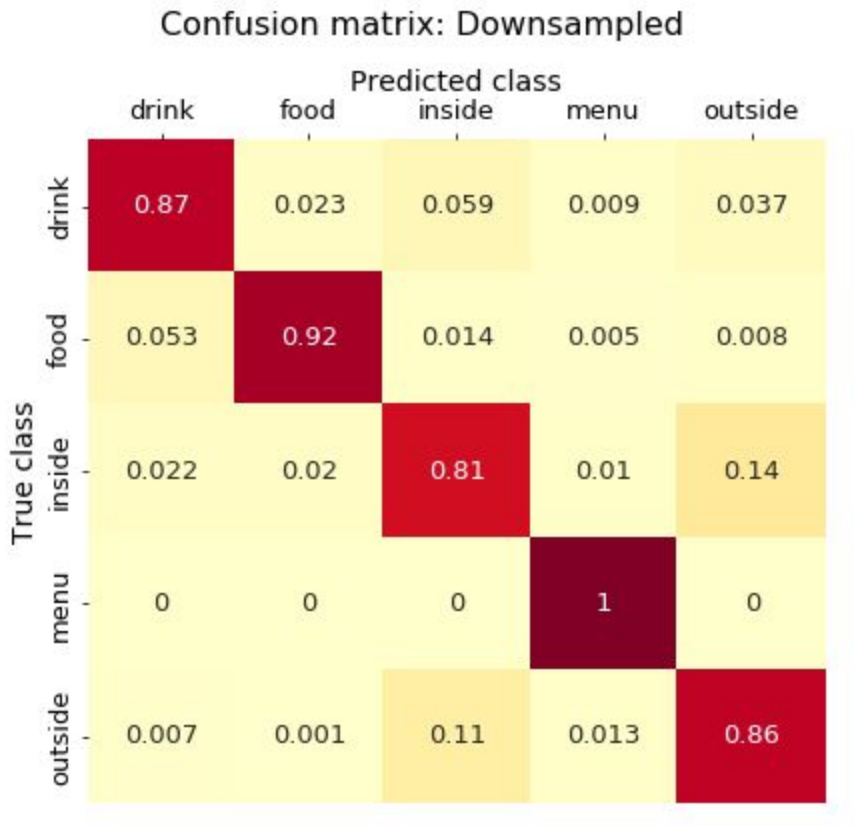

Figure 1: Confusion matrix for full validation set, normalized by class size

Training on the full image set

5 epochs of training on the full training set of 220,992 images yielded a decent overall accuracy of 93.1% (Table 9). However, inspection of the validation confusion matrix and class recalls reveals poor performance on the low-representation classes. This is a direct result of the class imbalance present in the dataset. While the recall of the highest-represented class (food) was 99%, the recall of ‘menus’ was an abysmal 21%. The recall for ‘drink’ was also poor, at 47%. The macro-recall was a low 67.8%, while the macro-F1 dropped to 0.721. This model will be referred to as ‘full-unbalanced’.

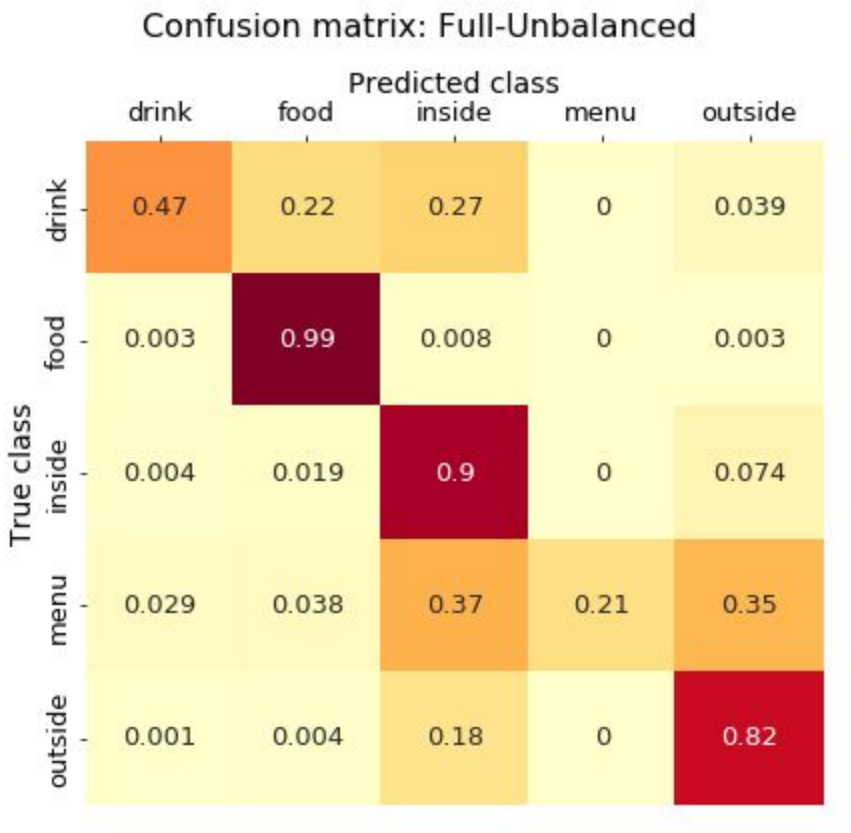

Figure 2: Confusion matrix for full validation set, normalized by class size

This performance is unacceptable for a model that needs to be able to label all classes accurately, regardless of how often they are uploaded. However, the performance of the model trained on the downsampled image set is similarly unimpressive. Therefore, the issue of class imbalance needs to be addressed in some manner that retains the benefit of the large amount of images in the full training set.

Balanced class weights

A simple approach to solving the class imbalance is to alter the relative contributions of the classes to the loss function. Class weights are said to be ‘balanced’ when the weight of each class is equal to the inverse of its abundance relative to the most abundant class. So if the major class in a binary classification makes up 80% of the dataset, the minor class will have a weight of four times the major class. This leads to each class having equal total impact on the loss function.

Class	Image Count	Weight	Root Weight
food	151588	1	1
inside	50684	2.99	1.73
outside	19138	7.92	2.81
drink	8486	17.86	4.23
menu	1096	138.31	11.76

Table 8: Balanced class weights and square-root weights for full training set

5 epochs of training on the full image set with balanced class weights (Table 8) yielded an overall accuracy of 86.0%, macro-recall of 84.1%, and macro-F1 of 0.688 (Table 9). While this is a large improvement in macro-recall over the model trained without balanced class weights, it is strictly worse performance than the model trained on the downsampled image set. This model will be referred to as 'full-balanced'.

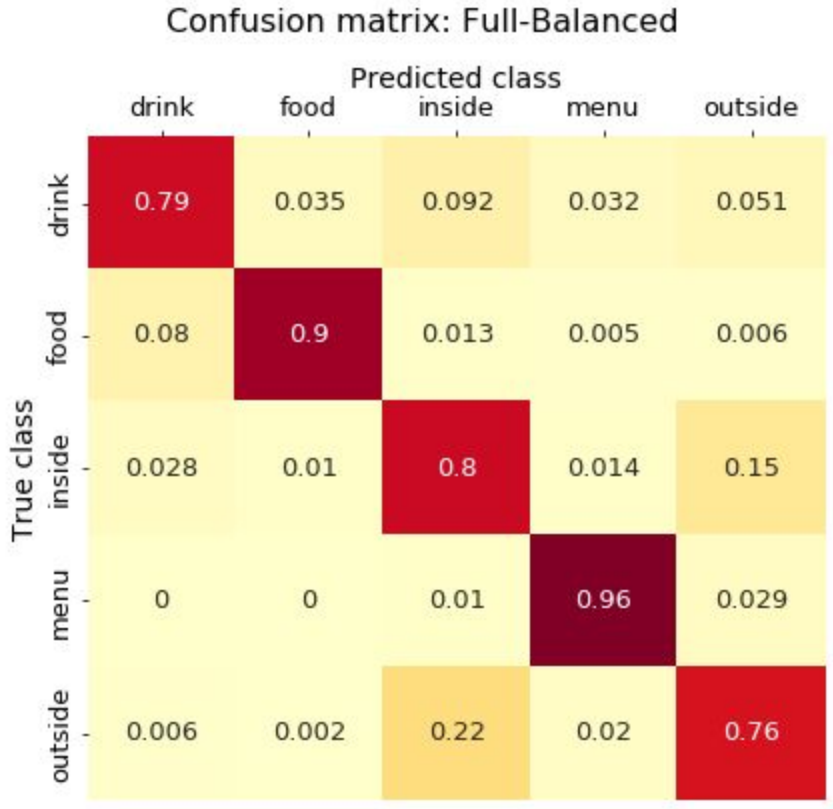

Figure 3: Confusion matrix for full validation set, normalized by class size

Combining models

Another potential solution is to combine both the model trained on the full image set and that trained on the downsampled image set. A new Keras model was created such that the input was fed into both models, and the outputs of the models were combined into a 10-node layer. Then, a new output layer of 5 nodes was trained on the downsampled image set to learn the patterns in this 10-node layer.

This combined model showed better performance than the downsampled model, achieving both an overall accuracy and macro-recall of 90.4%, and a macro-F1 of 0.759 (Table 9). These are the best values for macro-recall and macro-F1 so far. However, this solution still returns rather unexciting performance, and comes at the cost of requiring twice the amount of computation to make a prediction.

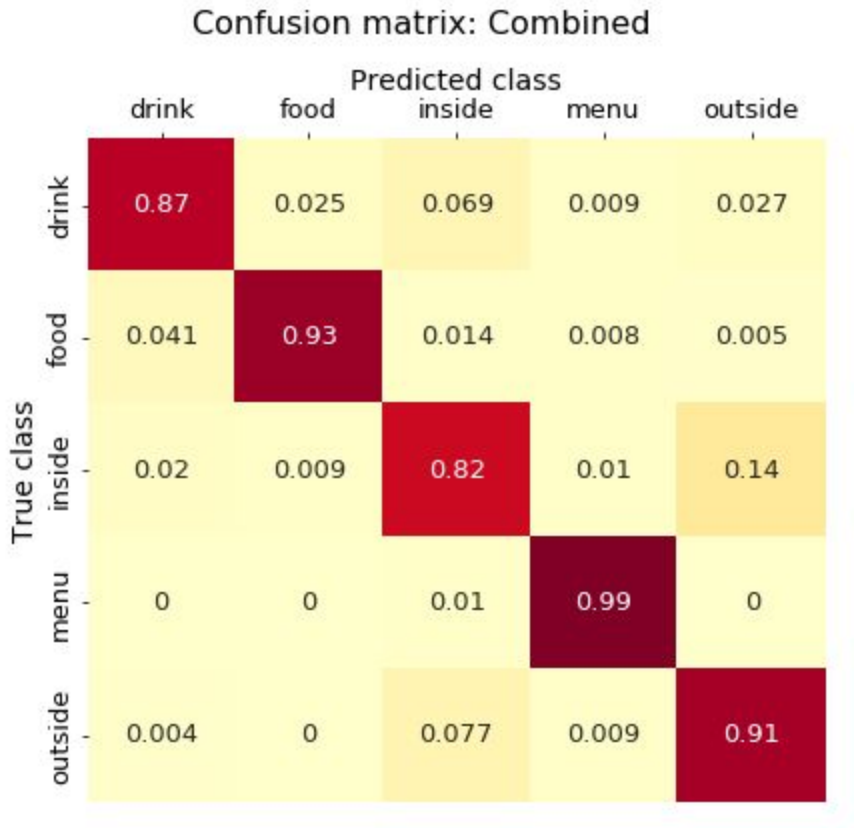

Figure 4: Confusion matrix for full validation set, normalized by class size

Hybrid training (two-phase training)

Another way to gain the benefits of both the downsampled and full image sets is to train the same model on both. In this technique, the model is first trained to convergence on the downsampled image set to establish decent performance on all classes. Then, the model is trained on the full image set to gain the benefits of the large set size for boosting performance on the higher-representation classes. In this implementation of hybrid training, all layers were allowed to update during the second phase. Balanced class weights were used in the second phase to prevent the decay in performance on the lower-representation classes that would occur if the higher-representation classes were mostly driving the gradient descent of the loss function.

After the initial training on the downsampled image set, training on the full image set for 5 epochs led to a model that achieved an accuracy and macro-recall of 90.2%, and a macro-F1 of 0.772 (Table 9). This roughly matches the performance of the combined model while only requiring half the computing power. This model will be referred to as ‘hybrid-balanced’.

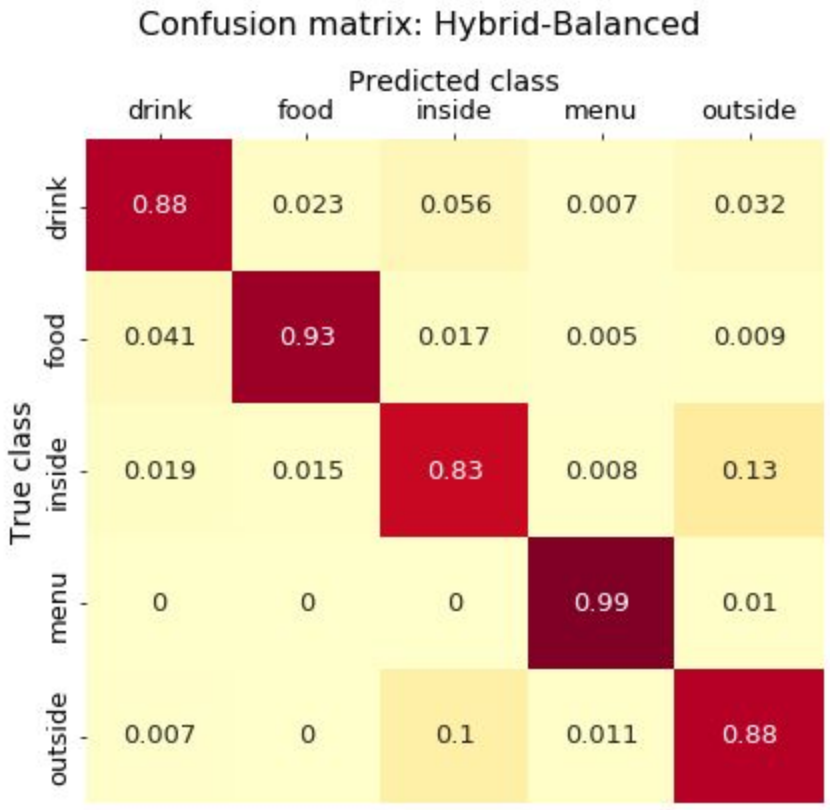

Figure 5: Confusion matrix for full validation set, normalized by class size

The dramatic class imbalance leads to very high weights for the lower-representation classes. Combined with the very low amounts of images in these classes, this might cause the stochastic gradient descent to become very noisy, since a minority of batches will have the low-representation, very-highly-weighted ‘menu’ and ‘drink’ classes. It also might cause the model to be too rigidly locked into maintaining good performance on the minor classes to allow it to improve much on the major classes.

For these reasons, a milder weighting scheme was tested, where the weight of each class in the second training phase was the square root of its weight in the balanced case (Table 8). This weighting scheme led to an accuracy of 93.6%, a macro-recall of 90.1%, and a macro-F1 of 0.855 after 5 training epochs (Table 9). This is by far the best performance yet, almost matching the macro-recall of the hybrid-balanced and combined models, exceeding the overall accuracy of the simple-unbalanced model, and achieving the highest macro-F1 out of any model by a large margin. The macro-F1 improved to such a dramatic degree because of the improved recall on the major classes, which greatly improved the precisions of the minor classes by reducing false-positive rates. In fact, the F1 score of every class improved relative to the hybrid-balanced model. This model will be referred to as ‘hybrid-root’.

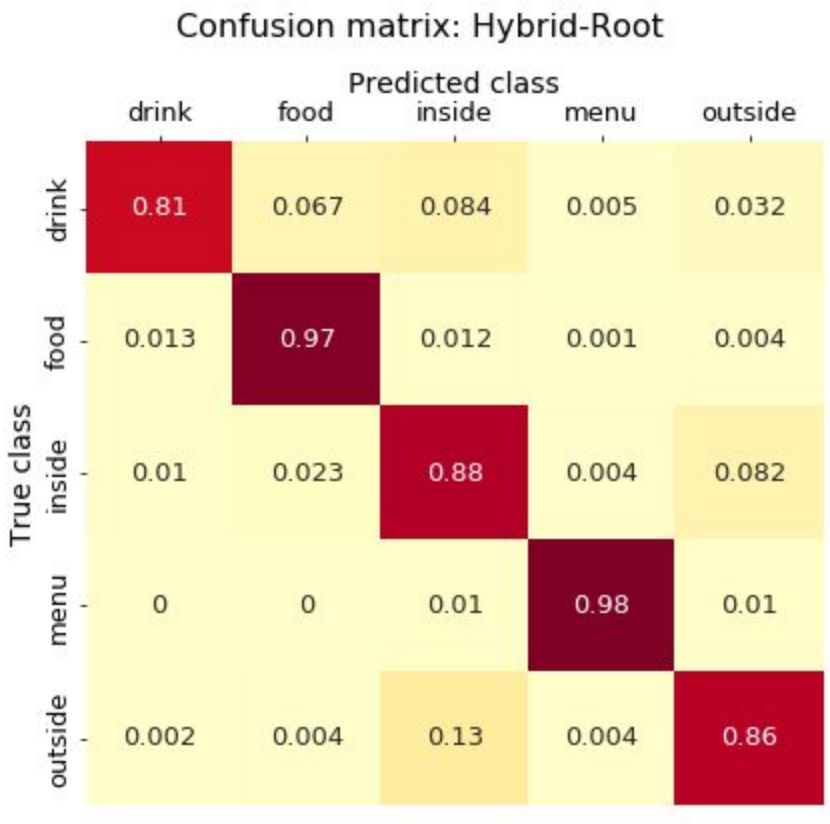

Figure 6: Confusion matrix for full validation set, normalized by class size

Comparison of class imbalance approaches

Out of all these approaches to solve the class imbalance, hybrid-root performed the best, with an accuracy of 93.6%, macro-recall of 90.1%, and macro-F1 of 0.855 (Table 9). Macro-AUC was very high for all models, but hybrid-root also had the highest score on this metric, at 0.991. The success of this model demonstrates the power of hybrid training in overcoming class imbalance.

The success of hybrid-root also demonstrates the importance of tuning the class weights in hybrid training, to allow the model enough flexibility to improve in the major classes while maintaining its good performance on the minor classes. A more thorough tuning regiment could test a variety of powers from 0 (equal weighting) to 1 (balanced weighting) with which to transform the class weights, instead of just 0.5 for the square root.

	Overall			
Model	Accuracy	MacR	MacF1	MacAUC
Downsampled	88.95	0.892	0.747	0.986
Full-Unbalanced	93.10	0.678	0.721	0.986
Full-Balanced	85.98	0.841	0.688	0.976
Combined	90.43	0.904	0.759	0.990
Hybrid-Balanced	90.24	0.902	0.772	0.989
Hybrid-Root	93.61	0.901	0.855	0.991

	Class F1s				
Model	Drink	Food	Inside	Menu	Outside
Downsampled	0.594	0.954	0.846	0.595	0.746
Full-Unbalanced	0.607	0.983	0.879	0.346	0.790
Full-Balanced	0.468	0.942	0.824	0.523	0.682
Combined	0.641	0.963	0.860	0.552	0.781
Hybrid-Balanced	0.649	0.960	0.859	0.628	0.764
Hybrid-Root	0.773	0.979	0.888	0.824	0.811

Table 9: Validation performance of different class imbalance approaches

Final Model Performance

Performance on the test set

Evaluating the performance of the hybrid-root model on the test set of 30,000 images yielded an accuracy of 93.4%, macro-recall of 89.5%, and macro-F1 of 0.849. This is similar to the validation set performance, and indicates that the model has learned a generalizable conception of the classes. Training the model for more than 5 epochs in the second phase could lead to further improvement in performance.

	Overall			
Model	Accuracy	MacR	MacF1	MacAUC
Hybrid-Root	93.43	0.895	0.849	0.991

	Class F1s				
Model	Drink	Food	Inside	Menu	Outside
Hybrid-Root	0.774	0.979	0.889	0.809	0.793

Table 10: Performance on the test set

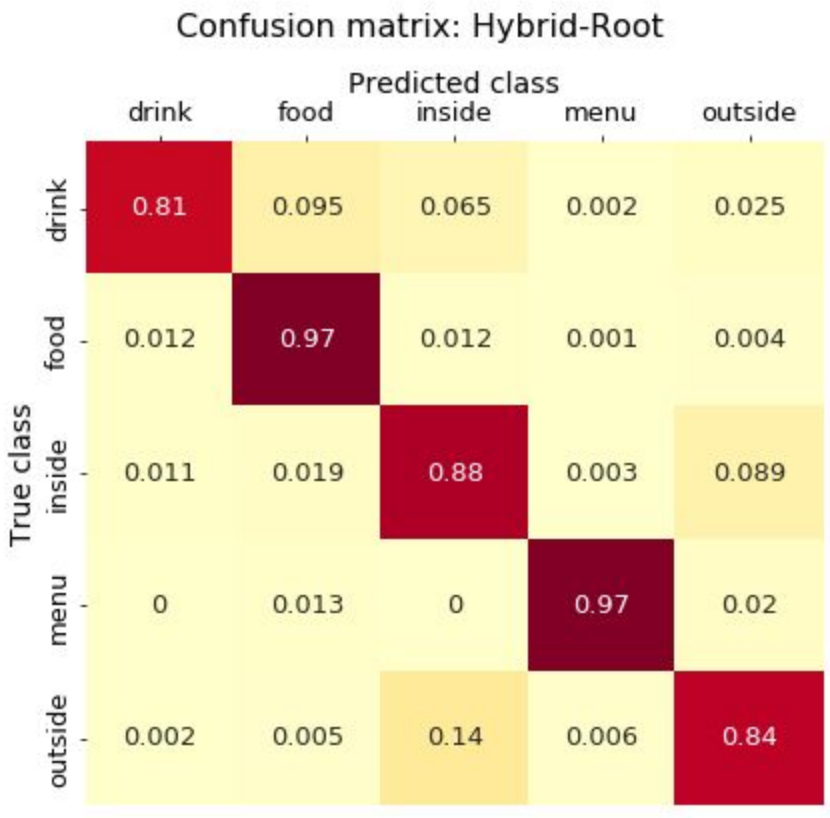

Figure 7: Confusion matrix for full test set, normalized by class size

Main sources of error

Inspection of the raw confusion matrix (Figure 8) reveals that a large proportion of the errors made by the model involve confusing images between the ‘outside’ and ‘inside’ classes. As mentioned previously, there is some degree of misclassification between these categories in the image set, and this may be negatively impacting the performance of the model. It also may simply be a difficult task for the model to distinguish between images taken inside and those taken outside, as this is a pretty abstract distinction.

Other notable sources of error include confusion among the food, drink, and inside classes. This may be due to the presence of images in the Yelp Dataset that contain both food and drink, or a food or drink item with the restaurant interior visible in the background. Therefore these errors may be due in part to misclassification in the dataset or images that are ambiguous and difficult to label as a single class.

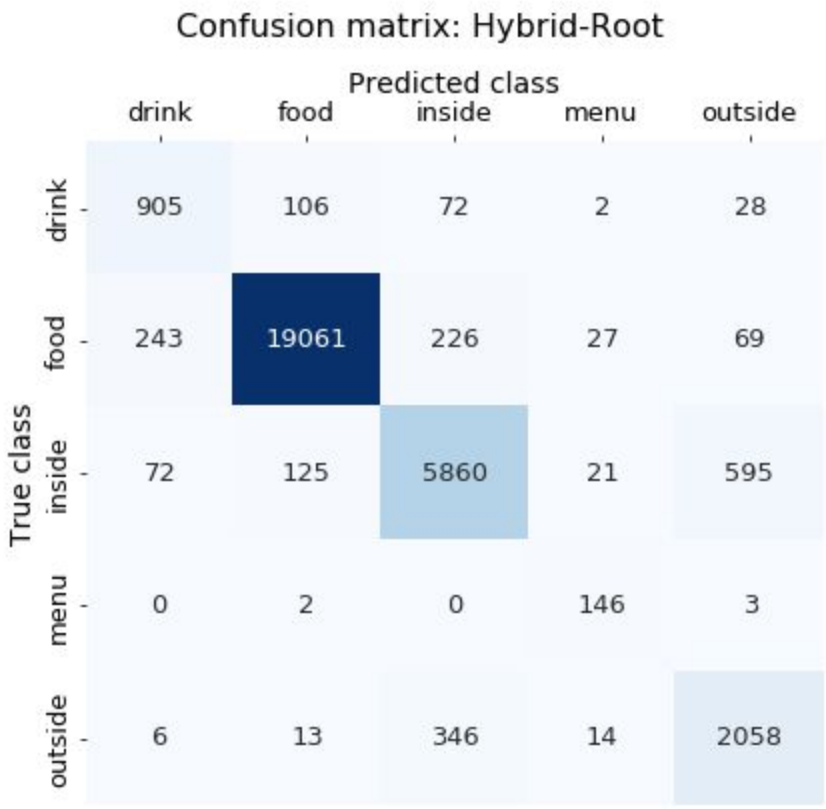

Figure 8: Raw confusion matrix for Hybrid-Root on full test set

Other Possible Directions

Custom architecture

With more computing power available, a CNN architecture could be built from scratch that is tailored for this task. Since the ImageNet problem is more complex than this one (far more images and classes), the VGG-16 model may be wider and/or deeper than required for this task. A leaner architecture tailored for this task could be run through more conditions and replicates during tuning and therefore get a more accurate picture of the best architecture.

Making a custom architecture would also enable easy implementation of dropout across all the convolutional layers in addition to the FC layers, which is the more standard dropout approach. Finally, advanced techniques such as batch normalization and replacing pooling steps with learnable stride-2 convolutional layers could be tested with a custom architecture.

Class imbalance

Other potential solutions to the class imbalance problem might be tried out in the future.

A simple possibility is to change the training of the model such that, for every epoch, a new random sample of the non-menu classes is chosen for the downsampled image set. Since the model is then able to learn from all the images in the full set, given enough training epochs, this

may lead to even better performance than hybrid training. However, implementing this kind of image pipeline in Keras is non-trivial and would likely require manual modification of the ImageDataGenerator or its flow_from_directory method.

With more computing power, upsampling is an approach that could be explored. In upsampling, the images in the minor classes are duplicated up to the image count of the highest representation class. This allows the model to learn from all the images in the higher representation classes while keeping the number of instances it sees for each class balanced. The downside of this technique is that it greatly expands the size of the image set, and therefore the amount of time required to train the model, without actually increasing the amount of information in the image set.

Another possibility is to split the classification task into 5 binary one-versus-all problems. In this scheme, the convolutional layers would be fine-tuned to the full image set using a simple two-FC layer architecture (possibly with two-phase training). Then, the FC layers would be dropped and two new layers and a single output node would be trained for each class. The training image set would be different for each class, and would be balanced for that particular class. For example, there are 50,684 images of the ‘inside’ class, so the training set for the ‘inside’ one-versus-all network would consist of those 50,684 ‘inside’ images, and 50,684 images randomly sampled without replacement from all the non-’inside’ images. The training set for the ‘outside’ one-versus-all network would be the 19,138 ‘outside’ images, and 19,138 from the non-’outside’ classes.

After all the one-versus-all networks were trained, they would be combined into a new CNN architecture such that the convolutional layers fed their output into each one-versus-all network, and the five binary outputs of these networks would be the new output layer, with the highest-probability node being selected as the class prediction. In this way, the convolutional layers will only need to be calculated once for each image. Furthermore, since each FC branch has had the benefit of training on an image set tailored to its class, each branch should have very good performance for its particular class. This approach could also be adapted to produce a multiple-positive classifier, to account for the fact that some images in the Yelp dataset contain combinations of food, drinks, menus, and setting. Essentially, the classifier would be answering the question ‘Is there a drink in this image?’ rather than ‘Is this an image of a drink?’. The downside of this technique is, of course, the extra effort of creating 5 different datasets, and the longer time needed to train all the one-versus-all networks.

References

Yelp Dataset:
https://www.yelp.com/dataset/challenge

Keras VGG-16 implementation:
https://keras.io/applications/#VGG-16

VGG-16 Architecture:
Simonyan and Zisserman. (2015) 'Very Deep Convolutional Networks for Large-Scale Image Recognition.' arXiv:1409.1556. 10 Apr 2015.

Addressing Class Imbalance in CNNs:
Buda et al. (2017) 'A systematic study of the class imbalance problem in convolutional neural networks.' arXiv:1710.05381v1. 15 Oct 2017.

Hybrid training (a.k.a. two-phase training):
Havaei et al. (2016) 'Brain Tumor Segmentation with Deep Neural Networks.' arXiv:1505.03540v3. 20 May 2016.

Appendix

Software Versions:

Python	3.6.4
Anaconda	4.5.4
Numpy	1.15.1
Pandas	0.23.4
Scikit-learn	0.19.1
Tensorflow-gpu	1.10.0
Keras	2.2.2
CUDA	9.0.176

Windows 7