Yelp Restaurant Multilabel Classification using Images from Reviews

Team: Samantha Chu, Jenil Kansara, Shweta Malla

BUDT758B

Big Data and Artificial Intelligence

Instructor: Dr. Kunpeng Zhang

Introduction

Yelp is a popular online and mobile application especially known for allowing users to review, upload pictures, and comment on food establishments and/or restaurants for other users to see. In addition, restaurants themselves can provide the basic information about their business and label themselves using certain characteristics, such as, “outdoor seating”, “takeout”, “casual dining.” On Yelp, users can upload photos along with their reviews for restaurants.

Having contextual information about those images in regards to the characteristics of the restaurant can significantly improve the quality of the classification of the restaurants on Yelp’s database. Having these insights about the restaurants would improve the customer recommendation system, search results and the company will be able to provide better services to its customer base.

During this project, we worked with a user-submitted image dataset from Yelp with 9 possible labels. We worked on this problem, which is a multi-instance, and multi-label classification problem, using Convolutional Neural Network (CNN) in Python using keras deep learning library. CNN is commonly used to analyze visual image data. With its architecture supporting analysis of hierarchical pattern analytics, CNN was a fitting choice for the concerned problem. Some advantages of using CNN also include the ability to detect (1) small patterns within a whole image, (2) patterns in images appearing in different regions (a pattern detector), and (3) the ability to subsample the the pixels to reduce the size of the image and therefore reduce the number of parameters for the network to process.

We faced problems relating to managing and working with a large dataset containing more than 230,000+ images, finding a suitable environment to perform the model training, and handling hyperparameters. At the end, we achieved AUC score of 0.80 and on test data, F1 Score of 0.74**,** which is close to the highest F1 achieved in the Kaggle competition, which was 0.83.

The ability to accurately and quickly classify restaurant images is both an interesting and an important problem to solve because this algorithm can be applied to many other problems. Upon further improvement of the model, this trained neural network can be transferred to a multitude of different problems in different industries and especially for those industries that benefit from computer vision tasks. By refining our model, and using a very specific, deep, and large convolutional neural network architecture we are confident that it will perform well in large-scale image recognition tasks. The general idea of our method is to classify images into multi-label categories, as opposed to single labels. We used the CNN architecture to achieve the best performance due to the characteristics of CNN mentioned above.

Existing Methods

This problem is a multi-class classification problem and therefore has some challenges. The classes we intend to categorize the restaurants into include “good for lunch”, “good for dinner”, “takes reservations,” etc. Many of these labels are abstract and can be difficult to ascertain from an image of the restaurant. Multi-class classification problems have been tackled using methods such as Neural Networks, K-nearest neighbors, Naive Bayes, Random Forests, etc.

It is a challenging problem due to the complex nature of the images in real life. Simple Neural Networks can be used as a baseline comparison model for the problem; both fully connected neural networks and CNN have weights and biases, however with each neuron connected with other neurons in subsequent layers, the number of parameters exceeds tremendously. As our dataset is large in size, we decided to avoid training a model using a fully connected neural network considering the timeline of the project.

In CNN, the main functional difference is that the image matrix is reduced to a lower dimension through Convolution, which makes it faster and less prone to overfitting compared to fully connected neural networks.

Data: Description and Characteristics

The dataset, obtained from Kaggle but originally provided by Yelp, has images submitted by users for 2000 restaurants. The dataset includes and matches business IDs to photo IDs, and business IDs to business labels. The restaurant businesses themselves, not the images, are labeled, sometimes with more than one of the following labels: (1) “good for lunch”, (2) “good for dinner”, (3) “takes reservations”, (4) “outdoor seating”, (5) “restaurant is expensive”, (6) “has alcohol”, (7) “has table service”, (8) “ambience is classy”, (9) “good for kids”.

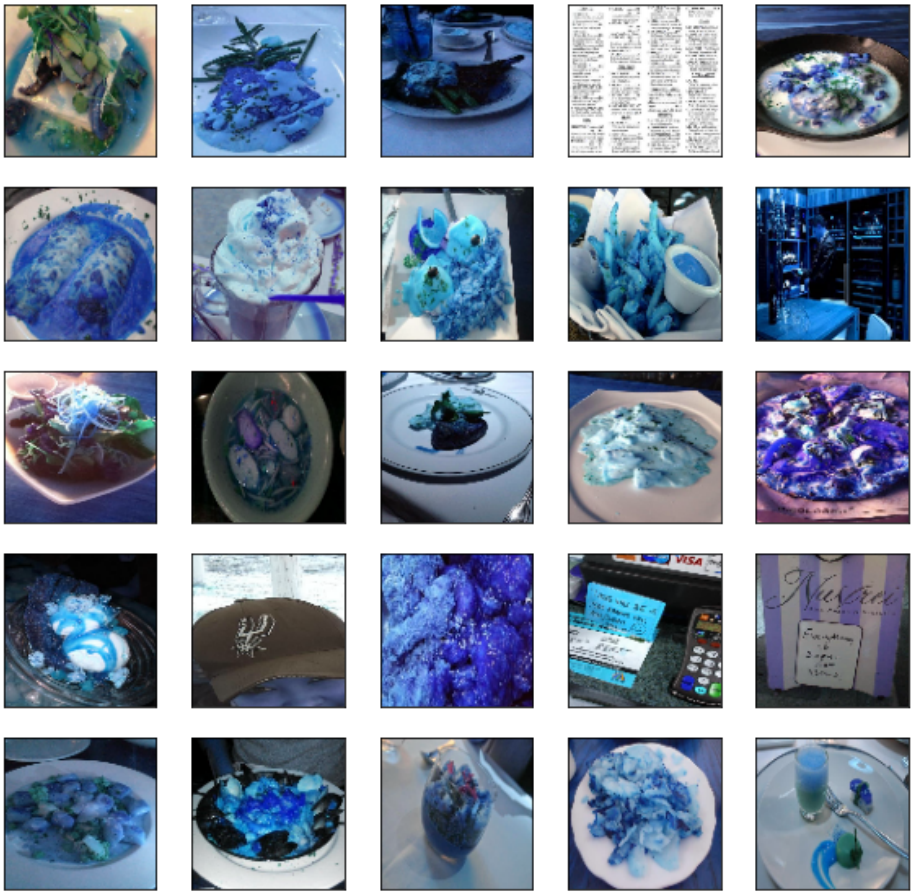
One of the challenges was that the photos themselves did not have labels to them but rather the businesses had the labels and we had to merge these two datasets accordingly. In addition, there were a different number of images for each business. There was also the chance of having duplicate photos, however, we concluded that having duplicate photos will not affect the overall training and learning of the model, and would be too cumbersome, computationally, to remove. Finally, different photos contained different information pertaining to different labels. For example, an image of a drink would yield different results and patterns compared to an image of the interior of the restaurant.

Our Methods

From the beginning of the project, the team made a lot of efforts to figure out how to handle the 234,842 images exceeding storage size of more than 6 GB.

The initial step was to pre-process and normalize the images. To perform that, we first needed to get the images on Google Drive to be accessed by Google Colab. Due to the inability to upload images directly, considering the large number of images, we decided to upload the zip file containing the images on Google Drive and extracted that using Python in Google Colab. After getting a basic infrastructure set up for the project, we then reshaped the images to 100 by 100 pixels with 3 color channels giving an array of

(100, 100, 3) for each image. Next, we normalized each channel to the interval (0,1), to help the algorithm learn faster and allow the learning rate hyperparameter to be fined-tuned.



After pre-processing and normalizing the images, they were saved along with their labels in an h5 file for better access moving forward. After getting our dataset ready to go for model training, we implemented a multi-layer convolutional neural network (CNN) to do image classification, including both convolution layers and pooling layers.

A CNN is architected using the following components

1) Input Layer

   The images are given as an input to the model, and as the images are in RGB 3-channel format the input dimensions are Number of Images X Channel (3) X Height (100) X Weight (100)

2) Convolution Layer

   A convolution layer is a tensor/matrix, which applies a convolution operation on the input layer. The layer has a matrix, with a dimension smaller than the input layer, but using this layer, a dot product is found on a portion of the image with a dimension similar to the convolution layer's weights. The sum is the output of the layer.

3) Activation Layer

   Here, ReLU is used for the layers except the last one. ReLU can be mathematically explained as y=max(0,x). A benefit of using ReLU over sigmoid function is the reduced likelihood for vanishing gradient phenomenon. Note that sigmoid is used in the last layer as the output of the model will be a probability of an image belonging in the concerned class/es.

4) MaxPooling Layer

   Maxpool is used to reduce the dimension of the data that passes to the next step. Usually it is a square matrix and the maximum value from a small set of values is sent to the next step.

5) DropOut

   Neural nets can sometimes overfit the training data, and to overcome that, dropout is one regularization method. During the training process, while dropout is employed, some of the nodes would randomly dropout. By dropping out, it means that that specific node will be removed from that training iteration, with all its input and output connections.

Moving forward, the hyperparameters are set and the model training process began. Batch size was set to 100 and the model was run for 20 epochs. With each iteration, loss and AUC value was printed as output, as shown in the following image.

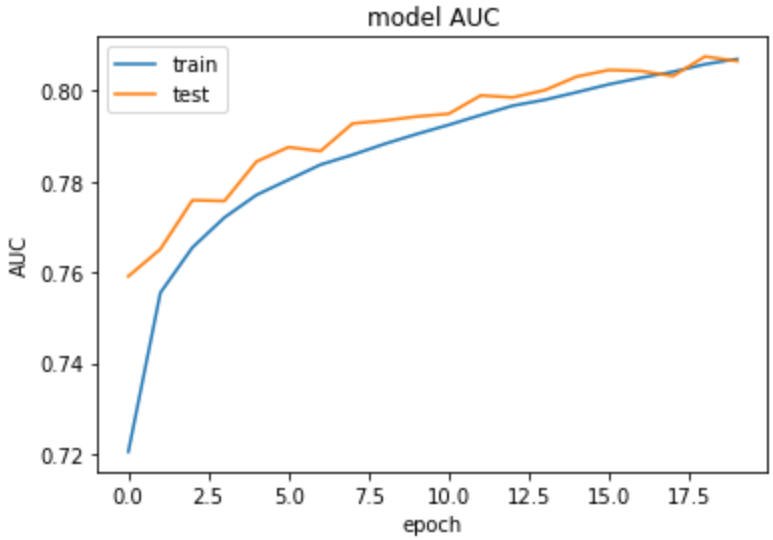
The dataset originally had 2000 businesses and about roughly ~230,000 images in the training set. Due to limited time and lack of computational power and time, we used a total of 150,000 images, with our training set containing 70% (105,000 images), validation containing 15% (22,500 images), and testing set containing 15% (22,500 images).

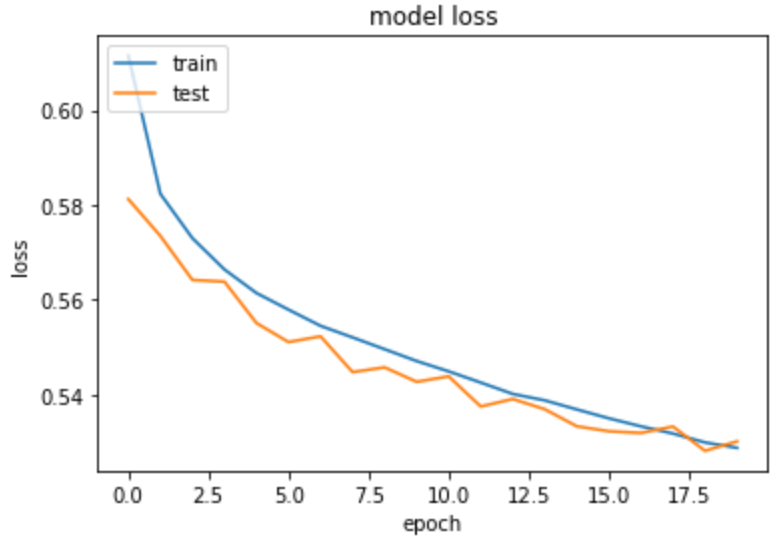
Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 100, 100, 32)	896
activation_12 (Activation)	(None, 100, 100, 32)	0
conv2d_9 (Conv2D)	(None, 98, 98, 64)	18496
activation_13 (Activation)	(None, 98, 98, 64)	0
max_pooling2d_4 (MaxPooling2	(None, 49, 49, 64)	0
dropout_6 (Dropout)	(None, 49, 49, 64)	0
conv2d_10 (Conv2D)	(None, 49, 49, 64)	36928
activation_14 (Activation)	(None, 49, 49, 64)	0
conv2d_11 (Conv2D)	(None, 16, 16, 64)	36928
activation_15 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_5 (MaxPooling2	(None, 8, 8, 64)	0
dropout_7 (Dropout)	(None, 8, 8, 64)	0
flatten_2 (Flatten)	(None, 4096)	0
dense_4 (Dense)	(None, 512)	2097664
activation_16 (Activation)	(None, 512)	0
dropout_8 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 9)	4617
activation_17 (Activation)	(None, 9)	0

Total params: 2,195,529
Trainable params: 2,195,529
Non-trainable params: 0

```
Epoch 1/20
1050/1050 [==============================] - ETA: 0s - loss: 0.6116 - auc: 0.7205
Epoch 00001: loss improved from inf to 0.61157, saving model to weights.01-0.58126.hdf5
1050/1050 [==============================] - 793s 756ms/step - loss: 0.6116 - auc: 0.7205 -
Epoch 2/20
1050/1050 [==============================] - ETA: 0s - loss: 0.5824 - auc: 0.7555
Epoch 00002: loss improved from 0.61157 to 0.58237, saving model to weights.02-0.57349.hdf5
1050/1050 [==============================] - 771s 734ms/step - loss: 0.5824 - auc: 0.7555 -
Epoch 3/20
1050/1050 [==============================] - ETA: 0s - loss: 0.5730 - auc: 0.7655
Epoch 00003: loss improved from 0.58237 to 0.57300, saving model to weights.03-0.56419.hdf5
1050/1050 [==============================] - 777s 740ms/step - loss: 0.5730 - auc: 0.7655 -
Epoch 4/20
1050/1050 [==============================] - ETA: 0s - loss: 0.5664 - auc: 0.7721
Epoch 00004: loss improved from 0.57300 to 0.56641, saving model to weights.04-0.56383.hdf5
1050/1050 [==============================] - 772s 735ms/step - loss: 0.5664 - auc: 0.7721 -
Epoch 5/20
1050/1050 [==============================] - ETA: 0s - loss: 0.5614 - auc: 0.7771
Epoch 00005: loss improved from 0.56641 to 0.56143, saving model to weights.05-0.55510.hdf5
1050/1050 [==============================] - 770s 734ms/step - loss: 0.5614 - auc: 0.7771 -
```

Results





The model is trained on the data with loss getting less with each iteration, and providing a final AUC value of **0.80**. The model now is able to make predictions on any input images.

```
Epoch 20/20
1050/1050 [==============================] - ETA: 0s - loss: 0.5288 - auc: 0.8070
Epoch 00020: loss improved from 0.52990 to 0.52876, saving model to weights.20-0.53012.hdf5
1050/1050 [==============================] - 741s 705ms/step - loss: 0.5288 - auc: 0.8070 - val_loss: 0.5301 - val_auc: 0.8065
```

An ROC curve is a graph showing performance of a model. The plot takes into consideration True Positive and False Positive Rates. ROC curve plots the value at different classification thresholds. AUC (Area Under Curve) measures the two dimensional space below the ROC curve. AUC provides an aggregate measure of performance across the classification threshold range. Higher AUC is desired as the AUC value of a model whose 100% outputs are wrong is 0 and for a model that outputs 100% correct values is 1.

It is interesting to note that AUC gives equal weightage to the full range of sensitivity and specificity value while only a portion of the threshold might be used in real life, but for the purpose of this specific problem statement, with its 9 output variables which are sometimes subjective such as ‘good for children’, Ambience is Classy’ compared to for example, handwritten digit recognition problem, where the outputs can mostly be comprehended objectively from an image, we came to a conclusion that the threshold value will largely be dependent upon the business functionalities where the model will be deployed, and for that reason, AUC was chosen as an output metric.

On training data, we also measure F1 score, which is a harmonic mean of precision and recall. F1 score is used when a balance between precision and recall is needed, and gives us a better idea on how a model is performing using the test data. We achieved an F1 score of **0.739**.

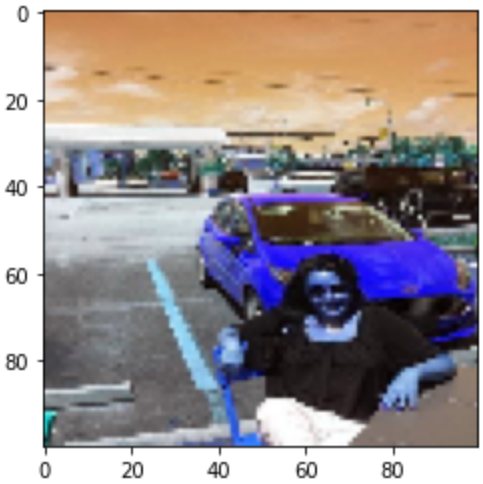
$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

In our case, a false negative value would be costly. If our model predicts that a restaurant does not have alcohol when in actuality it does, a customer looking through these reviews would not go to the restaurant which would result in loss of business for that business ID.

Following are some examples of the model prediction in action. The images below were given as inputs to the model and the output labels are shown below respective images.

1) Image of outdoor dining

After rescaling and normalizing

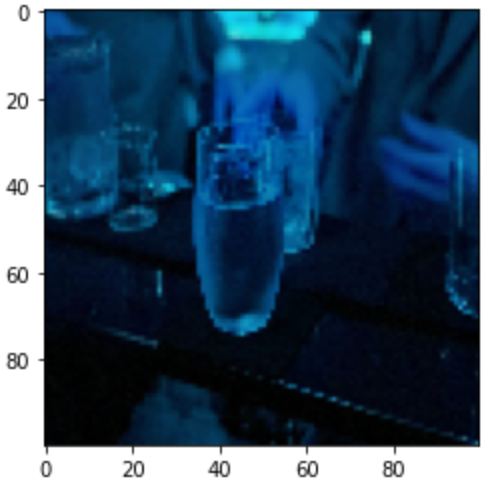


```
finalPrediction
```

```
['Outdoor seating', 'Has alchohol', 'Has Table Service', 'Good for kids']
```

2) An image containing alcohol and a table

After rescaling and normalizing

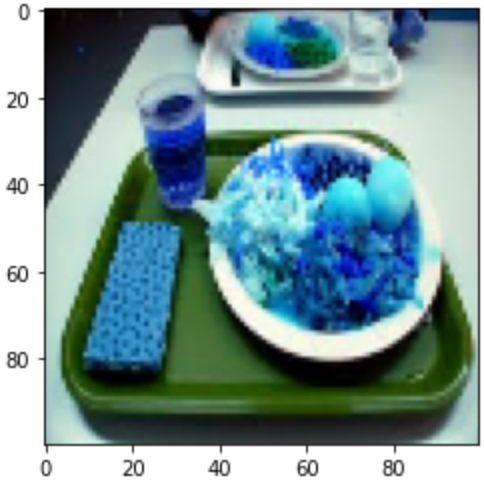


```
finalPrediction
```

```
['Has alchohol', 'Has Table Service']
```

3) An image of a dish

After rescaling and normalizing



```
finalPrediction
['Has alchohol', 'Has Table Service', 'Good for kids']
```

Compared to a Fully Connected Neural Network, CNN usually outperforms [6]. Due to the nature of the problem - with the dataset containing a lot of images, the baseline model training was not performed by the team, and an assumption is made that CNN has better performance.

In conclusion, we tackled the problem by employing convolutional neural network. Our model was able to achieve AUC score of 0.80, building a high performing model that can be put into production with further improvements to support Yelp's business functionalities and enhance their customer experience.

References

[1] Convolution Neural Networks vs Fully Connected Neural Networks | by Poulastya Mukherjee | Data Driven Investor

[2] Image Classification with PyTorch

[3] https://github.com/kaanertas/yelp-photo-classification

[4] Yelp Restaurant Photo Classification

[5] The Complete Beginner's Guide to Deep Learning: Convolutional Neural Networks

[6] Comparing the Performance of Fully-Connected, Simple CNN, and ResNet50 for Binary Image…

[7] Classification: ROC Curve and AUC | Machine Learning Crash Course

[8] Loss and Loss Functions for Training Deep Learning Neural Networks.

Appendix

Source Code

Google Drive Link:
https://drive.google.com/drive/folders/13dXQ6P-zAtoe6DtwHyhTRnJWjzNxeRt_?usp=sharing

README

File Structure

1) Files.rar (Please download from the link above and extract)
- Images_with_labels.h5: Image data with labels
- Train_labels.csv: training images to businesses to labels mapping
- Test_1.jpg, Test_2.jpg, Test_3.jpg - images for test illustration purposes
- Modelweights.hdf5 - model weights after training

2) Project.ipynb - project code notebook to be opened in Colab Notebook

Steps to run the code

1) Enable a runtime environment on Google Colab, and copy the content from **files.rar** in your Google Drive.
2) Upload and run **project.ipynb** on your Google Colab. Follow the instructions in the code to attach the Google Drive with the Colab.
3) Please note that Google Colab Pro was used for the purpose of this project as the memory and processing requirements were much higher.