Food: It's all about presentation?
A deep learning approach to predicting Yelp ratings using photos of food

Team Members: Nirmal Krishnan (nkrishn9), Jonathan Liu (jliu118), and Emily Brahma (ebrahma1)

May 5, 2018

Abstract

This project aims to use deep learning methods for predicting a restaurant's overall rating based on that restaurant's food images. We used the Yelp Open Dataset which consists of over 200,000 photos. We surveyed 4 different deep learning methods including a basic convolutional neural network (CNN), a more advanced CNN with batch normalization and dropout, a pretrained network using resnet18, and an autoencoder with a CNN. We found that the pretrained network using resnet18 performed the best and most consistently on validation, with a root mean squared error (RMSE) of 0.78 yelp "stars." This conclusively indicates that presentation indeed is involved in the enjoyment of food.

1 Problem Statement

1.1 Summary of Problem Statement

A recent study from Oxford University has showed empirically that "making something look good makes it effectively taste better too." The researchers found that "even with basic dishes, thoughtful presentation meant diners found the food more flavourful." In this study, we would like to test this theory rigorously, using photos of food to predict the rating of a restaurant on popular food-rating app Yelp.

1.2 Data Collection

For this project, we used the Yelp Open Dataset [1]. This consists of 174,000 restaurants, 5,200,000 review data, and 200,000 user-submitted photos with associated metadata.

For each photo collected in the dataset, we identified the corresponding restaurant where the photo was taken and the restaurant's "star rating" on Yelp. This "star rating" is an average of user ratings ranging from 1-5 of the quality of a restaurant.

After tying the photos to the restaurants and their star rating, we performed significant data filtering, including:

- Removing all restaurants in the dataset with fewer than 10 user ratings in order to ensure that the "star rating" was an unbiased representation of the quality of the food.
- Removing non-food pictures from the dataset
- Ensuring an even distribution of "star ratings" corresponding to photos in our dataset. This step specifically reduced the size of our dataset to 10,500 photos (with associated star rating labels).

To our knowledge, this is the first study on this dataset (or any dataset) that attempts to predict the quality of the food using photos of food. While we acknowledge potential shortcomings of our data collection– namely that there is variance associated with the presentation of food and quality of user-submitted photos– we, nevertheless, feel this is an interesting problem that could be explored more methodically in later studies.

2 Methods

2.1 Preprocessing

There was significant preprocessing performed for this project. As discussed in the data collection section we had to map between separate json files containing restaurant summaries, star ratings, and photos. In addition to this, we also needed to down-sample and scale each image to a consistent size of 224x224 (RGB), since the photos were taken on different types of cameras with different resolutions and sizes.

2.2 Model Input

As discussed in data collection, the input to our model was a single picture and the output is a predicted "star rating" for the restaurant. Since the labels range from 1-5 and we have an equal numbers of 1s, 1.5s, 2s, 2.5s..5s (10 total subdivisions), for our models results to be meaningful, our models must have an average root mean squared error (RMSE) of below 1.1. This is because if the model predicted the average label every time (star rating of 3), it would results in a RMSE of 1.1.

2.3 Training Hyperparameters

We used 70% of the data (7,350 samples) for training and 30% (3,150) for validation. For our models collectively, we used a batch size of 256, 20 epochs total, and the stochastic gradient descent optimizer because these parameters allowed our models to converge quickly (within our computational resource limit) and collectively have accurate results.

2.4 Basic Net

To begin, we used a basic convolutional neural network in order to establish baseline performance (accordingly named basic net). This basic net consists of two convolutional layers and two linear layers with relu activations after each layer except the final. The hyper-parameters for the convolutional net, including output channel size, kernel, stride, and padding were explored thoroughly and the following model setup generated the best results:

```
BasicNet(
  (conv1): Conv2d (3, 5, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d (5, 10, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=28090, out_features=100)
  (fc2): Linear(in_features=100, out_features=1)
)
```

2.5 Intermediate Net

After we established a baseline with BasicNet, we decided to improve upon this net by including batch normalization and dropout. In studies, batch normalization has been show to decrease the internal covariance shift between layers [2]. This helps to maintain consistency of inputs between layers and can reduce the drastic effects of input variance, which can be especially problematic in our dataset, since our photos are user-submitted. Dropout is a technique in which neurons are randomly zeroed out with a probability p [3]. In intermediate net, the convolutional layers have a dropout probability of 0.2 and 0.3. In practice, this has shown to increase the generalizability of the model to new data. There two techniques–batch normalization and dropout–improved performance significantly over basic net as seen in the results section. As was discussed in basic net, all of the hyper-parameters of the convolutional layers, batch normalization layers, and dropout probabilities were explored in detail.

The model setup for Intermediate Net (sans functional dropout layers) can be seen below:

```
IntermediateNet(
  (bn1): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True)
  (conv1): Conv2d (3, 10, kernel_size=(3, 3), stride=(1, 1))
  (bn2): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True)
  (conv2): Conv2d (10, 10, kernel_size=(6, 6), stride=(1, 1))
  (bn3): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True)
  (fc1): Linear(in_features=116640, out_features=1024)
  (fc2): Linear(in_features=1024, out_features=1)
)
```

2.6 Pretrained Net

Because training deeper neural networks is difficult and time consuming, we employed the help of a pretrained resnet18, a model known to be successful for image recognition [4]. Using the pretrained weights for transfer learning is a popular strategy and worked well in practice as seen in the results section. In order to apply resnet18 to our problem, we kept its parameters fixed and added a linear layer to the end of it, which had trainable weight parameters. This allows us to keep the encodings of resnet18 while adapting the output to our problem.

Another project with a similar task also found that using a residual network lead to a higher accuracy [5]. They were able to achieve their highest accuracy using Alexnet. However, in our experimentation, we found we had the best pretrained model results with resnet18. We do not include a model setup image for this model because it is extremely long, but it can be found in reference 5.

2.7 Autoencoder w/ CNN

The quality of the photos in our dataset may vary greatly in the color histogram as well as in the brightness and general amount of noise in the photos. This is due to the nature of how the photos are submitted to Yelp. We believed that using an autoencoder would help learn to ignore some of the noise by encoding the image into a smaller latent representation, thereby performing dimensionality reduction.
To train the autoencoder, we had two separate sequential modules–an encoder and a decoder. The encoder takes as input the original image and outputs our compressed representation. The compressed representation is then passed to the decoder, which then tries to reconstruct the original image. L2 loss is then minimized between the original image and the image produced by the decoder. We trained our autoencoder for 10 epochs, which admittedly may be too few but computational resources induced a bottleneck. After the 10 epochs, we trained a CNN similar to basic net on the compressed representation outputted by the encoder that predicted yelp star ratings for the corresponding photo. The model for the autoencoder can be seen below:

```
AutoEncoder(
  (encoder): Sequential(
    (0): Conv2d (3, 16, kernel_size=(3, 3), stride=(3, 3), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d (16, 8, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): ReLU(inplace)
  )
  (decoder): Sequential(
    (0): ConvTranspose2d (8, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): ReLU(inplace)
    (2): ConvTranspose2d (16, 3, kernel_size=(3, 3), stride=(3, 3), padding=(1, 1), output_padding=(1, 1))
    (3): Tanh()
  )
)
```

The CNN trained using the outputs from the encoder can be found below:

```
AutoBasicNet(
  (encoder): Sequential(
    (0): Conv2d (3, 16, kernel_size=(3, 3), stride=(3, 3), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d (16, 8, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): ReLU(inplace)
  )
  (conv1): Conv2d (8, 30, kernel_size=(2, 2), stride=(1, 1))
  (conv2): Conv2d (30, 10, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=640, out_features=100)
  (fc2): Linear(in_features=100, out_features=1)
)
```

3 Results

3.1 Experimental Setup

The implementation of this project involved dataset extraction, preprocessing, and then regression on the processed data. The code for this project was written in Python 3.6, using Pandas for data handling/preprocessing and Pytorch v0.4 for building the nets.

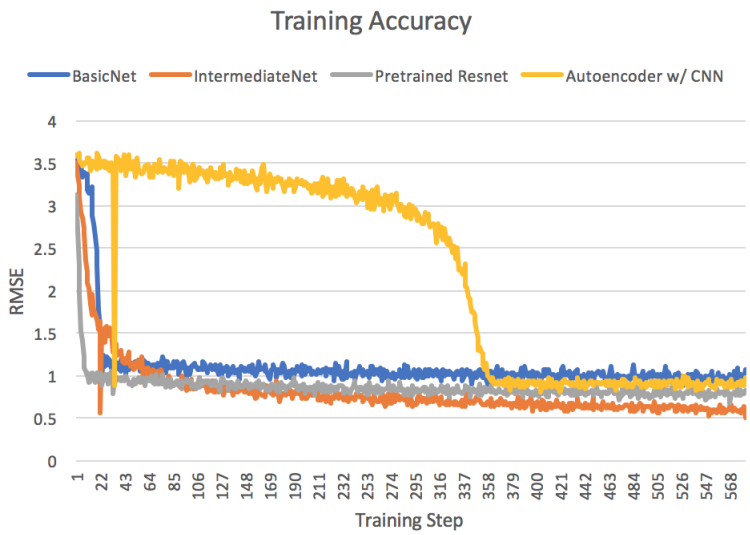
We used a Google Cloud Compute Instance with 2 vCPUs and 7.5 GB memory, with one NVIDIA Tesla K80 GPU. In hindsight, this was not enough and we should have opted for a CPU with more memory and multiple GPUs in order to speed up training.

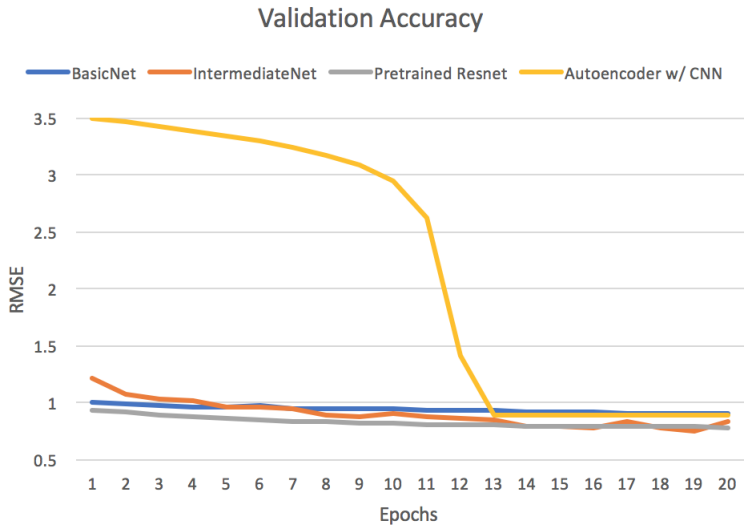
3.2 Evaluation Methodology

For each of the nets, we used an L1 loss function for our optimization. We have a very constrained regression space, having only 10 categories: 1 star - 5 stars (0.5 intervals). Because of this, using an L2 loss function would punish the more extreme predictions (predictions closer to 1 and 5), very harshly, thus leading to mostly average (3 star) predictions. So we decided to use an L1 loss function in order to ensure our model "takes risks"- predicts extremes when necessary. For training the autoencoder, we used an L2 loss function because this seems to the be industry standard according to the literature.

As discussed in the section 2.3, we used 70% of the data (7,350 samples) for training and 30% (3,150) for validation, with training batch loss tracked at the end of each training step and validation loss tracked at the end of each epoch. Since this is regression data, our accuracy and loss terms are the same (L1 loss is equivalent to RMSE).

Performance





Optimal Validation RMSE Table

Model	Epoch	RMSE
Basic Net	20	0.90
Intermediate Net	14	0.78
Pretrained Net	20	0.78
Autoencoder w/ CNN	20	0.89

4 Discussion

4.1 Analysis of Results

From our results, we can see that intermediate net has the lowest loss in training, and converges to approximately 0.58. In our validation results, we see that the intermediate net loss is 0.78 by epoch 14, but it oscillates (and even increases) once it gets to the higher epochs, indicating overfitting. In validation, the pretrained resnet model consistently performs well, with a monotonically decreasing loss function.

The model using the Autoencoder w/ CNN behaved in a peculiar way. It took a longer time to converge compared to the other nets. This may have been a result of the convolutional autoencoder we used which increased the number of channels to 16. The landascape of the optimization space with 16 channels is likely more difficult, which may resulted in longer convergence times. This model could in practice be the best since it may have needed more than 20 epochs to converge. However, due to limited computational resources, we can only report our results on 20 epochs.

Overall, we believe that the pretrained resnet performs the best. This is because while the intermediate net had a lower training loss, the oscillations in its validation accuracy leads us to believe that it is overfitting. If early stopping is performed on the intermediate net (around epoch 14), we believe its performance in practice would be comparable to the pretrained resnet model. It's exciting to see that all of our models significantly outperformed the RMSE of 1.1 discussed in section 2.2. Based on our results, there is basis in the idea that presentation is important factor in taste.

Known Limitations and Future Extensions

As was discussed thoroughly in this paper, computational resources were a significant problem. In future papers, more CPU memory and GPUs should be allocated towards training, such that results can be garnered for more epochs. Additionally, we did not have time to perform this step, but photos of restaurants are included in the Yelp dataset and it would be interested to include "ambiance" as a factor in our predictions.

Takeaways from the Project

1. Training autoencoders takes incredibly long! In the future, we should make sure to allocate enough resources towards them (potentially more than 1 GPU).

2. It is incredibly easy to incorporate pretrained models using pytorch. Downloading and incorporating both alexnet and resnet18 into our model setup was intuitive and successful.

3. Batch normalization and dropout work very well in practice. The jumps in performance between basic net and intermediate net just with these two additions were surprising.

Advice for Next Year's Students/Instructors

For students, some advice would be to start the assignments early since training often times takes a while. Another remedy is to be sure to have access to GPUs (through Google Cloud or AWS). Additionally, make sure you have a good basis in probability and statistics before taking the course as a theoretical foundation in mathematics is critical to thoroughly understanding abstractions in deep learning.

For instructors, some advice would be to enforce checkpoint deadlines more and have midpoint checkpoints for homework assignments so that students would be forced to start the assignment earlier and potential problems in the assignment would also be discovered earlier. Additionally, we recommend giving more smaller homework assignments that allow students to explore deep learning specific genres like recurrent neural networks, generative adversarial networks, and variational autoencoders.

References

1. Yelp Open Dataset- An all-purpose dataset for learning

2. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

3. Dropout: A Simple Way to Prevent Neural Networks from Overfitting

4. Deep Residual Learning for Image Recognition

5. Computer Vision for Food Cost Classification