Tanaya Pole

Introduction:

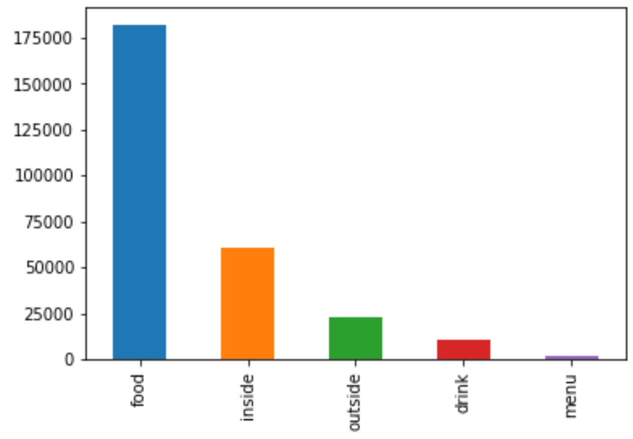
For our project, we are solving the Yelp Dataset Challenge. Yelp provided us with data without instructions. The goal of their challenge is to use their data in innovative ways that might be able to help them with their research. Their data consists of images and a json file with information on the images such as photo id and labels.

Yelp is a search engine service powered by a crowd sourced review forum. It has reviews and recommendations about nightlife, shopping, restaurants. Users who upload images to Yelp, are given an option of tagging the images which most users do not complete. This leaves many images without tags. Yelp wants to come up with a method that can automatically tag the uploaded images.

We decided to take up this challenge and to use convolutional neural networks to classify the images we will be experimenting with different parameters using PyTorch, TensorFlow, and Keras frameworks.

We were provided with a set of photos (200,000+), a json file with photo id and their labels (food, drink, inside, outside, menu), and a csv file with photo id and 99 columns of data for each picture. We do not have any information on what the data in each of the columns represents. Since we do not know what they are, we decided not to use them. Instead, we created a new csv file with the photo id and merged it with the labels from the json file. We will be using the new processed csv file (with photo ids and their respective labels) and the images.

Below is a plot of the labels in our data set with their respective frequency. 'Food' has the highest number of photos in comparison to the others while 'menu' has the least number of photos.

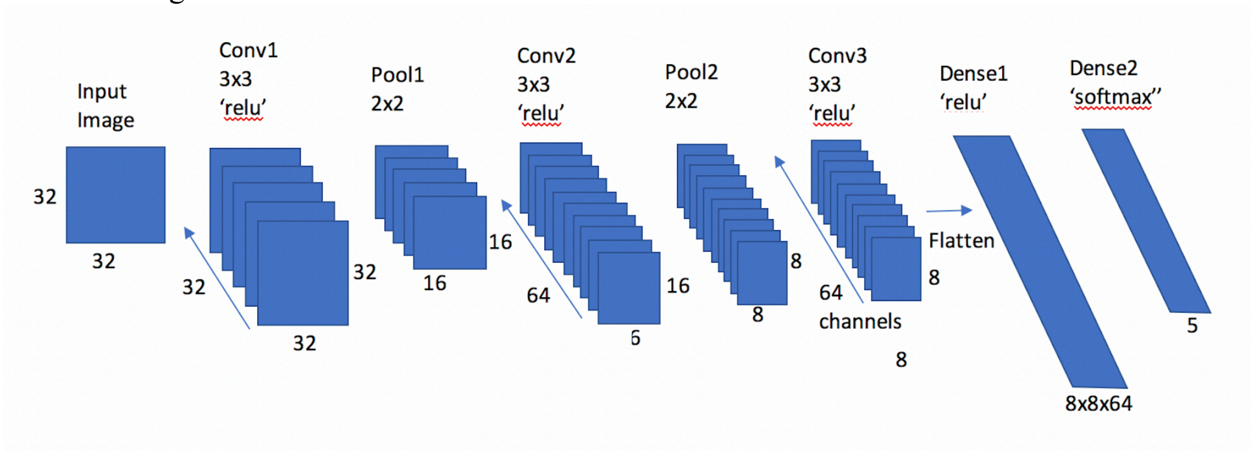


We all worked together on preprocessing and each of us decided to run a convolutional neural network using a different framework. Selvyn used PyTorch, Ritika used Tensorflow, and I used Keras.

Description of individual work:

Keras is an open source neural network library that runs on top of Tensorflow and Theano. It is user friendly, allows for fast results, and is commonly used for convolutional neural networks.

The following shows the architecture of the convolutional neural network I used in Keras.



In Keras, it is not needed to split the data beforehand, but because we had already split our data using Scikit Learn, I decided to input our already normalized and split data into the model.

In our convolutional layers, I used ReLu as our activation function because it is one of the most common activation functions, and it also speeds up training. Any negative numbers are set to 0. The dense layers are the fully connected matrices. In the first dense layer, I used ReLu again because it is computationally efficient. i used softmax in the second because softmax outputs a probability range for each label, which is how we want our output to look like. I needed to be able to see the probability of each label for each image. I used cross entropy to calculate the loss because I am using a classification model.

My portion of the project:

We all worked on the preprocessing together. First, we had to upload our data to the google bucket. We split the images up because it was taking about 30 hours to upload. My portion had to be uploaded overnight.

I chose to work on the Keras framework. All of the code for Keras was written by me. I had experimented with all the different parameters and also used Tensorboard to visualize my network. I wrote the section of the report on Keras. We all equally wrote the rest of the report that didn't include frameworks.

Results:

I decided to use 3 Convolutional layers with two poolings because it gave me the highest accuracy. The following is a table that shows the other architectures I considered. For each one, the batch size is 100, epochs is 5, optimizer is Rmsprop.

Layers	Accuracy	Loss

1 Conv, 1 Max Pool	0.8663	0.4296
2 Conv, 2 Max Pool	0.9033	0.2847
3 Conv, 2 Max Pooling	0.9045	0.2828
3 Conv, 3 Max Pooling	0.8948	0.3000

Here, you can see that the highest accuracy with the lowest loss is for 2 convolutional layers with 2 max poolings. Now that I have decided on the architecture, I decided to experiment with optimizers. The optimizers I chose to investigate are Rmsprop, Adam, SGD, and Adagrad.

Optimizer	Accuracy	Loss	Time (s)
Rmsprop	0.8923	0.3177	71.3795
Adam	0.8929	0.3057	75.2057
SGD	0.7605	0.6912	71.3718
Adagrad	0.8360	0.4748	69.3098

Adam and Rmsprop have very similar accuracies and losses. Because Adam took the longest time to run, I decided to go with Rmsprop.

I also experimented with several different learning rates to decide on an optimal one.

Learning rate	Accuracy	Loss	Time(s)
0.001	0.8923	0.3177	71.3794
0.0001	0.8542	0.4189	71.1081
0.01	0.6548	5.5634	72.3446
0.005	0.8073	0.5508	74.1131

All of the learning rates took similar amounts of computational time. The accuracy for 0.001 was the highest so I decided to choose that one.

Batch Size	Accuracy	Loss	Time (s)
100	0.8923	0.3177	71.3794

500	0.8541	0.4046	61.5452
1000	0.8247	0.4973	64.0554
50	0.9040	0.2931	97.5911

The accuracy for the batch size of 50 was the highest, however it was significantly slower to run. The accuracy of a batch size of 100 was slightly lower, but it ran about 30 seconds faster, so I decided to keep that the batch size.

I decided to select 5 epochs because it took the least amount of time and the accuracies for a higher number of epochs did not change significantly.

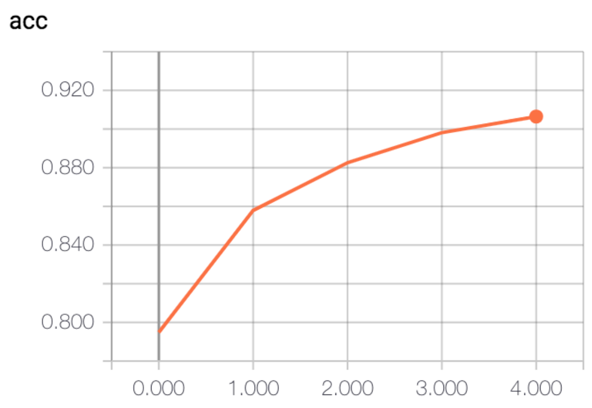
Epochs	Accuracy	Loss	Time(s)
5	0.8923	0.3177	71.3794
100	0.8904	1.1933	752.6329
500	0.8925	1.6946	3521.7869

Last, to check for overfitting, I decided to add a dropout layer.

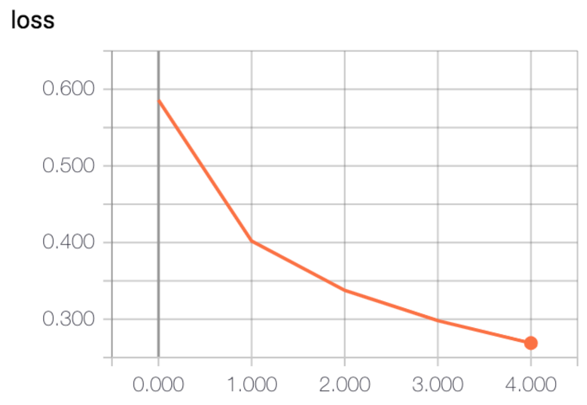
Dropout	Accuracy	Loss	Time(s)
0.0	0.8923	0.3177	71.3794
0.2	0.8839	0.3387	74.8656
0.4	0.8937	0.31295	72.7018

Adding a dropout layer and increasing the percentage of dropout didn’t affect the accuracy much. This showed that my model was not overfitting.

Using Tensorboard, I graphed my accuracy and losses.



Here, you can see that as the number of epochs increases, the accuracy also increases.



As the number of epochs increases, the loss decreases. This is because loss is minimized during training.

Conclusion:

Overall for CNN using keras, the test accuracy was 0.8923 with a loss of 0.3177. This network took 71.4 seconds. I compared the model with my other team mates' and noticed that mine was the most computationally efficient with one of the highest accuracies. Keras was simple to use and made it convenient to visualize using TensorBoard.

One improvement that could be made is to have a more balanced dataset by having more images from labels other than food or by sampling. Most of the images in the dataset provided were food images, and I feel this may have affected the training of the model. A more balanced data set will have a better probability of classifying all of the labels, not just food, more accurately.

Percentage of code from resources:

- About 90% of my code is from the internet (https://keras.io/callbacks/) and from Deep Learning with Python by Francois Chollet.

References:

- Deep Learning with Python by Francois Chollet
- https://keras.io/callbacks/