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. al Final Report

Description of Individual Work

I worked on the Pytorch and portion for the neural network model while Ritika worked using Tensorflow and Tanaya worked using Keras. As I work as a Software Engineer currently, I had the most experience coding and so I also wrote the code that preprocessed the data. After this code, I also wrote the code that deployed all of the images and files to our google cloud storage. I helped the team with any other issues relating to environment setup using Tensorflow and Keras as well but my main portion was working with Pytorch.

Description of Work

Data Preprocessing:

The images that we downloaded from the Yelp challenge all have different dimensions. In order to use a convolutional neural network, we decided that we needed to resize the images to 32x32. This process was very time consuming as we had to leave the program running overnight to get it all resized.

The yelp dataset came with roughly 280,000 images. The folder size storing all of these images was over 3GBs. We decided to upload these images to the google cloud storage bucket. Google has a utility that can be installed locally and allows you to upload/update files to your bucket. We used this utility to upload all of the images. This also took a long time, and we had to use the terminal to upload it into the bucket overnight. After they were uploaded, I needed to set the permissions to allow public access to view them. This is an important step because it

allowed us to write our models without needing to have the images on our directory. All the images would be retrieved from the cloud.

The first pre-processing step I had to complete was to map the json file with the csv image file. This was because the json file had the image labels which were what we were going to try to predict while the csv had the image ids which would be needed to retrieve them from the bucket.

Code:

```
with open('yelp_academic_dataset_photo.json') as f:
    for line in f:
        data.append(json.loads(line))

yelp_frame = pd.read_csv('yelp_academic_dataset_photo_features.csv')


yelp_frame.insert(loc=0, column='type', value=0)

yelp_frame.iloc[0,0] = "TEST"


yelp_size = len(yelp_frame)

values = set()

index = 0

for i in data:
    values.add(i['label'])
    photo_id = i['photo_id']
    label = i['label']
    #print (label)
    #print (i['photo_id'])
    for j in range(yelp_size):
        yelp_frame_ind = yelp_frame.iloc[j,1]
        if photo_id == yelp_frame_ind:
            yelp_frame.iloc[j,0] = label
            #break
            #index += 1
            break

yelp_frame.to_csv("yelp_dataset_preprocessed.csv")
```

Next, using urllib we were able to retrieve each individual image using the picture_id and building a response from the url. We saved the images and labels into separate .npy files and also uploaded them to the bucket and made them public. From here on, we were able to directly reference the .npy files in each of our python files in order to speed up the run time significantly

Neural Network:

PyTorch is an open-source machine learning library for Python, based on Torch, used for applications such as natural language processing. Pytorch is a good framework for running a CNN but the performance is slower. After getting the model to read in the npy data, I needed to split the data in 70/30 train/test. Then the model was ready to train. For computing the loss, we used the CrossEntropyLoss() function and used the SGD optimizer function. Also, for the purposes of helping the performance, we ran the model using the GPU. We wanted to how our accuracy would change if we removed the Convolution layers from these runs.

Results

We ran three different models switching the layers, batch size and number of epochs. For the first run, we used the following parameters with one layer using the Softmax activation function:

```
input_size = 3072
hidden_size = 500
num_classes = 5
num_epochs = 100
batch_size = 100
learning_rate = .0001
```

Results for first run:
Epoch [100/100], Step [700/700], Loss: 0.7141
Accuracy of the network on the 100000 test images: 76 %
Accuracy of drink : 42 %
Accuracy of food : 95 %
Accuracy of inside : 63 %
Accuracy of outside : 32 %
Accuracy of menu : 15 %

To start, we felt we could run using a learning rate of .0001 and start with a larger batch size. As the results show, the we had a 76% accuracy using this method and a final loss of .7141.

For the second run, we used the following parameters:

```
input_size = 3072
hidden_size = 500
num_classes = 5
num_epochs = 100
batch_size = 10
learning_rate = .0001
```

Results for second run:
Epoch [100/100], Step [7000/7000], Loss: 0.5483
Accuracy of the network on the 100000 test images: 81 %
Actual: food food inside food inside food food food food food
Predicted: food inside inside food inside food food food food food
Accuracy of drink : 48 %
Accuracy of food : 91 %
Accuracy of inside : 77 %
Accuracy of outside : 32 %
Accuracy of menu : 50 %

By reducing the batch size significantly, the accuracy of the model increased significantly. The final loss was also reduced significantly, down to .5483.

For the final run, we added a couple more layers to the network architecture, still using the softmax function as the activation:

```
input_size = 3072
hidden_size_1 = 500
hidden_size_2 = 100
num_classes = 5
num_epochs = 100
batch_size = 20
learning_rate = .0001
```

Epoch [100/100], Step [3500/3500], Loss: 1.1605
Accuracy of the network on the 100000 test images: 65 %
Actual: food inside food food food food inside inside inside food food food food food food menu food food inside food
Predicted: food food food food food food food food food food food food food food food food food food food food food
Accuracy of drink : 39 %
Accuracy of food : 100 %
Accuracy of inside : 80 %

Accuracy of outside : 46 %
Accuracy of menu : 23 %
RUNNING TIME: 529.432516098
What we found was that the best model was when running on one layer and using the a small batch size. The performance suffered significantly when adding more layers using pytorch. As our later results would see for the other frameworks, this would not be the case when adding more layers to them.

Summary and Conclusion

Our group decided to use 3 different frameworks for this project: Pytorch, Keras, and Tensorflow. We expected that all 3 frameworks would produce similar results since we were planning on using convolutional neural networks. So for 1 framework, Pytorch, we ran our model without a Convolution Layer and strictly used a Linear layer. This resulted in all 3 runs being lower in accuracy than when using Keras or Tensorflow for which we used convolutional layers for. The strongest accuracy was 81% with Pytorch for which our loss was .54. This was decent performance but as noted, the other frameworks using convolutional layers performed better. So what I learned is that for deep networks and huge image datasets like this yelp one, that a convolutional neural network works best. Running these models without a convolution layer or multiple layers will not return very strong results.

Code Copied

(This includes all preprocessing code)

$$\frac{254 - 106}{254 + 22} \times 100 = 53.62\%$$

References

https://pytorch.org/docs/0.3.1/nn.html#linear

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

https://blog.algorithmia.com/convolutional-neural-nets-in-pytorch/