Yelp Image-to-Rating Final Report

G. Lau (1005865921), H. Sahak (1005903710), V. Tran (1004988775)

ECE324, Prof. Guerzhoy

April 13, 2022

Table of Contents

NOTE: HERE IS THE LINK TO THE ENTIRE YELP PROJECT FOLDER https://drive.google.com/drive/folders/1QnKmXui7nYeHTvnnS6sP_4_65uxjcTyj?usp=sharing

I. Abstract

We explore various approaches to predict the Yelp rating of a restaurant using photos of the food from customer reviews. On a dataset of over 20,000 image-rating pairs, we first train CNNs from scratch and find best benchmark results (MSE = 0.299) using a network with two convolution layers and a fully connected layer. Next, we test three popular pre-trained networks: VGG16, ResNet50 and MobileNetv2, as well as an early-fusion ensemble approach combining the three. The pretrained/ensemble networks fail to improve upon the test-set MSE from before. MobileNetv2 is the best of the pretrained/ensemble networks with MSE = 0.308. Last, we propose a novel architecture VGH-Net. VGH-Net training occurs in two stages. First, using a CNN to predict BERT embeddings of a restaurant's text reviews from image data. Then, using a feed-forward neural network to predict the review-specific restaurant rating from the review BERT embedding. VGH-Net achieves a MSE = 0.308.

II. Introduction

Using online business directories such as Yelp, Google Maps, and Trip Advisor has become an incredibly popular way to compare restaurants and decide where to visit. As of December 31, 2021, Yelp reported 83 million unique visitors and 244 million cumulative reviews on its platform [1]. Crowdsourcing ratings on a scale from 0 to 5 stars is seen as a democratic approach to ranking restaurants. However, opportunity exists to provide a more robust rating by factoring in additional sources of information such as text reviews and review photographs. Generating ratings from text reviews is an area of machine learning that has already been well researched. A paper by Liu applied various natural language processing techniques to predict restaurant ratings from Yelp reviews [2]. The goal of this project is to predict a restaurant's Yelp rating using photos from customer reviews. We want to know how accurately an image-based model can predict restaurant ratings and how an image-based model compares to a text-based model. For platforms such as Yelp, an image model can be integrated with their existing rating system. Image data is more difficult to fake than ratings or reviews and thus, its inclusion in a rating algorithm should improve the integrity of the scoring. Many social media platforms have existing image data but not explicit rating systems. They would be able to leverage an image model to create travel guide or business recommendation features. This report documents the multitude of approaches we attempted to try to predict ratings from image data including using CNNs created from scratch, using pre-trained CNNs, and using our group's novel architecture: VGH-Net.

III. Background

For our prior work, we value papers that use a similar dataset to answer similar questions. In the subsections below, we provide detailed outlines of papers and connections to what we have done for this project.

III.1 Predicting Restaurants' Rating And Popularity Based On Yelp Dataset [3]

The goal of this paper was to analyze several ML models to determine which performs best at predicting restaurants' rating and popularity based on data from the Yelp Dataset. The paper is a good pick because it is written by students as part of a project, so it is a realistic example of the kind of analysis we could do. More importantly, they use the Yelp Database and investigate many ML models. Unlike the ensemble approach we discuss next, the output of all rating-based models is a number from 1-5, discretized to increments of 0.5. This is exactly what we want for our model's output, so it is worthwhile to investigate the paper's approach.

The data of the paper comes from the Yelp Dataset Challenge. They focus on restaurants in Toronto- 5000 is used for training and 1750 for testing. Another 5700 restaurants near Toronto are included in testing to assess generalization. Unlike our model which inputs images, they use restaurant features like available services, price level, locations, opening hours, etc. In total, 42 features are used. While input is different from our model, it is still non-textual information, so gives us an idea of how successful region-based instead of user-based analysis is (note: we assume images of food would be user-independent with respect to model outputs; thus, the pipeline is not user-based).

Before assessing performance for various ML models, the authors conduct a forward feature selection using MATLAB "sequentialfs" function. This is done to avoid overfitting and only include a number of features that keeps loss low enough.

The first model is linear regression, with selected features from the feature selection above. To calculate prediction error, they use a simple discretization rule. A continuous variable is predicted by the regression, and then rounded to the nearest 0.5. This way, results are comparable with other methods we now describe. The second model used multinomial logistic regression. This tries to predict the probability of dependent variable falling into each category and selects the class with highest probability. The third model is Naive Bayes model. Here, the class with the highest conditional probability of output given restaurant features is selected. The authors choose a multinomial prior for the feature vector, since all variables are discrete or are discretized. The prior and conditional probabilities are both attainable from the dataset. The final model is a 3-layer neural network with hidden layer size 100, and sigmoid activation. MATLAB is used for training all models.

The result of experiments is that linear and logistic regression perform slightly better than Naive Bayes and Neural Network. Their best model is logistic regression and has accuracy of 32%. According to the authors, they don't have enough input features for a decent prediction. They claim they can benefit from more features that capture factors affecting ratings and also more training samples. Furthermore, model performs better for restaurants in Toronto than near Toronto. However, they note that the forward feature selection is successful in solving overfitting. The problem was thus identified as data being very local and only representative of Toronto.

This paper highlights the importance of using representative data. According to [4], the most important factor for the success of a restaurant is the food. By using food images on Yelp, we will have both data that is much more in quantity and more representative of factors affecting rating. If we can use a CNN or other architecture to extract features from food images or restaurants and use this instead of metadata (as in this paper), our project can be seen as an improvement to this paper; especially if we use their techniques and come up with better results.

We observe a clear connection to our problem. Since we are using images, our approach is largely dependent on the Convolutional Neural Network (CNN) to go from image to rating. The main advantage of using CNNs is that they automatically detect the important features without any human supervision [5].

Specifically, learning important features from the input image is part of the CNN architecture, so we do not need to collect metadata or run the "sequentialfs" function in MATLAB.

Furthermore, we have attempted to address limitations of this paper in terms of using a representative dataset and better features. Similar to this paper, we try several different architectures, and compare them using one single metric- mean squared error. We also use a larger dataset, since the size of their dataset may have contributed to poor results of a maximum 32% accuracy on the test set.

III.2 A Deep Learning Ensemble approach to the Yelp Restaurant Classification Challenge [6]

This paper attempts to train a deep learning model to label restaurants with multiple categories based on user-submitted photos of the restaurant. Specifically, they use the Yelp Restaurant Classification Challenge (YRCC) data to train several CNNs and select the one with the highest F1-score in the YRCC. The final model is a multi-model ensemble of fine-tuned VGGNet architectures trained on 3 separate databases: ImageNet, Places MIT, and Food101, which is expected to capture the diversity of the dataset and combine the strengths of the different models at predicting labels that correspond to its training data (e.g., Places MIT for restaurant's interior and exterior, Food101 for food, etc). The outputs of the 3 models can be fused by max-pooling or averaging the probabilities- assigning label = 1 or label = 0 for each class' neuron depending on probability > 0.5 or not.

Since we are also using Yelp data, we are interested in how they modify their learning architecture to specialize on the dataset. Their training data consisted of 2000 restaurants and ~230,000 images. Each image belonged to a restaurant, and each restaurant was labeled with one of 9 categories: {good for lunch, good for dinner, takes reservations, outdoor seating, expensive, alcohol, table service, ambiance is classy, good for kids}. A particular challenge, which also exists in our project, is that instead of having labels for every image, there is just one set of labels for each restaurant. There is no information about individual images, except for matching them with associated restaurants.

The paper's first approach was to assign the labels of each restaurant to all its images. Transforming the problem to a binary relevance problem, the authors found that using CaffeNet to extract features from the images and then Scikit-Learn to train a SVM for each label gave "quite good" results. However, they abandoned the restaurant → image labeling method because it assumes every image reflects every label, which is obviously not true.

The improved baseline used a method that assigned a feature vector to each restaurant. Image feature extraction was first done by extracting the first fully connected layer of CaffeNet pre-trained on ImageNet, which is the most general of all fully connected layers but still benefited from previous convolution layers. Restaurant features were constructed by averaging feature vectors for all its images. Improvements to the baseline were made by experimenting with different layers, architecture, pooling, and classification techniques. The authors investigated whether it is better to use more specific or more general layers (fc6 vs fc7 vs etc). They also tried VGGNet as it is substantially deeper than CafeeNet but not as computationally expensive or complex as ResNet or GoogLeNet, respectively. Max pooling was investigated instead of averaging for restaurant vectors. Finally, a multi-layer perceptron was used instead of 9 different SVMs, to include relations between labels. This was more relevant to us because a SVM is for classification and our project is a regression problem, so we would use a fully connected layer if we wanted the end result to be a rating rather than probabilities of different classes.

After optimizing the above, the authors focused on different datasets to pre-train or fine-tune a CNN on. The idea was to extract more expressive features for certain types of images; e.g by using a model used for food when extracting features for a food image rather than relying on ImageNet which is trained on a very general image database. The final result was 3 fine-tuned VGGNet models trained on ImageNet, a places database (MIT's Places) for scene recognition, and a food database (Food-101), to be more representative of Yelp images. Six fusion methods were investigated and the best was found to be late fusion, specifically bagging, which uses a voting procedure such that at least 2 models have to output probability > 0.5 for a class for the ensemble to predict that class. The model achieves an F1 score of 0.82, which is 12% higher than baseline (0.73), and obtained 25th of 355 participants in the competition.

Based on the results presented in this paper, it is worth exploring pre-trained and fine-tuned models. VGGNet seems to be a good choice for a base model since it is very expressive but not too computationally expensive. A paper by Yosinki et al. argues that the first layer learns very general features that primarily represent color blobs and filters, whereas the final layer learns very task-specific features that depend on the dataset and task.

More importantly, we would like to assess how good an ensemble approach is. The paper states that a reliable approach to improving the performance of CNNs is to train multiple independent models and combine their results. So, we would like to compare an ensemble-based approach with the best-performing CNN model.

This serves as further inspiration for our work. In this project, we have attempted to perfect CNN architectures to minimize mean squared error on the test set, individually fine-tuning three common networks: VGG16, ResNet50, and MobileNetv2. They will ultimately be combined into one ensemble network. The results are discussed in Section V and are summarized in Table 1. To conclude this section, this paper essentially inspires the use of an ensemble of CNNs. The ensemble learning approach is based on the assumption that each model in the ensemble produces errors of different types on different classes. We hope that by combining the CNNs, a more general model should be reached that performs better on the overall task (i.e., for all rating classes).

III.3 A Collaborative Neural Model for Rating Prediction by Leveraging User Reviews and Product Images [7]

This paper presents a novel rating prediction model (NRPTV) by jointly modeling text features from user reviews and visual features obtained from product images. It is useful to compare the performance of our image-based model with this to see whether images alone are a relatively good predictor of rating compared to textual and visual information, which is the main question our research seeks to address.

This paper also lays the foundation for the development of an architecture that incorporates both text and visual information. While our proposed model will not end up combining user reviews and images into one representation, we take a more innovative approach by considering text embeddings that represent emotion and summarize the text that can be predicted from images. In this way, we combine both visual and textual information, while also leveraging the benefits of the embedding-to-rating pipeline.

IV. Data Collection

We used the 40,000 images tagged as “food” from the Yelp Images Dataset which contains 200,000 images. The Yelp dataset also includes a file that provides the Business ID of the restaurant where each photo was taken. We wrote a Python script to make calls to the “Business Details” endpoint of the Yelp Fusion API to get the restaurant rating for each Business ID in our dataset. We ran this script over several days due to API call limits to collect 39,808 image-restaurant rating examples. A histogram of the restaurant ratings is shown in figure 1. We wrote a similar script to call the “Reviews” endpoint of the API to get up to 3 reviews (and the review-specific rating) for each Business ID. A total of 89,763 reviews were collected. The image and restaurant rating data was used in the preliminary architectures while the image and review data was used in VGH-Net. The full dataset can be found in the file *YelpProject/foodEntriesWithReviews.json*

The reviews returned by the Yelp API are truncated and have an average length of 28 words. Attempts were made to web-scrape full-length reviews using Beautiful Soup. However, bypassing website blocks proved to be challenging and we were only able to scrape 11,752 full-length reviews from 1,207 restaurants. The full reviews can be found in the file *YelpProject/LongReviewScrapingJSONs/foodEntriesWithLongReviewsA1.json*.

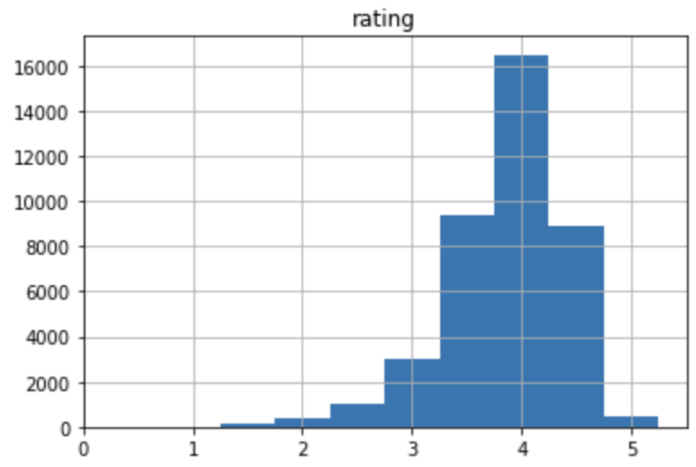


Figure 1: Histogram of Restaurant Ratings from the 40k image Yelp Dataset

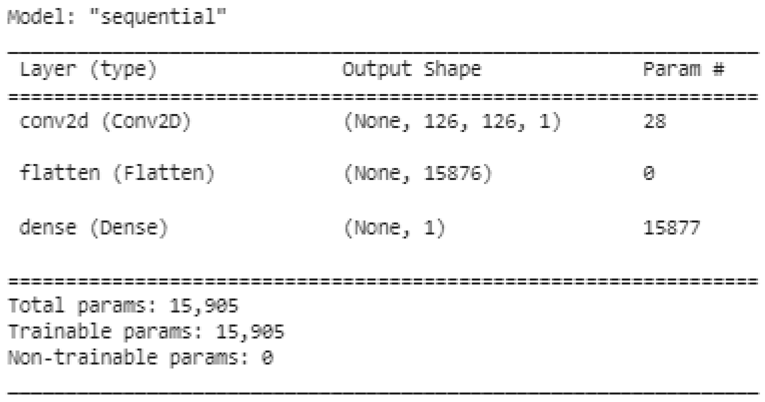
V. Preliminary Architectures

V.1 CNNs for Image to Rating Predictions

Our first approach to this problem was to use a basic CNN, which we know is ubiquitous in image classification tasks. The primary motivation behind using CNNs for this kind of task is the weight sharing feature, which reduces the number of trainable network parameters, in turn helping the network to enhance generalization and avoid overfitting [8]. Furthermore, CNNs will ensure translation invariance through the sliding filters and generalization through max-pooling layers [9]. We expect that layers at the beginning are responsible for detecting general features such as edges and colors and the layers at the end are responsible for detecting task-specific features such as foods that directly help with the regression task.

V.2 Basic CNN Performance

Before progressing to more complex architectures motivated by our prior work section (e.g., ensemble method), we first trained a very basic CNN to use as a baseline when assessing the performance of future networks. This CNN had only one convolution layer, with filter size (3,3) and ReLU activation function. This layer was then flattened and connected to a dense layer with one output that attempts to predict the final rating. The reasoning behind the architecture was to have something very simple, similar to logistic regression architecture, except keeping a CNN architecture and an activation function. See the figure below for the model summary.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 1)       28

 flatten (Flatten)           (None, 15876)             0

 dense (Dense)               (None, 1)                 15877

=================================================================
Total params: 15,905
Trainable params: 15,905
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Simple CNN Model Summary

Performance obtained on our test set was assessed using the mean squared error. The simple CNN achieved a mean squared error of 0.3249, which performs worse than a network that outputs 4 all the time (mean squared error = 0.3175). This motivates us to build more complex architectures. We start by sticking to a CNN but making the architecture more complex by adding more convolutional layers and increasing the dimensionalities of layer channels. Pooling layers are also introduced. These experiments are discussed below.

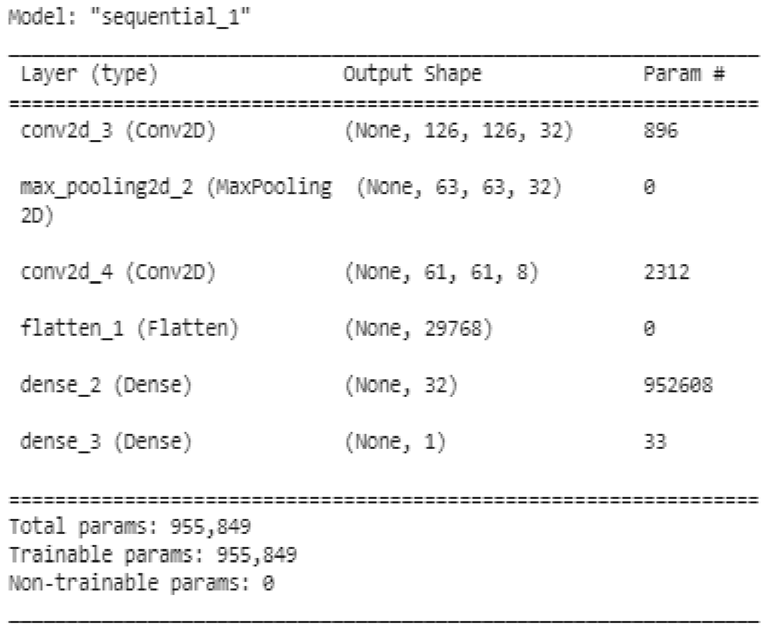
V.3 Improving Simple CNNs Through Hyperparameter Tuning

As is normally done with ML models, we perform hyperparameter tuning and select a model that performs best on the validation set. We then present our final model by stating the mean squared error on the test set. We choose mean squared error because it is an intuitive metric for measuring the distance from a true value and is most commonly used in regression tasks [10].

In our case, we performed 6 iterations of hyperparameter tuning. We essentially use the random search technique first taught to us through ECE421; we change the model by selecting different hyperparameters and changing their values individually. The final model gives the best result on the validation set and has a mean squared error on the test set of 0.2991.

Model 1:

Instead of having just one convolution layer, we change the architecture to be significantly more complex. We have a 2D convolution layer with 32 channels, a 5 by 5 filter, and ReLU as the activation function, followed by a max-pooling layer with a 2 by 2 window, another convolution layer with 64 channels, a 3 by 3 filter, and ReLu as the activation function again. Another max-pooling layer (identical to the previous max-pooling layer), and finally a convolution layer (identical to the previous convolution layer). The output is then "fed" to a flattening layer and then a fully connected layer to obtain values for 32 hidden neurons, and finally, a fully connected layer with one output neuron. The model summary is shown below.



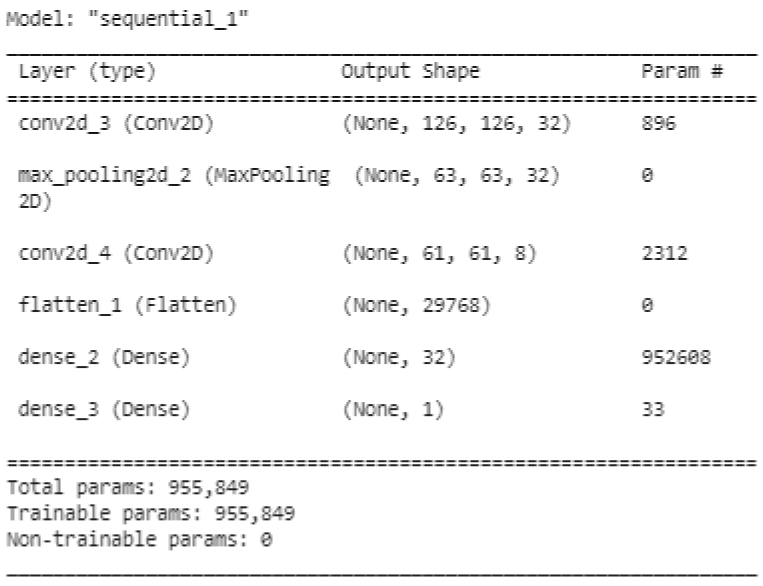
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 126, 126, 32)      896

 max_pooling2d_2 (MaxPooling  (None, 63, 63, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 61, 61, 8)         2312

 flatten_1 (Flatten)         (None, 29768)             0

 dense_2 (Dense)             (None, 32)                952608

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 955,849
Trainable params: 955,849
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Model 1 Summary

This architecture was chosen because it gave good results for the task of classifying CIFAR dataset images [11]. In our case, it gives a mean squared error of 0.5529, worse than just guessing 4 all the time!

Model 2:

We suspect that our model may be too complex. So, we make it simpler by removing one of the convolution and pooling layer pairs and reducing the number of channels from 64 to 8 in the last convolution. Furthermore, we reduce the filter from 5 by 5 to 3 by 3. We do not want to have too big a filter size in the first step of convolution, because we may miss out on localized regions within the pictures that we can learn from [12]. Model 2 gives a mean squared error of 0.3013, better than guessing 4 all the time. The model summary is shown below.

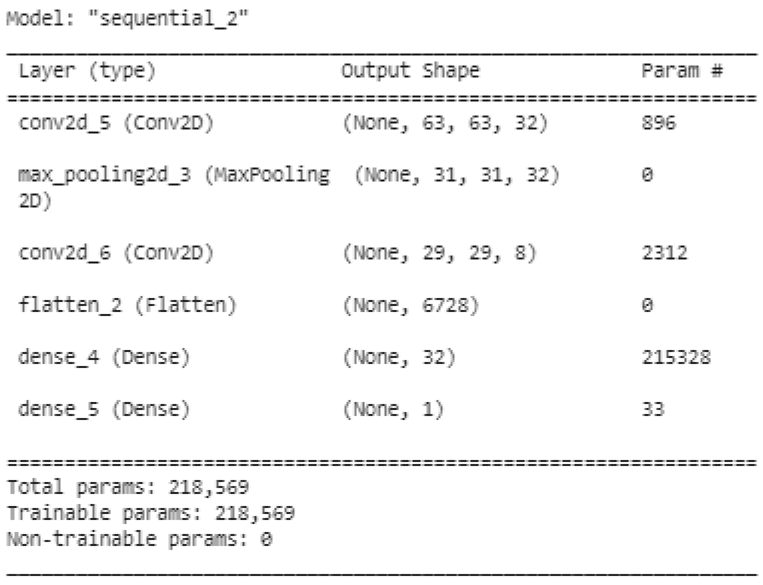


```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 126, 126, 32)      896

 max_pooling2d_2 (MaxPooling  (None, 63, 63, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 61, 61, 8)         2312

 flatten_1 (Flatten)         (None, 29768)             0

 dense_2 (Dense)             (None, 32)                952608

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 955,849
Trainable params: 955,849
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: Model 2 Summary

Model 3:

This is the same as model 2, but we change the stride parameter in the first convolution layer to be (2,2). The stride parameter is the number of pixel shifts over the input matrix when producing the output of a convolution layer. The idea here is to capture more general features while maintaining locality (3 by 3 filter) and whilst reducing network complexity [13]. The model summary is shown below.



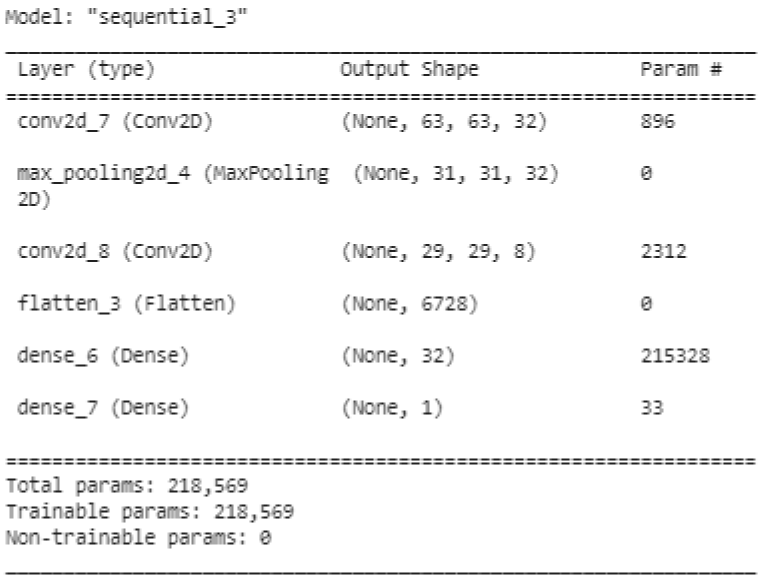
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 63, 63, 32)        896

 max_pooling2d_3 (MaxPooling  (None, 31, 31, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 29, 29, 8)         2312

 flatten_2 (Flatten)         (None, 6728)              0

 dense_4 (Dense)             (None, 32)                215328

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 218,569
Trainable params: 218,569
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Model 3 Summary

The result of training is a test loss of 0.3134. It turns out the trained network performs worse than model 2 on the test set. However, the deviation is not far off (0.0121/0.3013 = 4%), so we will keep the stride parameter to speed up training in future models, and we can remove it in our final model if it gives better performance for it.

Model 4:

We are unsure which activation function is best for our case. The literature seems to suggest ReLu is very popular for CNN training [14], but we will try tanh as it has given us good results in previous CNN tasks (ECE421 assignments). The network trained with tanh activation gives 0.3021 loss. This is better than 0.3134, so we will continue using tanh activation for our convolution layers. The model summary is shown below. Since we only change the activation function, it appears the same as Model 3.

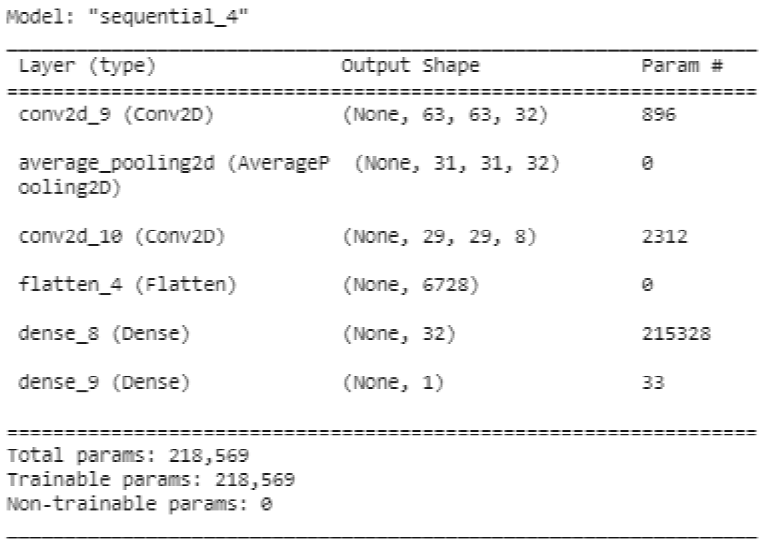


```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_7 (Conv2D)           (None, 63, 63, 32)        896       
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 31, 31, 32)       0         
 2D)                                                             
                                                                 
 conv2d_8 (Conv2D)           (None, 29, 29, 8)         2312      
                                                                 
 flatten_3 (Flatten)         (None, 6728)              0         
                                                                 
 dense_6 (Dense)             (None, 32)                215328    
                                                                 
 dense_7 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 218,569
Trainable params: 218,569
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: Model 4 Summary

Model 5:

The fifth model is obtained by replacing max-pooling with mean pooling, or average pooling. This turned out not to affect the mean squared error on the test set by any significant value. The mean squared error was 0.3072, which is larger than 0.3021 (Model 4), so we will stick with max pooling. This is consistent with many CNN implementations in literature and in practice [15]. The model summary is shown below.

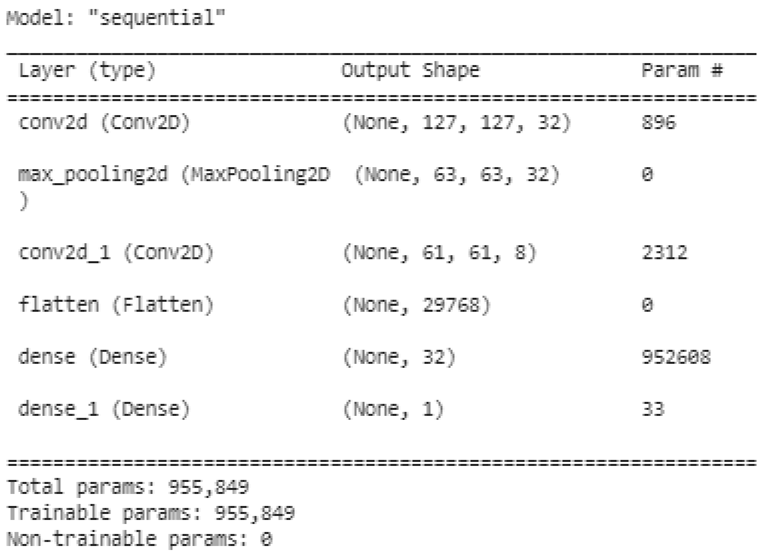


```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_9 (Conv2D)           (None, 63, 63, 32)        896       
                                                                 
 average_pooling2d (AverageP  (None, 31, 31, 32)       0         
 ooling2D)                                                       
                                                                 
 conv2d_10 (Conv2D)          (None, 29, 29, 8)         2312      
                                                                 
 flatten_4 (Flatten)         (None, 6728)              0         
                                                                 
 dense_8 (Dense)             (None, 32)                215328    
                                                                 
 dense_9 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 218,569
Trainable params: 218,569
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: Model 5 Summary

Model 6:

The last model is obtained by copying the architecture for model 4, but changing the image size to be 256 by 256. We suspect that a bigger image size may lead to learning better representations of the image when reducing dimensions through convolution or pooling.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 127, 127, 32)      896

 max_pooling2d (MaxPooling2D  (None, 63, 63, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 61, 61, 8)         2312

 flatten (Flatten)           (None, 29768)             0

 dense (Dense)               (None, 32)                952608

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 955,849
Trainable params: 955,849
Non-trainable params: 0
```

Figure 8: Best-performing CNN-based model

This model gives us our best results: a mean squared error of 0.2991 on the test set. The model summary is shown above. A summary of our simple CNN and hyperparameter tuning process is shown in the table below.

Table 1: Summary of Hyperparamter tuning process and results

Model	Test Accuracy	Architecture	Notes
Simple CNN	0.3249	• 2D convolution layer with 3 by 3 filter and ReLu activation • flattening layer • dense layer (final output)	

Model 1	0.5529	• 2D convolution layer with 32 channels, 5 by 5 filter and ReLu activation function • max pooling layer with a 2 by 2 window • convolution layer with 64 channels, a 3 by 3 filter and ReLu activation function • another max-pooling layer • a convolution layer • flattening layer • fully connected with 32 hidden neurons • final output	• chosen because it performs well on CIFAR dataset
Model 2	0.3013	• 2D convolution layer with 32 channels, 3 by 3 filter and ReLu activation function • max pooling layer with a 2 by 2 window • convolution layer with 8 channels, a 3 by 3 filter and ReLu activation function • flattening layer • fully connected with 32 hidden neurons • final output	• remove one of the convolution + pooling layer pairs • reduce number of channels from 64 to 8 in the last convolution • reduce the filter from 5 by 5 to 3 by 3.
Model 3	0.3134	• 2D convolution layer with 32 channels, 3 by 3 filter, 2 by 2 stride and ReLu activation function • max pooling layer with a 2 by 2 window • convolution layer with 8 channels, a 3 by 3 filter and ReLu activation function • flattening layer • fully connected with 32 hidden neurons • final output	• same as model 2 but add stride parameter

Model 4	0.3021	• 2D convolution layer with 32 channels, 3 by 3 filter, 2 by 2 stride and tanh activation function • max pooling layer with a 2 by 2 window • convolution layer with 8 channels, a 3 by 3 filter and tanh activation function • flattening layer • fully connected with 32 hidden neurons • final output	• same as model 3 but replace ReLu with tanh
Model 5	0.3072	• 2D convolution layer with 32 channels, 3 by 3 filter, 2 by 2 stride and tanh activation function • mean pooling layer with a 2 by 2 window • convolution layer with 8 channels, a 3 by 3 filter and tanh activation function • flattening layer • fully connected with 32 hidden neurons • final output	• same as model 4 but replace max pooling with mean pooling
Model 6	0.2991	• 2D convolution layer with 32 channels, 3 by 3 filter, 2 by 2 stride and tanh activation function • max pooling layer with a 2 by 2 window • convolution layer with 8 channels, a 3 by 3 filter and tanh activation function • flattening layer • fully connected with 32 hidden neurons • final output	• same as model 4 but change input dimensions from 128 * 128 to 256 * 256

The final model gives a relatively good mean squared error on the test set of 0.2991. However, in both our simple CNN and final CNN using hyperparameter tuning, a major problem is evident. Qualitative observations for our 6 CNNs are shown. It is clear that our network predicts between 3.5 and 4.5 all the time.

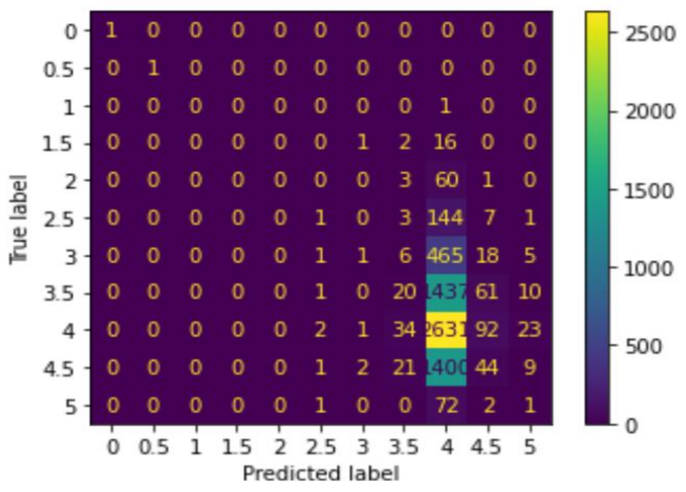
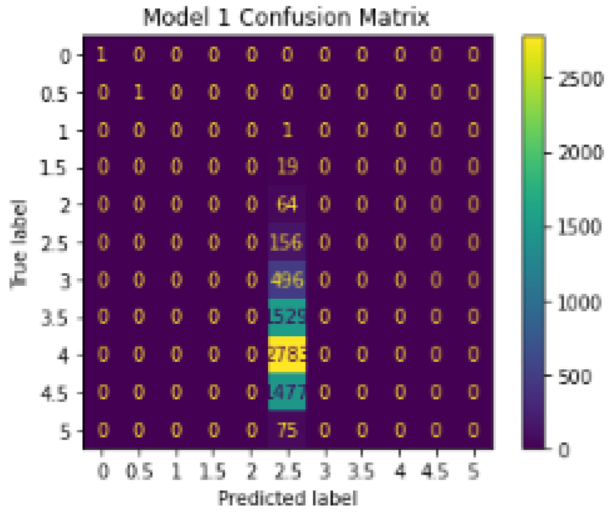
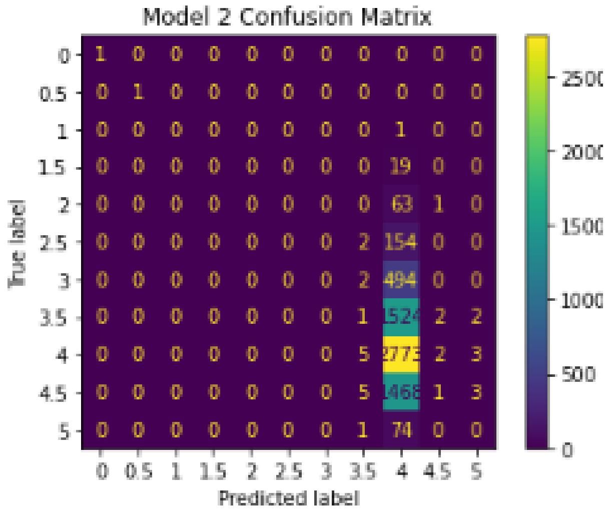
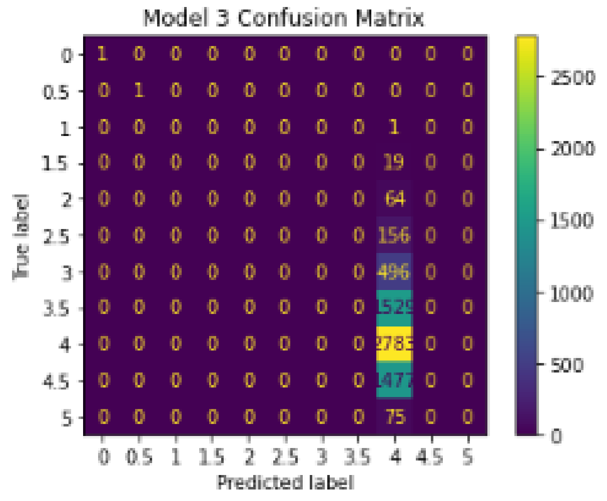
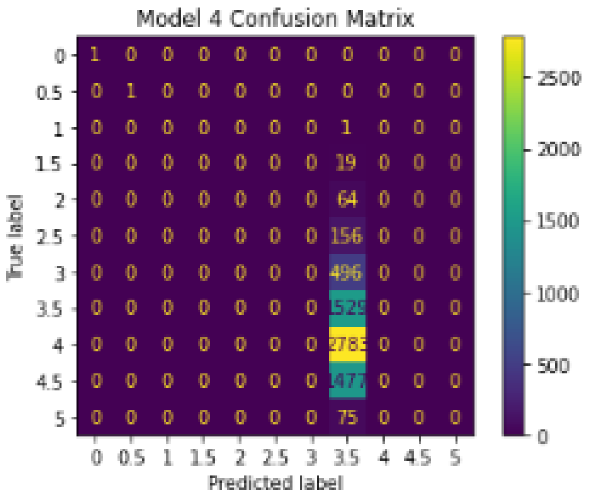


Figure 9: Confusion Matrix for Simple CNN









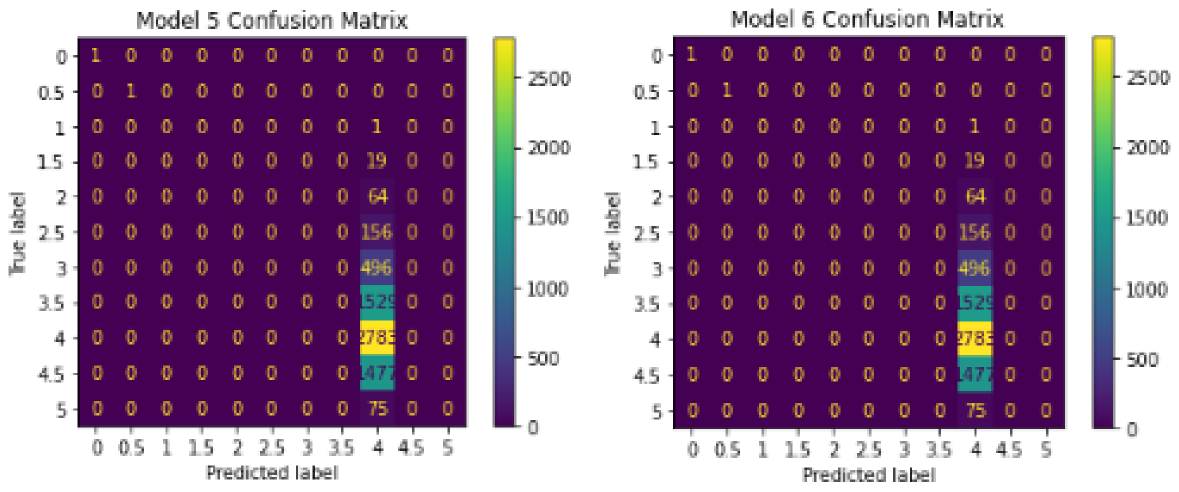


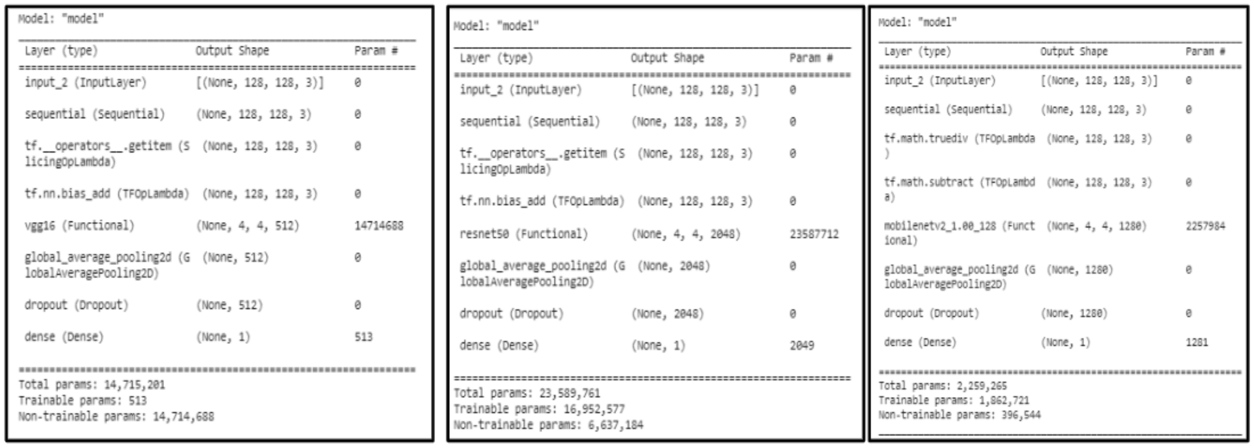
Figure 10: Confusion Matrix for the 6 models explored during hyperparameter tuning

V.4 Exploring Pre-Trained Models: Pre-trained CNNs as Feature Extractors

We now pivot to explore pre-trained and fine-tuned networks. A paper by Zeiler et al. attempted to gain a better understanding of features extracted from CNNs and determined that the hidden features generated are actually expressive patterns with increasing invariance and discrimination between classes (if the task is classfication) as the layers get deeper [6]. Furthermore, features get more powerful and discriminative as they are extracted from deeper/later layers [6]. So, for the three pre-trained networks we assess; VGG16, ResNet50, and MobileNetv2, we simply remove the top layer as well as add a global average pooling layer and a dense layer to get a 1-dimensional output. This is also more efficient (for fine-tuning) from a computational view since the features from the fully connected layers are much smaller in size than features from the convolutional layers[6].

When using these CNNs as feature extractors, we do not need to train the entire model. The base model already gives us features that are generally useful for image-based tasks. However, the last layer is responsible for the downstream task of predicting ratings, so we must train that on our own.

The 3 chosen networks were selected due to their prominence in literature. The model summaries after removing the top layer from the base layer and adding our own GlobalAveragePooling2D + Fully Connected Layer are shown below. We train each of these for 20 epochs, using base models pre-trained on the ImageNet dataset.



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 128, 128, 3)]     0

 sequential (Sequential)     (None, 128, 128, 3)       0

 tf.__operators__.getitem (S  (None, 128, 128, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 128, 128, 3)      0

 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 1)                 513

=================================================================
Total params: 14,715,201
Trainable params: 513
Non-trainable params: 14,714,688
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 128, 128, 3)]     0

 sequential (Sequential)     (None, 128, 128, 3)       0

 tf.__operators__.getitem (S  (None, 128, 128, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 128, 128, 3)      0

 resnet50 (Functional)       (None, 4, 4, 2048)        23587712

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 23,589,761
Trainable params: 16,952,577
Non-trainable params: 6,637,184
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 128, 128, 3)]     0

 sequential (Sequential)     (None, 128, 128, 3)       0

 tf.math.truediv (TFOpLambda  (None, 128, 128, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 128, 128, 3)      0
 a)

 mobilenetv2_1.00_128 (Funct  (None, 4, 4, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,862,721
Non-trainable params: 396,544
_________________________________________________________________
```

Figure 11: Refined VGG16, ResNet50 and MobileNetv2 architectures

V.5 Fine-Tuning VGG16, ResNet50, and MobileNetv2 architectures

In fine-tuning, we “unfreeze” a few of the top layers of the base model and train both the last few layers and the fully connected layer that we add ourselves. This allows us to fine-tune the high-level features to make them more representative of features that are specifically useful to learn restaurant food ratings [16].

The results of training the fully connected (FC) layer for 20 epochs, then unfreezing the last few layers and jointly training the last layers and our FC layer is shown below. We observe a small spike down when we begin fine-tuning, which demonstrates the effectiveness of this method.

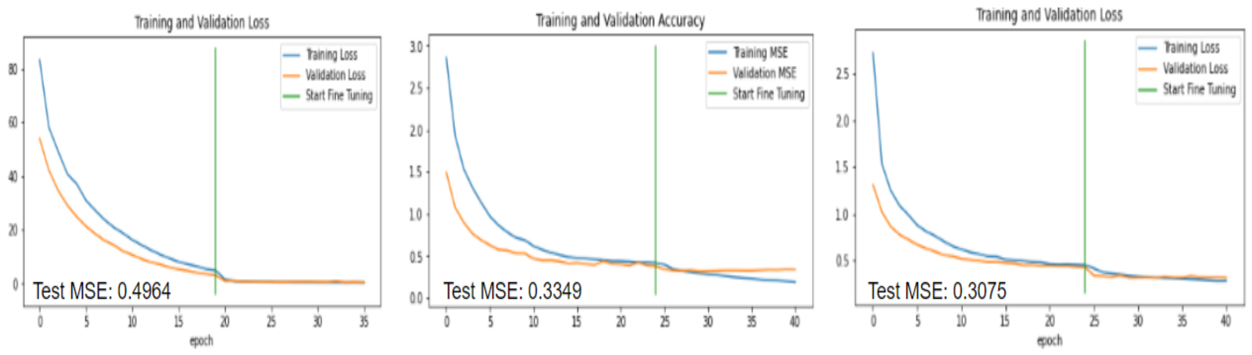


Figure 12: Train and Validation Loss (MSE) for VGG16 (left), ResNet50 (middle), and MobileNetv2 (right) after using the base model as a feature extractor for 20 epochs and fine-tuning for 15 epochs

V.6 CNN Ensembles

As mentioned in the Prior Work section, a reliable approach to improving overall performance would be to “combine” the models, in an attempt to derive a more general model to perform better on the overall task by assuming that the models will balance out any inherent biases one particular model may have [6].

As discussed in a paper by Wei et al. [17], there are two general classes of fusing the CNNs- early fusion and late fusion. In early fusion, we combine the features extracted by the individual CNNs and use this representation for the downstream task. In late fusion, we combine the predictions of the individual models. Examples of early fusion strategies include taking the average of the vectors and concatenating them. Examples of late fusion strategies include bagging, which involves a voting procedure, and stacked generalization, which trains an extra classifier on the outputs of the different CNNs.

We first try early fusion by stacking the feature vectors and training a classifier to predict ratings. The ensemble architecture diagram is shown below. It is clear that the 3 outputs are concatenated to give a final rating. The mean squared error of this approach is 0.3085.

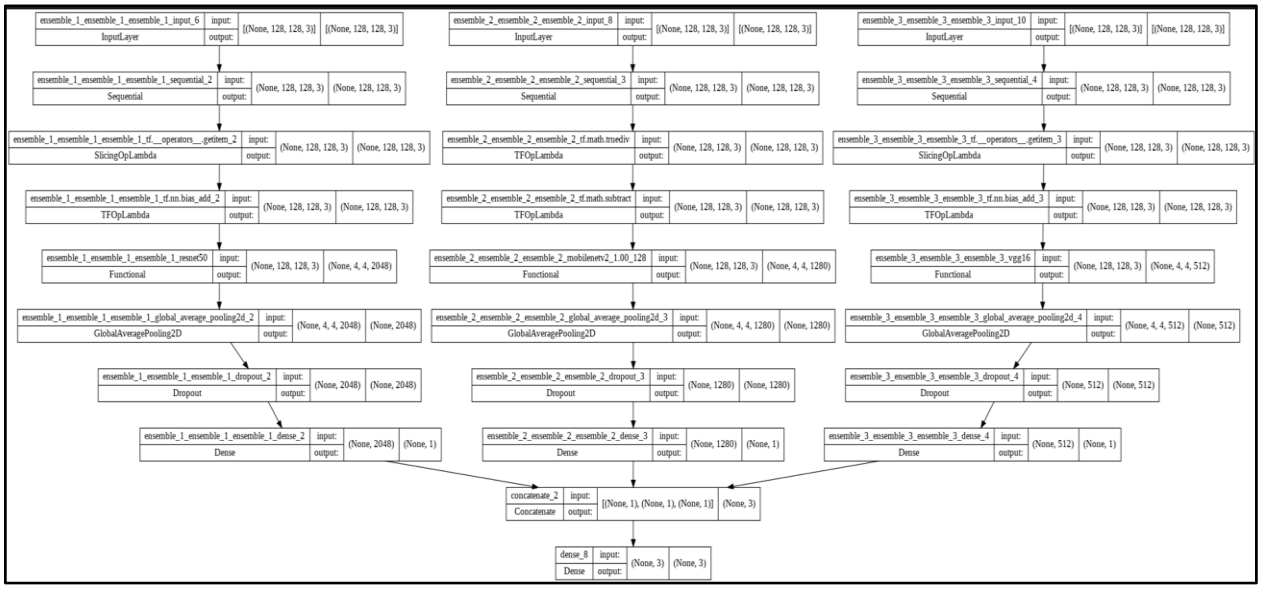


Figure 13: Ensemble Architecture Diagram

The ensemble approach does not give any better results in terms of the MSE. We used the early fusion method, but believe late fusion methods like bagging will give similar results. This is because we credit the error in our predictions mostly to a heavily skewed dataset and the fact that we use only half (20,000 / 40,000) of the images, not our choice of fusion.

The results of the ensemble approach are shown below. It is compared with the individual pre-trained networks in the table below. Analyzing the prediction distributions for the different true ratings, we see that the bars do move slightly to the right as rating increases, but we are still predicting 3 to 5 for images rated 1.5. Therefore, we will prioritize coming up with a more innovative architecture that we believe should give lower mean squared errors, before we play with early/late fusion, or using different pre-trained networks and datasets.

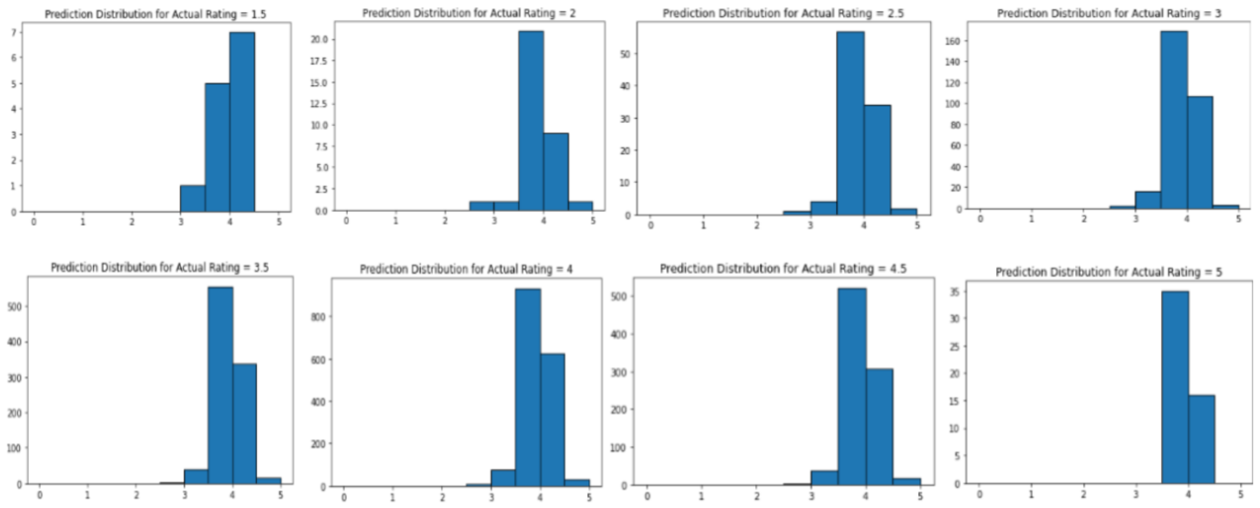


Figure 14: Prediction Distributions for different true ratings using ensemble approach

Table 2: Summary of Pre-trained and Ensemble Approaches

Model	Mean Squared Error (on Test Set, to nearest 4 decimals)
[For Reference] Simple CNN	0.3249
[For Reference] Model 6	0.2991
Fine-Tuned VGG16	0.4964
Fine-Tuned ResNet50	0.3349
Fine-Tuned MobileNetv2	0.3075
Ensemble Network	0.3085

VI. VGH-Net

VI.1 Synthesis of Past Work

The fairly lackluster performance we observe from basic convolutional networks calls for further exploration in the architecture design space. Given that we propose a novel input space—namely, images—for rating prediction, it is paramount that appropriate technical methods and formulations are used. We seek to first formulate an architecture that is capable of both image processing and regression to a linear rating scale from one to five stars.

We propose preliminary, high-level architecture choices that are known to perform well for the objectives outlined above. Though deep CNNs are most popularly used in computer vision tasks for object detection, classification, and segmentation in a wide variety of fields [18], the architecture has also shown great levels of efficacy in audio- [19] and text-based [20] tasks. Interestingly, we observe that CNNs are capable of sentiment analysis over both mediums to a high degree of performance [19, 20].

The foundations for additional architectural specifications for our CNN take inspiration from similar previous works that analyze data sourced from consumer review platforms—albeit, these papers take a more conventional approach through the use of metadata and text reviews to train their architectures. One work in particular by Tang et al. describes an effective neural network method that predicts Yelp and Rotten Tomatoes star ratings from text reviews. Text reviews are fed into a novel sentiment analysis pipeline that produces a vector representing the overall sentimental representation of the text. This sentiment vector is then used as an input into a final review prediction framework [21].

The applications of CNN architectures to sentiment analysis, its ubiquity in vision tasks, and the sentiment-to-rating pipeline above give us a reasonable degree of confidence that a CNN-based methodology will be successful. Tang et al.'s work also provides an attractive starting point for our image-based rating prediction architecture. In particular, the text-to-sentiment substructures detailed above can be replaced with some CNN-based image substructure that produces an analogous vector representing the image in sentiment or semantic space.

Indeed, CNNs may be most popularly known for their use in common tasks such as semantic labeling of images [22], they are capable of much more. It has been shown that they can also be used for sentiment analysis of images, which is an incredibly important aspect of our proposed objective. In particular, work by You et al. illustrates a fine-tuned CNN with transfer learning incorporated that is capable of image sentiment analysis over a training set of weakly labeled Flickr images [23]. Further research pertaining to the sentiment of social media images using various approaches such as colour composition and shape analysis may also prove to be useful [24][25].

We also take inspiration from DeViSE, a model that places images in semantic space by incorporating the use of both labeled images as well as training with unannotated text. DeViSE was able to glean useful semantic relationship information from its text training, which was then used to improve visual task success rates by 18%. This was done by making a visual network architecture target the specific embedding vectors produced by their text model during training. The primary focus for images was classification. Most notably, however, is that semantic information given by the text-trained component of the model gave DeViSE the ability to make meaningful inferences when classifying images to labels that it was not trained for [26]. We hope that some form of explicit image-to-text training will assist our model in learning semantic features that are indicative of ratings. Novel approaches to further fine-tune our sentiment targets also exist and may prove to be beneficial to the overall architecture. Specifically, a context-based approach as described by Kim et al. may help deal with concerns of subjectivity, especially in user-written reviews [27].

VI.2 Proposed Image-to-Rating Pipeline

The works discussed above provide invaluable insight for designing an effective architecture. The Yelp database is rich with text reviews that we can leverage to the same ends as DeViSE, where specific sentiment knowledge learned from reviews can be fed into our CNN as training targets. We have established that CNN sentiment analysis is feasible, and we expect a higher proportion of sentiment/emotion-nuanced context from reviews as opposed to a generic corpus.

To this end, we propose VGH-net, an architecture that combines two sub-structures that injects explicit human knowledge—or perhaps, intuition—into the learning process. Deliberate sentiment analysis will play a key role in our pipeline, as it provides a clear direction to achieving accurate rating predictions. Training a network to learn the *emotion* or *sentiment* depicted by an image appears to be more effective than classification on an intuitive level. Indeed, it would make sense that identifying a human sense of deliciousness or disgust from images, for example, could very well be more useful than identifying objects. We first aim to learn an embedding vector from images; we expect these vectors to contain valuable sentiment information. This embedding is then passed through a feedforward neural network to produce a single rating on the common five-star scale. These two distinct parts of our pipeline are trained independently and combined afterward.

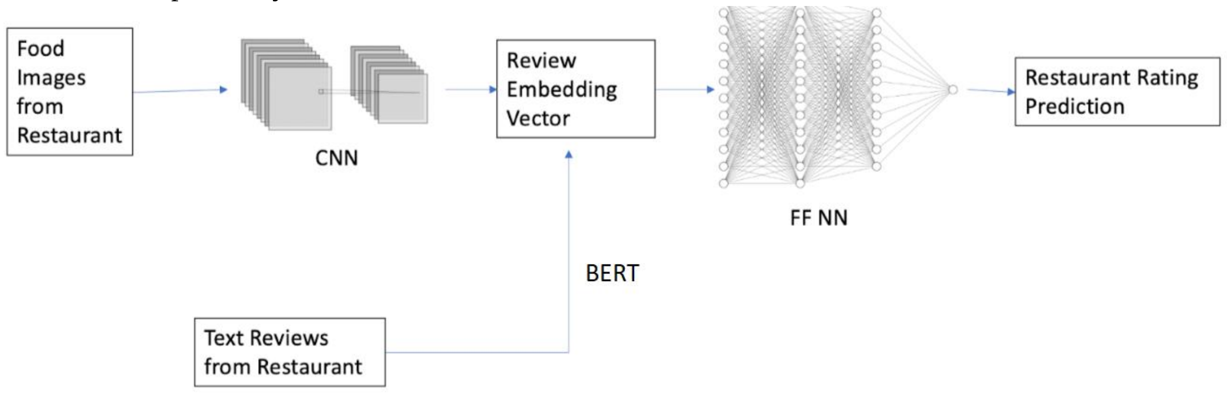

Food Images from Restaurant
CNN
Review Embedding Vector
FF NN
Restaurant Rating Prediction
BERT
Text Reviews from Restaurant

Figure 15: A high-level schematic of our proposed architecture.

VI.3 Image-to-Embedding Sub-Architecture

Our image-to-embedding architecture is a CNN that takes advantage of a novel training approach. Here, we incorporate text not as the input for our embedding, but instead as the target for which we wish to align our images. Text reviews are first transformed into a BERT embedding vector using a pre-trained BERT model from the popular Python package HuggingFace. Images corresponding to the reviews are then passed into the network as usual and trained.

This is achieved in the first sub-architecture, which consists of a CNN with a design informed by previous works and our results over pre-trained models as discussed in Section V. Our CNN consists of two convolutional layers and three fully connected layers. ReLU, max pooling, and batch normalization are performed after each convolutional layer, and sigmoid is used as the activation for the two hidden FC layers to capture non-linearities. The final layer of the sub-architecture has tanh applied, which restricts the output values to the same domain as the BERT embeddings—namely, the range [-1, 1]. Kaiming initialization is used on each layer with ReLU, as it is more effective and helps models converge faster [28]. Similarly, Xavier initialization is used on each layer that sigmoid or tanh is applied to for similar reasons [29]. Further specifics are depicted below.

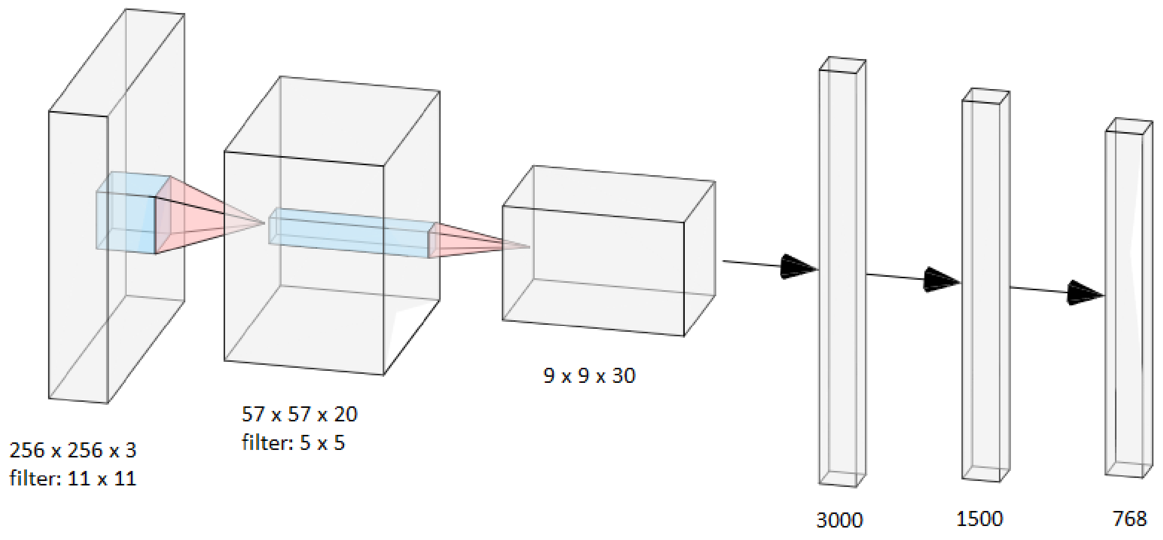


Figure 16: Image-to-Embedding CNN with all dimensions documented.

VI.4 Review-to-Rating Sub-Architecture

Our review-to-rating network consists of two fully connected layers. The network takes an embedding vector as the input and outputs a five-star rating. The dimensionality is 768 to 50 to 1. This architecture is trained explicitly on embedding to rating pairs. Similar to our CNN, we apply Xavier initialization to the weights of our fully connected network.

VII. Results

VII.1 Fully-Trained Dataset

We first train our CNN on image and review embedding pairs. Images are all resized to 256x256 with three channels for RGB. Here, approximately 30,000 examples with batch size 128 are used over five epochs using custom PyTorch dataloaders. A standard 7:2:1 split is used for training, test, and validation data respectively. Empirically, a learning rate of 1e-5 is shown to provide an adequate balance of computation time and weight update size. Updates of 1e-4 converge before the first epoch is finished; 1e-3 and higher results in the complete saturation of outputs to either -1 or 1 as the gradient disappears, even in the presence of aggressive weight decay parameters. Average MSE loss is minimized here to decrease the distance between the output and target embedding vectors. We favour Adam over SGD as adaptive optimizers are shown to converge more easily in the presence of architectures comprised of sensitive layers such as those with normalization and dropout. This is especially the case when these layers are applied to architectures that have not undergone hyper-specific design [30]. A final training loss of 0.00650 and a final test loss of 0.00637 is observed. Loss curves are shown below.

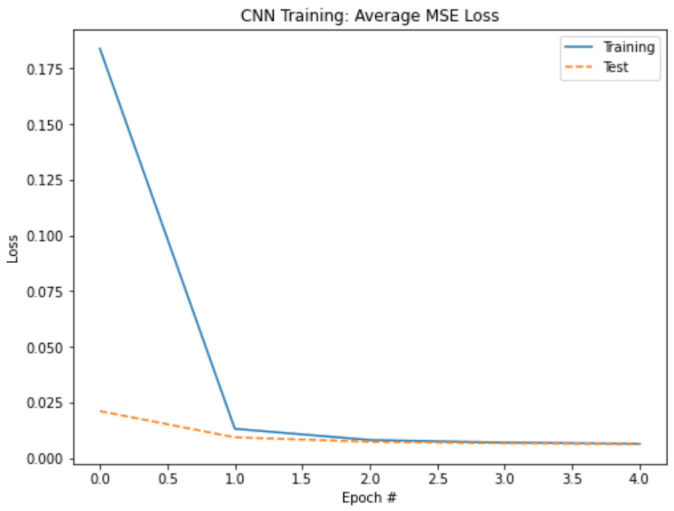


Figure 17: Average MSE Loss per epoch vs epoch number during full training of our CNN. 0 here refers to the 0th epoch as opposed to the starting loss.

The fully connected network is trained on this same training set, but now using the corresponding embeddings to ratings. A batch size of 512 is used over Adam and MSE loss. Again, the learning rate is empirically chosen as 1e-5. A final training loss of 1.481 and a final test loss of 1.492 is observed. Loss curves are shown below.

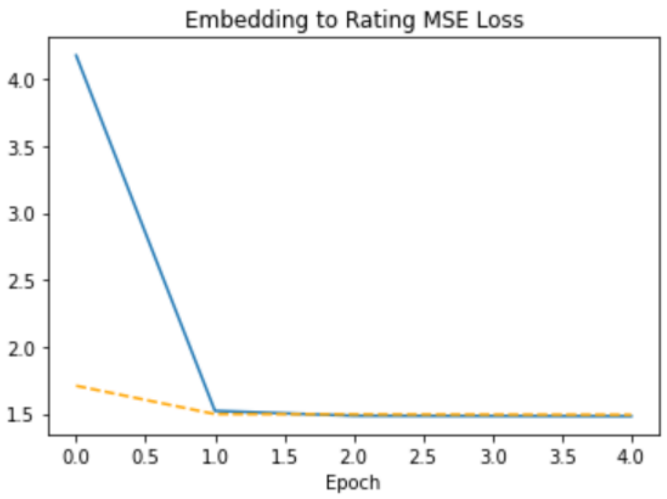


Figure 18: Average MSE Loss per epoch vs epoch number during full training of our FCN. The same epoch numbering convention as above applies.

Remarkably, we note a surprising amount of increased performance when we combine our two individual architectures. We now compare image-rating pairs taken from our validation and test sets over the full VGH-Net architecture. We attain a test MSE loss of 0.308 and a validation MSE loss of 0.309—significantly lower than the ratings loss using only review embeddings. Over our full model, we observe a 76% correct prediction rate. We define a prediction as correct if the predicted rating for an image is within a half-star margin of error for the target rating.

VII.2 Training on Pruned Dataset

Given that the full dataset was incredibly skewed, we attempt to provide a distribution of examples that better represents poor reviews. A dataset consisting of approximately 1200 reviews 2.5 stars and under and 2000 reviews of 3 stars and above is used. We note that this dataset is incredibly small compared to our original dataset. A plot of the new distribution is shown below.

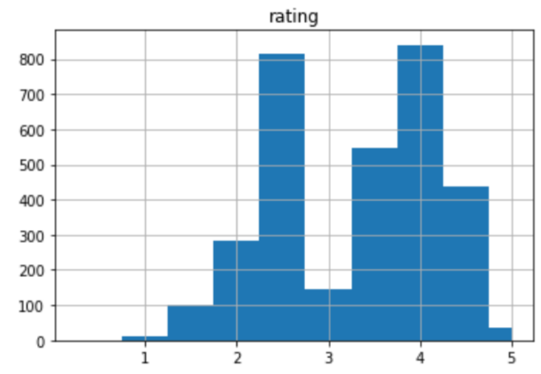


Figure 19: Distribution of pruned dataset for training.

We apply the same training procedure as outlined for our full dataset report, but with a batch size of 64 to account for the smaller dataset. Training runs for eight epochs. Average MSE Loss for the fully connected network is 1.674 and 1.377 for the training and test sets respectively. Loss curves are shown above.

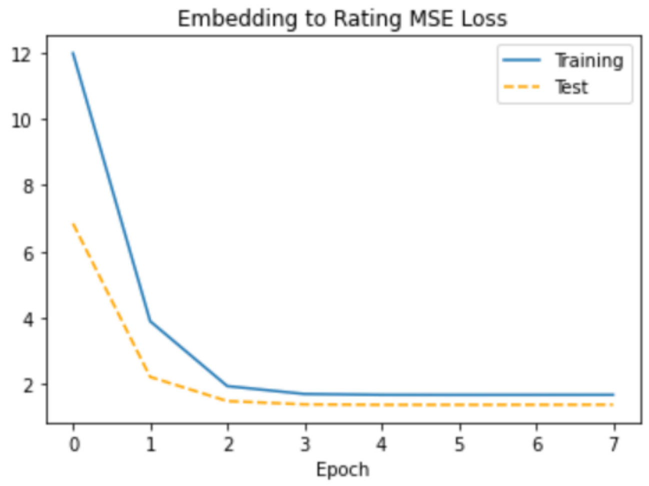


Figure 20: Average MSE Loss for fully connected layer during training.

The average MSE Loss for the CNN is 0.011 and 0.010 for the training and test sets respectively. Loss curves are shown below. The full architecture's test and validation losses are significantly higher at 0.829 and 0.844 respectively. This is most likely due to the significantly smaller dataset, so we do not pursue this design decision any further.

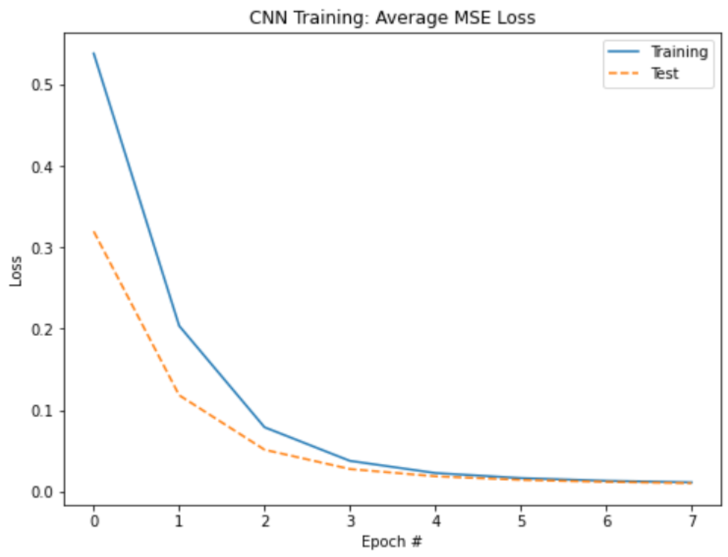


Figure 21: Average MSE Loss for CNN training.

IIX. Ethical Implications

The introduction of a widespread image-based review predictor comes with notable benefits and drawbacks. The effects of encoded racism and other biases in machine intelligence have been well-documented [31], and we acknowledge that an image-to-rating network has significant ethical impacts. This is further emphasized by the presence of racism and other discriminatory factors that cannot be neglected in reviews left in the hospitality industry [32].

Intuitively, a model trained on the Yelp dataset—or any publicly collected dataset, for that matter—is subject to the discriminatory content found within it. Racist sentiment can easily lead a network to associate lower scores with images containing food of a certain culture when given a set of racially motivated reviews and poor ratings. This is not limited to the confines of just racist text, either. Prevalently, coordinated attacks on a restaurant's ratings for any number of reasons, known as "review bombing" [33], can be propagated through the model as well. Without measures in place to isolate the negative effects of such incidents, a model trained on this data may incorrectly associate poor ratings with what it *can* see—namely, the food it is presented. What is dangerous is if, again, it learns to associate these reviews with some notion of race or culture that it learns from these images.

Additionally, BERT-trained embeddings leverage information gained from BookCorpus and Wikipedia [34]. Wikipedia is composed primarily of academic writing, and BookCorpus is a massive dataset of English book text. BookCorpus has been shown to be biased in several key areas including genre but more importantly religion [35]. Presumably, underrepresentation of race and culture is certainly present. Consequently, a model that employs pre-trained BERT embedding targets such as ours will not be able to appropriately capture the meaning of cultured language. The systemically negligent treatment of AAVE (African-American Vernacular in English), for instance, has been shown to result in poor performance of NLP models [36]. The aforementioned biases culminate in a model that perpetuates a discriminatory feedback cycle beginning with artificially lower ratings and ending with reduced patronage and increased prejudice against affected businesses. It has been shown that, indeed, ratings on platforms such as Yelp influence revenue and traffic [37].

Conversely, a model built for the same purposes as VGH-Net has the ability to become a very powerful tool to further democratize ratings while supporting and protecting small businesses. This is the case if an image-based review predictor is less susceptible to those wishing to game the system on the basis that faking pictures is considerably harder than posting fraudulent reviews. Furthermore, taking advantage of the sheer number of images on social media helps greatly mitigate the impact that malicious parties have when attempting to submit bad photos.

IX. Conclusion

In this project we have explored three general approaches to predicting Yelp restaurant ratings using photos of the food from user reviews. Firstly, using CNNs trained from scratch, we found that having two convolutional layers and a fully connected layer resulted in the benchmark performance. Secondly, using a pre-trained network or ensemble of pre-trained networks, we found the best performance from fine-tuned MobileNetv2. Thirdly, we created a novel architecture VGH-Net, which involves integrating BERT embeddings of restaurant reviews to create a two part network with a CNN component and feed-forward neural network component. After hyperparameter tuning, all three approaches performed similarly with MSE on the test dataset equal to 0.299, 0.308, and 0.308 respectively. However, there is considerable opportunity to further refine VGH-Net as outlined in the discussion section below. From a macro perspective, the project demonstrates the viability of using images, either alongside explicit ratings or alone, as an input to predict restaurant ratings.

X. Discussion

Our current VGH-Net architecture performs no worse than a simple CNN and is about the same as a fully tuned, more complex CNN (Model 6). While it is already showing promising results in terms of generalization, we have a set of easily implementable next steps that will help the next generation of VGH-Net, VGH-Net-Plus, to generalize better and avoid learning strategies that predict around the average all the time.

First, we would like to continue training VGH-Net on a larger dataset and perform hyperparameter tuning. Ideally, we would perform grid search, and explore different weight initializations, learning rates, and activation functions, as well as playing with different review-to-embedding strategies. We also wish to perform random search on architectural choices, specifically, exploring the design space by varying the number of features for the convolution layers and the number of hidden neurons in the fully connected layer.

While an initial attempt was made to balance the dataset, the loss (mean squared error) ended up being much higher. We credit this to a very small dataset after pruning. So, one next step would be to explore data balancing once we collect more data. We wish to explore different techniques other than just removing images. For example, balancing and rotating images to increase sample size; these include augmentations such as geometric and color transformations, random erasing, adversarial training, and neural style transfer [38]. Alternatively, we can continue scraping additional images only from restaurants with extreme (very low and very high) ratings.

Another way to deal with the tendency of our network to predict around the average is to adjust the loss function. Given any set of numbers, the scalar estimator that minimizes the mean squared error among the entire dataset is simply an estimator that predicts the mean. It is therefore not surprising that the network learns that predicting roughly the mean all the time minimizes the loss within some region of the loss curve. We thus want to explore loss functions that penalize predictions at a faster and faster rate the further off we are. This will avoid our network computing that it is sufficient to guess the mean for images with, say, a true rating of approximately 1.5 because there are only a few of them. Sample losses we would like to explore immediately are mean-cubed and mean to the fourth. Other losses are expected to be informed by literature.

Continuing, we are still in the process of downloading long reviews for all 40,000 images. We expect that these complete reviews will be much better at informing a network that outputs embeddings given images, since we are currently training a network to go from images to embeddings of truncated reviews.

Currently, we are using a pre-trained BERT model to go from text reviews to embeddings. While BERT is widely recognized for its successes in sentiment classification, amongst other NLP tasks [39], we are exploring using custom embeddings. These custom embeddings are formulated by training a network to go from sentence to sentiment prediction, then only using the last hidden layer in deployment. The model diagram is shown below; we have circled the embeddings we plan to use in deployment. Note that this network is already trained, and all that remains is to test it.

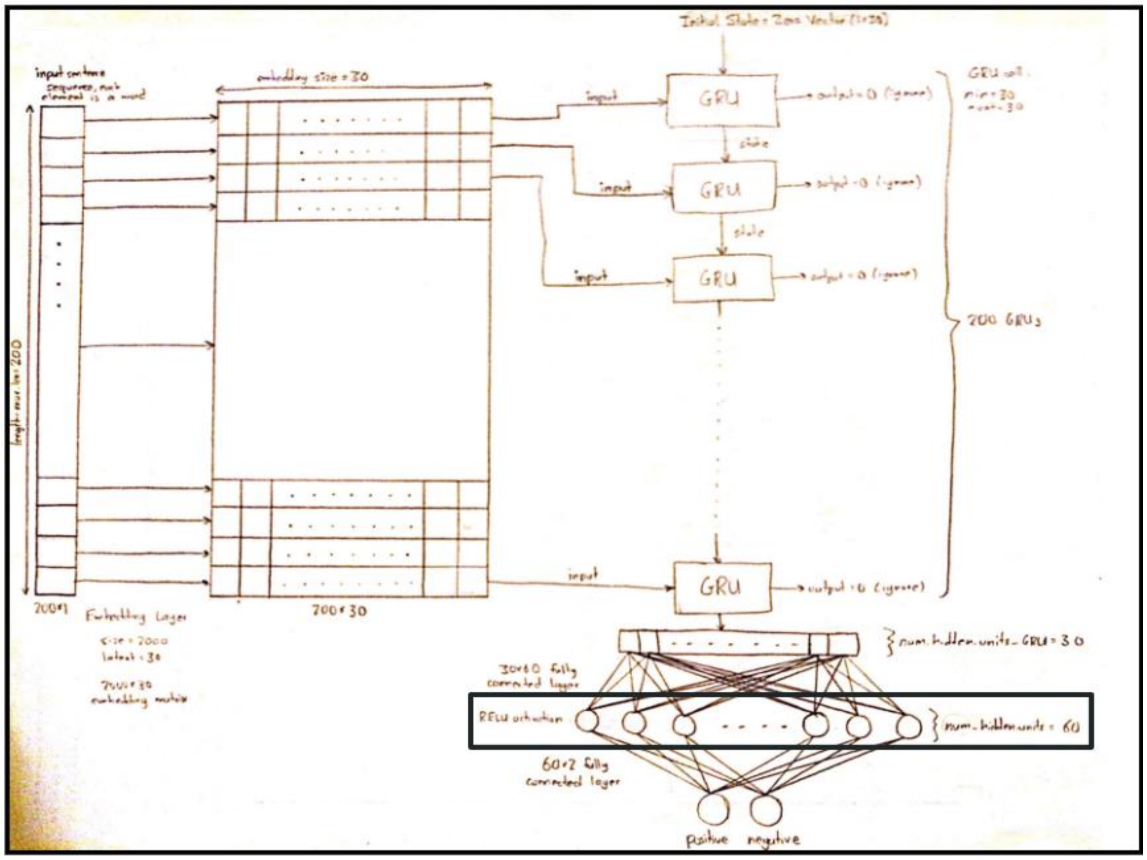


Figure 22: Proposed Network to generate Custom Embeddings

Finally, it is important to revisit the pre-trained networks and play with hyperparameters. Now that we have tested our VGH-Net and it shows similar results as the ensemble approach using pre-trained networks on the ImageNet dataset, it is worth going back and trying different datasets We plan to use the Food101 dataset, this way the dataset is tailored to our specific task.

XI. References

[1] Q4cdn.com. [Online]. Available: https://s24.q4cdn.com/521204325/files/doc_presentations/2022/Yelp-Investor-Presentation-Feb-022-update.pdf. [Accessed: 15-Apr-2022].

[2] S. Liu, “Sentiment analysis of Yelp reviews: A comparison of techniques and models,” arXiv [cs.CL], 2020.

[3] Y. Guo, A. Lu, and Z. Wang, “Predicting restaurants’ rating and popularity based on Yelp dataset,” Stanford.edu. [Online]. Available: http://cs229.stanford.edu/proj2017/final-reports/5244334.pdf. [Accessed: 15-Apr-2022].

[4] M. Zhang and L. Luo, “Can user generated content predict restaurant survival: Deep learning of yelp photos and reviews,” SSRN Electron. J., 2016.

[5] L. Alzubaidi et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” J. Big Data, vol. 8, no. 1, p. 53, 2021.

[6] Uva.nl. [Online]. Available: https://staff.fnwi.uva.nl/b.bredeweg/pdf/BSc/20152016/Baan.pdf. [Accessed: 15-Apr-2022].

[7] W. Ye, Y. Zhang, W. X. Zhao, X. Chen, and Z. Qin, “A collaborative neural model for rating prediction by leveraging user reviews and product images,” in Information Retrieval Technology, Cham: Springer International Publishing, 2017, pp. 99–111.

[8] L. Alzubaidi et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” J. Big Data, vol. 8, no. 1, p. 53, 2021.

[9] Wikipedia contributors, “Convolutional neural network,” Wikipedia, The Free Encyclopedia, 04-Apr-2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=1080936638.

[10] “Mean squared error,” Peltarion. [Online]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-f nctions/mean-squared-error. [Accessed: 15-Apr-2022].

[11] “Convolutional neural network (CNN),” TensorFlow. [Online]. Available: https://www.tensorflow.org/tutorials/images/cnn. [Accessed: 15-Apr-2022].

[12] H. Kumar, “Filters in convolutional neural networks,” Github.io. [Online]. Available: https://kharshit.github.io/blog/2018/12/14/filters-in-convolutional-neural-networks. [Accessed: 15-Apr-2022].

[13] Seb, “Understanding padding and stride in convolutional neural networks,” Programmathically - A Blog on Building Machine Learning Solutions, 03-Dec-2021. [Online]. Available: https://programmathically.com/understanding-padding-and-stride-in-convolutional-neural-netwoks/. [Accessed: 15-Apr-2022].

[14] H. Ide and T. Kurita, “Improvement of learning for CNN with ReLU activation by sparse regularization,” in 2017 International Joint Conference on Neural Networks (IJCNN), 2017, pp. 2684–2691.

[15] Z. Li, S.-H. Wang, R.-R. Fan, G. Cao, Y.-D. Zhang, and T. Guo, “Teeth category classification via

seven-layer deep convolutional neural network with max pooling and global average pooling,” Int. J. Imaging Syst. Technol., vol. 29, no. 4, pp. 577–583, 2019.

[16] “Transfer learning and fine-tuning,” TensorFlow. [Online]. Available: https://www.tensorflow.org/tutorials/images/transfer_learning. [Accessed: 15-Apr-2022].

[17] X.-S. Wei, B.-B. Gao, and J. Wu, “Deep spatial pyramid ensemble for cultural event recognition,”

in 2015 IEEE International Conference on Computer Vision Workshop (ICCVW), 2015, pp. 280–286.

[18] A. Dhillon and G. K. Verma, “Convolutional neural network: a review of models, methodologies and applications to object detection,” Prog. Artif. Intell., vol. 9, no. 2, pp. 85–112, 2020.

[19] M. T. García-Ordás, H. Alaiz-Moretón, J. A. Benítez-Andrades, I. García-Rodríguez, O. García-Olalla, and C. Benavides, “Sentiment analysis in non-fixed length audios using a Fully Convolutional Neural Network,” Biomed. Signal Process. Control, vol. 69, no. 102946, p. 102946, 2021.

[20] Y. Chen and Z. Zhang, “Research on text sentiment analysis based on CNNs and SVM,” in 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), 2018, pp. 2731–2734.

[21] D. Tang, B. Qin, T. Liu, and Y. Yang, “User modeling with neural network for review rating prediction,” Ijcai.org. [Online]. Available: https://www.ijcai.org/Proceedings/15/Papers/193.pdf. [Accessed: 15-Apr-2022].

[22] X. Qu, H. Che, J. Huang, L. Xu, and X. Zheng, “Multi-layered Semantic Representation Network for Multi-label image classification,” arXiv [cs.CV], 2021.

[23] Q. You, J. Luo, H. Jin, and J. Yang, “Robust Image Sentiment Analysis Using Progressively Trained and Domain Transferred Deep Networks,” Aaai.org. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/viewPDFInterstitial/9556/9270. [Accessed: 15-Apr-2022].

[24] N. Mittal, D. Sharma, and M. L. Joshi, “Image sentiment analysis using deep learning,” in 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2018, pp. 684–687.

[25] E. Ko, C. Yoon, and E. Y. Kim, “Discovering visual features for recognizing user’s sentiments in social images,” in 2016 International Conference on Big Data and Smart Computing (BigComp), 2016, pp. 378–381.

[26] A. Frome et al., “DeViSE: A deep visual-semantic embedding model,” Googleusercontent.com. [Online]. Available: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/41473.pdf. [Accessed: 15-Apr-2022].

[27] Y. Wang, G. Huang, J. Li, H. Li, Y. Zhou, and H. Jiang, “Refined global word embeddings based on sentiment concept for sentiment analysis,” IEEE Access, vol. 9, pp. 37075–37085, undefined 2021.

[28] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” arXiv [cs.CV], 2015.

[29] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” Mlr.press. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf. [Accessed: 15-Apr-2022].

[30] C. Garbin, X. Zhu, and O. Marques, “Dropout vs. batch normalization: an empirical study of their impact to deep learning,” Multimed. Tools Appl., vol. 79, no. 19–20, pp. 12777–12815, 2020.

[31] R. Benjamin, “Assessing risk, automating racism,” Science, vol. 366, no. 6464, pp. 421–422, 2019.

[32] S. Li, G. Li, R. Law, and Y. Paradies, “Racism in tourism reviews,” Tour. Manag., vol. 80, no. 104100, p. 104100, 2020.

[33] “Third international conference on Data Science & Social Research,” Uniba.it, 2020. [Online]. Available: https://www.uniba.it/ateneo/editoria-stampa-e-media/linea-editoriale/fuori-collana/BoA_DSSR.pdf#page=26. [Accessed: 15-Apr-2022].

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional Transformers for language understanding,” arXiv [cs.CL], 2018.

[35] J. Bandy, “Dirty secrets of BookCorpus, a key dataset in machine learning,” Towards Data Science, 12-May-2021. [Online]. Available: https://towardsdatascience.com/dirty-secrets-of-bookcorpus-a-key-dataset-in-machine-learning-6e2927e8650. [Accessed: 15-Apr-2022].

[36] S. Lin Blodge and B. O’connor, “Racial disparity in natural language processing: A case study of

social media African-American English," Arxiv.org. [Online]. Available: https://arxiv.org/pdf/1707.00061.pdf. [Accessed: 15-Apr-2022].

[37] M. Luca, "Reviews, reputation, and revenue: The case of yelp.Com," SSRN Electron. J., 2011.

[38] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," J. Big Data, vol. 6, no. 1, 2019.

[39] B. Lutkevich, "What is BERT (language model) and how does it work?," SearchEnterpriseAI, 27-Jan-2020. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model. [Accessed: 15-Apr-2022].