Recipe image classification using transfer learning

Leveraging VGG16 and Inception V3 to predict the presence of food items in images

Vanessa Grass
M.S. Data Science
University of New Haven
West Haven, CT

***Abstract* —** *The focus of this work is to build a model that classifies recipe images into food and non-food categories. Neural networks, namely convolutional neural networks, further enhanced by transfer learning have been shown to be very effective at these types of image classification tasks, hence different neural network architectures were explored in order to extract and classify features in food and non-food images. The best performing model achieves 93% accuracy which is on par with similar experiments.*

I. INTRODUCTION

Wellio is a recipe planning platform dedicated to making eating well easier and more convenient — this means personalizing meals by different metrics such as nutrition, variety, and cost, with the end goal of improving the user's diet and overall health.

Wellio's recommendation system allows users to search, save, and purchase ingredients for recipes they wish to cook. As such, users need to be shown enough at-a-glance insight into recipes, enabling them to make a recipe selection with confidence and ease. Often a photo of the recipe is the best way to achieve this. The image, therefore, must be descriptive of the recipe — generally a photo of the final product. In addition to users being more inclined to select recipes with accompanying images that are relevant to their search query, users tend to choose recipes that are also more visually appealing.

In Wellio's recipe database, there exist many instances in which the recipe image is not relevant to the recipe or is visually unappealing. To be clear, relevancy and appealingness are open-ended concepts. In the context of Wellio's problem, non-relevant images are generally represented by placeholder images (from recipes that don't have an accompanying photo of the actual final product), images of plates, eating utensils and cooking tools, illustrations, and images in which people are the focus. We can broadly categorize these as non-food images. Unappealing food images are more subjective, but in general are represented by food images with large graphic or text overlays, food images with poor focus, bad lighting, and sloppy plating of food items.

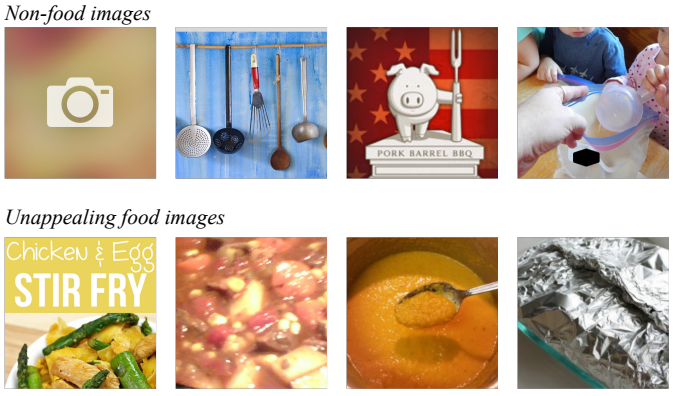

Figure 1. Examples of Wellio recipe images

The main goal of the work presented is to build a classifier that can filter out non-food and, to some extent, unappealing food images from Wellio's search results. The hypothesis driving this goal is the assumption that both relevance and appealingness of a recipe image are factors in the user's assessment of whether the recipe itself is appealing. This, by extension, greatly influences whether the user selects a recipe to cook. Central to this is the idea of user experience — surfacing highly relevant and appealing recipe images in Wellio's search channels will lead to higher customer satisfaction among Wellio users.

II. EXPLORATORY DATA ANALYSIS (EDA)

Wellio currently has over 15 million recipe images in its database. This represents a potentially huge training dataset, but unfortunately these images are entirely unlabeled regarding relevance and appeal. However, each recipe image is associated with a list of ingredients, preparation steps, and source. Recipes come from a variety of sources such as blogs and online publications. Anecdotally, certain sources are associated with higher incidences of non-food or unappealing food recipe images and often particular sources are associated with both.

The aim of EDA was to ascertain if Wellio recipe source could be a good proxy for recipe image relevance and appeal. The ultimate goal is to use this proxy relationship to automatically label recipe images on relevance and appeal based on which recipe source they come from. The data examined came from CrowdFlower (a data mining and crowdsourcing platform) in which participants were asked to score the relevance of search results given a particular query and were asked the question "Does this recipe look appealing?" given a recipe title and image.

In short, the EDA was inconclusive as to whether recipe source is a good proxy for image relevance and appeal. More data is needed — many of the sources from the CrowdFlower data only have one recipe associated with them which is not enough to make any inferences. Another factor is the role of personal preference and taste when evaluating the appeal of recipe images. Someone might rate a recipe image as unappealing even though it may have elements of a more objectively high-quality image simply because the person

evaluating the image doesn't like a certain ingredient used in the recipe. A related (but unvalidated) observation is it seems as though "unhealthy" recipe images, for example, images of burgers, pizza, etc., are more often rated as appealing as opposed to images of vegetables and salads. There is also the issue of foods that inherently don't photograph as well as others (for example, soup).

Without quantitative proof, it's intuitively obvious that relevance and appealingess are related to some extent. For example, images of food that are very visually unappealing are often not even immediately recognizable as food. As such the goal of the project was simplified to detecting whether an image contains food or not, a binary image classification task.

III. DEEP LEARNING FOR IMAGE CLASSIFICATION

Before the widespread adoption of deep learning, image classification was performed via hand-crafted extraction of features in images (such as edges and corners). Algorithms like Scale-Invariant Feature Transform (SIFT) [1], which is aptly named, were used to extract features. These features were then passed to a linear classifier, such as a Support Vector Machine (SVM) to ultimately classify the image.

Modern day image classification generally involves the use of a convolutional neural network (CNN). The basic premise behind a CNN is that it takes an image and then through a series of several layers and operations recognizes increasingly complex features in the training data. The CNN then outputs (in the case of binary classification) a single value between zero and one. This value represents the probability associated with the input being of a particular class. The novelty of a CNN over hand-crafted feature extraction techniques is that features extracted by a CNN are learned using the data rather than manually constructed [1].

A. Convolutional Neural Networks

CNNs are made up of four main components: convolution, non-linearity, pooling (subsampling or downsampling), and a final fully connected layer (for classification). Backpropagation is the crucial step that powers learning during the training process.

In the convolutional layers, image features are extracted by sliding filters (also known as feature detectors), often a 3 x 3 or 5 x 5 dimensional matrix, over the image (which is also a matrix of pixel values). The dot product is then computed between the filter and image to generate feature maps. The filter size, as well as the number of filters to use, are hyper-parameters under the user's control. The image features are learned hierarchically — lower convolutional layers learn lower-level features such as corners and edges, middle layers learn color and shape, and higher layers learn high-level features representing the objects in the image (for example, in the case of an image of a cat, higher layers learn what features make up a cat ear) [2].

After each convolutional layer, a non-linearity is introduced, an activation function which encodes when the individual "neurons" (or nodes) in the network should "fire" (or activate) given certain features. Generally, a Rectified Linear Unit (ReLU) activation works well in CNNs [3]. ReLU activation replaces all negative values in the feature map with zeros. Since convolution is still a linear operation but image data is non-linear, a non-linear activation function helps the network to learn non-linear data [3].

Next is pooling, which similar to convolution, involves the use of a two-dimensional matrix as a filter. These pooling filters reduce each feature map's dimensionality, while still retaining the most important information. There are several types of pooling, such as max, average, and sum. Max pooling, in which the rectified feature map is reduced to the max values ascertained from passing a m x n filter over the map, has been shown to work quite well [4]. Pooling helps make feature learning more manageable by reducing the number of parameters (and reduces the number of computations) which helps in over-fitting. Another benefit of pooling is it helps make the network invariant to transformations and translations. For image classification this is very beneficial since the same type of object can appear at different scales and orientations.

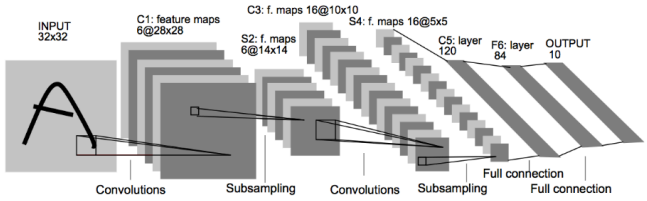

Figure 2. The architecture of LeNet-5 demonstrates the different layers and operations in a CNN [5]

The final layer in a CNN is a fully connected layer. High-level features are extracted from the previous convolutional and pooling layers and the function of the fully connected layer is to use these feature maps to classify the images into various classes. The term "fully connected" means every node in the previous layer is connected to every node in the next layer [4]. This is good for classification as each high-level feature gets a vote in the class prediction. Again, a non-linear activation is applied to the final output. In the case of binary classification, this activation is a sigmoid function [4], which squashes values between zero and one. These then represent the probabilities of belonging to a given class. A threshold is selected and probabilities below the given threshold are assigned to the negative class, probabilities above the given threshold are assigned to the positive class.

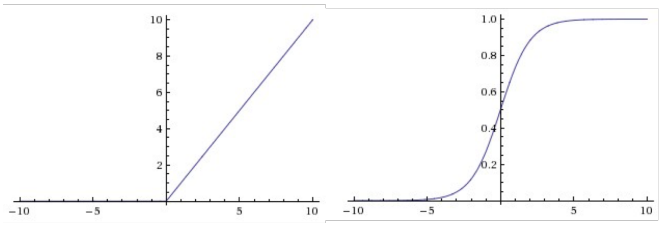

Figure 3. Different activation functions — left: ReLU, right: Sigmoid [3]

During the forward pass of the network, the CNN takes an image as input, goes through convolution, ReLU, pooling, and then the final fully connected layer. Because the network is

initialized with random values, the final outputted probability for each class is also random. To make improvements to this prediction, the error is calculated using a loss function (such as mean square error). Backpropagation then calculates the gradient (that is the derivatives) of the loss function with respect to all the network's parameters (or filter values, also known as weights) [6]. The weights are then updated via gradient descent to minimize the error. An important hyper-parameter is the learning rate, which specifies the step size to take when performing gradient descent [6]. This whole process is then performed across all images in the training set.

$$E_{total} = \sum \frac{1}{2}(actual - predicted)^2 \qquad (1)$$

B. *Transfer Learning*

Instead of training a CNN from scratch (which is difficult for multiple reasons — it requires a very large dataset, compute power, and time), it is much more common and efficient to use a pre-trained CNN. A pre-trained CNN is generally a deep network, comprised of many convolutional and pooling layers, previously trained on a very large and varied set of images. Such a large, varied dataset allows the pre-trained model to make very good generalizations on universal features occurring in images that can be repurposed for a variety of computer vision tasks. There are two ways to approach transfer learning — feature extraction and fine-tuning [7].

C. *Feature Extraction*

Feature extraction uses patterns learned by a previously trained network to extract representative features from new data. These features are then used to train a new classifier from scratch. As discussed previously, the convolutional base (the convolutional layers) in a CNN is essentially a feature extractor. The base of a pre-trained model can therefore be used to extract features from a new set of images, generating a set of feature maps which, in conjunction with their associated labels, are passed to a classifier to generate class predictions.

Generally, this approach is good for problems in which the dataset is small and similar to the data the original pre-trained network was trained on [7]. However, it can also be leveraged for problems in which the data is small but dissimilar from the original data. In this case, only the first few layers of the convolutional base are used to generate feature maps, as these lower layers detect very generic features universal to all images [7]. While this approach of a convolutional base + classifier is computationally fast to run (and can be performed on CPU), it does not leverage data augmentation. Data augmentation is a technique to artificially create additional training examples by randomly transforming images (i.e. rotation, flips, etc.) and has shown to be quite effective in reducing overfitting [8]. The reason it can't be applied in this approach is because the pre-trained convolutional base isn't learning new features from the data, only generating feature maps based on what's already learned from the previous data it was trained on.

D. *Fine-tuning*

Fine-tuning is another approach to transfer learning. It involves extending the convolutional base by adding additional fully connected layers on top and training the whole network end-to-end [7]. As such, it can make use of data augmentation. However, this approach is much slower and computationally expensive (and requires GPU). The term "fine-tuning" comes from the unfreezing (which allows the weights to get updated during training) of a few of the top layers in the convolutional base. These convolutional layers are then trained along with the newly added fully connected layers. There are two benefits to this approach — it can leverage data augmentation and it can learn more nuanced features relevant to the problem at hand [7].

Fine-tuning works well on a dataset that is large and similar to the original dataset — overfitting is less of an issue with more data. Fine-tuning can also be leveraged when the dataset is large but very different from the original dataset. It's often the case that it's still advantageous to use a pre-trained network with its initialized weights instead of starting from scratch [7].

IV. RELATED WORK

While CNNs have been around since the early 1990s, they were still "in beta" so to speak. With the advent of more computing power and data, CNNs were able to solve much more challenging problems [1]. The past five years have seen a boom in their usage in computer vision problems. Some influential projects and architectures built upon in this paper are discussed below.

A. *ImageNet*

ImageNet is a project that contains over 1.4 million labeled images organized into ~22,000 categories [9]. These categories encompass a multitude of objects such as different animals, vehicles, household items, and even various kinds of foods. The state-of-the-art pre-trained networks are trained on ImageNet in response to the ImageNet Large Scale Visual Recognition Challenge. The aim of this challenge is to correctly classify an image into 1,000 separate classes and is used as a benchmark for image classification problems.

B. *VGG*

Developed at the University of Oxford's Visual Geometry Group, the VGG architecture took 2nd place at the ImageNet Challenge in 2014 in the classification task. Both VGG16 and VGG19 refer to the number of weight layers in the network

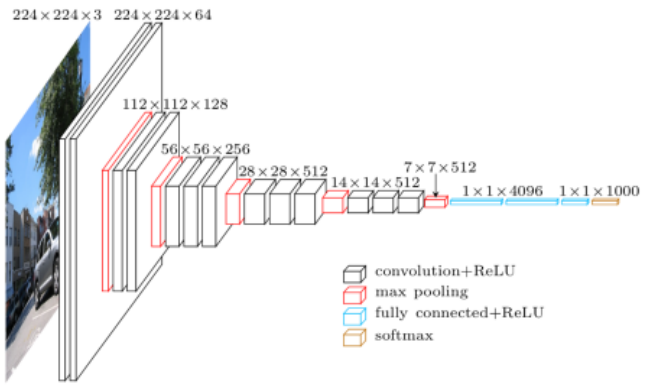

Figure 4. The architecture of VGG16 [11]

and demonstrate that the number of layers in a network is crucial for performance. While relatively straightforward in terms of architecture, VGG is slow and quite large [10].

C. *GoogLeNet*

The GoogLeNet (since renamed Inception) architecture was the ImageNet Challenge winner in 2014 and comes out of research at Google. Its novelty is the Inception module, which operates as a mini model instead a bigger model (very much like the 2010 movie by the same name). These modules act like multi-level feature extractors which perform convolutions in parallel. This results in a faster and smaller model architecture than VGG.

D. *Food Image Detection and Recognition*

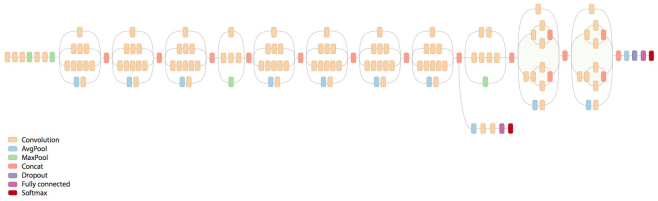

Figure 5. The architecture of Inception V3 [12]

Food image detection is binary classification, that is, classifying a given image into food or non-food classes. Food image recognition, on the other hand, is a multi-class classification problem that seeks to recognize different food item classes. These are both similar to other computer vision tasks, and research into this area follows the current trend of using CNNs, leveraging pre-trained models, and performing fine-tuning to solve these types of tasks with rather successful results [13], [14].

V. METHODOLOGY

The general approach in building a recipe image food classification system was to first create a labeled dataset of food images for the positive training examples and non-food images for the negative training examples. While Wellio has a large amount of recipe image data, it is unlabeled. Therefore, ImageNet image data was used. Next two pre-trained models, VGG16 and Inception V3 were explored to create feature maps. These feature maps were then passed through various model architectures to evaluate the classifier's performance using a variety of evaluation metrics.

A. *Data & Preprocessing*

The image data comes from sampling ImageNet images (as well as providing some additional training data in the form of food images from recipe sites and blogs). ImageNet images are organized into various different synsets (synonym sets), according to the WordNet (a lexical database) hierarchy. The food image dataset was created by randomly sampling from ten manually selected food related synsets. The non-food image dataset was also created by randomly sampling from all synsets (excluding the manually selected food related synsets mentioned previously, plus some additional food related synsets).

This resulted in:

<u>2,400 training samples</u>

- 1,200 food images
- 1,200 non-food images

<u>800 validation samples</u>

- 400 food images
- 400 non-food images

A limitation of both the training and validation data is that the resulting data is noisy. The non-food image data contains some food images. This is because randomly sampling all synsets minus only *some* food related synsets, does not actually remove all food related synsets. Similarly, the food image data contains some images that are not quite of food per se. For example, an image of a group of people sitting around a table eating dinner or an image of a bag of Doritos. The food image data also contains several low-quality food images, which include food images with text overlay.

The hold out data is very small and consists of two manually labeled datasets — one for non-food images and another for visually *unappealing* food images from Wellio recipe image data. The desire being that the model recognizes neither hold out set as food images.

Some image preprocessing is necessary — namely normalizing pixel intensity values by 255 (this keeps the network's weights from oscillating wildly during training) and resizing all images to be the same height and width (which allows convolution and pooling to be performed in a reasonable manner). For this dataset, all images were resized to be 150 x 150 pixels.

B. *Technical Details*

All experiments were run on Google Cloud Platform's Compute Engine using only CPU. The Keras API was chosen as the deep learning framework for its python compatibility and ease of use for fast experimentation. Keras also contains several state-of-the-art pre-trained networks such as VGG16 and Inception V3 which were used for feature extraction. The scikit-learn library was implemented to establish a baseline model as well for its model evaluation metrics. And last but not least, pandas, matplotlib, and seaborn were used for data visualizations.

C. *Feature Extraction*

As mentioned previously, the general strategy for leveraging transfer learning on small datasets that share similarity to the original data the pre-trained network was trained on, is to first generate feature maps using the pre-trained convolutional base and separately run these feature maps through a final classifier for predictions. Since the data used in this project is both small and similar (or rather the exact same) data used to train the selected pre-trained models (VGG16 and Inception V3) this is the method that was employed. To reiterate, the training and validation data both come from ImageNet so a caveat here is that the feature maps generated from VGG16 and Inception V3 may be more resolved as these networks have learned the features in these

images over many training epochs. Whether or not this leads to some over-fitting is unclear. However, since this approach removes the last fully connected layers in the pre-trained networks used, nodes that detected features that are more high-level and nuanced to the given classification task at hand — in the case of VGG16 and Inception V3 classifying 1,000 classes, are not present and therefore overfitting may not be problematic after all.

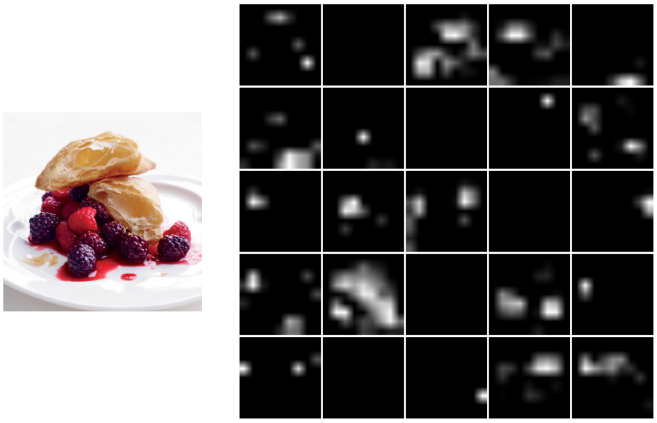

Figure 6. Original recipe image (left), 25 randomly selected feature maps generated by VGG16's convolutional base (right)

The original VGG16 network was trained on 224 x 224 pixel input images and Inception V3 was trained on 299 x 299 pixel input images. Because the images in this project's dataset were resized to 150 x 150 pixels (to speed up feature map generation), a new input dimension of 150 x 150 x 3 was specified for both pre-trained models. For each image in the dataset this resulted in the generation of 512 4 x 4 pixel and 2,048 3 x 3 pixel feature maps for VGG16 and Inception V3 respectively.

D. Baseline Models

A non-neural linear model was used as a "baseline" model to ascertain the best final classifier to use in conjunction with the feature maps. Scikit-learn's stochastic gradient descent (SGD) classifier was selected with hinge loss, returning an SVM classifier which has been successfully used in past computer vision problems.

E. Neural Network Architectures

In addition to the aforementioned more traditional machine learning SGD classifier, several relatively simple multi-layer perceptron (MLP) architectures were explored using the feature maps generated by both VGG16 and Inception V3. Specific hyper-parameters such as the number of hidden layers, hidden units, as well as regularization (dropout) were manually tuned, and the network was trained for 30 epochs.

VI. EXPERIMENTAL RESULTS

A. Model Comparison

Passing the feature maps and associated labels generated via VGG16's convolutional base through an SGD classifier with log loss resulted in 77.4% accuracy on the validation set. Using an SGD classifier with hinge loss resulted in 84.4% accuracy on the validation set. The same strategy was employed with feature maps generated via Inception V3's convolutional base. An SGD classifier with log loss resulted in 91.4% accuracy and using hinge loss resulted in 90.1% accuracy on the validation set. These were already quite favorable results.

VGG16 feature maps + MLP classifier				
Layers	***Hidden Units***	***Dropout***	***Validation Accuracy***	***Delta***
1 Dense	0	0	85.3%	0.13%
1 Dense + 1 Hidden	64	0	87.9%	0.86%
1 Dense + 1 Hidden	256	0.5	88.7%	0.75%
1 Dense + 2 Hidden	64, 64	0.25, 0.25	87.9%	0.54%
1 Dense + 2 Hidden	256, 64	0.25, 0.25	87.4%	0.97%

Inception V3 feature maps + MLP classifier				
Layers	***Hidden Units***	***Dropout***	***Validation Accuracy***	***Delta***
1 Dense	0	0	91.7%	0.63%
1 Dense + 1 Hidden	64	0	92.5%	0.75%
1 Dense + 1 Hidden	256	0.5	92.9%	0.71%
1 Dense + 2 Hidden	64, 64	0.25, 0.25	92.5%	0.75%
1 Dense + 2 Hidden	256, 64	0.25, 0.25	92.2%	0.77%

Table 1 & 2. Hyper-parameter Exploration and Results

The results for the different MLP architectures are also quite favorable. The VGG16 feature maps + MLP variants achieved an average of 87% accuracy on the validation set. The Inception V3 feature maps + MLP variants achieved an average of 92% accuracy on the validation set. See Tables 1 and 2 for enumerated results for each pre-trained model feature map and MLP variant.

Learning curves also serve as a nice way to compare model performance. Note, the learning curves show that validation accuracy is sometimes higher than training accuracy almost immediately. This is because Keras has two modes: training and validation. Regularization techniques, such as dropout are not applied to the validation set. Also, the training accuracy is an average of the accuracies over each batch of training data. Since the networks change over time, the accuracy for the first batches of an epoch is generally lower than over the last batches. The validation accuracy for an epoch is calculated using the network as it is at the end of the epoch, resulting in a higher accuracy for that epoch.

B. Evaluation Metrics

To assess additional evaluation metrics, the model with the best balance of validation accuracy and complexity was selected for further analyses. For experiments using VGG16 feature maps, the model architecture of two layers with 64 hidden units and 25% dropout yields the model with the best

balance of validation accuracy (88%) and complexity. For experiments using Inception V3 feature maps, the model architecture of one layer with 256 hidden units and 50% dropout yields the model with the best balance of validation accuracy (93%) and complexity.

Although there is no class imbalance, additional evaluation metrics such as precision, recall, and f1-score are still of interest. In this particular case, it may be beneficial to prioritize precision over recall as it would likely yield the better user experience to be very sure a recipe image is of food. Also, the consequence of a false negative is rather trivial as it means the recipe with a falsely labeled non-food image is simply demoted in the search results. Prioritizing precision over recall also has the added benefit of potentially weeding out unappealing food images as well. Often unappealing food images aren't even immediately recognizable as food.

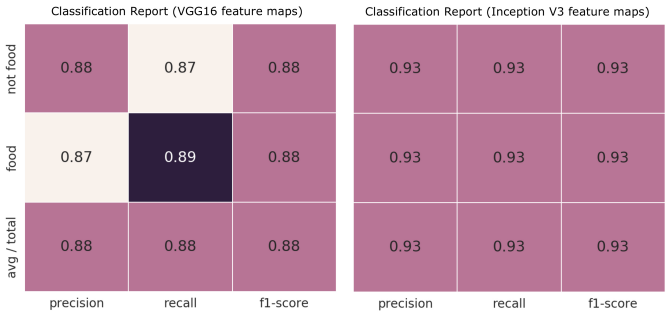

Figure 7. Classification reports — VGG16 feature maps (right), Inception V3 feature maps (left)

Another valuable evaluation metric is a Receiver Operating Characteristic (ROC) curve where the true positive rate is plotted as a function of the false positive rate. An Area Under the Curve (AUC) of up and to the left is good — as the threshold decreases not many more false positives are being accrued while the true positives increase (Figure 11). This demonstrates the model is good a separating the food and non-food images.

C. *Qualitative Analysis*

In addition to quantitative metrics, it is also useful to analyze the results qualitatively by visually examining a random sample of the images the models classified as food or non-food images. See Figures 9–14 for examples of images the models correctly labeled, incorrectly labeled, and images the model was confused about.

Classification using VGG16 feature maps

Classification using Inception V3 feature maps

Figure 9. Non-food images the model correctly and very assuredly classifies as not food (with a raw probability of 0)

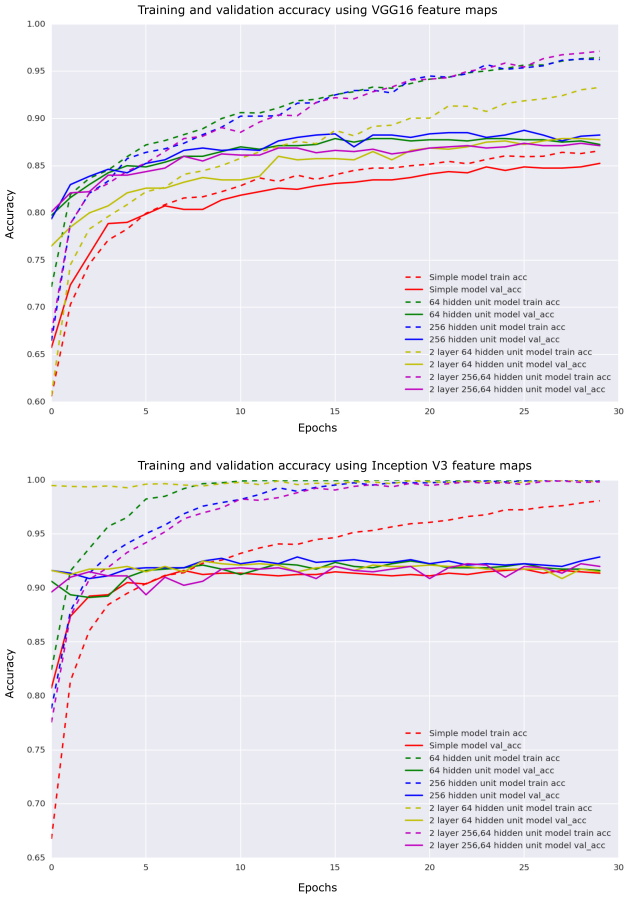

Figure 8. Learning curves — VGG16 feature maps (top), Inception V3 (bottom)

Classification using VGG16 feature maps

Classification using Inception V3 feature maps

Figure 10. Food images the model correctly and very assuredly classifies as food (with a raw probability of 1)

VII. CONCLUSIONS & FUTURE WORK

All in all, the models performed decently well but not without shortcomings — namely overfitting and noisy data. Overfitting is quite high for some of the model explorations which could be reduced if data augmentation was

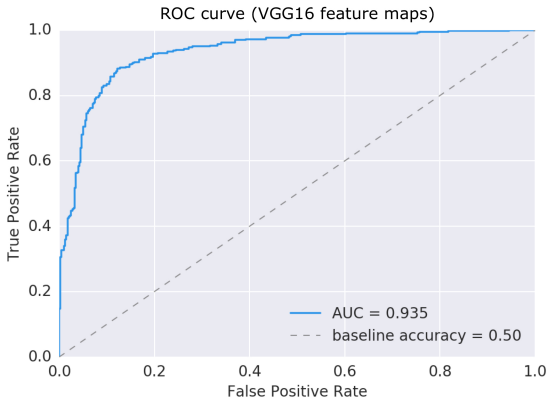

Figure 11. ROC curves

employed. Leveraging data augmentation necessitates extending the convolutional base and connecting it to a final classifier and running the whole architecture end to end which is very compute intensive and requires GPU. However, an additional benefit of an end to end implementation is it allows fine-tuning — training the top convolutional layers of the models which would allow the models to learn more high-level features specific to the dataset. Data augmentation, fine-tuning, recollecting data to be less noisy, as well as adjusting the learning rate, are techniques to explore in the future to enhance the performance of these models.

Classification using VGG16 feature maps

Classification using Inception V3 feature maps

Figure 12. Food images model incorrectly but still very assuredly classifies as not food (with a raw probability near or at 0). *Note several of the images have people as the main focus.*

Classification using VGG16 feature maps

Classification using Inception V3 feature maps

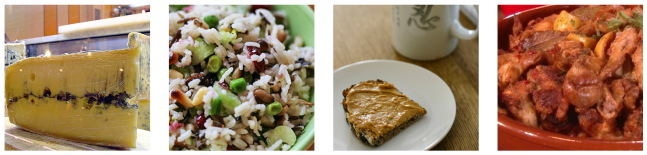

Figure 13. Non-food images the model incorrectly but still very assuredly classifies as food (with a raw probability near or at 1). *Note these images are indeed labeled as not food, when they are in fact clearly food images. This was a shortcoming of the data collection process, as some food images were unavoidably randomly sampled from ImageNet to build the not food training set. On the bright side, the model does actually correctly label them as food images, although not "officially".*

Classification using VGG16 feature maps

Classification using Inception V3 feature maps

Figure 14. Images the model is unsure about (as the raw probabilities are near or at the 0.5 class boundary threshold).

REFERENCES

[1] Nanni, Loris, Ghidoni, Stefano and Brahnam, Sheryl, “Handcrafted vs Non-Handcrafted Features for computer vision classification,” Pattern Recognition, 2017. DOI: 10.1016/j.patcog.2017.05.025

[2] CS231n: Convolutional Neural Networks for Visual Recognition, “Linear classification: Support Vector Machine, Softmax,” 2017, [Online], Available: http://cs231n.github.io/linear-classify, [Accessed: 14 - Oct - 2017].

[3] CS231n: Convolutional Neural Networks for Visual Recognition, “Neural Networks Part 1: Setting up the Architecture,” 2017, [Online], Available: http://cs231n.github.io/neural-networks-1, [Accessed: 14 - Oct - 2017].

[4] CS231n: Convolutional Neural Networks for Visual Recognition, “Backpropagation, Intuitions,” 2017, [Online], Available: http://cs231n.github.io/convolutional-networks, [Accessed: 14 - Oct - 2017].

[5] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov 1998. DOI: 10.1109/5.726791

[6] CS231n: Convolutional Neural Networks for Visual Recognition, “Convolutional Neural Networks: Architectures, Convolution / Pooling Layers,” 2017, [Online], Available: http://cs231n.github.io/optimization-2, [Accessed: 14 - Oct - 2017].

[7] CS231n: Convolutional Neural Networks for Visual Recognition, “Transfer Learning and Fine-tuning Convolutional Neural Networks,” 2017, [Online], Available: http://cs231n.github.io/transfer-learning, [Accessed: 14 - Oct - 2017].

[8] S. Wong, A. Gatt, V. Stamatescu and M. McDonnell, “Understanding data augmentation for classification: when to warp?” Computer Science - Computer Vision and Pattern Recognition, I.5.2, I.4.7, Sept 2016.

[9] ImageNet, 2016. [Online]. Available: http://www.image-net.org. [Accessed: 14 - Oct - 2017].

[10] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, CoRR, abs/1409.1556, 2014.

[11] D. Frossard, “VGG in TensorFlow,” 2016. [Online]. Available: http://https://www.cs.toronto.edu/~frossard/post/vgg16. [Accessed: 14 - Oct - 2017].

[12] J. Shlens, “Train your own image classifier with Inception in TensorFlow,” 2016. [Online]. Available: https://research.googleblog.com/2016/03/train-your-own-image-classifier-with. [Accessed: 14 - Oct - 2017].

[13] A. Singla, L. Yuan, and T. Ebrahimi, “Food/Non-food Image Classification and Food Categorization using Pre-Trained GoogLeNet Model,” In Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management, March 2016.

[14] L.Yang, C.Hsieh, H. Yang, N. Dell, S. Belongie, C. Cole, D. Estrin, “Yum-me: A Personalized Nutrient-based Meal Recommender System,” ARXIV, May 2016.