Deep Dish : Deep Learning for Classifying Food Dishes

Abhishek Goswami
Microsoft
Redmond, WA
agoswami@microsoft.com

Haichen Liu
Dropbox
Seattle, WA
haichen@dropbox.com

Abstract

We consider the problem of classifying food dishes. Food items have unique characteristics - they come in different colors and shapes, can be clustered into groups (e.g. fruits, vegetables), and can be combined in several ways to prepare a meal etc. This makes images of food dishes particularly interesting to classify. We show that convolutional neural networks are quite suitable for this task, and outperform traditional machine learning approaches in classifying food dishes.

1. Introduction

This project aims to use deep learning on images of food dishes. Food images are unique: there are multiple cuisines around the world; food items have unique color, size, shape and texture; and food items can be combined in several ways to prepare a meal. Using artificial intelligence on food images has the potential to revolutionize the field of dining, promote healthy eating, prevent food waste etc.

To that end, we are working on the problem of classifying food dishes. We formulate this problem as a classification task with one class per image, i.e given an image of a food dish, we want to correctly predict what dish it is. Figure 1 shows sample images of two popular food categories. Being able to accurately predict a food category from an image could be useful for several application scenarios, such as knowing the calorie count for that food item, identifying its ingredients etc.

The remainder of the paper is organized as follows. In Section 2 we survey related work in the area of image classification. In Section 3 we introduce the key components used in image classification tasks. In Section 4 we provide details about our dataset. Section 5 presents the experimental results from our modeling techniques. Finally, we present our conclusions in Section 6.

(a) burger (b) pizza

Figure 1: Sample images in our dataset

2. Related Work

Deep Convolutional Neural Networks have been shown to be very useful for visual recognition tasks. AlexNet [17] won the ImageNet Large Scale Visual Recognition Challenge [22] in 2012, spurring a lot of interest in using deep learning to solve challenging problems. Since then, deep learning has been used successfully in multiple fields like machine vision, facial recognition, voice recognition, natural language processing etc.

There are several flavors of image classification tasks that have been been proposed over the years. These range from being able to recognize hadwritten digits [18] to classifying plants using the images of leaves [7].

Our problem of classifying food dishes is unique. We did not find any existing work focussed on classifying food dishes from images. The closest example of a food-related task we could find was a *restaurant classification* problem from Yelp [8]. In their scenario, they want to classify the images of *restaurants* along some business attributes (e.g restaurant is kid friendly, has table service etc). Classifying food dishes from images presents several distinct characteristics that we discuss in Sections 4 and 5

3. Methods

Image classification is the task of assigning a single label to an image (or rather an array of pixels that represents an

image) from a fixed set of categories. A complete pipeline for this task is as follows:

- **Input** : A set of N images, each labeled with one of K different classes. This data is referred to as the *training set*.
- **Learning** (aka Training) : Use the *training set* to learn the characteristics of each class. The output of this step is a *model* which will be used for making predicions.
- **Evaluation** : Evaluate the quality of the *model* by asking it to make predictions on a new set of images that it has not seen before (also referred to as the *test set*). This evaluation is done by comparing the true labels (aka ground truth) of the *test set* with the predicted labels output by the learned *model*.

The formal approach for solving the problem of image classification can be broken down into several key components which we discuss next.

3.1. Score Function

The score function maps the raw data to class scores. For a linear classifier, the score function can be defined as:

$$f(\boldsymbol{x_i}, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\boldsymbol{x_i} + \boldsymbol{b}. \tag{1}$$

where $\boldsymbol{x_i}$ represents the input image. The matrix $\boldsymbol{W}$, and the vector $\boldsymbol{b}$ are the parameters of the function, and represent the weights and bias respectively.

In image classification, the score function takes an image $\boldsymbol{x_i}$ and computes the vector $f(\boldsymbol{x_i}, \boldsymbol{W})$ of the raw class scores (which we abbreviate as $\boldsymbol{s}$). So, given an image $\boldsymbol{x_i}$, the predicted score for the j-th class is the j-th element in $\boldsymbol{s}$: $\boldsymbol{s}_j = f(\boldsymbol{x_i}, \boldsymbol{W})_j$. We use the class scores from our training data to compute the loss.

3.2. Loss Function

The loss function quantifies the match between the predicted *scores* and the ground truth labels in the training data. The loss function (also referred to as the cost function or objective) can be viewed as the unhappiness of the predicted scores output by the score function. Intuitively, the loss would be low if the predicted scores match the training data labels closely. Otherwise the loss would be high. Next, we discuss the two common classifiers with details about their respective loss functions.

3.3. Classifiers

In this section we discuss two common classifiers that are often used in image classification tasks: the **SVM Classifier** and the **Softmax Classifier**. For both of them the function mapping the input image $\boldsymbol{x_i}$ to the raw class scores $\boldsymbol{s} = f(\boldsymbol{x_i}, \boldsymbol{W})$ remains the same. But the Softmax classifier has one additional step : it uses the softmax function to squash the raw scores in $\boldsymbol{s}$ into a vector of values between zero and one, that sum to one. We discuss the details of each classifier below.

3.3.1 SVM Classifier

The SVM classifier uses the *hinge loss* (also referred to as *max-margin loss*, or *SVM loss*). For the i-th example in our data, the *hinge loss* is given as:

$$L_i = \sum_{j \neq y_i} \max(0, \boldsymbol{s}_j - \boldsymbol{s}_{y_i} + \Delta). \tag{2}$$

where Δ is a hyperparameter which represents that the SVM loss function in equation 2 wants the score of the correct class $\boldsymbol{y_i}$ to be larger than the incorrect class scores by at least Δ. Otherwise we incur loss.

3.3.2 Softmax Classifier

The Softmax classifier uses the *cross entropy loss* (also referred to as *softmax loss*). For the i-th example in our data, the *cross entropy loss* is given as:

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}. \tag{3}$$

where f_j means the j-th element of the vector of class scores f. Note that the softmax classifier uses the softmax function to squash the raw class scores $\boldsymbol{s}$ into normalized positive values that sum to one, so that the cross entropy loss can be applied. The **softmax function** can be represented as:

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}. \tag{4}$$

It takes a vector of real-valued scores (in z) and squashes it to a vector of values between zero and one, that sum to one.

3.4. Total Loss

For both the SVM Classifier and the Softmax Classifier, the full loss for the dataset is the mean of L_i over all training examples, together with a regularization term, $R(W)$

$$L = \frac{\sum_i L_i}{N} + \lambda R(W). \tag{5}$$

where N represents the total number of images in the training set. λ is a hyperparameter, often referred to as *regularization strength*. The loss function lets us quantify the quality of any particular set of parameters in our model, the lower the loss the better. We next discuss strategies of how to minimize the loss.

3.5. Optimization

Optimization is the process of finding the set of parameters of our model that minimize the total loss, defined in equation 5

The core principle behind optimization techniques is to compute the **gradient** of the loss with respect to the parameters of the model. The gradient of a function gives the direction of steepest ascent. One way of computing the gradient efficiently is to compute the gradient analytically using a recursive application of the **chain rule**. This technique is called **backpropagation** [19] and it allows us to efficiently optimize arbitrary loss functions. These loss functions may be expressing different kinds of network architectures (e.g. fully connected neural networks, convolutional networks etc). Backpropagation is our tool of choice for computing the gradients in all such cases.

3.5.1 Parameter Updates

Once the analytic gradient is computed using backpropagation, the gradients are used to perform a parameter update. There are several approaches for performing the update that have been proposed in literature: SGD [10], SGD+Momentum [21, 25], Nesterov Momentum [20], Adagrad [11], RMSprop [13], Adam [16] etc.

4. Dataset and Features

We start with a discussion about our data collection methodology. We then present details about the data preprocessing steps. Finally we round up this section with details about our dataset.

We provide some details about our dataset below.

4.1. Data Collection

We collected our dataset using the Google Image Search [5] and the Bing Image Search API [1]. We also explored the use of ImageNet [6] and Flickr [3] for collecting images. However, we found the images from Google and Bing to be much more representative of the classes they belonged to, compared to the images from ImageNet and Flickr. ImageNet and Flickr seem to have a lot of spurious images (images which clearly do not belong to the class). Hence we decided to use the images we could collect from Google and Bing.

4.2. Pre-Processing Steps

We re-sized all of our images to have height, width and channel dimensions of 32, 32 and 3 respectively. This was done primarily for computational efficiency in performing our experiments. We filtered out images which we were unable to resize to our specified height, width and channel requirements. Unfortunately, this meant losing approx 10% of the data from our original dataset. Figure 1 shows a two sample images from our dataset. As a part of preprocessing, we also subtract the mean image from all the images in our dataset. The mean image is computed using the image mean of the training data

Dataset	Num of Images
Train	18,927
Validate	5,375
Test	2,682

Table 1: Dataset split for train, validation and test sets.

Food Item	Number of Images
dumplings	1,091
dal	1,031
ramen	1,023
icecream	1,021
naan	1,020
sushi	1,020
cordonbleu	1,018
pasta	1,002
lasagna	971
friedrice	966
roastturkey	930
padthai	919
burger	912
samosa	897
burrito	888
pizza	885
bratwurst	876
biryani	865
sandwich	847
fries	745

Table 2: Class distribution

4.3. Dataset Details

After the pre-processing steps described in Section 4.2 we had a total of 26,984 images. We then split our dataset randomly into 3 disjoint sets: Train(70% approx.), Validate(20% approx.) and Test(10% approx.). Table 1 provides a count of the number of images in each set.

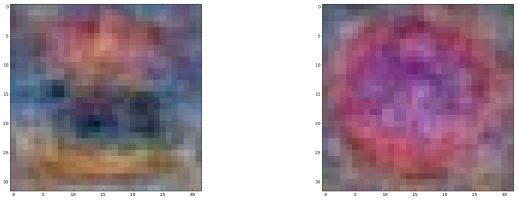
Currently our dataset has 20 classes. This corresponds to 20 popular food dishes from around the world. Table 2 shows the class label distribution of the dataset. The distribution of the number of images in each class is mostly uniform.

5. Evaluation Results

In this section we discuss our experiments and results. We chose accuracy as our evaluation metric when comparing different models. For brevity, we are reporting the accuracy numbers to two decimal places.

For Section 5.1, Section 5.2 and Section 5.3 we repurposed code from assignments 1 and 2 in Stanford University's Spring 2017 course, CS231N: Convolutional Neural Networks for Visual Recognition [2]. For Section 5.4 and Section 5.5, we use TensorFlow [9] for training our convolutional network models.

5.1. Linear classifiers on raw image pixels



(a) burger (b) pizza

Figure 2: Visualizing the weights learned by the SVM model

To set our baseline, we first use a linear classifier using raw image pixels as features. For this we tried out both a SVM classifier and a Softmax classifier. The best validation accuracy of 0.18 was achieved using the SVM classifier with a learning rate 1e-07 and regularization strength 2.5e+04. The corresponding test set accuracy was 0.16.

One interpretation of a linear classifier is that of a *template match*, where each row of the learned weights matrix corresponds to a template for the corresponding class. Figure 2 shows the learned weights for the burger and pizza classes in our dataset. We note that both the templates match our intuition; the burger contains a lot of brown pixels, the pizza has a round shape and contains a lot of red pixels at the center.

5.2. Neural networks on raw image pixels

The next set of models we tried out were fully connected neural networks, again using raw image pixels as features. Our network architecture was a six layer fully-connected network. Each of the five hidden layers had 100 neurons each. We used ReLU nonlinearity, and a softmax loss function. Below we note some interesting observations from training these models.

- Batch normalization was [15] **very useful** in training our model. Without batch normalization our model was performing quite poorly.
- The best validation accuracy of 0.19 was achieved using the **Adam** [16] update rule with a learning rate of 1e-03. The test set accuracy was 0.18

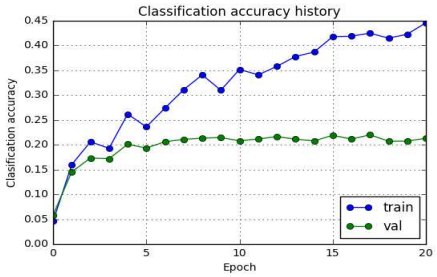


Figure 3: Classification accuracy history of a fully connected five layer neural network using raw image pixels

Figure 3 shows the classification loss history for the training and validation sets over 20 epochs while training this network.

5.3. Image features

We did a set of experiments using features extracted from the images. For featurizing each image, we compute a Histogram of Oriented Gradients (HOG) as well as a color histogram using the hue channel in HSV color space. We form our final feature vector for each image by concatenating the HOG and color histogram feature vectors. This gives us a total of 155 features for each image. Below we summarize the results using image features with a SVM and a Two Layer Fully Connected Neural Network classifier.

- Using the images features with a linear SVM classifier we were able to get a validation accuracy of 0.21, using SGD with a learning rate of 1e-03 and regularization strength of 1e+00
- Using the images features with a Two Layer Fully Connected Neural Network gave much better performance. We got the best validation accuracy of 0.26 while using SGD as our update rule with a learning rate of 0.9, learning rate decay of 0.8 and regularization strength 0. The corresponding test set accuracy was 0.27

Figure 5 shows the classification loss history for the training and validation sets over 10 epochs while training the Two Layer Fully Connected Neural Network classifier with image features.

5.4. Convolutional Networks

Using Convolutional Networks we were able to get the validation and test set accuracy of 0.40 each. Figure 4 shows

Modeling Approach	Best Validation Accuracy	Test Set Accuracy
Linear SVM on raw image pixels	0.18	0.16
Five layer fully connected neural net on raw image pixels	0.19	0.18
Linear SVM on image features	0.21	0.22
Two layer neural net on image features	0.26	0.27
Training a Convolutional Network from scratch	0.40	0.40
Transfer Learning (by fine tuning a VGG model)	0.46	0.45

Table 3: Summary of results across different modeling approaches.

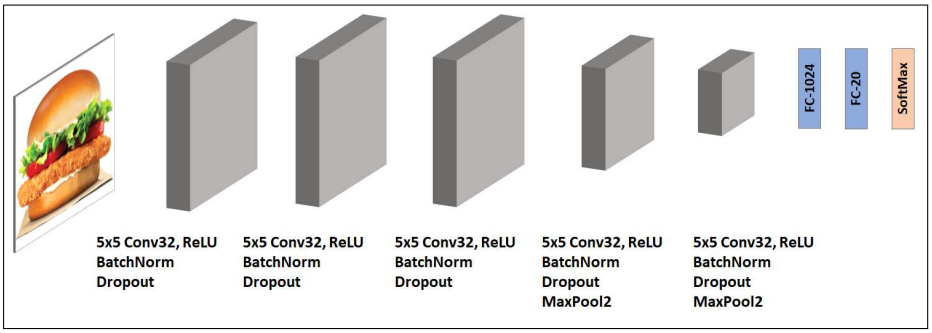


Figure 4: Convolutional network architecture

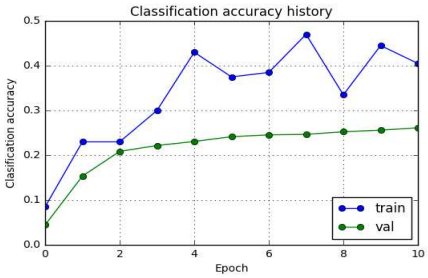


Figure 5: Classification accuracy history of a fully connected two layer neural network using image features

the architecture we used. Below we note some of the things we tried out.

- Batch normalization [15] was quite useful in training our model.
- For the weights in our network, using Xavier initialization [12] helped.
- Dropout [14, 24] (with keep probability 0.75) helped improve the validation accuracy from 0.38 to 0.40.
- We kept the number of filters fixed at 32.
- We tried different sized filters (3x3, 5x5 and 7x7) but they did not help much. So we fixed the filter size at 5x5.
- For the first three conv layers we preserve the height and width dimensions. For the fourth and fifth conv layers, we used max pooling with stride 2 (across both height and width).
- After the five conv layers, we added two fully connected layers with 1024 and 20 neurons respectively.
- For the last layer we use softmax with cross entropy loss.
- The best validation accuracy of 0.40 was achieved using the **Adam** [16] update rule with a learning rate of 1e-04. The test set accuracy was 0.40.

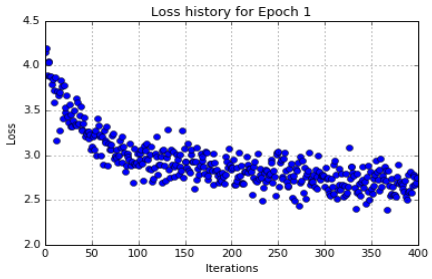


Figure 6: Reduction in loss over several mini-batches in the first epoch of the convolutional network

Figure 6 shows the reduction in the loss over multiple iterations in the first epoch. We see the loss reduces very sharply in the beginning, and then flattens out gradually.

Figure 7 shows the classification loss history for the training and validation sets over 25 epochs of the conv net.

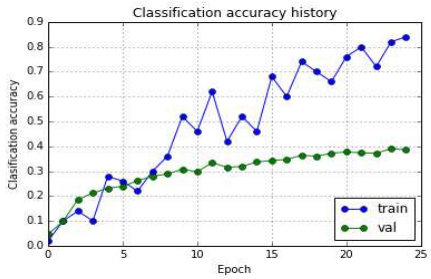


Figure 7: Classification accuracy history of the convolutional network over 25 epochs

5.5. Transfer Learning

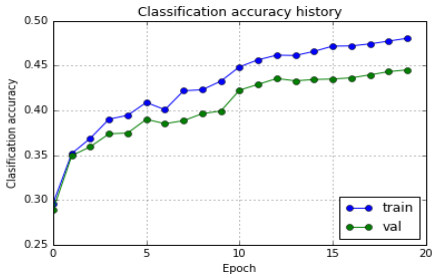


Figure 8: Classification accuracy history after fine-tuning a VGG model

To improve the accuracy of our model further, we did a set of experiments around transfer learning. Interestingly this gave us the best results on our dataset. Some salient observations from this approach are as follows:

- We are using the VGG-16 [23] model pretrained on ImageNet
- We remove the last fully connected layer (fc8) and replace it with our own, with output size 20
- We first train the last layer for 10 epochs. This allows us to get meaningful weights for the fc8 layer first. Subsequently, we train the entire model on our dataset for 10 more epochs.

For this approach, we referenced the TensorFlow finetune sample on GitHubGist [4]. Following the example in the gist, we did similar pre-processing on our dataset to make it work for the VGG-16 model. The pre-processing steps are listed below:

- Resize the image so its smaller side is 256 pixels long. Recall that our existing dataset has dimensions (32, 32, 3).
- Take a random 224x224 crop of the scaled image (for the train, validation and test sets)
- Horizontally flip the image with probability 1/2 (for the train set only)
- Substract the per color mean VGG_MEAN [123.68, 116.78, 103.94) (for the train, validation and test sets)

Figure 8 shows the classification loss history for the training and validation sets over all the 20 epochs. Table 3 shows a summary of results across different modeling approaches.

6. Conclusion

We observe that convolutional neural networks are quite suitable for the task of classifying food dishes, and outperform traditional machine learning approaches at this task. The transfer learning approach looks most promising, especially because both the training and validation accuracy are improving with the number of epochs (i.e. we have not overfit our model). This suggests that more data (and/or running it for more epochs) could improve the accuracy metric further.

From a data collection perspective, we plan on leveraging ImageNet [6] and Flickr [3] to build a larger dataset of images. From a modeling perspective, we also want to try out using convolutional nets as a *fixed feature extractor*, and use the extracted features with linear classifiers or decision trees to improve accuracy.

There are several interesting problems around food images that we wish to investigate in the future. This includes being able to detect individual food items on plate, accurately predicting the number of calories given an image of a food dish etc. Convolutional networks seem to be a natural fit for these visual recognition tasks.

References

[1] Bing image search api. https://azure.microsoft.com/en-us/services/cognitive-services/bing-image-search-api/.
[2] Cs231n: Convolutional neural networks for visual recognition. http://cs231n.stanford.edu/index.html.
[3] Flickr. https://www.flickr.com/.
[4] Github. tensorflow finetune gist. https://gist.github.com/omoindrot.
[5] Google image search. https://images.google.com/.
[6] Image-net. http://www.image-net.org/.
[7] Kaggle. leaf classification. https://www.kaggle.com/c/leaf-classification.
[8] Kaggle. yelp restaurant photo classification. https://www.kaggle.com/c/yelp-restaurant-photo-classification.
[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.

[10] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[11] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[12] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[13] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini–batch gradient descent. 2012.

[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[16] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[18] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[19] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[20] Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/k2). In *Doklady an SSSR*, volume 269, pages 543–547, 1983.

[21] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[25] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.