Bun with CNNs - A Sandwich Image Classifier

Yihan Wang
https://github.com/yihanw91/sandwich-classification

I. Abstract

My goal was to find the best model for classifying images into one of 6 types of sandwiches. I tried CNN architectures of varying complexity and also experimented with fine-tuning, learning rates, and data augmentation. My best model had a ResNet34 architecture and was trained using discriminative fine-tuning and a 1-cycle triangular learning rate. It achieved a test set accuracy of 0.875.

In the context of this problem, I discovered that transfer learning worked well and a simpler architecture performed the best. However, data augmentation for mitigating variations in illumination, namely the Grey World and Histogram Equalization techniques, did not end up helping model performance.

II. Description of Problem and Dataset

My overarching goal was to find the best model for this particular image classification problem by trying different CNN architectures and deep learning methods. Specifically, my task was classifying sandwich images into one of 6 types: club sandwich, grilled cheese, hamburger, hot dog, lobster roll, or pulled pork sandwich.

I used “The Food-101 Data Set”, which was originally used in a 2014 paper [1]. It contains 101 food categories that each has 1,000 associated images. For each category, the authors hand-selected 750 images to be the training set and 250 to be the test set, and I used the exact same split to evaluate my final performance. This was a unique and challenging dataset because “on purpose, the training images were not cleaned, and thus still contain some amount of noise. This comes mostly in the form of intense colors and sometimes wrong labels”.

Since I used 6 of the 101 categories, I ended up with 4,500 training images and 1,500 test images. To compare models and decide on the best one, I further split the training images into fixed training (80%) and validation (20%) sets. I used classification accuracy as my evaluation metric because I had balanced classes. In the original paper, the authors achieved an accuracy of 0.564 using AlexNet. Since I classified images into 6 classes instead of 101, I expected my model to outperform that benchmark.

III. Solution Sources

My main solution source was content covered in class. I applied concepts around data augmentation, fine-tuning, and learning rates and adapted the corresponding code [2] to this use

[1] https://www.vision.ee.ethz.ch/datasets_extra/food-101/
[2] https://github.com/yanneta/deep-learning-with-pytorch

case. In addition, I implemented CNN architectures not covered in class (InceptionV3, InceptionResNetV2, and DenseNet-121) and extended the in-class code to those architectures. For InceptionV3, for example, I modified the code to handle its Auxiliary Logits. To fine-tune each architecture, I wrote code to unfreeze parameters and also created different groups of layers to unfreeze.

I also applied additional image processing techniques, namely Grey World and Histogram Equalization. The idea came from a paper written by Stanford students who worked on a similar project [3]. However, I implemented those techniques by adapting code that I found [4] [5] into my workflow as part of image preprocessing or data augmentation.

IV. Machine Learning Methods

For each architecture, I began with a model that was pre-trained on ImageNet and replaced the last fully-connected layer with the appropriate linear classifier. I froze all parameters except those of the final (classifier) layer and trained the model with a 0.01 learning rate (LR) for 10 epochs. These models were my baseline and I used their initial validation accuracies below as benchmarks.

Architecture	Validation Accuracy
ResNet34 (10 epochs)	0.768
InceptionV3 (10 epochs)	0.672
InceptionResNetV2 (10 epochs)	0.708
DenseNet-121 (10 epochs)	0.750

Figure 1: Comparison of initial baseline models

I then fine-tuned these models by unfreezing some earlier layers and training at a lower learning rate. I was interested in the relationship between the degree of fine-tuning and predictive performance, so for each architecture, I experimented with unfreezing different numbers of earlier layers.

For ResNet34 and InceptionResNetV2, I also applied discriminative fine-tuning with a 1-cycle triangular learning rate in several steps. I first used the learning rate range test to identify the optimal maximum learning rate. I then used that maximum to define the cosine triangular learning rate over the training iterations. Next, I created groups of layers that I

[3] http://cs229.stanford.edu/proj2016/report/YuMaoWang-Deep%20Learning%20Based%20Food%20Recognition-report.pdf
[4] https://pypi.org/project/colorcorrect/
[5] https://stackoverflow.com/questions/42651595/histogram-equalization-python-for-colored-image

gradually unfroze over the course of training and trained the earlier groups with a lower learning rate.

V. Additional Image Augmentation

One of the challenges of working with this dataset is that some images have intense colors. “In general, the distribution of color values in an image depends on the illumination, which may vary depending on lighting conditions, cameras, and other factors” [6]. To mitigate these variations, I employed two color normalization methods: Histogram Equalization and Grey World. At a high level, Histogram Equalization is used for adjusting image intensities to enhance contrast [7] and Grey World is used for color balancing [8] (see Figure 4 in the Appendix for a visual example).

I experimented with Histogram Equalization and/or Grey World, both during the image preprocessing step and as a random data augmentation. Afterward, I applied various ResNet34 models on the modified images.

VI. Experimental Results

The table below details the best model that I found for each architecture:

Architecture	Best Model Details	Validation Accuracy
ResNet34 (30 epochs)	Discriminative fine-tuning with 1-cycle triangular LR: 1. Train the final layer 2. Unfreeze the 2 earlier layers after 10% of iterations and train with lower LR	0.851
InceptionV3 (30 epochs)	1. Train the final layer with 0.01 LR for 10 epochs 2. Unfreeze the 2 earlier layers and train with 0.001 LR for 20 epochs	0.692
InceptionResNetV2 (30 epochs)	Discriminative fine-tuning with 1-cycle triangular LR: 1. Train the final layer 2. Unfreeze the 4 earlier layers after 10% of iterations and train them lower LR	0.846
DenseNet-121 (30 epochs)	1. Train the final layer with 0.01 LR for 10 epochs 2. Unfreeze the 2 earlier “layers” (denseblock4 and norm5) and train with 0.001 LR for 20 epochs	0.849

Figure 2: Comparison of best models

[6] https://en.wikipedia.org/wiki/Color_normalization
[7] https://www.math.uci.edu/~icamp/courses/math77c/demos/hist_eq.pdf
[8] https://web.stanford.edu/~sujason/ColorBalancing/grayworld.html

For each architecture, unfreezing the few layers immediately preceding the final (classifier) layer boosted accuracy. For ResNet34, I tried discriminative fine-tuning that involved eventually unfreezing all layers, but even after training for 100 epochs, it still performed worse than just unfreezing the top few layers.

Fine-tuning the entire network did not improve performance because my training set was too small (only 600 images per class) for avoiding overfitting, as evidenced by the growing divergence between training and validation loss over 100 epochs. Moreover, my dataset is already somewhat similar to ImageNet. The image sizes are similar and ImageNet actually has 42 food classes, including hamburger and hot dog [9]. Therefore it makes sense that transfer learning worked well and heavy fine-tuning was not beneficial.

The table below details the best Resnet34 model trained on images that have been processed with Grey World (GW) and/or Histogram Equalization (HE). One possible explanation for why these techniques actually slightly hurt performance is that the resulting images became more unlike the images from ImageNet, and this, in turn, made transfer learning less effective.

Details	Validation Accuracy
Images without augmentation	0.851
Images preprocessed with HE	0.816
Images preprocessed with GW	0.836
Images preprocessed with HE + GW	0.822
GW + HE as a random data augmentation	0.838

Figure 3: Comparison of best ResNet34 models

VII. Final Model Evaluation

On the basis of validation accuracy, I chose the best ResNet34 as my final model and evaluated it on the test set. It achieved an accuracy of 0.861 without Testing Time Augmentation (TTA) and 0.875 with TTA, exceeding my original minimum baseline of 0.564.

In the resulting confusion matrix (see Figure 5 in the Appendix), two things jumped out and I manually scanned the test images to come up with possible explanations. First, my model was the most confused between grilled cheese and club sandwiches. This is probably because both are triangular in shape and the images did not contain other strong signals to distinguish between them. Second, my model made a lot of incorrect predictions on true hamburgers. This could be because many images of different sandwich types also contained French fries. If the

[9] https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

training images had bounding boxes around the actual sandwiches, then this noisy factor could have been mitigated.

VIII. Conclusion and Key Lessons Learned

First, I discovered that transfer learning worked well in this scenario. This was because my dataset was small and similar to the original ImageNet dataset that produced the pre-trained model weights. Beyond unfreezing the few layers immediately preceding the final (classifier) layer, any more fine-tuning did not improve performance.

Second, a simpler architecture performs just as well as a more complex one in some cases, as evidenced by my best ResNet34 model outperforming my best InceptionResNetV2. A more complex architecture can easily overfit the training data. At the end of training, the aforementioned ResNet34 had a validation loss of 0.590 and a training loss 0.086 (difference of 0.504), compared to 1.307 and 0.016 (difference of 1.291) for InceptionResNetV2, respectively. My dataset was small enough that overfitting was a problem for the more complex InceptionResNetV2.

Third, image augmentation for mitigating variations in illumination was not actually beneficial for this problem. This suggests that differences in lighting conditions, cameras, and other factors did not prove to be an issue here. The bigger issue was probably noisy extraneous items (e.g. side dishes) in the images, which would have been helped by bounding boxes.

IX. Team Member Responsibilities

- Processed the image data: train-test split, resized images, and wrote DataSet.
- Applied pre-trained CNN models with ResNet34, InceptionV3, InceptionResNetV2, and DenseNet-121 architectures.
- For each architecture, fine-tuned models and experimented with learning rates.
- Performed additional image augmentation (Grey World and Histogram Equalization) and fitted ResNet34 models on the resulting data.

X. Appendix

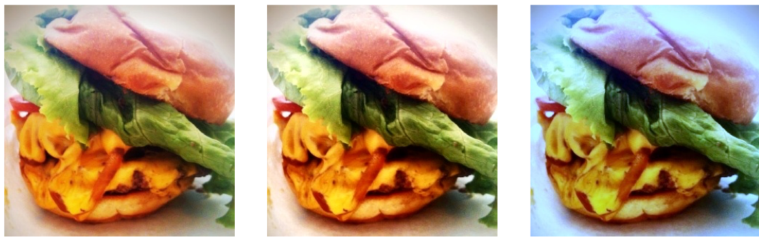
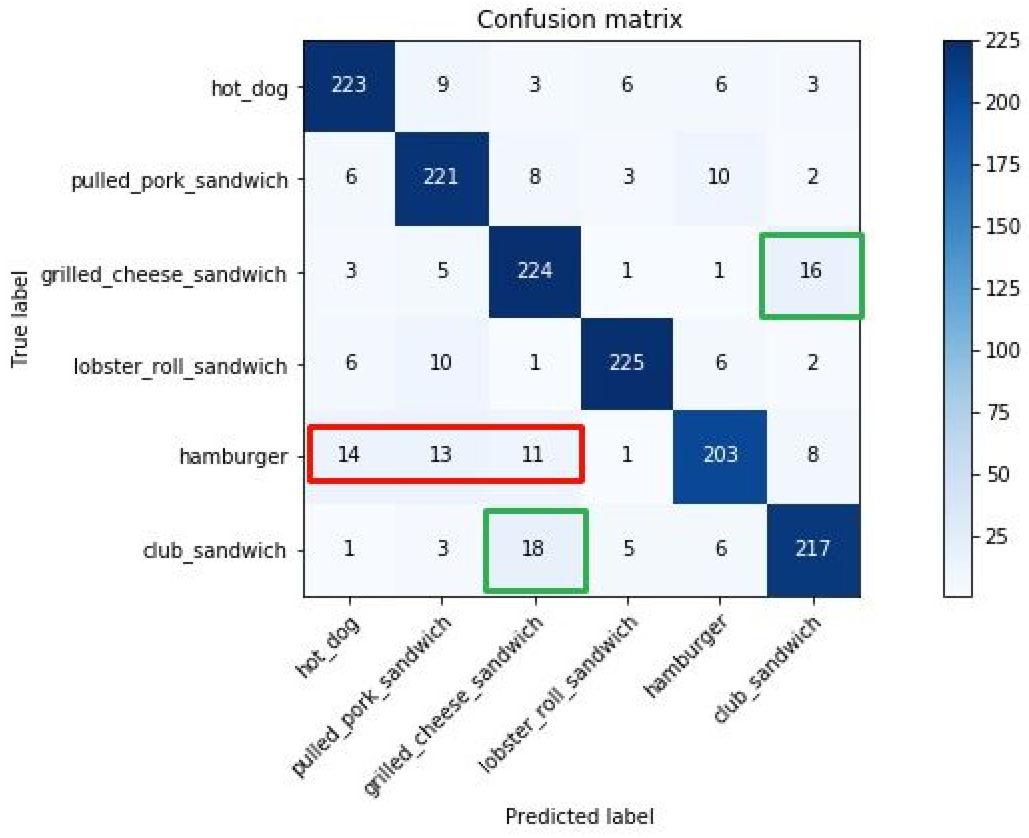


Figure 4: original image (left), Histogram Equalization applied to the left image (middle),

Grey World applied to the middle image (right)



	hot_dog	pulled_pork_sandwich	grilled_cheese_sandwich	lobster_roll_sandwich	hamburger	club_sandwich
hot_dog	223	9	3	6	6	3
pulled_pork_sandwich	6	221	8	3	10	2
grilled_cheese_sandwich	3	5	224	1	1	16
lobster_roll_sandwich	6	10	1	225	6	2
hamburger	14	13	11	1	203	8
club_sandwich	1	3	18	5	6	217

Figure 5: Confusion matrix from test set predictions