Transfer Learning for State Recognition of Food Images

Christopher Collazo

***Abstract*— Using transfer learning on VGG-19 pretrained with ImageNet images, we extend it with additional layers and hyperparameter tuning to achieve high accuracy on state recognition of images of food at various stages of cooking, from raw to diced to mashed to cooked, converging after a small number of epochs.**

I. INTRODUCTION

State recognition is an important problem to solve in order for deep neural networks to apply to more general problems in the real world. Objects can exist in states that are not necessarily recognizable in any way as the same base object. The scenario tested in this study is a perfect example: During the cooking process, foods such as bread, vegetables, meats, etc. can exist in states which present enormous variations in size, color, and shape. These would confuse the training process for an object recognition deep learning algorithm by introducing untenable fluctuations in inputs for a single class, forcing weights to diverge from their intended target; this results in lower accuracy and, ultimately, useless classifiers.

One solution to this problem would be to simply discretize the states of objects as separate objects entirely. Thus, a "chopped onion" would be a distinct object class from a "whole onion". However, in order to approach true understanding of the world, we would like for a deep learning algorithm to recognize that different states still belong to the same object class; in other words, we would like to recognize the difference between the states "chopped" and "whole", but also recognize that both are of the object class "onion".

This study presents a deep learning architecture designed to recognize differences in states, but not objects, as a stepping stone to that combined classification. Thus, we are simply attempting to classify objects of any type as "chopped", "diced", "flattened", "whole", etc.

II. DATA, PREPROCESSING, AND COMMON FEATURES

A. Inputs

Our dataset consists of images of food in various states which would manifest during cooking. While the foods vary enormously, they all fall into 11 distinct states: "creamy_paste", "diced", "floured", "grated", "juiced", "julienne", "mixed", "other", "peeled", "sliced", and "whole".

The dataset consists of 6348 training samples and 1377 validation samples, for a total of 7725 images. This results in a roughly 82% training, 18% validation split. Normalization of 8-bit pixel values in the range [0, 255] clamps them to the range [0.0, 1.0]. These images vary in size, and so are normalized with a resize to 224x224, and augmented randomly with a zoom of [-0.2, 0.2], shear of [-0.2, 0.2], and/or a horizontal flip. That is, the image may be randomly zoomed in or out within its frame up to 20%, sheared left or right up to 20%, and/or flipped left to right but not upside-down.

Fig. 1. Sample training images, clockwise from the top left: diced, whole, sliced, grated.

B. Common Model Features

All models were generated using a truncated, frozen VGG-19 trained on ImageNet [1]. That is, VGG-19's weights (those obtained by training on ImageNet) were frozen such that training did not modify their values, and the final fully connected layers were excluded, instead replaced by custom-chosen layers; the combinations thereof will be specified for each model revision. The exception to the freezing rule comes with model version 7, when the last four layers of VGG-19 were unfrozen and modified by training on this dataset.

In all models, the outputs of the truncated VGG-19 are fed through a Global Max-Pooling layer. After being "wrung" through numerous convolutional layers [3] of various depths, the inputs to the network will have been squeezed into a narrow, deep vector; in extreme cases, this can be a series of 1x1 pixels stacked dozens deep.

In the case that we have not reached 1x1xD size, Global Max Pooling [4] does this for us. In the same way that Max Pooling reduces small subregions, often 2x2 pixels, to the maximum value within them, downsampling the image by drawing through the most intense values, Global Max Pooling reduced an entire image down to a single number: The maximum pixel value within it. So, for any stack of convolved "images" of dimension (W, W, D) where W is the square width of the current layer outputs and D is the depth, we will always reduce that to a (1, 1, D)-dimensional

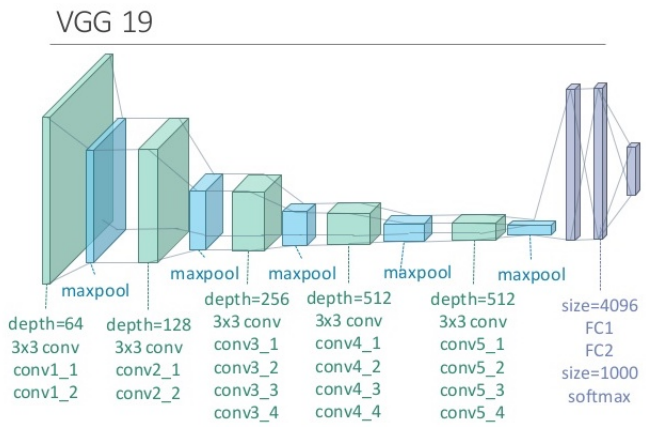


Fig. 2. Architecture of VGG-19 before modification. Note that the final three fully connected layers are replaced for this study. [2]

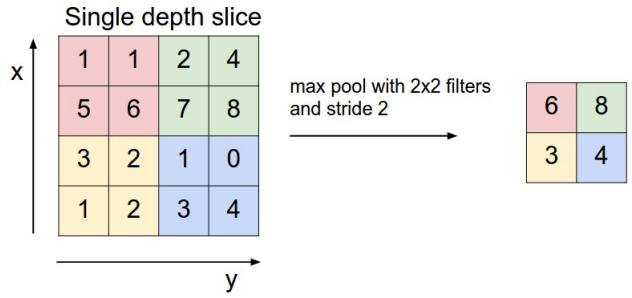


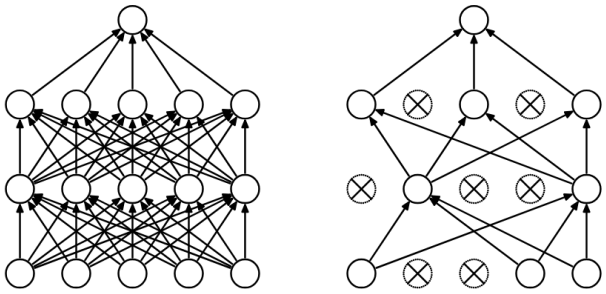
Fig. 3. A simple demonstration of 2x2 max pooling with a stride of 2 [5].

output, which can effectively be flattened into a D-length array and fed to new layers.

All models also end with a densely-connected layer with 11 outputs, one per class, activated using a Softmax function. Simply put, the dense layer attempts to automatically perform feature extraction and classification on the inputs fed to it; the Softmax function takes the outputs and normalizes them as an array of probabilities of each class being the class of the model input, each probability being some slice of the [0.0, 1.0] range such that the sum of the output vector will always be 1. The highest probability is the chosen class.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Many of the parameterized layers also feature dropout. Dropout is a method of regularization (avoiding overfitting) which "drops" weights and neurons in the network for each training sample [6]. At 0.2, for example, 20% of the weights in the network are simply excluded from training. This effectively reduces the layer to some randomly selected subconfiguration of the weights and neurons. The whole network still exists, but for one sample, only a subset of the network is trained. Because dropout is applied freshly with each training sample, we effectively train thousands of different subnetworks, with the possibility that we retrain the same configuration twice remaining very low. This has the effect of improving the accuracy of models while preventing overfitting by dramatically reducing the possibility that weights overtrain for a specific set of samples.



(a) Standard Neural Net (b) After applying dropout.

Fig. 4. Diagram depicting the application of dropout during model training. [6]

During training, neural networks use a backpropagation algorithm, using the derivative of the loss chained with the derivatives of the activation and weighted inputs, to calculate how much weights should change at each layer and at each neuron. These derivatives are chained back layer by layer until the whole network has been modified, "learning" from its degree of incorrectness on the previous input.

$$L(\mathbf{y}) = -\sum_i t_i log(y_i)$$

All models discussed use categorical crossentropy as the loss function which acts as the root of the backpropagation algorithm. Once we have the softmaxed output vector, we apply crossentropy by passing the expected output, which will usually be one-hot encoded (1 for the correct class, 0 for all others). In the crossentropy formula, when the output and expected value are close, we obtain very small loss values. However, for large deltas between expected value and output value, we can obtain very large loss values. This forces weights to move quickly towards their trained values, as crossentropy quickly decays once the predicted output nears the true expected value. The cross entropy curve is plotted in Figure 5 for reference.

III. MODEL REVISIONS AND METHODOLOGY

A. Versions 1 through 5-SGD

Most of these models failed to break 50% accuracy; some failed to increase in accuracy at all. Their architecture was composed of a similar layout to later models, with convolutional layers followed by dense layers, but with RMSprop or Stochastic Gradient Descent optimization and no regularization.

B. Version 5 Adadelta

The greatest leap at the median point in the model refinement process is the adoption of Adadelta optimization [7]. Adadelta dynamically tunes the learning rate, and much less aggressively than previous optimization algorithms, as the model is trained. With this change, accuracy leapfrogs over previous attempts in the same number of epochs, breaching 50% accuracy within 10 epochs, while previous versions of

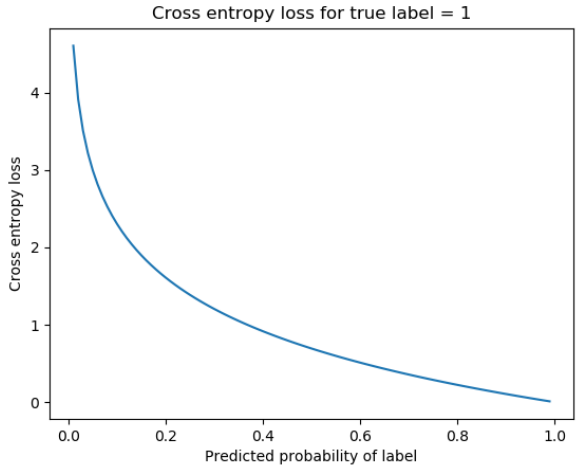


Fig. 5. Plot showing the quick decay of cross entropy loss as the prediction nears the true value

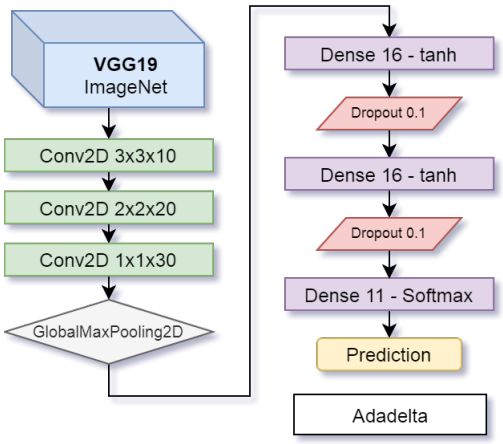


Fig. 6. Network architecture for version 5 with Adadelta optimization

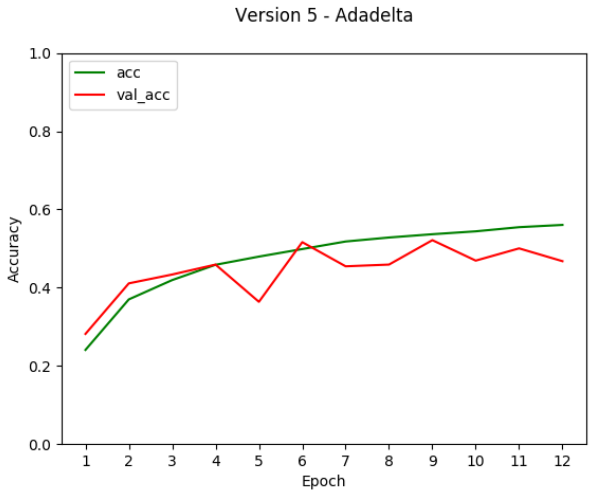


Fig. 7. Results for model version 5 with Adadelta optimization

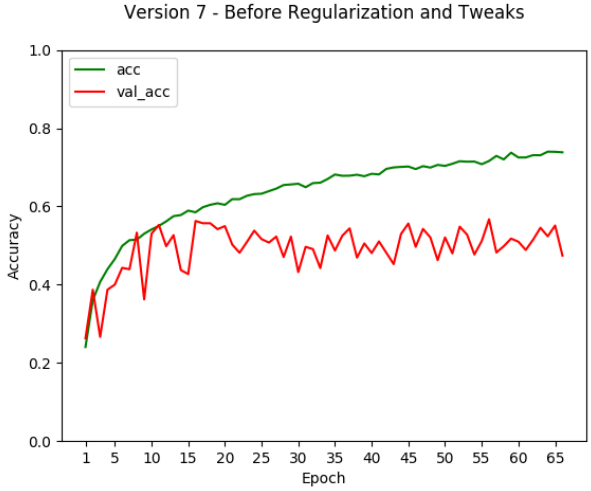


Fig. 8. Results for model version 7 before tuning and optimizations

the model had begun leveling off in the 40% range. The goal was to find an optimization algorithm which would find an error minima faster than methods like RMSprop or SGD, and it appeared to be working.

C. Version 7 Before Regularization

With this version, we see a dramatic shift in the upper range of training set accuracy by adopting the Nadam optimization algorithm [8] with a learning rate of 0.01. Nadam is the addition of Nesterov momentum to Adam (adaptive moment estimation) [9]. Adam uses calculations of lower-order moments to achieve good gradient descent on noisy, non-convex gradients of various types. Nesterov momentum, shown to be generally superior to classical forms of momentum [10], is applied for an overall superior optimization algorithm.

However, with dropout being our only regularization method, this model tends to overfit the training data. Our training accuracy rises at a consistent rate: We see it nearing 75% accuracy on the training set. But once the validation accuracy reaches 50%, it begins to fluctuate around that point, exactly what we'd expect to see with overfitting. The model is essentially guessing on the validation set.

D. Version 7 After Regularization

The most significant changes in this version are the reduction of the learning rate, addition of L2 regularization to deep learning layers, and the unfreezing of VGG-19's latter layers.

With our Nadam optimization, we change the learning rate to 0.0001. This lessens the amount by which loss is reduced with each backpropagation. Because the loss stays high at each epoch, decaying much more slowly, the network's weights see large changes every epoch and are free to converge more readily towards values that can classify accurately. In fewer epochs, we see large increases in training set accuracy.

L2 regularization is an additional term applied to the loss function in order to conditionally modify the loss and prevent

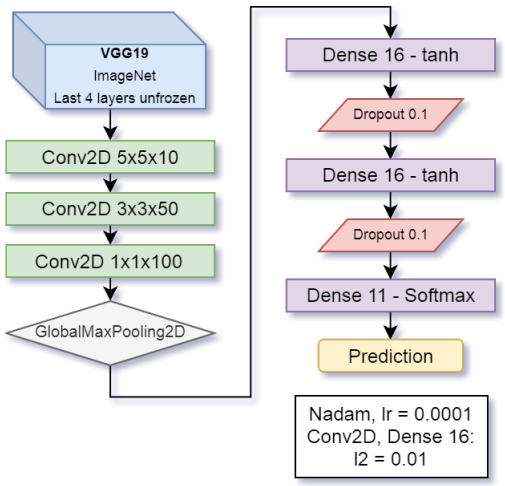


Fig. 9. Network architecture for version 7 after regularization

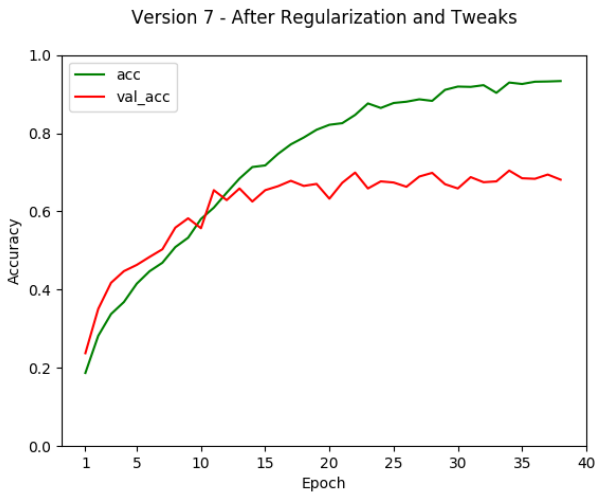


Fig. 10. Results for model version 7 with regularization, Nadam optimization, unfrozen VGG-19 tail layers

overfitting. If weights are allowed to get too large, they could overfit the training examples by essentially encoding them into the network; this results in high accuracy on the training samples, but low accuracy on any unknown data. With a validation set present during training, overfitting manifests as the validation accuracy at each epoch fluctuating at 50% or worse; this effectively means that, regardless of high accuracy on the training samples, the algorithm is guessing on other data.

$$L' = L(\mathbf{y}) + \lambda \sum_{i=1}^{k} w_i^2$$

The process of L2 regularization is as follows: By adding the sum of squares of the magnitudes of the weights to the loss function, larger weights incur exponentially greater loss, resulting in the decay of most weights closer to (but not exactly) zero. This thinner weight matrix ensures that the network does not overweight inputs as they propagate deeper into the network. Instead, weights are constantly suppressed at each backpropagation, forcing the network to discern more generally applicable features from the inputs. This ensures those features can propagate to the outputs and effect the ultimate classification decision. Thus, we should see high accuracies on the validation set during training, and consummately higher accuracy on unknown data.

With these changes, and the unfreezing of the last several layers of VGG-19 such that these layers' weights can change during training, we see a dramatic improvement: The model converges after less than 38 epochs, at which point early stopping triggers due to a lack of change in validation accuracy.

IV. Evaluation and Results

While model revision 5 with Adadelta and revision 7 before regularization show overfitting on the training set, the promising increases in accuracy translate well after version 7 is further tweaked and regularized, such that we see excellent training accuracy and satisfactory validation accuracy. For all models, training accuracy has improved from a low of 0.10 to a high of 0.93.

The test set consists of 680 images. The trained model, (version 7 after regularization) achieved 0.7029 accuracy with 1.7135 loss. This is on-par with the validation accuracy we saw during training, which was 0.7044 with a loss of 1.7496.

TABLE I
Best Accuracy and Loss for Model Revisions Discussed

Model	Epoch	Accuracy	Loss	Val. Accuracy	Val. Loss
5_adadelta	11	0.5546	1.3198	0.5004	1.4833
7_before	65	0.7396	0.7914	0.5512	1.4464
7_after	34	0.9295	0.7717	0.7044	1.7496
Test 7_after	-	-	-	0.7029	1.7135

V. Conclusions

Our model had a relatively similar structure through most of the testing; we see that the greatest improvements to transfer learning (an extension and retraining of a pretrained network), is via the use of a low learning rate, robust regularization, and an aggressive optimizer. We also see that convolutional neural networks are indeed robust for solving the problem of classifying states among a great variety of foods; the promising results of this study suggest that further model extension and hyperparameter tuning would yield a highly accurate model. In addition, this is promising for further extension into the combined object-and-state classification problem.

Thus, using transfer learning on VGG-19 pretrained with ImageNet images, we have extended it with additional layers and hyperparameter tuning to achieve promising accuracy on state recognition of images of food at various stages of cooking, from raw to diced to mashed to cooked, converging after a small number of epochs.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[2] T. Lodaya, "Neural style transfer." https://github.com/tejaslodaya/neural-style-transfer, 2017.

[3] Y. Bengio and Y. Lecun, "Convolutional networks for images, speech, and time-series," 11 1997.

[4] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013.

[5] F.-F. Li, A. Karpathy, and J. Johnson, "Cs231n: Convolutional neural networks for visual recognition 2016,"

[6] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[7] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012.

[8] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[10] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *30th International Conference on Machine Learning, ICML 2013*, pp. 1139–1147, 01 2013.