Using K-Fold Cross Validation and ResNet Ensembles to Predict Cooking States

Fred Mubang

Abstract— **In this paper, we discuss the usage of Neural Networks in predicting the cooking state of various food items. This paper focuses on 11 cooking states: (1) sliced, (2) grated, (3) mixed, (4) floured, (5) juiced, (6) whole, (7) peeled, (8) creamy paste, (9) jullienne, (10) diced, and (11) other. Using k-fold cross validation, transfer learning, and a majority voting ensemble of various fine-tuned Resnet50 models, we achieved a test accuracy of 78.72%. We also provide insights regarding how optimizer, regularization, and layer selection can affect results.**

I. INTRODUCTION

Within the last few years, the fields of computer vision and robotics have been gaining more prominence and usage than ever before. Many industries and organizations are realizing the capabilities of these fields in order to complete tasks humans are incapable of achieving on their own.

One domain in particular that computer vision and robotics could aid in is food preparation. There are currently many factories and even a few restaurants that utilize automated food preparation methods, but there is still much room for improvement. In order for a robot to correctly and efficiently undertake in food preparation, a deep understanding of the various types of foods and the ways in which they can be prepared is required. If a robot could properly distinguish between various food states, that would pave the way for more versatile robotic chefs. In [1], Jelodar et al. found that neural networks can better understand cooking videos if they can first understand the difference between various cooking states. The work in this paper is based off the insights revealed in that work.

In this paper, we discuss various experiments performed with deep neural networks in order to recognize the states of various foods. In particular we focus on the following cooking states: (1) sliced, (2) grated, (3) mixed, (4) floured, (5) juiced, (6) whole, (7) peeled, (8) creamy paste, (9) jullienne, (10) diced, and (11) other.

This paper seeks to make the following contributions:

- We performed transfer learning with the Resnet50 architecture and achieved an accuracy of 78.72%. We will outline our methodology.
- We will show the effectiveness of cross-validation and ensemble learning within the paradigm of food state recognition.
- We will show the different architecture and parameter configurations we made and discuss how these changes effect performace.

This paper's sections are as follows. Section II contains information regarding previous work. In Section III, we discuss our methodology for data preprocessing, architecture design, and ensemble training. In Section IV, we reveal and analyze our results. In Section V, we discuss potential future work and in Section VI we provide a closing summary of our experiments.

II. PREVIOUS WORK

A. *Resnet and Transfer Learning*

In this paper, we chose the Resnet50 architecture for our experiments. In 2015, this architecture won 1st place on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation in the ILSVRC and COCO 2015 competitions [2].

The creators of ResNet note that that the depth of a neural network can help it better learn how to classify various inputs. This insight is further backed by the fact that when the creators made their very deep architecture, they obtained a 28% relative improvement on the COCO object detection dataset [2].

The ResNet model has been proven to be helpful in transfer learning tasks. In [13], Kensert et al. used the pretrained Resnet50 model, InceptionV3, and InceptionResnetV2 models to classify cell images . They achieved accuracies between 95-97%. In [14], Gao et al. used Resnet Transfer learning for image-based structural damage recognition.

B. *Food State Classification*

There are many papers related to food state classification. In [3], Chen used transfer learning with the VGG19 network in order to perform food state classifications. He was able to achieve an accuracy of 80% when predicting 7 different states. In [4], Paul also used a modified VGG architecture to classify 7 states and achieved an accuracy of 77%. In [5], Sharma used transfer learning with the Inception v3 module of the GoogleLeNet and achieved a 76% accuracy. Lastly, in [6], Salekin et. al also used the Inception v3 module of the GoogleLeNet and achieved a 73.3% accuracy.

C. *Object and State Classification*

Lecun et al. discuss the power of Convolutional Neural Networks in classifying various images in [15]. Lester et al. [7] sought to predict to predict 10 different human activities and successfully did so with an accuracy of 95%. He used an ensemble of static classifiers and Markov models. In [11], the authors use neural networks to learn human walking patterns. In [12], Krizhevsky et al. used deep convolutional neural networks to classify Imagenet images.

D. Ensemble Learning and Cross Validation

Cross Validation and the use of ensemble models are effective methods for improving model performance accuracy and reducing predictive bias. In [17], Kohavi compares and contrasts the cross validation and bootstrap accuracy estimation methods. In [18] Ju et al. discuss the mechanics of various types of model ensembles. They discuss the advantages of majority voting ensembles, which are the types of ensembles discussed in this paper.

III. METHODOLOGY

In this section, we discuss our methodology for food state prediction task. In subsection A, we discuss our process for creating folds; in B, we discuss data augmentation; in C we describe the different architectures we used; in D, we discuss our training process; and in E, we discuss our majority voting ensemble method.

A. Data Preprocessing: Creating Folds

We started with a training set of 6,348 images and a test set of 1377 images. Since we had little data to begin with, we decided we would use K-fold cross validation in order to reduce predictive bias [17]. Using the python library Sci-kit Learn [19], we performed a stratified K fold, with K set to 3. As a result, we ended up with 3 training and test sets made from the 6,348 original training images. The first two folds contained 4,761 training images and 1,587 test images. The third fold contained 4,757 training images and 1,591 test images.

Usually when creating K folds, K is usually set to 10 [17], however we chose 3 because our computational resources were limited.

B. Data Preprocessing: Data Augmentation

For our next step, we performed data augmentation. Using the fit-generator function in the python library Keras [20], we performed various transformations on the data. Specifically, we did the following:

- Rotation with a range of 0 to 8 degrees
- Width shifting with a range of 0 to 8%
- Shearing with a range of 0 to 30%
- Height shifting with a range of 0 to 8%
- Zooming with a range of 0 to 8%

After applying the various transformations, we ended up with much larger training and validation sets. The first 2 folds contained 19,044 training images and 6,348 validation images each. The 3rd fold comprised of 19,028 training images and 6,364 validation images.

To be clear, both the augmented training images and augmented validation images came from the original training set of 6,348 images. We did not use any images from the original 1,377 test images when creating our new training and validation sets in order to avoid training or validating on test data.

Also note that we separated our validation images from the training images before performing data augmentation. If we had created our K folds after data augmentation, then we would have run the risk of having similar images in both our training and validation sets, which could potentially lead to overfitting.

Ensemble	Model Name	Fold #	Training Samples	Validation Samples
OR	OR-1	1	19,044	6,348
	OR-2	2	19,044	6,348
	OR-3	3	19,028	6,364
MR1	MR1-1	1	19,044	6,348
	MR1-2	2	19,044	6,348
	MR1-3	3	19,028	6,364
MR2	MR2-1	1	19,044	6,348
	MR2-2	2	19,044	6,348
	MR2-3	3	19,028	6,364

TABLE I: **This table describes the 3 different ensembles used, as well as the models that make up each ensemble. For example, the Original Resnet 50 ensemble (OR), is made up of 3 models, OR-1,OR-2, and OR-3. These 3 models correspond to the cross validation folds 1,2, and 3 respectively. OR-1 and OR-2 were trained on 2 distinctive training sets of 19,044 images, and were validated on two different validation sets of 6,348 images. OR-3 was trained on a set of 19,028 images and validated on a set of 6,364 images.**

C. Neural Network Architectures

Next, we created 3 different neural networks, all based the Resnet50 Architecture [2]. We chose this architecture because it is known to be very robust as made evident in the works [2], [13], and [14]. In 2015, this architecture won 1st place on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation in the ILSVRC and COCO 2015 competitions [2].

In order to distinguish between the 3 architectures,in this paper we will refer to the original Resnet50 Architecture as “OR”, and we will refer to the modified architectures as Modified-Resnet1 and Modified-Resnet 2; or “MR1” and “MR2” respectively. Note that in our version of the “Original Resnet50” architecture, we changed the output layer of 1000 units to an output layer of 11 units because in our prediction tasks we only have 11 classes.

Table I shows the 3 different ensembles used, as well as the models that make up each ensemble. For example, the Original Resnet 50 ensemble (OR), is made up of 3 models, OR-1,OR-2, and OR-3. These 3 models correspond to the cross validation folds 1,2, and 3 respectively. OR-1 and OR-2 were trained on 2 distinctive training sets of 19,044 images, and were validated on two different validation sets of 6,348 images. OR-3 was trained on a set of 19,028 images and validated on a set of 6,364 images.

We performed experiments on the original Resnet50 architecture in order to have a standard by which we can judge the other architectures’ performance. As one can see in Table II, all 3 architectures take in 224x224x3 images as input. In addition, MR1 and MR2 both start with the same 5 Convolutional blocks as the original Resnet50. MR1 differs in that it has an additional block that contains a Dropout layer with a rate of 20%, and 3 fully connected layers with 4096, 2048, and 1024 units respectively. Each of these layers use relu activation. Dropout was added was for regularization purposes, and the reason the fully connected layers were added was because Chen in [3] used fully connected layers in one of his model architectures and found success with that approach. However, Chen also added 3 more convolutional layers to his architecture, which we did not do because the Resnet50 architecture already contains many convolutional layers. Chen used the VGG network in his approach, so the addition of 3 more convolutional layers is a more logical decision for his architecture. In lieu of more convolutional layers, we tried adding 3 more fully connected layers for a total of 6 additional fully connected layers, but our models did not perform well with this approach, so in this paper we only report the results of our models with the 3 fully connected layers.

MR2 differs from MR1 in that we added an additional convolutional layer to this architecture in order to measure the effect that an additional convolutional layer would have on the results. We only added 1 because the baseline Resnet50 architecture that MR2 is based on is incredibly deep to begin with. We speculated that perhaps even more convolutional layers would be unnecessary. After the convolutional layer is a 2D Maxpooling layer with 2x2 pool size. We added this layer because it is standard practice to place maxpooling layers after a convolutional block [15]. Similar to MR1, we added the 3 fully connected layers. However, this time, we added an L2 regularizer set to 0.01. We wanted to see if adding additional regularization in addition to Dropout would help reduce overfitting while still allowing the model to learn appropriately.

D. Training Parameters

We trained each model with an initial learning rate of 0.001 and a decay rate of

$$9^{-5}. \tag{1}$$

We chose those parameters because Chen in [3] found success with those values in his experiments. We kept the learning rate and decay rate the same across all model configurations for the sake of consistency. Each model was trained for 50 epochs because more epochs did not seem to make a difference and less epochs negatively effected performance on the validation data. A batch size of 32 was used.

When training, we froze all layers of convolutional blocks 1 through 5 of the Resnet50 model for all 3 architectures we used. Table II describes these convolution blocks in detail. The only layers that remained unfrozen were the additional layers we added. For example, in the case of OR, which is the original Resnet50 model with a modified output layer, the only weights that were updated were the weights connected to our modified final output layer of 11 units.

We performed experiments with 3 different optimizers, which were Stochastic Gradient Descent (SGD), Adam, and Adadelta. We chose these 3 because these are some of the most popular optimizers used today. The effectiveness of these optimizers are discussed in [8], [9], and [10]. In total, we created an ensemble for every architecture-optimizer pair. There were 3 architectures and 3 optimizers, so in total we created 3*3, or 9 ensembles. Each ensemble comprised of 3 folds as discussed in section III-A, so in total there were 27 models. Using a Keras callback function, each of the 27 models' weights were the weights who achieved the highest validation accuracy for that particular model during the 50 epoch training period. So in theory, if a model M's best weights were in epoch 10, those weights were the ones that were saved in the final version of the model, not the weights at epoch 50.

Block Name	Output Size	Convolutions	Number of times the Blocks are Repeated
Input	224 x 224 x 3	Input images with dimensions 224 x 224 x 3	1
Conv1	112 x 112	64 kernels of size 7x7 and stride 2, followed by 3 x3 max pooling layer with stride 2	1
Conv2	56x56	1x1, 64, 3x3, 64, 1 x1, 256	3
Conv3	28x28	1x1, 128, 3x3, 128, 1x1, 512	4
Conv4	14x14	1x1, 256, 3x3, 256, 1x1, 1024	6
Conv5	7x7	1x1, 512, 3x3, 512, 1x1,2048	3
Output	11	Fully connected layer with 11 units and softmax activation	1

TABLE II: **The original Resnet50 architecture as seen in [2]. The "Block Name" column explains the name of the current block. The "output size" column shows the output size of that block. The "Convolutions" column explains the size and number of filters in each block. Lastly, the "Number of times the Convolutions are Repeated" shows how many times a sequence of convolutions is repeated. For example, in the Conv2 block, there are 3 repitions of the following convolution sequences: (1) 64 kernel operations of 1x1 size, (2) 64 kernel operations with 3x3 size, and (3) 256 kernel operations of 1x1 size.**

E. Majority Voting

Recall that there were 9 ensembles and 3 models per ensemble. When testing on the unseen test set, the 3 models in each ensemble would predict their own respective classes for the dataset. Then majority voting was used for each ensemble to choose each ensemble's final set of predictions for the unseen test data.

Block Name	Output Size	MR1	MR2
Input	224 x 224 x 3	Input images with dimensions 224 x 224 x 3	
Conv Blocks 1 through 5	7x7	**The same Conv1 to Conv5 blocks as original Resnet50.**	
Additional Block	56x56	(1) Dropout with a rate of 20%,(2) 3 Fully connected layers with 4096, 2048, and 1024 units respectively. All have relu activation.	(1) 2D Convolutional layer with 512 units and 3x3 kernel., (2) 2D Maxpooling layer with 2x2 pool size., (3) 3 Fully connected layers with 4096, 2048, and 1024 units respectively. All have relu activation. (4) Note: the FC layer with 2048 units has an L2 regularizer set to 0.01.
Output	11	Fully connected layer with 11 units and softmax activation	

TABLE III: **The Modified Resnet50 1 (MR1) and Modified Resnet50 2 (MR2) architectures. The block name column shows the name of the block. The output size column shows the output size of that block. Note that the first blocks of MR1 and MR2 are the same as the Resnet50 blocks. The row labelled "Additional Block" shows how MR1 and MR2 each differ from the original Resnet50 (OR). For example, MR1 has an added Dropout layer and 3 more fully connected layers than OR.**

IV. EVALUATION AND RESULTS

A. *Ensemble Accuracy Analysis*

The results of each ensemble are outlined in Table IV. As one can see, all 3 Resnet50 architectures performed with the best accuracies when using the SGD optimizer. SGD's average accuracy across all 3 ensembles was 76.98%. Specifically, the MR1 with the SGD optimizer performed the best with a 78.72% accuracy. In second place was MR1 with an accuracy of 78.72%, and in third place was OR with an accuracy of 75.53%. Table V shows the accuracy of each of the 27 individual models of which the 9 ensembles are comprised.

The stability of SGD is attested to in a paper by Wilson et. al [8]. In it the authors state that adaptive methods such as Adam and Adadelta have the potential to generalize worse than SGD. They note that these adaptive methods can overfit [8]. That seems to be the case in this context.

Although the ensemble with the best accuracy overall was from the MR1 group, the MR2 ensembles had the best ensemble average across all optimizers, which was 76.04%. In other words, the MR2 architecture was the most stable across the Adam, SGD, and Adadelta optimizers. Perhaps this was due to the use of L2 regularization in the MR2 architecture. This additional regularization parameter may have caused the architecture to have a better ability to learn a more general approximation function in contrast to the OR and MR1 architectures.

Note that both the modified Resnet50 architectures, MR1 and MR2, performed better than the baseline Resnet50 model, OR. Recall that both of the modified architectures contained an additional 3 fully connected layers on top of the original Resnet50 architecture. It would seem that these additional fully connected layers helped the ensembles fit the data more appropriately than OR.

What is also noteworthy is how poorly the MR1 architecture performed with the Adam optimizer. Its accuracy of 7.33% is much worse than any other result. Although MR1 had the best performance accuracy, overall the MR1 group of ensembles had the lowest performance accuracy of 53.37% across all optimizers. Perhaps it needed more regularization, such as an additional Dropout Layer, or L2 regularization, similar to MR2. Maybe, MR1 required an additional convolutional layer and Maxpooling layer in the same manner as MR2. Further experimentation is required to know for certain.

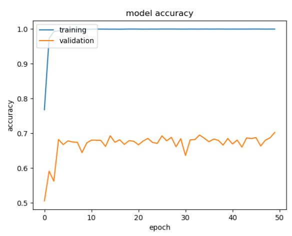
In summation, SGD was the most stable optimizer and MR2 was the most stable architecture. The MR1 had the highest accuracy of all architectures when using SGD, but it was the most unstable architecture across all 3 optimizers.

Figures 1, 2, and 3 contain the training/validation curves for loss and accuracy of selected models. They were made with the Python graphing library Matplotlib [21]. We analyze these graphs in the following sections.

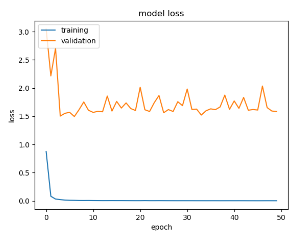
B. *SGD Training Graph Analysis*

As one can see in the SGD validation accuracy and loss graphs in Figure 1, there seems to be an overfitting issue prevalent among all the models graphs except for the MR2-1 loss graph, which is Figure (1f).

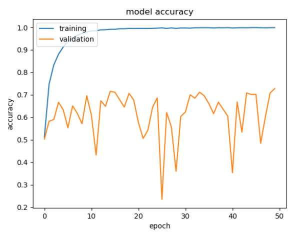
Note that the blue, smooth line is the training curve, and the orange, jagged line is the validation curve. The training accuracy quickly reaches close to or exactly 100% in each of the accuracy graphs, while the validation accuracies never seem to go any higher than 75% throughout all epochs. In the case of the accuracy graphs of MR1-1 and MR2-1 (graphs (c) and (e), respectively), the validation accuracy seems to be unstable and erratically "jumps" up and down, going as low as about 35% and as high as approximately 75%. and in all the loss graphs except for f, the training loss quickly reaches 0, while the validation losses do not. For OR, the loss never goes lower than 1.5, and for MR1-1, loss never goes below 1. For MR2-1, the training and validation loss follow a similar overall descending trend, with the validation curve tending to be much more erratic in behavior. This model was trained on 50 epochs like all the others, but this graph suggests that if this model was trained on more epochs perhaps the validation and training losses would have decreased even more. However, we trained all models with 50 epochs because for most of the models, using more epochs did not increase validation accuracy. We did not increase the



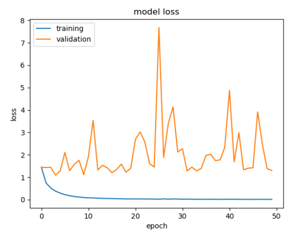
(a) SGD Original Resnet50 Model 1 (OR-1) Accuracy Graph



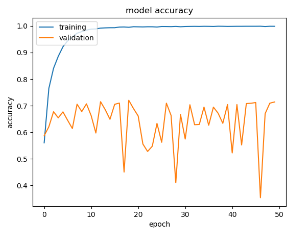
(b) SGD Original Resnet50 Model 1 (OR-1) Loss Graph



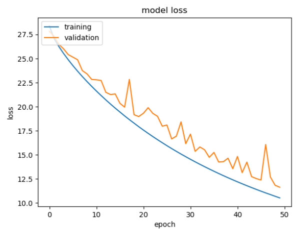
(c) SGD Modified Resnet Architecture 1, Model-1 (MR1) Accuracy Graph



(d) SGD MR1-1 Loss Graph

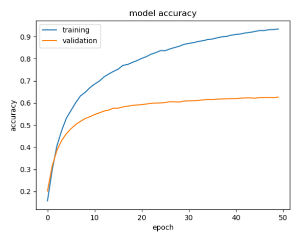


(e) SGD MR2-1 Accuracy Graph

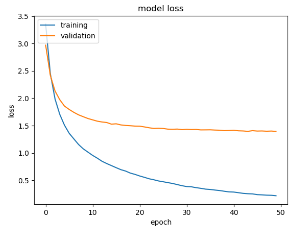


(f) SGD MR2-1 Loss Graph

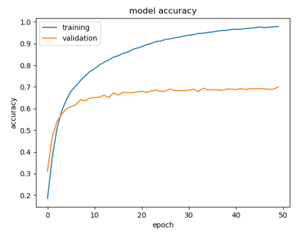
Fig. 1: **The training and validation graphs for selected models that used the SGD optimizer.**



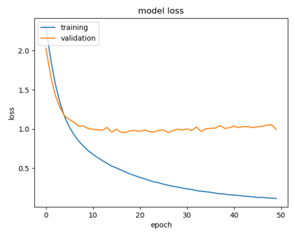
(a) Adadelta OR-1 Accuracy Graph



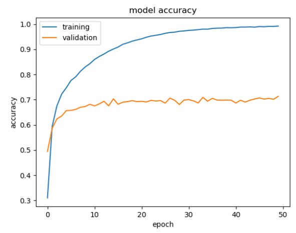
(b) Adadelta OR-1 Loss Graph



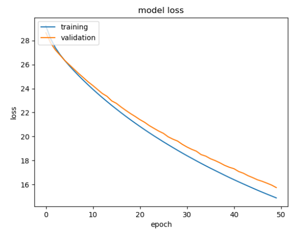
(c) Adadelta MR1-1 Accuracy Graph



(d) Adadelta MR1-1 Loss Graph



(e) Adadelta MR2-1 Accuracy Graph



(f) Adadelta MR2-1 Loss Graph

Fig. 2: **The training and validation graphs for selected models that used the Adadelta optimizer.**

epochs for MR2-1 so that we could have a fair comparison of all architectures.

C. Adadelta Training Graph Analysis

Figure 2 shows the Adadelta graphs for each model. Overall, the Adadelta curves are much smoother than the SGD curves, despite the fact the SGD model ensembles outperformed all the Adadelta ensembles.

Recall that the Adadelta optimizer is calculated as a decaying average of all past squared gradients. The current running average E at time step t depends on the previous average and the current gradient ([8] and [9]). The Adam optimizer also uses a similar approach involving decaying running average. Note that SGD does not use a running average in gradient calculation, but merely the current change in error. Perhaps this is the reason that the training curves for Adadelta and Adam are smoother than the SGD curves. However, further experimentation is needed to know for certain.

However, there is still a large disparity between training losses and validation losses, as well as training accuracies and validation accuracies.

D. Adam Training Graph Analysis

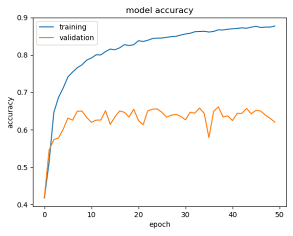
The accuracy and loss graphs for the Adam optimzer look similar for both the OR-1 and MR2-1 models. There is a large disparity between the training and validation curves, indicating overfitting, and the validation curves are erratic in a similar fashion to the SGD validation curves, albeit, to a lesser extent.

Recall that the worst ensemble performance out of all 9 ensembles was from the MR1 ensemble trained with the Adam optimizer. It had an accuracy of 7.33%. Graphs (c) and (d) in Figure 3 illustrate the poor training performance for model 1 of this ensemble. In the accuracy graph, both the validation and training curves plateaued at values less than 8%, and both the training and validation losses plateued at values slightly under 15.

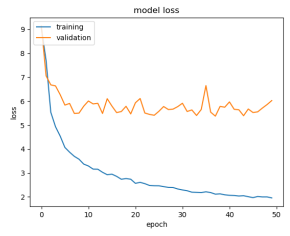
E. Confusion Matrix Analysis

Figure 4 is a normalized confusion matrix of the SGD MR1 ensemble results. This matrix was created with the Matplotlib Library [21]. We chose to display the SGD MR1 results because that was the ensemble with the best accuracy of 78.72%.

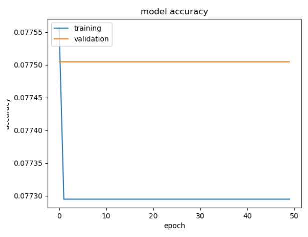
As one can see from the normalized confusion matrix, the ensemble did the best job at categorizing the “juiced” and



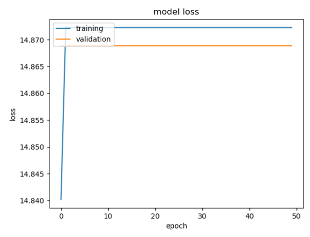
(a) Adam OR-1 Accuracy Graph



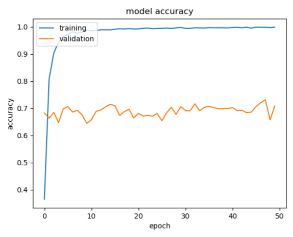
(b) Adam OR-1 Loss Graph



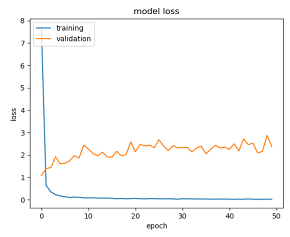
(c) Adam MR1-1 Accuracy Graph



(d) Adam MR1-1 Loss Graph



(e) Adam MR2-1 Accuracy Graph



(f) Adam MR2-1 Loss Graph

Fig. 3: **The training and validation graphs for selected models that used the Adam optimizer.**

Optimizer	OR Acc	MR1 Acc	MR2 Acc	Optimizer Avg. Across All Models
SGD	**75.53%**	**78.72%**	**76.69%**	**76.98%**
Adam	73.64%	7.33%	76.11%	52.36%
Adadelta	69.93%	74.07%	75.31%	73.10%
Model Avg. Across All Optimizers	73.03%	53.37%	**76.04%**	

TABLE IV: **The results of the ensemble models with the various optimizers tried.**

Model	SGD	Adam	Adadelta
OR-1	73.20%	**73.64%**	67.02%
OR-2	70.58%	71.60%	65.06%
OR-3	**72.55%**	62.52%	**68.63%**
MR1-1	75.38%	7.33%	72.04%
MR1-2	75.16%	7.33%	73.06%
MR1-3	**76.03%**	**10.38%**	**73.78%**
MR2-1	74.87%	**73.20%**	72.33%
MR2-2	73.42%	69.28%	**74.22%**
MR2-3	**75.01%**	72.62%	73.63%

TABLE V: **The accuracies of the individual models before combining them into their respective ensembles. The "Model" column shows the name of the specific model. For example, "OR1" is the Original Resnet50 architecture Model 1. OR-2 is model 2 of the OR architecture, and so on.**

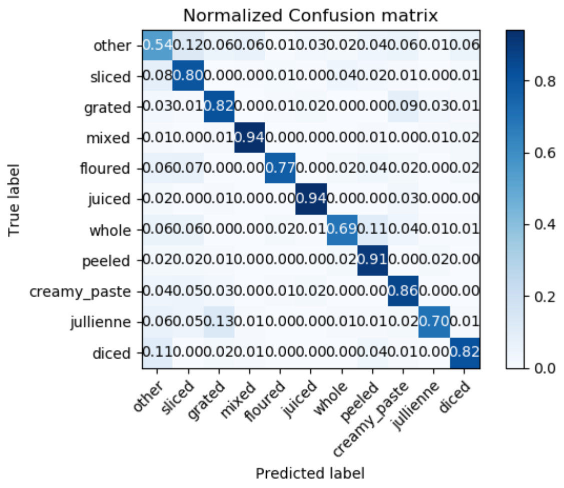


Fig. 4: **The normalized confusion matrix for the SGD MR1 ensemble results.**

"mixed" categories with an accuracy of 94% each. In second place, was the "peeled" category with 91%.

We re-examined the image data in order to speculate the reason "juiced" and "mixed" did so well, while "other" did not. We noticed that the images from the high-performance classes were, for the most part, unambiguous in their depiction. We show an example of this observation in Figure 5. The human eye can look at these images and clearly tell they are of "juiced" foods.

Some of the images of "other" class, however, are a bit more ambiguous. We show an example in Figure 6. Figure 6a and 6b look as if they could belong to the "diced" class, however, only Figure 6a was labelled as being in the "diced" class in the test data. Figure 6b was labelled as belonging to the "other" class. Similarly, Figures 6c and 6d look as if they both belong to the "julienne" class, however, only 6c is labelled as belonging to that class.

We also took note of some frequent mislabellings done by our networks. For example:

- "Other" was mislabelled as "sliced" 12% of the time, and "sliced" was mislabelled as "other" 8% of the time.
- "Julienne" was mislabelled as "grated" 13% of the time, and "grated" was mislabelled as "julienne" 3% of the time.
- "Diced" was mislabelled as "other" 11% of the time, and "other" was mislabelled as "diced" 6% of the time.

For future work, we must take more care to properly label the data to facilitate better model learning.

V. FUTURE WORK

There is much that can be done with regards to future work. For starters, we can spend more time labelling our data to make sure the labels properly match the data. We could also try K-clustering to see if there is a clean way we could categorize our data so that there is as little noise as possible.

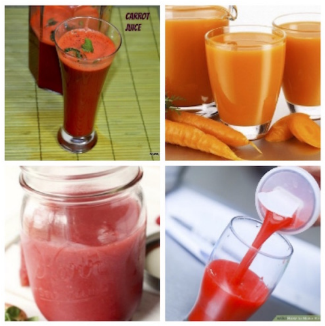
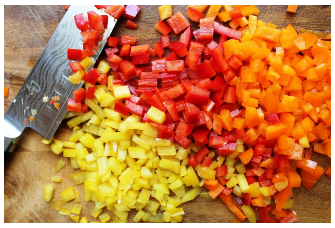
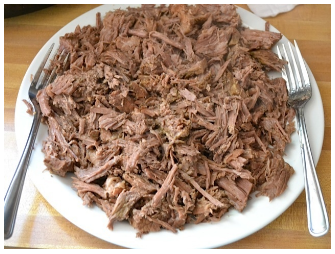


Fig. 5: **Examples of "juiced" images that are unambiguous in their depiction. Perhaps this is why the models classified this class with high accuracy.**

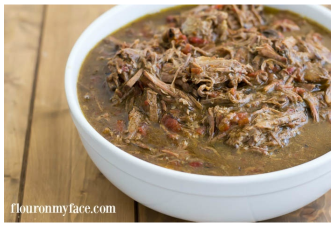


(a) Testing Image from the "Diced" Class

(b) Testing Image from the "Other" Class



(c) Training image from the "Julienne" class



(d) Training image from the "Other" class

Fig. 6: **Examples of possibly noisy images from the training and test sets.**

In terms of our cross validation and ensemble approach, we could make a few changes. We could try K=10 folds and try other ensemble methods such as bagging and boosting.

We could also perform more experiments to figure out why the MR1 design performed poorly with the Adam Optimizer. We could create 2 more ensembles- an MR1 architecture that uses L2 Regularization in addition to the Dropout Layer it already possesses, and an MR1 ensemble that contains no L2 Regularization, and a Convolutional Layer followed by a Maxpooling Layer. An experiment of this kind could possibly reveal to us why MR2 performed well with the Adam optimizer, while MR1 did not.

VI. Conclusion

In this paper, we discussed our experiments with Resnet50 transfer learning within the paradigm of food state classification. Using cross validation with the majority voting ensemble method, we created various ensembles, the best of which had an accuracy of 78.72% on the test data. We found Stochastic Gradient Descent to be the most reliable optimizer for our approach, and we found that our MR1 architecture was the most robust architecture. Recall that the MR1 architecture comprised of the Resnet50 architecture, topped with an additional Dropout layer, and 3 fully connected layers.

Furthermore, we observed that although SGD was the best optimizer in terms of accuracy, the gradients in the Adam and Adadelta models were updated in a "smoother" fashion relative to the SGD gradient updates, as made evident in the training and validation graphs in Figures 2 and 3. We speculated this may be due to the fact that Adam and Adadelta utilize expotential decaying averages in their formulas ([8] and [9]).

We then analyzed a confusion matrix our best ensemble's results. We discussed the importance of properly labelling the data and how mislabelling can contribute to prediction error. Lastly, we discussed potential undertakings for future work. Although we made many strides in our experimentation, there is still much room for improvement.

References

[1] Ahmad Babaeoan Jelodar, David Paulius, and Yu Sun, "Long Activity Video Understanding using Functional Object-Oriented Network", arXiv:1807.00983 [cs.CV], July 2018.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition" arXiv:1512.03385[cs.CV], Dec. 2015

[3] Tianze Chen, "Identifying States of Cooking Objects Using VGG Network", 2018. [Online].Available: `http://rpal.cse.usf.edu/reports/course_reports/2018_DL_Chen1.pdf.` [Accessed: 12- March- 2019]. http://rpal.cse.usf.edu/reports/

[4] Rahul Paul, "Classifying Cooking Objects State using a Tuned VGG Convolutional Neural Network",arXiv:1805.09391 [cs.CV], May 2018.

[5] Astha Sharma, "State Classification with CNN", arXiv:1806.03973 [cs.CV], June 2018.

[6] Md Sirajus Salekin, Ahmad Babaeian Jelodar, Rafsanjany Kushol, "Cooking State Recognition from Images Using Inception Architecture", arXiv:1805.09967 [cs.CV], Jan 2019

[7] Lester, J., Choudhury, T., Kern, N., Borriello, G. and Hannaford, B., 2005. A hybrid discriminative/generative approach for modeling human activities.

[8] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, Benjamin Recht,"The Marginal Value of Adaptive Gradient Methods in Machine Learning", arXiv:1705.08292 [stat.ML] May 2018

[9] Matthew D. Zeiler, " ADADELTA: An Adaptive Learning Rate Method", arXiv:1212.5701v1 [cs.LG] , Dec. 2012

[10] Sebastian Ruder, "An overview of gradient descent optimization algorithms" , arXiv:1609.04747v2 [cs.LG], 15 Jun 2017

[11] W. Kong, M. H. Saad, M. A. Hannan and A. Hussain, "Human gait state classification using artificial neural network," 2014 IEEE Symposium on Computational Intelligence for Multimedia, Signal and Vision Processing (CIMSIVP), Orlando, FL, 2014, pp. 1-5. doi: 10.1109/CIMSIVP.2014.7013287

[12] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, 2012.

[13] Alexander Kensert , Philip J Harrison and Ola Spjuth, "Transfer learning with deep convolutional neural networks for classifying cellular morphological changes", Published in "SLAS DISCOVERY: Advancing Life Sciences R&D doi: 10.1177/2472555218818756", Jan 2019

[14] Y. Q. Gao, K. B. Li, K. M. Mosalam and S. Gunay, " DEEP RESIDUAL NETWORK WITH TRANSFER LEARNING FOR IMAGE-BASED STRUCTURAL DAMAGE RECOGNITION", Published in "11th National Conference on Earthquake Engineering, At Los Angeles", June 2018

[15] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 1995.

[16] Baldi, P. and Sadowski, P. (2014). The dropout learning algorithm. Artificial Intelligence, 210C:78122.

[17] Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of International Joint Conference on AI. 1995, pp. 11371145, URL: `http://citeseer.ist.psu.edu/kohavi95study.html`.

[18] Cheng Ju and Aurelien Bibaut and Mark J. van der Laan, "The Relative Performance of Ensemble Methods with Deep Convolutional Neural Networks for Image Classification", arXiv:1704.01664v1 [stat.ML], 5 Apr 2017

[19] Fabian Pedregosa, Gal Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, douard Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011) (publisher link)

[20] Francois Chollet and others, *Keras*, Publisher: Github, 2015. URL: `https://github.com/fchollet/keras`

[21] J.D. Hunter, *Matplotlib: A 2D graphics environment*, Published in "Computing In Science & Engineering, Volume 9, No. 3", in Pages 90-95. Publisher: IEEE COMPUTER SOC, DOI: 10.1109/MCSE.2007.55, 2007.