COMP 540: iFood Challenge Report

comp540_yz154_jx24

April 25, 2020

Yue Zhuo NETID: yz154.
Jose Antonio Lara Benitez NETID: jx24.

1 Introduction

1.1 Overview

The goal of this project is to build a model to predict the fine-grained food-category label given an image. We used 251 fine-grained (prepared) food categories with 118,475 training images from FGVC6. There are human verified labels for both the validation set of 11,994 images and the test set of 28,377 images. The original datasets including files as following:

- class_list.txt – lookup table between class labels and names
- train/val/test.zip – the train/validation/test images
- train/val_labels.csv – the training and validation set labels
- ifood2019_sample_submission.csv – a sample submission file in the correct format

Food classification is a challenge task due to the large number of categories, high visual similarity among food categories, as well as the lack of datasets that are large enough for training deep models.

The main challenges are:

- Fine-grained Classes - The classes are fine-grained and visually similar.
- Noisy Data - the training images are crawled from the web, they often include images of raw ingredients or processed and packaged food items.

1.2 Exploratory Data Analysis

1.2.1 Clusters and Patterns

These image data belong to 251 classes, with different sizes and pixels. In order to check the patterns of images, we count the training data and validation data by class, then calculate the ratio between them. (sort by ratio).

	NameID	#Training	#Validation	%
149	poi	500	2	0.004000
241	limpet_food	551	9	0.016334
27	rugulah	376	11	0.029255
28	rock_cake	464	17	0.036638
72	crab_food	617	27	0.043760
...	...	...	...	...
183	seaweed_salad	326	57	0.174847
37	moo_goo_gai_pan	329	59	0.179331
141	french_fries	311	57	0.183280
34	matzo_ball	321	61	0.190031
120	marble_cake	34	49	1.441176

251 rows × 4 columns

Figure 1: Ratio Rank of Validation and Training Data Amount by Class

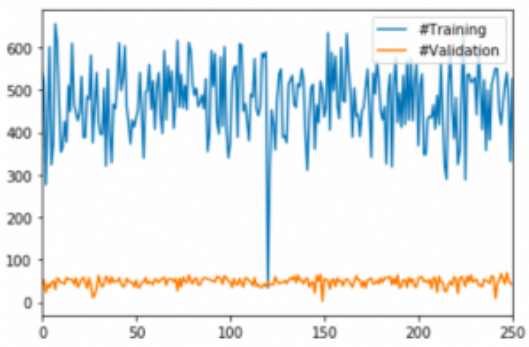

Figure 2: Number of Training and Validation Data by Class

As we can see in figures above, for most of classes, $\#Validation/\#Training$ is $5-15\%$. However, there are some abnormal points. For *marble_cake*, the number of validation data excesses the number of training data, which is unhealthy, we could consider deleting this categories, merging it with others or download additional images from other resources manually if permitted . Besides, for *poi* and $limpet\_food$, who have very few validation data, we could consider copying and adjusting images (like flip or rotation) to increase the number of validation data.

1.2.2 Outliers

When we look at the training data, we could find two kinds of outliers.

- **Wrong Classified Data** This kind of outliers are images classified with wrong labels, and we can correct the classes for them.
- **Data out of Sample Space** This kind of outliers are those who should not include in the training data.

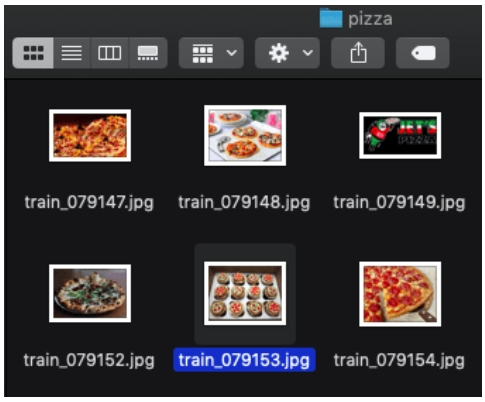

Figure 3: Image should belong to cupcake is wrongly classified to pizza

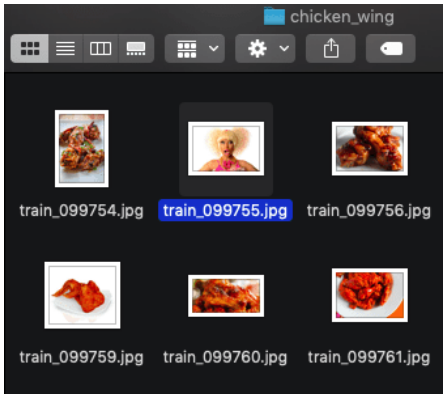

Figure 4: Some portraits and cartoon pictures instead of food images

1.3 Pipeline Plan

Based on previous analysis of data, we break down the project into 3 parts.

- **Data Pre-processing** We resized images into the same scale, then transform and augment images to prevent overfitting.
- **Training Models** We trained models including ResNet-50, ResNet-101, ResNet-152 and ResNeXt-101 individually to compare the performances of them.
- **Ensemble Learning** Combine different models and assign weights for them to get a better predictive performance.

2 Data Pre-Processing

In order to help models better recognize images and prevent overfitting, we use data transformation and data augmentation to clarify characteristics of images.

- **Data Transformation** We randomly transformed data by flipping, vert, warping, rotation, zooming, lighting and affine transformation.
- **Data Augmentation** We resized data into 224*224 square and normalized each image matrix.

Then we generated data batch by batch, the result of data pre-processing is shown below:

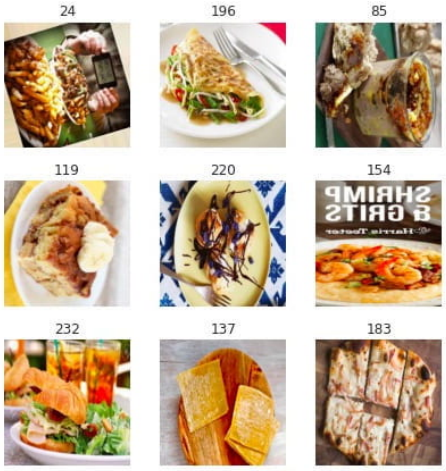

Figure 5: Image Data Examples after Pre-Processing

The numbers above each image are their labels. As we can see, we randomly selected data transformation methods for images and resized image size into the same scale.

3 Models and Architecture

3.1 Introduction

Given the wide range of images and classes in the competition, one of the main challenges was the selection of a model capable of being complex enough to capture the essential aspects of the images, as well as simple to train using low resources and the time limitation presented in *Google Colab*.

In the Computer Vision community is an standard practice to treat image classification on ImageNet [7] as an starting point for training deep convolutional neural networks [5, 4] to learn, what is called: the *essential general features*. This practice of first training a CNN to perform image classification on ImageNet (i.e. pre-training) and then adapting these features for a new target task (i.e. fine tuning) has become the standard approach for solving a wide range of computer vision problems, thanks to their success. This approach has been supported by an overwhelming empirical evidence.

For all the cases we chose ImageNet as our starting point for transfer learning. In [1, 3] the authors provided evidence that pre-trained models using ImageNet works

remarkably well for a large set of image classification tasks and often *outperform standard classification approaches* not only in accuracy, if not also in training time. Achieving in some of the cases, state of the art performances [8] using less resources. Such a properties are exactly the ones we desired for our classifier. In the rest of the project we follow such paradigm for different architectures.

3.1.1 Architecture family

We choose ResNet as well as its variation as our starting point. The motivation for that selection is the variety of members in the family, as well as the the empirical evidence, presented in [2] showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth.

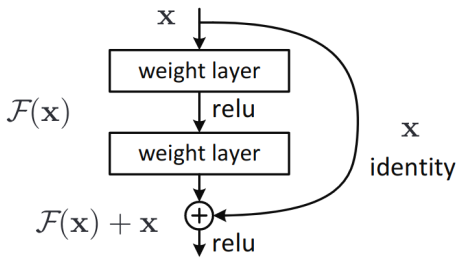

Figure 6: Residual learning: a building block.

As the authors in [2][1] showed, there is a strong evidence that the residual learning principle is generic, and applicable in other vision and non-vision problems.

3.1.2 Residual Learning

The idea is actually very simple. Under the hypothesis that multiple nonlinear layers can asymptotically approximate complicated functions, $\mathcal{H}(x)$ [2] so it can also approximate the function $\mathcal{H}(x)-x$ (if both are of the same dimension). So the stacked layers rather than approximate the function $\mathcal{H}(x)$ shall approximate the residual function $\mathcal{F}(x) = \mathcal{H}(x) - x$. Then the original function becomes $\mathcal{F}(x) + x$. Although both forms should be able to asymptotically approximate the desired functions,the ease of learning might be different.

[1]The image was taken directly from there.
[2]This is still an open question. It has been proven by shallow layers.

The main idea behind ResNet is the introduction of the identity connection that allows to skip some of the layers, as it is show in figure 7. This allows us to stack more layers without having problems of changing the performance or making more difficult the training, due that the skip layer are only identity maps.

This indicates that the deeper model should not produce a training error higher than shallower networks. So letting the stacked layers fit a residual mapping is easier. That is the idea behind the residual block.

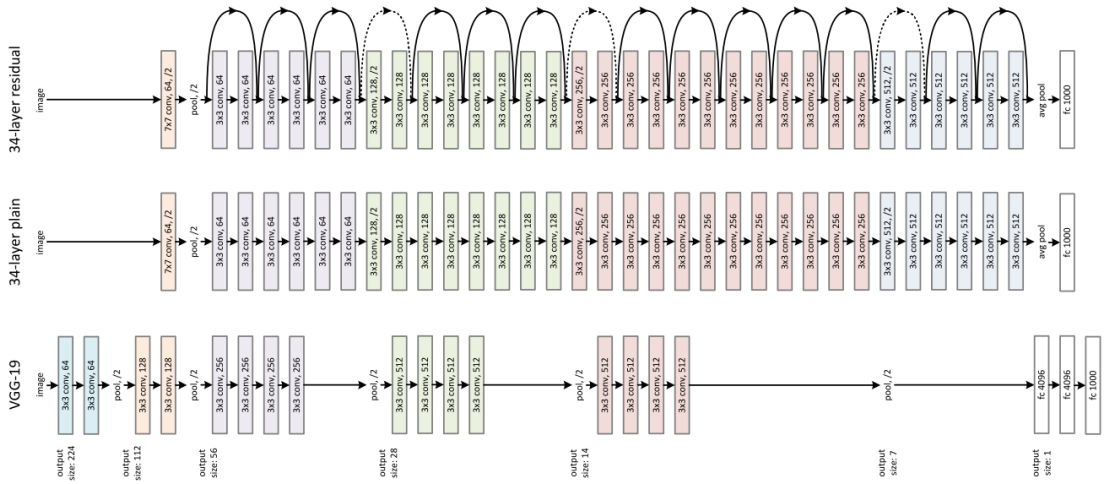

Figure 7: ResNet architecture.

3.2 Model selection philosophy

So our main idea in solving this problem is to start with the simplest models in the ResNet family to see which one fits well enough to have competitive results in the competition. As many researchers and practitioners of Machine Learning we invoke the Occam's razor: *the simplest one or the one with fewest assumptions will be best applied.*

To this end, firstly we try with the simplest elements in the ResNet's family and see its final score in the Kaggle competition. If the score is not high enough; high enough for purpose of this project is to be in the top ten leadership board. We tried hyper-parameters tuning, and see the improvement, otherwise we move to the next element in the family. Saving the previous model to build-up a more complex one which shall be used in the ensemble of CNNs for the final submission; weighting them according to its accuracy for the ensemble.

The above procedure can be described succinctly through the following algorithm for model selection.

Algorithm 1: Model Selection Algorithm.

```
Data: Images
Result: Kaggle Top10
Initialization;
while not at Top10 do
    Choose K ∈ (18, 34, 50, 101, 152);
    Train ResNetK;
    if Prediction are not in Top10 then
        save the model;
        go back to the beginning of current section;
    else
        Finish model selection;
        Start Ensemble
    end
end
```

3.2.1 Learning Rate Policy

In the literature, there are many ways to approach the problem of training a neural network. However, the main motif presented in each one is a "grid search" of hyper-parameters. This approach may lead to long training times and sub-optimal hyper-parameters. Currently setting the hyper-parameters still remains a black box which requires experience and extensive trial and error; which translate in time, expertise and reseources, which unfortunately we do not have[3].

Specifically, to get around this issue we follow the ideas that comes from [9]. One of the main point we strictly follow is the Cyclic Learning Rates (CLR) approach.

Let us briefly explain this approach. Instead of using a fixed, or a decreasing learning rate, the CLR method allows learning rate to continuously oscillate between reasonable minimum and maximum bounds.

For all the cases we search through a random sub-sample of the set of images, around the 15% to be more specific, and we find a learning rate value which is big enough, in a sense that we shall describe.

When the step size increases on this boundary value the loss function also increases and if we keep doing this, it becomes too large and causes the test/validation

[3]For the record, this is our first end to end trained neural network

loss to increase and the accuracy to decrease. For instances, the learning rate at this extrema is the largest value that can be used for the cycle of learning rates. As you can see on example in figure 10

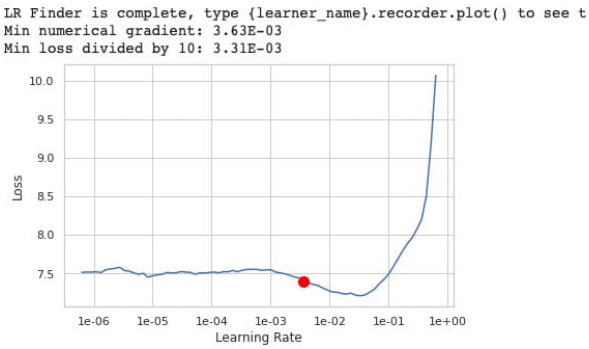

Figure 8: Example of extrema learning rate.

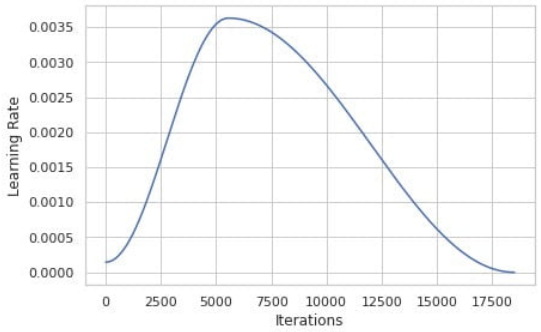

Figure 9: Example of Cycle scheme for the learning rate.

For the other boundary, there are several ways in which one can choose the minimum learning rate bound, in our case we appeal to simple rule of thumb that it seems to work well in practice: *the minimum learning rates are a tenth of the maximum.*

As you can see in figure 11, in all the cases we try the most simple case of one cycle per epoch. The only reason for that was simplicity's sake and the good results obtained.

3.2.2 Chosen Batch Size

In [9] the author mentioned the following:

> "*Total batch size (TBS): A large batch size works well but the magnitude is typically constrained by the GPU memory. If your server has multiple GPUs, the total batch size is the batch size on a GPU multiplied by the number of GPUs. If the architecture is small or your hardware permits very large batch sizes, then you might compare performance of different batch sizes. In addition, recall that small batch sizes add regularization while large batch sizes add less, so utilize this while balancing the proper amount of regularization. It is often better to use a larger batch size so a larger learning rate can be used*".

Unfortunately for our case the batch size is limited by the GPU memory. Using *Google Colab* this limitation is 15 GB, allowing batches of not more than 64 images. Therefore in all the cases, depending of the RAM available, the size of our batches in each one of the experiments was either 64 or 32.

3.2.3 Chosen Momentum

Since we follow [9] in the learning rate cycle policy, with good results the momentum term was also chosen to be cyclical. In all the experiments the momentum increases linearly between the lower bound of 0.85 and 0.95 with the same schedule than the learning rate. We observe empirically that these values stabilize the convergence when using large learning rate values more than a constant momentum does. See figure 15.

The momentum schedule we follow with the learning rate are presented below:

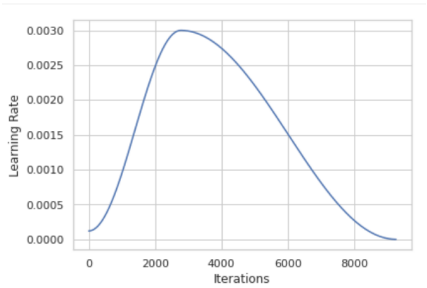

Figure 10: Learning Rate Cycles.

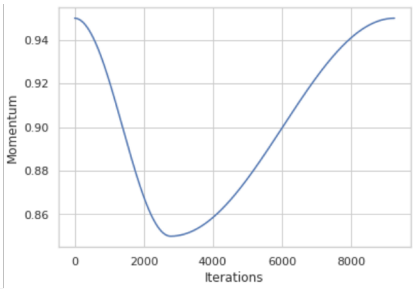

Figure 11: Momentum Cycles.

In general as we discussed before we get a very stable loss function behavior.

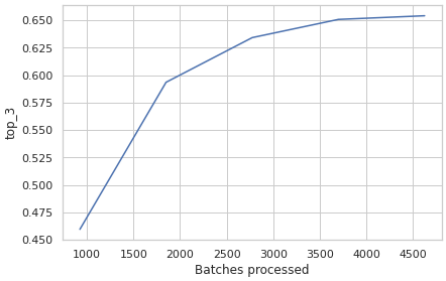

Figure 12: Example of validation accuracy with the cyclic policies in momentum and learning rate.

3.2.4 Chosen Loss function, Weight decay and Optimizer

For the optimizer since we use a pre-trained model and we trained in small cycle for all the cases we keep the ADAM optimizer and we did not experiment with the results using others.

For all the cases, the loss function was the label smoothing cross entropy, which by itself is as a *regularization* technique for classification problems. The main idea behind is that it prevents the model from predicting the training examples with high confident, helping in the generalization as it is presented in [10, 6].

Let us define in simple words how the labeling smoothing works. Instead of using directly the one-hot vector a random variable $u(y|x)$ is introduced such that

$$\mathbf{Pr}_\varepsilon(y|x_i) = (1-\varepsilon)\mathbf{Pr}(y|x_i) + \varepsilon\, u(y|x_i) = \begin{cases} 1-\varepsilon+\varepsilon\, u(y|x_i) & \text{if } y = y_i \\ \varepsilon\, u(y|x_i) & \text{if } y \neq y_i \end{cases}$$

The value ε lies on the unit interval and is weight that ensures $\mathbf{Pr}_\varepsilon$ being a probability distribution. In our case $u(y|x_i)$ is a uniform distribution independent on the data.

So the label smoothing loss function is

$$L_\epsilon = \sum_i^n \sum_k^K -y_k \log\left(\mathbf{Pr}_\varepsilon(y|x_k)\right)$$

For the weight decay (WD) since our dataset is complex and all the family ResNet is deep, following [9] and particularly thanks to the CLR, our CNNs require less regularization so we tested smaller weight decay values, in general 10^{-4} and 10^{-5} with the best results.

3.3 Our approach

This is our first end-to-end machine learning project using deep convolutional neural networks. As someone new in the area is actually very difficult to dig into the literature, first by the abundance of material, and second to the seemingly contradictory statements in papers. Unfortunately, there are plenty of contradictory recommendations of this kind in literature, or some techniques may help in some domains but not in others. This reflects better the lack of theory in the community and make a haunting task for someone new in the area without the experience hyper-parameters tuning. Our best bet was to follow the recommendations presented in [9], as it is one of the few with a very detail set of instructions that are independent of the expertise.

It is easy to follow and the experimental results are satisfactory. Thus, our general approach can be summarized as follows

- Using a Pre-trained CNN from the ResNet family. Starting from the simplest elements in the family and going into more complex neural networks depending of the position in the Kaggle's leaderboad. See algorithm 1 for further information.

- All the training was implemented in *Google Colab* with its time limitations.

- The chose image size was 224 in all the cases. It is still open to check the results with smaller sizes like 128.

- We follow the CLR (Cyclic Learning Rate) Policy. Starting from 0.03 to 0.003 in the beginning of the training, usually less than 10 epochs. When no changes are presented in the loss function the points are multiplied by 10^{-1} and so on, until the changes in the loss function are negligible.

- We follow the Momentum Cycle, moving the values linearly from 0.85 to 0.95. Opposite to the cycle of CLR.

- Our batch size was as big as allowed by the size of the memory. For all the cases, this values changes between 32 and 64 GB, depending of the GPU's memory, either 16 GB or 25 GB.

- The weight decay was fixed in the 2 or 3 stages of the training. For the first stage this was set to the value of 10^{-5}. In the second stage, this increases to 10^{-4} and later to 10^{-3} depending of the results in the loss function.

- The smoothed loss function was chosen to help generalization of the model.

- When there is not improvement we freeze the first layers and we only train the Fully Connected Layer.

- All the trained models, in the great majority simple models were saved to make an ensemble method weighted according to its accuracy and top-3-accuracy.

3.4 Results of Models

We followed the previous instruction. We firstly tried ResNet-18, a very simple[4] architecture. In the next figures we can appreciate the behavior of it.(one cycle with 5 epochs)

epoch	train_loss	valid_loss	accuracy	top_3	time
0	4.594469	3.910287	0.226863	0.396865	09:06
1	4.080281	3.544745	0.301317	0.495081	08:57
2	3.908153	3.387412	0.341171	0.537602	08:38
3	3.794395	3.284056	0.366266	0.566033	08:37
4	3.715261	3.263224	0.372686	0.570369	08:48

Figure 13: ResNet-18 learning rate.

After that and keep moving the learning rate we got an accuracy of around 0.61. Impressive but not enough to be in the top ten leaderboard.

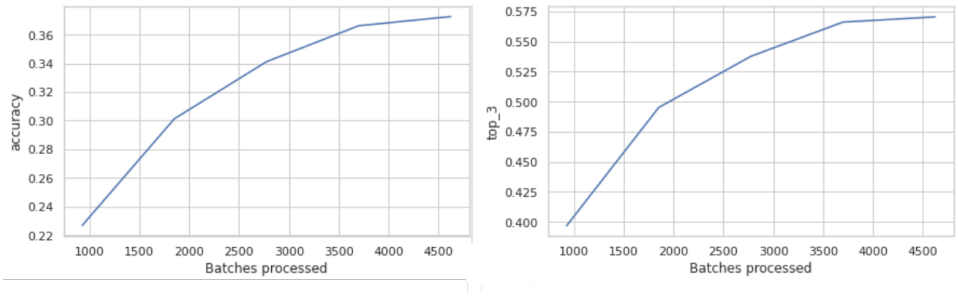

Figure 14: ResNet-18 learning rate.

Likewise, we checked the behavior of ResNet-34, ResNet-50, ResNet-101, ResNet-152 and ResNeXt-101. The final Top-3 accuracy of each model is as following table.

	ResNet18	ResNet34	ResNet50	ResNet101	ResNet152	ResNeXt101
Top3 Acc	0.7123	0.7285	0.8302	0.8686	0.8680	0.8836

As we can see, ResNeXt-101 has the highest performance, and it gives us enough flexibility to stay at least for a couple of days in the top three in the leaderboard. The model stabilizes at almost.

[4]simple for the current standards

4 Ensemble Learning

Neural network models are nonlinear and have a high variance, which can be frustrating when preparing a final model for making predictions. A successful approach to reducing this variance is to train multiple models instead of a single model and to combine the predictions from these models. This is called ensemble learning and not only reduces the variance of predictions but also can result in predictions that are better than any single model.

Here we used two methods to ensemble 4 models with Top-3 accuracy higher than 0.8 (ResNet-50, ResNet-101, ResNet-152 and ResNeXt-101):

- **Simple Arithmetic Average** Calculate the average predictive probability of each label.

$$finalpred = \frac{pred_1 + pred_2 + pred_3 + pred_4}{4}$$

  Result: Top-3 accuracy 0.8876

- **Weighted Arithmetic Average**

  - Take the inverse of RMSE as weight

$$finalpred = \frac{\frac{1}{RMSE_1}pred_1 + \frac{1}{RMSE_2}pred_2 + \frac{1}{RMSE_3}pred_3 + \frac{1}{RMSE_4}pred_4}{\frac{1}{RMSE_1} + \frac{1}{RMSE_2} + \frac{1}{RMSE_3} + \frac{1}{RMSE_4}}$$

    Result: Top-3 accuracy 0.8923

  - Manually tune weights based in the previous weights as a reference, and checking the validation set for a final decision.
    Result: Top-3 accuracy 0.9013

Same to our expectation, Weighted Arithmetic Average works better than Simple Arithmetic Average since it highlight the performance of stronger classifier. After ensemble learning, the Top-3 accuracy of our final prediction increased to 0.9013, which is our final and best result.

5 Submission Timeline

Since our team got 41 entries in total, here we just recorded representative results.

Date	Model	Selection	Score
April 6	VGG-16	Top1	0.99577
April 14	MobileNet	Top1	0.79200
April 14	ResNet-50	Top1	0.54070
April 15	ResNet-50	Top1	0.35451
April 15	ResNet-50	Top3	0.16844
April 17	ResNeXt-101	Top3	0.16363
April 17	ResNet-101	Top3	0.14495
April 18	ResNeXt-101	Top3	0.13438
April 19	ResNeXt-101	Top3	0.13050
April 19	ResNet-152	Top3	0.16527
April 20	ResNet-152	Top3	0.15388
April 21	ResNet-152	Top3	0.14166
April 23	ResNet-152&ResNeXt-101(simple average)	Top3	0.11535
April 23	ResNet-101+152&ResNeXt-101(simple avg)	Top3	0.11241
April 23	ResNet-50+101+152&ResNeXt-101(simple avg)	Top3	0.11488
April 24	ResNet-50+101+152&ResNeXt-101(weighted avg)	Top3	0.10771
April 24	ResNet-50+101+152&ResNeXt-101(weighted avg+mannual)	Top3	0.10677
April 24	ResNet-50+101+152&ResNeXt-101(weighted avg+mannual)	Top3	0.10301

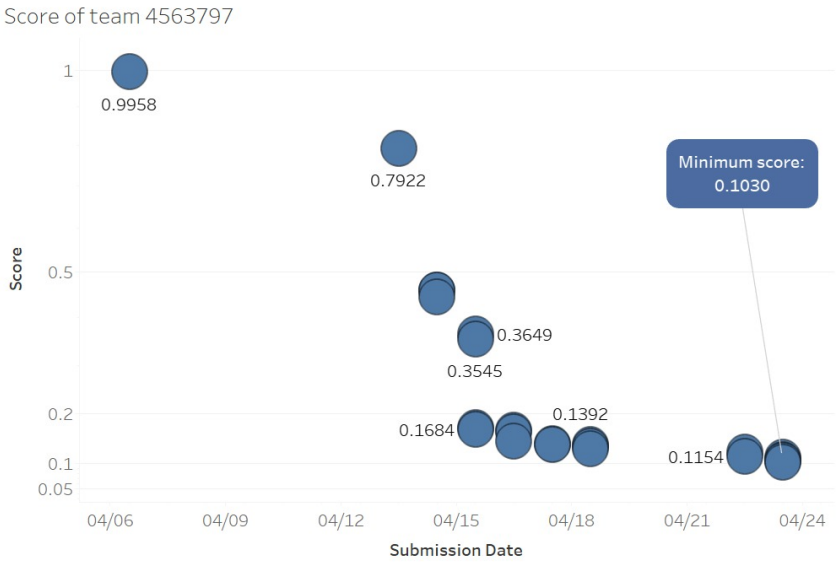

Figure 15: Kaggle's Timeline.

6 Conclusion

In this project, we tried different Neural Network models, tuned parameters following [9] disciplined approach and improved models step by step. Finally, we got the model score 0.10301, which led us to the second place on Kaggle's Leaderboard.

However, after some trial and errors in the ensemble we were able to reach the 90% top-3 accuracy in the validation. To be specific, our best ensemble reach, after several experiments, the value of 90.13%. Although we can not improve our models any more, we have some ideas for improvement; perhaps for next year competition.

- Filter outliers in pre-processing to eliminate noise (K-means or downloading other food image datasets).
- Better ensemble methods (bagging, boosting, etc.).

References

[1] Jeff Donahue et al. "Decaf: A deep convolutional activation feature for generic visual recognition". In: *International conference on machine learning*. 2014, pp. 647–655.

[2] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[3] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. "What makes ImageNet good for transfer learning?" In: *arXiv preprint arXiv:1608.08614* (2016).

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[5] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[6] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. "When does label smoothing help?" In: *Advances in Neural Information Processing Systems*. 2019, pp. 4696–4705.

[7] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[8] Ali Sharif Razavian et al. "CNN features off-the-shelf: an astounding baseline for recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014, pp. 806–813.

[9] Leslie N Smith. "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay". In: *arXiv preprint arXiv:1803.09820* (2018).

[10] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.