Inferring Ingredients from Food Images

Nikhil Sunil Nandoskar
Chenrong Qin
Qingtian Zou

ABSTRACT

Many attentions have been given to food classification from images, but little is given to predicting the ingredients in a food image. Inferring food ingredients is important for the elders, the sick and many other groups. The last several years have seen the great advancement in food classification using neural network. We decide to build a neural network to infer the ingredients in a food image with TensorFlow. We leverage the Vireo-Food 172[8] dataset and VGG-16[12] architecture to build and train our model. We also leverage transfer learning using a pre-trained VGG 16 model.

1 INTRODUCTION

Nowadays people are exceptionally good at human face recognition, however, not too many researchers care about inferring ingredients from food images. In our opinions, this is also a new but meaningful topic because different age groups of people have different food requirements and they need to concern their daily diets, and then decrease the risk of illness. For example, people who want to lose some weight or keep in shape cannot eat food with high calories. Moreover, for elder people, they should eat more healthy food like fewer pork and more vegetables because it can reduce the possibility of disease. People who have diabetes, or liver or kidney disease should also pay attention to food with sugar. Parents should help their children to avoid food, like nuts and seeds, hard food or sticky food. This technology is also crucial to health relevant applications.

About five years ago, we can use classification models to distinguish different classes. For instance, we can identify a dog or cat. Image classification attaches much importance in machine learning, since it is very useful and important in many fields, such as telecommunication, medical science, traffic record, security, human face recognition, multimedia, and so on. Given large number of images, it would take much time with manual classification. However, it would take just a few minutes using trained image classifiers like those listed before. It is still difficult for computers to recognize and build models since the input of the pictures are written in pixels. Thus, now there are still a large number of challenges in robotics, self-driving cars, drones, etc.

Deep learning is a powerful method that allows computational models which consist of multiple neural networks to learn representations of data with multiple levels of abstraction. Featured experiments like multi-task learning use deep convolutional neural network (DCNN). In [5], multi-task DCNN models are proposed for simultaneous categorization and pose estimation of general objects (e.g., airplane, sofa, car). Deep convolutional neural networks have made awesome breakthroughs in processing images, videos, texts and audios, whereas recurrent neural networks have made mass progress on sequential data such as text and speech.[10] We can say that deep learning is good at perception problems and very complementary. Many traditional computer vision problems have been improved a lot since the fusion of deep learning techniques, while few have applied them to food.

The current approaches mostly focus on recognition of food category based on global dish appearance without explicit analysis of ingredient composition.[4] As a result, the main goal of our project is to identify the ingredients presented in a given dish image. Similar dishes are made with almost same ingredients, just the way of making it differs. Also, the ingredients found may differ in appearance or name. So our task is to classify these ingredients correctly. For instance, in India, cheeseburger is actually a veggie-burger but in US it actually contains âĂŸbeefâĂŹ. Such ambiguities can be solved by our approach. One of methods is by comparing selected features from the image and a food database. In simple words, this process needs a huge amount of machine learning algorithm to analysis. Open source computer vision is one of the simplest programming functions which aims to be used at real-time computer vision. Moreover, SVM (support vector machines) are high level learning models with associated learning algorithms that analyze data used for classification and regression analysis.

Along with the paper, we will use Vireo-Food 172 Dataset and leverage VGG-16 model. They will be explained in the later sections.

2 LITERATURE REVIEW

Based on previous research, it is interesting that tons of works are focusing on recognizing food, but few people work on inferring ingredients from food images for the end users. In a similar study[4] about Chinese food, the author mentioned two modules: ingredient recognition and zero-shot retrieval. The first module is already very challenging because the number of food categories will be more than number of ingredients. Generally speaking, ingredient recognition is more difficult than food categorization. The size, shape and color of ingredients can exhibit large visual differences due to diverse ways of cutting and cooking, in addition to changes in viewpoints and lighting conditions. One early model we find is pairwise local feature distribution (PFD). Based upon the appearance of image patches, pixels are softly labeled with ingredient categories. The spatial relationship between pixels is then modeled as a multidimensional histogram, characterized by label co occurrence and their geometric properties such as distance and orientation[14].

There are many difficulties in recognition area so that the error rate cannot be ignored. One of the constraints is image condition. When a photo is taken in different sessions, it can cause some difference even though the scenario is the same. As a result, we have to establish a basic to compare. Still food recognition algorithm is used to extract similar pictures from a large dataset by calculating the score of each picture, however, the verification rate is still not high today. Well-constructed challenge problems support and promote research over many years, and it is our hope that the challenge will be the catalyst for substantive and necessary improvements in recognition from point-and-shoot images.[1]

Other than still images, morphable images are even harder to catch and use in food recognition. Photos are normally two dimensional when we take a picture in mobile devices or point-and-shoot camera. Variations in visual appearance and composition of ingredients show the challenges of predicting ingredients even for dishes within the same food category. Afterwards people use algorithm to simulate the process of image formation in 3D space, using computer graphics, and it estimates 3D shape and texture of faces from single images[2]. It is easy to understand that each individual has to take a photo from different angles and rotate the head to catch all features. 3D models can make the recognition rate become much higher because we can observe the dish from everywhere. This method can effectively reduce the variations and help in the correction step. Similarly, we also use this method in mathematical modeling.

Moreover, there are still some other factors which lead to variations that affect the ingredient or food recognition, such as spatial layout, feature mining and image segmentation.

2.1 Spatial layout

ingredient regions are detected by shape and texture models, where the shape is based on deformable part-based model (DPM), while the texture is based on semantic texton forest (STF). The spatial relationship is explicitly modeled such as âĂIJaboveâĂİ, âĂIJbelowâĂİ and âĂIJoverlappingâĂİ, and thus is hard to generalize.[6]

2.2 Feature mining

This step mines the composition of ingredients as discriminative patterns depend on image segmentation. Feature mining is related to parameter setting and really affect the accuracy of recognition. Surprisingly, as reported in [3], the performance is not better than of DCNN without image segmentation on Food-101 datasets.

2.3 Image segmentation

Image segment is also based on DCNN using a more advanced technique. Training labels requires manual image segmentation, which can be really a pain when it comes to foods with fuzzy composition and placement of ingredients[11].

3 EXPERIMENT DESIGN

The goal of this project is to infer food ingredients from the image provided. We converted every raw image file into an RGB value matrix as input, and a vector, whose size is the number of ingredient, as the model's output. We leveraged the VGG-16[12] model to build our own classifier. The rest of this section will explain the dataset and the models in detail.

3.1 Dataset

We leverage the Vireo-Food 172 dataset[8] for our experiment. It contains 110,241 food images from 172 categories, and annotated manually according to 353 ingredients. The food categories are popular dishes which were compiled from "Go cooking" and "Meishijie". For each category, the name was issued as keywords in Chinese to search engines. Every category has more than 100 images. The 172 categories covers eight major groups of foods, including "Vegetables" (e.g., "Fried beans"), "Soup" (e.g., "Sour beef soup"), "Bean products" (e.g., "Braised tofu"), "Egg" (e.g., "Scambled egg with loofah"), "Meat" (e.g., "Braised pork"), "Seafood" (e.g., "Spicy clams"), "Fish" (e.g., "Grilled fish"), and "Staple" (e.g., "Yangzhou Fried rice"). The image are labeled with more than 300 ingredients based on the recipes of 172 food categories. The ingredients range from popular items such as "shredded pork" and "shredded pepper" to rare items such as "codonopsis pilosula" and "radix astragali". In addition, additional labels are created for ingredients with large visual appearance.

3.2 Feature engineering

The feature engineering consists of two parts, namely to process the X (inputs) and Y (outputs).

The input fed to the model is the RGB values of an image. For every image file, we first resize it to 224*224. Then the RGB values of every pixel is read, so that the image file is converted to a matrix of shape *(224, 224, 3)*. After this we normalize the RGB values for every image matrix. We calculated the average RGB value, and subtract the original values to the mean value accordingly, and finally divided by 255, which is the max RGB value. The processed input data adds up to more than 37GB and 18GB for training set and test set respectively, which is quite demanding and posts some difficulties for us to do the training.

As for the output, because the number of ingredients is 353, so the output for every image is a vector of size 353. The final layer of the model, which will be explained further in the following subsections, is a softmax activation function, which normalize the output to a float number from 0 to 1. The index of the number serve as the identifier to the ingredient. The Vireo-Food 172 dataset provides a list of all ingredients being classified, and the index of one ingredient in the list is the same as that in the output vector. For example, if in the output list, the 5th value is 0.32, this means that the 5th ingredient in the list is predicted to be in the image with 0.32 probability.

3.3 Model design

Deep learning flourished in recent years due to availability of huge chunk of data, improved algorithms and strong computational power. Convolutional Neural Network (CNN) is a special kind of deep learning network which is responsible for recognizing visual patters directly from pixel images. The CNN are developed to extract relevant features on their own instead of using a special engineer to do this task. The depth which refers to the number of layers in CNN is an important factor to consider. Simpler CNN cannot be used to solve complex problems. This led to invention of Deep Convolutional Neural Networks (DCNN). DCNN are used in Image-recognition, classification task. There are various DCNN available nowadays. They developed in the mid 2000âĂŹs beginning from AlexNet[9], GoogleNet/Inception[13], VGGNet, ResNet[7]. Researchers always try to either modify existing architecture or come up with an altogether different new architecture to solve real world problems using Deep Learning. Nevertheless, the base architecture remains the same when we compare DCNN with simple CNN. In

CNN we have convolutional filters which are applied on input images, immediately after this we add a ReLU non-linear activation function to introduce non-linearity. After this we apply pooling to extract more relevant pixels. We even apply batch normalization just to make sure that our weights and bias donâĂŹt explode while training. The above steps are repeated several times in order to create a Deep Neural Network. After the final pooling layer of our DNN we flatten out the outputs and feed them as inputs to a Fully Connected Neural Network (FC). The loss function of our model is defined in the FC layer. The following section contains details of various layers of DCNN.

3.3.1 Convolutional filters. Convolution filters, also referred as kernels are convoluted over an input image of size height * width * channels to extract important pixels and form a feature map which is used during learning phase. The convolution operation is given by the following formula

$$FeatureMap = f(N * I)$$

where N is the convolution filter related to a feature map; I is the input image; the multiplication is a 2D convolutional operator.

In CNN at every step the output size of image decreases if padding is not provided. Suppose we have an image of size 6 x 6 and filter/kernel size of 3 x 3, then the convolution operation will output an image of size 4 x 4. This is given by a formula

$$OutputImage = ImageSize - KernelSize + 1$$

In DCNN if we keep doing this, we will soon land up having no pixel left for computation for next deep layers. Thus, we need to zero pad our input image. Other reason for padding input image is when we stride over our input image the pixels present on the edges are only visited once whereas the pixels present in center part of the image are seen frequently. In Tensorflow for conv2D library function we set the padding parameter to âĂŸSameâĂŹ. It pads the input in such a way that we get the output image as same size of input image. The size of output image is given by following formula

$$OutputImage = (ImageSize - 2 * Padding + KernelSize)/Stride+1$$

3.3.2 ReLU activation function. In CNN after applying convolution on the input image we apply ReLU function in order to introduce non-linearity in the model. Experimental results convey that ReLU performs better than sigmoid and tanh activation functions. ReLU speeds up learning process and tackles vanishing gradient problem (where weights donâĂŹt really change for lower layers during backpropagation). The ReLU function is as follows

$$f(x) = max(0, x)$$

3.3.3 Pooling. After applying ReLU we make our image go through various pooling layers. There are two types of pooling techniques used such as Average pooling and Maximum pooling. In this layer we stride over our convoluted image picking up relevant pixels and discarding other pixels. In practice Maximum pooling also termed as max pooling is used where we take only the maximum pixel in a given stride. Pooling helps in reducing computational cost as irrelevant data is dropped.

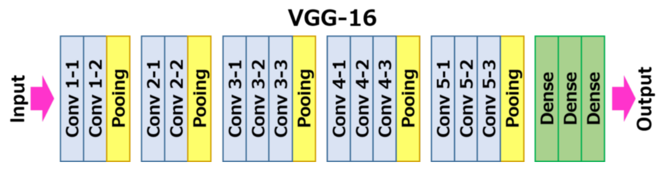


Figure 1: The architecture of VGG-16.

3.3.4 Batch normalization. In general, we normalize our input features so that our neural network train properly. For images we normalize the RGB values of our image before training. But the weights of hidden layers can explode during training process. This can be avoided using batch normalization. It does this by subtracting the batch mean and dividing by the batch standard deviation for the output of previous activation layer

3.3.5 Fully connected layers. The above-mentioned steps are repeated several times, thus stacking layers on each other, abstracting more information and features of inputs. After taking the batch normalization of the final convolutional layer we flatten out the output and feed it to a fully connected layer. The number of layers in fully connected layers depends on the inventor. The total number of neurons and output classes of the final layer of this fully connected layer depends on a particular problem statement. We again use a ReLU activation functions except for the last layer where we use Softmax activation function as we want to have a probabilistic distribution to confirm as to which class our model is predicting the input belongs to. The Softmax activation function is given by the following formula:

$$\mathbf{P}_k = \frac{e^{\mathbf{f}_k}}{\sum_j e^{\mathbf{f}_j}}$$

The summation in the above formula is along axis = 1/ along columns of the matrix.

The Loss function is given by the following formula:

$$\Gamma = -\frac{1}{N}\sum_i (log(\mathrm{p}_{y_i})) + \frac{2}{\lambda}\sum_d \sum_k (W_{d,k})^2$$

The λ is a hyperparameter for regularization which is used to avoid overfitting.

3.3.6 Architecture of VGG-16. In our project we had various options to choose between VGG-16, VGG-19, Inception, ResNet. We decide to go for VGG-16 as our dataset is less as compared to the original dataset (Imagenet) used for training the VGGNets by the inventors. Also, the complexity of VGG-16 is less as compared to others. We built our VGG-16 model from scratch following the paper [3]. Tensorflow framework was used to build our model. The architecture of our model is shown in Figure 1.

We resized our images to be of size 224*224*3 in order to maintain the kernel size of 3*3 and stride 1 for the convolutional layers. One interesting point to note here is the kernel size is fixed throughout. The reason for it is that when we stack three 3*3 convolutional layers with a stride of 1 we have same effective receptive field as 7*7 convolutional layer. Max-pooling is performed over 2*2 pixel window with a stride of 2. More interestingly in our model we are using batch normalization for the reasons stated earlier. As batch normalization concept was not discovered when VGG was originally invented. Also, dropout was not used then. In our case we are using dropout of 40% to avoid overfitting. In dropout our model randomly mutes few neurons outputs. For learning, we are using Adam optimizer as it takes advantages of both AdaGrad and RMSProp. Unlike Stochastic Gradient Descent (SGD), Adam changes its learning rate as the learning unfolds. As we were getting less accuracy on our version of VGG-16 model we tried Transfer Learning. We obtained the pre-trained model from Tensornets. Using the concept of transfer learning we changed the output layer i.e. the Softmax layer as per our 353 number of classes and trained the model again. In Tensorflow while building a model from scratch we use tf.nn.softmax_cross_entropy_with_logits_v2 function which performs softmax along with cross entropy at the same place. However, in pre-trained model we already have softmax applied so we just need a function only performs cross entropy loss function. Such a function is available in Tensorflow called tf.losses.softmax_cross_entropy. We are using this in our transfer learning model implementation.

3.4 Experiment environment

All the scripts for training and testing are written in Python 3. All experiments are ran on a Windows 10 Pro machine with 32GB of RAM and a GTX 1080 GPU.

4 RESULTS

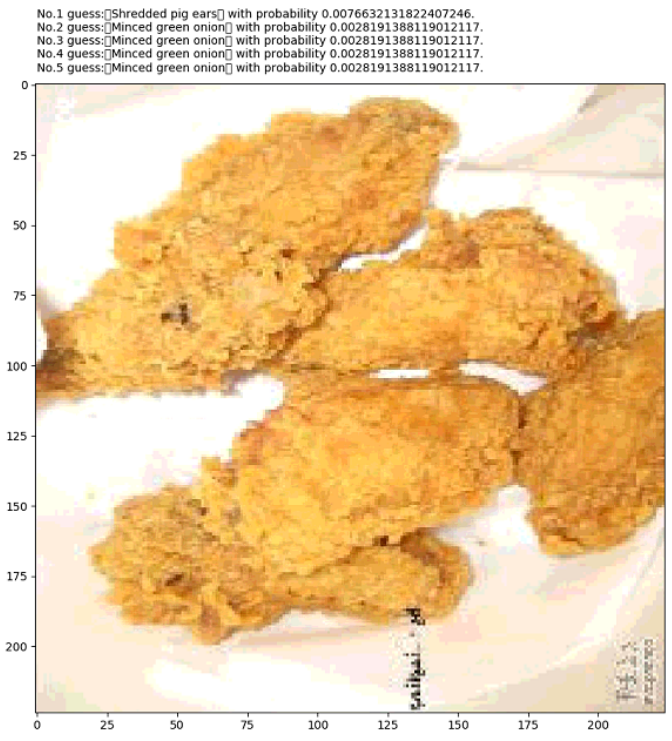
In this section, we discuss results of descriptive analysis, that includes visualizations and some exploratory models, followed by results from the transfer training model implemented to analyze the experiment. At the beginning, we used models trained from scratch which number of epochs is 50000, train_batch_size is 100 and test_batch_size is 20. The accuracy and loss outputs are shown in Figure 3 and 4 respectively.

We can see that both train and test accuracy are ranged from 5% to 20% and it is very unstable. Afterwards the test loss is 298574540000 and train loss is 724004400000 after 50000 of iterations.

In the next step to improve the accuracy, we used transfer models to train. Particularly, previous trained models are from TensorNets and we only trained the last layer to fit our data. After we adjusted the settings of some parameters, the results did not change a lot. So we would post two best experiments which only change the train_batch_size and test batch size. Learning_rate is 0.0002 and number of epochs is 100.

For the first experiment of transfer learning, train_batch_size=20 and test_batch_size=4. The accuracy and lost outputs are shown in Figure 5 and 6 respectively.

The final test accuracy after 100 iterations is 19.1% and training accuracy is 5%. On the other hand, the training loss and testing loss are similar. They are 21 and 16.3.



```
No.1 guess:▯Shredded pig ears▯ with probability 0.0076632131822407246.
No.2 guess:▯Minced green onion▯ with probability 0.0028191388119012117.
No.3 guess:▯Minced green onion▯ with probability 0.0028191388119012117.
No.4 guess:▯Minced green onion▯ with probability 0.0028191388119012117.
No.5 guess:▯Minced green onion▯ with probability 0.0028191388119012117.
```

Figure 2: Sample output.

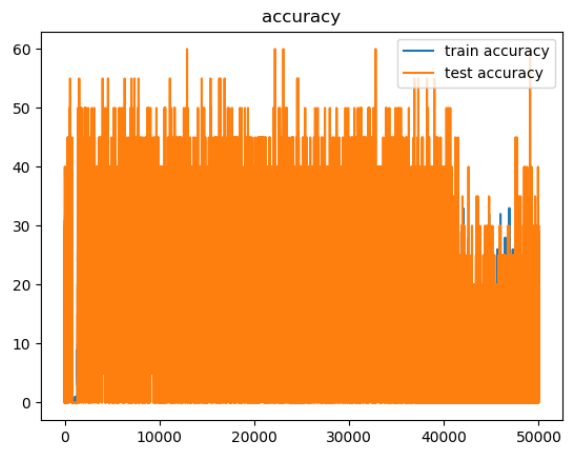


Figure 3: The accuracy by the number of epochs for the model trained from scratch.

In the second transfer learning experiment, we just changed the test_batch_size to 10. The accuracy and lost outputs are shown in Figure 7 and 8 respectively.

The final test accuracy after 100 iterations is 17.5% and training accuracy is also 5%. The training loss and testing loss are 18.1 and 13.7.

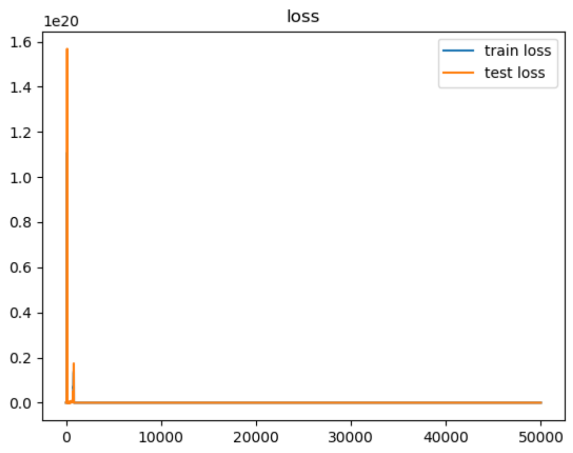


Figure 4: The loss by the number of epochs for the model trained from scratch.

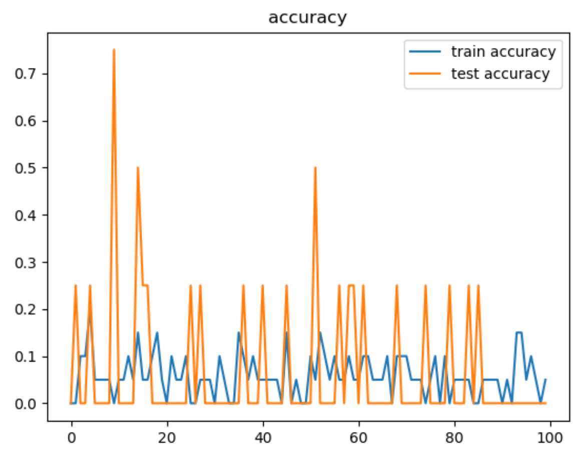


Figure 5: The accuracy by the number of epochs.

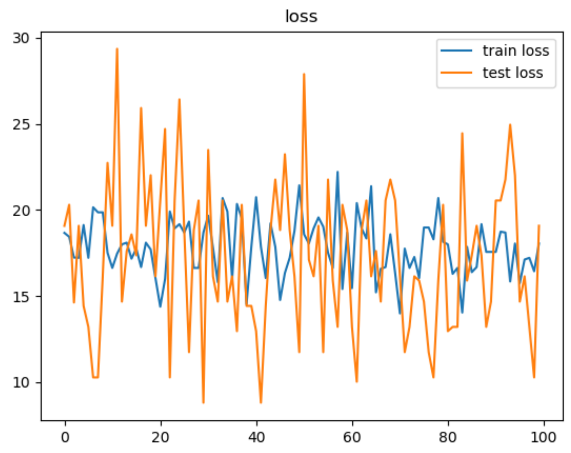


Figure 6: The loss by the number of epochs.

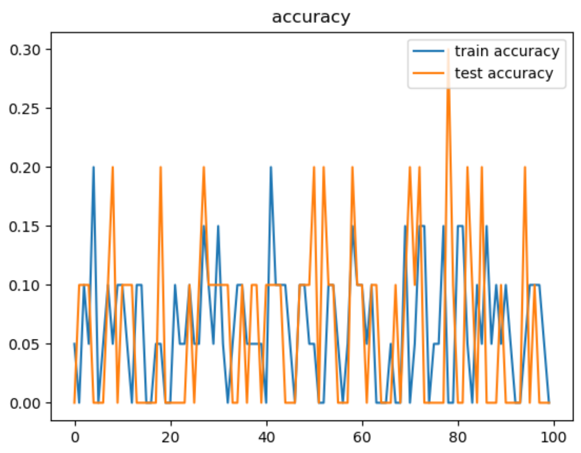


Figure 7: The accuracy by the number of epochs.

Based on the output, both accuracy are relative low compared with other experiments. There exist other factors which lead to this low accuracy. Noted that, We used pictures of height and width of 224*224 pixels, since our raw data pictures are 256*256 pixels. As a result, we probably missed some useful features when extracting from the raw images. Another reason for such low accuracy is, we chose Adam optimizer as it takes advantages of both AdaGrad and RMSProp. It can save much memory and time. Unlike other optimizers like SGD, Adam changes its learning rate as the learning unfolds.

The results from transfer learning is also not very satisfactory. The probable reason behind this are:

(1) The number of epochs may not be high enough. Although the idea of transfer learning is to just train the last layer, and thus it does not need the number of ephochs to be as high as that of if trained from scratch, 100 may still be not large enough to train a robust neural network.
(2) The batch size of training and test set are slow. In our experiment, we set it so low as we are limited on the resources we have, namely the RAM and VRAM. One solution is to run the script on aci.ics cluster, but the memory assigned to one user is also limited.

5 CONCLUSION

In this project, we tried to train a neural network to infer the ingredient from the food image fed to it. We leverage the VGG-16 model and the Vireo-Food 172 dataset, but the results are not satisfactory. We then leverage the pre-trained VGG-16 model in TensorNets and do the transfer learning to train the last layer only,

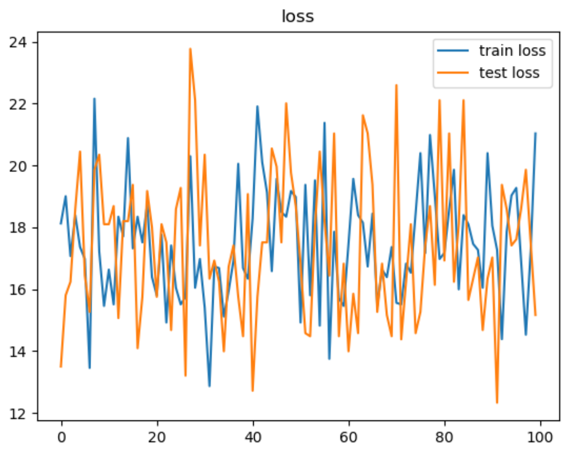


Figure 8: The loss by the number of epochs.

but the results are still not satisfactory. We discussed some possible reasons leading to this.

In conclusion, the models we developed, no matter it is trained from scratch or based on pre-trained model, are not competitive at the current stage. It may be due to the limited resources at our hands, feature engineering or the model we chose, and we will look for ways to improve this model in the future.

REFERENCES

[1] J Ross Beveridge, P Jonathon Phillips, David S Bolme, Bruce A Draper, Geof H Givens, Yui Man Lui, Mohammad Nayeem Teli, Hao Zhang, W Todd Scruggs, Kevin W Bowyer, et al. 2013. The challenge of face recognition from digital point-and-shoot cameras. In *Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*. IEEE, 1–8.

[2] Volker Blanz and Thomas Vetter. 1999. A morphable model for the synthesis of 3D faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 187–194.

[3] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. 2014. Food-101–mining discriminative components with random forests. In *European Conference on Computer Vision*. Springer, 446–461.

[4] Jingjing Chen and Chong-Wah Ngo. 2016. Deep-based ingredient recognition for cooking recipe retrieval. In *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 32–41.

[5] Mohamed Elhoseiny, Tarek El-Gaaly, Amr Bakry, and Ahmed Elgammal. 2015. Convolutional models for joint object categorization and pose estimation. *arXiv preprint arXiv:1511.05175* (2015).

[6] Hongsheng He, Fanyu Kong, and Jindong Tan. 2016. Dietcam: Multiview food recognition using a multikernel svm. *IEEE journal of biomedical and health informatics* 20, 3 (2016), 848–855.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[8] Chong-wah NGO Jing-jing Chen. 2016. Deep-based Ingredient Recognition for Cooking Recipe Retrival. *ACM Multimedia* (2016).

[9] Alex Krizhevsky, I Sutskever, and G Hinton. 2014. ImageNet Classification with Deep Convolutional Neural. In *Neural Information Processing Systems*. 1–9.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[11] Austin Meyers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin P Murphy. 2015. Im2Calories: towards an automated mobile vision food diary. In *Proceedings of the IEEE International Conference on Computer Vision*. 1233–1241.

[12] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[13] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning.. In *AAAI*, Vol. 4. 12.

[14] Shulin Yang, Mei Chen, Dean Pomerleau, and Rahul Sukthankar. 2010. Food recognition using statistics of pairwise local features. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2249–2256.