Project Report : CS 7643 Summer 2021, Group 69

Marcus Chong
mchong32@gatech.edu

Yilin Shi
yshi401@gatech.edu

Jijiao Zeng
jzeng63@gatech.edu

Srikanth Subramanian
ssubramanian91@gatech.edu

Abstract

The objective of this project is to create the best possible deep learning model that can classify a variety of food items from images. To do this, we tested the data against ResNet50, VGG, AlexNet, and Inception (in addition to adjusting the hyperparameters for each model) to determine which model performs the best. One example of a challenge we face in food recognition is that the food in images may be distorted depending on how it's cooked. We found this project to be very interesting because food is a very common part in our loves, and many people cook a lot by themselves. This paper can potentially help people to achieve healthier lifestyles, or simply offer some pleasure on a busy day.

1. Introduction/Background/Motivation

In this study, our team is aiming to experiment classical image classification algorithms based on deep convolutional neural network (e.g. VGG, ResNet50, AlexNet, Inception) for recognition of food types in images.

According to the estimates of World Health Organization, 40 percent of adults were overweight and 13 percent of the population were obese [4]. Thus, it is necessary to differentiate how much nutrition should be taken daily for a healthy live. As a result of an increase demand of balanced nutrition intake, automatic classification of food type becomes more and more critical. Yanai and Kawano (2015), reported 78.77 percent accuracy for UECFOOD100 using deep convolutional neural network pretrained on ImageNet Data [3]. Moreover, Single et al.(2016), reported 83.6 percent on the food category recognition using GoogleNet [6]. Although the successful applications were widely published, there is lacking of a systematic comparison for food recognition.

Nowadays, several state of art image classification models were developed on ImageNet. The top five models were FixResNeXt-101, DPN-131, ResNet-152, Inception and SPPNet, respectively. It is interesting to explore the performances of state of art algorithms on food type recognition.

In this experiment, we initially planned on using the dataset mention in the Image-to Recipe Translation Blog to recognize food from images and output potential recipes. Due to difficulties in collecting data and compiling the original code, we decided to change out approach and focus on Food Classification (Piazza Post with our changes: https://piazza.com/class/kni1g8lh43l3ie?cid=223_f25).

For this project, Food 101 dataset was used. The Food-101 dataset (https://www.kaggle.com/kmader/food41) consists of 101 food categories with 750 training and 250 test images per category, making a total of 101k images. The labels for the test images have been manually cleaned, while the training set contains some noise.

Based on the available information we have on the models tested, we predict that ResNet50 and Inception will outperform VGG and Alexnet. ResNet50 uses residual connections which leads to better forward and backward propagation while Inception uses multiple filters at a given layer which can potentially capture features that other models may not.

2. Approach

2.1. Data acquisition

The original blogger that describes the story is selected from one of the topics listed on Canvas:
https://towardsdatascience.com/this-ai-is-hungry-b2a8655528be

It provides a tutorial about how to built the model and comes with a GitHub repository. However it does not provide the data set and asks the readers to download the data set by themselves from a "recipe website" by "website scrapping" (script provided). At the beginning, we were trying to gather data by ourselves, however it kept saying "connection refused" and we could not download any data even by using their scripts. Also this "recipe website" is

mostly in German; none of the team members speaks German so it's hard for people to understand what's happening on this website so it is probably not a good target for our project.

Fortunately, we later found a very helpful post on piazza (@223_f7), in which the TA commented "you should avoid gathering your own data" and provided several sources to download data set directly. We finally picked the Food-101 data set:

https://www.kaggle.com/kmader/food41

The Food-101 data set has 101 categories and is well organized. It is divided into a training and a testing folder; it also comes with a meta folder containing the list of contents of the categories, training and testing images. The data set is of 5.42 GB; although it increases the running time a lot, it is quite handy to use.

2.2. Model

We use PyTorch and the models are from Torchvision.models library. We did not wrote the functions manually as in Assignment 1.

2.3. Structure of the code

First, we tried to write code similar to Assignment 1, i.e., write training and evaluation functions separately and recall them inside each epoch loop. However, when using PyTorch's built in function, (instead of manually write the functions as Assignment 1), the performance did not change. Later we realized the model get from the training function needs to be applied on the evaluation function for the current epoch (there are weight parameters that just learnt from training need to be passed onto the evaluation function), so we moved the training and evaluation function together inside the epoch loop, then it starts to give correct results.

2.4. Computation time

Environment is Anaconda in PyCharm. Default setting uses CPU, which took 32 min to run 1 epoch on the training data with AlexNet. Later the GPU with CUDA is used (CUDA and PyTorch version has to match), which reduced the computation time to 15-20 min per epoch. To further speed up, multi-threading was added; since PyTorch can support a maximum of 16 workers, worker number of 4, 8, 16 were tried, and worker = 8 seems to give the shortest time, 1.5 - 2 min per epoch, which greatly speeds up the process.

2.5. Resources

1. Assignment 1: the structure of calculating loss and accuracy, making plots on epoch number vs. performance were partially adapted from Assignment 1.

2. PyTorch official website [5] provides documentation of available Torchvision models and examples about how to apply them.

3. Kaggle website [1]: data set used in this project is downloaded from here; the website also provides description about this data set and possible usage.

4. Piazza website: discussion on final project and communication with TAs.

3. Experiments and Results

We choose four types of model to implement on the Food-101 data set and the experiment details and results are listed below. The code for running the experiment can be downloaded from:

https://github.gatech.edu/mchong32/DL_Final_Project

3.1. ResNet50

ResNet50 is a variant of ResNet model which as 48 convolutions layers along with a single MaxPool and a single Average Pool layer.

3.1.1 Data preparation

For this experiment, both the training and validation data is resized to 224x224 and Normalized to ((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) respectively. In addition, the training data is transformed with Color Jitter (brightness=0.1, contrast=0.1,saturation=0.1), Random Affine (15), Random Horizontal Flip, and Random Rotation (15).

3.1.2 Hyperparameters

The model runs with a batch size of 128 and a epoch of 10. For the learning rate, torch_lr_finder is used to find the learning rate with the steepest slope (5.11E-5) and a scheduler is implemented to reduce the learning rate by a factor of 0.1 when the performance plateaus (via ReduceLROnPlateau). The optimizer used is Adam and the criterion used to calculate loss is cross entropy loss.

3.1.3 Results

In Figure 1, the training accuracy is 90.85% and the validation accuracy is 81.16% at epoch 10. In Figure 2, the training loss is 0.359 and the validation loss is 0.708. One interesting this to note is that around epoch 5, the training accuracy is greater than the validation accuracy and the training loss is less than the validation loss. These two observations indicates that the ResNet50 may be overfitting.

Several ways to potentially solve this is to get more data, add more data augmentation (perhaps increasing the dropout), or reduce the network size.

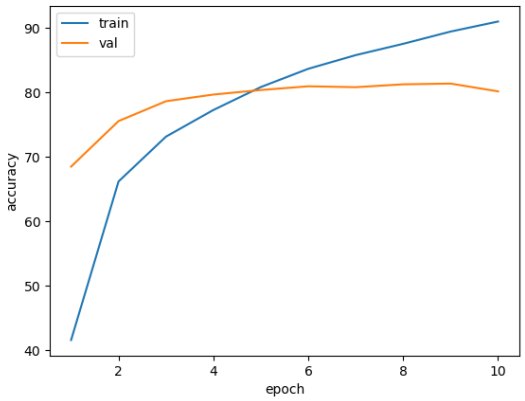

Figure 1. ResNet50 Accuracy

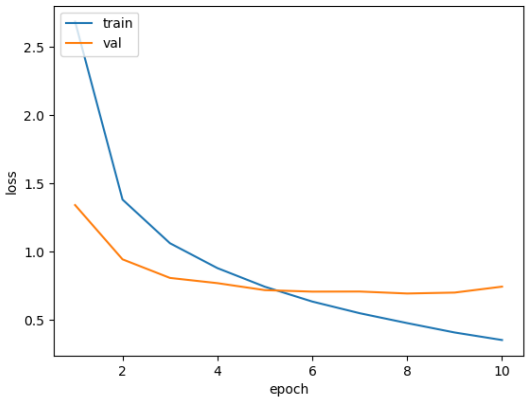

Figure 2. ResNet50 Loss

3.2. AlexNet

AlexNet is a one of the popular variants of the convolutional neural network and used as a deep learning framework.

3.2.1 Data preparation

First, the torch vision library is used to import the data set and models. The transforms library is used to transform the downloaded image into the network compatible image data set, since AlexNet model requires the input images to be size 224×224. Then the image data set is converted to the PyTorch tensor data type. To normalize the input image data set, the mean and standard deviation of the pixels data is used as per the standard values suggested by the PyTorch ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]).

3.2.2 Structure of the model

The .eval() function was used to check the model structure, and two changes were made in order to better reflect the structure of the problem. First, the original PyTorch AlexNet has a very large number of nodes in the dense layer (4096), it was reduced to 1024 to avoid overfit or having heavy losses during the training [2]. Second, the output layer is updated to out feature = 101 since our data set has 101 classes.

3.2.3 Hyperparameters

(1) Optimizer: Stochastic gradient descent (SGD) is used as an optimizer, learning rate = 0.001, momentum = 0.9. lr = 0.01 is tested but performs poorly. Optimizer Adam is tried but performed poorly.

(2) Cross-entropy is used for the loss.

(3) Batch size is set to 16. Batch size = 4 is tried but performed poorly; batch size = 32 is also tested and have slightly weaker performance than 16.

3.2.4 Results

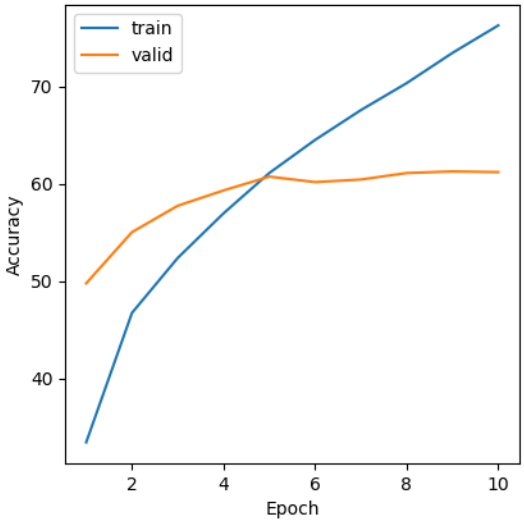

Figure 3. AlexNet Accuracy

The plots show that the accuracy of the model increased with epochs and the loss has decreased. Validation loss is still far below train loss, which could mean we might need more epochs; Validation accuracy is on the higher side than training accuracy for the first few epochs then drops, This could be caused by using a pre-trained model trained on ImageNet which contains data from a variety of classes compared to the limited 101 classes in our data set.

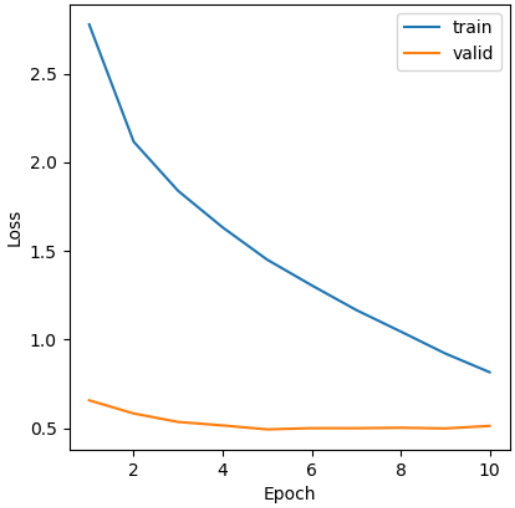

Figure 4. AlexNet Loss

3.2.5 Pre-tained model

For the AlexNet, without using pre-trained mode, I got an accuracy of 25% after 10 epochs. Then I employed the AlexNet Model provided by PyTorch as a transfer learning framework with pre-trained ImageNet weights (enable the pretrained parameter and need to download another 300 MB data). I got accuracy of 76% after 10 epochs.

This accuracy can certainly be improved if I run the training for more epochs say 50 or 100. That is far better than the AlexNet which was not using the pre-trained weights received from the ImageNet data set. It can be concluded that the AlexNet model has a very good performance when it is used as a transfer learning framework.

3.3. VGG

VGG16 was first proposed by K.Simonyan and A.Zisserman (2014) in the paper "Very Deep Convolutional Networks for Large-Scale Image Recoginition" achieving 92.7 % accuracy in ImageNet. The input size is of fixed as 224 by 224 RGB image prior to normalization.

3.3.1 Data preparation

For this experiment, both the training and validation data is resized to 224x224 and Normalized to ((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) respectively. In addition, the images were randomly cropped and flipped horizontally.

3.3.2 Hyperparameters

The model runs with a batch size of 64 and a epoch of 10. The learning rate was initiated as 0.02 and a scheduler was implemented to reduce the learning rate by a factor of 0.1 when the performance plateaus (via ReduceLROnPlateau). The optimizer used is SGD and the criterion used to calculate loss is cross entropy loss.

3.3.3 Results

Firstly, The VGG16 was pretrained on ImageNet dataset and then fine-tuned last layer for food recognition. Model was optimized with stochastic gradient descent (SGD) with 0.001 learning rate, momentum = 0.9, 32 batch size and 10 epochs. The results indicated that the model was significantly under fitting. After 3 epoches, training loss and validation loss were converged while validation loss was much lower than training loss. Thus, two additional fully connected layers were introduced and learning rate was set as 0.02. After 4 epochs, accuracy was stayed as 0.65 for both training set and validation set, respectively. However, the loss plot still suggested the model was still slightly under fitting.

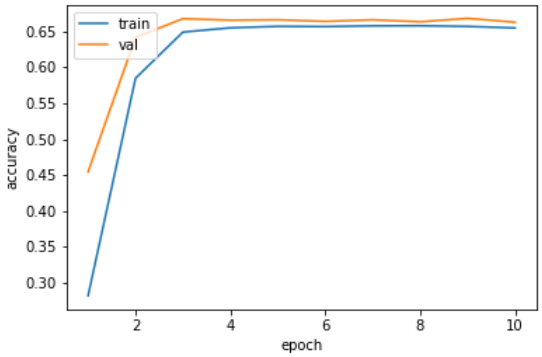

Figure 5. VGG16 Accuracy

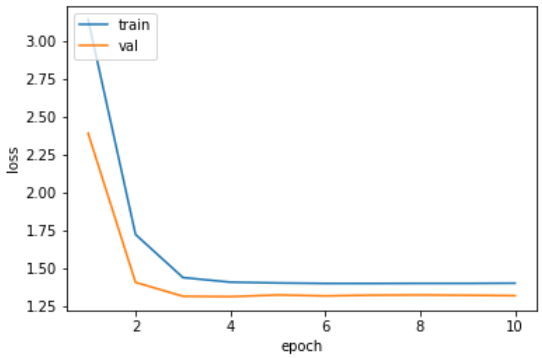

Figure 6. VGG16 Loss

3.4. Inception-v3

Inception Architecture was introduced by Google and was designed to perform well even under a stricter con-

straint for memory and computational cost. The core tenet followed the principle of going wider rather than deeper by having parallel filters with different strides at a level. The approach reduced the number of parameters significantly as compared to other more deeper models like AlexNet and VGG. Additionally the model also helps capture variations in an image where the same core object might be present such as the same dog species in different environment and sizes.

3.4.1 Structure of the model

For this experiment, the inception-v3 model in PyTorch was chosen. V3 is a subsequent improvement to the original inception model and uses an auxiliary classifier to improve convergence. The input images were reshaped to a size of (299, 299) as expected by the model. In addition, transformations like Random Affine (15), Random Horizontal Flip, and Random Rotation (15) were also performed on the input to train the model for distortions. Since the model used auxiliary layer, loss = output$_loss + 0.4 * aux_output_loss$.

3.4.2 Hyperparameters

The model was run for 10 epoch and the output of the last layer was fed to a softmax layer to get the probability distributions for the 101 food labels. Loss was then calculated using cross entropy. The model was evaluated using SGD and Adam as the optimizer to evaluate back propagation strategies. Additionally, the model was also tested with and without Pre-training to determine of impact on transfer learning.

3.4.3 Results

3.4.4 Non Pre-trained Model with SGD and Adam

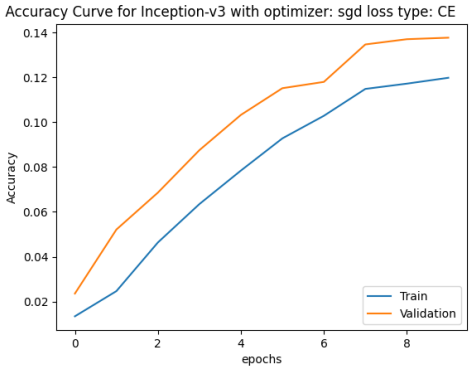

Figure 7.

Without Pre-training, the model's performance was poor with both optimization strategies. Even with 74k Training samples, the model performed poorly during validation

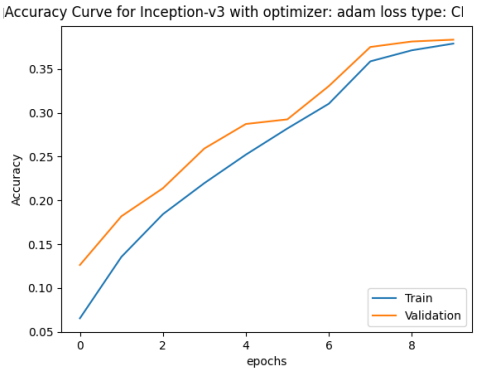

Figure 8.

across different classes. The transformations in texture and shapes couldn't help the model. This indicates lack of sufficient data for performing classification and importance of transfer learning.

3.4.5 Pre-trained Model with SGD

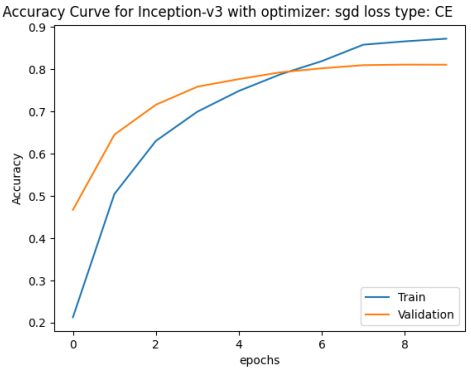

Figure 9.

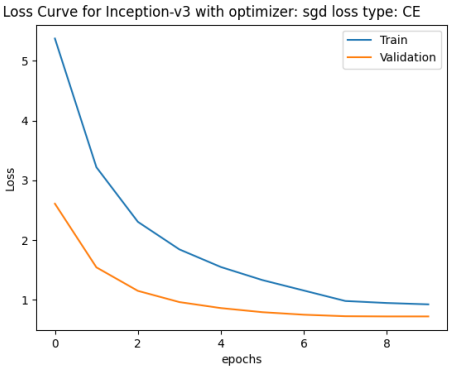

Figure 10.

In this case, the model performed much better and was

to hit an accuracy of 81.10%. Transfer learning from training on ImageNet helped the model capture the feature more effectively. After some fine tuning, the model produced the best output for a learning rate of 0.01 and a regularization of 0.005. A higher value for learning rate affected convergence and a lower value for regularization caused over fitting as penalization was insufficient given the large number of parameters.As observed in the curves, no overfitting was observed across epochs. So pre-training along with transformations helped the model learn both textures and shapes to achieve good accuracy

3.4.6 Pre-trained Model with Adam

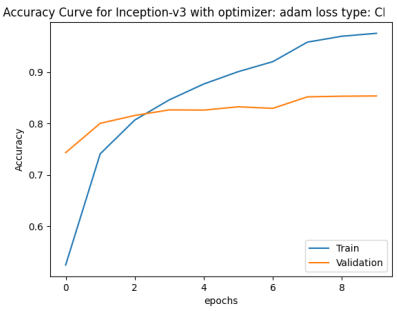

Figure 11.

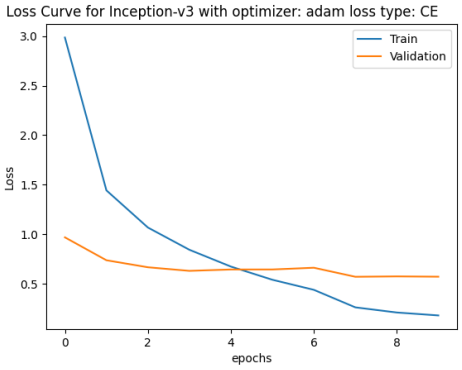

Figure 12.

With Adam as optimizer, the model performed even better and achieved a 4 percent bump over SGD to get an accuracy of 85.34%. This can be attributed to the fact that Adam has a dynamic learning rate for each weight and hence is able to adjust the weights better. beta1 value of 0.85 and be beta2 value of 0.99 provided the right gradient statistics for the model to converge. Additionally lowering the learning rate to 0.0001 helped avoid overfitting. In general, the model didn't over fit in this case though a slight dip could be observed in both training and validation accuracy during epoch 6. This suggests that the data in that batch probably introduced more noise in the training data an subsequently also impacted the validation accuracy. It can also be observed that the validation loss for epoch 6 also shows a small bump. It can also be observed that the model starts with better validation accuracy than SGD and this shows the efficacy of having dynamic learning for each weight. A slightly reduced values for beta1 and beta2 than the typical values indicate that a faster decay rate for the first and second order gradient movement helps model. Overall the dynamic weighting across the output and auxiliary layers helped the model achieve high accuracy.

4. Conclusions and Future Work

As hypothesized, Inception and RestNet provided the best accuracy and performance. VGG and AlexNet with deeper networks with large number of parameters tend to be computationally expensive and provide sub-optimal performance. Inception provided the best performance while using lower number of parameters with the main reason being parallel filters with different strides that both reduce computational cost and help capture varying feature sizes with multiple strides across multiple images for the same object. For potential future work and what the research community could possibly focus on to make improvements in the direction of our project's topic: (1) we will try to apply the trained model on images that it never saw before, for instances, images downloaded from internet, and see if the model can predict correctly, as we have the future vision that the users from internet can just upload their photo of food and get identified by our model. Images from internet users might be blurred or in wrong dimension, we are thing about performing some pre-treatments on the target images before feeding into our model. (2) We are also trying to train the model on more categories of food and go beyond the 101 classes in Food-101 data set. (3) Further more, we could build a database that connects the recipes with food, so each time a food is identified, recipes that make use of this food will be pulled from the database and presented to the user. We would love to keep improving our model and adding extra capabilities so it will be helpful to people and improve their health and quality of life. It would also be an interesting work to combine the residual propagation from ResNet and multiple filters from Inception model to check if it's possible to improve accuracy by combining the best of both worlds.

5. Work Division

Delegation of work among team members is provided in Table 1.

References

[1] Kaggle. https://www.kaggle.com/kmader/food41. 2

[2] Avinash Kappa. Multiclass food classification using tensorflow. 3

[3] Y. Kawano and K. Yanai. Foodcam: A real time food recognition system on a smartphone, 2015. 1

[4] World Health Organization. About diabetes, 2014. Archived from the original on 31 March 2014. Retrieved 4 April 2014. 1

[5] PyTorch. https://pytorch.org/vision/stable/models.html. 2

[6] Yuan L. Single, A. and T. Ebrahimi. Food/non-food image classification and food categorizationusing pre-trained googlenet model. *Proceedings of the 2nd international workshop on multimedia assisted dietary management (MADIMA 2016)*, 16:3–11, 2016. 1

Student Name	Contributed Aspects	Details
Marcus Chong	Implementation and Analysis	Trained the data set with ResNet50 model and analyzed the results; contribute to abstract section.
Yilin Shi	Implementation and Analysis	Trained the data set with AlexNet model and analyzed the results; contribute to approach, future work section, and sort the references.
Jijiao Zeng	Implementation and Analysis	Trained the data set with VGG model and analyzed the results; contribute to introduction section.
Srikanth Subramanian	Implementation and Analysis	Trained the data set with Inception model and analyzed the results; contribute to conclusion section.

Table 1. Contributions of team members.