Food-101 Deep Learning Classification

Yifei Wang
Department of Data Science
University of California San Diego
yiw014@ucsd.edu

Rui Zhang
Department of Data Science
University of California San Diego
r2zhang@ucsd.edu

Abstract

In this paper, we classified 20 food categories in the Food-101 image recognition dataset. We show that CNN and RNN, the deep learning technique for general object categorization, can solve this task accurately. We employed four different general deep learning neural network categories for the experiments: (1) Baseline CNN, (2) Custom CNN, (3) Transfer Learning for VGG16, and (4) Transfer Learning for ResNet18. Using all four network architectures, we studied the effects of training, validation, and testing. As a result, we attained great accuracy in the three general approaches, with 36.66%, 47.48%, 74.40%, and 71.98% accuracy, respectively. Lastly, the weight maps and feature activation maps were thoroughly analyzed, providing the users of the model with better hints on what's really happening under the hood. We find that while adding extra convolutional layers, changing dropout ratio, and fine-tune hyperparameters of CNN could elevate the model performance, transfer learning by fine-tuning Vgg16 and ResNet18 could greatly improve the image recognition model accuracy and with fewer epochs.

1 Introduction

With the advent of deep neural networks, the field of computer vision has been revolutionized. The combination of deep networks with convolutional feature extractors has drastically improved the accuracy when doing object classification. In this paper, we will use 20 categories in the Food-101 as the image recognition dataset. The Food-101 dataset contains some images that were not cleaned and of intense colors and wrong labels and for each food class, 250 manually reviewed test images are provided. Many interesting computer vision and deep learning work has been done using Food-101. For instance, Wang and his colleagues utilized vision-based and text-based technologies on Food-101 to implement deep neural network recipe recognition and inspire their further application of a real application for users to identify recipes [4]. For our purpose, we choose 20 categories of food and perform neural network classification on the selected food classes.

Generally speaking, the more complicated the network architecture is, the better the network could perform when classifying complex objects. However, there are only a few theories out there telling people what is the best design for a convolutional neural network model. Designing and training a good model is still a challenge to us as it demands a significant amount of time and resources to find the optimal solution. Luckily, when designing a neural network model to perform classification tasks, we can leverage the method of transfer learning: the model doesn't have to be trained from scratch; instead, we can just directly reuse the pre-trained state-of-art models that have been extensively studied by researchers. In most cases, one only needs to train the last several layers of the pre-trained model to gain a pretty decent accuracy. Presenting the outcome of using the ReNnet and Vgg model, we demonstrate the effectiveness and power of transfer learning. We find that while adding extra convolutional layers, changing dropout ratio, and

fine-tune hyperparameters of CNN could elevate the model performance, transfer learning by fine-tuning Vgg16 and ResNet18 could greatly improve the image recognition model accuracy and with fewer epochs.

2 Related Work

Image recognition has been a heated topic in the deep learning area. In Dive into Deep Learning, the authors describe Networks Using Blocks (VGG) as the designed neural network architectures where repeating blocks represent the patterns of layers [4]. A VGG block is made up of a series of convolutional layers followed by a spatial downsampling maximum pooling layer. VGG is more computationally heavy than AlexNet, and it allows for a more efficient design of neural networks.

Kaiming He and his colleagues present a residual learning approach (ResNet) for training networks that are far deeper than previously utilized networks, and they provide proof for how residual networks with reference to previous layers would be easier to optimize and the increased depth of the deep learning framework could also contribute to higher accuracy for image recognition [1]. Instead of hoping that stacking more layers could finish the classification task with higher accuracy, they resort to making the layers fit a residual mapping.

3 Models

3.1 Baseline CNN Model Description

For our baseline CNN Model, we use the cross-entropy loss function as the loss function and Xavier Initialization to train the CNN model for 10 epochs. And we use the Adam optimizer with a learning rate of 0.001 as our objective function.

The basic building blocks of the neural network architecture is a first convolutional layer with 64 out channels and a kernel size of 3, followed by a Batch Normalization layer and ReLU activation layer. A second convolutional layer with 128 out channels and a kernel size of 3, followed by a Batch Normalization layer and ReLU activation layer. A max pool of kernel size 3 is then added. A third convolutional layer with 128 out channels, stride 2, and a kernel size of 3, followed by a Batch Normalization layer and ReLU activation layer is then added on top of the max pooling. The baseline also includes an adaptive average pooling layer of output size 1 and two fully connected layers with dropout.

3.2 Custom CNN Model Description

To improve on the baseline model accuracy, we first experiment with different neural network architecture designs by adding two extra convolutional layers. In CNN, models with deeper architecture would often work better. We find out that with two extra convolutional layers, we could increase the model accuracy by increasing the depth of the model. We test different positions to add convolutional layers: we first add extra convolutional layers after conv2 by stacking 2 64x64x3x3 layers. Yet the training and validation loss seems not satisfactory. We then add extra layers after conv4 in the baseline models, and we also attach an extra layer of max pooling so that we could amplify some important features the model looks for.

Secondly, we investigate how dropouts could help the model generalize to the unseen data. Adding dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs and it would be less likely to overfit. We compare the difference in training/validation loss and accuracy between adding dropout with p=0.2 and adding dropout of p=0.5. We also experiment with where we should add the dropout: between the first few layers, between the middle layers, or between the final layers. After careful implementation, we figured out that the model would work best if we could add dropout with p=0.2 between conv1 and conv2, and add dropout with p=0.5 between fc1 and fc2. The reasoning is that the dropout of the previous layers should mute fewer connections between nodes than the dropout in the latter layers of the neural network.

We also try different epoch numbers. We first observe how our custom model would behave by

running 100 epochs. Although the training accuracy reaches 96%, the validation accuracy is only 33.26%, which suggests serious overfitting. We then limit the epoch number to be 30 and observe a significant increase in the last epochs validation accuracy and a drop in the last epochs validation loss. Large epoch numbers would easily cause the model to overfit the training portion of the data and therefore lead to poor generalization; small epoch numbers would, on the other hand, underfit the model and therefore could not give a proper representation of the Food101 dataset.

Finally, we propose a 6-layer design for the custom model, with six convolutional layers and two maximum pooling layers in between, all of which are activated by the ReLU activation function. The loss function we use is cross-entropy loss evaluation, the optimization method is Adam Optimizer with a learning rate of 0.001. Before testing test accuracy with the holdout set, the model is fine-tuned with the best hyperparameters. We propose this architecture in place of the baseline model to address the model's overfitting problem. Using this method, we were able to reach a test accuracy of 47.48%, an increase of 10.82% from the baseline model. And we include a table describing the architecture of our custom CNN model in the below Table1.

Table 1: Custom CNN Model Architecture Demonstration

Layer	**Dimension**	**Activation**
conv1	3x64x3x3	ReLU
conv2	64x128x3x3	ReLU
conv3	128x256x3x3	ReLU
conv4	256x256x3x3	ReLU
conv5	256x512x3x3	ReLU
maxpool	512x512x3x3	ReLU
conv6	512x128x3x3	ReLU
maxpool	128x128x3x3	ReLU
fc1	128x128	ReLU
fc2	128x20	None

3.3 Vgg16 Discussion

Vgg network was developed in 2014 by the Visual Geometry Group at Oxford University [2]. Simonyan and Ziserman experimented with various architectures and they found that several layers of deep and narrow convolutions (3×3) were more effective than fewer layers of wider convolutions [2]. It's much deeper than previous convolutional neural networks, and outperformed other models at the time .

A Vgg block is made up of a series of convolutional layers followed by a spatial downsampling maximum pooling layer. VGG allows for a more efficient design of neural networks. It is novel in that it proposes Networks Using Blocks, so that it could now use blocks as repeating patterns of layers [4]. The architecture is composed of a series of uniform convolutional building blocks followed by a unified pooling layer, where

1. All convolutional layers have 3 by 3 kernel size filters with a stride value of 1 and a padding value of same
2. All pooling layers have a 2x2 pool size and a stride value of 2.

3.4 ResNet18 Discussion

In this section, the basics of resnet will be reviewed. In addition to that, the architecture of the resnet that we used for this assignment is also discussed.

3.4.1 The problem of vanishing gradient

There is one question that is constantly raised when designing a convolutional neural network: Can we build a deeper network for the problem we are working on? Indeed, the deeper our network is, the better our network will be in terms of learning meaningful representation of complex data. But one commonly occurring phenomenon usually prevents the deep learning researcher from doing so, and that problem is called vanishing gradient.

As the network backpropagates the error signal back to the earlier layer, the error signal is multiplied by the weight matrix at each step; Thus the gradient can decrease exponentially to zero, leading to a vanishing gradient phenomenon that prevents the early layers from learning.

The resnet architecture mitigates this issue to some extent; This is one of the reasons why resnet is so powerful compared with other models when handling complicated tasks.

3.4.2 Novelty of Residual Neural Networks

To deal with the problem of vanishing gradient, He et al. created a shortcut that allows the gradient to be directly backpropagated to earlier layers. These shortcuts are called skip connections. Another benefit of the skip connections is that they allow the model to learn an identity function, which ensures that the layer will perform at least as well as the previous layer.

Just like the inception model(though not implemented in this PA), the residual model consists of normal building blocks(like conv2d, or batch normalization), and residual blocks where the skip connection comes into play.

3.4.3 Detail regarding the architecture of ResNet

Below is the table documenting the architecture of the ResNet model that we used in this paper.

Table 2: ResNet Model Architecture Demonstration

Layer	Dimension
conv1	3×64×7×7
bn1	3×64×7×7
relu	3×64×7×7
maxpool	3×64×3×3
Residual Block1	64×64×3×3

Residual Block2	64×128×3×3
Residual Block3	128×256×3×3
Residual Block4	256×512×3×3
Average pooling	256×512×1×1
Fully Connected Layer	512×20

4 Experiments

4.1 Baseline CNN Model Loss and Accuracy

First of all, we build a baseline model for the classification and achieve 36.66% accuracy, which is way better than the random guess of 5%. The details of the model have been discussed thoroughly in the previous section. One can see the performance boost by comparing other models to this baseline.

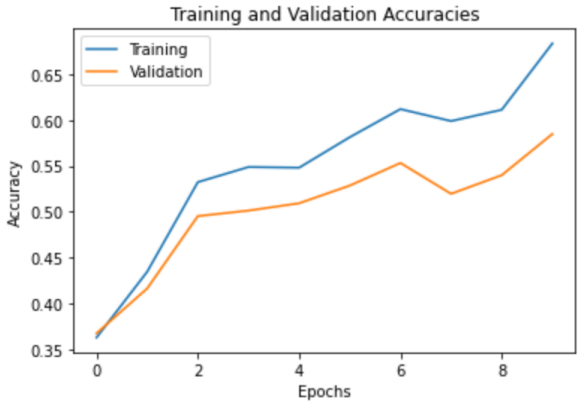

Figure 1: Training/Validation Accuracy for a Baseline Model

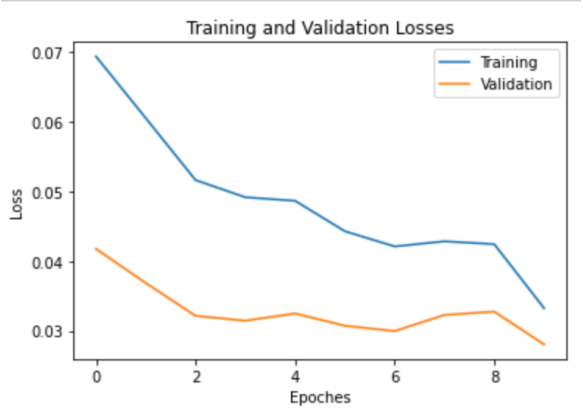

Figure 2: Training/Validation Loss for a BaseLine Model

4.2 Custom CNN Model Loss and Accuracy

(a) Other Custom Model

CNN Model: adding convolutional layers

To improve on the baseline model accuracy, we first experiment with different neural network architecture designs by adding two extra convolutional layers. In CNN, models with deeper architecture would often work better. We find out that with two extra convolutional layers, we could increase the model accuracy by increasing the depth of the model. We test different positions to add convolutional layers: we first add extra convolutional layers after conv2 by stacking 2 64x64x3x3 layers. Yet the training and validation loss seems not satisfactory. We then add extra layers after conv4 in the baseline models, and we also attach an extra layer of max pooling so that we could amplify some important features the model looks for.

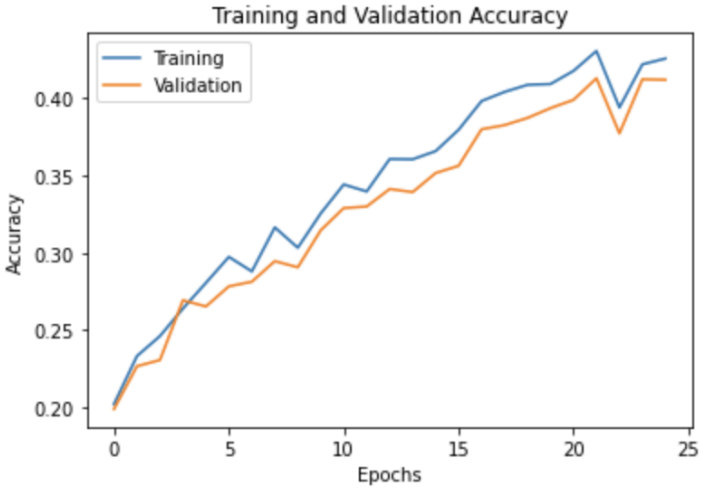

Figure 3: Training/Validation Accuracy for adding layers CNN

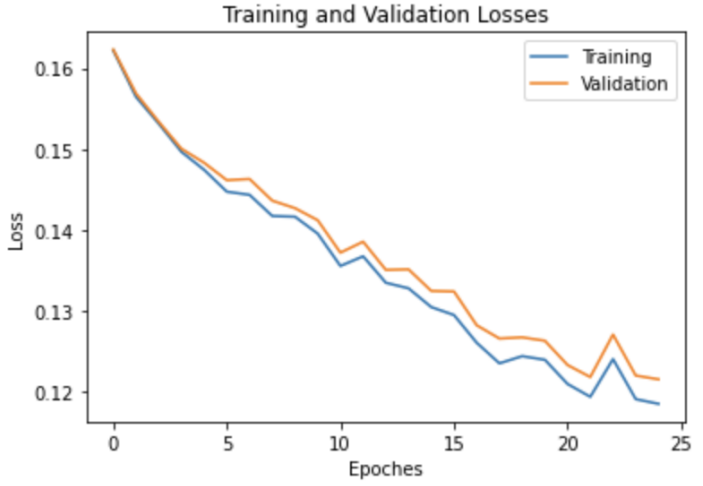

Figure 4: Training/Validation Loss for adding layers CNN

CNN Model: change dropout

We then investigate how dropouts could help the model generalize to the unseen data. Adding dropout has the effect of making the training process noisy, forcing nodes within a layer to

probabilistically take on more or less responsibility for the inputs and it would be less likely to overfit. We compare the difference in training/validation loss and accuracy between adding dropout with p=0.2 and adding dropout of p=0.5. We also experiment with where we should add the dropout: between the first few layers, between the middle layers, or between the final layers. After careful implementation, we figured out that the model would work best if we could add dropout with p=0.2 between conv1 and conv2, and add dropout with p=0.5 between fc1 and fc2. The reasoning is that the dropout of the previous layers should mute fewer connections between nodes than the dropout in the latter layers of the neural network.

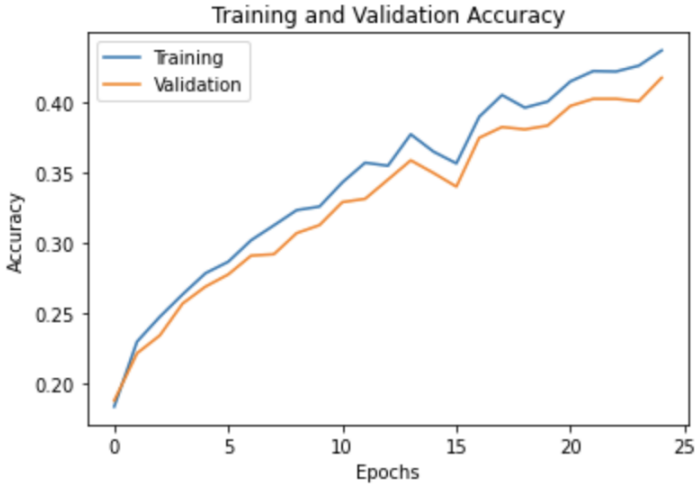

Figure 5: Training/Validation Accuracy for Change Dropout CNN

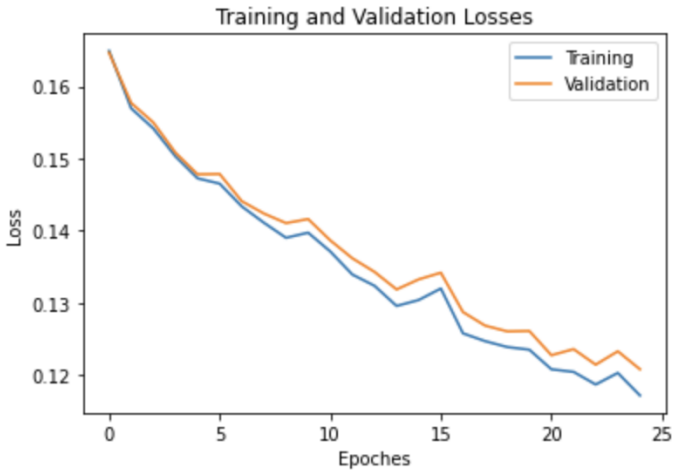

Figure 6: Training/Validation Loss for Change Dropout CNN

CNN Model: different epoch number

We also try different epoch numbers. We first observe how our custom model would behave by running 100 epochs. Although the training accuracy reaches 96%, the validation accuracy is only 33.26%, which suggests serious overfitting. We then limit the epoch number to be 30 and observe a significant increase in the last epochs validation accuracy and a drop in the last epochs validation loss. Large epoch numbers would easily cause the model to overfit the training portion of the data and therefore lead to poor generalization; small epoch numbers would, on the other hand, underfit the model and therefore could not give a proper representation of the Food101 dataset.

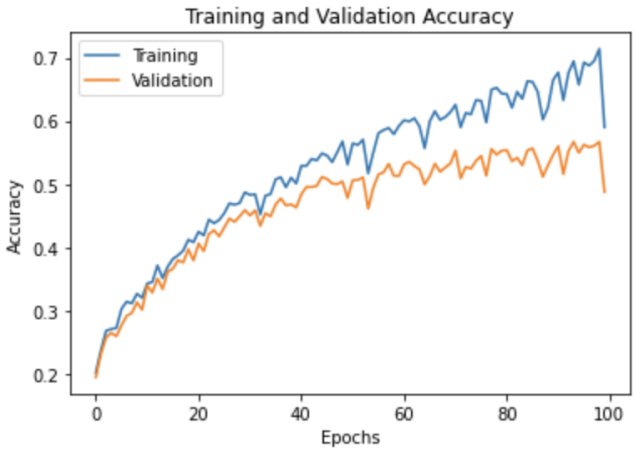

Figure 7: Training/Validation Accuracy for Other Model 100 epoch

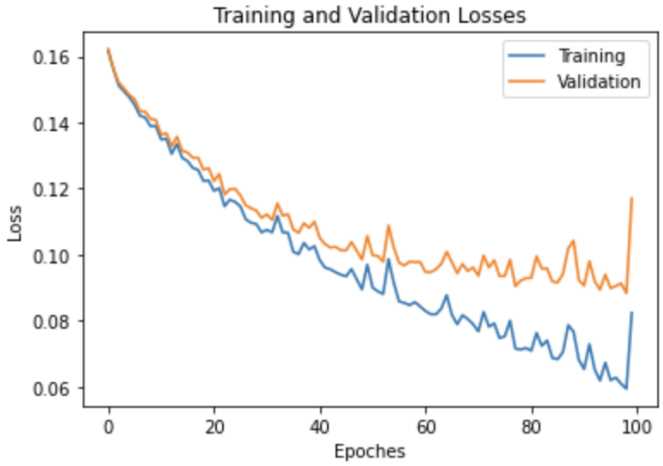

Figure 8: Training/Validation Loss for Other Model 100 epoch

To sum up, the performance of our CNN experimented models, we include the experimentation name and the validation accuracy in the below table 3:

Table 3: Custom CNN Model Performance

Experimentation name	**Validation Accuracy**
adding layers CNN	38.21%
change dropout CNN	34.98%
100 epoch CNN	33.26%
Final custom CNN Model	47.98% (test accuracy)

(b) Final Custom Model

For our final model, we propose a 6-layer design for the custom model, with six convolutional layers and two maximum pooling layers in between, all of which are activated by the ReLU activation function. The loss function we use is cross-entropy loss evaluation, the optimization method is Adam Optimizer with a learning rate of 0.001. Before testing test accuracy with the holdout set, the model is fine-tuned with the best hyperparameters. We propose this architecture in place of the baseline model to address the model's overfitting problem. This model was thoroughly described in the previous **3.2 Custom CNN Model Description** section of this paper.

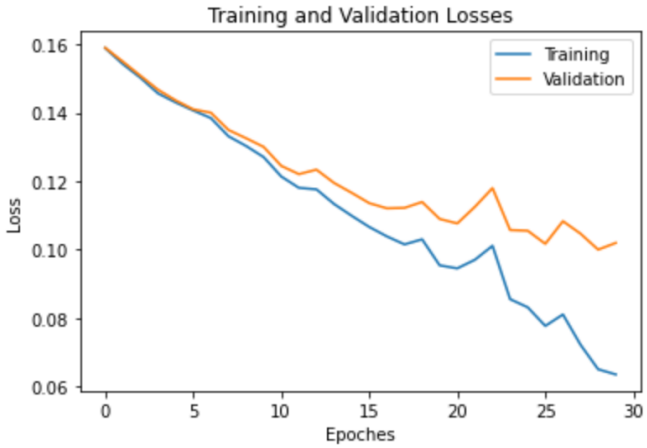

Figure 9: Training/Validation Loss for Final Custom Model

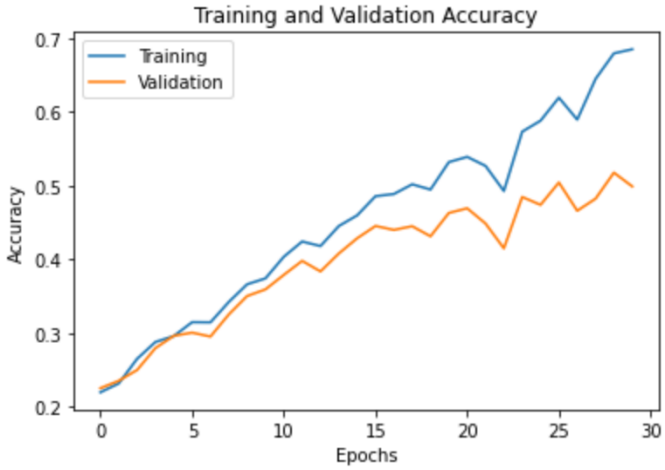

Figure 10: Training/Validation Loss for Final Custom Model

Using this method, we were able to reach a test accuracy of 47.48%, an increase of 10.82% from the baseline model.

4.3 Resnet model and Vgg model

ResNet Model Experimentation

For the resnet model, we experimented with different transfer learning paradigms. In the first paradigm, we only retrained the parameters in the fully connected layer, which means, due to a drastic reduction in the number of trainable parameters, our training will be much easier and therefore much faster.

The second paradigm, however, is to fine tune the whole pretrained model. Although the training process is much slower than we did in paradigm 1, it turns out that this is still faster than training from scratch, and usually has higher accuracy than a situation where we only fine tune the fully connected layer. The reason could be, for example, transfer learning tends to perform well when the source domain is close to the target domain. If the two domains are quite different, the transfer learning could still be useful but one should fine tune more layers in the pretrained network.

The third paradigm is to remove certain layers or building blocks of the pretrained model. The reason could be that, for instance, we believe that the model that was built to fulfill the original task is not 100 percent compatible with the current task, or the original task is much more complicated than the current task. Then it might be a good idea to remove certain layers in the deep neural network.

Since we are comparing the impact that changing the model architecture will have on the classification performance, the structure or the topology of our model is our main focus here. Nevertheless, we did find that tuning the learning rate and optimizer helps boost the performance a lot, which we will present in table 4.

Vgg Model Experimentation

For Vgg model experimentation, we experiment with different learning rates and different layers to freeze/unfreeze. We first train the Vgg freeze model where we keep the parameters in the vgg16_bn Pytorch pre-trained model and we only modify the classifier by changing the number of output channels to 20, which is the number of food classes in our Food-101 dataset. We then unfreeze the Vgg16 model and re-train the parameters in each layer to compare the difference in the model performance. The loss function we use is the cross-entropy loss function, the optimizer we use is Adam optimizer, and we experiment with different learning rates and different numbers of epochs.

For the Vgg Freeze model, we freeze all layers before the classifier, and we only modify the classifier by changing the number of output channels to 20. Our training is easier and much faster compared to the Vgg Fine Tune model where we had to wait to change all the parameters in the model. Then we first use Adam optimizer with a learning rate of 0.01 to run 25 epochs on the dataset. We notice that the learning rate is too large, so the model would not converge. We then change the learning rate to be 2e-4, which is too small and the model does not seem to converge yet. Finally, we set the learning rate to be 5e-4 and an epoch number of 10 to curtail overfitting and obtain an acceptable improvement on the Vgg Freeze model performance (68.62% validation accuracy).

For the Vgg Fine tune model, we unfreeze the parameters in the model and we modify the classifier by changing the number of output channels to 20. We first use Adam optimizer with a learning rate of 0.01t. We notice that the learning rate is too large, so the model would not converge. We then change the learning rate to be 1e-4, which is also too big and the model does not seem to converge yet. Finally, we set the learning rate to be 1e-6 and an epoch number of 25 to curtail overfitting and obtain an acceptable improvement on the Vgg Freeze model performance (75.6% validation accuracy).

We include a model performance summarization chart below in Table 4.

Table 4: ResNet and Vgg Model Performance

Changes being made	Validation/Test Accuracy
Resnet: with all layers fine tuned	71.98%
Resnet: all fine tuned but with high learning rate	44.21%
Resnet: fine tune the fully connected layer	66.78%
Resnet: fine tuned all but with high learning rate	50.3%
Resnet: fine tuned with 1 residual block removed. (high learning rate)	53.12%
Resnet: fine tuned with 2 residual blocks removed. (high learning rate)	51.6%
Resnet: fine tuned with 1 residual block removed. (good learning rate)	67.43%
Vgg: all fine tuned good learning rate	74.40%
Vgg: freeze the feature extraction part	68.62%

For Clarification, we include the training/validation accuracy/loss plots for each ResNet/Vgg model we experimented with in the below figures.

Resnet experiment: Fine tune all the layers.

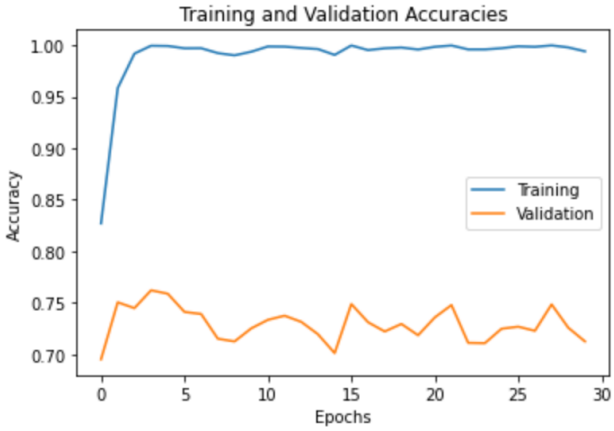

Figure 11: Training/Validation Accuracy for ResNet Fine Tune

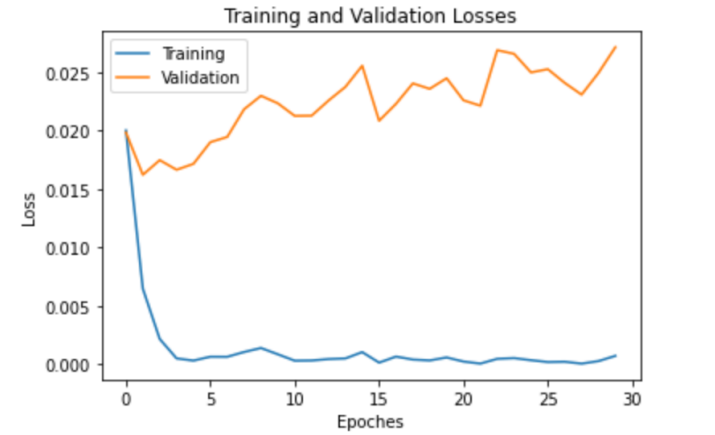

Figure 12: Training/Validation Accuracy for ResNet Fine Tune

ResNet experiment: Freeze all convolutional building blocks

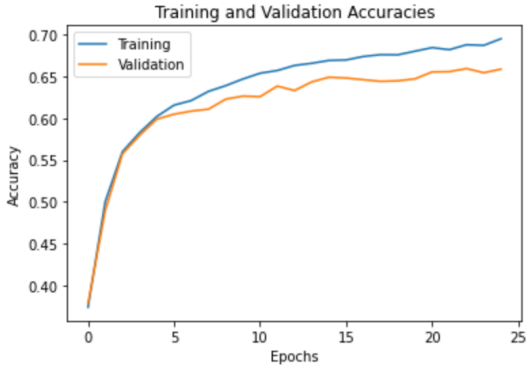

Figure 13: Training/Validation Accuracy for ResNet Freeze

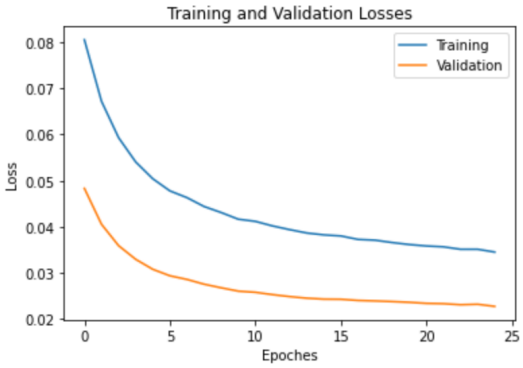

Figure 14: Training/Validation Accuracy for ResNet Freeze

Resnet experiment: Remove 1 residual block and fine tune all.

As one can see from the graphs, overfitting occurred.

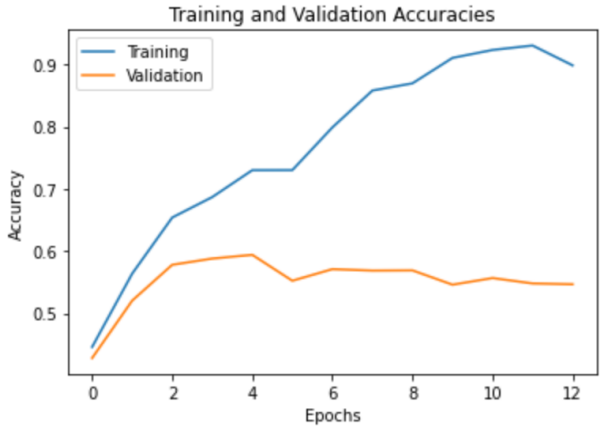

Figure 15: Training/Validation Accuracy for ResNet remove 1 block

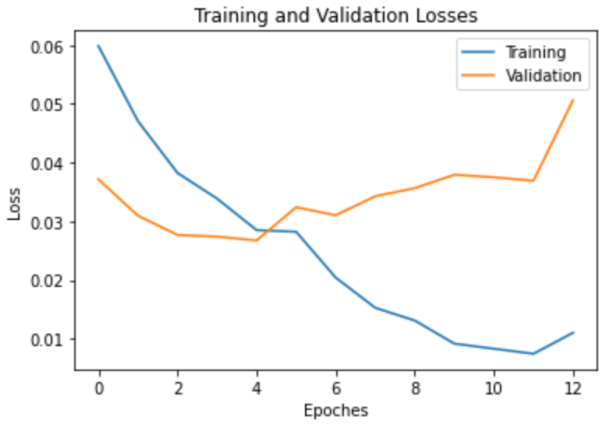

Figure 16: Training/Validation Accuracy for ResNet remove 1 block

Resnet experiments: Remove two residual blocks and fine tune the rest.

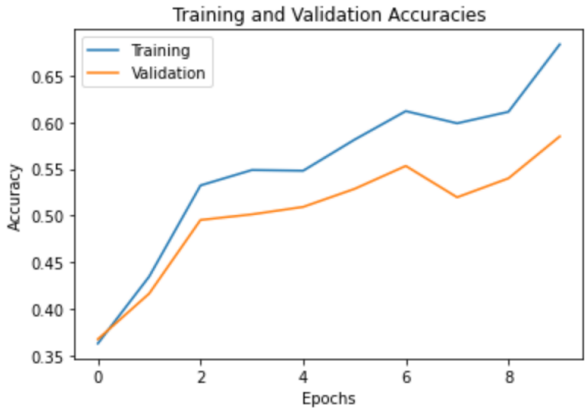

Figure 17: Training/Validation Accuracy for ResNet remove 2 blocks

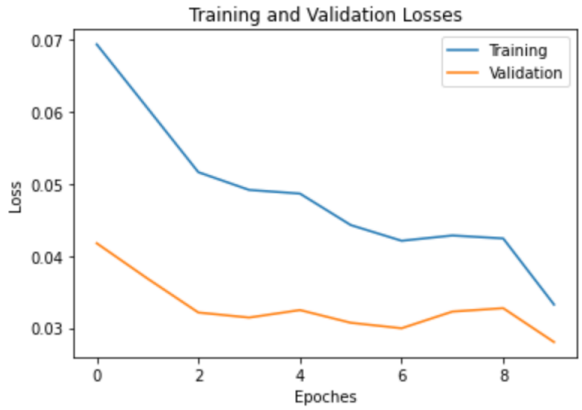

Figure 18: Training/Validation Accuracy for ResNet remove 2 blocks

Vgg experiments: Fine tune all the layers

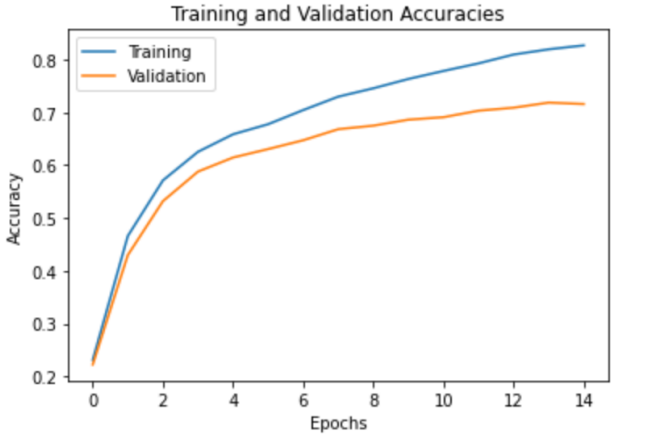

Figure 19: Training/Validation Accuracy for Vgg Fine Tune

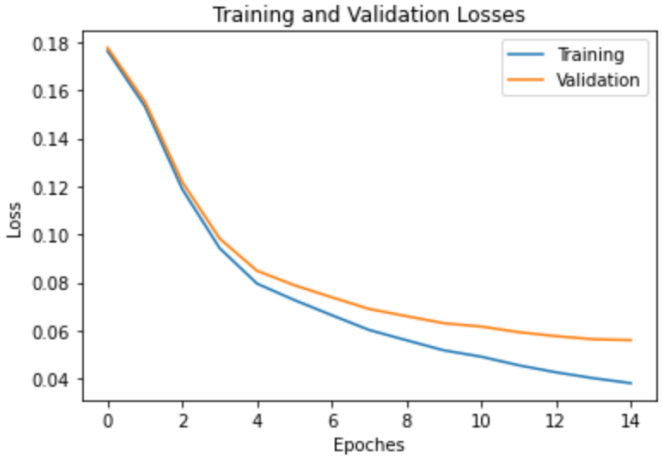

Figure 20: Training/Validation Accuracy for Vgg Fine Tune

Vgg freeze: Freeze the feature extracting part and only tune the fully connected layer

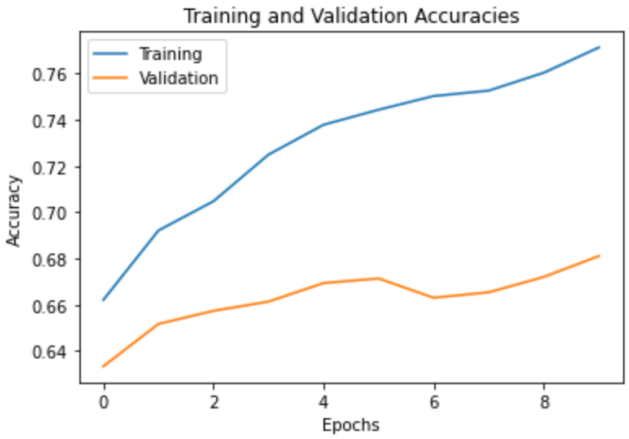

Figure 21: Training/Validation Accuracy for Vgg Freeze

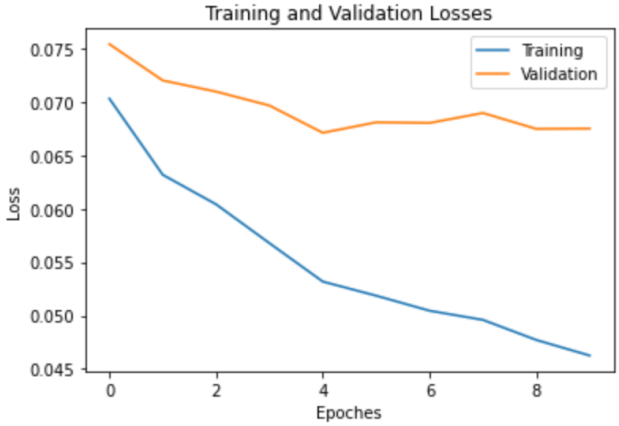

Figure 22: Training/Validation Accuracy for Vgg Freeze

5 Feature map and weights analysis

In this section, we analyze the feature activation maps and weights map for our best performing models (our best Vgg, our best resnet and our best custom model). We show what our model has learned during the training process by analyzing the filters and how our input is transformed when it is passing through the deep neural network.

5.1 First Convolutional Layer Weight Maps

As one can see from the following graphs, the brighter the pixel is, the higher the weight will be. Dark pixel means zeros or negative weight. The model will focus more on the area where the weight values are more when doing the convolution process. Some of them are edge detectors whereas others could be more responsive on the texture. We can see what those filters are doing in the next section, where we pass a real image into the dataset and get a series of feature activation maps. We can then see what the filter is doing by looking at the activation map.

The resnet model:

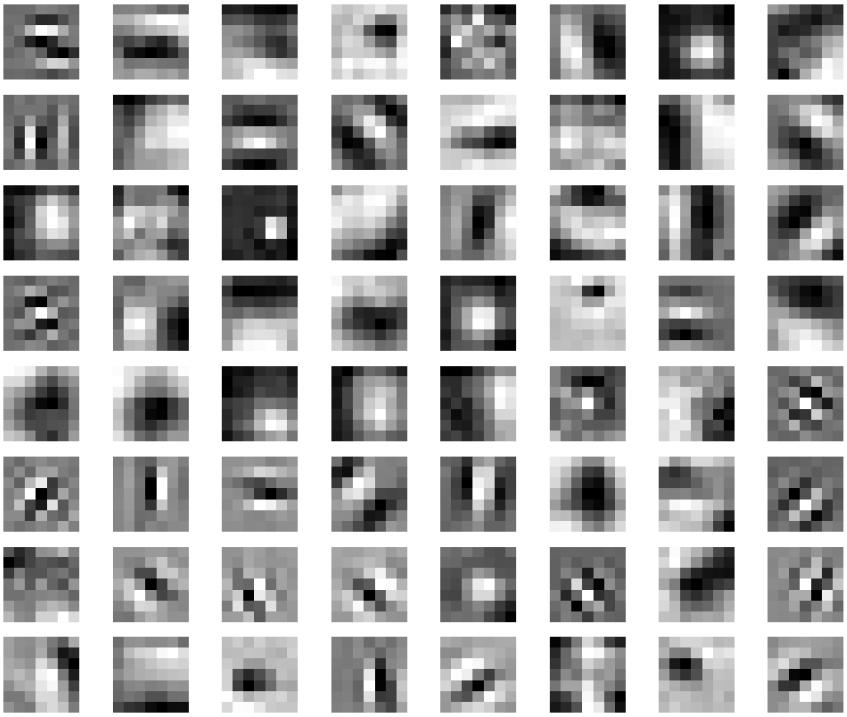

Figure 23: ResNet Model Weight Map

The vgg model:

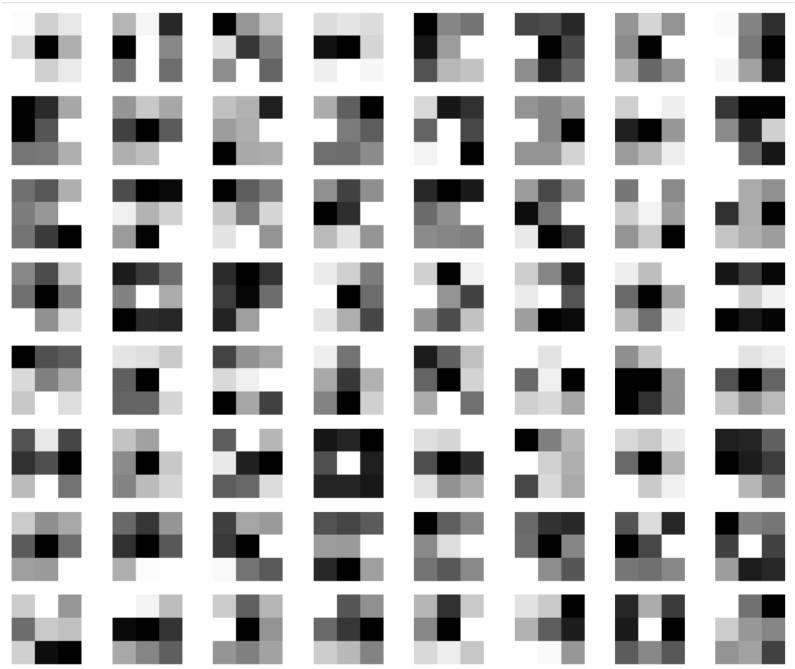

Figure 24: Vgg Model Weight Map

Custom model:

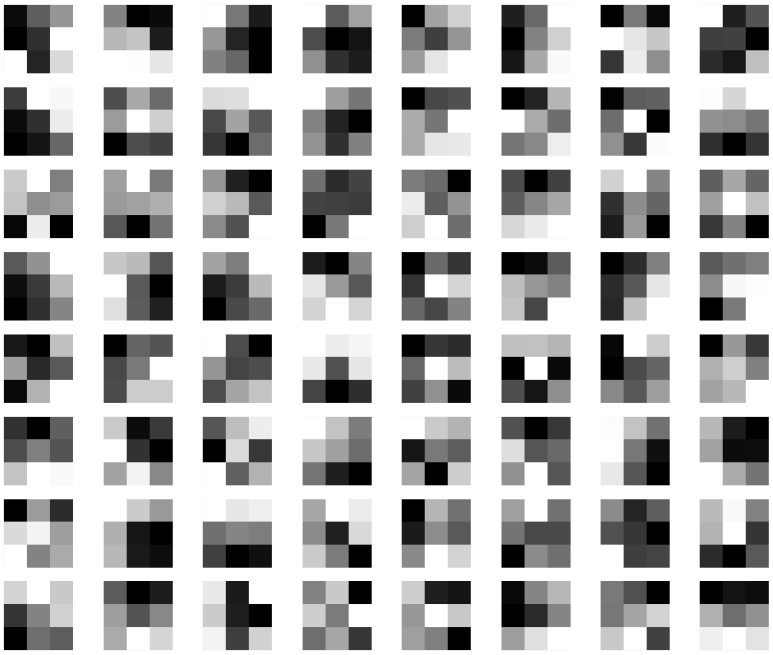

Figure 25: Custom Model Weight Map

5.2 Feature Maps

To get an illustration of how each neural network maps the image and what features the neural network might be looking for , we first take the first image in the test dataset and pass it through the CNN model. And then we get three feature activation maps: One from the initial layer, one from a middle layer, and one from the final convolutional layer. We show the image we use in Figure 26.

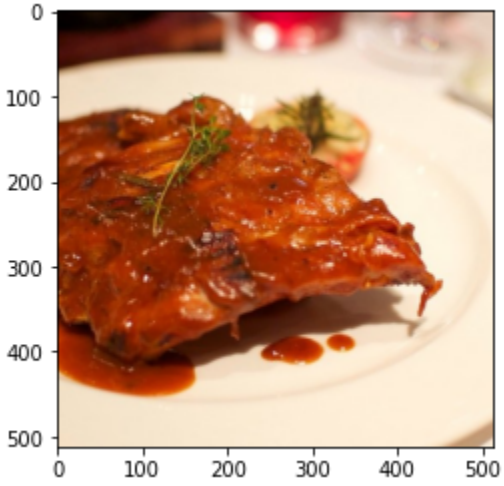

Figure 26: Image to Visualize

For the ResNet model:

First Layer: (we could clearly see what this is)

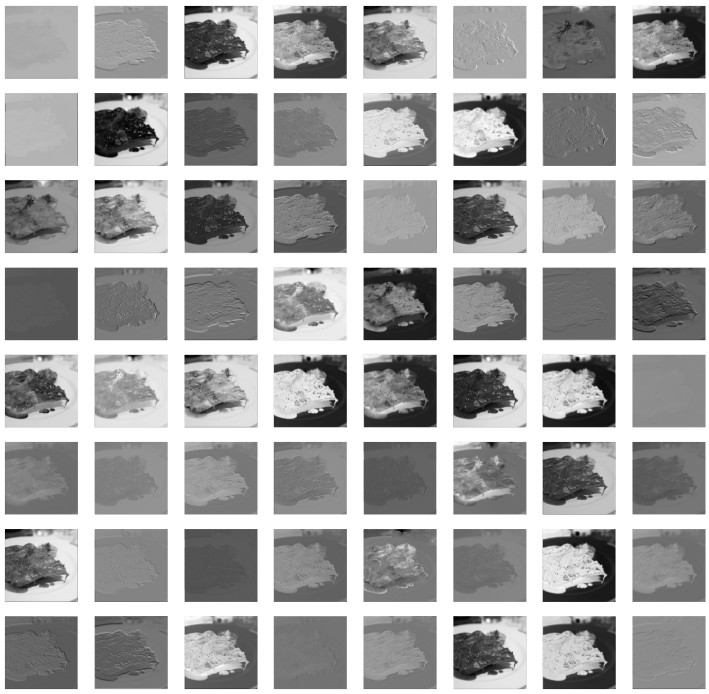

Figure 27: ResNet First Layer Feature Map

Somewhat middle in the network: (I only take out 64 maps for the sake of simplicity)

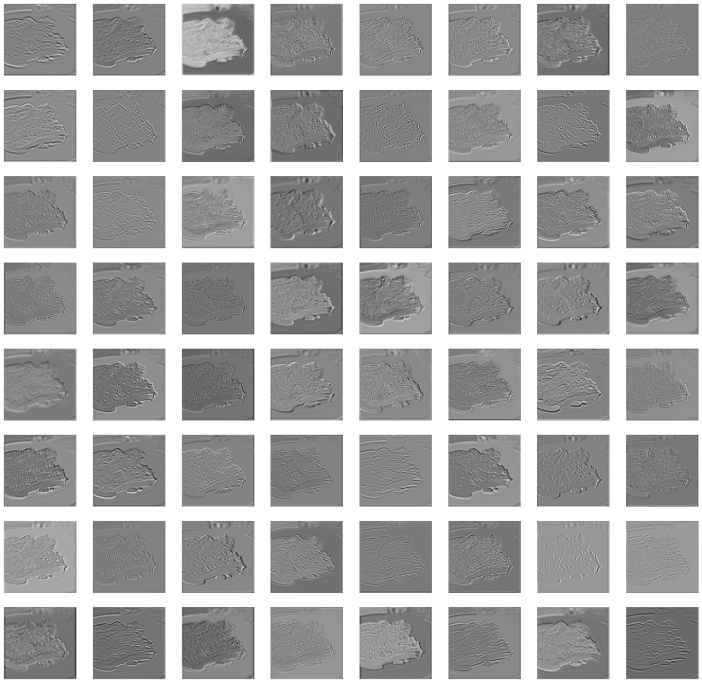

Figure 28: ResNet Middle Layer Feature Map

At the end of the network: (less interpretable and lower resolution)

Again I only keep 64 of them for the sake of simplicity

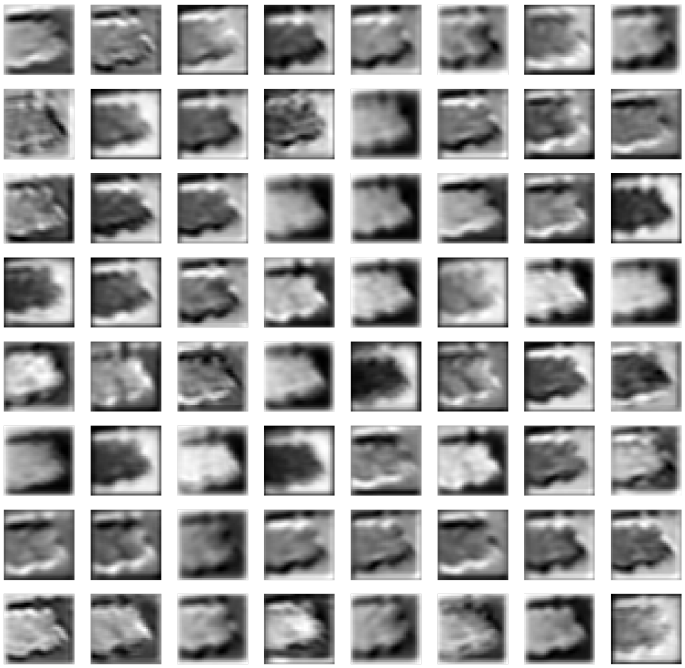

Figure 29: ResNet Last Layer Feature Map

For the Vgg model:

First layer:

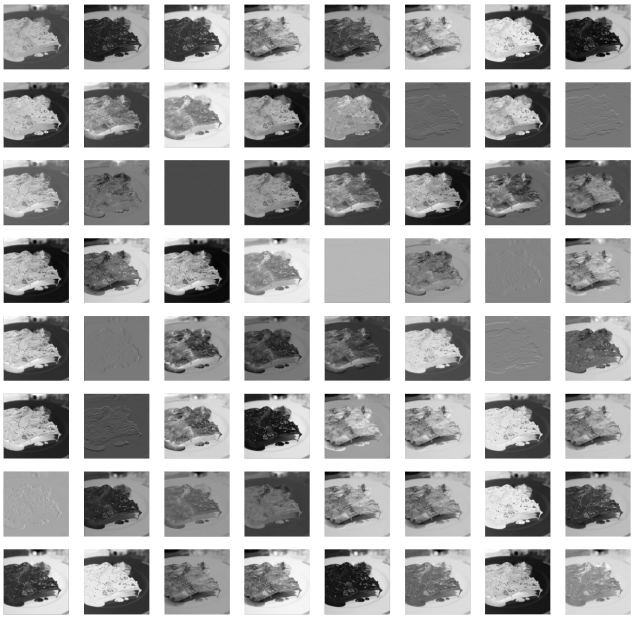

Figure 30: Vgg First Layer Feature Map

Middle layers:

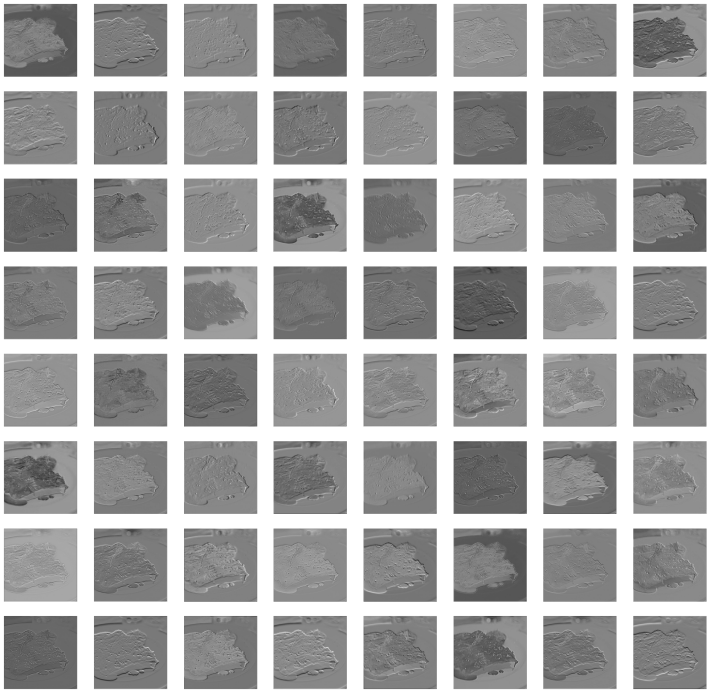

Figure 31: Vgg Middle Layer Feature Map

Final layer:

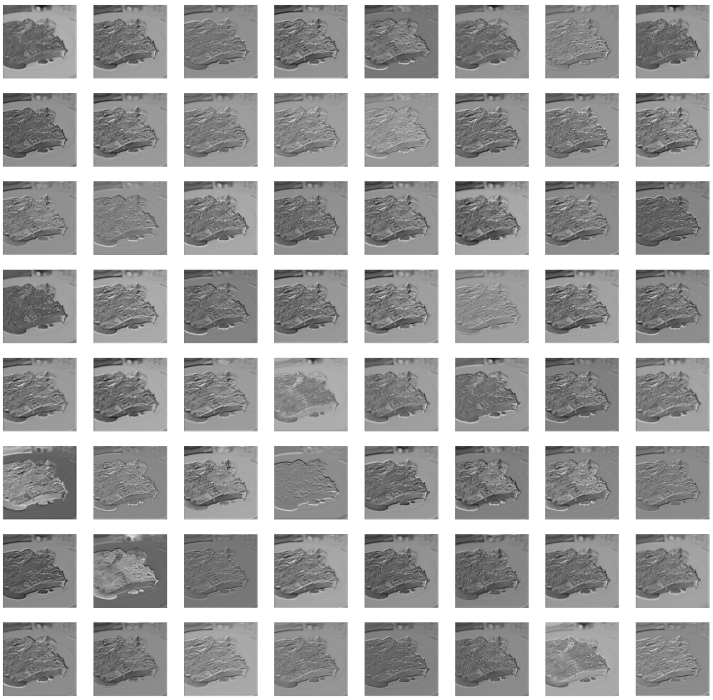

Figure 32: Vgg Last Layer Feature Map

For the Custom model:

First layer:

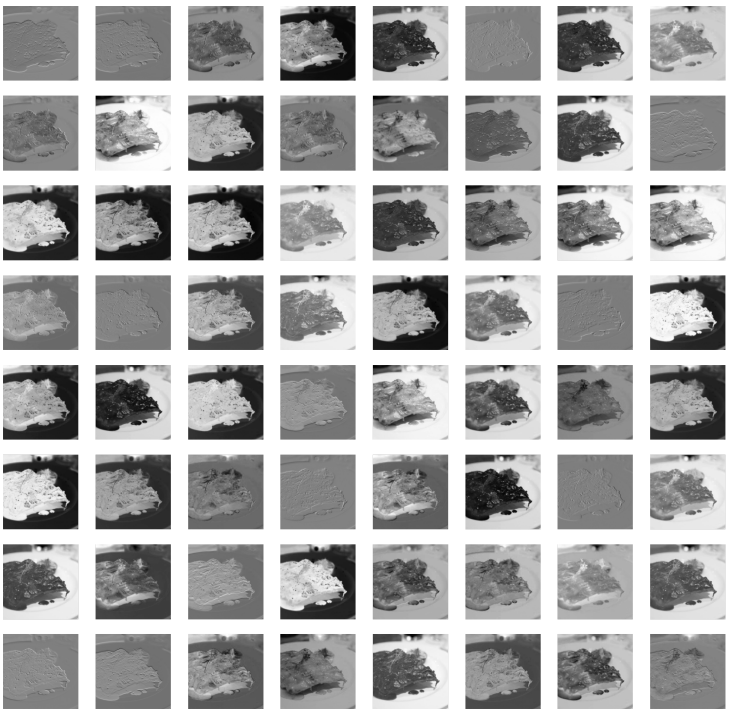

Figure 33: Custom Model First Layer Feature Map

Second Layer:

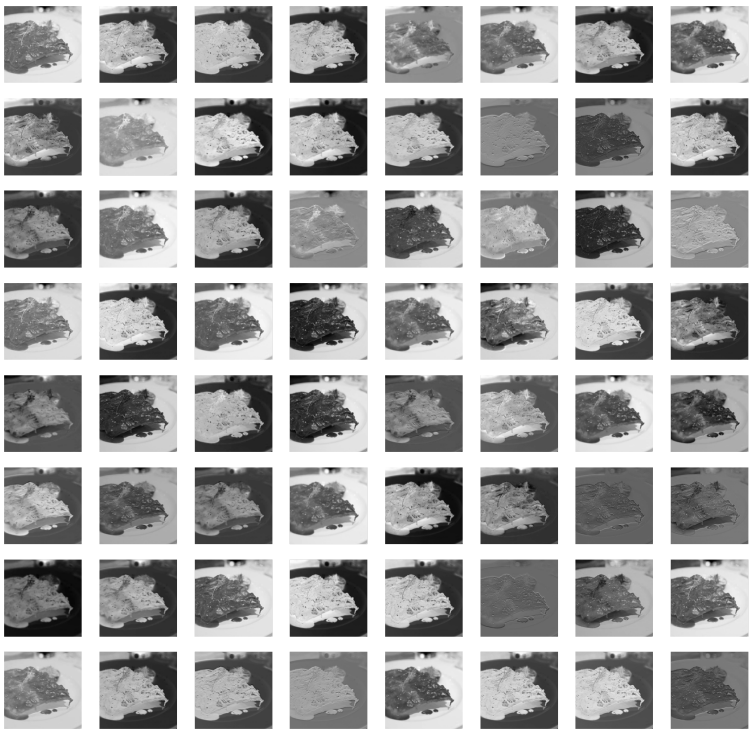

Figure 34: Custom Model Second Layer Feature Map

Third layer:

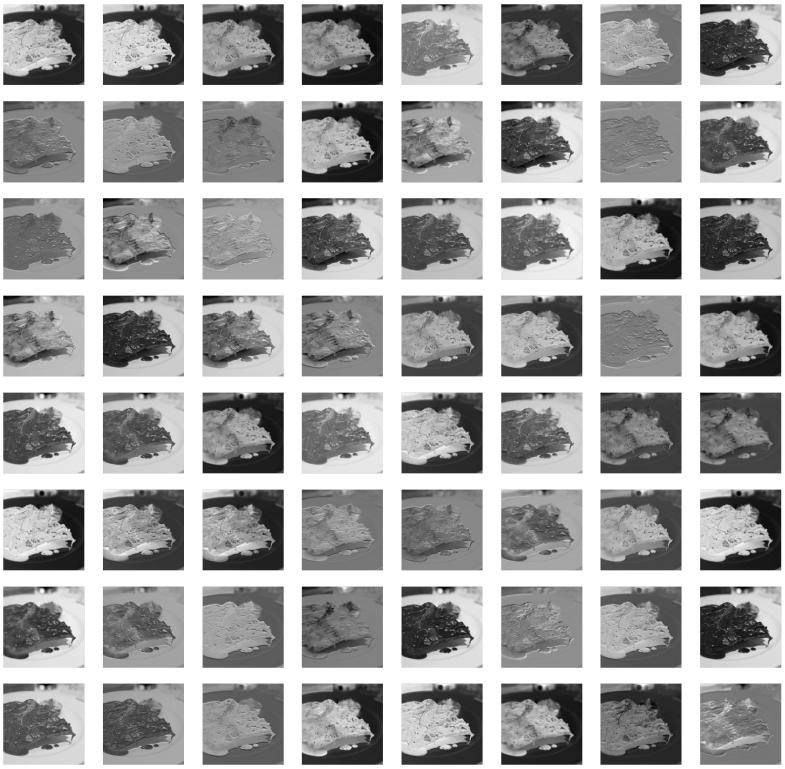

Figure 35: Custom Model Third Layer Feature Map

In conclusion, different filters emphasize different parts of the image. Also, one may notice that the image that we passed in is recognizable at the very beginning. After that, it becomes less recognizable to humans but more meaningful to the computer.

Some feature maps focus on the background of the image. Some others create an outline of the image. A few filters create feature maps where the background is dark but the image of the ribs is bright. This is due to the corresponding weights of the filters. It is very clear from the above image that in the deep layers, the neural network gets to see very detailed feature maps of the input image.

6 Discussions

It seems that the VGG model is best for solving this task. The reason could be that ResNet is too powerful for solving this task that it could easily lead to overfitting. Baseline Model is a simplistic abstraction of the dataset and could provide limited information about the features of the Food-101 data, while the custom model improves on the model representation by adding depth to the architecture, changing dropout ratio, and fine tuning hyperparameters. Transfer learning on Vgg16 and ResNet could utilize the pretrained features and therefore reduce the training resources required and allow the model to “stand on the shoulders of the giants”. Freezed parameters could lead to better model performance for transfer learning, but accuracy might be traded for efficiency if we choose to freeze certain layers.

When designing the final custom model, we find out that with two extra convolutional layers, we could increase the model accuracy by increasing the depth of the model. We test different positions

to add convolutional layers: we first add extra convolutional layers after conv2 by stacking 2 64x64x3x3 layers. Yet the training and validation loss seems not satisfactory. We then add extra layers after conv4 in the baseline models, and we also attach an extra layer of max pooling so that we could amplify some important features the model looks for. We also experiment with where we should add the dropout: between the first few layers, between the middle layers, or between the final layers. After careful implementation, we figured out that the model would work best if we could add dropout with p=0.2 between conv1 and conv2, and add dropout with p=0.5 between fc1 and fc2. The reasoning is that the dropout of the previous layers should mute fewer connections between nodes than the dropout in the latter layers of the neural network. We also try different epoch numbers. We first observe how our custom model would behave by running 100 epochs. Although the training accuracy reaches 96%, the validation accuracy is only 33.26%, which suggests serious overfitting. We then limit the epoch number to be 30 and observe a significant increase in the last epochs validation accuracy and a drop in the last epochs validation loss. Large epoch numbers would easily cause the model to overfit the training portion of the data and therefore lead to poor generalization; small epoch numbers would, on the other hand, underfit the model and therefore could not give a proper representation of the Food101 dataset.

When doing transfer learning, ResNet's best performing model is to fine tune the whole pretrained model. Although the training process is much slower than we did in paradigm 1, it turns out that this is still faster than training from scratch, and usually has higher accuracy than a situation where we only fine tune the fully connected layer. The reason could be, for example, transfer learning tends to perform well when the source domain is close to the target domain. If the two domains are quite different, the transfer learning could still be useful but one should fine tune more layers in the pretrained network. For the best performing Vgg Fine tune model, we unfreeze the parameters in the model and we modify the classifier by changing the number of output channels to 20. We first use Adam optimizer with a learning rate of 0.01t. We notice that the learning rate is too large, so the model would not converge. We then change the learning rate to be 1e-4, which is also too big and the model does not seem to converge yet. Finally, we set the learning rate to be 1e-6 and an epoch number of 25 to curtail overfitting and obtain an acceptable improvement on the Vgg Freeze model performance (75.6% validation accuracy).

We found that the learning rate is a very important hyperparameter to be tuned. Too high a learning rate would cause the model hard to converge, while a low learning rate would consume too much time and computing power to learn a representation of the dataset. Same two models with the same model architecture can vary significantly in their performance if they have different learning rates. If time and resources permit, we can train more epochs and further tune the learning rate to achieve better results. Also, we could augment our training dataset, which could also lead to improvement in the accuracy.

The number of epochs is crucial to the representation of a model. If the model is trained for limited epochs, it would likely underfit the dataset and thus could not well generalize to the unseen portion of the data. However, if the model is trained for too many times, then the model would overfit the training data and could hardly learn to provide an accurate depiction of all the data. It is hard to determine the “right” amount of epochs for a model, different changes in hyperparameter and model architecture could require different epoch numbers.

For the Food-101 dataset specifically, one fun fact is that in the training dataset, one image of frog was mislabeled as “chicken_wings/477991”, so we removed the image from the training dataset to reduce unnecessary model noise. There might be other messy data that could be cleaned.

For future improvements, one could also change the architecture of the pretrained models used in this paper. For example, we could use a deeper version of the ResNet or other powerful models like inception model and GoogLenet. Removing certain layers in the pretrained model might also be a good choice if we are concerned about overfitting. We could also selectively freeze certain

layers. For example, if we have 10 layers, we can freeze the first 3 layers and fine tune the rest. All of these mentioned experiments can be done if we have more time and computational resources.

7 Team contributions

Rui Zhang: Baseline +Resnet

Yifei Wang : Custom + Vgg

Collaboratively done: Feature map and weight maps; Report.

References

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[2] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[3] Wang, X., Kumar, D., Thome, N., Cord, M., & Precioso, F. (2015, June). Recipe recognition with large multimodal food dataset. In 2015 IEEE International Conference on Multimedia & Expo Workshops (ICMEW) (pp. 1-6). IEEE.

[4] Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning. arXiv preprint arXiv:2106.11342.