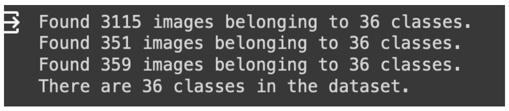
Homework 3

Spencer Au

Introduction

The dataset I am working on is a food dataset that consists of images of various fruits and vegetables. The dataset is available on Kaggle and is called Fruits and Vegetables Image Recognition Dataset. In terms of fruit, there are pictures of bananas, apples, pears, grapes, oranges, kiwis, watermelons, pomegranates, pineapples, and mangos. In terms of vegetables, there are cucumbers, carrots, capsicum, onions, potatoes, lemons, tomatoes, raddishs, beetroot, cabbage, lettuce, spinach, soy beans, cauliflower, bell peppers, chilli peppers, turnips, corn, sweetcorn, sweet potatoes, paprika, jalepeños, ginger, garlic, peas, eggplants. The problem we are trying to solve is to classify the images of fruits and vegetables. This is important as social media platforms such as Instagram and Yelp can leverage this technology to help users identify fruits and vegetables and streamline the process of posting and classifying pictures.

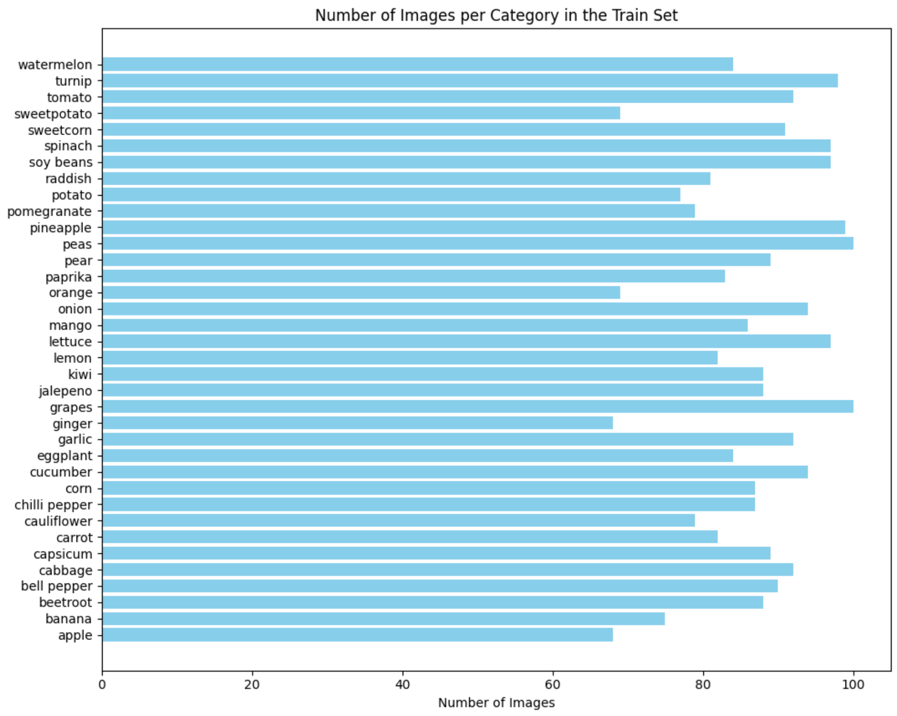
Analysis

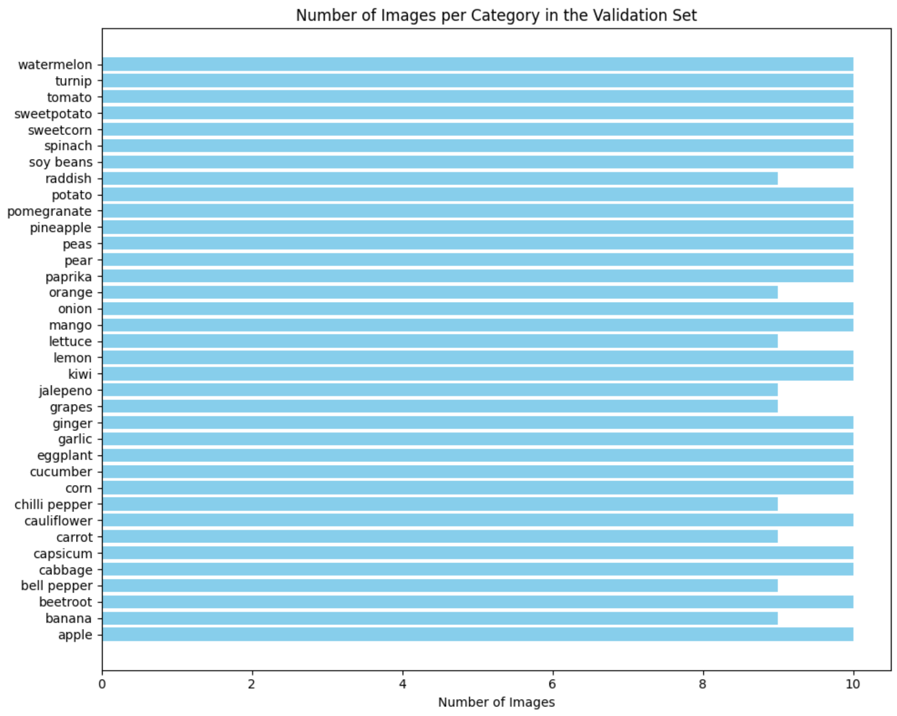


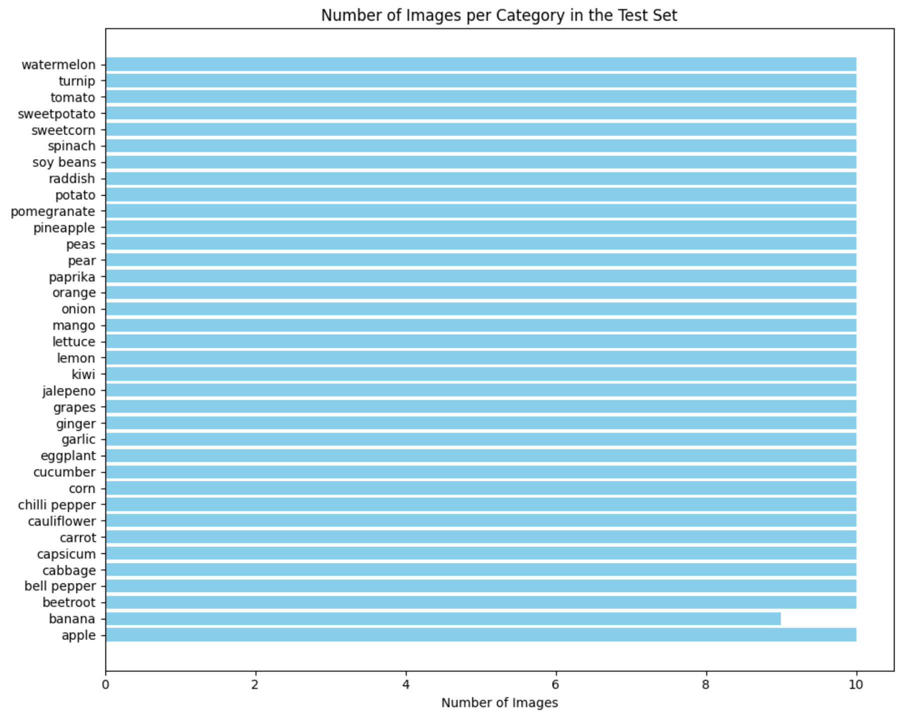
```
Found 3115 images belonging to 36 classes.
Found 351 images belonging to 36 classes.
Found 359 images belonging to 36 classes.
There are 36 classes in the dataset.
```

Figure 1: Class Count

The dataset contains 3115 images in the training set, 351 images in the validation set, and 359 images in the testing set. There are 36 classes in total, with each class corresponding to a different fruit or vegetable.


Number of Images per Category in the Train Set
watermelon
turnip
tomato
sweetpotato
sweetcorn
spinach
soy beans
raddish
potato
pomegranate
pineapple
peas
pear
paprika
orange
onion
mango
lettuce
lemon
kiwi
jalepeno
grapes
ginger
garlic
eggplant
cucumber
corn
chilli pepper
cauliflower
carrot
capsicum
cabbage
bell pepper
beetroot
banana
apple
0
20
40
60
80
100
Number of Images

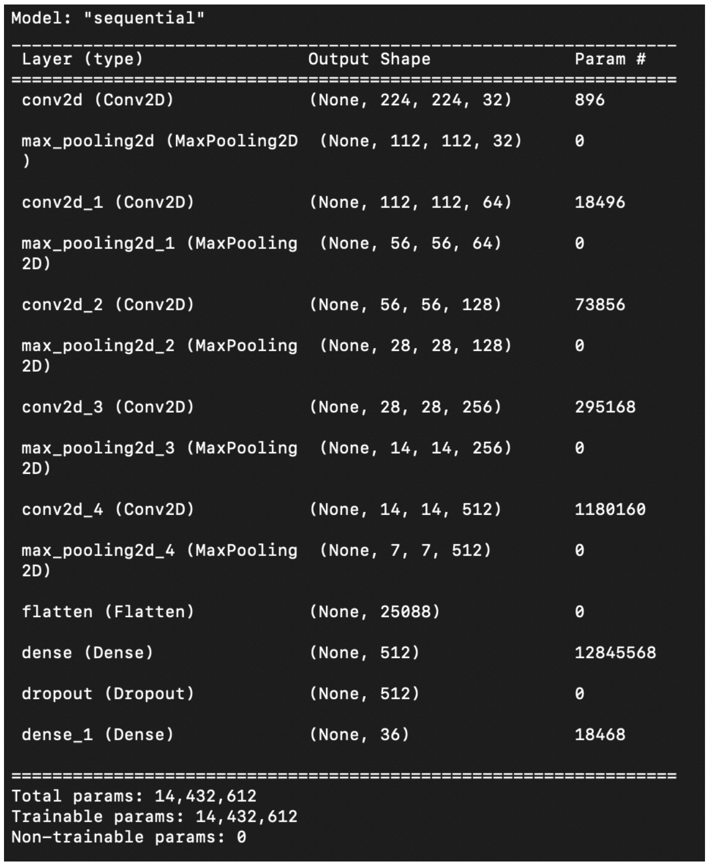

Number of Images per Category in the Validation Set
watermelon
turnip
tomato
sweetpotato
sweetcorn
spinach
soy beans
raddish
potato
pomegranate
pineapple
peas
pear
paprika
orange
onion
mango
lettuce
lemon
kiwi
jalepeno
grapes
ginger
garlic
eggplant
cucumber
corn
chilli pepper
cauliflower
carrot
capsicum
cabbage
bell pepper
beetroot
banana
apple
0
2
4
6
8
10
Number of Images



The training set is imbalanced, with the number of images per class ranging from 70 to 100. The validation sets is also slightly imbalanced, with the number of images per class ranging from 9 or 10. The testing set only has the banana class with 9 images, and the other classes have 10 images.

Methods

Custom Model

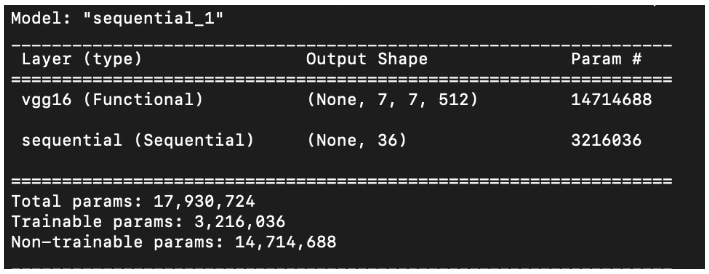


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 32)      896

 max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 112, 112, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 256)      0
 2D)

 conv2d_4 (Conv2D)           (None, 14, 14, 512)       1180160

 max_pooling2d_4 (MaxPooling  (None, 7, 7, 512)        0
 2D)

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 512)               12845568

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 36)                18468

=================================================================
Total params: 14,432,612
Trainable params: 14,432,612
Non-trainable params: 0
```

Figure 2: Custom Model Architecture

For the custom model I built from the ground up, I used a Convolutional Neural Network (CNN) architecture consisting of 5 convolutional blocks, with each block consisting of a convolutional layer with a 3x3 kernel filter, reLu activation, and same padding followed by a 2x2 pooling layer. The convolutional layer is basically the computer using a tiny grid to look closely at small sections of a picture, helping to highlight patterns or details. The pooling layer is the computer simplifying what it sees by picking the most important bits, making the picture easier to analyze without all the extra noise. A 3x3 kernel filter is a 3x3 grid that the computer uses to look at the picture. The reLu activation function is a function that helps the computer decide whether or not to pass information to the next layer. The same padding is a technique that helps the computer keep the size of the picture the same after it goes through the convolutional layer. The convolutional blocks are followed by a flatten layer, a dense fully connected layer, and a dropout layer of 0.4. The flatten layer is a layer that helps the computer turn the 2D image data into 1D data that it can use to make a decision. The dense fully connected layer is a layer that helps the computer make a decision based on all the information it has gathered. The dropout layer is a technique that helps the computer avoid overfitting by randomly ignoring some of the data, and 0.4 means we dropout 40% of the data. The model is compiled with the Adam optimizer, categorical crossentropy loss function, and accuracy and the top k categorical accuracy metric. The Adam optimizer is a technique that helps the computer adjust its learning rate as it learns. The categorical crossentropy loss function is a technique that helps the computer measure how well it is doing and can account for multiple categories such as various foods. The accuracy metric is a technique that helps the computer measure how well it is doing. The top k categorical accuracy metric is a technique that helps the computer measure how well it is doing by looking at the top k most likely categories. The model is trained for 200 epochs along with an early stopping callback that stops the training process if the validation loss does not improve for 10 epochs. The model is trained on the Chapman University DGX Cluster, which is a high performance computing cluster with 8 NVIDIA A100 GPUs.

Transfer Learning Model



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 sequential (Sequential)     (None, 36)                3216036

=================================================================
Total params: 17,930,724
Trainable params: 3,216,036
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 3: Transfer Learning Model Architecture

For the transfer learning model, I used the VGG16 architecture as the base model. The VGG16 architecture is a pre-trained model that has been trained on the ImageNet dataset, which is an extremely large dataset with millions of images. The base model is followed by a Dense layer with 128 nodes, reLu activation, and l2 regularization, which is a technique used to prevent a computer model from focusing too much on a small number of details by penalizing it for having too large values, helping it better generalize to unseen data. The dense layer is followed by a dropout layer of 0.4, and a final dense layer with 36 nodes and softmax activation. Softmax activation is basically compressing the output weights between 0 and 1 to show how much the model thinks the image belongs to that class. This model is compiled with the Adam optimizer, categorical crossentropy loss function, and accuracy and the top k categorical accuracy metric as well. We started off with freezing all the layers, and then trained it for 150 epochs with an early stopping callback with a patience of 10, meaning that it stops the training process if the validation loss does not improve for 10 epochs. After training the model for 150 epochs, we unfroze the last 3 layers and fine tuned it for another 150 epochs with the same early stopping callback. The model is also trained on the Chapman University DGX Cluster.

Results

Custom Model Metrics

Custom	Train	Val	Test
Loss	1.0208	0.3383	0.3159
Accuracy	0.6812	0.9003	0.9081
Top KCA	0.9464	0.9915	0.9889

Transfer Learning Model Metrics

Transfer	Train	Val	Test
Loss	3.5900	3.6002	3.5980
Accuracy	0.0321	0.0256	0.0279
Top KCA	0.1570	0.1396	0.1393

Model History Performance

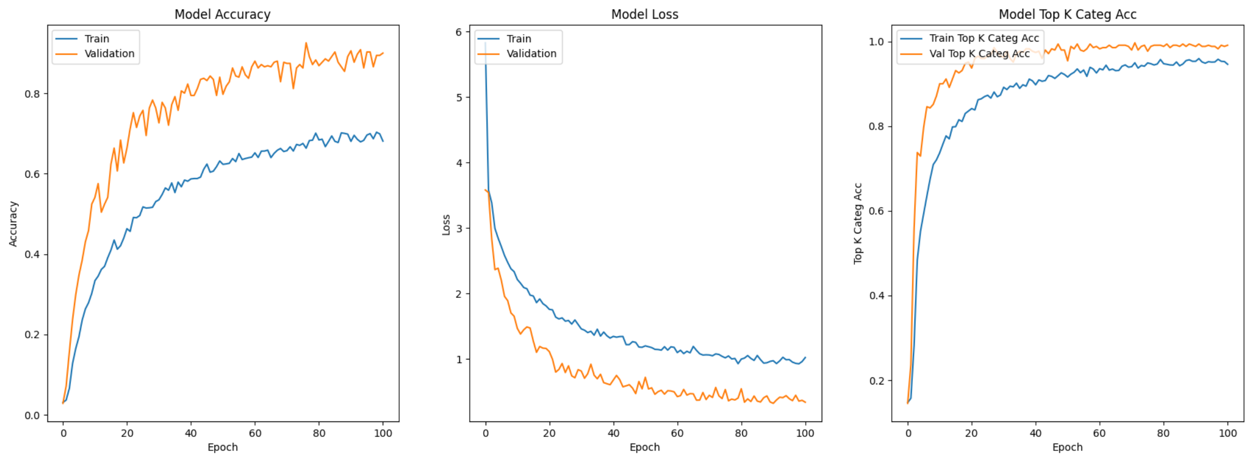


Figure 4: Custom Model History

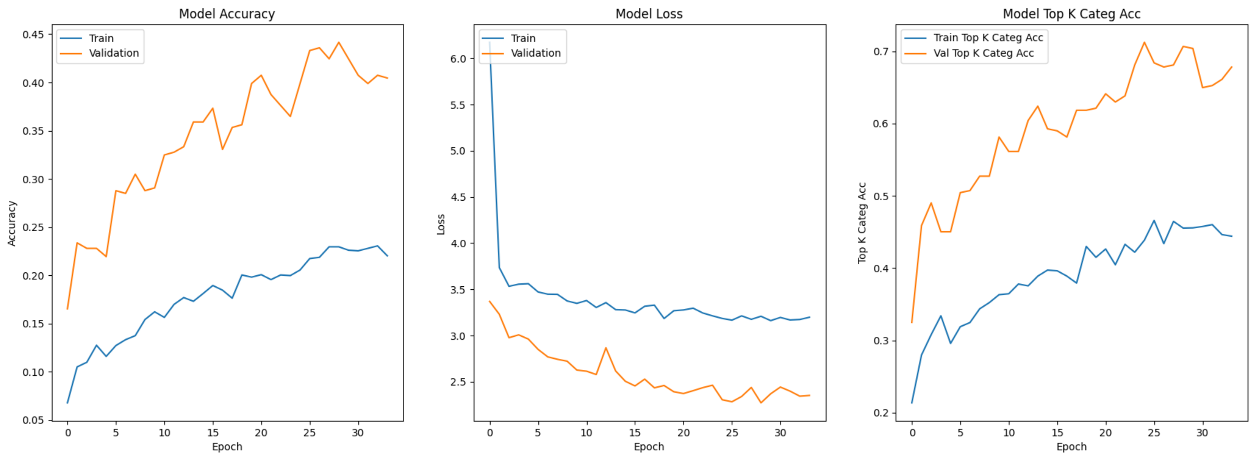


Figure 5: Transfer Learning Model History

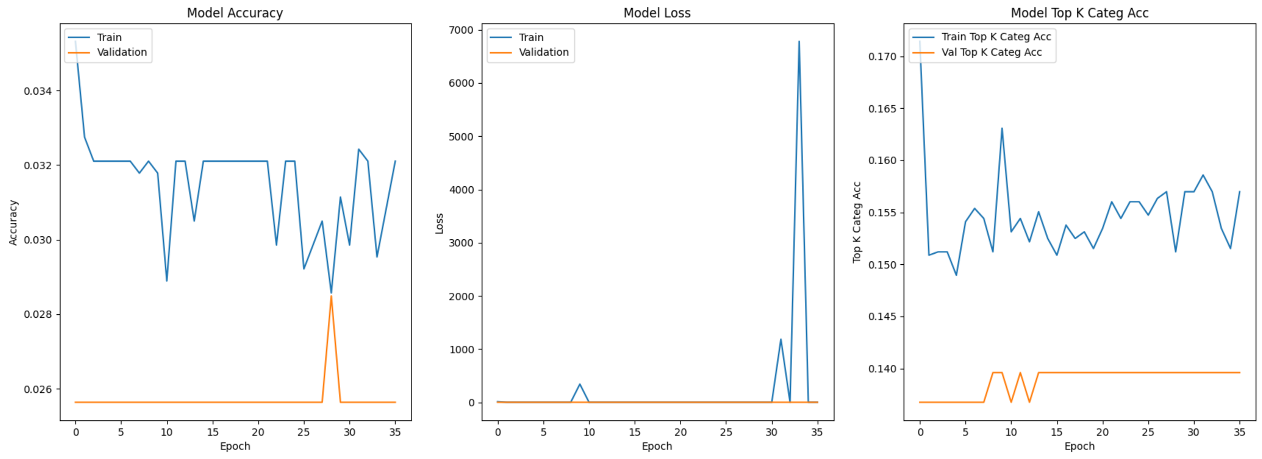


Figure 6: Transfer Learning Model Fine Tuning History

The custom model I made performed well on the validation and testing sets, with an accuracy of 0.9003 and 0.9081, respectively. The top k categorical accuracy was also high, with a value of 0.9915 and 0.9889 on the validation and testing sets, respectively. The custom model was able to generalize well to unseen data, as the validation and testing accuracy were close to each other. Given that I had used a large amount of data augmentation on the data, namely performing various operations ont the images in order to better let the model generalize to data, it's not surprising that the performance on the validation and test sets were remarkably better vs the training set. The history seems to make sense as well, as the loss, accuracy, and Top K Categorical Accuracies are improving over time over both the training and validation sets. The transfer learning model, on the other hand, did not perform well on the validation and testing sets, with an accuracy of 0.0256 and 0.0279, respectively. The top k categorical accuracy was also low, with a value of 0.1396 and 0.1393 on the validation and testing sets, respectively. The transfer learning model struggled, showing high loss and low accuracy across all datasets, hinting at overfitting and poor generalization. In contrast, the custom model demonstrated excellent performance, with high accuracy and low loss, suggesting effective learning and good generalization to unseen data. The transfer model's unfreezing of layers likely led to its overfitting, requiring adjustments in the training approach as well as significantly augmneting the size of the training and validation sets to improve performance and generalization.

Reflection

I learned that building custom models from scratch can be very effective, as well as the fact that transfer learning models may not necessarily be the best option, especially if you have a small dataset to begin with. I struggled with the transfer learning model, as the performance was significantly decreased as I entered the fine tuning stage and started unfreezing layers. In the future, I would like to try differnt fine tuning techniques as well as a larger dataset to see if I can improve the performance of the transfer learning model.