

Team Members



Danielle Angelini, MS Data Science



Wylie Borden, MS Data Science



Knut Peterson, MS Robotics Engineering



Shannon Song, PhD Data Science

Abstract

Reducing food waste has become a major focus by producers and consumers to decrease their carbon footprint and to fully make use of their resources. Forty percent of this waste happens at grocery stores, restaurants, and homes, so being able to track food freshness in each area would be key in tackling this problem. One solution would be to use a combination of photo capture and image-based artificial intelligence to monitor freshness. In this study, image-based techniques such as convolutional neural networks, AlexNets, and HRFormers will be discussed and compared in their ability to detect fruit freshness on apples, bananas, and oranges. AlexNet trained with PyTorch appears to outperform these models in terms of computational cost, having the lowest training time, while still maintaining comparable accuracies.

Keywords: Classification, Convolutional Neural Network, AlexNet, HRFormer, optimizer, Adam, stochastic gradient descent, TensorFlow, PyTorch, Python

1 Introduction

1.1 Motivation

Each year, a staggering one-third of all food intended for human consumption goes to waste. Within developed countries, 40% happens at grocery stores, restaurants, and consumer homes. In developing countries, 40% of food goes to waste before it is even processed or reaches the end of the supply chain [1]. This wastage represents a significant loss of human labor and resources in an industry generating trillions of dollars in revenue annually. Thankfully, advances in artificial intelligence and machine learning are helping to address this issue. By improving food consumption forecasting models and optimizing transportation routes, among other tactics, we can significantly reduce food waste. One key solution involves developing sensors to monitor food freshness, as many emerging techniques have shown promising results[2]. These types of sensors will be useful with the image-based artificial intelligence methods discussed in this paper to monitor fruit freshness.

1.2 State of the Art Applications

Current state-of-the-art (SOTA) applications that have been researched for image recognition include convolutional neural networks (CNNs), AlexNet, and High Resolution Transformers (HRFormers). Research has also shown CNNs and AlexNet being applied specifically to detecting fruit freshness.

1.2.1 Convolutional Neural Networks

A convolutional neural network (CNN) is a type of artificial neural network (ANN) that has become the standard for modeling image-based data[3]. With Python packages like PyTorch and TensorFlow, CNNs can be easily implemented to fit a variety of datasets. They offer other advantages such as minimizing computation compared to other neural networks and being a reliable method for image recognition and classification [?]. However, CNNs struggle with image segmentation and require a lot of training data [?]. Looking at previous studies, CNN's runtime is unknown, which is important for detecting fruit freshness in real-time; this is a gap we hope to address with our study.

1.2.2 AlexNet

AlexNet is a deep CNN that is typically eight layers deep and was designed by Alex Krizhevsky. Being a deeper CNN offers an advantage over other architectures by being better at extracting features from images [4]. It was also the first major CNN able to use graphics processing units for training, leading to faster run times [4]; these advantages are important to have for this study. However, compared to more advanced architectures like ResNet, it is not deep enough to support higher accuracy scores [4]. In previous studies, AlexNet CNNs tend to use Adam optimizers [5]. Although Adam optimizers tend to be computationally and memory efficient, it may not always converge to the most optimal solution. More recent optimizers also have proven to produce potentially better results [6].

1.2.3 High Resolution Transformer (HRFormer)

The HRFormer builds off of the architecture of Vision Transformers (ViTs). HRFormer aims to bring high resolution to ViTs. The HRFormer is based on the HRFormer block, which is diagrammed in figure 1. This architecture makes it possible to maintain high resolution representations throughout the model, which has been found to be important for dense prediction tasks [7]. However, HRformers tend to require a lot of training data and are more complicated to implement than the CNNs discussed previously.

1.2.4 Addressing Advantages and Disadvantages

Based on these current state-of-the art applications, the aim of this study is to measure the run times of an AlexNet and a basic CNN as there is no current precedent. The proposed AlexNet will be a good measure of run time success for detecting fruit freshness given its ability to use graphics processing units. The proposed AlexNet will also be important when training on the chosen dataset. The dataset contains around 4GB of data, which can be handled well by an AlexNet unlike an HRFormer or basic CNN that requires a lot more data to train successfully. Altering the current version of the AlexNet through changing the types of optimizers typically used, such as Adam [5], will also be a beneficial exploration as Adam does not always converge to an optimal solution [6]. This study also plans to implement an AlexNet with an additional layer to help increase depth and improve the image features generated by training of a basic AlexNet.

1.3 Proposed Methods

This study proposes to alter the AlexNet architecture as seen in reviewed papers through further hyperparameter tuning. Alterations include adjustments to optimizers, number of epochs, and learning rate [8]. Implementation of an AlexNet with an additional convolutional layer will also be discussed. Previous studies lacked information about run times, so our goal is to provide a comparative study of an AlexNet to a CNN created from information based on previous studies. This study will also discuss the challenges faced during HRFormer implementation.

1.3.1 Dataset

The dataset selected for this project is the "Fruits Fresh and Rotten Dataset for Classification" from Kaggle [9]. This dataset includes a train and test directory with images of fresh and rotten apples, oranges, and bananas. Below are example images from the dataset:







	Fresh	Rotten
Apple		
Orange		
Banana		

Figure 1: Example images of fresh and rotten fruit from the Kaggle dataset [9].

1.3.2 Evaluation Metrics

Since the proposed project is a binary classification problem, basic evaluation metrics may be used such as accuracy and log-loss score. AlexNet models from both PyTorch and TensorFlow will be optimized and compared to see which package offers better results. In terms of determining model success, the implemented AlexNet CNN will be compared to both previous CNNs (included in the bibliography) and the CNN built during this study. The HRFormer implementation process will be discussed in comparison to both the AlexNet and CNN process.

The following section will be highlighting the results of previous studies on CNNs, AlexNet, and HRFormers in more detail and how it can benefit our study. In the third section, we discuss the methodology used in implementing an AlexNet based CNN in both PyTorch and TensorFlow, a personalized CNN, an HRFormer, and an AlexNet Plus One to compare model run times and classification accuracy for freshness for each type of fruits. These results will be discussed further in the fourth section using comparison tables between models. The final section will highlight the work accomplished and what future work can be done to improve results.

2 Related Work

Research has shown CNNs and AlexNet being applied specifically to detecting fruit freshness. Although there was not much support for using HRFormers to detect fruit freshness, it offers great potential.

2.1 Convolutional Neural Networks

The structure of a typical CNN includes convolutional layers, pooling layers and fully-connected layers. The convolutional layer is used to transform images into a format where features may be extracted. Pooling layers often times will reduce the size of the data input unless padding is used. Pooling layers can help speed up the model by reducing the amount of data that is fed through the CNN. The fully connected layer is typically the end of the CNN, where the input from the previous layers are used to produce some sort of output - such as a classification [3].

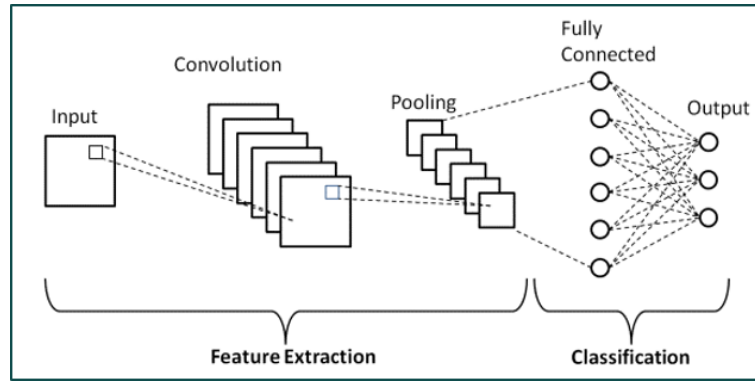


Figure 2: A diagram of a typical CNN block which displays the interaction between the input, convolutional layer, pooling layer, fully connected layer, and output.

For fruit freshness classification, CNNs were typically constructed using at least two convolutional and two pooling layers, with an Adam optimizer, and a learning rate less than 0.001. Image augmentation is highly recommended to avoid model overfitting while supporting CNN's need for a lot of training data. CNN models from previous studies generally had a high accuracy score of 97% on the Kaggle fruit freshness dataset, which will be used for this study. Therefore, this model seems to be a promising solution to classify fruits based on freshness.

2.2 AlexNet

An AlexNet architecture is typically made up of eight layers: three pooling layers and five convolutional layers [5]. Each convolutional layer consists of convolutional filters and an activation function. The pooling layers are used to perform max-pooling, which decreases the size of the feature map to avoid overfitting, reduce dimensionality, and to help identify different parts of an image. A diagram of a typical AlexNet CNN is shown below:



Figure 3: The architecture of the AlexNet implemented in the study discussed below. [5].

One study uses AlexNet to predict fruit maturity and quality, specifically for bananas [5]. The study claims that AlexNet was able to achieve a 98.18% training accuracy and an 81.75% validation accuracy. This likely suggests that the AlexNet overfitted to the dataset and did not generalize as well as it could. However, the authors also experimented with an augmented dataset and reported a training accuracy of 99.80% as well as a validation accuracy of 99.44% [5]. This suggests that a larger, more diverse dataset is needed in order for AlexNet to perform optimally. It should be noted that this study compared the AlexNet to an implemented CNN composed of 3 convolutional layers with 2 max-pooling layers. The study reported a slightly better performance from the implemented CNN than from the AlexNet on the Fruit 360 dataset. However, information was not included about which model had a better runtime, which would be interesting to know given AlexNet's general ability to train quickly.

2.3 HRFormer

From the literature that has been reviewed, it does not appear that an HRFormer has yet been applied to fruit classification problems. However, other related architectures have been seen - ResNet50. One study claims that involving high resolution is a must for semantic segmentation problems [7]. HRFormer offers this as a solution to ViTs - which classically is unable to maintain high resolution due to the nature of its architecture [10]. This study aims to implement a basic HRFormer architecture and compare it to the implemented CNN.

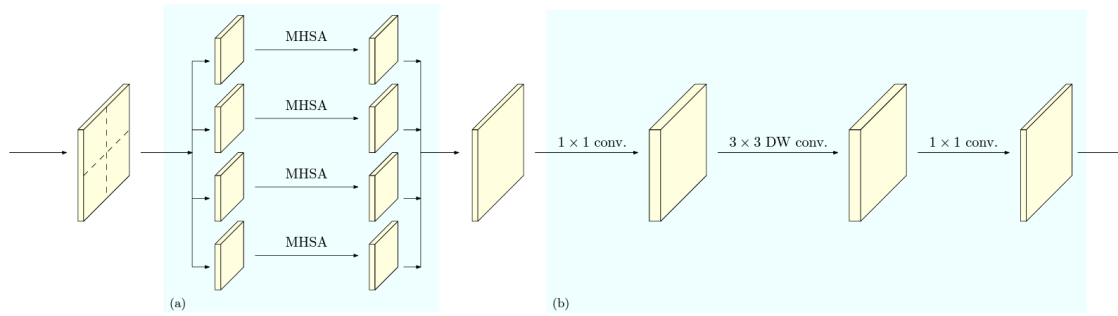


Figure 4: A diagram of the HRFormer block which contains (a) a local-window self attention and (b) a feed-forward network with depth-wise convolution [10].

3 Proposed Methodology

This study first implements CNNs to classify fruit as being either fresh or rotten. The CNN to be implemented first includes the deep CNN architecture, AlexNet. After implementing AlexNet within Python, it will be evaluated by the classification accuracy, log-loss value, and training time. Next, the study will hyperparameter tune the AlexNet architecture to increase classification accuracy while minimizing runtime. An alternative CNN and its results will be discussed and compared to the optimized AlexNet.

This study has implemented the AlexNet in two different Python libraries: PyTorch and TensorFlow. Based on results published in one study, the TensorFlow library performed quicker than the PyTorch library for a given amount of GPU [11]. This study seeks to compare the two libraries with the dataset selected for this study. The following subsections discuss in more detail about the implementation of the AlexNet CNN in TensorFlow and for PyTorch.

3.1 AlexNet Implementation - TensorFlow

To implement the AlexNet CNN, this study performed many sequential steps. Ultimately, the code needed to load the images from the dataset, identify the portion of the dataset used for training, validating, and testing. Then, the code needed to transform the images (initially in .PNG format) into a data format consistent with the TensorFlow library. Then any type of data augmentation may be performed if desired before feeding into the CNN. The CNN layers needed to be defined in the code. This study used the AlexNet CNN architecture as a starting point, which was a CNN model used in a related paper to this study [5] The following block diagram breaks-down the overall strategy for implementing the AlexNet CNN model.

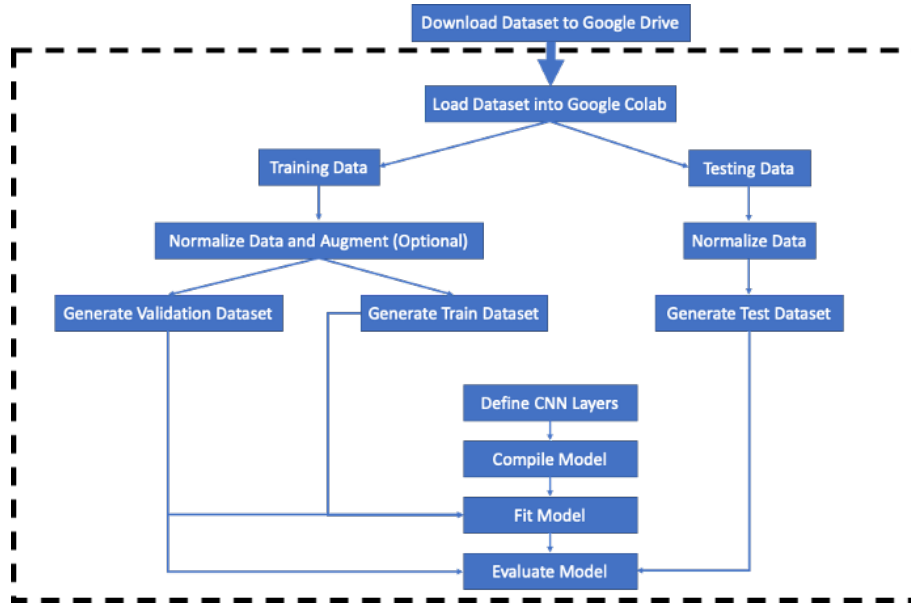


Figure 5: Block Diagram of CNN Implementation

Each element of the block diagram will now be discussed in more detail below.

3.1.1 Download Dataset to Google Drive

The dataset of interest in this study came from Kaggle, and it consists of about 4GB of fruit images [9]. The dataset has images of both fresh and rotten apples, bananas, and oranges. To download the dataset, this study downloaded the images directly to Google Drive, by using Google Colab and utilizing the Kaggle API's. Once the dataset was downloaded, this study organized the folders and names into the following format (shown below). This format allowed for the TensorFlow library to automatically interpret classification of images for a specific folder.

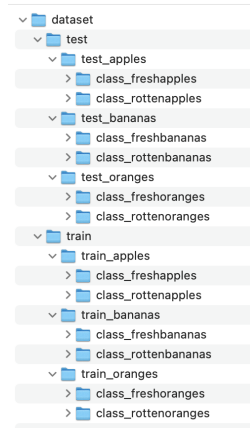


Figure 6: Dataset Layout in Directory

3.1.2 Load Dataset into Google Colab

Once the fruit image dataset downloaded into the Google Drive, the Google Drive was mounted into the Google Colab Python file. This allowed accessing the contents of the Google Drive from the Google Colab script in a way similar to accessing files from a local directory.

3.1.3 Split Dataset

The dataset used in this study had pre-defined training and testing sets. This study loaded them into separate objects within the script.

3.1.4 Normalize Data

This step normalized the training and testing data by using the method “ImageDataGenerator” from the library “tensorflow.keras.preprocessing.image.” Both the training and testing sets would be normalized by converting the raw pixel values into a value between 0 and 1. Initially, the pixel values existed as a value between 0 and 255. For the training set, the ImageDataGenerator method was also used to perform data augmentation. Data augmentation is a technique in machine learning to slightly alter the original training images to increase the generalizability of the model. The slight alterations performed in the TensorFlow implementation include rotation, horizontal and vertical movement of the image, vertically/horizontally flip the image, zoom in on the image, and shearing the image. All of these image transformations happen at random by the TensorFlow library, and the code may be adjusted to control the magnitude of these random transformations. For the training set, the ImageDataGenerator method was also used to fill in any empty pixels in the images to the value of their nearest neighbors. This initial TensorFlow implementation used the data augmentation.

3.1.5 Generate Validation, Train, and Testing Datasets

This step makes the train/test datasets compatible with the TensorFlow Neural Network by using the “flow_from_directory” method. This method must be applied to an object created from the ImageDataGenerator method (used above). In this case, there are already training and testing objects created from the ImageDataGenerator in the previous step. First, the flow_from_directory method was used on the training dataset. It began by separating 20 percent of the training dataset for validation purposes. Next, the flow_from_directory method reshaped every image to a pre-defined pixel dimension. As an initial value, this study used 256 x 256 pixels for every image fed to the CNN. In addition, the method is told this is for binary classification, that the input images are all “RGB” format, and that the images should be shuffled.

3.1.6 Define CNN Layers

This is the step where the CNN architecture is defined. As discussed previously, this study initially implements the AlexNet CNN architecture. The table below summarizes the structure of the AlexNet CNN, which consists of 5 convolutional layers with a combination of max pooling layers. The implementation built the CNN based on the parameters in the below table: [12]. After implementing the AlexNet architecture, it will be used as a baseline to compare against future CNN iterations on the same dataset. Future CNN iterations will consist of variations to the AlexNet CNN, such as by adding/removing layers.

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-

Figure 7: Table of AlexNet Parameters

3.1.7 Compile Model

When compiling the CNN with the TensorFlow library, the code must define how the model will learn and be optimized. Specifically for this initial implementation, the optimizer, loss, and metrics must be specified. This study used the Adam optimizer with a learning rate of 0.00001 to start out. It also used the ‘binary_crossentropy’ loss function, which is the function to be minimized by the optimizer. Lastly, the study used the accuracy score as the main metric and will be monitored during training. Based on model performance using the Adam

optimizer and defined learning rate, future implementations will see if changing these parameters would benefit the CNN in terms of classification accuracy.

3.1.8 Fit and Tune Model

The model will be trained and hyperparameter tuned on the training data now that the TensorFlow neural network and CNN layers have been defined. To speed up the tuning process, only 50% of the training data was used for training and validation. From this subset of the training data, 80% was used for training and 10% was used for validation. The hyperparameter tuning process consisted of looping through various parameters, including the optimizer (Adam or Stochastic Gradient Descent), learning rate (0.00001, 0.0001, 0.001, 0.01), dropout (0.3, 0.5, 0.7), and batch sizes (32 or 64). In total, 29 combinations of parameters were tested. The output for each trial included the training time and validation accuracy, and the final runtime was recorded after the 26 trials. The parameters giving optimal accuracy and runtime results will be selected for the TensorFlow AlexNet. The results of the tuned model will be compared to a base model using the Adam optimizer with a 0.00001 learning rate, dropout of 0.5, and batch size of 32.

3.2 AlexNet Implementation - PyTorch

The dataset download process into Google Colab for PyTorch mimics what was discussed in the TensorFlow section and underwent data augmentation that resulted in 8721 training set images and 2180 validation set images. The train-test split, validation split, and the CNN layers used for the AlexNet implementation in PyTorch were also the same as what was used in TensorFlow.

Like with TensorFlow, the data was transformed and normalized before being an input to the model. Each image was transformed into a PyTorch tensor where the pixel values were scaled to 0.5. This was done using the `torchvision.transforms.ToTensor()` and `torchvision.transforms.Normalize()` functions.

3.2.1 Compile Model

The AlexNet CNN was initially compiled using a learning rate of 0.001 and an stochastic gradient descent (SGD) optimizer to minimize the cross-entropy loss function. A batch size of 32 was also used for the initial model run, but will be altered during hyperparameter tuning.

3.2.2 Fit and Tune Model

To optimize results, a hyperparameter optimization framework called Optuna was implemented. This framework would suggest the parameters using tree-structured Parzen estimation to find which parameters would give the most optimum results after a given number of trials. For this study, fifty trials were used to test the following sets of parameters:

Parameter	Possible Values
Optimizer	SGD, Adam, RMSprop, Adadelta, Adagrad, Adamax
Learning Rate	[0.00001, 0.1]
Number of Epochs	[20,100]

During these trials, the model ran on 50% of the data for time efficiency and recorded the selected parameters and the loss score for each trial. This process was repeated for to compare the results of the following batch sizes: 32, 64, 128, 256. Results of these trials will be discussed in the next section and will be used as the parameters for the final PyTorch AlexNet implementation.

3.3 CNN Implementation

The basic CNN was implemented in PyTorch and follows similar data preprocessing as the AlexNet did. The only difference between the basic CNN and the AlexNet in PyTorch is the model architecture. The architecture consisted of a convolution layer, a max pooling layer, a convolution layer and then three fully connected layers. The CNN architecture as it is coded is shown below:


```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

3.4 HRFormer Implementation

HRFormer was unable to be implemented during this project period. The research group that created the HRFormer have the code accessible through Github [13] and use Linux to run the installation commands. We were unable to install the correct version of Apex due to constraints of operating system access. HRFormer would be an interesting model to implement in the future to compare high resolution representations to the AlexNet and CNN.

3.5 AlexNet Plus One

This method was based in PyTorch and follows the same procedures as that implementation. The only difference is the addition of one convolution layer before the fully connected layer. Below is a snippet of the AlexNet Plus One architecture with the additional layer highlighted:

```
#ALEXNET PLUS ONE
class AlexNet(nn.Module):
    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=1, stride=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
```

4 Experimental Results and Discussions

Model performance and success will be measured by training run time, loss score, and training and validation accuracy. Based on previous studies, a validation accuracy score of at least 97% would be considered successful. Note that there is no precedent for a successful run time, so this section will rely on comparing run times between the three model implementations to measure success.

4.1 AlexNet Results - TensorFlow

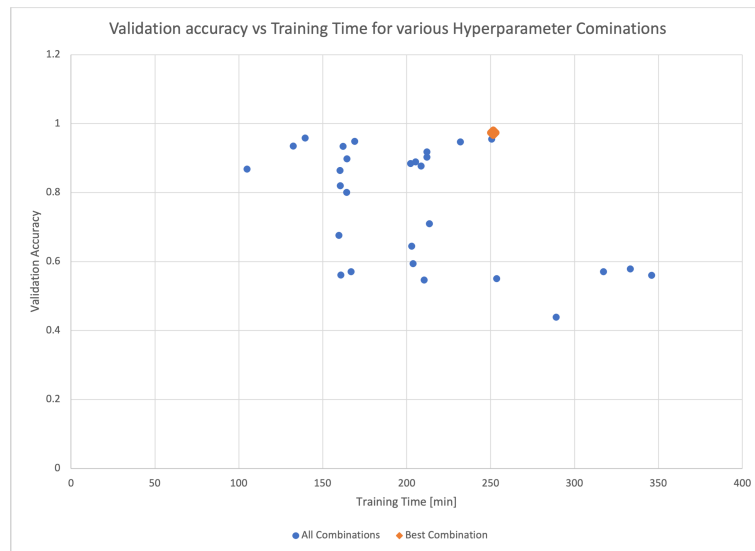
The TensorFlow model tuning successfully observed how various hyperparameters influence the model's ability to detect fresh fruit. Specifically for the optimizer, the Adam optimizer outperformed the SGD optimizer in classification accuracy, and run times seemed relatively close between the two. The results of the learning rates tuning suggest the smaller learning rate values (0.00001) resulted in significantly better classification accuracy without much impact to the training time. The dropout value seemed to not play a significant impact on the accuracy whilst keeping the other parameters consistent. However, the highest overall validation accuracy

occurred with a dropout value at the default 0.5. The batch rate tuning suggests a value of 32 gives the highest validation accuracy, but also a longer run time. The tuning suggests that in general, the training time decreased with batch size, as did the validation accuracy. In conclusion, the best parameters identified for the TensorFlow model was actually the baseline parameters used for the study (Adam optimizer, 0.00001 learning rate, 32 batch size, and 0.5 dropouts).

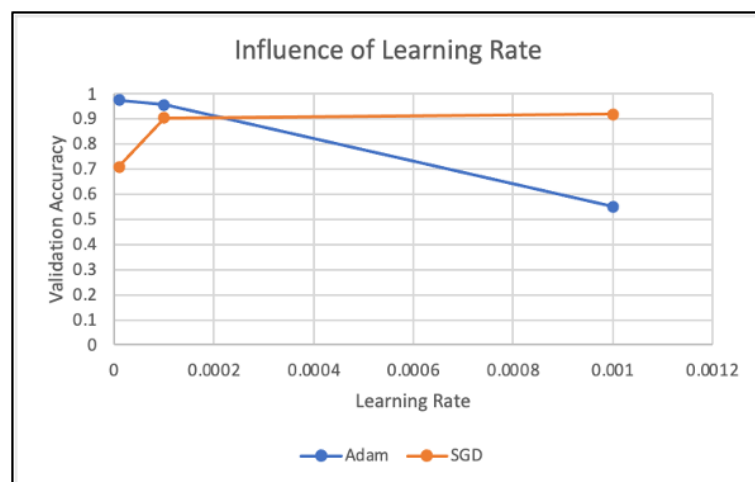
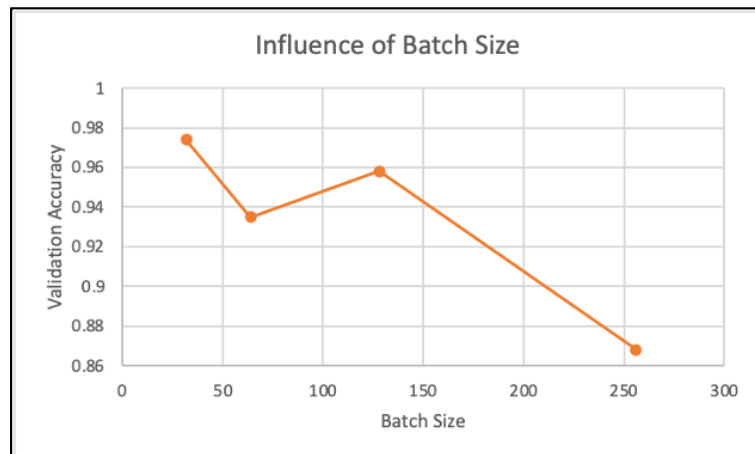
The results of the TensorFlow hyperparameter tuning are summarized in the table below. In total, the tuning for TensorFlow evaluated 29 combinations of hyperparameters. One notable drawback to this analysis was the individual training time for each model stage, which limited this study's ability to explore even more combinations than the 29 presented here. Future analysis may benefit from faster processors or research faster tuning techniques from the TensorFlow library.

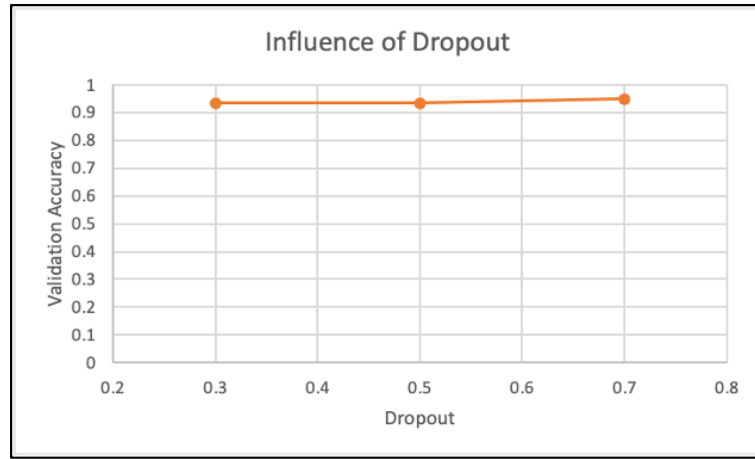
Model Stage	Optimizer	Learning Rate	Batch Size	Dropout	Training Time [min]	Validation Accuracy
1	Adam	0.1	32	0.3	346.1166667	0.56
2	Adam	0.01	32	0.3	333.3333333	0.579
3	Adam	0.001	32	0.3	317.3666667	0.571
4	SGD	0.1	32	0.3	289.15	0.439
5	Adam	0.001	32	0.5	253.75	0.551
6	Adam	0.0001	32	0.5	250.75	0.955
7	Adam	0.00001	32	0.5	251.5	0.974
8	SGD	0.001	32	0.5	212.15	0.918
9	SGD	0.0001	32	0.5	212.1833333	0.903
10	SGD	0.00001	32	0.5	213.6	0.71
11	Adam	0.00001	64	0.7	169.1666667	0.949
12	Adam	0.0001	64	0.7	166.9333333	0.571
13	Adam	0.001	64	0.7	160.8833333	0.561
14	Adam	0.01	64	0.7	210.5333333	0.5467
15	SGD	0.00001	64	0.7	159.6666667	0.676
16	SGD	0.0001	64	0.7	160.5666667	0.82
17	SGD	0.001	64	0.7	160.4333333	0.864
18	SGD	0.01	64	0.7	232.1333333	0.947
19	Adam	0.00001	64	0.3	162.2	0.934
20	Adam	0.0001	64	0.3	164.35	0.801
21	Adam	0.001	64	0.3	164.5166667	0.898
22	Adam	0.01	64	0.3	203.8333333	0.594
23	SGD	0.01	64	0.3	205.316667	0.889
24	SGD	0.001	64	0.3	208.616667	0.877
25	SGD	0.0001	64	0.3	202.366667	0.884
26	SGD	0.00001	64	0.3	203.0333333	0.645
27	Adam	0.00001	64	0.5	132.45	0.935
28	Adam	0.00001	128	0.5	139.5833333	0.958
29	Adam	0.00001	256	0.5	104.9	0.868

The following plot shows the validation accuracy against the training time for each hyperparameter combinations. The point highlighted in orange represents the optimal hyperparameter combination determined by this study (model stage 7).



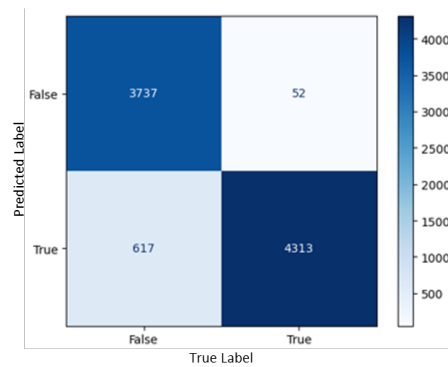
The following three plots show the individual impact of each hyperparameter tested for with the TensorFlow model. The plot for the batch size influence uses model stages 7, 27, 28, and 29. The plot shows that overall, the validation accuracy declines with the batch size. The plot for learning rate's influence shows how the increased learning rate decreases the performance of the Adam optimizer, and increases the performance of the stochastic gradient descent optimizer. The plot also highlights how the Adam optimizer with its optimal learning rate outperformed SGD with its optimal learning rate. The model stages for this plot include 5, 6, 7, 17, 18, and 19. Lastly, the third plot shows the influence of dropout rate, using model stages (8, 14, 27). From this plot, one can see that overall the dropout rate did not impact the validation accuracy much.





With the best performing set of parameters, the following confusion matrix gives more detail in how the model performed during its training. Note that the images labeled false indicate fruit freshness, and images labeled true indicate that the fruit is rotten.

Figure 8: TensorFlow AlexNet Confusion Matrix



False positives in this scenario would indicate that a fresh fruit is falsely labeled as rotten. To avoid additional food waste, a higher precision score is ideal. False negatives in this scenario would indicate that a rotten fruit is falsely labeled as fresh. To avoid selling rotten fruit to consumers, a higher recall score is ideal. It is preferred to have a balance of these costs, so a higher F1-Score would be representative of a model that addresses both false positives and false negative similarly.

For TensorFlow, the confusion matrix suggests a Precision of 0.988, a Recall of 0.875, and an F1-Score of 0.928. Therefore, this model is best at addressing false positives.

4.2 AlexNet Results - PyTorch

As a result of hyperparameter tuning, the best loss score obtained for each batch size, optimizer, learning rate, and number of epochs were as follows:

Batch Size	Optimizer	Learning Rate	Number of Epochs	Loss Score
32	Adamax	0.0013	26	0.00075
64	Adagrad	0.0019	60	0.00053
128	Adagrad	0.0011	32	0.00042
256	RMSprop	0.00044	69	0.0003

Based on the results above, the last set of parameters results in the lowest loss score of around 0.0003. For better comparison, all sets of hyperparameters were used to train on the entire dataset. The training loss scores were measured with each epoch:

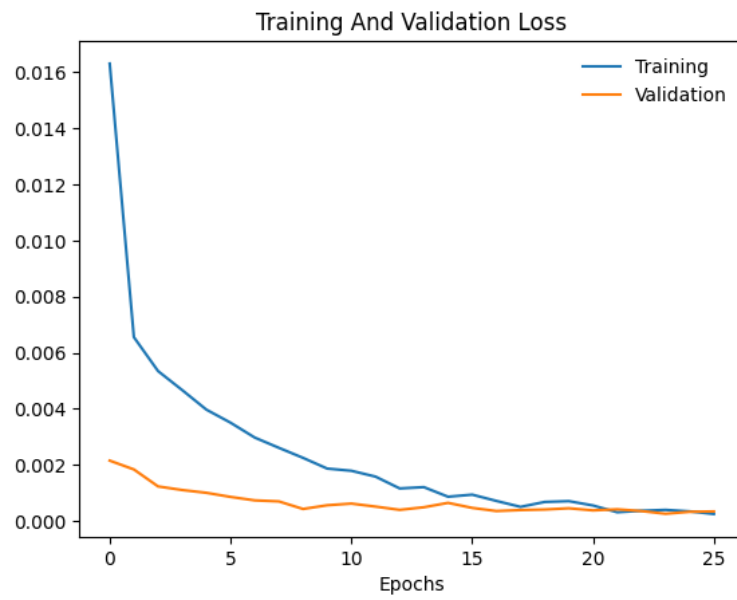


Figure 9: Batch Size 32: Loss Function Value Per Epoch

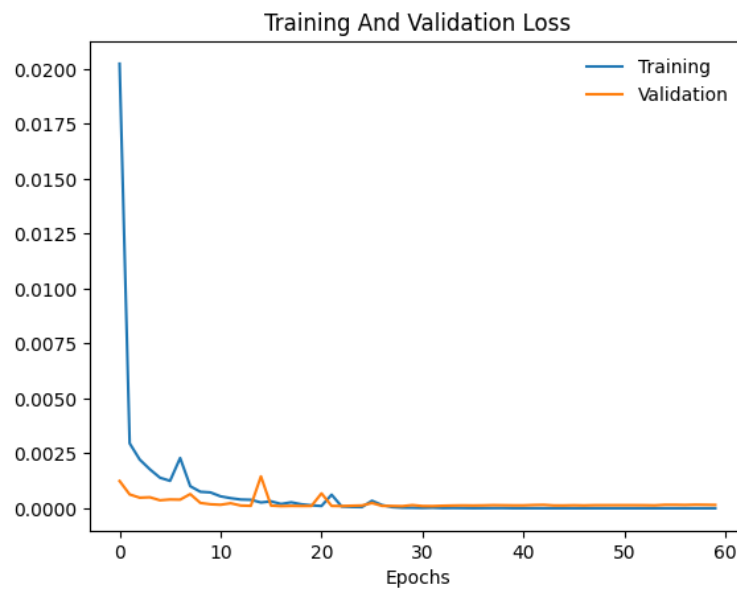


Figure 10: Batch Size 64: Loss Function Value Per Epoch



Figure 11: Batch Size 128: Loss Function Value Per Epoch

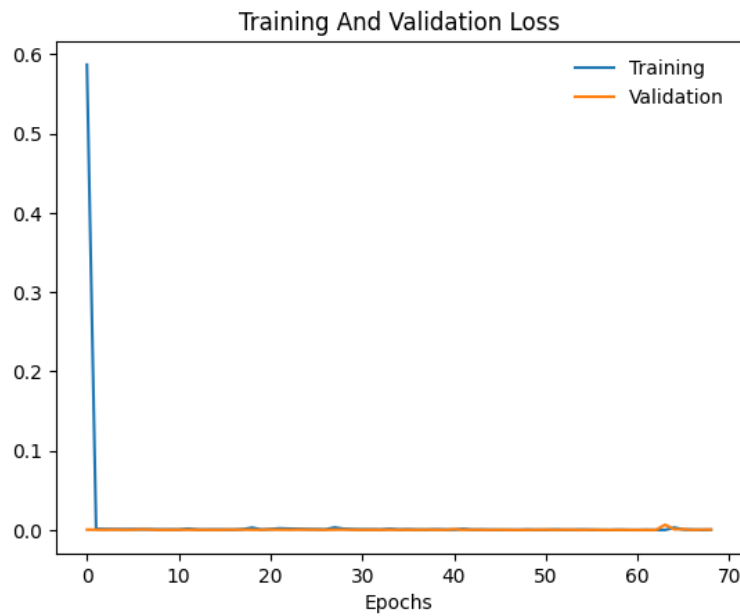


Figure 12: Batch Size 256: Loss Function Value Per Epoch

Based on Figure 6, the loss function score stabilizes between 0 and 0.001 for both the training and validation set. In Figure 7 and 8, the loss function scores improved slightly as they both stabilized between 0 and 0.0025. Figure 9 shows loss function scores that were consistently around 0. The run times and accuracy scores for the entire dataset were also recorded for each set. The following table summarizes these metrics for each set of parameters when applied to the entire dataset:

Table 1: AlexNet Results from PyTorch

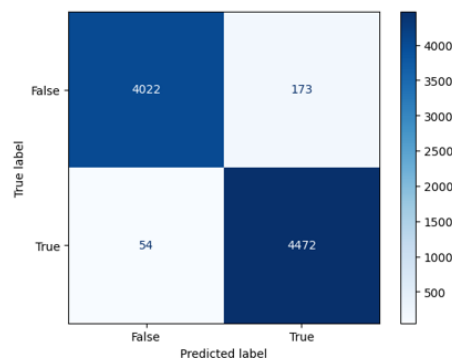
Batch Size	Optimizer	Learning Rate	Number of Epochs	Training Runtime (minutes)	Training Accuracy	Validation Accuracy
32	Adamax	0.0013	26	60	0.9993	0.9821
64	Adagrad	0.0019	60	52	1	0.9908
128	Adagrad	0.0011	32	27	0.9998	0.989
256	RMSprop	0.00044	69	60	0.8335	0.8349

In general, the runs using an Adagrad optimizer gave better accuracy scores over the other optimizers. The third set of hyperparameters gives the fastest training run time of 27 minutes and the highest validation

score of 0.989. Meanwhile, the fourth set of hyperparameters presents a notable disconnect between the performance from tuning and the performance using the entire dataset.

With the third set of parameters, the following confusion matrix gives more detail in how the model performed during its training. Note that the images labeled false indicate fruit freshness, and images labeled true indicate that the fruit is rotten.

Figure 13: PyTorch AlexNet Confusion Matrix



Similarly to TensorFlow, the same logic applies for what false positives and false negatives are representing. This matrix suggests a Precision of 0.963, a Recall of 0.988, and an F1-Score of 0.975. Unlike the TensorFlow AlexNet, the PyTorch model is better at addressing both false positives and false negatives.

4.3 CNN Results

The basic CNN was unable to fit to the data, as suggested by the training and validation loss curves. Additionally, the accuracy scores were too low to justify further use of this model. A deeper model is required for image based fruit classification tasks.

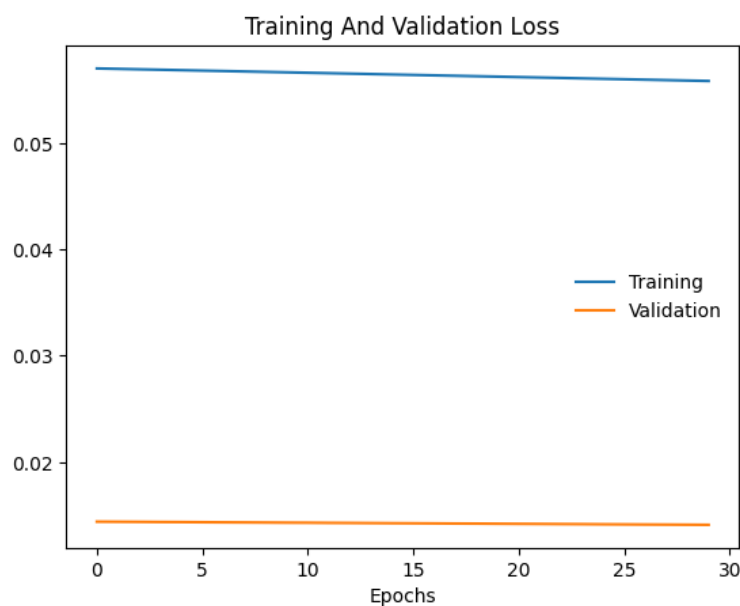


Figure 14: Basic CNN Results from PyTorch

Table 2: CNN Results from Pytorch

	Training Set Size (images)	Validation Set Size (images)	Training Time (minutes)	Training Accuracy	Validation Accuracy
Hyperparamter Tuned Basic CNN	8721	2180	78	0.01	0.01

4.4 AlexNet Plus One

The AlexNet Plus One was run before hyperparameter tuning and after. Before hyperparameter tuning, the base suggested settings for image classification and AlexNet were used. After hyperparameter tuning, the recommended settings were:

Parameter	Optimized Value
Optimizer	Adam
Learning Rate	0.000172
Number of Epochs	84

The two training instances suggest that the AlexNet Plus One might be a viable competitor for the base AlexNet, however additional hyper-parameters should be tweaked before it can be a serious contender. With base tunings, the AlexNet Plus One had not yet converged to an optimal value. With the tuned hyper-parameters, the model had an abnormal spike. This may be prevented with the implementation of an early stopping algorithm or perhaps was an unusual randomly assigned batch of data that caused an anomaly. Due to GPU constraints, further testing of these ideas was not feasible for this project.

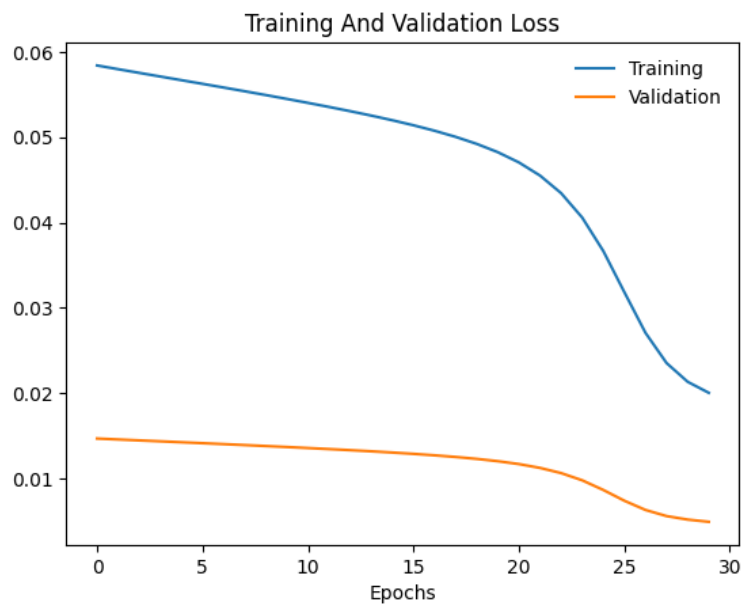


Figure 15: AlexNet Plus One before hyperparameter tuning.

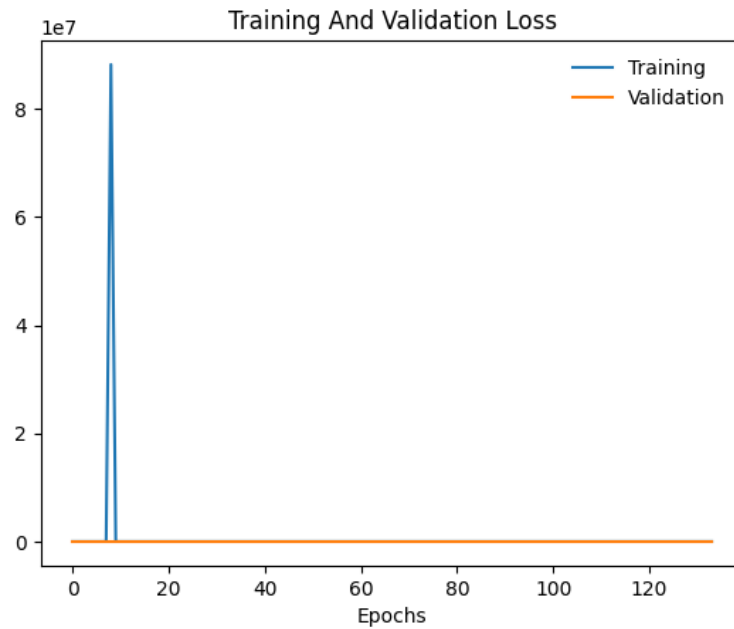


Figure 16: AlexNet Plus One after hyperparamter tuning.

Table 3: AlexNet Plus One Results from PyTorch

	Training Set Size (images)	Validation Set Size (images)	Training Time (minutes)	Training Accuracy	Validation Accuracy
Hyperparamter Tuned AlexNet Plus One	8721	2180	74	0.99	0.99

4.5 Discussion

Both AlexNet implementations proved to be quite effective at predicting fruit freshness, but there were several key differences between the results of the TensorFlow and PyTorch implementations. The first was the difference in run time, with the TensorFlow model taking between 2 and 4 hours to completely train, and the PyTorch model only taking about 10 minutes. This time difference was likely due to a few factors. One factor is the difference in the hyperparameter tuning method, where the tuning for the TensorFlow model was done using a loop of possible parameters that selected the best one, but the PyTorch implementation instead used an existing hyperparameter optimization package to more efficiently search for parameters that would minimize the resulting error of the model. The second factor is the difference in the two packages themselves, where the intrinsic differences in package implementation could also play a role in the final run times.

Table 4: Results of models compared to others.

	Fruit Tested	Runtime (minutes)	Training Accuracy	Validation Accuracy
Proposed AlexNet - TensorFlow	apples, bananas, oranges	252	0.9756	0.974
Proposed AlexNet - PyTorch	apples, bananas, oranges	27	0.9998	0.989
Proposed AlexNet Plus One	apples, bananas, oranges	74	0.99	0.99
Aherwadi AlexNet	bananas	N/A	0.998	0.9944
Proposed Basic CNN	apples, bananas, oranges	78	0.01	0.01
Palakodati CNN	apples, bananas, oranges	N/A	N/A	0.9782

In table 4, the results of our proposed networks are compared against published models for comparison. These models do not contain run times - which is one aspect our work aimed to accomplish. While the accuracies are comparable between the models, without run times, it is hard to say whether our models were better than previously published models. Additionally, the Aherwadi AlexNet would need to be expanded to additional fruit classes in order to gauge generalizability.

5 Conclusions and Future Work

5.1 Conclusions

Between the tested models, the framework to recommend in general would be PyTorch. PyTorch seems to be better suited for scaled work provided that it performed similarly to TensorFlow but at a much faster speed. As of now, the AlexNet is the best model tested for image based fruit classification problems. However, AlexNet Plus One could show promising results with more architecture tinkering. The base CNN was unable to be well-fitted to the dataset - which was to be expected given the complexity of the data. The basic CNN was severely under-fitted and it confirmed the idea that a deeper architecture was necessary in order to achieve better performances.

While HRFormer was originally a potential option for further performance comparison between the AlexNet and custom CNN models, it was unable to be implemented due to blockers encountered with its dependencies on an available NVIDIA GPU and Linux operating system for training, as well as installation errors encountered with the NVIDIA Apex dependency. However, this leaves it as an excellent option for future work, as the concept of high-resolution vision transformers has proven to have great potential for image classification, and has not yet been applied to the problem of food freshness identification.

One limitation to this study was the lack of consistent GPU access - as Google CoLab only allowed 4 hours of use every 24 hours. For future studies, securing constant access to a GPU would speed up the model training process as well as allow for more tinkering with the models.

5.2 Future Work

In the future, this project can be expanded to include more varied data. Additional images of different types of fruits would be the first step to expanding. Additionally, this model can be tested with different types of foods - such as vegetables, breads, etc.

Another possibility might be incorporating a user interface system, such as a smart phone application which would allow people to upload their own photos to see the classification of their foods. This can be expanded as an aid to grocery stores and other food distribution systems.

5.2.1 Improved AlexNets

Since the AlexNet has been released, improvements have been made to the model that allows for less parameters and higher accuracy. This could help reduce computational costs while still providing comparable results [14]. In a future project, variations on the AlexNet can be explored on the fruit classification problem.

5.2.2 Tasks for the Project

In general, most of the tasks set aside for this project were completed on time. However, issues with running the HRFormer, as previously discussed, prevented the implementation. In future work, having access to more operating systems and support for accessing outdated packages/dependencies would assist with running the HRFormer. Alternatively, developing a custom high-resolution model might help circumvent the dependency issue we encountered with HRFormer.

Table 5: Project Tasks

Goal	Accomplished?
Implement AlexNet Model in PyTorch	Yes
Implement AlexNet Model in TensorFlow	Yes
Refine AlexNet Models using Hyperparameter Optimization	Yes
Implement Custom CNN Model	Yes
Implement AlexNet+1 Model	Yes
Implement HRFormer Model	No

References

- [1] David Rolnick, Priya Donti, Lynn Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, Alexandra Luccioni, Tegan Maharaj, Evan Sherwin, s. Karthik Mukkavilli, Konrad Kording, Carla Gomes, Andrew Ng, Demis Hassabis, John Platt, and Y. Bengio. Tackling climate change with machine learning. 06 2019. doi: 10.48550/arXiv.1906.05433.
- [2] Bambang Kuswandi. Freshness sensors for food packaging. In *Reference Module in Food Science*. Elsevier, 2017. ISBN 978-0-08-100596-5. doi: <https://doi.org/10.1016/B978-0-08-100596-5.21876-3>. URL <https://www.sciencedirect.com/science/article/pii/B9780081005965218763>.
- [3] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015. URL <http://arxiv.org/abs/1511.08458>.
- [4] Abhijeet Pujara. Concept of alexnet:- convolutional neural network, Jun 2021. URL <https://medium.com/analytics-vidhya/concept-of-alexnet-convolutional-neural-network-6e73b4f9ee30>.
- [5] Nagnath Aherwadi, Usha Mittal, Jimmy Singla, N. Z. Jhanjhi, Abdulsalam Yassine, and M. Shamim Hossain. Prediction of fruit maturity, quality, and its life using deep learning algorithms. *Electronics*, 11 (24):4100, 2022. doi: 10.3390/electronics11244100.
- [6] Layan Alabdullatif. Complete guide to adam optimization, Sep 2020. URL <https://towardsdatascience.com/complete-guide-to-adam-optimization-1e5f29532c3d>.
- [7] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Minghui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. Deep high-resolution representation learning for visual recognition. *CoRR*, abs/1908.07919, 2019. URL <http://arxiv.org/abs/1908.07919>.
- [8] Yuhang Fu, Minh Nguyen, and Wei Qi Yan. Grading methods for fruit freshness based on deep learning. *SN Computer Science*, 3, 2022. doi: 10.1007/s42979-022-01152-7.
- [9] Sriram Reddy Kalluri. Fruits fresh and rotten for classification, 2018. URL <https://www.kaggle.com/datasets/sriramr/fruits-fresh-and-rotten-for-classification>.
- [10] Yuhui Yuan, Rao Fu, Lang Huang, Weihong Lin, Chao Zhang, Xilin Chen, and Jingdong Wang. Hrformer: High-resolution transformer for dense prediction. *CoRR*, abs/2110.09408, 2021. URL <https://arxiv.org/abs/2110.09408>.
- [11] Venkatesh Wadawadagi. Tensorflow vs pytorch: Deep learning frameworks, 2023. URL <https://www.knowledgehut.com/blog/data-science/pytorch-vs-tensorflow>.
- [12] Shipra Saxena. Introduction to the architecture of alexnet, Mar 2021. URL <https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/>.
- [13] Yuhui Yuan, Rao Fu, Lang Huang, Weihong Lin, Chao Zhang, Xilin Chen, and Jingdong Wang. Hrformer: High-resolution transformer for dense prediction. 2021.
- [14] Shaojuan Li, Lizhi Wang, Jia Li, and Yuan Yao. Image classification algorithm based on improved alexnet. *Journal of Physics: Conference Series*, 1813(1):012051, feb 2021. doi: 10.1088/1742-6596/1813/1/012051. URL <https://dx.doi.org/10.1088/1742-6596/1813/1/012051>.