|  |
| --- |
| SA4110 Machine Learning Application Development |
| Image Classifier |
| GDipSA56 Team 3  Choong Hui Ying Abigail  Jason Tay Neng Wei Muhamad Ashsiddique Bin Mohamed Omrul Hauk  Ng Ka Wing  Thant Thiri Shwe  Thida Aye Win  Seah Isabelle Xu Cong |

|  |
| --- |
|  |

Table of Contents

[Table of Contents 1](#_Toc136115171)

[1 Introduction 2](#_Toc136115172)

[2 Base Model and Results 2](#_Toc136115173)

[3 Data Gathering: Increasing the number of training images 4](#_Toc136115174)

[4 Data Preprocessing: White-space padding to maintain aspect ratio of images 6](#_Toc136115175)

[5 Data Preprocessing: Rescaled image size 9](#_Toc136115176)

[6 Data Preprocessing: Edge Mapping and Channels 10](#_Toc136115177)

[7 Data Augmentation: Image Flipping 12](#_Toc136115178)

[8 Data Augmentation: Image Rotation 13](#_Toc136115179)

[9 Data Augmentation: Translation 14](#_Toc136115180)

[10 Data Augmentation: Zoom 15](#_Toc136115181)

[11 Model Augmentation: Adjusting Layers 16](#_Toc136115182)

[12 Model Augmentation: Adjust kernel size 18](#_Toc136115183)

[13 Model Augmentation: Random oversampling 19](#_Toc136115184)

[14 Summary and Recommendations 21](#_Toc136115185)

# Introduction

In this project, an image Classifier that can correctly identify different types of fruits is developed. The dataset consists of four classes: Apple only, Banana only, Orange only, and a mix of fruits.

The objective is to train a Convolutional Neural Network (CNN) model and employ various techniques to improve its accuracy. In this folder you will find four items:

* This report documents the experiments, results, and insights gained during the process of enhancing the performance of the fruit image classifier.
* (**FruitClassifier.ipynb**) file: A Jupyter notebook that supports this report and shows all experiments.
* (**ML\_Proj\_Data**) folder: A folder which contains the different types of training data utilized, as well as best-performing models (having test accuracy of **0.98**% and **0.93**% respectively).
* (**FruitClassifier-Sandbox.ipynb**): A Jupyter notebook which integrates the numerous variables that tweaked and used in this project, that can be experimented with.

# Base Model and Results

The training data consisted of 75 apple images, 76 banana images, 72 orange images, and 22 mixed-fruits images. Pre-processing of images was limited to rescaling the images to 224 x 224 pixels. No data augmentation was implemented. Please refer to the codes in ***2. Baseline model*** in Jupyter Notebook.

## Base Model Development

There were initial teething issues with the base model, which used `*tf.keras.layers.Flatten()`*.The model could not be loaded because it failed to copy an input tensor. The following code was added to troubleshoot this issue:

*`tf.keras.backend.clear\_session()`*

*`gc.collect()`*

This enabled us to discover that the error originated from an Out-Of-Memory (OOM) *ResourceExhaustedError* when attempting to allocate a tensor with shape[1548800,128].

The initial model, which had two sequential Conv2D layers of kernel\_size (3,3) and an input size of 224x224 pixels was at fault, likely during the flattening stage.

Based on the formula for output size after convolution (Kiao, 2020), the final output image would be of size 220x220 pixels after two convolution cycles (222x222 after first convolution). A tensor of size 1,548,800 (the same as the tensor that threw the error) is obtained from this model setup, which is unfortunately untenable.

As such, `*MaxPooling2D()*` and `*GlobalAveragePooling2D()*` layers replaced `*Flatten()*` in order to reduce the dimensionality of input tensors and prevent OOM errors from re-occurring.

Using `*MaxPooling2D()*` after every single layer was also attempted in order to further reduce the dimensions and avoid the risk of a second OOM occurrence. However, this resulted in poor accuracy and high loss, possibly because the filtered features were being averaged out far too often. This initiative was therefore abandoned.

## Base Model Finalization

The CNN model layer setup consists of several convolutional layers followed by max pooling layers, a global average pooling layer, a dense layer, and a final dense layer for classification. The model is finally compiled using the categorical cross-entropy loss function, the Adam optimizer, and accuracy as the evaluation metric.

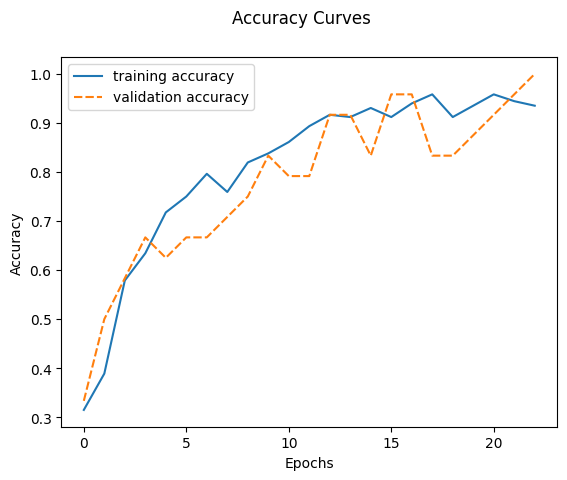
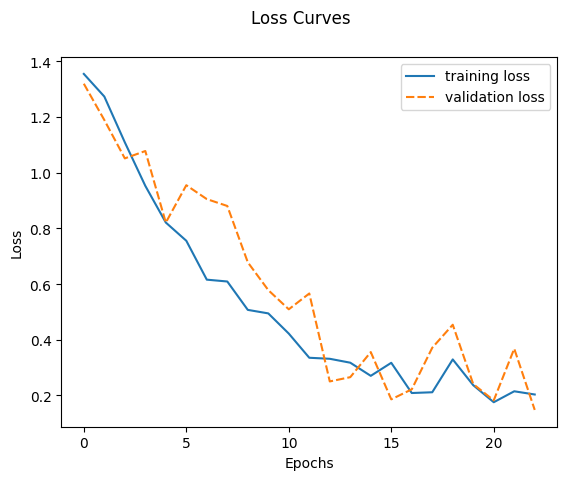
Random seeds were fixed to a value of 42 to maintain consistency across all runs of the training model.

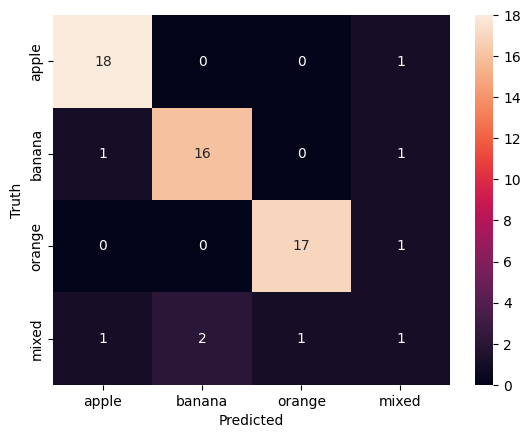
For the model fitting process, the batch size was fixed to 10. Early stopping was implemented to monitor accuracy with patience parameter set to 5. This stops the training process if accuracy does not improve for 5 consecutive epochs and helps to prevent overfitting. Finally, the model was tested against the 19 apple images, 18 banana images, 18 orange images, and 5 mixed fruit images.

## Results

Below are the results for the base model, which was able to identify 52 out of 60 test images correctly.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Loss** | **Accuracy** | **Last Epoch** | **Training Loss**  **(At last epoch)** | **Training Accuracy**  **(At last epoch)** | **Validation Loss**  **(At last epoch)** | **Validation Accuracy**  **(At last epoch)** |
| Else | 0.3578 | 0.8667 | 23 | 0.2036 | 0.9352 | 0.1481 | 1.00 |





# Data Gathering: Increasing the number of training images

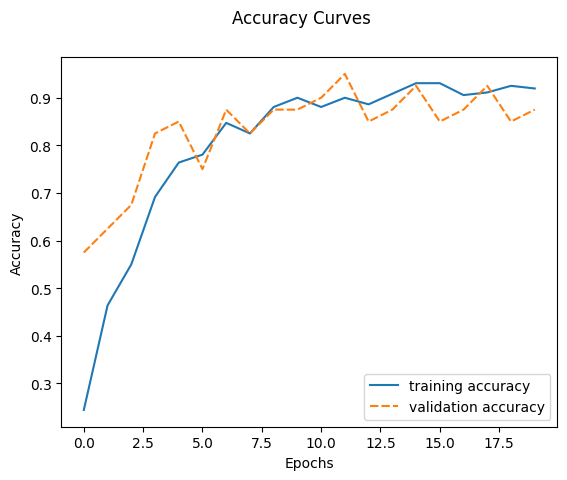
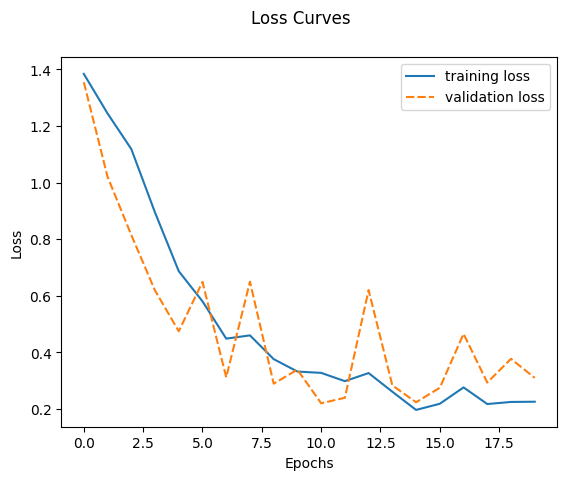
## Modifications

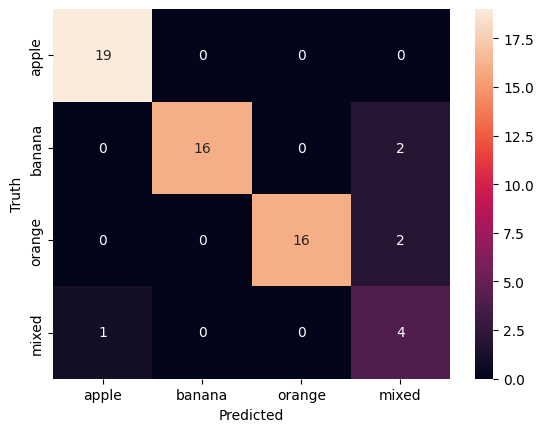
* Number of apple images: Increased from 75 to 100 images
* Number of banana images: Increased from 76 to 100 images
* Number of orange images: Increased from 72 to 100 images
* Number of mixed images: Increased from 22 to 100 images

Please refer to codes in ***3. Increase Training Image Count Manually*** in Jupyter Notebook.

## Results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Loss** | **Accuracy** | **Last Epoch** | **Training Loss**  **(At last epoch)** | **Training Accuracy**  **(At last epoch)** | **Validation Loss**  **(At last epoch)** | **Validation Accuracy**  **(At last epoch)** |
| Else | 0.3265 | 0.9167 | 20 | 0.22500 | 0.9194 | 0.3093 | 0.875 |





## Discussion

|  |  |
| --- | --- |
|  | Accuracy |
| Original Dataset | 0.8667 |
| Enhanced Dataset | 0.9167 |

By manually adding in more images to all classes, the dataset imbalance is reduced, enabling the model to have a more balanced representation across the classes. The model can then learn from a greater variety of patterns and object configurations, improving its ability to generalize and make more accurate predictions. A larger data set also reinforces the model’s learning ability on robust and discriminative features for greater improvement on unseen data. This impact can be seen in the significant improvement in model accuracy % from 86.67% to 91.67%.

# Data Preprocessing: White-space padding to maintain aspect ratio of images

## Modifications

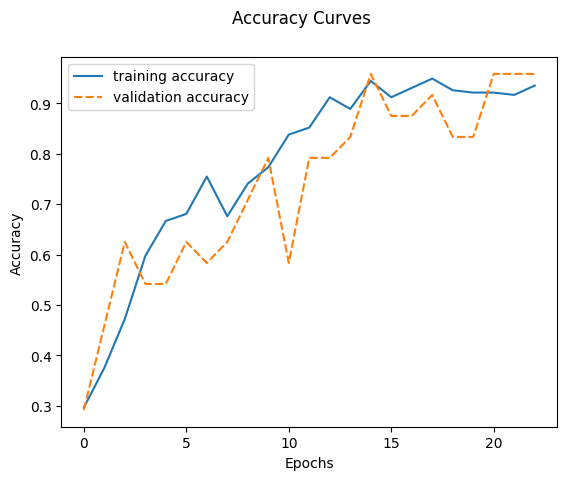
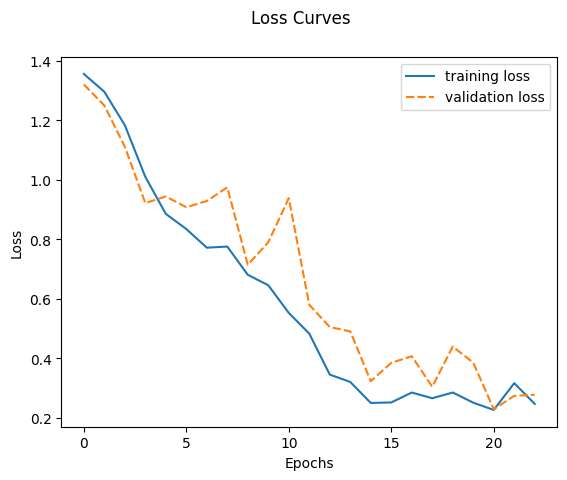
Retain image aspect ratio during rescaling process by proportionally scaling the longer side of the image to match the desired size and then fill the remaining blank space with white pixels. Please refer to codes in ***4. Maintain Aspect Ratio*** in Jupyter Notebook.

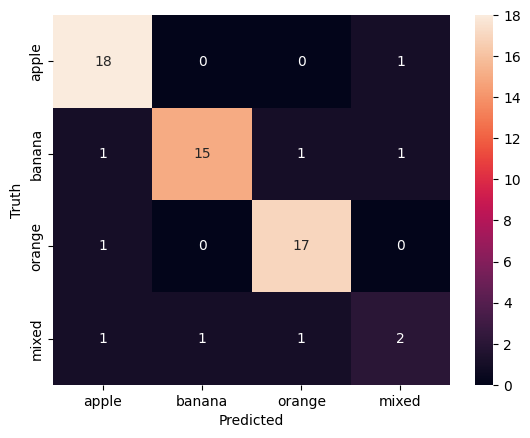


## Results

**Original training dataset:**

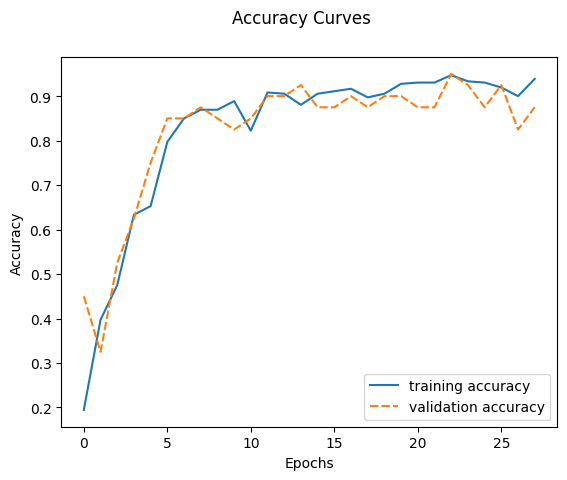
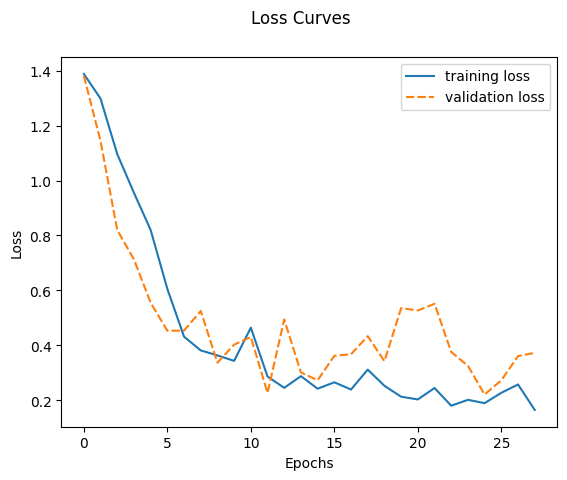
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Loss** | **Accuracy** | **Last Epoch** | **Training Loss**  **(At last epoch)** | **Training Accuracy**  **(At last epoch)** | **Validation Loss**  **(At last epoch)** | **Validation Accuracy**  **(At last epoch)** |
| Else | 0.3774 | 0.8667 | 23 | 0.2460 | 0.9352 | 0.2766 | 0.9583 |

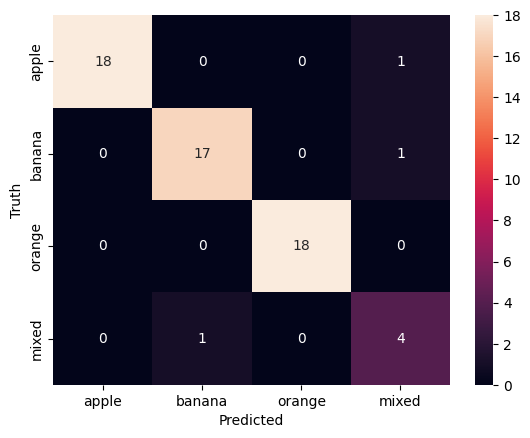




**With balanced and expanded training dataset:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Loss** | **Accuracy** | **Last Epoch** | **Training Loss**  **(At last epoch)** | **Training Accuracy**  **(At last epoch)** | **Validation Loss**  **(At last epoch)** | **Validation Accuracy**  **(At last epoch)** |
| Else | 0.2233 | 0.9500 | 28 | 0.1650 | 0.9389 | 0.3728 | 0.875 |





## Discussion

|  |  |  |
| --- | --- | --- |
|  | Without Aspect Ratio | With Aspect Ratio |
| Original Training Dataset | 0.8667 | 0.8667 |
| Expanded Balanced Training Dataset | 0.91 | 0.95 |

In the base model, training images may appear squashed and elongated after the rescaling step, which may negatively affect the model’s learning and generalizing process.

Rescaling the training images to the desired size while preserving the aspect ratio ensures that the features in the images are not distorted or stretched unnaturally, thus retaining the integrity of visual information of the training data.

When training on the original set of training images the overall accuracy did not improve. However, when training with the expanded and balanced set of training images, preserving the aspect ratio further enhanced the increase in accuracy of the model, further correctly identifying 1 banana and 2 orange test images, though mislabeling 1 apple test image.

It is believed that this is due to the added images being less ‘square’ than the original training images. Thus, while adding the extra training images improved the model’s overall accuracy, it was limited to a certain extent by the distortion caused by the rescaling step. With the distortion removed during the rescaling step, the impact of increasing the training images on the model’s accuracy was more pronounced.

# Data Preprocessing: Rescaled image size

## Modifications

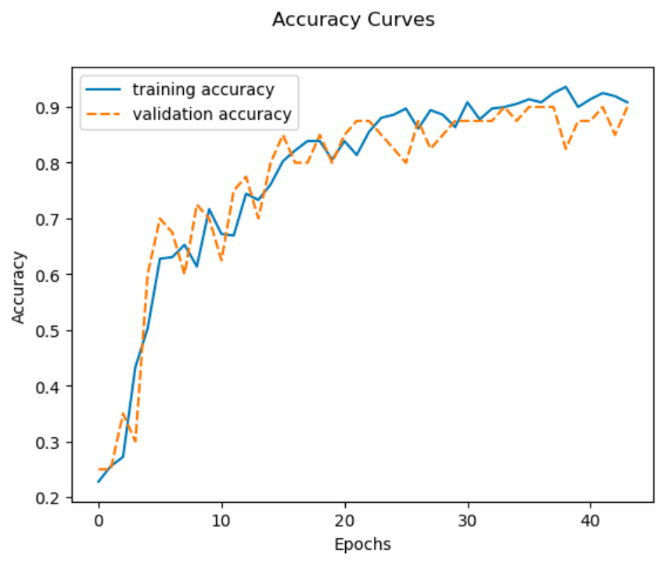
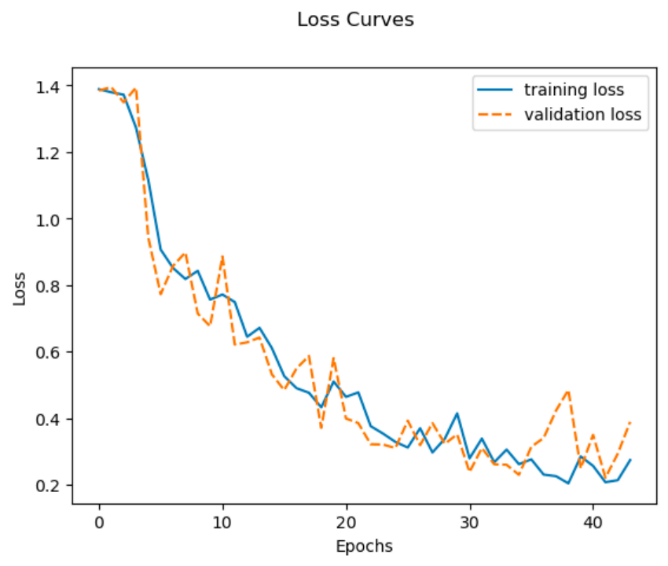
In this run, experiments were conducted to rescale to image size, to test the performance of CNN model. The image size ranges from 64 pixels to 256 pixels. Please refer to the codes in ***5. Image Size Rescaling*** in Jupyter Notebook.

## Results

Below are the accuracy and loss scores for training, validation and test datasets:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Image Size (pixel)** | **Test Loss** | **Test Accuracy** | **Training Loss** | **Training Accuracy** | **Last Epoch** | **Validation Loss** | **Validation Accuracy** |
| 64 | 0.4242 | 0.8500 | 0.2559 | 0.9056 | 33 | 0.2486 | 0.8750 |
| 73 | 0.4446 | 0.8333 | 0.2451 | 0.9111 | 33 | 0.2236 | 0.9250 |
| 98 | 0.5743 | 0.8833 | 0.2738 | 0.9083 | 44 | 0.3886 | 0.9000 |
| 104 | 0.6386 | 0.7167 | 0.4177 | 0.8361 | 21 | 0.4888 | 0.8000 |
| 128 | 0.4956 | 0.8333 | 0.3983 | 0.8639 | 30 | 0.4914 | 0.8250 |
| 224 | 0.7539 | 0.6833 | 0.7949 | 0.6667 | 13 | 0.7428 | 0.7250 |
| 256 | 0.4641 | 0.8333 | 0.3428 | 0.8583 | 44 | 0.7401 | 0.8500 |

Graph of loss curves and accuracy curves with image size = 98 as below:



## Discussion

When using very small image sizes, the model may find it difficult to capture the details in the image, leading to potential overfitting as it merely memorises the training data. This can be seen in the examples when image size is 64 or 73, where the accuracy curve and loss curve are unstable despite the accuracy score being higher.

However, when the image sizes are too large, this introduces more noise, making it challenging for the model to generalize more effectively. There is also an increase in computational demands in processing larger images. This occurs when the image size was at 224 with the model taking the longest to run. This resulted in the loss being the highest at 0.7539 and accuracy being its lowest at 68.33%.

Hence, there is a need to find out a proper image size to balance the model complexity and accuracy while avoiding excessive complexity. In this case, the image size of 98 pixels seems to be the best fit, showing the most stable and consistent accuracy and loss scores across the datasets.

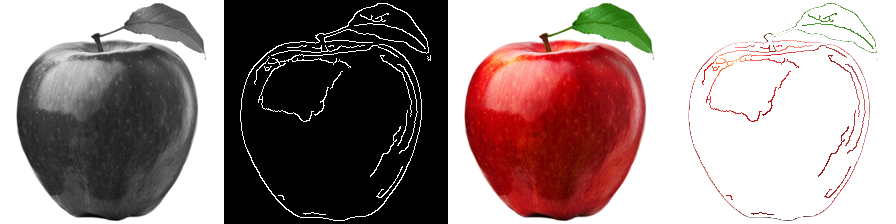
# Data Preprocessing: Edge Mapping and Channels

## Modifications

Four types of channel-based modification were carried out:

1. Changing the image to grayscale, in the interest of reducing dimension count and increasing training speed. This modification is named “Grayscale” in the first parameter of var\_aug (augmentation variant).
2. Extracting a canny map from the image, in the interest of increased training speed in addition to edge recognition. This modification is named “Canny”.
3. Compressing the alpha layers of images that had 4 channels(RGBA) into 3 channels (RGB) to white, in order to reduce dimensionality while simultaneously capturing the base features. This modification is named “RGB”.
4. Compressing the alpha layers as with modification 3), then extracting the canny map and appending it to the fourth channel to provide the model with additional features to learn from. This modification is named “RGBcanny”.

Visual representation from left to right (Grayscale, Canny, RGB, RGBCanny):

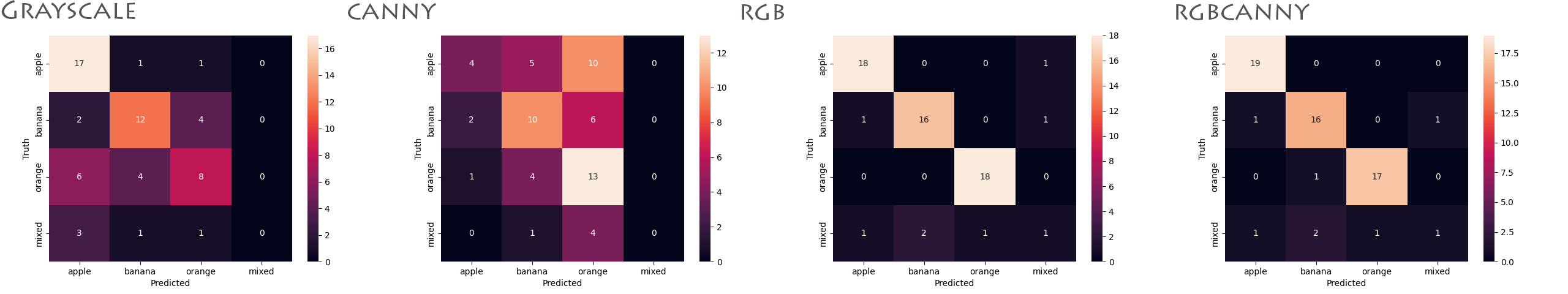
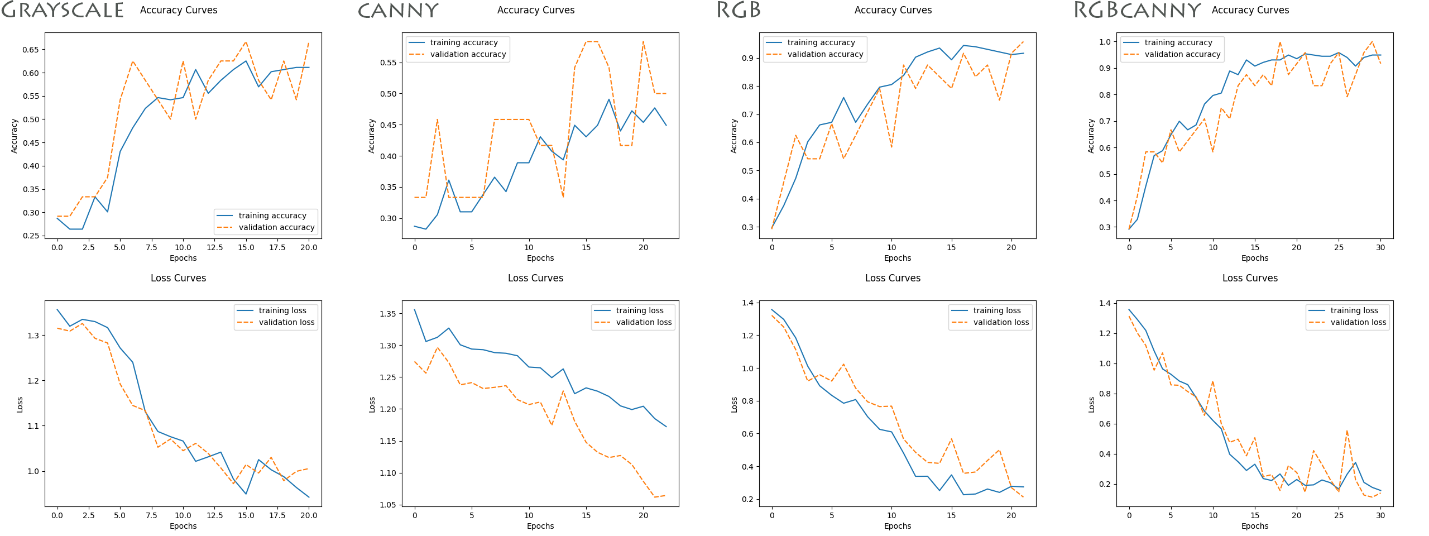


Please refer to the codes in ***6. Data Preprocessing: Edge Mapping and Channels*** in Jupter Notebook for the details.

## Results

Multiple runs were attempted, with other variables kept ceteris paribus while conducting model training. The results while using random\_state= 42 were as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Preprocessing** | Test Loss | Test Accuracy | Training Loss | Training Accuracy | Last Epoch | Validation Loss | Validation Accuracy |
| Grayscale | 0.9568 | 0.6167 | 0.9425 | 0.6111 | 20 | 1.0056 | 0.6667 |
| Canny | 1.2293 | 0.4500 | 1.0642 | 0.4491 | 22 | 1.0643 | 0.5 |
| RGB | 0.4292 | 0.8833 | 0.2742 | 0.9167 | 21 | 0.2116 | 0.9583 |
| RGBcanny | 0.3716 | 0.8833 | 0.1565 | 0.9491 | 30 | 0.1340 | 0.9167 |



## Discussion

It can be deduced that being trained on colored images is easily the most important variable in fruit classification.

Models trained on both RGB and RGBcanny exhibited high levels of accuracy and low loss across all domains (training, validation and testing). Although RGB-trained models tended to have higher accuracy, RGBcanny-trained models tended to have lower loss and more closely-adhering curves (which exhibits slightly less overfitting). It is possible that the extra channel of canny data is providing the model with more information from which to draw spatial patterns from the data, which leads to a better-fitted model.

Grayscale-trained models performed poorly in accuracy and loss, but performed relatively well when it came to fitting. Additionally, canny-only models performed significantly worse than expected, exhibiting very erratic, poorly-fitted graphs with very high loss and low accuracy.

We conclude that relative to color, shape recognition is not a significant factor in fruit classification. Nevertheless, it can be added onto color features to reduce loss and produce a better-fitting graph.

# Data Augmentation: Image Flipping

## Modifications

In this experiment, flipped images are incorporated to enhance the model training. There are 3 flipping methods that are being used - flip images vertically only, flip images horizontally only, and flip images both vertically and horizontally. Please refer to code snippet ***7. Image Augmentation: Image Flipping*** in Jupyter Notebook.

## Results

With the random\_state 42 to control the randomness or reproducibility of the results, the model loss and accuracy scores are as below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Augmented Method** | **Test Loss** | **Test**  **Accuracy** | **Training Loss** | **Training Accuracy** | **Last epoch** | **Validation Loss** | **Validation Accuracy** |
| Else | flip horizontally only | 0.2086 | **0.9333** | 0.2271 | **0.9206** | 34 | 0.5460 | 0.8721 |
| flip vertically only | 0.2674 | 0.9000 | 0.2273 | 0.9176 | 24 | 0.3724 | **0.9070** |
| flip both vertically and horizontally | 0.2275 | **0.9333** | 0.1505 | **0.9441** | 37 | 0.5076 | 0.8721 |

Accuracy scores and loss of flipping images horizontally:

A picture containing text, diagram, line, plot

Description automatically generatedA picture containing text, diagram, plot, line

Description automatically generated

## Discussion

Using flipped images to train the model noticeably increases the performance of the CNN model, and this is supported by the training, validation and test data sets shown above- with accuracy scores ranging from 0.9 to 0.9333 and loss around 0.2. The fluctuations of Loss of validation data may be caused by insufficient data which can be improved in future experiments. The reasons for the better performance might be flipped images provide more variations in object appearances in terms of angles and spatial relationships. With such variations in the training, CNN model becomes less sensitive to specific variations or noise in the training data and promotes better generalization.

# Data Augmentation: Image Rotation

## Modifications

In this run, experiments were conducted to rotate the images 90 degrees clockwise.

## Results

Loss and accuracy metrics:

|  |  |  |
| --- | --- | --- |
| Model | Loss | Accuracy |
| Else | 0.4924 | 0.8833 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Last Epoch | Training Loss  (Last Epoch) | Training Accuracy (Last Epoch) | Validation Loss  (Last Epoch) | Validation Accuracy  (Last Epoch) |
| Else | 14 | 0.373891 | 0.861111 | 0.282613 | 0.916667 |

Here is a sample output of the rotated image:



## Discussion

The method only marginally improved the accuracy (+0.0166) but also introduced greater losses (+0.1345) as compared to the base model.

While the rotation provided more data variations for the model to learn from, the fixed rotation of the dataset might not have provided enough variations for the model to learn from, hence the limited improvement in accuracy. Also, the rotation method might have performed anti-aliasing which might have introduced artificial noise in the data, resulting in more loss.

# Data Augmentation: Translation

## Modifications

In this run, experiments were conducted to translate the images 50 pixels on the x-axis and 30 pixels on the y-axis. Please refer to the code snippet in cell ***8. Image Augmentation: Image Translation*** of the Jupyter notebook.

## Results

Loss and accuracy metric

|  |  |  |
| --- | --- | --- |
| **Model** | **Loss** | **Accuracy** |
| Else | 0.4033 | 0.9000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Last Epoch** | **Training Loss**  **(Last Epoch)** | **Training Accuracy (Last Epoch)** | **Validation Loss**  **(Last Epoch)** | **Validation Accuracy**  **(Last Epoch)** |
| Else | 25 | 0.154918 | 0.953704 | 0.135912 | 0.958333 |
|  |  |  |  |  |  |
|  | | |  | | |

Here is a sample output of the image:



## Discussion

The translation augmentation method produced a higher accuracy score compared to the base model (+0.0333). However, the method also introduced a greater loss as compared to the base model (+0.0455).

The method increased the robustness of the model by introducing more variations in the model for the model to train on, but it might have also introduced outliers which caused spikes in the losses. Nonetheless, the loss and accuracy curves of both the training and validation data are still smoother when compared to the base model.

# Data Augmentation: Zoom

## Modifications

In this run, experiments were conducted to zoom the images with a zoom factor of 2.

## Results

Loss and accuracy metric

|  |  |  |
| --- | --- | --- |
| Model | Loss | Accuracy |
| Else | 0.5342 | 0.8667 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Last Epoch | Training Loss  (Last Epoch) | Training Accuracy (Last Epoch) | Validation Loss  (Last Epoch) | Validation Accuracy  (Last Epoch) |
| Else | 21 | 0.249204 | 0.925926 | 0.359841 | 0.875000 |

## Discussion

The zoom augmentation method produced the same accuracy score as the base model (0.8667). However, the method increased the loss from 0.3578 to 0.5342.

Introducing a zoom factor on the images might have caused a loss in more intricate details of the dataset, which led to the increased loss.

# Model Augmentation: Adjusting Layers

## Modifications

In the runs below, adjusting layers are experimented to test the performance of the CNN model. Please refer to code snippet ***9. Model Augmentation: Adjusting Layers*** in Jupyter Notebook. The following changes were made to the base CNN model:

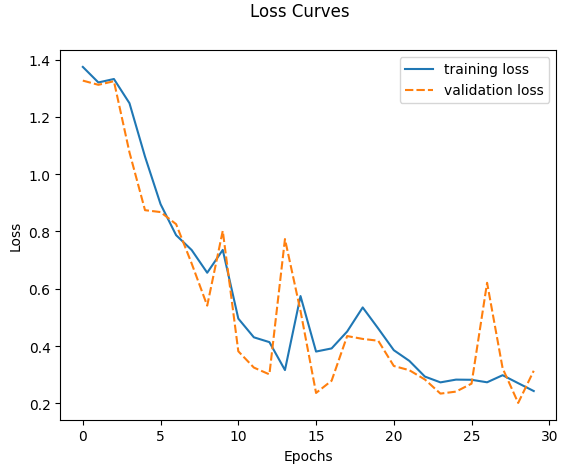
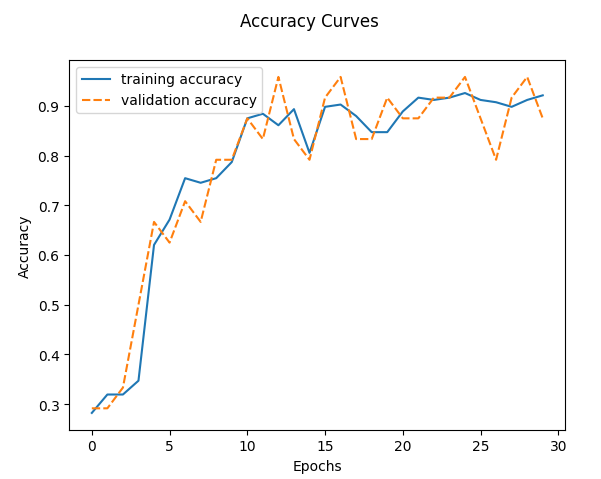
* Experiment 1: removing the dropout layers which were in base model.
* Experiment 2: keeping only one convolutional layer and one dense layer.
* Experiment 3: reordering the dense layer and the Conv2D layer.
* Experiment 4: removing activation function after dense layer.
* Experiment 5: adding convolutional layer.

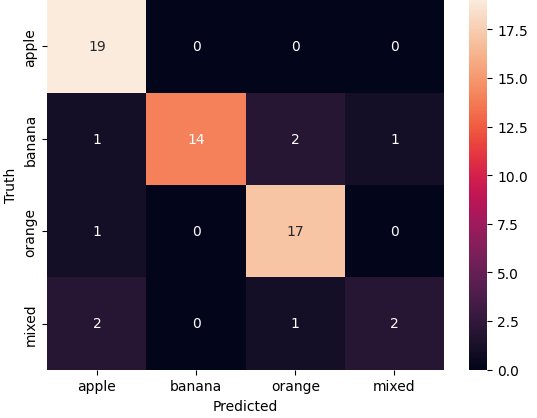
## Results

Below shows the accuracy score and the loss of each experiment listed above (including training, validation and test data):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment** | **Test Loss** | **Test**  **Accuracy** | **Training Loss** | **Training Accuracy** | **Last epoch** | **Validation Loss** | **Validation Accuracy** |
| 1 | 0.8367 | 0.8667 | 0.2434 | 0.9213 | 30 | 0.3141 | 0.8750 |
| 2 | 0.89814 | 0.8945 | 0.2434 | 0.9213 | 30 | 0.3145 | 0.8750 |
| 3 | 0.8667 | 0.8667 | 0.2434 | 0.9213 | 30 | 0.3561 | 0.8910 |
| 4 | 0.8667 | 0.8319 | 0.2512 | 0.3092 | 30 | 0.3472 | 0.8962 |
| 5 | 0.8957 | 0.7333 | 0.5926 | 0.7778 | 30 | 0.7917 | 0.7500 |

Accuracy score and the loss of Experiment 1:





## Discussion

Although the training and validation datasets’ accuracy scores and the loss scores are acceptable, it is noticed that the testing data’s loss score for each experiment is quite high, ranging from 0.83 to 0.89. This may indicate the overfitting situation. It could be due to lack of generalization or insufficient training data.

# Model Augmentation: Adjust kernel size

## Modifications

The experiments involved modifying the kernel size, filter size, and image size in the CNN model. In each experiment, different kernel sizes including (3x3), (7x7) and (5x5), and (9x9) with Filter sizes of 32 and 64 were tested by using Image sizes of 64, 225 and 224. The modifications in kernel size, filter size, and image size aimed to assess their influence on the model's performance. The results indicated that these factors have varying effects on the CNN model. Please refer to the code snippet ***10. Model Augmentation: Adjust Kernel Size*** in Jupyter notebook.

## Results

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model Types | Filter | Image size | Kernal size | Test  Loss | Test  Accuracy | Training Loss  (At last epoch) | Training Accuracy  (At last epoch) | Last  Epoch | Validation Loss  (At last epoch) | Validation Accuracy  (At last epoch) |
| Else | 32 | 225 | (3,3) | 0.5765 | 0.8333 | 0.358998 | 0.857678 | 40/64 | 0.4637 | 0.8333 |
| '000' | 64 | 225 | (3,3) | 0.6526 | 0.8000 | 0.300652 | 0.897674 | 57/64 | 0.3007 | 0.8977 |
| ‘001’ | 32 | 224 | (3,3) | 0.6651 | 0.8167 | 0.437644 | 0.842697 | 32/64 | 0.1921 | 0.9333 |
| ‘002’ | 32 | 64 | (7,7) (5,5) | 0.6811 | 0.8500 | 0.253718 | 0.908046 | 45/64 | 0.2537 | 0.9080 |
| ‘9x9’ | 64 | 64 | (9,9) | 1.383 | 0.30 | 1.386299 | 0.249042 | 14/64 | 1.3863 | 0.2490 |

## Discussion

Based on the results, it can be deduced that adjusting the kernel size can have a significant impact on the accuracy scores of a CNN model. With larger kernel sizes, the model can capture more spatial information and larger patterns in the input data. This is helpful for complex images and objects that require broader context for increased accuracy. From the above table, sizes such as (5x5) and (7x7) show an increase in the test and validation accuracy. However, too large a kernel size will lead to information loss and blurring of details, reducing the accuracy. The model will be less sensitive to details and generalize the information a lot more as compared to smaller kernel sizes. This can be seen when the kernel size increases to (9x9), showing an exponential drop in test and validation accuracy.

On the flip side, smaller kernel sizes allow for increased focus on features and details in the images which can be beneficial for objects with very detailed patterns. However, too small a kernel size may result in a reduced understanding of important global patterns and generalization of the patterns, in turn reducing accuracy.

With the results, it can be seen the optimal kernel size is at about (5x5) or (7x7) as it gives the most balanced test and validation accuracy results.

|  |  |
| --- | --- |
|  |  |
|  |  |

# Model Augmentation: Random oversampling

## Modifications

In this experiment, the RandomOverSampler Library is used to balance the datasets, after which, the datasets are with:

* Increased representation of minority class
* Better balanced class distribution

Please refer to codes in ***11. Model Augmentation: Random oversampling*** in Jupyter notebook.

## Results

|  |  |  |
| --- | --- | --- |
| Model | Loss | Accuracy |
| Else | 0.1166 | 0.9500 |

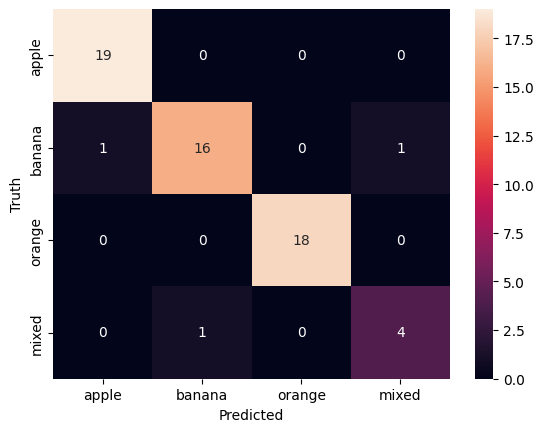
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Last Epoch | Training Loss  (Last Epoch) | Training Accuracy (Last Epoch) | Validation Loss  (Last Epoch) | Validation Accuracy  (Last Epoch) |
| Else | 46 | 0.1865 | 0.9365 | 0.4703 | 0.9412 |

Accuracy scores and the loss scores for training and validation datasets:

A graph with blue and orange lines

Description automatically generated with low confidence A picture containing text, diagram, line, plot

Description automatically generated



## Discussion

The obtained results suggest that random oversampling is a valuable technique for addressing class imbalance in datasets, improving the test accuracy significantly from 86.67% to 95%.

Through randomly replicating and generating new instances of the Mixed fruits class, it provides a more robust and representative dataset. This can improve the model’s ability to differentiate and classify instances from the minority class, resulting in higher accuracy.

Nevertheless, it’s important to note that random oversampling can lead to overfitting, as synthetic samples are generated based on the existing minority class instances. As training accuracy increases, the test and validation accuracy may not as the model becomes too specialized by memorising the training set. Therefore, it is recommended to evaluate the performance of the models trained on the oversampled dataset using appropriate validation methods.

# Summary and Recommendations

In this project, different methods and techniques are employed to enhance the performance of the CNN image classifier model, including:

* increasing training dataset’s size,
* improving training dataset’s accuracy (correcting mislabeled pictures),
* adjusting model layers & kernel sizes & model filter sizes,
* augmentation techniques including padding & flipping & rotating & translating & zooming & scaling images, edge mapping and channels, random sampling etc.

While increasing dataset’s size, correcting mislabeled pictures, zooming & rotating images and edge mapping are useful in increasing the model’s accuracy score and reduce the loss, it is noticeable that padding, flipping, translating images and oversampling techniques play more significant roles in improving the model's accuracy and robustness. Nonetheless, adjusting the layers & kernel size & model filter sizes & image sizes don’t make remarkable changes to the performance of the model.

Therefore, it is recommended to consider a diverse range of data manipulation & augmentation techniques when training the CNN model as each technique or method brings unique benefits and contributes to better model performance. By having a more balanced colored training dataset, combined with augmentation techniques such as flipping & translation & oversampling, the CNN model's ability to generalize and have more accurate predictions would most likely to be improved significantly.

**Reference:**

Ue, K. (2020, April 15). Calculate output size of convolution. OpenGenus IQ: Computing Expertise &amp; Legacy. https://iq.opengenus.org/output-size-of-convolution/