VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

COMP309 Project
Image Classification based on Convolutional Neural Networks

Han Han (300474699)

Supervisor(s): Xue Bing

Introduction

An image classification pipeline was built using python to classify an RGB images dataset into 3 classes: cherry, strawberry and tomato. A CNNs model was constructed using Keras and trained. The hyper-parameters were tuned and optimised. As a result, the classification accuracy was improved to 98.6% on test set (30% of the original dataset). If using 5-fold cross validation to tune the model with the same hyper-parameters but a smaller (2/3) training dataset (due to the constraint of hardware RAM capacity), the classification accuracy was 89.20% (+/- 1.34%) on test set (10% of the original dataset).

Problem Investigation

4500 RGB images were given as the training data. First of all, they need to be read into memory for computer to process.

Reading Images Data from Files

All the images in the designated directory were read into memory one by one. Each of them was stored as a NumPy array with dimension of (300, 300, 3). The first two dimensions record the brightness information of the 300 pixels in x and y direction, respectively; the last dimension records the colour information (R, G, B).

```
for imgname in os.listdir(directory):
    img       = Image.open(directory + imgname)
    img_array = img_to_array(img)
```

Storing Images Data into NumPy Arrays

2 dynamic 1-dimensional array variables were defined:

```
locals()['X_list_'+str(last_2digit).zfill(2)]
locals()['y_list_'+str(last_2digit).zfill(2)]
```

which are actually 200 array variables `X_list_00, X_list_01, …, X_list_99` and `y_list_00, y_list_01, …, y_list_99.`

All the images in the folder with the last 2 digits of '00' before '.jpg' were stored in `X_list_00`; all the images in the folder with the last 2 digits of '01' before '.jpg' were stored in `X_list_01`, … Their corresponding labels were stored in `y_list_00, y_list_01`, and so on.

The reason why using 200 variables to store the whole dataset is because it's more time-saving than if using only 2 variables `X_list, y_list`, because when more images are read into memory, the appended/concatenated array become larger and takes much longer time in operation. Other options like 20 and 2000 variables were also tested, 200 variables (100 X_lists, 100 y_lists) gave the highest reading speed.

Reshape and Concatenate the 100 X_lists into one 4-dimension Array

All the 100 X_lists were reshaped to (-1, 300, 300, 3) for concatenating. Only in this way, the data can be pre-processed to the input shape of a convolutional layer in the CNNs will be built later on.

Normalization is also a very important step when pre-processing the raw data. Because the array for each image actually records the brightness data on each pixel for the 3 colours, the range of the values in all the arrays are from 0 to 255. So, the arrays were divided by 255.0 to normalize the data into range 0 to 1.

```
    for last_2digit in range(100):
        locals()['X_'+str(last_2digit).zfill(2)] =
                                    locals()['X_'+str(last_2digit).zfill(2)] / 255.0
```

In case of missing last-2-digit (eg. there is no image file ending with 00 or 03, etc.), the smallest last-2-digit was checked first before any further data operating, like concatenation.

Finally, the 100 X arrays and 100 y vectors were concatenated to 2 parameters X and y, which finished the data reading process.

```
for last_2digit in range(min_last_two_digit + 1, 100):
    X = np.concatenate((X, locals()['X_'+str(last_2digit).zfill(2)]))
    y = np.concatenate((y, locals()['y_list_'+str(last_2digit).zfill(2)]))
```

Garbage Collection

Thousands of images in this application take a lot of memory space, especially during the reading and pre-processing, many temporary variables were created. They need to be deleted in time to save space for the images data by `del` and `gc.collect()`. Before that, variables in RAM were listed by `dir()` and the sizes were checked by `sys.getsizeof()`.

EDA

Statistical information was checked for X (the input of the whole dataset). The results are within expectation. The relatively low average value for the input dataset implied that the input images are a bit dark in average.

```
    Max:    1.0
    Min:    0.0
    Mean:   0.43472528
    STD:    0.3138554
    Median: 0.4
```

Test Reading

By test reading the first 10 images data in parameter X and comparing to the original images,

```
      for a in range(10):
          new_im = Image.fromarray(X[a].astype('uint8'))
          new_im.show()
```

X can be considered as reading successfully.

Generate 1-hot Array for y

1-hot array was generated for y by below code for the model construction in a later step. Then, the parameter y was cleared.

```
    encoder = LabelBinarizer()
    y_1hot  = encoder.fit_transform(y)
```

Returning Data

`X_train, X_test, y_train` and `y_test` were split into by `train_test_split()` and returned for the next model construction step. The `test_size` was set to `0.3`. And, shuffling the data here is a good choice.

Error Data

Image strawberry_1581.jpg caused error during reading process. In the beginning, it was found that no data can be read into X_list_81 array, which means all the images with the 81 ending were missed. By renaming all the 81 ending files, strawberry_1581.jpg was found caused the error and was removed from the dataset. All the other images can be read in to memory successfully. Hence, the parameter X includes the data from 4499 images in total.

Methodology

The structure of the CNN model was designed as below:

Layer	Activation Function	Setting
Convolutional Layer 1	relu	filters = 64, kernel_size = 5, strides = 3
Max Pooling Layer 1	-	pool_size = 4, strides = 1
Convolutional Layer 2	relu	filters = 16, kernel_size = 7, strides = 5
Max Pooling Layer 2	-	pool_size = 6, strides = 2
Flatten Layer	-	-
Dense Layer 1	relu	units = 32
Dense Layer 2	softmax	units = 3

Then, the model was compiled using `Adam` optimizer with the learning rate of `1e-4`, the loss function was chosen as `'categorical_crossentropy'`. `'accuracy'` was selected as performance metrics for this classification case.

The model was fit by `X_train` and `y_train` as training data with `100 epochs`. The validation data was set to `(X_test, y_test)` which are 30% of the original dataset.

1. Hyper-parameter Settings

The hyper-parameters in the CNN was scanned and combined by looping tests. The parameters used in the tuning process are listed below:

	Parameter	Tested Value	Final Setting	Remarks
Convolutional Layer 1	filters	4, 8, 16, 32, 64	64	the larger the better
	kernel_size	2, 3, 4, 5	5	the larger the better
	strides	1, 2, 3	3	the larger the better
Pooling Layer 1	pool_size	1, 2, 3, 4	4	the larger the better
	strides	1, 2	1	
Convolutional Layer 2	filters	4, 8, 16	16	the larger the better
	kernel_size	3, 5, 7	7	the larger the better
	strides	1, 2, 3, 4, 5	5	the larger the better
Pooling	pool_size	1, 2, 3, 4, 5, 6	6	the larger the better

Layer 2	strides	1, 2	2	
Dense Layer 1	units	8, 16, 32	32	the larger the better
Dropout Layer 1~4	rate	0, 0.1, 0.2, 0.3	0	no dropout gives the best result
Compile	lr	5e-5, 1e-4, 5e-4	1e-4	
Fit	epochs	10~100	100	

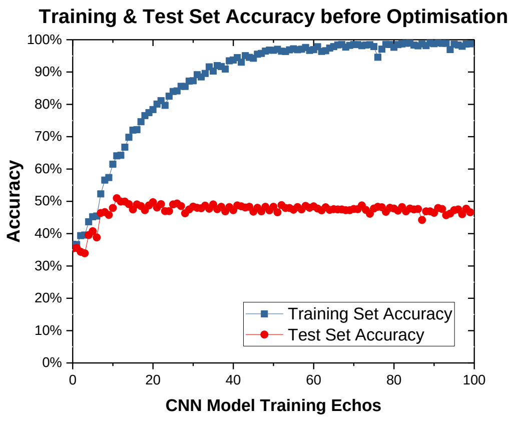
For most of the parameters in above table, the larger they were set, the better model trained (in term of higher accuracy and lower loss in both 70% training set and 30% test set). However, the strides in pooling layers didn't show the same trend. This may be because when the strides parameter is larger, more information in the figures are missed.

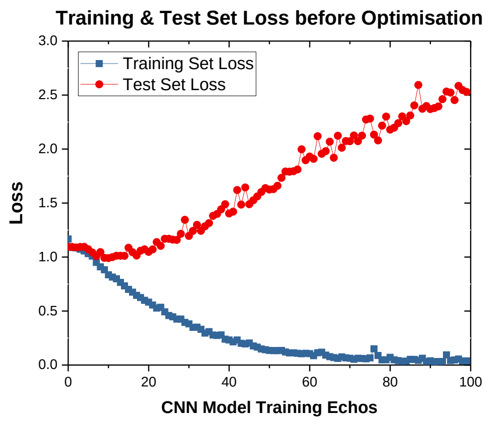
For the 1st dense layer, the accuracy of the model improves as the units number increases from 8 to 16 to 32, but if more neurons are added, the accuracy didn't improve any more, and the overfitting problem becomes more serious.

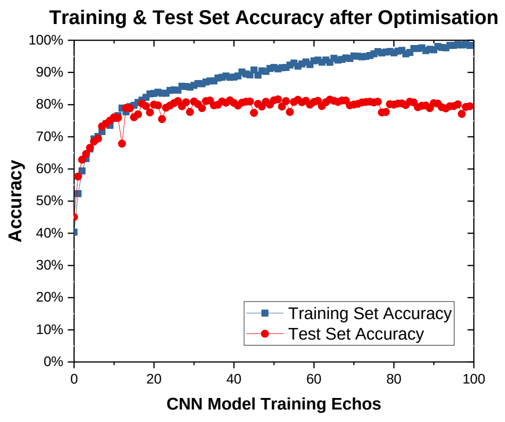
4 dropout layers were added to the model between each main core layers to test whether they can help to improve the overfitting. But the accuracy does not increase but drop even with only 10% drop rate in one of the dropout layers. This may be due to the training set is not large enough.

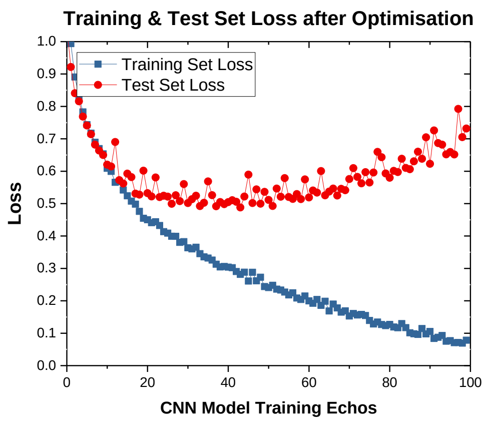
Learning rate and epochs were also adjusted in compiling and fitting steps, respectively. 1e-4 gave the best and fastest result. The epochs number was set to 100 to make sure the model was fully trained by the training set (till almost 1.00 of accuracy on training set).

The accuracy and loss for 70% training set and 30% test set before and after parameter optimisation were compared and showed below:









We can see that, after optimisation, the accuracy of test set improved from 50% to 80%; the loss also performed better. However, from the plots, over-fitting is found after around 20 epochs, which means more actions need to be done to improve the model.

2. Activation Function & Loss Function for Convolutional Layers and Dense Layers

Relu was selected as activation function in convolutional layers to introduce non-linearity into the neural network. If without activation function like Relu, the output of neural network will be the result of a linear combination among inputs. Activation function helps the model to approximate non-linear function in any shape, which is more closed to real world problem.

Compared to using sigmoid and tanh as activation function, the amount of calculation in both forward propagation and back propagation if using Relu are smaller. And gradient vanishing problem is less possible to happen than using sigmoid or tanh as activation function (because of the low gradient and low gradient change rate in their saturation zones), which may cause information loss. Relu (outputs 0 at some neurons) can also alleviate overfitting by reducing the inter-dependency between weights.

3. Activation Function & Loss Function for Output Layer

Sigmoid and Softmax are the common activation functions for the output layer of a neural network, which are used for binary classification and multi-class classification, respectively. Binary cross-entropy and categorical cross-entropy are the corresponding loss functions. In this project, because we have 3 classes, softmax was chosen as the activation function.

The matching between activation functions and loss functions are key for the error of neurons in output layer to be equal to the difference between the output and the ground truth during the back propagation. Hence, softmax and categorical cross-entropy were chosen in the output dense layer.

4. Optimisation Method

Several optimisation methods were studied in this project. SGD was the first one. However, the gradient using SGD falls slowly and vibrates in a large range. If using momentum setting in the SGD, the previous gradient descent information can also be taken into account. The descending of gradient will be faster in a higher slop area, and then be slower in a lower slop area. Nesterov acceleration can help to find global minimum better than pure SGD by considering both the current gradient and the first derivative of historical gradient.

Then, the AdaGrad optimizer was tried. This adaptive optimizer learns more on the weights updated not frequently but keeps low learning rates for frequently updated weights. In this way, we can learn more on samples with more occasionally-appeared features. The historical updating frequencies were calculated by the second derivative of the gradient.

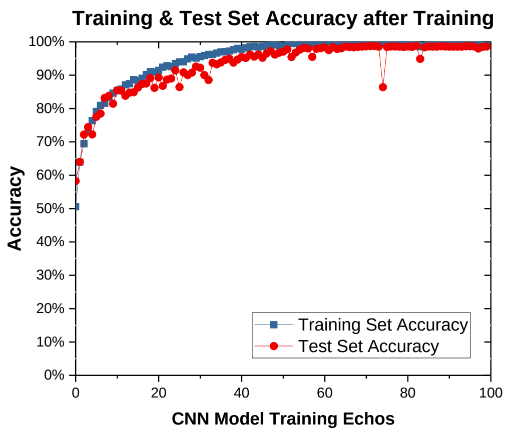
Finally, Adam was chosen as the optimisation method, because it combined the advantages of both the first derivative and the second derivative from SGD Momentum and AdaGrad, and gave good result.

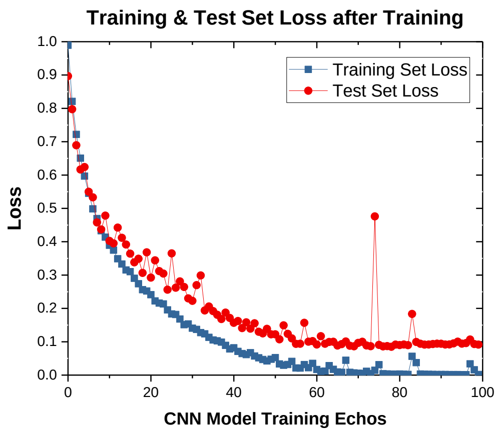
5. Data Augmentation

The training data set was tripled to 4500 * 3, by rotating the original 4500 figures by -90-degree, 0-degree and +90-degree. In this way, the model can learn more in different angle of the targeted objects, and can also prevent over-fitting.

After trained by 13500 figures, it can be seen from the plot on the right side below, the over-fitting issue was alleviated, comparing to if use only the 4500 original figures. The loss of the test set is also more closed to the loss of the training set.

The accuracy of the 30% test set reaches 98.6%, which follows the improvement of training set accuracy quite closely during the whole training process.





6. K-fold Cross-Validation

5-fold cross-validation was also used to train models on the original training set and a training set combined the original and +90-degree rotation datasets (4500*2 figures in total), with 10% test set, 70% and 30% in the other 90% as training set and validation set. The reason why a larger data set (4500*3) or a higher fold wasn't used is because the training process uses more RAM than without using cross-validation, which made the training process out of memory.

The performance of different validation methods on different training datasets are compared below:

Cross-Validation	Dataset	Instance	Accuracy	Loss
without	Original	4500	80.07%	0.7050
5-fold	Original	4500	76.71% (+/- 2.00%)	0.9202 (+/- 0.1101)
without	Original + 90-deg	9000	93.33%	0.3916
5-fold	Original + 90-deg	9000	89.20% (+/- 1.34%)	0.6174 (+/- 0.0758)
without	Original + 90-deg + -90-deg	135000	98.60%	0.0913

We can see that, as the instance number in training set increases, the accuracy increases (from 80.07% to 98.60%), the loss is also improved (from 0.7050 to 0.0913). If using 5-fold cross validation, the average accuracy is around 4% lower than without using cross validation, which is reasonable. The variation of accuracy and loss among the 5 folds are also in a small range, which shows the robustness of the models trained in a certain degree.

7. Model Test

300 new figures (100 figures for each fruit) were downloaded from Google Image randomly as a separate test set to evaluate the performance of the 5 models trained above.

test.py was modified to work with train.py, mainly because they were using different image-reading library. PIL.Image is the one used in the train.py which read the images into NumPy arrays in the sequence of R, G and B. However, in test.py, the images were read into arrays by methods in cv2 library. The sequence is B, G and R.

Hence, below code was added to the test.py to convert BGR to RGB for all the test images so that they can be evaluated by the model.

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

The 5 models were applied on the 300 new figures. The performance of the models on this 300 images test set are compared below:

Cross-Validation	Dataset	Instance	Accuracy	Loss
without	Original	4500	92.33%	0.2718
5-fold	Original	4500	92.67%	0.1925
without	Original + 90-deg	9000	92.99%	0.3046
5-fold	Original + 90-deg	9000	88.00%	0.6088
without	Original + 90-deg + -90-deg	135000	93.67%	0.2819

The accuracy for all these 5 models are around 90%, which are not unsatisfactory. The loss are around 0.2-0.3, except for an outlier. The performance of these 5 models on new data does not show the difference as test on part of the original data set. This may be because most of the figures I chose from Google Image show the target items clearly in clean background, which may be easy for all the 5 models to classify. If I selected more pictures with more complicated picture composition, the performance difference might be more obvious.

Result Discussions

The parameters of the baseline method MLP and the final method after tuning are listed below:

	Parameter	Original Setting	Final Setting
Convolutional Layer 1	filters	32	64
	kernel_size	5	5
	strides	1	3
	padding	'same'	'same'
Pooling Layer 1	pool_size	2	4
	strides	2	1
	padding	'same'	'same'
Convolutional Layer 2	filters	64	16
	kernel_size	5	7
	strides	1	5
	padding	'same'	'same'
Pooling Layer 2	pool_size	2	6
	strides	2	2
	padding	'same'	'same'
Dense Layer 1	units	1024	32
Compile	lr	1e-4	1e-4
Fit	epochs	1	100

The prediction performance was improved a lot, in terms of classification accuracy, from below 40% initially to 80%. And after the data augmentation, the accuracy was improved to 98%.

During the tuning process, looping the combinations among different parameters was done. And 680 combinations among 10 CNN parameters and their results (accuracy, loss in test set) were recorded. 2 simple linear regression models were built to explore whether there are stronger correlations between the test results and certain parameters than other parameters.

The result is shown below:

10-fold Cross Validation	accuracy_test = 0.0204 * Conv1filter + 0.0197 * Pool1size + 0.0137 * Pool2size + 0.0135 * Pool2stride + 0.0109 * dense1neuron + 0.0061 * Conv2stride + 0.0043 * Conv2kernel + 0.3481	loss_test = -0.032 * Conv1filter + -0.0295 * Pool1size + -0.0294 * Pool2size + -0.0172 * dense1neuron + -0.0164 * Conv2stride + -0.015 * Pool2stride + -0.0078 * Conv1stride + 1.2821
Correlation coefficient	0.773	0.8655

* filter numbers in conv layers & neurons in dense layers were converted to $\log_2 n$ for linearity

The number of filters in the first convolution layer contributes the most in improving both accuracy and loss performance, followed by pooling sizes for the 2 pooling layers, which have comparable importance.

Comparing to these parameters affected the performance of CNN the most in this case, the kernel size and stride in convolutional layer 1 and filter number in convolutional layer 2 seem not affect the model too much.

Since the model parameters were initialised in small numbers, and adjusted to higher numbers, the regression equations indicate that the larger the setting (the larger the CNN), the better the result. However, this is not the case. It only applies in a certain range. When the CNN is built to be too large, the overfitting problem can be worse, even sometimes the accuracy keeps going up.

The model building time did not change a lot after layers adding to the model and parameters tuning (all within 10 seconds). But, if the epochs parameter was set to a larger number, or more images were used in training, or cross validation was used, the training time would be longer correspondingly. Normally, the whole training time for one parameters set is proportional to the epochs number and the folds of cross validation.

Conclusions and Future Work

A CNN model was built in this study to classify 3 kinds of fruits from images. 98.6% accuracy was achieved after initial CNN model building, parameters tuning and data augmentation. The model was tested on new data downloaded from Google Image. The result shows the model built is scalable to a certain degree.

Although the model can achieve a relatively high accuracy in this project, the loss performance is not very stable, and has the tendency to overfitting, which can be studied and improved further.

Dropout (which was only used in training 4500 images dataset) may help to improve the slight overfit after the data augmentation.

During the whole training process in this project, the out of memory issue happened for several times. However, 4500 images or even 13500 images cannot be considered as big data. There must be a better solution to overcome this issue. Stream seems a good start.