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-CNN image Classification Project Report

Jiawei Luo

300434604

Abstract

Convolutional neural networks(CNN) have been widely studied and used in various industries. In this paper, The objective is to solve an Image Recognition Problem for cherries, strawberries and tomatoes. A baseline model with one convolutional layer was constructed initially. Also, A CNN with multiple layers of convolutional layers and pooling layers is developed based on the Baseline. The Baseline and the CNN model achieved 62.56%and 83.30% accuracy, respectively.

1. *Introduction*

In the field of modern artificial intelligence, Convolutional neural networks(CNN) have been widely studied and used in various industries. Among them, image recognition is the most important application of CNN. In this paper, The objective is to solve an Image Recognition Problem for cherries, strawberries and tomatoes. A baseline model with one convolutional layer was constructed initially to understand the process of neural networks and obtain the fundamental performance matrix of the model. After that, a CNN with multiple layers of convolutional layers and pooling layers is developed based on the baseline. The model that be constructed by CNN should correctly classify the label of a given image between cherry, strawberry and tomato.

2. *EDA Discussion and Problem investigation*

2.1. *Initial data analysis*

The initial dataset contains 3 Classes, which is three kinds of Fruit: Cherry, Strawberry and Tomato. The original dataset was split to the training set, and test set, 4500 instances and 1500 instances are in the training set and test set respectively. For the training set that is given, Each folder of 3 classes includes 1500 .JPG images and the size of each image is 300*300, which is preprocessed in terms of the size of the image. Therefore, the training set is well-balanced as the number of each class is the same.

The images are represented at an RGB model. Generally, RGB has 256 levels of Luminance, represented by numbers from 0,1,2.. Until 255. Although the highest number is 255,0 is also one of the values, so there are 256 levels. All colours can be created by mixing the red, green, and blue lights in different proportions. A set of red, green, and blue is the smallest display unit. Any colour on the screen can be recorded and represented by a set of RGB values. So an image in the training set is basically made up of 300 x 300 RGB units, each pixel having a set of values representing the corresponding colour Luminance.

In the training data, each folder contains some noise pictures. There are six kinds of noise images in figure 1. In the first cherry image, compared to a normal cherry, the small bubbles around the cherry could be seen as noise when training the CNN model. In RGB colour image, this kind of noise can be seen as Salt and Pepper noise which change the number of colour combination based on the noise radio[1]. Similarly, the cherry in the fifth image has a different colour as a general one, the number of RGB colour would significantly degrade the accuracy when classifying this kind of images.

The second image only contains a part of a strawberry, which may not be fully classified as the shape of the strawberry is hard to catch by the trained model. Also, the third image does not include a cherry at all, and the word ‘CHERRY’ is not to be trained as a cherry classification. The fourth and sixth image shows the occlusion of an image, which only a small portion of the target object could be visible and the target object is very small and invisible. In reality, the cherry and strawberry could be smaller than a tomato, which makes the process of classification harder.

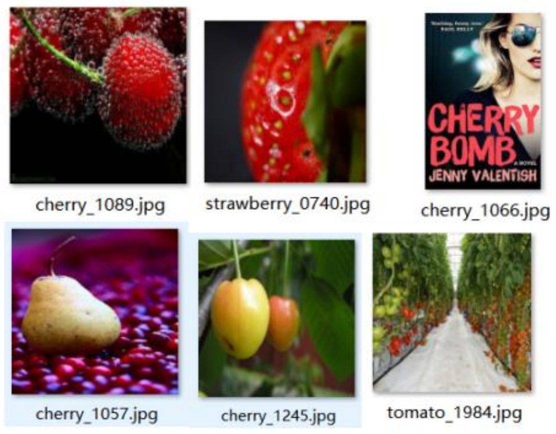


Figure 1: Examples of noise images

On the other hand, some images have a background with similar RGB colour or sharp should be considered as noise images as they provide much extra information in the image. When zoomed to 300x300, some of the images became distinctly slanted or faded. However, in real life, it would be unrealistic to expect pictures of these three fruits to contain no noise. The quality of image dataset is quite random, and it provides significant variation and randomness to the dataset, preprocessing and model-training process. However, reasonable noise should be included in the training dataset as it can introduce the capability of recognizing noisy images to the test set. But the training performance would be degraded if many noise images are involved.

It is noticeable that, in the training set, a few images is grayscale. The grayscaled image only has one dimension when the colour model is ‘grayscale’. It can also be used as the RGB model, but the ‘colour’ feature will not be the same as other RGB images.

2.2. *Data Preprocessing*

All images in the dataset were previously preprocessed to 'JPG' images with 300*300 image width. To further process the image dataset, some of the methods in the Keras ImageDataGenerator could be useful.
Some methods of **Data Augmentation** is the generation of training data from existing data sets. it is

the processing of an image that already exists in the Training Dataset and the processing of that image to create a version of many changes to the same image. This not only provides more image training but also helps expose my classifier to a wider range of both shapes and colours, making my classifier more robust.

featurewise_center/std_normalization: it is used to make the image have a mean pixel value of 0 and a standard deviation of 1, but I've tried it several times, and it's not very intuitive.

zca_whitening: The function of Zca Whitening is to reduce the dimension of Images by PCA, to reduce the redundant information and to preserve the most important features. When my image resizes to 256 * 256, the code reported a memory error that was too large during the calculation.

Rescale: The function of a resale is to multiply each pixel of the image by a scaling factor, which is performed before all other transformation operations. In some models, directly entering the pixel value of the original image may cause the activation function to fail. Therefore, setting the scaling factor to 1 / 255 and scaling the Pixel Value Between 0 and 1 is beneficial to the convergence of the model and ensures the normal operation of the neural network. After an image has been rescaled, the images stored locally are indistinguishable to the eye. If we print the image values directly in memory, they will all be floating-point values of 0 to 1.

On the other hand, some images have a high pixel range and some images have a low pixel range. Because these images share the same model, they have the same weight and learning speed. Without scaling, images with a high pixel range have a greater impact on the weight update and cause more loss, so these images will have a greater weight in training, but this is not the case. However, the loss of the low-pixel range image is small, and the influence of the weight update is small. The total loss is reduced by scaling the range of the image, regardless of the range of the original image.

width_shift_range/height_shift_range: These two parameters are the horizontal position shift and the Upper and lower position shift of the picture, respectively. The parameters can be floating-point numbers between [0,1] or greater than 1. It is set to 0.2 in my program. When images are moved, areas beyond the original image range are typically filled in according to the fill parameters. This is significant for the training data which has target objects in the corner or the edge of the image.

rotation_range: it is used to specify the image rotation angle range, its parameters can specify an integer, but is not fixed at this angle for rotation, but in the [0, desired angle] range for random angle rotation. In my program, I set it to 20 as the training data have already provided many different angles of target objects.

shear_range: The effect of Shear is to leave the x (or y) coordinates of all points unchanged, while the corresponding y (or x) coordinates are shifted proportionally, and the magnitude of the shift is proportional to the vertical distance from the point to the x (or Y) axis. After this operation, the shape of the image is usually changed.

zoom_range: This allows the image to be zoomed randomly within the range of a set number and improve the performance on the scaled objects in the images.

Considering the colour of three target objects: cherry, tomato and strawberry, They are similar in terms of colour when standard images are provided. To reduce the dimensionality of trained model, Grayscale in the function 'flow_from_directory' was attempted. The training speed of grey-scaled images was significantly improved as the dimension of the colour model was decreased from 3(RGB) to 1(Grayscale). However, it did not work properly as the shapes of three objects are similar too, The accuracy was degraded, and the loss was increased. So I decided to use the RGB model to introduce more features and parameters to the model and improve the performance

2.3. *Test/Validation set*

The original training data has 4500 instances. I add other 1368 images from google images with same preprocessing(modify the size to 300*300, change to ‘JPG’ images and use the same format on image names). There are 2 ways for using the dataset. Firstly, i only using the provided 4500 images as training data and using the other 1368 images as a test and validation dataset only, treat them as unseen instances and not used for training. Then, i add the two datasets together as an entire dataset, then split 20% (a validation factor of 0.2) as the validation set.

The final step of preprocessing is that i deleted the unreasonable images in the dataset(both given training set and added dataset). Those data only contain incorrect information such as the third image in figure 1, and it only has a ‘CHERRY’ in English, but not a real cherry.

3. *Methodology*

To properly train and construct the model to classify three-label dataset, A baseline network, a CNN and a transfer learning method ‘VGG16’ is used.

Machine image recognition is not to recognize a complex image completely at once, but to segment a complete image into many small parts and extract the features of each part, Then combine these small parts of the features together.

3.1. Baseline network

A one-layer CNN with a Convolution layer and 3 Dense layers is used as my baseline network. I tried to use Multi-layer perceptron(MLP), but the MLP did not work very well in my program, the accuracy can only converge about 33% with a huge loss.

3.2. CNN

The CNN that I constructed includes 5 convolution layers, 5 pooling layers and Dense layers. The BatchNormalization was used in some layers. Also, dropout was used to prevent the overfitting in the CNN.

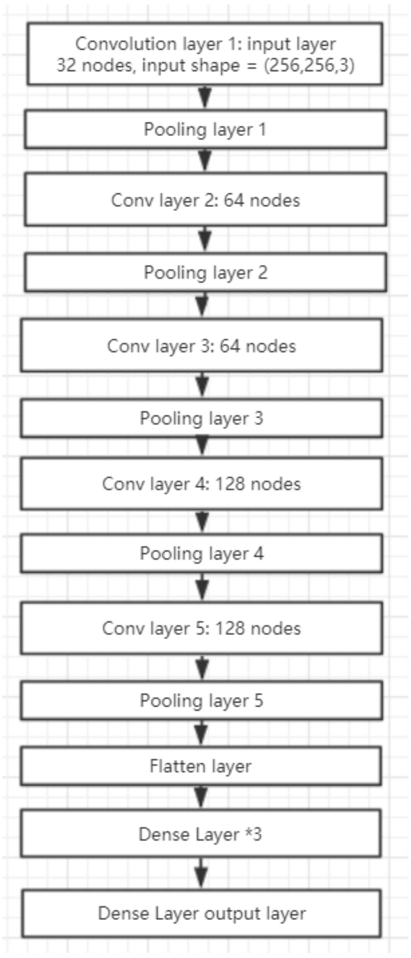


Figure 2: structure of CNN

3.2.1. Convolution layer

The purpose of the convolution layer is to extract the features of each small part of the image.[3] We define an image with a length and width of 300 x 300, and then we define a convolution kernel, which is weight, to extract certain features from the image. The values of convolution kernels can be randomly generated by functions without a previous learning experience, and then trained and adjusted step by step. When all the pixels are covered at least once, the convolution kernel and the corresponding phase of the digital matrix are multiplied and added to get the output of the convolution layer.

3.2.2. Pooling layer

The input of the pooling layer is the output matrix of the convolution layer after multiplying the original data with the corresponding convolution kernel. The purpose of the pooled layer is to reduce the number of training parameters, reduce the dimension of the feature vector output by the convolution layer, and at the same time, it can also reduce over-fitting, retain only the most useful image information and reduce the transmission of noise. Here all pooling layers are selected Max-pooling. While both max-pooling and average-pooling down-sample the data, max-pooling feels more like feature selection, selecting features with better classification recognition and providing non-linearity.[3] Average-pooling more emphasis on the transmission of the whole feature information, and the reducing of the parameter dimension so will retain more image background information.

3.2.3. Flatten

The Flatten layer is used to "Flatten" the input from multiple dimensions to one dimension. This is often used in the transition from the convolution layer to the fully connected layer(Dense layer).

3.2.4. Dense layer

The purpose of the convolution layer and the pooling layer is to extract features and reduce the parameters of the original image. However, in order to generate the final output, we need to apply the Dense layer to generate a classifier equal to the number of classes we need.

The Dense layer is like a neural network that accounts for about 80 percent of the parameters of the entire CNN network. The fully connected layer is mainly used for classification. Each layer of the Dense layer is made up of many neurons. A single neuron in one layer of the Dense layer can be regarded as a polynomial. Each value in neuron was integrated from the feature map, and if this value is large, the feature of our target object exists, regardless of its location, and Robustness can be greatly enhanced. But one Dense layer can sometimes not solve a non-linear problem, whereas a Dense layer of two or more layers can solve a non-linear problem by using more activation functions.

An increase in the number of Dense layers and the number of neurons leads to an increase in the complexity of the model. If I add a layer of neurons, the model becomes more complex. Increasing the length and width of the Dense layer could theoretically improve the learning ability of the model. However, over-complicated learning ability may lead to over-fitting of the model and significantly increase the training time, thus reducing learning efficiency.

3.2.5. BatchNormalization

In the training of the deep network, the input of each layer will change because of the change of the parameters of the previous layer, and sometimes the data will become very big or very small again. This requires a very small learning rate and good initialization of the parameters, but it makes the training process slow and complicated. Batch Normalization adjusts the distribution of data without considering the activation function, normalizing the output of each layer to a distribution with a mean of 0 and a variance of 1, which ensures the effectiveness of the gradient.

3.2.6. Dropout

The role of dropouts is to ignore a random number of neurons at each training session so that different combinations of neurons can identify a feature of the data. Each neuron depends on the characteristics of its neighbour's behaviour, and if it depends too much, it will cause overfitting. If The model dropout a random number of neurons at a time, then the remaining neurons need to replace the missing neurons, and the whole network becomes a collection of independent networks that can produce different solutions to the same problem to prevent it.

3.3. Method

3.3.1. Use of given images

As mentioned, the given training data was initially used as training data, and an added test dataset from google contains 1368 unseen images were used for validation set and test set. This is aim to see the quality of the original dataset and the ability of model of classifying unseen images.

Then in order to enrich the training set, i added the two datasets together and used 8:2 as the training set and validation set. Therefore, there are enough data for training, and a reasonable validation set can be used for testing in each epoch.

A validation set can be used to test the performance of the model in training, which is equivalent to testing each epoch. If the accuracy and loss values of validation are close to training, this model training performs well. If the accuracy of validation is significantly lower than the accuracy of training and the loss is much larger than that of training, this situation may be considered to be over-fitting.

3.3.2. The loss function

The loss function is used to estimate the degree of inconsistency between the predicted value and the true value of the model, and it is a non-negative real value function. The loss function maps a set of parameter values for the neural network to a real number, which indicates how well the parameters are tested for the network. The smaller the value of the loss function, the better the robustness of the model. The loss function is also a parameter to Keras.model.compile. The objective of the function can be regarded as an optimization objective, and the ultimate goal of the optimization model is to maximize or minimize the objective function.

There are many common loss functions in Keras, including mean squared error, mean MAE (absolute error), MAPE (mean absolute percentage error) , categorical crossentropy, and binary crossentropy. MSE, MAE, and MAPE are commonly used in regression problems. However, our problem is classification, so these three loss functions do not apply to our classification problem. Classification problems are commonly used on CNN as categorical crossentropy and binary crossentropy, where binary crossentropy is applicable in only two classes of output, but our classification has three different classes. So here I've chosen categorical crossentropy, which is more appropriate for a variety of classes.

When using the categorical crossentropy in the CNN network, the label values of the three categories used by the model to train the network should be vectorized 3-dimensional vectors, used as one-hot coding, where one vector has an index of 1 and the others zero, corresponding to one class. Such a class is vectorized to correspond to the 3 probability values trained by the neural network, and the vector corresponding to the output of the maximum probability value represents the label.

3.3.3. The optimisation method

The optimisation method is the other one parameter of the compile function in a model of Keras. The optimisation method could be used to minimise

the loss value. Three optimisation methods are explored, which is SGD, RMSprop and adam.
SGD usually refers to the mini-batch gradient descent or stochastic gradient descent. SGD is the process of calculating the mini-batch gradient each iteration and then updating the parameters. SGD is completely dependent on the current batch gradient, so the learning rate can be understood as how much of the current batch gradient is allowed to affect parameter updates. The disadvantage of SGD is the difficulty of selecting the appropriate learning rate, which uses the same learning rate for all parameter updates.[2] For data with infrequent features, we may want to update faster, for frequent features update slower, then Sgd can not meet the requirements.

RMSprop adaptively retains the learning rate for each parameter by averaging the most recent order of magnitude based on the weight gradient(the squared gradients). The Adaptive Moment Estimation(adam) is essentially an RMSprop with momentum term, which uses the first-order Moment Estimation and the second-order Moment Estimation of the gradient to dynamically adjust the learning rate of each parameter.[2] The advantage of Adam is that after bias correction, the learning rate of each iteration has a certain range, which makes the parameters more stable, and Adam needs less memory, so it can calculate different adaptive learning rate for different parameters.

The performance(accuracy and loss) of three methods is shown in the figure 3 and figure 4. The performance of adam and RMSprop is significantly better than the SGD. The performance of adam is slightly higher than RMSprop in the full training of CNN. Therefore i choose adam as the optimisation method.

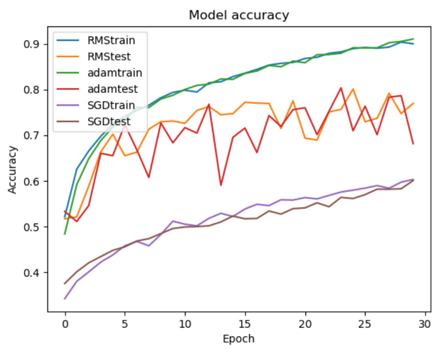


Figure 3: Accuracy plot of RMSprop,SGD&adam

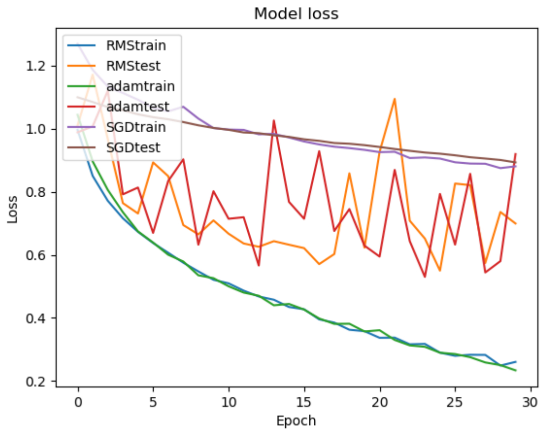


Figure 4:Loss plot of RMSprop,SGD&adam

3.3.4. the regularisation strategy

The effect of the regularization strategy is to reduce generalization error and therefore prevent overfitting in a training process of a neural network. During the training process, the model tries to learn the details and noises in the training data well. However, the final performance in the unknown test data is poor because as the model complexity increases, even if the training error decreases, the test error will not decrease.

There are many regularisation strategies could be used in CNN involve L1&L2 regularisation strategy, Dropout data augmentation and early stopping. L1 and L2 are the common methods used for regularisation, and it basically adds a regularisation term in the loss function. We can add the function in our Dense layer and convolution layer and import from Keras.regularizers. But when I used L1, I found that there was no significant change in the performance of my model except the loss was dramatically increased due to the function.

The most practical way to do this is the Dropout in my experiment. As mentioned in 3.2.6, during each iteration, some nodes are randomly selected, and their forward and backward connections are deleted. Therefore, each iteration has a different combination of nodes in each iteration, resulting in a different output. Accordingly, the Dropout makes the neural network model better than the normal model and changes the single model into the centralized model by making the trained model produce much randomness.[2] In this project, because my model is relatively complex, I dropped three times in the convolution layer and three times in the dense layer. The drop rate was set to0.2, 0.3 and 0.4 as it can have great performance in preventing overfitting.

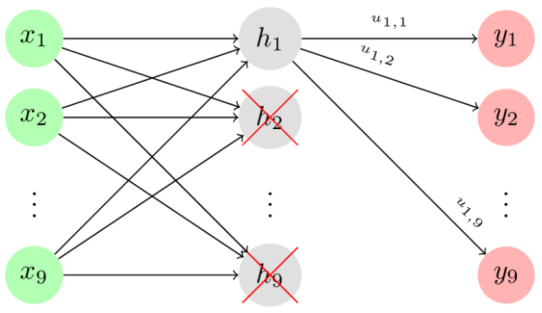


Figure 5: dropout function[4]

Furthermore, The easiest way to reduce overfitting is to increase the training sample. For image problems, there are several ways to increase training samples such as Rotaing, scaling, and shiftting. This technique is mentioned in 2.2 as data argumentation, which can greatly improve the robustness of models.

The earlystopping means having a portion of the training set as a validation set. When the performance of the validation set deteriorates, the training of the model is stopped immediately. I set the validation loss as the monitor as i want to obtain better performance on robustness instead of accuracy only. However, the response of the training process is not ideally stable with significant oscillation, the earlystopping will stop the train too early when the model is underfitting, so i did not use it in my final model.

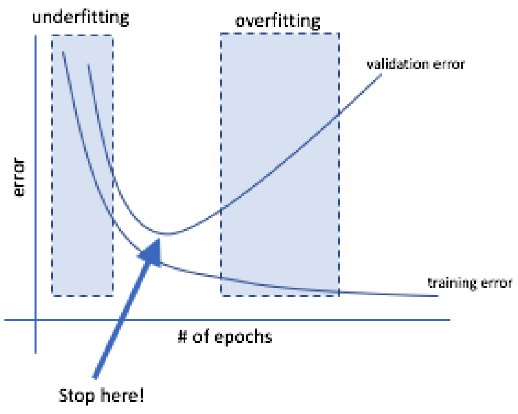


Figure 6：earlystopping function[5]

3.3.5. The activation function

Activation function is used to determine whether the information received by the neuron is relevant and introduce ability to solve non-linear problems. In CNN, the signal enters one neuron, passes through the nonlinear activation function, into the next layer of neurons, passes through the activation of that layer of neurons, and continues down, and so on, until it reaches the output layer. It is because of the repeated superposition of these nonlinear functions that the neural network has enough capacity to grasp complex patterns. The choice of activation function is whether it can be used to optimize the entire neural network. Two kinds of activation function were used in this paper, ReLU(Rectified Linear Unit) and softmax.

In each convolution layer, ReLU was applied due to the non-linearity of the image objects. ReLU is the most commonly used activation function in neural networks. It retains the feature of step function, which activates neurons only when the input exceeds the threshold. But when the input is positive, the derivative is not zero, allowing gradient-based learning.[5] Using this function makes calculations very fast, because neither the function nor its derivatives contain complex mathematical operations. However, when the input is negative, the learning speed of the Relu may become slow or even invalidate the neuron directly, because the input is less than zero and the gradient is zero, so its weight can not be updated and remain silent for the rest of the training.

$$\mathrm{ReLU}(x) = \max(0, x)$$

In the last layer of the Dense layer, the activation function uses softmax. The output vector of softmax is the probability that the sample belongs to each class. The input of softmax is the product of the input and the weight of the Dense layer.[7]

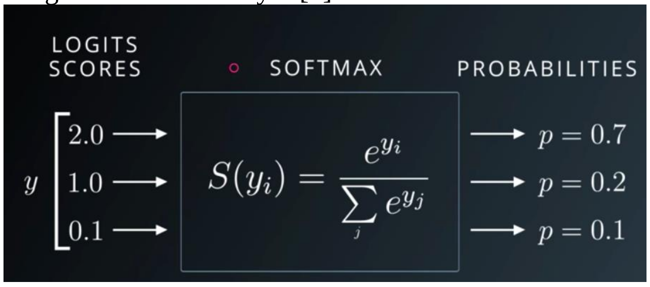


Figure 7: softmax function[7]

Suppose the input sample of the model is INPUT. For this 3 classification problem, class is represented by 1,2,3. Assuming the real class of sample INPUT is 1. So this sample INPUT gets a vector of 3 * 1 before it reaches the softmax layer through all the layers of the network, and yi in the above formula represents the i(index) value of this vector of 3 * 1. yj in the denominator represents three values in a vector of 3 * 1. So there is a summation sign where the summation is j from 1 to 3. Because the exponential is always greater than 0, the numerator is always positive, and the denominator is the sum of multiple positive numbers, so the denominator must also be positive, so S(yi) is positive, and the range is (0,1). The index of the largest number in the vector is taken as the prediction label for the sample.

3.3.6. hyper-parameter settings

The hyper-parameters in this paper involve the number of the epoch, Batch size, image length and width, learning rate, activation function, regularisation strategy, optimisation method, dropout, loss function and so on. Some of them were discussed previously. It is significant that trying different components of these hyper-parameters and tuning the parameters to get the best model and enhance the training efficiency.

3.3.6.1. Epoch

The number of epoch depends on the target performance of the model and the performance in terms of overfitting. A small number of epochs may cause the model to be undertrained, or underfitting. Such models do not achieve the ideal accuracy and loss values. Too many epochs can lead to model overfitting, and it takes a lot of time to train the model. Such a model might have validation's loss, starting to increase at some point in the middle and accuracy starting to decrease. The selection of the epoch number can be based on the performance of our model in training when the decrease of loss of the validation in training starts to slow down, or the increase in accuracy starts to slow down. So the number of epochs can be selected just after the model converge, In this project, the number of the epoch was chosen to 100 as it can greatly fit the model without significant overfitting.

3.3.6.2. Batch_size

Choosing the appropriate batch size is very significant in Mini-batches Learning. Increasing the value of the batch size will increase memory utilization and the efficiency of the large matrix multiplication. At the same time, the number of iterations needed to run one epoch will be reduced, and the processing speed will be further speeded up. The greater the Batch size is, the more accurate the gradient descent is, and the smaller the training oscillation is.

However, for large batch sizes, memory utilization is increased, but memory capacity may not be sufficient. The number of iterations required to complete an epoch is reduced, and the amount of time it takes to achieve the same precision is greatly increased so that corrections to the parameters are made more slowly.

Because of the preprocessing i did(the data generator and complex model structure), the memory does not allow me to use a large Batch size. I choose a Batch size of 32 for CNN, which can provide great performance in validation.

3.3.6.3. Image length and width

The length and width of the image determine the amount of CNN input, and the larger number of length and width will increase the number and precision of the convolution layers' feature map. If the target object in the image is too small, using a small number of length and width may result in the map feature not being extracted from the image. But increasing the length and width of the image significantly increased the CNN training time. Properly increasing the number of pixels in length and width can increase the model's accuracy and hence, the model's performance. In this case, the image length and width is 256 in the program, which can provide great accuracy and loss, but the training speed was slow.

3.3.6.4. Learning rate

The learning rate indicates how fast can the model learn, A high learning rate can introduce very fast training speed to the model, but high learning rate may lead to the problem that the model can not converge. The small learning rate can provide better performance in finding the optimal point, but the training speed could be significantly decreased. Except for using SGD, the learning rate could be adaptive during the process of training. I used the learning of 0.0001 for adam in the CNN model, and SGD has a learning rate of 0.01 to speed up the converge in the baseline training. These can provide the best performance in my model.

3.3.7. how many more images obtained

A large data set is useful for training models, and it can increase the robustness of the model. As mentioned, extra 1368 images from google images with the same preprocessing(modify the size to 300*300, change to ‘JPG’ images and use the same format on image names). There are about the same number of images for each class before some noise were deleted. The added dataset was merged with the original training set(1500 images for each label). The quality of added image dataset was bad as there many noise images, e.g. many ‘oriental cherry’ in the cherry data, and many ‘tomato sauce’ in the tomato data. After deleting many noise images, there are 1737 images for cherry, 1798 images for strawberry and 1924 images for tomato, and the proportion are 32%, 33% and 35% respectively, The balance of classes have not been significantly impacted.

3.3.8. The use of existing models

A transfer learning model: VGG16 pre-trained(by imagenet) model was explored. I expect the model can have better performance in terms of accuracy and loss than the CNN model i constructed, but the model has only 76.87% accuracy and 0.5682 loss in the training set and only 72.01%accuracy and 0.6687 loss in the validation set. Also, the training speed is very slow, it spends 1 hour and 53 minutes to finish the 60 epoch.

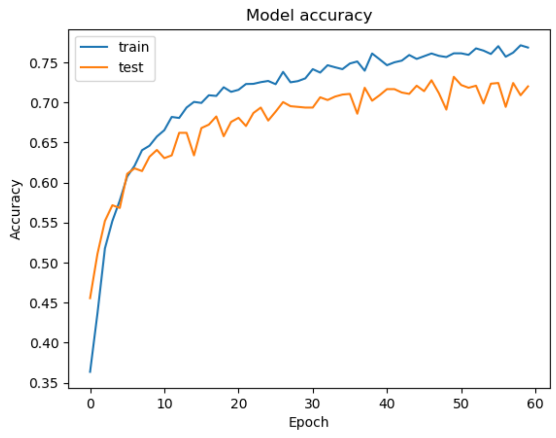


Figure 8: Accuracy of VGG16

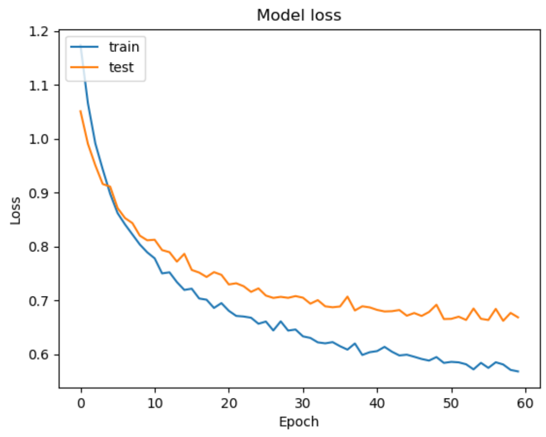


Figure 9 : loss of VGG16

4. Result discussions

4.1. Baseline result

Maxpooling layer and three Dense layers. The structure of my baseline model is in Appendix 1. An MLP should only contain dense layers with input at The Flatten layer, But such a model can not achieve good performance in my program, so i add a convolution layer to extract the feature information. With image length and width of 256, a batch of 32. To avoid overfitting, dropouts of 0.3 are added to every Dense layer. Activation function ‘ReLU’ is used at input convolutional layer and each Dense layer, while the ‘softmax’ is also used at the output layer. The Baseline model got an accuracy of 62.56% in training and 60.39% in the validation set, as well as the loss of 0.8343 at training and 0.8521 in the validation set. The figure 10 and figure 11 shows the plot of the training process with a 20% validation performance.

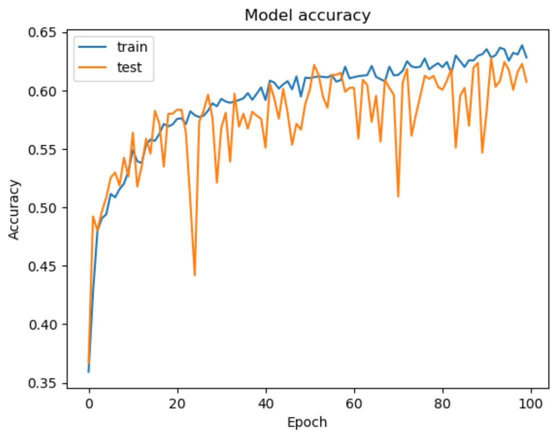


Figure 10: accuracy of Baseline

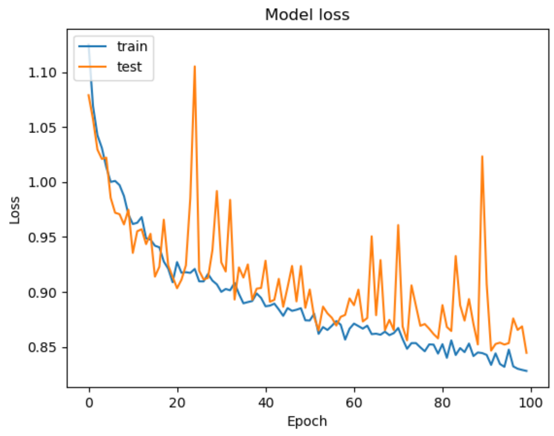


Figure 11 : loss of Baseline

4.2. MLP result

The MLP in my program is basically the Baseline without convolutional layer and the pooling layer. The performance was significantly degrade. The figure 12 and figure 13 shows the plot of accuracy and loss respectively. **The structure of Baseline, MLP and CNN are show in the appendix.**

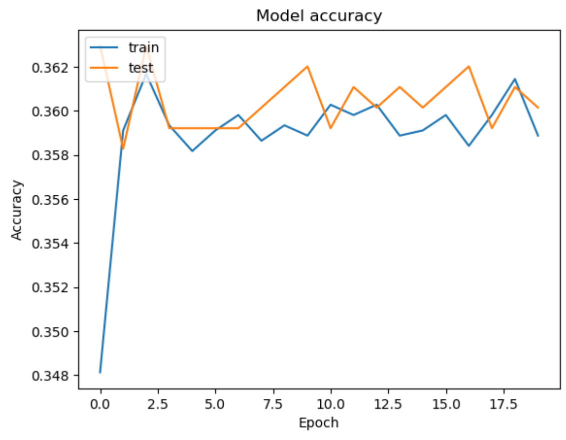


Figure 12 : accuracy of MLP

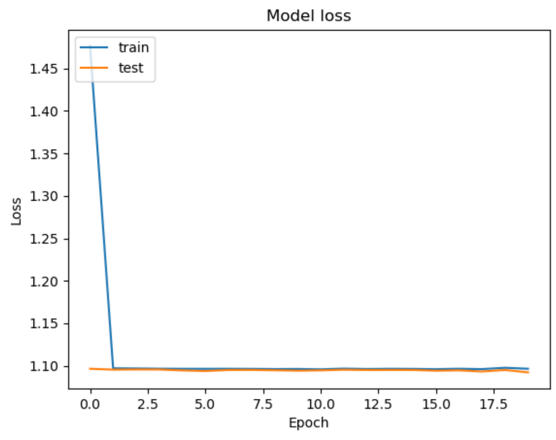


Figure 13 : loss of MLP

4.3. *CNN result*

The figure 14 and figure 15 shows the accuracy and the loss of CNN training process with 100 epoch, The model should be greatly converged, The oscillation after 60 epoch is enhanced than before, but the final model was greatly fitted.

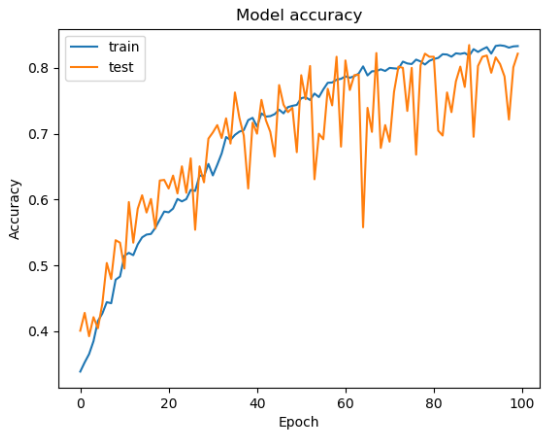


Figure 14 : accuracy of CNN

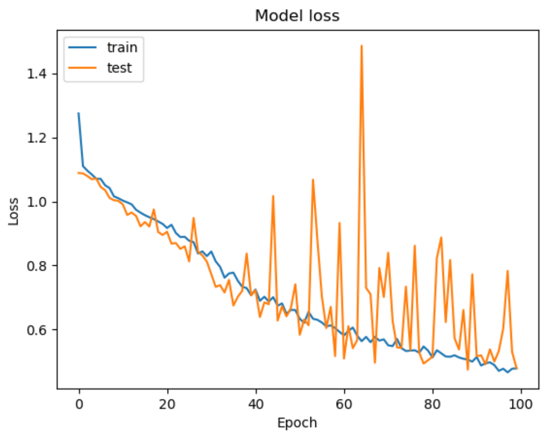


Figure 15 : loss of CNN

4.4. *Overall result*

	Baseline	MLP	CNN	VGG16
acc	62.56%	35.89%	83.30%	76.87%
loss	0.8343	1.0965	0.4784	0.5682
Val_acc	60.39%	36.01%	82.15%	72.01%
Val_loss	0.8521	1.0923	0.4790	0.6687
Test_acc	65.93%	33.37%	84.42%	73.43%
Test _loss	0.7710	1.0991	0.4292	0.6273
time	3:19:34	0:45:39	5:51:58	2:38:02
epoch	100	20	100	60

The table above is the overall result, including Baseline(1 Convolutional layer CNN), MLP, CNN, and VGG16. In terms of training time, the MLP should be the fastest one as it only has dense layers to train. Because of the higher complexity of CNN, The training time of CNN is much more than the Baseline. The VGG16 should be the slowest one because the VGG16 has lots of convolution layers and hidden layers then the CNN, due to the memory problem, The input of VGG16 here have not applied any preprocessing in the data generator(such as zoom, shift or rotation). The time will be doubled or more if more data images is generated.

Although MLP takes less time to train the model than CNN, its accuracy and loss are much worse. This is because CNN uses convolution and pooling layers, but the MLP model does not contain any of these layers, so CNN is better suited to the task of image classification than MLP. With one convolutional layer and one pooling layer, The performance was strongly enhanced.

MLP is difficult to achieve very high accuracy. In the case of fully connected, if the image is N*N Pixel, then the number of hidden layers is the same as the size of the input layer, if the RGB colour channel is not considered, The number of weights is going to be (N*N)^2. If the larger picture is taken into consideration, there is going to have an extremely large number of parameters, the computation is huge and the gradient vanishing problem would easily happen when the network is too deep, and the model is hard to be trained. Moreover, MLP can not recognize distorted images. The same image with rotated, or slightly shifted, is not able to be recognized by the MLP.

As for CNN, The convolution layer extracts the feature information from the input, and the multi-layer convolution can grasp the essential feature of the space. Moreover, the weights in the convolution layer are shared. CNN uses a convolution kernel to convolution the image, and each position of the image is calculated by the same convolution kernel, that is, the same weight is used for convolution and the value of weights in the convolution kernel does not change by the position in the image, so the number of CNN parameters can be significantly reduced. This leads to CNN being very effective in image recognition. CNN and MLP was designed for different tasks, CNN has more advantages on mapping image data(raw data), while MLP can achieve better performance on regression problems.

5. *Conclusion and Future work*

In conclusion, A CNN model with five hidden layers was able to construct. The model can achieve an 82.15% accuracy and 0.4790 loss on the validation set, while the baseline and MLP can only achieve 65.93% and 33.37% accuracy respectively. The model was developed based on the baseline and MLP, so it makes sense than the performance is much better than theirs.

5.1. Pros:

My model has great complexity in terms of convolutional layer and three dense layers(include output layer), so it can handle a highly non-linear problem. Enriched image data and varies preprocessing method significantly enhanced the robustness and reliability of the model.

5.2. Cons:

Because of the complexity and preprocessing, The model needs a very long training time and a huge memory capacity. The tuning of hyper-parameters could be further explored due to the low time efficiency. The structure of the model can also be improved because the oscillation of validation curve(unstable).

5.3. Future works:

1. A dataset with high quality images would be enriched to the training set.
2. To find a more robust model and achieve better performance.
3. To use the model into a real-world application by adding more components.
4. To do the exploration of a multi-class image classification in terms of activation functions, loss functions and better model structures.
5. More transfer learning method, pre-trained model in imageNet and ensemble learning could be used to improve the model performance.

6. *Reference*

[1] A.Jamil, Z.Bilal and A. Ziad, “Salt and Pepper Noise: Effects and Removal,” INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION, VOL 2,NO.4,pp 252-256, 2018

[2] P.Murugan and S.Durairaj, “Regularization and Optimization strategies in Deep Convolutional Neural Network”, arXiv preprint arXiv:1712.04711, 2017.

[3] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In International Joint Conference on Artificial Intelligence, pages 1237–1242, 2011.

[4] J.Reinhold, “Dropout on convolutional layers is weird”, towards data science, 10 February, 2019.[online]. Available: https://towardsdatascience.com/dropout-on-convolutional-layers-is-weird-5c6ab14f19b2.[Accessed Oct.28 2019]

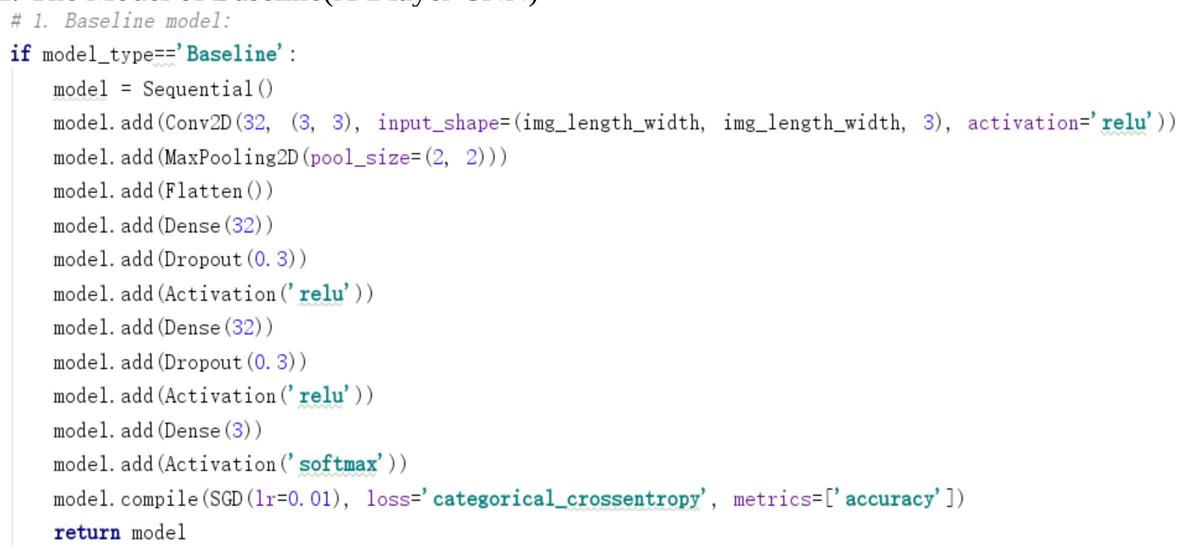
[5] J.Jordan, “Deep neural networks: preventing overfitting”, 25 July 2017.[online].Available: https://www.jeremyjordan.me/deep-neural-networks-preventing-overfitting/.[Accessed Oct.28 2019]

[6] U.Udofia, “Basic Overview of Convolutional Neural Network (CNN)”,13 February, 2018.[online]. Available: https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17.[Accessed Oct.28 2019]

[7] Uniqtech, “Understand the Softmax Function in Minutes”, towards data science, 31 January, 2018.[online].Available: https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d.[Accessed Oct.28 2019]

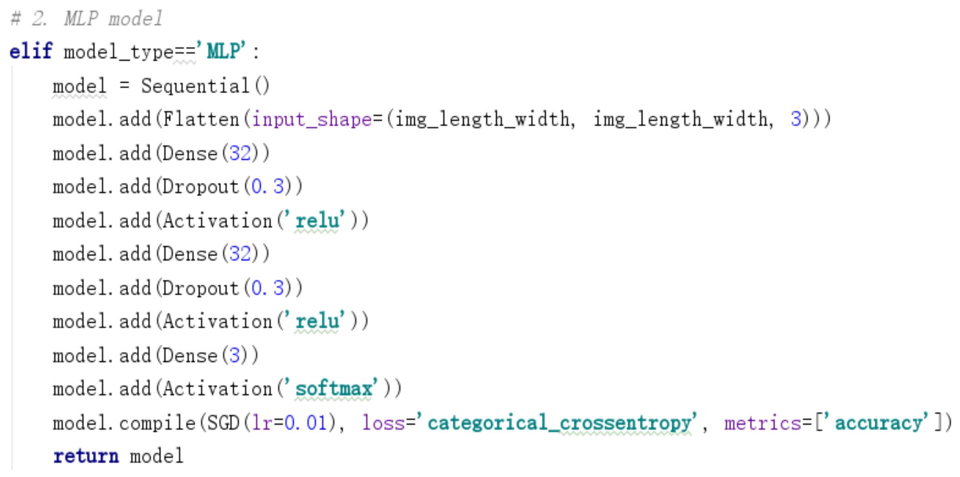
7. Appendix

1. The Model of Baseline(A 1 layer CNN)



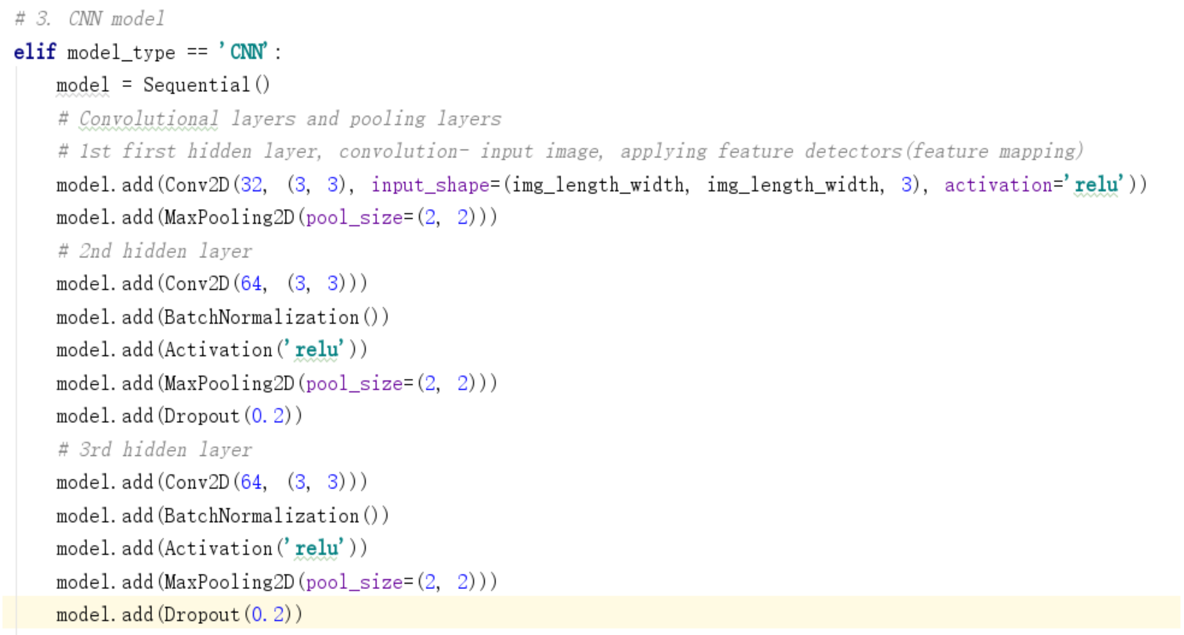
```
# 1. Baseline model:
if model_type=='Baseline':
    model = Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(img_length_width, img_length_width, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(32))
    model.add(Dropout(0.3))
    model.add(Activation('relu'))
    model.add(Dense(32))
    model.add(Dropout(0.3))
    model.add(Activation('relu'))
    model.add(Dense(3))
    model.add(Activation('softmax'))
    model.compile(SGD(lr=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

2. The Model of MLP

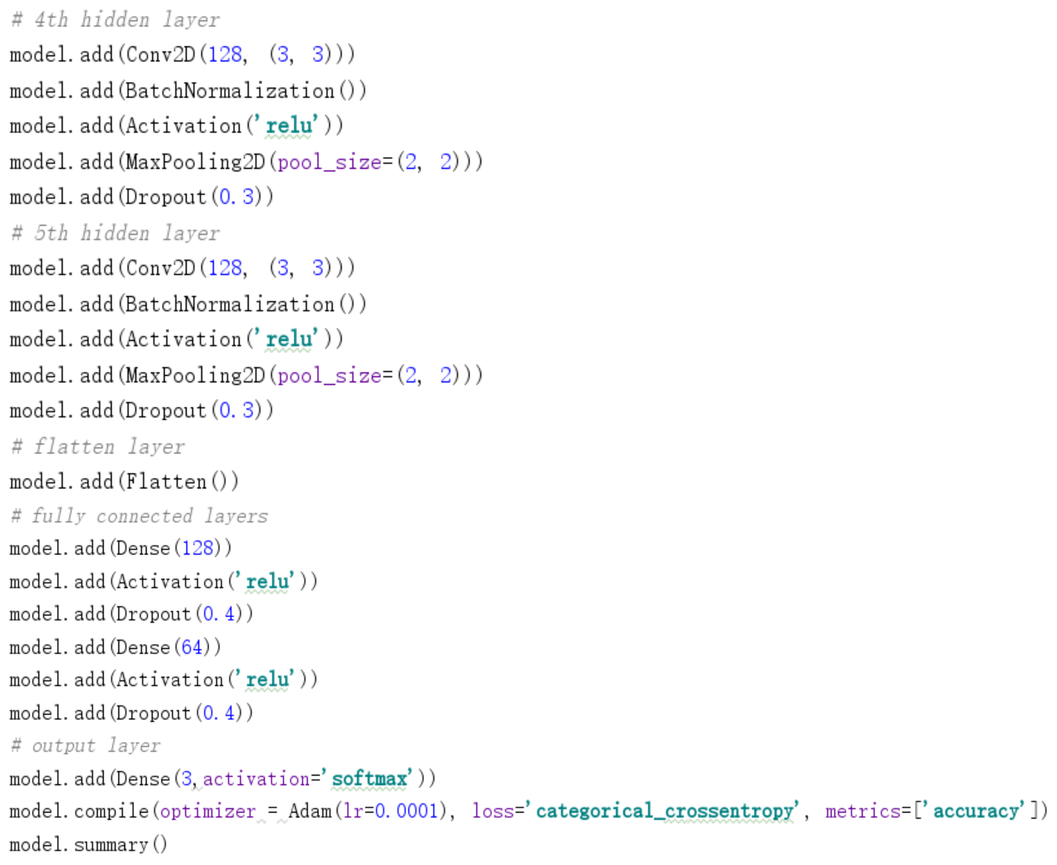


```
# 2. MLP model
elif model_type=='MLP':
    model = Sequential()
    model.add(Flatten(input_shape=(img_length_width, img_length_width, 3)))
    model.add(Dense(32))
    model.add(Dropout(0.3))
    model.add(Activation('relu'))
    model.add(Dense(32))
    model.add(Dropout(0.3))
    model.add(Activation('relu'))
    model.add(Dense(3))
    model.add(Activation('softmax'))
    model.compile(SGD(lr=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

3. The Model of CNN



```
# 3.  CNN model
elif model_type == 'CNN':
    model = Sequential()
    # Convolutional layers and pooling layers
    # 1st first hidden layer, convolution- input image, applying feature detectors(feature mapping)
    model.add(Conv2D(32, (3, 3), input_shape=(img_length_width, img_length_width, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # 2nd hidden layer
    model.add(Conv2D(64, (3, 3)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    # 3rd hidden layer
    model.add(Conv2D(64, (3, 3)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
```



```
# 4th hidden layer
model.add(Conv2D(128, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
# 5th hidden layer
model.add(Conv2D(128, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
# flatten layer
model.add(Flatten())
# fully connected layers
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.4))
# output layer
model.add(Dense(3,activation='softmax'))
model.compile(optimizer = Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```