Project

Image Classification

COMP 309 Machine Learning Tools and Techniques

Gaoxiang(Nelson) Deng

300442469

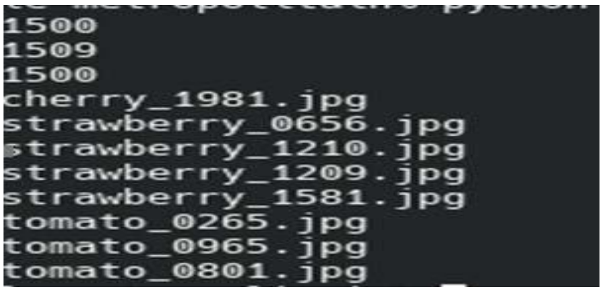
1 Introduction

The objective of this project is to construct on convolutional neural networks (CNNs) function in image classification. There are three different classes images: cherry, tomato and strawberry. The model after training should be able to classify an image into one of these three classes. This project was all done in python, and use the package Keras from tensorflow. There are 6,000 images in total, 4,500 images are given as the training data to train/learn a CNN model and rest of 1500 images are given from the google image download. The problem is that those images may include noisy images, noisy objects or outliers, and the images show different properties, is difficult to use computer for distinguish the object. As a result, my approach was to build a baseline MLP model first, and then using knowledge such as loss functions, optimization techniques, regularization strategy, activation to construct the CNN model. Finally, we can strive for achieve higher performance in my CNN model. However, the limitation of this report is the model only focus on these images and not have a big data to training the built CNN model. The goal of this project is to be familiar with Keras and CNNs model in image classification.

2 Problem Investigation

In recent years, Convolutional Neural Network (CNN) became popular for the image classification, and a lot of attention has been paid to deep layer CNN study. The success of CNN is attributed to its superior multi-scale high-level image representations as opposed to hand-engineering low-level features [1].

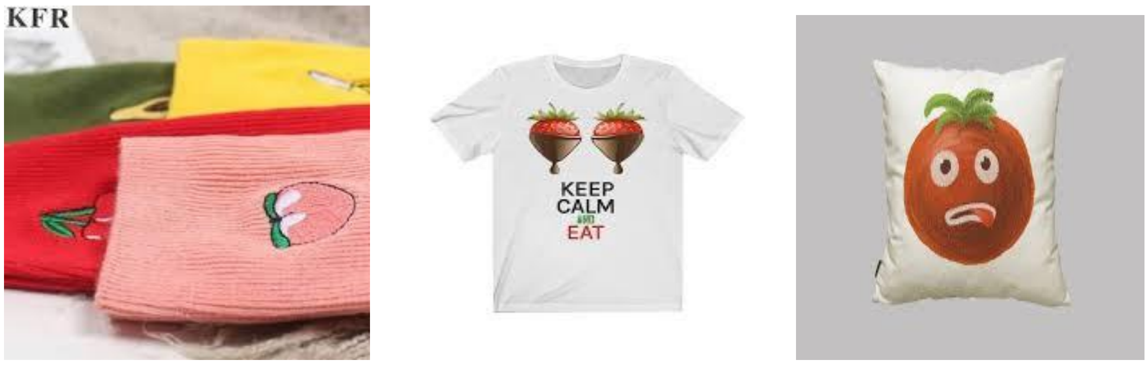
In this case, these images include three different classes: tomato, strawberry and cherry. Each class has nearly 1500 images for training the model. Therefore, we had a perfectly balanced dataset. But the data set is not perfect; for some parts, the images were labelled correctly, therefore containing the fruit of the class folder. There was an EDA code shown at Fig 2.1. It was included the number of each class file and several greyscale images.



```
1500
1509
1500
cherry_1981.jpg
strawberry_0656.jpg
strawberry_1210.jpg
strawberry_1209.jpg
strawberry_1581.jpg
tomato_0265.jpg
tomato_0965.jpg
tomato_0801.jpg
```

Fig 2.1 EDA result

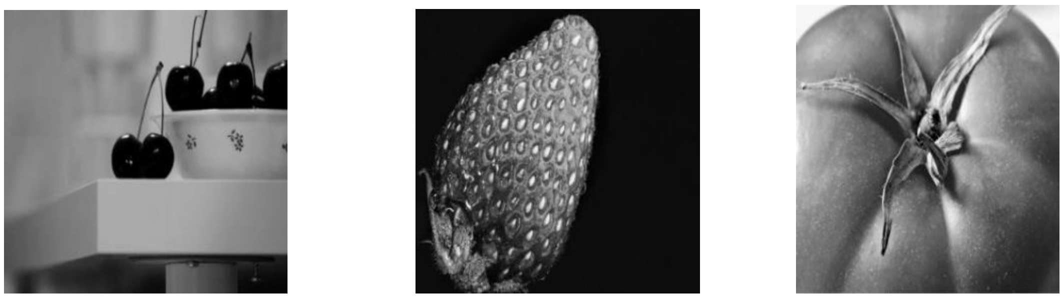
However, there were some images in the data set where the fruit was obviously an ingredient that was used in the meal that the picture was of, the image did not contain the fruit at all, or the image contained fruit from two of the different classes. Some examples are shown below Fig 2.2.



cherry_1357.jpg strawberry_0083.jpg tomato_1957.jpg

fig 2.2 some examples of nosiy images

For some cases, it is noted that there are several greyscale images and the background such as the following:



cherry_1981.jpg strawberry_1210.jpg tomato_0265.jpg

fig 2.3 some examples of greyscale images

From visualizing above images in each of the classes, we can easy to gain an understanding of the types of images in the dataset with respect to things such as their labelling, location of the fruit in the image, amounts of noise in the image, quality and colours of the fruit. But the project was not to do data augmentation, such as performing geometric transformations, changes to color, brightness, contrast, and adding some noise. As a result, the computer may be difficult to distinguish the object from these images. Some examples are shown below fig 2.4.

cherry_1971.jpg strawberry_0028.jpg tomato_0169.jpg

fig 2.4 some examples of greyscale images

After EDA I removed these noise images to improve the data through preprocessing. I found that some of the methods that I tried yielded no changes in the results, the below pictures were the results of the base line model, whereas some improved the performance of the different models significantly. There was shown at fig2.5 about the result of the baseline model.

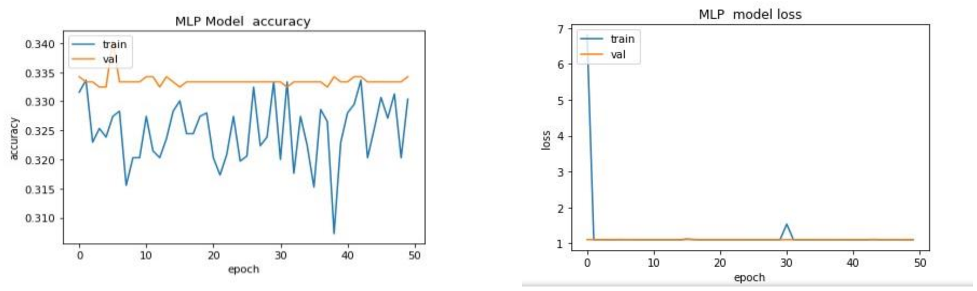
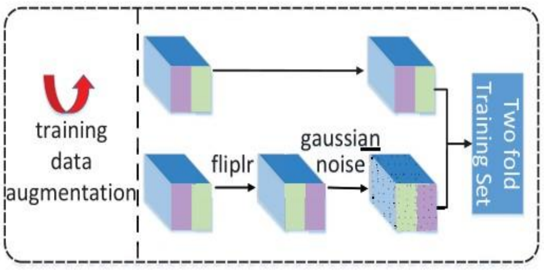
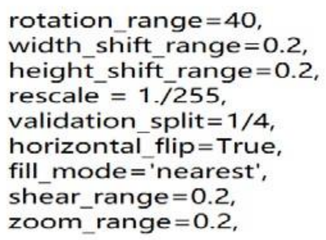


fig 2.5 the result of the baseline model

According to the lecture, the training of large images requires have the below points:

- Data preprocessing: resizing and per-channel normalization.
- Data augmentation can virtually increase the number of samples in the dataset using the given images: performing geometric transformations, like translation, rotation (rotation at finer angles), flipping, and changes to color, brightness, contrast, such as scaling, and or even adding some noise, for example, adding Salt and Pepper noise. The flow of data augmentation [2] and some information of the project were showed at the below picture fig 2.6.





```
rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
rescale = 1./255,
validation_split=1/4,
horizontal_flip=True,
fill_mode='nearest',
shear_range=0.2,
zoom_range=0.2,
```

fig 2.6 the information of training data augmentation

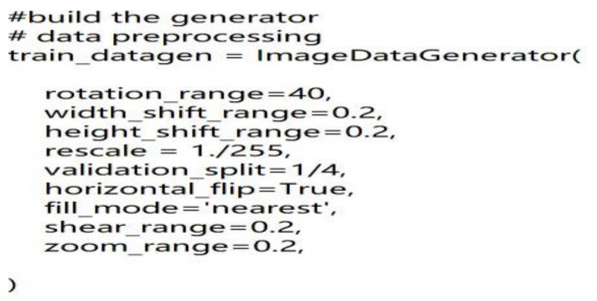
The below points were showed that using the most significant improvements to the performance of the models, which is Data augmentation

- **Shear_range**: This picks four points on the image and applies a transformation matrix on it, changing the angle between the points. Doing this changes the image slightly each time and creates new images for the model to train on.
- **Horizontal_flip:** This flips the image randomly horizontally which can help improve the models performance on a complex dataset.
- **Zoom_range:** This is the best form of the function random cropping from tensor flow that I was able to use to simulate random cropping, as it is not available in keras. Instead of cropping the image, zoom_range allows the model to zoom into an image a random amount, and analyse smaller amounts of the image.
- **Rescaling:** I found rescaling the images to 1./255 made the most significant difference to the accuracy, and in particular to the loss.

3 Methodology

After building my baseline MLP model, I moved onto building my CNN model.

- Firstly, doing data preprocessing and data augmentation. We used the ImageGenerator to preprocess the data to create a good model. It can generate batches of tensor image data with realtime data augmentation. The detail of information on ImageDataGenerator was shown on the fig 3.1.



```
#build the generator
# data preprocessing
train_datagen = ImageDataGenerator(

    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    rescale = 1./255,
    validation_split=1/4,
    horizontal_flip=True,
    fill_mode='nearest',
    shear_range=0.2,
    zoom_range=0.2,

)
```

fig 3.1 The detail of information on ImageDataGenerator

- Secondly, after building my baseline MLP model, I moved onto building my CNN model. The model that I made consisted of two convolutional layers. The aim of each convolutional layer is to create a feature map, otherwise known as a convolved map or an activation map. This reduces the size of the image and makes processing the image faster. For the first layer I set the number of features to be included in these maps to 64, and for the second layer, 16 features.
- The following details the methods that I used to try and optimise the efficiency, accuracy and loss of my CNN model. To find a best model for training data and we need find more pictures from the website for training the model.

3.1 How you use the given images

At the beginning, 4500 images were given as the training data provided image data, I did randomly split 80% of the data as training set and 20% of the data as validation set. That means training set contains 3600 images and validation set contains 900 images. The reasons for I split the data in this way is because I want the model have enough training before testing, otherwise will be course under-training problem. So, I designed to tune the proportion of training data and validation data to 3:1 as well. That means training set contains 3375 images and validation set contains 1125 images. Because the final proportion of training data and test data was 4500:1500, this was for make better simulate the final test environment.

3.2 The loss function(s)

The loss function is one of the parameters required to compile a model. A loss function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function, which means there become no error between the predicted output and the actual output.There are lots of loss functions in Keras library, such as MSE, MAE (which were used in the previous assignments) and categorical_crossentropy, binary_crossentropy. The most suitable loss function for my model was categorical_crossentropy. I think we can categorize the loss functions very broadly into two types, classification and regression loss functions.For this project, I tried both ways found out one-hot encoding and using categorical_crossentropy loss function works better; the accuracy was slightly higher.

3.3 Optimization method

The optimizer function helps to minimize the loss value, which is dependent on the model's trainable parameters, after each epoch. Different optimizer functions apply different methods and algorithms to evolve the optimizer and minimize the loss for the given training model. I tried three different optimizers, SGD, RMSprop and Adam.

SGD maintains a single learning rate for all weight updates, and does not change, whereas RMSprop which is an optimizer that is an extension of SGD, works by using momentum to drive convergence, yet restricts oscillations in a vertical direction to allow for larger horizontal steps to be taken. Therefore, the learning rates are adapted based on the gradients of the weights. Adam, however, combines advantages of Ada Grad and RMS prop and can be used instead of SGD. It iteratively updates network weights based on the training data and minimizes the loss value in fewer epochs, compared to the other methods. So, I found that RMS prop and Adam out performed SGD significantly and that Adam performs better the RMS Prop.

3.4 The regularization strategy

Regularization is used to avoid overfitting. The best model may not have the best performance on the accuracy, because an extremely overfitting model will have the highest accuracy while training. To have a "Good" model we need to care about the overfitting problem seriously. There are several regularization techniques in deep learning: **Dropout, L1 and L2 regularization, Early stopping.** To avoid overfitting, the first two techniques were used in this project.

- **Dropout** consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent.
- **L1 and L2 regularization.** Keras_regularizer and activity_regularizer were used to apply penalties on layer activity during optimization. In the two-regularization strategy, keras_regularizer is applied on the kernel weights matrix, and activity_regularizer is applied to the output of the layer. In detail of L1 and L2 regularization, L2 is the sum of the square of the weights, while L1 is just the sum of the weights. By contrast, L1 regularisation can help generate a sparse weight matrix, which can be used for feature selection. In predicting or

classification, so many features obviously difficult to choose, but if put those furthers into a sparse model, which only a few furthers have contribution to the model, but most of feathers have no contributions for the model, or have very few contribution (because they are in front of the coefficient is zero or is a very small value, even with is no effect on the model), then we can only focus on the feathers of the coefficient is not zero. For L2 loss function, the fitting process are generally inclined to let weight as small as possible, finally use all the small parameters to create a model. Because the model builds by small parameters, it can adapt to different data sets and avoids over fitting phenomenon to some extent. So, I apply L2 loss function in keras_regularizer and L1 loss function in activity_regularizer. different data sets and avoids over fitting phenomenon to some extent.

3.5 The activation function

The activation function is important in deciding whether the information a neuron is receiving is relevant. If the activation function is not included, the weights would perform a linear transformation and this has a limited capacity in solving complex problems such as image classification. Therefore, we need to introduce non-linearity in our ConvNet. The activation function does a non-linear transformation to the input, making it capable of performing more complex tasks. In my CNNs model, in each of the convolutional layers I used the activation method “relu”, and used "Softmax" activation function for the last dense layer. This is a generalisation of the sigmoid function, yet is used in multi-class classification whereas the sigmoid is used for binary classification. As there are three classes it is necessary to use the softmax function. "Softmax" activation function used in classification process, transform the output of the neutron, mapped to (0, 1) range, can be said as a probability, thus to classify more types.

3.6 Hyper-parameter settings

Hyper-parameters are parameters in the model such as the number of epochs, batch size, the learning rate of the optimiser, activation function and more. It is important to try and tune these parameters to optimise the training process and can help to reduce the time taken to train the model.

- **The number of epochs** being used was pertinent in optimising the relationship between the training set a validation set and this parameter required a lot of tuning. If the model was trained on too many epochs, I found that the loss began to increase, after initially decreasing, in the validation set, and the training accuracy would either plateau, therefore the model was no longer learning or the model would start to overfit, depending on the other parameters that were being used in the model. Too few epochs and the model would learn very little information. I found, after training the model on different numbers of epochs, that the optimum number of epochs for my model, where the gap between the validation and training accuracy was minimal and the loss values and accuracies were optimal, was around 50 epochs.
- **The batch size** divides the total sample size by itself to determine the number of samples that are propagated through the network in each epoch. It is typically faster to train the model on smaller batches as it requires less memory for the network to train with. Also, using batch’s means that the weights are updated after each propagation as well as the networks parameters.

Using a large batch size, and therefore a smaller sample size, can lead to less accurate results, therefore I tried multiple different batch sizes to try and optimise my results. I found that a batch size of 16 provided the best results when used with my model.

- **In regards to the optimiser,** I tried changing parameters such as the **learning rate, decay, betas and epsilon.** However, I found that, when using the **Adam optimiser**, the default settings where the optimum settings for my model.

3.7 How many more images obtained (should be at least 200 in order to get marks) and how you got them to enrich the training set

A large dataset is beneficial when training a model as it provides a greater selection of images and variation amongst the images for the model to learn on which allows the model to identify a greater number of features. It also helps in preventing the model from overfitting. To my split dataset, I added an extra 1500 images to the training set, over the three classes. I obtained these images from google using an extension to batch download and then I manually made sure that the images were of the correct fruits, contained little noise and were of a high quality. I rescaled all the images to be 300 x 300 to match the training data provided. I found that by increasing the size of my dataset, my accuracy on both training and validation. increased by nearly 5%. The following table indicates the difference in the results between the CNN model trained on the dataset without the added images, and the CNN model that's been trained on the dataset with the added images, the different were shown at fig 3.2.

Model	Accuracy	Accuracy Loss	Validation Accuracy	Validation Loss
CNN	0.7939	0.5352	0.7439	0.6419
CNN -more Images	**0.8489**	**0.4104**	**0.7902**	**0.5319**

fig 3.2 the different result from the two models

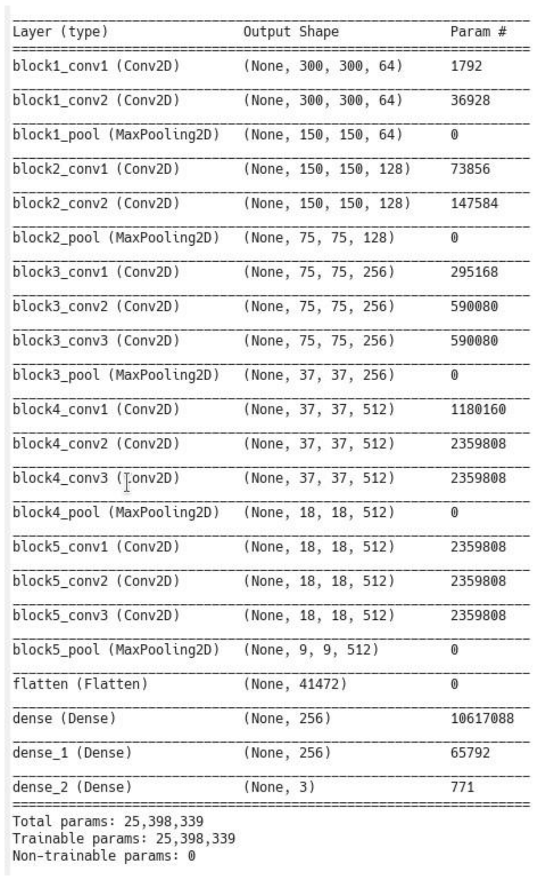
3.8 The use of existing models. For example, you may use transfer learning (e.g. models pre-trained by ImageNet)

My best model is basic build on the keras model VGG16, the figure on the left is the model after I transform VGG16. VGG16 model is a popular method with weights pre-trained on ImageNet. The processing of training shows as follow: Input a 300x300x3 image doing twice convolution using 64 filters (used only 3*3 size), then Max pooling layers (used only 2*2 size), then next block use 128 filter twice larger than last block, and so on. But the 5th block is the same as the 4th block, last fully connected with 256 nodes and output layer with **softmax** activation with 3 nodes.

Because considering the time factor, I am not design to use the more complex model than VGG16, and I also did decrease the dense layer nodes from the original VGG16 model. Despite this, the time for run single epoch of this model on **google colab (GPU)** use full data set (6000 images) will cost 6 min. Then to run 100 epoch that will cost about 8 hours to train for one model.

The result of the running test.py was below:

vgg16 model Test loss: 0.6955 Test accuracy: 0.9533 vgg16 model Time taken: 1401.50s



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 300, 300, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 300, 300, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 150, 150, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 150, 150, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 150, 150, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 75, 75, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 75, 75, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 75, 75, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 75, 75, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 37, 37, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 37, 37, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 37, 37, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 37, 37, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 18, 18, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 41472)             0
_________________________________________________________________
dense (Dense)                (None, 256)               10617088
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771
=================================================================
Total params: 25,398,339
Trainable params: 25,398,339
Non-trainable params: 0
```

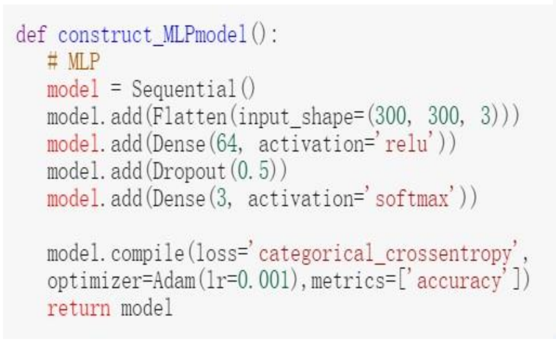
4 Result discussions: compare the results of your CNN with the baseline method MLP in terms of the training time and the classification performance. Analysis why. You should also describe the settings of your MLP here.

After finishing my CNN model, I was able to compare the efficiency of my baseline MLP model. The following table fig 4.1 was shown by the difference results:

Model	Accuracy	Accuracy Loss	Validation Accuracy	Validation Loss	Time Taken
MLP	0.3334	10.7438	0.3333	10.7454	47 mins
CNN	**0.8489**	**0.4104**	**0.7902**	**0.5319**	**46 mins**

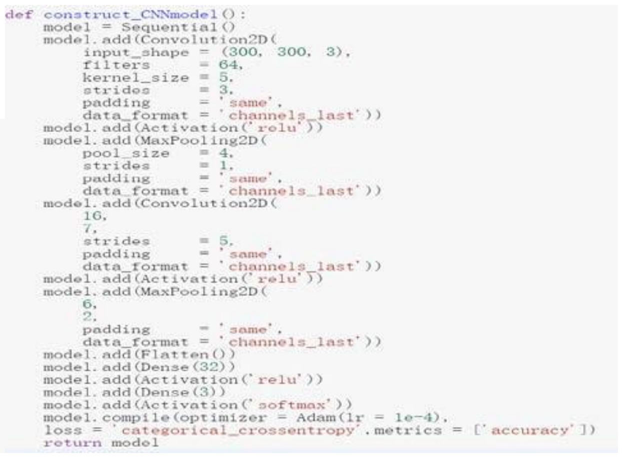
fig 4.1 the results from the two models

The time taken for the models to train is minimal, the difference between accuracy and loss is astronomically different. This is because CNN's are much better suited to the task of image classification than MLP's because CNN's use layers involving convolution and pooling. While the MLP model that I made does not contain any of these layers, which is the major difference. My MLP is a simple model that consisting of flatten layer, a dense layer using the **relu** activation function, a drop out layer and a final dense layer that utilizes the **softmax** activation function. The convolutional layers take advantage of the local spatial coherence of the input and they take the dimensional information of a picture in account, since the fruits appear in random portions within each of the images. Using this property, CNN’s can cut down on the number of parameters by sharing weights, however, MLP is deemed insufficient for modern advanced computer vision tasks. Because it has the characteristic of fully connected layers, where each perceptron relates to every other perceptron. Another disadvantage is that it disregards spatial information. It takes flattened vectors as inputs. While CNN’s were designed specifically to map image data to an output variable, whereas MLP’s learn to map generally from inputs to outputs. They were designed for use with problems more akin to regression problems and tabular datasets. The weights are smaller, and shared — less wasteful, easier to train than MLP. Layers are sparsely connected rather than fully connected. It takes matrices as well as vectors as inputs. The layers are sparsely connected or partially connected rather than fully connected. Every node does not connect to every other node. So, CNN is more effective than MLP. There was shown the code of two models at fig 4.2.



```
def construct_MLPmodel():
    # MLP
    model = Sequential()
    model.add(Flatten(input_shape=(300, 300, 3)))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(3, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
    optimizer=Adam(lr=0.001),metrics=['accuracy'])
    return model
```



```
def construct_CNNmodel():
    model = Sequential()
    model.add(Convolution2D(
        input_shape = (300, 300, 3),
        filters     = 64,
        kernel_size = 5,
        strides     = 3,
        padding     = 'same',
        data_format = 'channels_last'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(
        pool_size   = 4,
        strides     = 1,
        padding     = 'same',
        data_format = 'channels_last'))
    model.add(Convolution2D(
        16,
        7,
        strides     = 5,
        padding     = 'same',
        data_format = 'channels_last'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(
        6,
        2,
        padding     = 'same',
        data_format = 'channels_last'))
    model.add(Flatten())
    model.add(Dense(32))
    model.add(Activation('relu'))
    model.add(Dense(3))
    model.add(Activation('softmax'))
    model.compile(optimizer = Adam(lr = 1e-4),
    loss = 'categorical_crossentropy',metrics = ['accuracy'])
    return model
```

fig 4.2 the different codes from the two models

5 Conclusion and Future Work

My CNN model got 79.0% accuracy on the validation set and 84.9% accuracy on the test set. Comparing to the baseline model which only got the accuracy of 33.3%, the model was significantly improved. The model meets the requirement that to have a better performance than the baseline model. My model is much faster than VGG 16, but the disadvantage is the accuracy is slightly lower than VGG16.

Pros:

Since MLP is a simple model with only three layers, the training takes significantly less time than a multi-layer model. Testing with different models, as well as hyperparameter adjustments, optimized the model, added additional images, and finally made it perform better than the original application.

Cons:

Since my CNN is not designed to be complex, the depth of the neural network is quite shallow, and deeper CNN may produce better results. Besides, I don't have more time to train more models, such as VGG19, ResNet and DesnseNet, but only trained the VGG16 model, but its time cost is a big problem, Since the model still takes a long time to run. The efficiency of the parameter adjustment is not as good as expected, it was only improved 5%.

In Future work, there are several things that we can do better to deal with the image recognition problem. First, we would develop deeper into transfer learning, exploring more models that were trained on imageNet. Then we would perform a grid search on the data to try and configure the parameters to the optimal levels for the dataset. Besides, we would also perform a greater amount of preprocessing and try to create a more robust dataset with many more images of a high quality and look at using ensemble learning. Finally, we would need more perform machine to run the program.

REFERENCES

[1] Han, D., Liu, Q., & Fan, W. (2018). A new image classification method using CNN transfer learning and web data augmentation. *Expert Systems with Applications*, *95*, 43-56.

[2] Zhang, M., Li, W., & Du, Q. (2018). Diverse region-based CNN for hyperspectral image classification. *IEEE Transactions on Image Processing*, *27*(6), 2623-2634.