COMP 309 — Machine Learning Tools and Techniques
Project: Image Classification

Name(ID): Baiwei Fan (300376909)

•Introduction:

The main objective of this project is to produce a deep learning model that can correctly identify the content of the images without the help of any human being. Our given dataset contains 4500 RGB images which is divided into three different classes: cherry, tomato and strawberry, therefore our ideal model should have the ability to classify these data into the corresponding category. In order to achieve that, a MLP model and a CNNs model was constructed by using Keras written in python. The apporach of CNNs was an expansion based on the using of MLP. By adding convolutional layers to MLP and altering parameters within CNN model, my final model achieved with 77.05% accuracy on the test set. (separate 1648 RGB images downloaded from internet)

• Problem investigation

During the EDA process, which helped me to understand the data, the first thing I noticed is the perfect balance this data set has. The 4500 RGB images are evenly distributed into classes cherry, tomato and strawberry with same size of 300 x 300 pixels. This perfect balance may avoid the model to encounter with imbalanced problem and has positive influence on the accuracy.

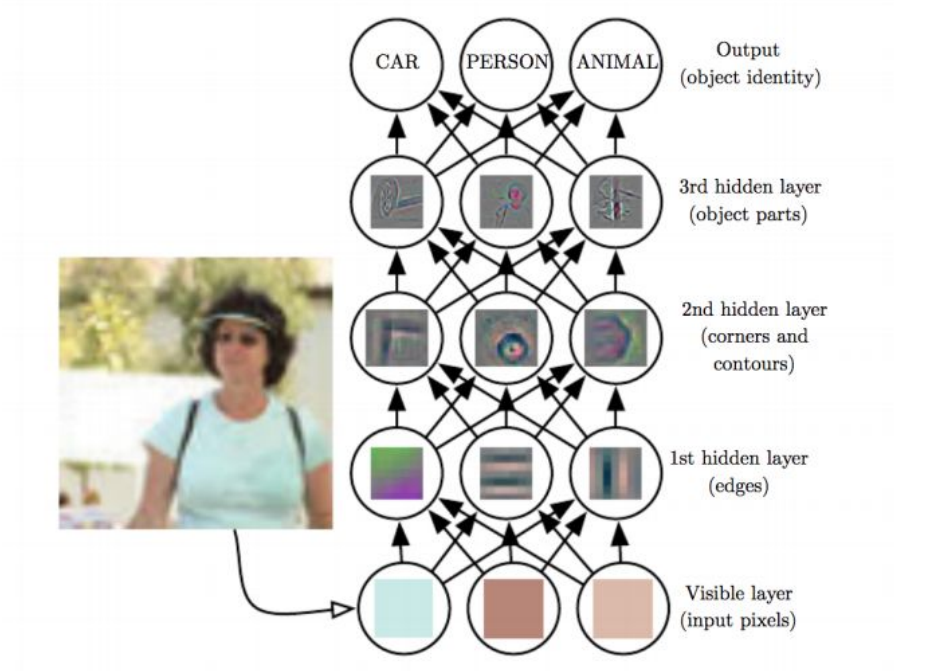
Next thing I looked at is the quality of this dataset. Most of the data can be interpreted clearly, but they are all slightly different. There are various properties of an image that could impact the result of classification. For example:

1. The objects of interest is occluded with only a small portion of an object is visible.

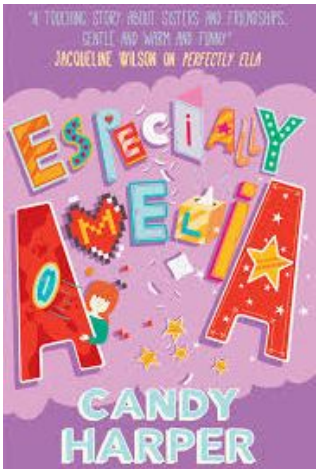
1. The objects of interest are not rigid bodies and deformed in extrem way

2. The single instance of the object is oriented in different ways respect to the camera.

Thanks to the feature learning techniques, these properties can be properly handled. Learning via hierarchy of feature extractors is the main approach. Each convolutional layers of hierarchy in CNN can extract features from output of previous layer, all the way from pixels to classifier.



The last thing I noticed is the existence of the noises data. For a data image to be considered as a noise, it will be the one that has no information or indication about the corresponding fruit.



These data will not contribute to the training process as they will only drag down the result, therefore I decided to remove the noise data in later processes(I started with the dataset containing noises, and the performance difference will be discussed later).

In terms of data preprocessing, I used ImageGenerator from keras to help me with this situation. The function of ImageGenerator is simply batches of tensor image data with real-time augmentation by looping them over and over. This method provided a wide range of techniques to augment the training data and also can implement a significant amount of variation in order to achieve a better model. Overall, I found that the following techniques helped me during the preprocessing process:

1. rescale(1./255). The rescale factor has a defult of none. This factor multiply the data by the value provided after applying all the other transformations. From the prospect of computer, when it processing RGB images, they are just number of pixels with value ranges from 0 to 255 while 255 stands for white and 0 means black. By using the rescale factor, I was able to manipulate pixel range. After a few attemps, I found that rescaling the image to

1./255 have the most improvement, because by unify the pixel range to [0,1] for all images, there will be no relatively large or small value. And most algorithm perform optimally when the value is small.

2. horizontal_flip(true). This beelean variable determines whether flips the inputs horizontally or not. It can improve the models performance.
3. shear_range(0.2): This factor shears the angle of pictures in counter-clockwise direction in degree and the shifted distance is proportionally to the vertical distance of all the coordinates (points) on the image.
4. zoom_range(0.2): This factor manipulate the image by randomly zoom it by the factor from 0.8 to 0.2 which could handle the picture with deformation.

• Methodology

After finishing building the baseline model MLP, I worked my way to the building CNNs. At first, I started my model with 3 layers of Convolutional layer. By adding layers during the configuration, I noticed that despite the fact that filters, activation function and strides have influence on the accuracy, the number of convolutional layer also can affect the accuracy, which raised from 75% to 85% when I added 3 additional layers. Therefore based on the result from testing , I have the following CNNs modle:

Layer	Activation Function	Setting
Convolutional Layer 1	relu	filters = 64, kernel_size = (3,3), strides = 3
Max Pooling Layer 1	-	pool_size = (2,2), strides = (2,2)
Convolutional Layer 2	relu	filters = 64, kernel_size = (3,3),

		strides = 3
Max Pooling Layer 2	-	pool_size = (2,2), strides = (2,2)
Convolutional Layer 3	relu	filters = 64, kernel_size = (3,3), strides = 3
Max Pooling Layer 3	-	pool_size = (2,2), strides = (2,2)
Convolutional Layer 4	relu	filters = 64, kernel_size = (3,3), strides = 3
Max Pooling Layer 4	-	pool_size = (2,2), strides = (2,2)
Convolutional Layer 5	relu	filters = 64, kernel_size = (3,3), strides = 3
Max Pooling Layer 5	-	pool_size = (2,2), strides = (2,2)
Convolutional Layer 6	relu	filters = 64, kernel_size = (3,3), strides = 3
Max Pooling Layer 6	-	pool_size = (2,2), strides = (2,2)
Flatten Layer	-	-
Dense Layer1	relu	unit = 64
Dense Layer2	sigmoid	unit = 1
Dense Layer3	softmax	unit = 3

-How you use the given images ?

From the 4500 given images, I split the dataset into a training set and validation test with proportion of 80:20 to prevent overfitting. Because if the mode is not tested is such way, the model will be only good enough to predict the value for data which was in training set.

On the other hand, in the very beginning, I tested my model based on two version of images, one with noises and one without. The result was not really surprising as the accuracy did not rise too much as I expected.

-Loss Function

The loss function is one of the parameters when compiling the model. It maps decisions to their associated costs, and in other words, it is just a method of evaluating how well my algorithm models the dataset. There are variety of loss functions to choose in Keras library, but the most suitable one for my model was “categorical_crossentropy”.

-optimisation method

The optimization method help us to minimize the loss value, which is dependent on the models trainable parameters, after each epoch. When I was testing the optimisaion method for my model, the one with best performance is ‘sgd’. I encountered with weird output that have consistent accuracy for all of my epoches when I was using ‘Adam’ for my optimizer, sadly I did not figure out the reason behind this.

-the activation function

Through the structure of my model, I have the activation function ‘relu’ for all of my convolutional Layer. And both activation function ‘sigmoid’ and ‘relu’ for my two dense layers. The purpose of activation functions is mainly to add non-linearity to the network, without it the whole neural network is equal to a linear regression. I did not notice that any obvious differences when testing the different activation functions

-hyper-parameter settings

There are three specific hyper-parameters I would like to discuss during the tuning process:

<u>batch size:</u>

During the process of load data, there is a parameter called batch size. The batch size is a hyperparameter that defines the number of samples

to work through before updating the internal model parameters.It is like a for-loop iterating over one or more samples and making predictions.
I noticed that the process of building my model will accelerate a lot when I decreased the batch_size.

Epoch and Epoch step:
The number of epoch is also a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. And the steps_per_epoch is the number of batch iterations before a training epoch is considered finished.
I tested several Epoch and Epoch step during my testing and result is shown below:

Epoch	steps_per_epoch	accuarcy
50	1500	0.80
50	5000	0.83
70	2000	0.83
60	3000	0.84

The optimal values for my model is 60 Epoch with 3000 steps_per_epoch. Although it seems that the accuracy will be higher if I set up the steps_per_epoch higher, the relationship is not strong enough to be considered as an important finding. I think the accuracy is still highly rely on how you construct the model, simply rise up the Epoch number will not help on the model performance.

• Result discussion

Model	Accuracy	Loss	Val_Acc	Val_loss

MLP	0.3449	1.0973	0.3247	1.0332
CNNs	0.9590	0.2056	0.8685	0.5771

-CNNs

```
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(IMAGE_SIZE, IMAGE_SIZE,3)))
model.add(Activation('relu'))   #Activation function
model.add(MaxPooling2D((2, 2), strides=(2, 2))) # max pooling layer

#---------------------------------Convolutional Layer
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Conv2D(128, (3, 3)))
#---------------------------------Flatten Layer

model.add(Flatten())
#---------------------------------Dense Layer
model.add(Dense(64))
model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.add(Dense(3))
model.add(Activation('softmax'))

#--------------------------------- complie the model
model.compile(loss='categorical_crossentropy',
          optimizer = 'sgd',
          metrics=['accuracy'])
return model
```

-MLP

```
def construct_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='sgd',
                  metrics=['accuracy'])
    return model
```

CNN final result:

```
2000/2000 [==============================] - 245s 123ms/step - loss: 0.2065 - acc: 0.9603 - val_loss: 0.7111 - val_acc: 0.8532
Epoch 68/70
2000/2000 [==============================] - 245s 122ms/step - loss: 0.1986 - acc: 0.9623 - val_loss: 0.6000 - val_acc: 0.8410
Epoch 69/70
2000/2000 [==============================] - 245s 123ms/step - loss: 0.1632 - acc: 0.9716 - val_loss: 0.6840 - val_acc: 0.8517
Epoch 70/70
2000/2000 [==============================] - 245s 123ms/step - loss: 0.2056 - acc: 0.9590 - val_loss: 0.5771 - val_acc: 0.8685
Time taken: 17371.15159
Model Saved Successfully.
```

My CNN model got 95% on the validation set, but only 77% on the test set. Compare to the baseline model MLP which only has a classification accuarcy of 34%, the model is significantly improved as the CNN is more suitable to the image classification task.

• Conclusions and future work

Overall, there are still a lot of parts where I can make improve on my CNN model. The outcome model is rather acceptable than good. My model is relatively simple and exists the potential of having a overfitting as the classification accuracy is heavily decreased on actual test dataset and I wasn’t able to figure out why. Maybe I need to add more techniques to prevent overfitting.

I should definitely practice more on tuning all kind of the parameters, as there are still a lot methods I haven't tried on my model. I also found the main obstructive when I was proceeding

the project was too time consuming, I may spend more time in the future to optimize my model .