COMP 309

Project Report

Image Classification

Author: Yongbo Yu

ID : 300390526

1 Introduction

The objective of this project is to construct a model which could classify images. 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. The goal of this project is to be familiar with Keras and construct convolutional neural networks.

2 Problem Investigation

The training set includes three different classes, tomato, strawberry and cherry. Each class has 1500 images for training the model. Therefore we had a perfectly balanced dataset. But the data set is not perfect; there was a lot of images are not appropriate to represent these fruit. Some examples are shown below.

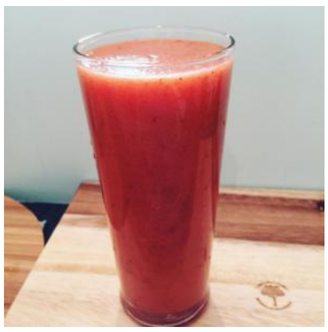


fig 2.1

From the graph is shown above, it was found in strawberry class, but even people can not tell the juicy is made from which fruits, so this kind of data should be deleted. This kind of data should be considered as noises.
Viewpoint variation was one of the problems, while I was exploring the data I found most data are viewing from the front or the back of different fruits which can give a good demonstration of shapes. But there are a few pictures viewing from the top or the bottom of the fruits which is hard to tell the difference between these three classes.
The CNN model classified images partially based on the shape and size. But there are scale problems in the dataset. Cherry is smaller than tomato in the real world but in our dataset, some of them are the same size, which makes the classification process even harder.
Some pictures in the dataset have a deformation problem, as the graph is shown below. It is easy for a human to recognise this is a tomato, but as there only very a few pictures have this shape which was labelled as tomato, it is hard for machines to learn the fruit with a shape of this can be a tomato.

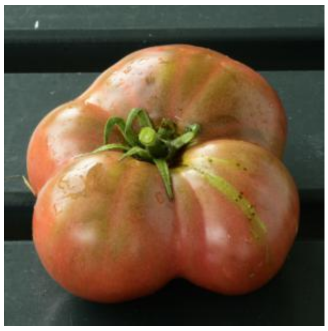


fig 2.2

The other big problem was occlusion, which means sometimes the Sometimes only a small portion of an object (as little as few pixels) could be visible. Here is an example I found in the dataset.

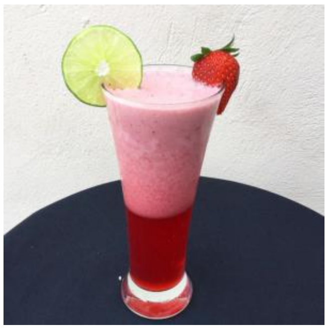


fig 2.3
The left bottom part of the strawberry is invisible

3 Data Pre-processing

3.1 Re-size

I enrich the data set by downloading more images online, but the images need to be resized to have the same shape with the given images. Otherwise, it could report a dimension problem. Also, I reduce the size of the images. The size of the given images was 300 × 300; I scaled the images to 64 × 64 which could save lots computations. This could highly improve the time performance for building the model.

3.2 Normalization

The values was from 0 to 255 for each pixel of the image, then I scaled the images that their values are between 0 to 1. The code for doing this is shown below :

```
X_train = np.array(images, dtype = "float32") / 255.0
```

The images we are working with are 8-bit. This means that each pixel in the image is one of 256 ($2^8 = 256$) possible values. While some machine learning algorithms can handle having relatively large pixel values, most methods perform optimally (train within our lifetimes) when small, floating point values are processed. Hence I divide by 255.

3.3 Image Generator

As the problems I mentioned in the previous sections, we need to pre-process the data to create a good model. I used ImageGenerator from keras to handle these problems. It can generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches).
The image generator function allows us to set a range of brightness, which can fix the pixels too dark or too bright. Also, I set the rotation angle range to 50 which allows the image randomly rotate clockwise or anticlockwise and the angle of the rotation was set from 0 to 50 degrees. And I set width_shift_range to 0.1; this could allow the image shift to the left or right, the length of shifting was decided by the image size and the ratio, in my project it was 6pixels. Similarly, I set the height_shift_range to 0.1 this could allow the image to shift vertically. These processes can move the image. It is important because maybe the important curve of the figure which can help the model to classify is in the corner or on the edges of the graph. These curves maybe are very important, but it is hard to detect. Hence we need to shift the graph to let these important details can be easily detected.
I also set the zoom range to 0.2, which could randomly zoom the graph by the factor from 0.8 to 1.2, this could help to handle the scale problem as mentioned before.
Fill_mode was set to "nearest" which could fill the outside boundary pixels with the values of nearest pixels.

3.4 Convert the image to gray-scaled

All the three different fruits can have the same colour (green/red/yellow) so I think using grey-scaled data can make the model more efficient because it can reduce the channel from 3 to 1 which can save a lot of computations. But it turned out the accuracy was lower than the model which was 3-channels.
But it did not improve the performance in accuracy, so I think maybe for our dataset it is better to use RGB instead of using grey scaled images. Using grey-scaled images may lose some information.

4 Methodology

4.1 How you use the given images

There are 4,500 images given. I split the dataset into a training set and validation tests. To split the data set into a train set and a test set, I used train_test_split which was imported from sklearn. (sklearn.model_selection.train_*test*_split). It can split arrays or matrices into a random train and test subsets. Doing this we do not need to use a for loop to shuffle the data set. The split was set to 0.1 for my final model, which set 90% of the data as training data and the rest of 10%were used as validation sets.

4.2 Loss Function

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, we should use the loss functions from classification type loss function. I chose to use *categorical_crossentropy* function because the class label was encoded into binary labels(one-hot encoded [1,0,0] [0,1,0] [0,0,1]). (Using *fit_transform* from sklearn.preprocessing.LabelBinarizer). *Categorical_crossentropy* works better for one-hot encoded class label dataset if the class label was converted into different integers like 1,2,3 for our dataset. I tried both ways found out one-hot encoding and using *categorical_crossentropy* loss function works better; the accuracy was slightly higher.

4.3 Optimization method

Optimizer is the other one argument required for compiling the model. We can instantiate an optimizer before passing it to model.compile(), or we can call it by its name. But in the latter case, the optimizer will operate functionality of default settings which maybe is not the optimal choice. Optimization algorithms help us to minimize an Error function E(x) which is simply a mathematical function dependent on the Model's internal learnable parameters which are used in computing the target values(Y) from the set of predictors(X) used in the model. For example, we call the Weights(W) and the Bias(b) values of the neural network as its internal learnable parameters which are used in computing the output values and are learned and updated in the direction of optimal solution i.e minimizing the Loss by the network's training process and also play a major role in the training process of the Neural Network Model .
The first one was used was SGD which was introduced in the lectures and also got involved in the previous assignment. This optimizer was used to built my baseline model.
The next optimizer I tried was RMSprop. It uses a moving average of squared gradients to normalize the gradient itself. That has an effect of balancing the step size,(Note: so the step size is another thing we need to take care.) Hence, decrease the step for the large gradient to avoid exploding, and increase the step for a small gradient to avoid vanishing. The result of using this optimizer gave me an accuracy of about 59%. This optimizer was a very clever way to deal with this problem, but maybe I did not find the optimal parameters.(Step Size)
The final version of my model was constructed by using Adam optimizer.
Adam stands for Adaptive Moment Estimation. Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like AdaDelta, Adam also keeps an exponentially decaying average of past gradients. So in practice like this project, Adam function usually works well.
Using this optimizer I got the highest performance, the test accuracy was 83%.

4.4 The regularization strategy

Regularization is used to avoid overfitting. The best model may not have the best performance on the accuracy, because a 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:
1.Dropout
2. Early stopping
3. L1 and L2 regularization
To avoid overfitting, the first two techniques was 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

overfitting. The key idea of this, is to randomly drop nodes from the neural network during training. The diagram is shown below demonstrate how this process actually works like :

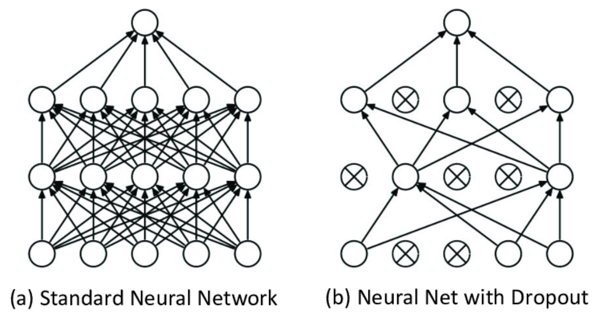


fig 4.4.1

In this project dropout was used three times, the drop rate was set to 0.2 and 0.5. The latter value was set for the dropout function before the output core layer.
Early stopping can also avoid overfitting and make the training process more efficient. It can stop training when a monitored quantity has stopped improving. Here I set the validation test accuracy as the monitored quantity.
By doing this, the result shows the overfitting problem was reduced.(Nearly removed)

4.5 The activation function

As we discussed in the lecture, activation function needed to be added after each convolution operation. And for this project, we need to introduce non-linearity in our ConvNet. Hence we need a non-linearity activation function. I tried both "Relu" and "sigmoid"(for hidden layers). The result turns out relu is more suitable for my model, which follows the information given in the lecture that relu has been found to perform better in most situations.
I used "Softmax" activation function for the last dense layer.
The softmax function squashes the outputs of each unit to be between 0 and 1.

$$\begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \longrightarrow \boxed{\text{Softmax}} \longrightarrow \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix}$$

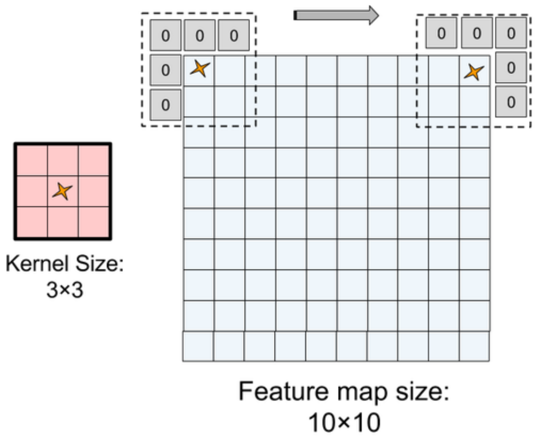
The output of the softmax function is equivalent to a categorical probability distribution, it can tell the probability that any of the classes are true.

4.6 Hyper-parameter settings

Hyper-parameters are the variables which determine the network structure and the variables which determine how the networks are trained.

4.6.1 Convolutional Layer

For Convolutional layer, I used Conv2D which was imported directly from Keras. There are lots parameters we can set. But there was one thing important in the first Conv2d layer that is to set the input shape, which is the size of the input images. Here I used (64,64,3) the last number 3 was the number of channels (RGB).
1.filters: Integer, the dimensionality of the output space
The first layer which is the input layer, the number of filters was set to 32, and the number of filters in the following hidden layers was doubled every time.
Usually, the initial layers represent generic features, while the deeper layers represent more detailed features (of the specific dataset that is used for training the model).
kernel size: specifying the length of the convolution window
The kernel size was set to 3 which is the convolution window size of 3. It convoluted with the matrices, and the reformed matrices can represent the important information of the images, such as the shape, the curve of the images, the colour etc.
In this project, the kernel size of all convolutional layers was set to 3. I tried different settings of this parameters; I found 3 was the most optimal choice.
The setting for padding was 'same '; the graph is shown below shows how the same padding works.



4.6.2 Pooling layers

Pooling Layers: spatial Pooling (down-sampling) reduces the dimensionality of each feature map but retains the most important information.
I used MaxPooling2D() layer as my pooling layer function. To reduce the dimensionality and retains the information, I set the *pool_size* to 2 which is the smallest value(default value) for it to make sure it will not lose too much information, and I tried to increase the pooling size but accuracy reduced. Therefore I kept pooling size value unchanged. And the stride was set to 2, which follow the pooling size. (Without specifying the value for stride Keras will automatically use the same size with the pooling size.)

4.6.3 Learning rate

Learning rate is an important parameter that controls how "fast" the model is learning. Low learning rate will slow the training process. But it can converge smoothly. Though high learning can speed up the training process, the model may not converge. Because I set early stop which will stop the training process once the accuracy can not increase any more, so I set the learning rate to a small value (0.001). The model converges very smoothly which was expected.

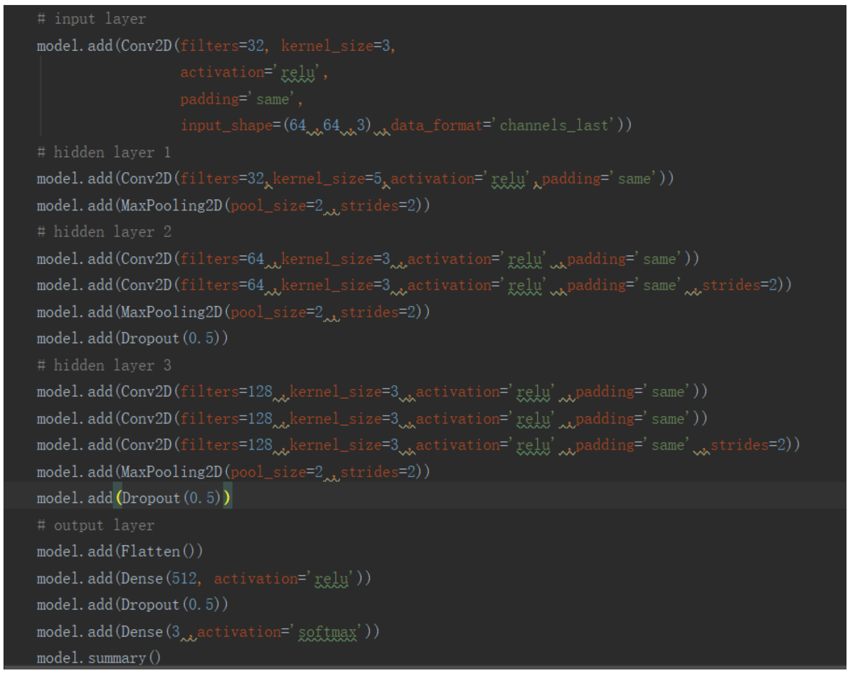
4.6.4 Epoch

The value of Epoch is the number of how many times the model will fit with the whole training data. Before setting early stop for my project, I tried epoch = 100, which was very slow. And the model was overfitting, which model has a training accuracy much higher than the validation test accuracy. I found 48 was the best epoch value for my model. But as I mentioned I added early stopping to my code, so I can set this value to 100 or maybe higher. To make sure the model reach the best performance it can be.

4.6.5 Batch Size

The batch size defines the number of samples that will be propagated through the network.
The number of batch size depends on the size of our dataset, for our dataset I set the batch size to 32 which is also the most frequently batch size for DCNN.

4.6.6 Construction for the final model



```
# input layer
model.add(Conv2D(filters=32, kernel_size=3,
                 activation='relu',
                 padding='same',
                 input_shape=(64,64,3),data_format='channels_last'))
# hidden layer 1
model.add(Conv2D(filters=32,kernel_size=5,activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=2,strides=2))
# hidden layer 2
model.add(Conv2D(filters=64,kernel_size=3,activation='relu',padding='same'))
model.add(Conv2D(filters=64,kernel_size=3,activation='relu',padding='same',strides=2))
model.add(MaxPooling2D(pool_size=2,strides=2))
model.add(Dropout(0.5))
# hidden layer 3
model.add(Conv2D(filters=128,kernel_size=3,activation='relu',padding='same'))
model.add(Conv2D(filters=128,kernel_size=3,activation='relu',padding='same'))
model.add(Conv2D(filters=128,kernel_size=3,activation='relu',padding='same',strides=2))
model.add(MaxPooling2D(pool_size=2,strides=2))
model.add(Dropout(0.5))
# output layer
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3,activation='softmax'))
model.summary()
```

4.7 Get More Data

More data can help to construct a better performance and a more robust system. As we discussed in the lecture if we can have a big infinite dataset we maybe can build a perfect model has the accuracy of 100%. So I download 200 images for each class. I download the images from Flickr. And reshape the size of the images to 300$\times$ 300 to match the size of the given data.

4.8 VGG 16 and VGG 19

There is lots pre-trained model provided by Keras.applications. I tried VGG16 and VGG19. These two models can reach a higher accuracy over 90%. I think the reason for it can reach a higher accuracy not only it has a more complex CNN construction, but more importantly, the pre-trained models have a defined weight matrix.
But the size of these model is much bigger than the one I created. And the time to train and run these model is way longer than mine one.
So here is a trade-off between efficiency and accuracy. Using transfer learning, the accuracy was improved by 10%, but the time for training was more than three times than train my model. I would like to use my model, which has a smaller size, and much shorter training time.

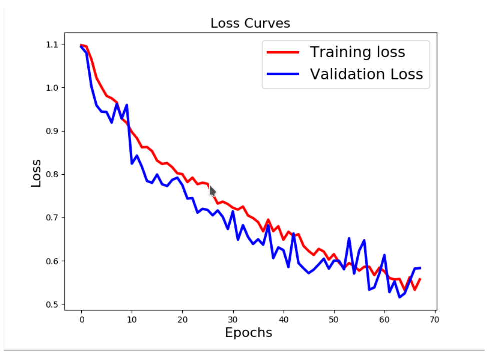
4.9 Combine Models

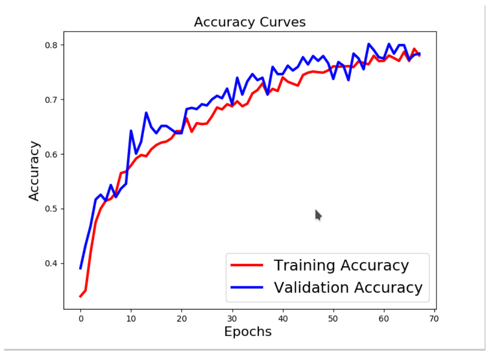
To further improve the performance, we can use multiple models to classify the test set. I did this project for more than two weeks; I have trained lots, different models, I can choose the model with very great performance. Combining the predictions of these models by taking the most frequently result as the final prediction. The ensemble prediction will be more accurate and more robust.

5 Result

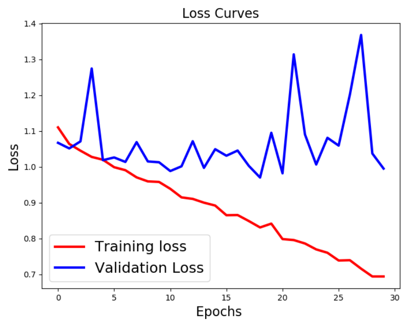
5.1 BaseLine model

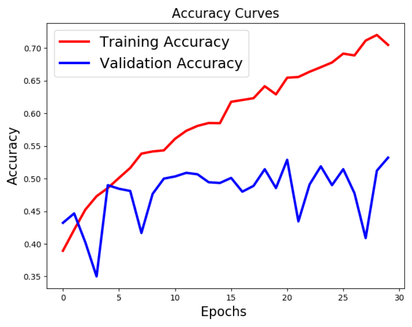
Comparing to the baseline model, my final model have much greater performance. The accuracy increased about 50%. The training time for my final model was 76.67 minutes, the time for training my baseline model was 50.5 seconds.
The diagrams are shown below are the accuracy and loss curves of my final model. From the graph we can see with the help of early stopping, the overfitting was avoided. And the loss of the final model dropped significantly.





The diagrams are shown below were the result for my base line model.





6 Conclusion

My CNN model got 78.9% accuracy on the validation set and 83.2% accuracy on the test set. Comparing to the baseline model which only got the accuracy of 39.8%, 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 and VGG 19, but the disadvantage is the accuracy is slightly lower than these two pre-trained models.

7 Future Work

To gain a better model, I should do more pre-processing for the dataset. Firstly I can add more data to enrich the dataset.
Then I can build three CNN models first. Use these three CNN model on each class to classify if the image is part of this class.
Take cherry as an example, build a model just to classify the image can represent a cherry or not. Hence I can filter out the noises.
But these could make the model not robust enough because we only train the model with the best quality data. Hence we can use the noise layers in Keras to add some noise to help the model become more robust. The last thing I think I can do is multiple select models, which all have the best performance and combine them. The combined model could be very robust and has great performance.