Project

Image Classification by CNN

Mingjie Xu

300435152

19th-October-2019

Introduction

In this project, we wanted to build a model that could perform classification on the images set and could give the probabilities it belongs to those three classes. The objective of the project is classifying images into three classes, “Strawberry”, “Cherry” and “Tomato”. All images are finely labeled, and some of the images in the dataset were not perfect. I plan to construct a CNN by Keras using python. Then it is better to perform preprocessing before training.

Pic 1

Pic 2

Pic 3

Pic 4

Problem Investigation

Exploratory Data Analysis

The dataset contains 4500 images in total and even spread into three different classes which are “Strawberry”, “Cherry” and “Tomato”. We see those fruits have the same color, except some images with unripe fruits, or rare kinds. Also in some images, the fruit is in the shade which means the colors are not the primary feature to classify the fruit. See Pic 1 for example.

The images in the dataset contain flaws, for example, a large scale of the irrelevant object in the image, some images where the fruits are obviously an ingredient that is used in a meal, the image did not contain the fruit at all, or the image contains fruit from two of the different classes. See Pic 2, 3, 4.

I want to see the size of all images in the dataset, load all the images into an array and check the shapes of each. If we want to use those data to train the model, we need to keep the size identical. By counting the shape of images through the whole set, we get the following chart.

Number of Regular Shapes (300*300*3)	4440
Number of Irregular Shapes (Other than 300*300*3)	60

This result indicates we need to resize our image data because the model requires the same value for all spatial dimensions.

Preprocessing

We used ‘ImageDataGenerator’ to solve the issues discovered in the EDA. From `keras.preprocessing.image.ImageDataGenerator` provides a range of functionalities and functions that can solve our problems.

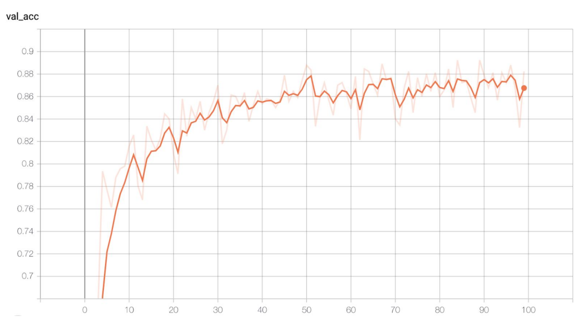
First, to resize images in the dataset, we can use `flow_from_directory` function and rescale the pictures by setting parameter `target_size` to (300, 300).

Second, we want to improve the data quality, in order to do so, we can give the setting on the Generator, but first, we need to manually remove the images that don’t include the desired objects. The reason is, each set of images is also batched from a search engine or manually collected album by a human. For example, an image is collected from a blog name “Cherry” by a search engine, or someone intended to collect an album that is widely related to the object. However, in our task, we did not include that additional information, but only the images. After removing those unclear images, we set the parameter on the ImageDataGenerator, as follows.

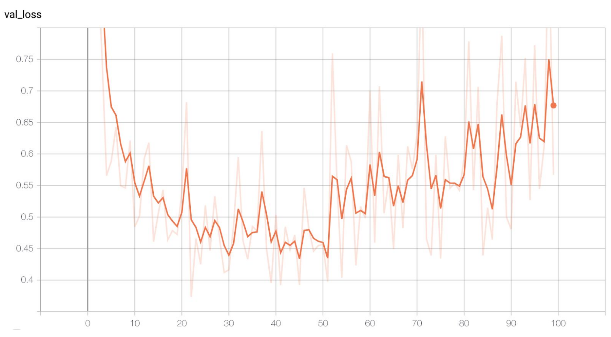
```
“rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
rescale=1. / 255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
fill_mode='nearest'”.
```

The process chosen covered rescaling, sequence of operations and filling mode. The first is rescaling, the artificial neuron prefers a small value of each pixel although they appear very dark to human eyes. This would effectively give an impact on accuracy. Pixel value ranges from 0 to 1 is best for artificial neurons to survive, thus divided all by 255. Secondly, a sequence of operations includes rotation, shift, sheer, zoom, and flip for improving data quality. The model will learn from different parts of images, after several times the model will give the most reasonable weight to each features thus preventing problems and improved the image quality. Finally, after applying those operations, there are gaps that may be caused by shift, sheer, etc. We use the ‘nearest’ filling mode, then fill the gap by the nearest pixel.

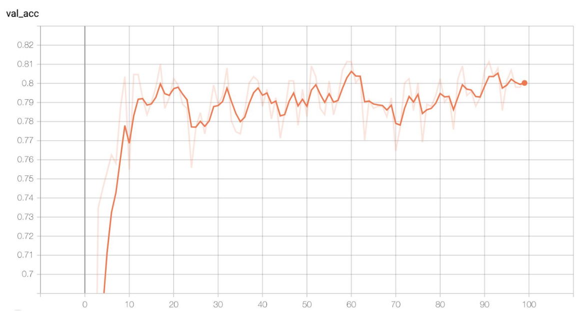
There is one with preprocessing and another without as comparison shows below.



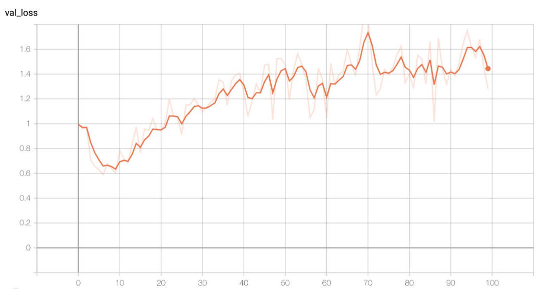
Pic5.final model--val_acc



Pic6.final model--val_loss



Pic7.noPreprocessing--val_acc



Pic8.noPrepocessing--val_loss

Feature Engineering

The features of the image are very intuitive to the human being because we can easily distinguish from the shapes, color, characteristic and even more to recall from our memory. Now for an artificial network like CNN, we need to exact features. In Keras, we were provided with a variety of layers. The Convolutional layer and the Pooling layer are designed for this purpose. By different filters in Conv2D, we can extract different features to the feature map, from low level like edges and becomes more abstract in later layers. We use the MaxPooling layer because it is not only downsampling but also remains the dominant value in the area. By adding these two kinds of layers to our network, that would enable us to extract features and save computational power at the same time.

Methodology

The CNN[1] model I built consisted of 7 convolutional layers. The aim of each convolutional layer is to create a feature map. This is the matrix that contains features after applying different filters.

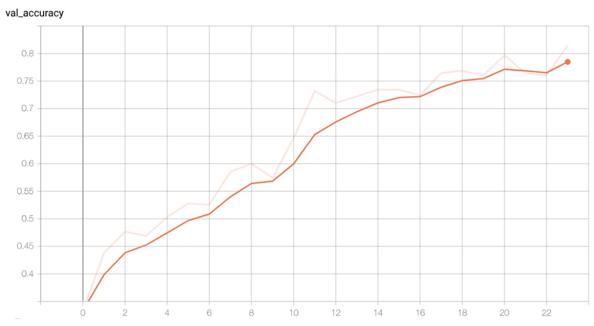
After each pair of convolutional layers, I applied the MaxPooling layer. This works by applying the 2*2 square of the feature map and saving only the dominant value in this square. It then travels all over the image and repeats this process. The selected features are then combined and a post-pooling feature map is created. This preserves the main dominant features of an image, while also downsample the image, saving the computation power. After that, a new flatten layer converts the matrix into a vector, then with the Dense layer, connects every neuron in one layer to every neuron and the layer after. Dense makes this model ready to make the classification.

1. **Make validation set.** This is a very important work as want to justify the work and evaluate the performance of our model. To achieve this, I used ImageDataGenerator again, by setting ‘validation_split’ by 0.2. This means 20% of images in the dataset will be used for validation purposes. The validation set is used to provide an unbiased evaluation of our training model because the model never learned from this dataset. Therefore it gives me a trustworthy feedback to further fine-tune.
2. **The loss functions.** This is used to measure the margin of error with predictions and the actuals. It indicates how well those parameters and techniques performed to the test data. Then leads to tuning parameters and reframing the model layers.
   In terms of the categorical loss function, Keras provides Cross_entropy loss function. One is called ‘binary_crossentropy’, and it is normally used for binary classifications, not suitable in our situation. Another is called ‘categorical_crossentropy’, that is

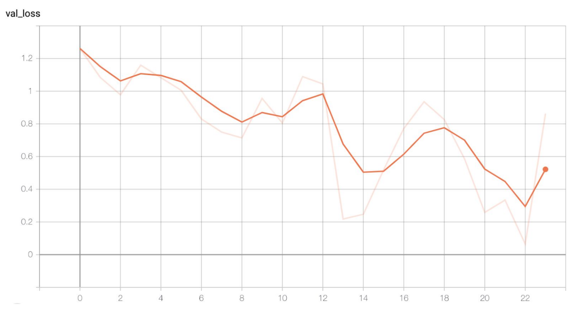
[1] "A Comprehensive Guide to Convolutional Neural Networks ...." 15 Dec. 2018, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed 27 Oct. 2019.

designed to be used with exclusive classes, as the case for this data set. Thus we choose 'binary_crossentropy' to suit the model.

3. **The optimization methods.** The optimizer is used to minimize the loss value. Some optimizers are descendants to another, so we can have one that is better in a specific situation compared to another. We chose several most popular algorithms, and they are 'SGD'[2], 'RMSprop' and 'Adam'. **SGD**[3] is the simplest one among all, after taking a certain amount of time, SGD would certainly achieve our goal, however, this process takes a long time. It may also be stuck on the saddle point. To solve this, we can give it momentum, and this can help SGD converge faster and reduce oscillations. **RMSprop** works by using momentum to drive convergence and causing the new formula in momentum preventing addition in momentum. This results in its faster convergence than normal gradient batch, and reduced oscillations than SGD. **Adam**[4] has been designed specifically for training deep neural networks. That can be considered as RMSprop with Momentum. The algorithms leverage the power of adaptive learning rates methods to find individual learning rates for each parameter. It also has advantages of Adagrad, which works really well in settings with sparse gradients. Thus Adam[5] is the quickest algorithm that converges and produces a reliable result and we can spend less learning time to achieve our result.
   I have run 'SGD' as optimizer and 'Adam', and the see Pic5 and Pic6 and below.



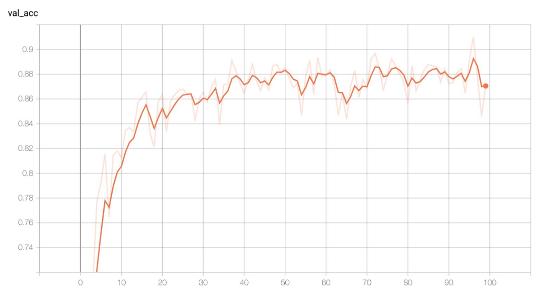
Pic9.SGD--val_acc



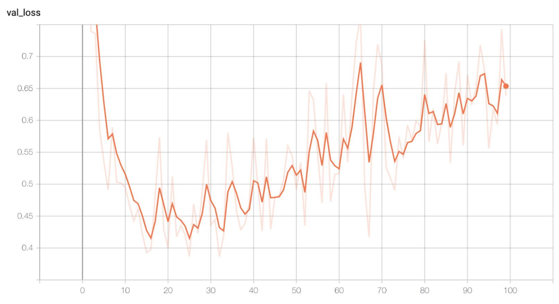
Pic10.SGD--val_loss

[2] "从SGD 到Adam —— 深度学习优化算法概览(一) - 知乎." 8 Mar. 2018, https://zhuanlan.zhihu.com/p/32626442. Accessed 27 Oct. 2019.
[3] "An overview of gradient descent optimization algorithms." 15 Sep. 2016, https://arxiv.org/abs/1609.04747. Accessed 27 Oct. 2019.
[4] "Adam — latest trends in deep learning optimization.." 22 Oct. 2018, https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c. Accessed 27 Oct. 2019.
[5] "Adam: A Method for Stochastic Optimization." https://arxiv.org/abs/1412.6980. Accessed 27 Oct. 2019.

4. **The regularization strategy**[6]**.** This is a common method to reduce overfitting and consequently improving the model's performance. There are 'L2' and 'Dropout' algorithms and these are popular to use. Dropout involves giving a random probability of keeping a certain node or not. The probability is completely set at random, we only decide on the threshold: a value that will determine if the node is kept or not. Also, this process means that the neural network cannot rely on any input node since each has a random probability of being removed. Therefore, the neural network will be reluctant to give high weights to certain features, because they might disappear. In this way, we can have a more robust model, and decreasing the chance to be overfitting.
   Results show that CNN without dropout function is more likely to be overfitting than CNN with dropout function.
   To see the Pic5 and Pic6 and below for comparison.



Pic11.No dropout--val_acc



Pic12.No dropout--val_loss

5. **The activation function**. The **activation function** give weight to the features based on the inputs. We cannot perform non-linear classification unless we include activation function. In the case of image classification is more complex than just using linear classification. The activation function in deep learning does a non-linear transform on the input, then make it possible to complex problems.
   In each of convolutional layer and dense layer excluded last dense layer, I use 'Relu' as the activation function. It only preserves the positive value and translates all negative value to zero. Therefore, there would be no more gradient disappear, and this makes converge faster than ever. In the final dense layer, I use the 'softmax' function here.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

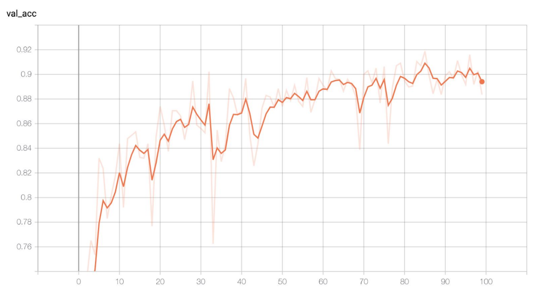
[6] "How to Improve a Neural Network With Regularization." 12 Mar. 2019, https://towardsdatascience.com/how-to-improve-a-neural-network-with-regularization-8a18ecda9fe3. Accessed 27 Oct. 2019.

It reveals that the prediction has a strong relation with input features. Conventionally speaking, with more input features belong to a class, then it is a higher possibility of this class. Thus activation function makes it possible to classify a non-linear problem and separates the whole as a set of feature weights.

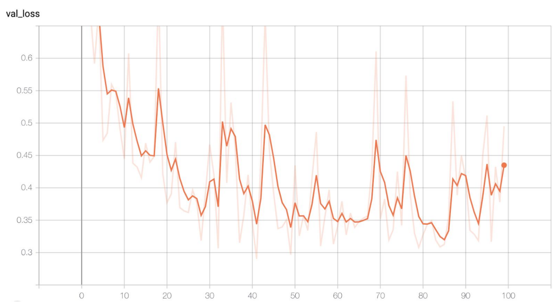
6. **Hyperparameters settings.** This is to optimize the parameter setting, in order to improve the training process.
   First I tuned the **number of epochs**, that is pertinent in optimizing the model learning and making predictions. The training accuracy is increasing as epochs number goes up, and vice versa. After the initial decreasing loss score, the validation set arrives plateau, and the model is no longer learning, then start to overfitting. On the other hand, insufficient epochs would result in the model immature, and we cannot have an optimum model. And learning time is best at around 50 epochs.
   Second I tuned **batch-size**. From my experiment, a large batch-size result in a slow learning time, but relatively high accuracy range from 2 to 64 in the model. A small batch-size result in quick learning time, but relatively low accuracy. I found that a batch size of 4 gives the best result in my model.
   Third I tuned the **number of the filters in the convolutional layer**. By experiment, all layers with a decent order of the filter numbers will result in divergence. First convolutional layer with too many filters will slow our training time because filters are feature detectors and take a large amount of time when applying it to a large input matrix, for example, the first convolutional layer, and we will use lots of filters in the last convolutional layer because input matrix would be small by downsampling. Thus numbers of filters in the different layers would take different time based on different sizes of the input matrix, and with more filters, it will produce a better feature map. Then I found setting 32, 32, 64, 64, 128, 128, 128 respectively to 7 layers gives the best results.
   Finally, I tuned the **dropout rate**. As mentioned in the regularization methodology, we can only set up a threshold to randomly keep this node in the network. High threshold would barely impact the network, cause nodes in the network are very likely to remain, or a low threshold would compromise the learning, because it may have fewer nodes to weight and result in incomplete weight network. Thus I set 0.5 to the dropout rate, and it did prevent overfitting to some degree, and I can have a greater learning rate. The reason I didn't tune 'Learning rate' and optimizer parameters, because they are very good default value based on research, and I don't have reason to tune them again.
7. **More obtained images.** A large training dataset is beneficial when training a model as it provides a greater selection of images and variation among the class. Also, it allows the model to identify a greater number of features.

To enrich my dataset, I added 900 images separated into 3 classes after I use a script[7] from GitHub to batch images from google, and I manually cleaned the damaged file and applying preprocess. Then I rescale those images by 300*300 to match the model criteria.

The result shows by increasing my data size, model accuracy on both training and validation increased by around 5%-7%. Result shown below.



More Images--val_acc



More Images--val_loss

8. **Use of existing models.** I chose VGG16[8] as an example of a pre-trained model.

   There are tons of pre-trained models and VGG16 is the most well-known one among all. VGG16 achieve over 90% accuracy. All the following results were trained on Google CoLab with GPU acceleration.

   Pros: The VGGNet[9] [10] has a clean structure, that whole network built by Convolutional kernel of size 3*3 and MaxPooling size of 2*2. We can see that a tuple of small size convolutional layer of size 3*3 is better than one large size convolutional layer of size 5*5 or 7*7. It also proves that to construct a deep network structure significant promotes the performance of the model.

   Cons: VGG takes more computational powers (Approx. 1h per epoch), and uses more parameters in each layer, occur more memory usage (140mb). The most features are processing from the first fully connected layer, VGG has even three fully connected layers. Some essay claims, there is no performance loss after removing all fully connected layers, and it significantly reduced numbers of parameters.

Result Discussions

[7] "hardikvasa/google-images-download: Python Script ... - GitHub." https://github.com/hardikvasa/google-images-download. Accessed 27 Oct. 2019.
[8] "IBM/image-classification-using-cnn-and-keras ... - GitHub." https://github.com/IBM/image-classification-using-cnn-and-keras. Accessed 27 Oct. 2019.
[9] "Very Deep Convolutional Networks for Large-Scale Image ...." 4 Sep. 2014, https://arxiv.org/abs/1409.1556. Accessed 27 Oct. 2019.
[10] "VGG in TensorFlow · Davi Frossard." 17 Jun. 2016, https://www.cs.toronto.edu/~frossard/post/vgg16/. Accessed 27 Oct. 2019.

Baseline model and final CNN model layers setting as below:

MLP	Dense	Dense	Dense	Drop out	Dense	Dens e					
CNN	Conv * 2	MaxPooling	Conv * 2	MaxP oolin g	Dropou t	Conv * 3	MaxPool ing	Dropout	Dense	Dropout	Dense

From left to the right, and “* 2” represents two of this object or layer.

The comparison between the baseline model and the final CNN model was listed here:

Model	MLP	CNN
Validation Accuracy	52%	87%
Validation Loss	0.87	0.52
Time (By Google CoLab with GPU acceleration)	7 Hours 5 Minutes	7 Hours 36 Minutes

We are making significant progress from the baseline model. Compared to baseline the accuracy improved over 50% and loss score improve around 45%. After data augmentation, the accuracy was improved to 89%.

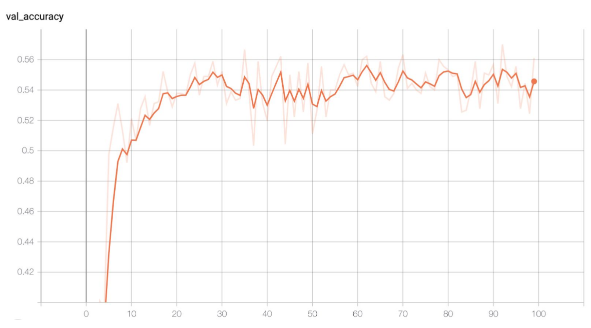
The difference between accuracy and loss score is numerous. This is because CNN is much better suited to the task of image classification than the baseline model. CNN uses both the convolutional layer and the pooling layer. Those are designed to extract features and downsampling, thus it becomes easier to handle and feature engineering improves the performance. Baseline model I built doesn’t include convolutional layers and lack dropout layers, the baseline model overfits much faster than the CNN model I build, and it cannot derive features to support classification, thus worse performance.

Convolutional layers take advantage of the *local spatial coherence* of the input. For images, this can be seen by the fact that the image loses its meaning when the pixels are shuffled[11]**.** By this property CNN is able to cut down on the number of parameters by sharing weights, however, the baseline model doesn’t share weights. And this group of neurons processes part of the input independently from other groups. This makes CNN more efficient than the baseline model.

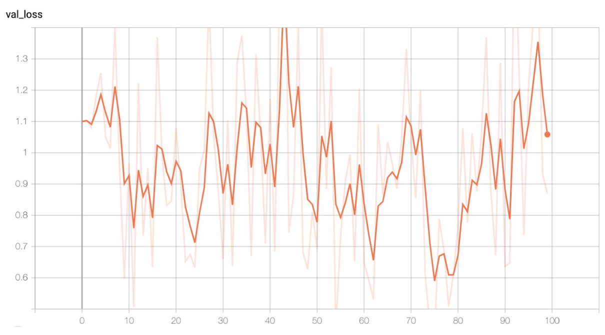
The number of convolutional layers is critical, more convolutional layers produce highly abstract features, and it is a benefit to the model learning, in contrast, more convolutional layers requires more calculates, and spend more time correspondingly. Then the pooling layer comes to

[11] **"What is the difference between a convolutional neural network and ...." 20 Mar. 2016,** [**https://www.quora.com/What-is-the-difference-between-a-convolutional-neural-network-and-a-multilayer-perceptron**](https://www.quora.com/What-is-the-difference-between-a-convolutional-neural-network-and-a-multilayer-perceptron)**. Accessed 28 Oct. 2019.**

downsample, this is very useful in image classification. The last layer of the network need to determine whether an object is present in the scene, but not *where*.
So the difference between those is the convolutional layer, a highly developed and particular for image classification tasks. The model will receive precise information in the feature map instead of the images itself, make information more designated for classification.
The result of my baseline model shows below.



MLP--val_acc



MLP--val_loss

Conclusion and Future Works

A convolutional layer is amazingly fitting in image classification tasks, and this makes CNN achieve a high accuracy. The prepossessing is also a necessary part, this will make data more robust and easy to feed to the model. A variety of optimizer help to separate features and weight them, this make model archive higher accuracy.
A CNN model was built to classify 3 classes of fruits in this study. We archived 89% accuracy by adding Convolutional layers, parameter tuning, data augmentation and so on. But loss performance is unstable and with overfitting trend. This may cause of optimizer chosen under a different situation, and training under a small dataset.
Pro:From difference I built my CNN model and VGG16, this shows use a deeper network and a larger scale of the training set are beneficial. If we want to optimize model performance, a deep neuron network should be considered, and with a large training set, which would help in preventing overfitting.
Con: In transfer learning, I have not applied the VGG16 model to combine with mine, so the performance remains the same. As the model took a decent amount of time to train, parameter tuning was not as efficient, also I desired to run SGD with momentum to compare to Adam I used.
Future work: I would construct a deeper neuron network from the transfer learning. Also, I would like to do a grid search on the data. Finally, I would like to have more images in high quality, to train a robust model.