Image Classification

Comp309

PROJECT REPORT

Author: Yuan Gao

ID: 300485853

1 Introduction

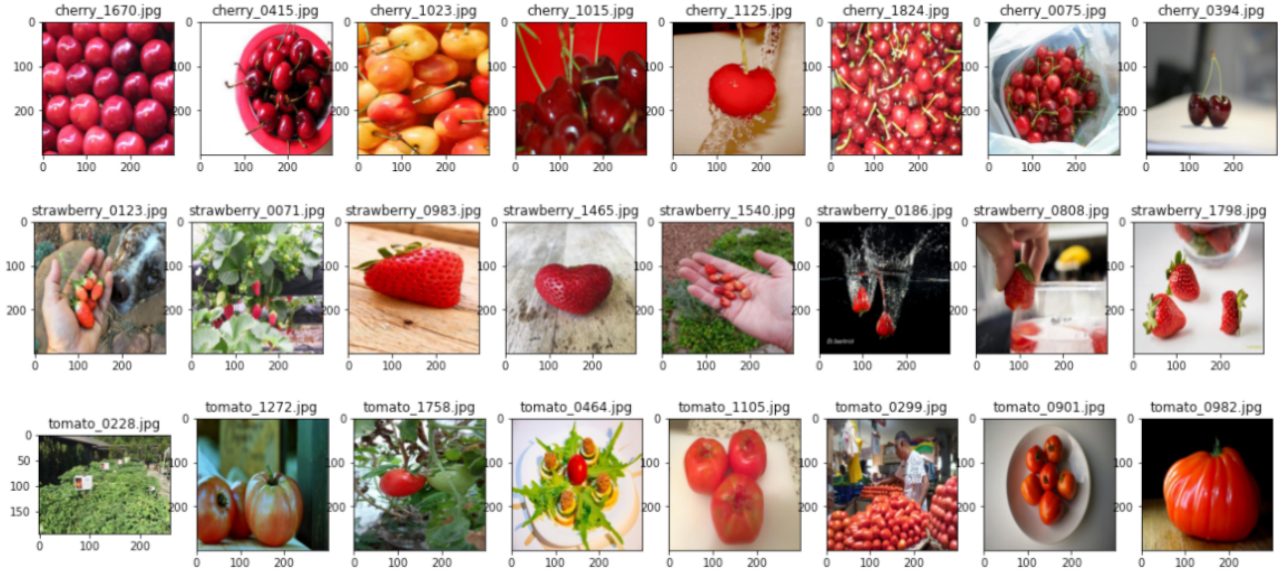
The goal of this project is to use CNN to classify images. We need to classify three classes of images, which are cherry, strawberry, and tomato. The images come from Flickr. In this project, I used Python and the package Pytorch from Python to construct the CNN model.

2 Problem investigation

2.1 EDA

2.1.1 First look at images

The data set has 4500 images and 1500 images for each class. Below are eight images I randomly picked from the data set for each class.



From the images, we can see most of the images are 300*300, but there is one image from tomato, tomato_0228, which is not 300*300. Therefore, an image-resize method is required.

In the cherry images, there are yellow cherries in the cherry_1023 and there are dark red cherries in the cherry_0394 and cherry_1015. The rest images are red. In the whole cherry dataset, there are fewer images for yellow cherries and dark red cherries than red cherries.

Similar to the cherry dataset, there are a little number of other color strawberries (unripe strawberry, green color) and tomatoes in the strawberry dataset and in the tomato dataset respectively.

Therefore, adding more images for a little number of other color fruits is a good choice for model learning.

2.1.2 Outliers

There are some outlier images in each class. There are two common issues in these classes.

Firstly, there are some cartoon fruits and unreal fruits in the images. (Below is the example.)

Secondly, there are no fruits in the images. (Below is the example.)

Both kinds of outlier images are shown in all three classes.

There is not enough data for the cartoon fruits and unreal fruits, therefore, it is hard for the model to detect the pattern. Also, I define this project as classifying real fruits. Therefore, they treat them as outliers.

In the second group of images, these images are outliers. Nothing to learn from these images.

2.2 Pre-processing

2.2.1 Delete outliers

The logic I followed about how to delete the outlier is cartoon fruits and unreal fruits in the images or no fruits in the images.

I deleted them manually. There are 21 outliers in cherry dataset, 14 outliers in strawberry dataset and 25 outliers in tomato dataset. (There are still some outliers in the dataset because I went through the dataset roughly.)

2.2.2 Resize

Resizing is a method to change the size of the image. 300*300 image is quite large and consume more memory for computing and makes longer computing time. Also, I used resnet101, pertained model from Pytorch. It is required 224*224 images. Thus, I resize the image from 300*300 to 224*224.

2.2.3 Normalization

When I used ToTensor() class in PyTorch, it automatically converts all images into [0,1]. Although this method can reduce all numbers in a range and make the model perform faster, it is not enough. Because it does not reduce the skewness in the data.

Therefore, I applied Normalization. It can reduce the skewness in the data to help learn faster and better.

Because we have three channels, RGB, therefore, the mean I used is (0.485, 0.456, 0.406) and the standard deviation is (0.229,0.224, 0.225). These means and standard deviations are from ImageNet. These figures are calculated from millions of images. Because my data are ordinary photos of "natural scenes" (fruits), they are well covered by ImageNet. The pre-trained model(resnet101) I used comes from ImageNet as well. These mean and standard deviation are well fitted in this model. Therefore, we can use these numbers to normalize the images.

3 Methodology

3.1 Random split the data

There are 4500 images. After I deleted the outlies, there are 4440 datasets left. I split the dataset into 80% training dataset and 20% test dataset.

3.2 Adding more data

I added the data for the three fruit classes from “fruits 360”[1]. It has 90380 images of 131 fruits and vegetables which also contain cherry, strawberry, and tomato.

I used these extra data to fill up the small portion of each class. For example, most cherries are red but few are black. Therefore, I added more black cherries to the training dataset to enrich the dataset to learn. Similarly, I also add the green tomato to the tomato and wedge strawberry to the training dataset.

I added 475 images more into the tomato training dataset, 739 more into the strawberry training dataset and 493 more into the cherry training dataset.

3.3 Transfer learning

In computer vision, transfer learning is usually expressed through the use of pre-trained models. A pre-trained model is a model that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve.[2]

Pytorch has the pre-trained model in the package torchvision.models. In this project, I used ResNet101. It has better accuracy than ResNet18.

3.4 Loss function

The loss function is very important in CNN. Loss is nothing but a prediction error from the model. Thus, the function to calculate loss is called loss function.

There are two types of the loss function. One is for regression problems, such as MSE, MAE. The other is for classification problems. Because classifying images is a classification problem, therefore, I tried the Negative Log-Likelihood Loss function and Cross-Entropy Loss function.

3.4.1 Negative Log-Likelihood Loss:

Formula:

$$\text{loss}(x, y) = -(\log y)$$

It maximizes the overall probability of the data. It penalizes the model when it predicts the correct class with smaller probabilities and incentivizes when the prediction is made with higher probability. [3]

It is basically used for simple classification problems. In my case, its accuracy is just over 0.33. Therefore, I would not use this loss function here.

3.4.2 Cross-Entropy Loss:

Measures the cross-entropy between the predicted and the actual value.

$$\text{loss}(x, y) = -\sum x \log y$$

where x is the probability of true label and y is the probability of predicted label.

Cross-Entropy penalizes wrong but confident predictions and correct but less confident predictions, compared to negative log loss which does not penalize according to the confidence of predictions.

In my case, using the Cross-Entropy loss function can reach over 90% accuracy. Therefore, I choose Cross-Entropy as the loss function.

3.5 Optimisation method

Optimisation algorithm works for the loss function. It helps reduce losses and give the model a better accuracy. In this project, I tried Adam, RMSprop and SGD.

3.5.1 Adaptive Momemt Estimation(Adam)

Adam's method considered as a method of Stochastic Optimization is a technique implementing adaptive learning rate. Whereas in normal SGD the learning rate has an equivalent type of effect for all the weights/parameters of the model.[4]

The official documentation is

```
torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08,
weight_decay=0, amsgrad=False)
```

I used the default parameters in the Adam, then, the accuracy of my model is 84.4 in 10 epoch.

3.5.2 RMSprop

RMSprop is a gradient based optimization technique used in training neural networks. It was proposed by the father of back-propagation, Geoffrey Hinton. Gradients of very complex functions like neural networks have a tendency to either vanish or explode as the data propagates through the function (*refer to vanishing gradients problem). Rmsprop was developed as a stochastic technique for mini-batch learning.[5]

The official documentation is

```
CLASS  torch.optim.RMSprop(params, lr=0.01, alpha=0.99, eps=1e-08, weight_decay=0, momentum=0,
       centered=False) [SOURCE]
```

I used the default parameters except momentum equals to 0.9 in the RMSprop, then, the accuracy of my model is 60 in 10 epoch.

3.5.3 Stochastic gradient descent (SGD)

SGD comes from gradient descent(GD). A Gradient Descent is an iterative algorithm, that starts from a random point on the function and traverses down its slope in steps until it reaches lowest point of that function.[6]

The official documentation is

```
CLASS torch.optim.SGD(params, lr=<required parameter>, momentum=0, dampening=0, weight_decay=0,
      nesterov=False) [SOURCE]
```

I set the momentum = 0.9, the rest parameters are as default. The accuracy of my model is 97 in 10 epochs.

3.6 Hyper-parameter settings

3.6.1 Learning rate

The learning rate is a hyper-parameter that controls how much to change the model in response to the estimated error each time the model weights are updated. [7] Low leaning rate will slow the training speed but make model converge smoothly, which means it can have higher accuracy. Inversely, high leaning rate will speed up the training but the model might not converge, which means it can have lower accuracy.

I tried some learning rate. Finally, I decided to set learning rate as 0.001. It has higher accuracy and not very long running time.

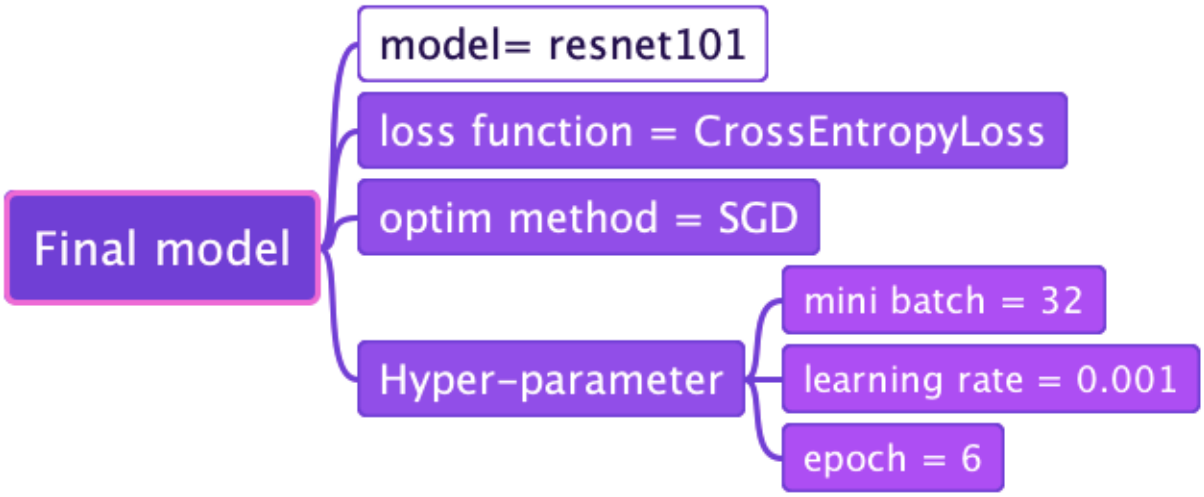
3.6.2 Epoch

The value of Epoch is the number of how many times the model training with the whole training data. The highest accuracy is reached the maximum in the 6th epoch. So, I used epoch = 6 in this case.

3.6.3 Mini Batch

The batch size is the number of training samples utilized in one iteration. I tried 18, 32 and 64 batch. 32 batch is the best one.

Conclusion for the Final model training



4 The Result of BaseLine model and CNN model

The architecture of my MLP

```
class MLP(nn.Module):
  '''
    Multilayer Perceptron.
  '''
  def __init__(self):
    super().__init__()
    self.layers = nn.Sequential(
      nn.Flatten(),
      nn.Linear(100 * 100 * 3, 64),
      nn.ReLU(),
      nn.Linear(64, 32),
      nn.ReLU(),
      nn.Linear(32, 3)
    )


  def forward(self, x):
    '''Forward pass'''
    return self.layers(x)
```

The best accuracy of this model is 0.4966 in 20 epochs in validation set. It cost 10 mins without GPU(ran on my MacBook Air).

However, my CNN model has 0.97 accuracy in validation set but cost 30 mins with GPU(ran in the Colab).

Therefore, CNN model has much better performance but slower than MLP model.

The difference between MLP and CNN

The main difference is CNN has Convolutional layer and Fully connected layer but MLP only has Fully connected layer. This will lead MLP has a poor learning process when training in a high dimension instance, such as image.

Another difference is MLP takes vector as input and CNN takes tensor as input so CNN can understand spatial relation(relation between nearby pixels of image)between pixels of images better thus for complicated images CNN will perform better than MLP.[8]

5 Conclusion

My CNN model has 97% accuracy on the validation set. The performance of CNN model is significantly improved compared to the baseline model. Therefore, The model meets the requirement that to have a better performance than the baseline model.

Is it possible it can reach 99% accuracy or 100% accuracy? The answer is no. Firstly from the given images, there are still outliers in the training set and validation set, because I did not check every image one by one. There are some outliers left.

Secondly, some fruits have very unique shape or color. There are very few to learn for the model. It is possible to classify wrongly when predicts this kind of fruits, such as the image below.

6 Future Work

97% is a good score. But if I want to have a better score or a more robust model, I will do more pre-processing for the dataset.

Firstly, I will check every image one by one and find all outliers then delete them.

Secondly, I will add more methods when transforming the images, such as horizontal flips to make my model more robust.

Thirdly, I can do ensemble learning to help boost the performance. For example, I can build three high-performance CNN models. Then, let each model vote if this image belongs to one class. We can classify this image as a class because this image gets more votes to this class.

Reference:

[1] https://www.kaggle.com/moltean/fruits?select=fruits-360_dataset

[2] Oct 23, 2018, Pedro Marcelino, https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

[3] Jan 6, 2019, https://medium.com/udacity-pytorch-challengers/a-brief-overview-of-loss-functions-in-pytorch-c0ddb78068f7

[4] Jan 16, 2019, Biboswan Roy, https://medium.com/@Biboswan98/optim-adam-vs-optim-sgd-lets-dive-in-8dbf1890fbdc

[5] https://deepai.org/machine-learning-glossary-and-terms/rmsprop

[6] Sep9, 2020, Aarthi Kasirajan, https://medium.com/@minions.k/optimization-techniques-popularly-used-in-deep-learning-3c219ec8e0cc

[7] Sep 12, 2020, Jason Browniee, https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[8] Mar 7, 2019, Momen Negm, https://www.linkedin.com/pulse/mlp-vs-cnn-rnn-deep-learning-machine-model-momen-negm#:~:text=MLP%20stands%20for%20Multi%20Layer,stands%20for%20Convolutional%20Neural%20Network.&text=So%20MLP%20is%20good%20for,problem%20they%20are%20designed%20for.