Introduction

For this project, our main objective is to construct a CNN model that can let the computer to identify and classify which fruit class does this image belongs. These 3 fruit classes are tomato, cherry and strawberry. By using the constructed model after training, we expect that this fruit classification task can be used in the real life. For achieving this objective, first the typical baseline Model: multilayer perceptron (MLP) model will be used, and then apply the CNN model and tune the parameter based on the previous discovered result in order for the better performance. This project is all done with the help of PyTorch.

Problem investigation(Background)

The dataset come from Flickr. We have only got the subset of the whole dataset in which there are only 4500 images are provided out of 6000 images in total, the rest of 1500 images are used as the unseen instances to evaluate the performance to give marks. Each class got 1/3 of the data, which is perfectly balanced.

Exploratory Data Analysis (EDA):

noisy images that do not have any visible fruits on it

Through the observation on the provided images, I find that although classes are perfectly balanced, there are some noisy images across 3 classes that are absolutely irrelevant and can not represent these fruits. As we can see from above screenshots, some of them only got the fruit word like Cherry on the image, since we are classified images, not words, so these are considered as noisy images. For the rest, they are absurd such as there is a dog, no fruit is presented, so even a human can not see and figure the fruit inside the image. Therefore, these noisy data should be deleted since they can only confuse and affect the model.

Left image is not 300x300, other 3 images are grey images from cherry, strawberry and tomato classes

Beside these noisy images, through the further EDA, I find something interesting. From above images, we can see that although the handout point all images are RGB images and are re-sized to the same size/dimensions, there are still some exceptions. From the 1st col, we can clearly see this images is not 300 x 300 size. From what we have learnt so far, all images need to be normalized into the same size before training, otherwise it will greatly affect during the model fit process. Especially the classification images task of CNN is partially based on the size.

Then, we can clearly the rest 3 different fruit images do not have any colour. These images are very typical in which the main fruit object is placed in the middle. From the further search on internet and school slides, it reminds me that on CNN model, the colour is not a high weighted feature, the shape of fruit itself is more important. Due to this reason, these grey scaled images are not part of noisy data and worth to keep.

As I have mentioned, fruit shape is pretty important for the CNN to learn, so the occlusion is a big issue. Through the further EDA, as we can see from above, there are only small portion (only few pixels) of the main fruit object is visible, other larger areas are fulfil by others such as the jar which is noisy that need to be ignored. These are the big issue that need to be handled. For the right 2 images, although we can clearly recognize that they are tomatoes, there are only a little part images are with these shape, which means we might need to enrich the dataset.

Preprocessing and Data Augmentation

After the EDA is finished, through observations, I try to apply the proper pre-processing technique on that. Some of them are inspired by the tutorial website[1] [5] and the book[2]. I will just only describe what I have applied for my final code.

1. Remove noisy data
    - As I have mentioned above, these noisy data can only affect the model performance, there is no doubt that they are removed first.

Then, for the code part, the ImageFolder module is used for loading images and transformers module is used for applying and combining multiple transformations together.

```
# scale each images into the same size
transforms.Resize((300, 300)),
```

2. Resize and scale all images into the same 300*300 size.
    - From the EDA part, I obtain that the size of image is a great impact that will affect the CNN model and there are still images that are not 300x300. So, transforms.Resize is used for me to scale each image into the same 300x300 size.
3. Transform all images to the tensor object.
    - From the pytorch doc[3], I obtain that all images need to be the tensor object in order for the model to train, so I just call **transforms.ToTensor()** to transform them into tensor object.

```
1   transforms.Normalize(mean=(0.5, 0.5, 0.5),std=(0.5, 0.5, 0.5), inplace=False),
```

4. Normalize the channels of these tensor images object with mean and std.
    - For our images, RGB 3 channels are used and the value is range from 0 ~ 255. From what I have learnt so far, obviously, this range is too big that may also affect the performance of images. Therefore, from [4], as we can see from the above code snippet, the normalization is applied in order to make the RGB 3 channels to be the value range between [-1,1], the math formula is channel=(channel-mean)/std. By

applying this transformation, the skewness is reduced, which can make the model performance to become better.

```
transforms.RandomRotation(degrees=(0, 180))
```

5. Random rotate each image.
    - For this one, all image will randomly rotate between 0 ~ 180 degrees based on their own shooting angles. As I have mentioned above, the fruit shape is more important then colour, and through the EDA, I discover that only a small portion are visible on some of images and angles of fruit object are various. I want the model can be able to learn and detect the fruit from both positive and negative angles, so that's why this technique is applied.

Beside this, I have also try to apply some other techniques such as CenterCrop, Grayscale. For grayscale, it will grey scaled all images which can reduce the RGB 3 channels into 1 channel, this idea is inspired by [5] and EDA. However, the model performance accuracy is not better and even become worse then the original RGB 3 channels. Therefore, I think for our dataset, it would be better to keep the origin 3 channels. For the other transformation augmentation, such as PCA from book[2] and GaussianBlur from [5], due to I discover them a little bit late, I have not try them all, which is pity.

CNN Methodology

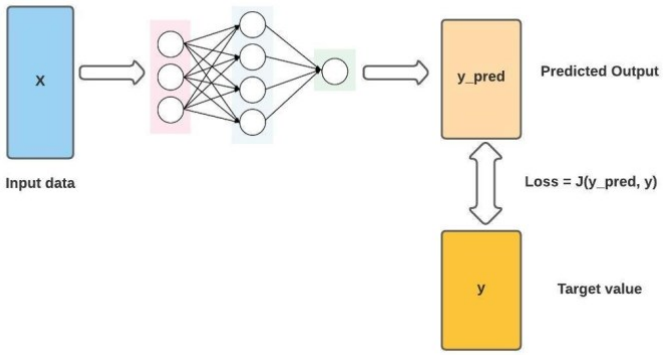
you are required describe and justify what you have done for training the `CNN` in terms of the following aspects if applicable, but you should include at least five of them:

1. how you use the given images (e.g. how you split them into training and validation sets or k-fold cross validation, and how you use them for training),

```
# split the data into train/valid set
train_set, valid_set = train_test_split(dataset_whole, train_size=0.8,
                                        random_state=309)
```

The above code snippet is how I use the given images. As we can see that, I just simply use the train_test_split() to split the whole given images into training and validation sets, the train size is set to 0.8. In order to control the uncertainties, the random_state is used to make sure that each train valid spilt will be the same so that it is helpful for further aspects such as the adjust hyper-parameters in which the loss and accuracy visualization plot is more trustable.

2. the loss function(s),



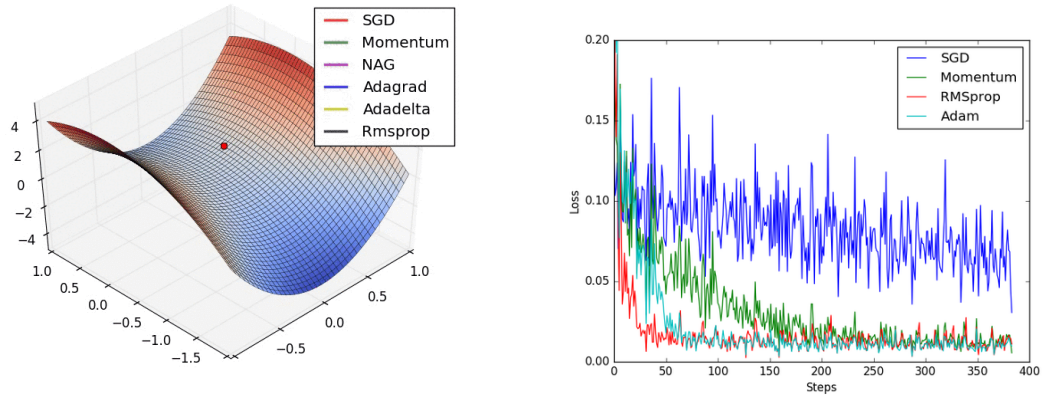
Graph from [6]

For loss functions, from [6], I obtain that the loss function take 2 args which are the predicted output(y_pred) and the provided real output (y_true) in order for calculating and gauging the error between them. Therefore, within use the loss function, we can know how far the algorithm model (i.e. CNN) is from realizing the expected outcome(y_true). And the word 'loss' means the penalty that the model gets for failing to yield the desired results so that by back

propagate it through the specified optimizer, the model performance will be optimized.

Through the search[8], due to our project is the classification task, I obtain that the CrossEntropyLoss() is good for classification task and used as my loss function. For the reason, it is because[9] the distance between two probability distributions: predicted and actual are minimized. For instance, if our the CNN model predicts 0.2 probability as cherry, 0.7 as strawberry and 0.1 as for tomato class. And the true class for this image is cherry, so the true prob should looks like [1,0,0] rather than [0.2,0.7,0.1]. Therefore, the most ideal expectation is that our predicted probabilities by CNN model should be as close to this original probability distribution as possible. The cross entropy can make sure the difference between the two probability are minimizing, so it is well suit to our classification problem and be used by me.

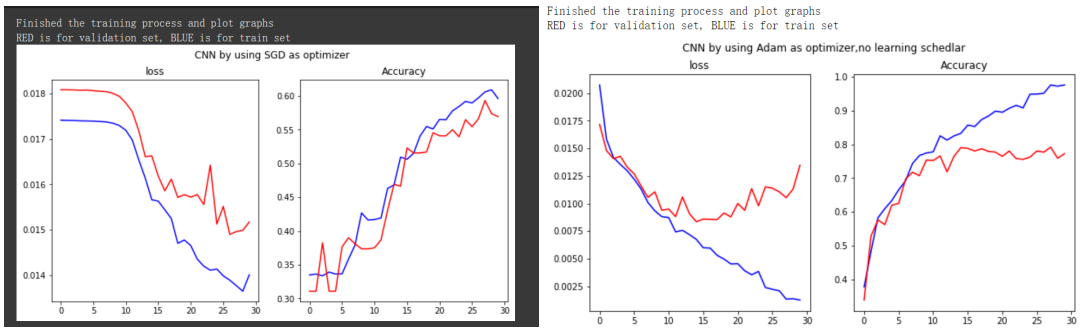
3. the optimisation method(s),



Left GIF[10] shows SGD optimization on loss surface contours and Right one[16] shows compartion between different optimizers

As I have mentioned above, the optimisation method is for minimizing and passing the error, between y_true and y_pred in order to optimize the performance. There are various options such as Adam, SGD, RMSProp, AMSGrad, etc that are gradient descent optimization algorithms. Through the tutorial[10], I obtain that the gradient decent is actually the way for helping us to reach the (local) minimum from the huge possible solution space and there are lots of variants.

Red Line is the validation set, BLUE is for training set



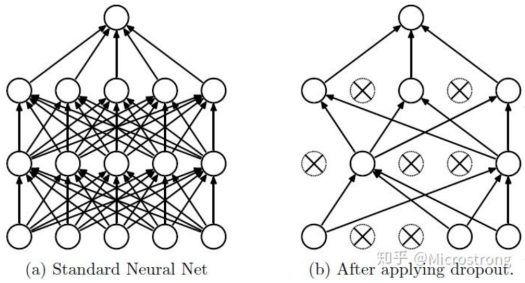
For me, I test to use the SGD and Adam as optimizer. I use momentum: 0.9 for SGD, and 0.0001 learning rate for both SGD and Adam.

As we can see from above images, by comparing these 2 plots, we can clearly see that the Adam perform better then SGD, and the coverage speed is much more faster. From [12], I obtain that basically SGD only maintains a single **learning rate** (termed alpha) for all weight updates and the learning rate does not change during training. For Adam, it realize this kind of issue and take the benefits of both AdaGrad and RMSProp, so it is much more effective then others. As we can see that, when 30 epoches end, the accuracy for SGD is only around 0.6 while for Adam, it is much more better in which is about 0.8. Therefore, I choose Adam as my final optimizer.

By comparing the two graphics we have that although steepy, the Adam is on the rise overall and the accuracy is pretty high, while the line for SGD floats up and down and the accuracy is always minus 70%. According to the comparison, I adapt Adam as the optimizer.

4. the regularisation strategy,

By googling[13], I obtain the knowledge that regularization is a general term for methods that introduce additional information into the original loss function in order to avoid over-fitting and improve the generalization performance of the model. Therefore, by applying the regularisation strategy, both the origional data will be kept that avoid the potential data loss and the case that the performance accuracy on the unknown and unseen dataset is extremely lower than the training set can be avoided, which improve the generalization of the model.



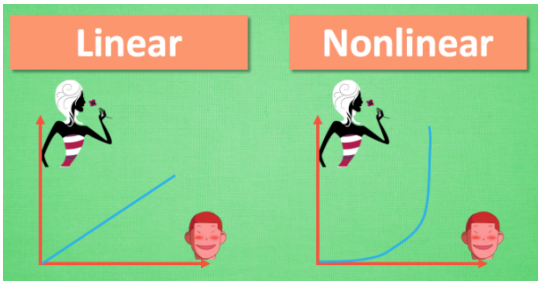
(a) Standard Neural Net (b) After applying dropout.

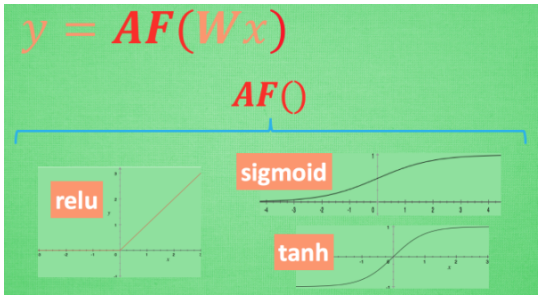
First, the dropout regularisation strategy is applied by me within my code. As we can see from the left graph[14], the left part is the full connected neural net in which all neuron nodes are used. For the right one, the dropout strategy is applied so only part of neuron nodes are in used. From [14], I obtain that each neuron node carries the data of the image partial feature and pass forwardly to the upcoming nodes, so, if they are fully connected, then the model will be overfit since it depend on partial features. By applying the dropout, some nodes will be dropped and so that the model will not rely too much on certain local features thus improve the generalization of the model. In my code, I set up the 0.25 as the prob for each layer, so that only 3/4 of neuron nodeswill be kept.

```
# check if the loss on valid set is improved or not
if final_valid_loss_mean <= valid_loss_min:
        print('-'*10)
        print('Validation loss decreased ({:.8f} --> {:.8f}).  Saving model ...'.format(
                valid_loss_min,
                final_valid_loss_mean))
        # torch.save(model.state_dict(), MODEL_SAVE_PATH)
        valid_loss_min = final_valid_loss_mean
        valid_acc_best = final_valid_acc
        best_model['Best epoch'] = epoch
        best_model['model'] = model.state_dict() # assign this model since it perform better
        best_model['Best Acc on validation set'] = valid_acc_best
        best_model['Best(lowest) validation loss'] = valid_loss_min
        p = 0   # reset P
else:
        print('-'*10)
        p += 1
        print(f'{p} epochs of increasing validation loss')
        if p >= 15:
                print(f'Stopping training since no improvement on the validation loss over {p} epoches ')
                stop = True
                break
```

Beside this, as we can see from the right screenshot, I have also add the code snippet to avoid the overfit. We can see that I add an if else to check whether the loss on the validation set for this epoch is the minimum or not. If yes, then the model.state_dict() will be saved to the variable, since it is the best. If it is not, and has already occur previously more than 15 epoch of times, then it will early stop the training process since it has already overfit. I think my code snippet is helpful since the best generalization mode is guaranteed saved.

5. the activation function (aka. AF)





For the activation function(aka. AF), from[15], we can know that it is used for taking the linear function as the arg and output to be the non-linear, the example image is shown on the top left. The existing of the activation function is because XOR instances are only non-linear separable, so that by applying the activation function, the performance can increase since some XOR instances can be classified correctly.

As we can see from the right screenshot, there are various options to choose. As [15] suggest, I finally use relu as my AF. For the simple network that only got 2 ~3 layers, we can apply any AF to the hidden layers, there will be no bigger difference. However, there are lots of layers in my CNN, so if randomly choose the AF, then it will involve the problem of gradient explosion and gradient disappearance. **Therefore, I choose relu since it can address this kind of problems and I apply it on after each hidden layers.**

6. hyper-parameter settings,

For hyper-parameter settings, it involve any parameters that is used to control the learning process, which include epoch numbers, convolutional layer parameters, optimizer learning rate, etc. The details setting for each of them are described below.

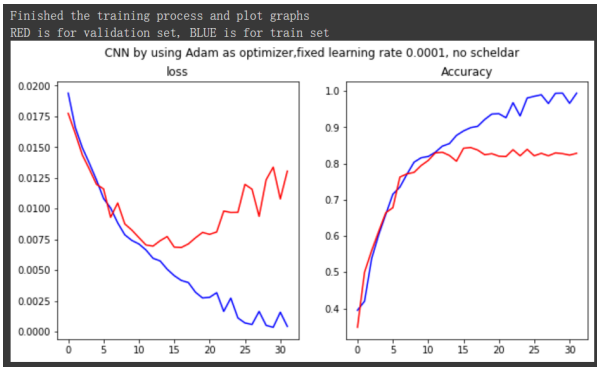
Batch Size

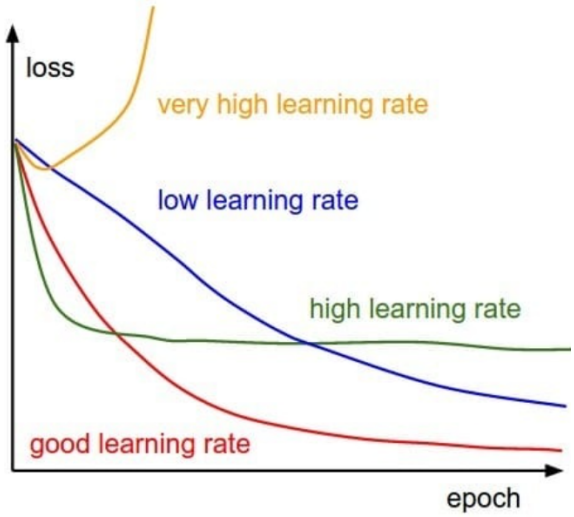
The batch size is the hyper parameter that affect the number of images are passed to the model to train in the current iteration, so it affect the time required to complete each epoch and the smoothness of the gradient between each iteration. If the batch size is too small, it will take too much time to finish training, and the gradient oscillation is serious, which is not conducive to convergence. If the batch size is too large, the gradient direction of different batches does not change, and it is easy to fall into a local minimum. Through the test and the recommendation, I finally choose 64 as the batch size.

Epoch

If all images are iterated once by the model to learn, then it means one epoch. The epoch can not be too big since it is time consuming and the model will overfit and also it can not be too small since it will cause the model is under-fit. Through the testing, I find out that the epoch should be set to higher then 15 in order to fit model well. As I have mentioned on 4. the regularisation strategy, I manually set up and store the best model if the validation decrease. **And stop the training process if the validation is increased over 15 epoches of time since it means the mode is not optimized and stuck at the local minimum, so there is no need to continue the training process anymore. We can see that for this hyper parameter, I adjust it dynamically and flexible.** Therefore, finally I set 150 epoch as maximum.

Learning Rate



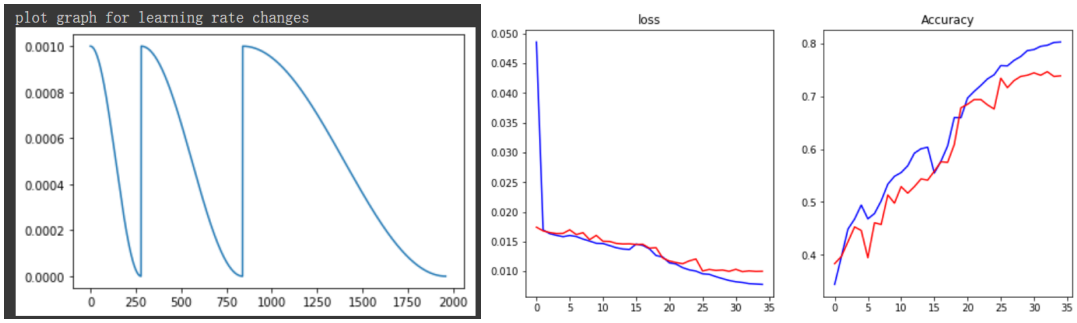


As I have mentioned above, Adam optimizer is used and the learning rate is the parameter of Adam. It basically determines the size of steps we take to reach a (local) minimum. In other words, it follow the direction of the slope of the surface downhill until the valley(i.e. local minimum) is reached. From the top right plot, we can see that the learning rate is hard to adjust since the low learning rate bring the consequence that the fitting is slow, and too high/high learning rate will let the loss unstable and stuck earlier.

Through the test, I find out that 0.0001 should be a good learning rate and the plot is shown at top left, it is the best one that I got. We can see that it validation accuracy finally stop increasing at the accuracy of around 0.8.

Learning schedulers

Red Line is the validation set, BLUE is for training set

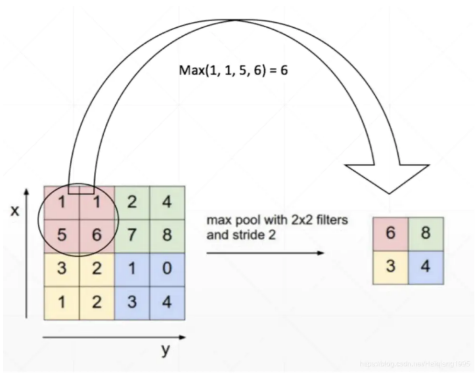


Beside by using the fixed learning rate 0.0001, follow the reddit post[16], I obtain that there is a technique called learning schedulers that can change the learning rate dynamically during the training process for different epoch. It is recommend us to use the learning scheduler with the warmup since without warmup, the optimizer may be leading to trapped in a bad local min. It also suggest us that for the Adam optimizer, learning scheduler include a warmup and some annealing, either linear or cosine is the best. Therefore, through the further research on pyTorch doc[3] and a very good tutorial[17] that talk about the learning scheduler, I decide to apply the suitable CosineAnnealingWarmRestarts() and write the code based on [18].

As [17] suggest, I try to set up a pretty high learning rate 0.001 as the initial start learning rate. From the above 2 plots, we can see how learning rate is changed as well as the associated loss/accuracy change plot.

Number of hidden layers and units on CNN

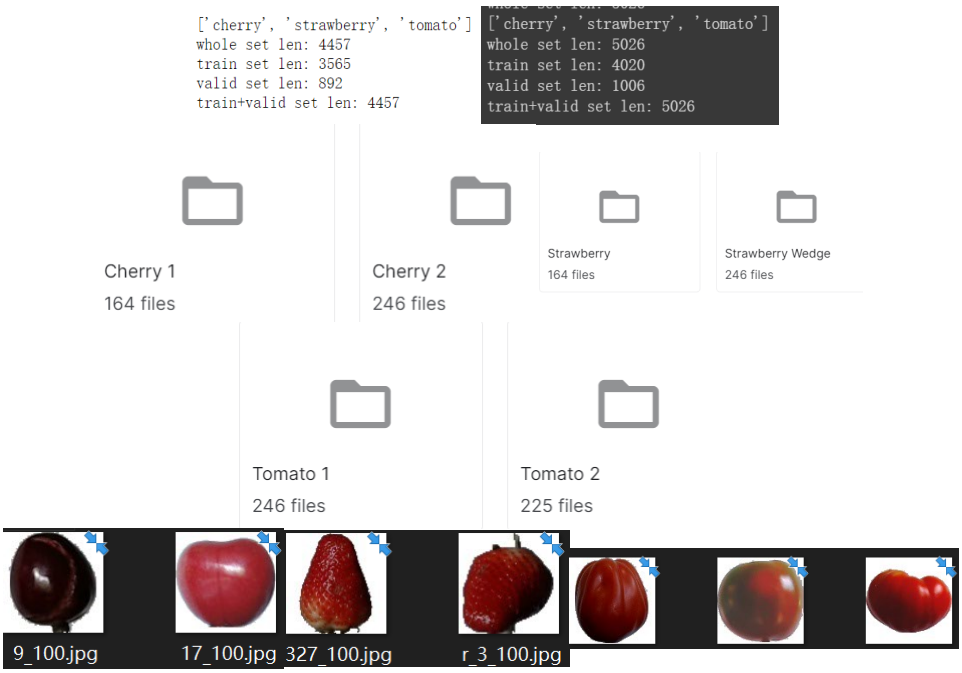
```
nn.Conv2d(in_channels=32, out_channels= 64, kernel_size=3, stride=1, padding=1),
          nn.BatchNorm2d(64),
          nn.ReLU(inplace = True), # activitation function
          nn.MaxPool2d(2, 2), # output: 64 x 75 x 75
          # use dropout to avoid over-fit
          nn.Dropout(0.25),
```



From [19], I obtain that until test errors are no longer improves, it is usually good to add more layers since small amount of units may cause the under-fit problem. On my code implementation, there are 6 layers in total. For each layer, nn.Conv2d, nn.BatchNorm2d and maxPool2D are used, which we can observe from the above snippet. Within use them, more important information features will be separated and discovered by the convolution_2D, and then the performance will guaranteed to be stable by the BatchNorm2D. And then, from right screenshot, we can see that the important data(highest value) will be extracted by MaxPool2d and unimportant noisy information will be removed so that the calculation cost is decreased. Finally, Dropout can make sure that our model will not depend on local features which improve the regularization.

By repeating these operations once and once across 6 layers, more important features are extracted and kept, which cause the model be well fitted.

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,

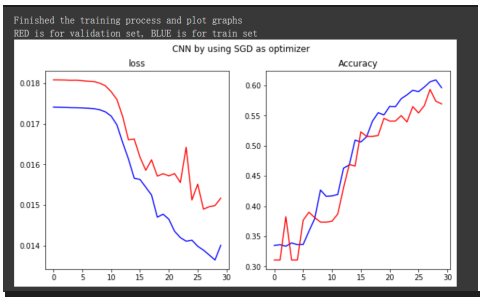


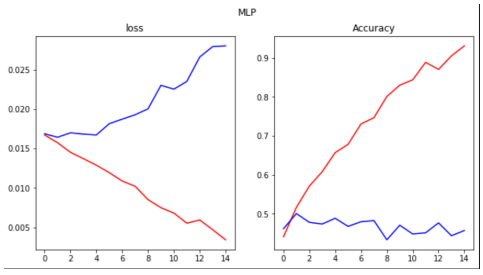
For enriching my training set, I search and download the exist dataset online from the **Kaggle[11]**. As we can see from the above screenshots, we can clearly see that there are much more then 200 images within the dataset and images itself are all fit the purpose. Therefore, I randomly add 200 more images to enrich my training dataset.

Result discussions

Note: this CNN within use the SGD is not the best that I've got!!
It is just for comparing the difference between MLP since they use the same SGD optimizer and all hyper-parameters are the same!!

Red Line is the validation set, BLUE is for training set






```
MLP(
  (layers): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=270000, out_features=64, bias=True)
    (2): ReLU()
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Linear(in_features=32, out_features=3, bias=True)
  )
)
```

```
Wed Oct 20 18:16:59 2021
Epoch 0:
 training step:60, train loss: 0.0167, valid loss: 0.0169
Train set accuracy: 0.4411
Valid set accuracy: 0.4619
```

```
Wed Oct 20 18:21:36 2021
Epoch 14:
 training step:900, train loss: 0.0034, valid loss: 0.0280
Train set accuracy: 0.9311
Valid set accuracy: 0.4574
```

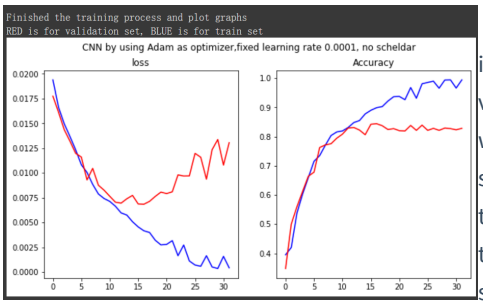
For my MLP setting, it is extremely simple. Just several linear transformations are used. For controlling the outlier, other hyper parameters are set as the same. As we can see above images, we can clearly observe that:

1. The CNN accuracy is much more better then the MLP. For CNN, for the validation accuracy by using SGD, it reaches around 0.55 ~ 0.6 and from the loss plot, we can see that the over-fit problem is not occur. However, for the MLP, we can clearly see that it is over-fit since start. The loss for validation set is increased but is decreased on training set, and finally the training accuracy is closer to 1, but for validation set, it just keep the same which is about 0.46.

2. For the training time, images on the top indicates the training time taken on MLP. We can see it just take 5 minutes from epoch 0 to epoch 14. At the same time, for CNN, it takes 15 minutes from epoch 0 to 14, we can see the time difference is huge.

```
Fri Oct 22 08:32:07 2021
Epoch 0:
```

```
Fri Oct 22 08:48:14 2021
Epoch 14:
```



For the best performance that I have got on CNN, the associated plot is shown on the left. We can see that on the first 17 epochs, both the validation and training loss works as what we expect, but for the rest, we can clearly see that the model is overfitted. It is because we can see that the loss on validation set is increasing whereas the loss on training set is decreasing and become much more closer to 0. Since the regularization is applied, so as we can see from the below screenshot, it shows the best data associated with the lowest loss(0.0068) on the validation set, in which the best epoch is on 16 and the accuracy is about 0.8439

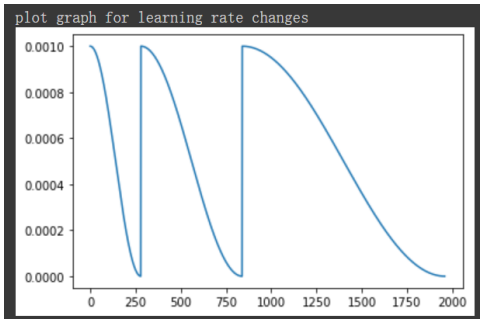
```
Best epoch: 16
Best Acc on validation set: 0.8439363837242126
Best(lowest) validation loss: 0.006838291083605104
```

Although the performance on my MLP is much more worse than CNN, MLP can still be used for the Image classification task. In my opinion, the reason why CNN and MLP are different is because the MLP takes the vector as input, but CNN forced to take tensor object as input. Due to this reason[20], the CNN model can understand the spatial relation(relation between nearby pixels of image)between pixels of images better. Therefore, the MLP can not handle the complicated image classification task but CNN can, so that's why the CNN perform better than MLP.

Conclusions and future work

In conclusion, I think I have done a good job overall. For the pros, I apply to lots of technique to control the overfit such as the regularization so that the best model with the lowest loss on validation set is guaranteed. Also, 6 layers are used for CNN model which means the model is fit pretty well.

For the cons, I think I still need to try something more, which is the future work that I need to do. I think I should try to apply more preprocessing techniques in order to make the performance better. For instance, the PCA from book[2], GaussianBlur and some more Functional Transforms from [5].



Also, for the learning schedular, my testing result is still not optimal, so it is worth to try again to adjust the hyper-parameter of the learning schedular. The concept of warmup restart and Cosine Annealing is interesting since it can help to jump out the local minimum to try the best to find the global minimum in order to achieve a better result. As we can see from the right image, warmup restart can make sure that the learning rate will back to initial highest so that the new period starts. The most optimal case is that the local minimum is reached at the final of each period, so that once the new period restart, it can jump out to find the next downhill(another local minimum).

Also, since I have not try existing models such as the transfer learning, VGG net and extra technical advancements such as the ensemble learning to help boost the performance. I think I might also try to apply them as well, website[7] might be an useful tutorial that can lead me to quick start.

References

0. school slides and tutorials

1. https://towardsdatascience.com/improves-cnn-performance-by-applying-data-transformation-bf86b3f4cef4
2. Book: Programming Computer Vision with Python
3. https://pytorch.org/docs/stable/index.html
4. https://pytorch.org/vision/stable/transforms.html
5. https://www.analyticsvidhya.com/blog/2021/04/10-pytorch-transformations-you-need-to-know/
6. https://neptune.ai/blog/pytorch-loss-functions
7. https://machinelearningmastery.com/stacking-ensemble-for-deep-learning-neural-networks/
8. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/
9. https://www.quora.com/When-should-you-use-cross-entropy-loss-and-why
10. https://ruder.io/optimizing-gradient-descent/index.html#gradientdescentoptimizationalgorithms
11. https://www.kaggle.com/moltean/fruits
12. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
13. https://towardsdatascience.com/regularization-an-important-concept-in-machine-learning-5891628907ea
14. https://zhuanlan.zhihu.com/p/38200980
15. https://mofanpy.com/tutorials/machine-learning/torch/optimizer/
16. https://www.reddit.com/r/MachineLearning/comments/oy3co1/d_how_to_pick_a_learning_rate_scheduler/
17. https://www.jeremyjordan.me/nn-learning-rate/
18. https://zhuanlan.zhihu.com/p/261134624
19. https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network-7f474f91d7bd
20. https://www.linkedin.com/pulse/mlp-vs-cnn-rnn-deep-learning-machine-model-momen-negm