COMP3359 Artificial Intelligence Applications

Cherry, Peach, Plum Blossom Classifier

Group Member:

Wan Tsun Wai	3035569017
Wong Ka Ngai	3035568881
Lui Pui Him	3035574115

Project Objectives

In spring, we often see people take photos with flowers. However, do they really know what flower they are taking pictures with? Can you distinguish blossoms at the front page? Probably not. Lots of flowers are very alike, especially cherry blossoms, plum blossoms and peach blossoms. Therefore, we are going to build an application to identify these flowers.

Application Description

We are interested in classifying different types of flower objects with a similar appearance. In particular, blossoms of cherry, plum and peach are hard to distinguish for human beings. Therefore, we would like to have an AI to learn to make decisions and differentiate for us. Users can input photos with flowers of one of these three types to our application for classification.

Highlight of the work

1. We constructed a dataset using 618 images with data augmentation from website Pixabay.

2. Convolutional Neural Networks (CNNs) are used for building the image processing and classification model. Besides building our own CNNs, we used transfer learning on ResNet, VGG19 and Inception V3 to analyze model performance.

3. We identified that the dataset and model complexity are the major constraints in affecting the performance of our model. Although due to limitations we have not completely implemented the solutions yet, we propose some ways in the report for further improvement.

Data Engineering

Dataset Source

Dataset for training and dataset for testing are of the same source (Pixabay) and the data images are of high quality and copyright-free. These images are uploaded by normal users like you and me. Using images like these can resemble the user input of our application, so the test dataset should be very close to real user input. Difference of data distributions between training data and test data is reduced to avoid data mismatch.

Data Processing

We gathered 2000+ images tagged as cherry blossoms, plum blossoms and peach blossoms. However, as most people fail to distinguish them, most images are wrongly tagged. We selected all the images that are correctly labelled, but only around 600 images remain. Therefore, we have 206 images for each of the class, so in total we have 618 images.

We used 498 images for training data, 120 for testing data. Test data is divided into Dev set and Test set, having 60 images for each. As we have the same number of images for different classes inside, it is hoped that it will not favour any class and will not train more on a particular class. We also have observed the problem of overfitting in our interim prototype, thus we have doubled the dataset size in order to avoid the occurrence of overfitting.

However, a dataset with 618 images is still not a big one and the overfitting may still be present, it may even lead to bad model performance. Therefore, data augmentation is used on the training set to increase diversity of our dataset by creating new data based on existing data. Random rotation with the range of degrees (-90, +90) and a 50 percent chance of random horizontal flip is used. These two augmentation methods are used as it is common that after taking a photo, the resulting image is rotated or horizontally flipped. So we should have no problem training the model with flipped or rotated blossoms. Even though it is also okay to do augmentation to our test data, we do not do so here because we believe that most users input images in the correct orientation.

Since our image quality is quite high, we need to first resize the image to a smaller size (200,200), then transfer it to tensor objects (with data augmentation on training set) which we can use for training and testing. Finally, we import the images and split them into training set (train_ds), validation set (dev_set) and testing set (test_ds).

```
No of train ds Classes:  3
['cherry', 'peach', 'plum']
Training Set Size:  498
Dev Set Size:  60
Test Set Size:  60
```

AI/ML model implemented

We are going to deal with a dataset of flower images, capturing cherry, peach or plum blossoms. A classifier model will be trained to determine what blossoms are captured in those images. It is worth emphasizing that every image captures one type of blossom only, so each prediction can only be one of the three classes. This means that every instance in the dataset is mapped to one label. This is known as a **Supervised, Multi-class Classification** problem.

Convolutional Neural Networks (CNNs) is a type of Neural Networks that is well-known for its effectiveness on image processing. As we are processing images of blossoms, we will build our Multi-class Classification application by constructing a CNN model.

Through feeding three types of similar but yet different flowers into the Convolutional Neural Network, we hope to train a model that can classify the type of flower we input.

Configuration of CNN Model

Fine tuned CNN Model (CNN2)

In this final CNN model, we added an extra sequential layer to our prototype model and fine-tuned the dropout rate. Even though it seems like nothing big has changed, numerous experiments have been done to get this result. We can see the detailed reasons behind using these hyperparameters in section Experiment Results Analysis on Fine-tuning CNN2 model.

This model consists of five sequential layers and one fully connected layer. It takes RGB (3 channels) images of size (200, 200) as inputs and outputs 3 values, representing the confidence score of the three classes.

Convolution Layer
In each sequential layer, images/feature maps will go through convolution layers (Conv2d) to extract features from input images/feature maps to the layer, and return some feature maps.

Normalization Layer
Then we go through a normalization process (BatchNorm2d) to force them to fall into range between 0 and 1 in case it slows down our next process.

Activation Layer
After normalization of data, we go through the activation layer (ReLU) to transform all input to either 0 or the value itself. The reason that we didn't use sigmoid but ReLU is that it has fewer vanishing gradient problems. Also, the computation cost of ReLU is significantly lower than other activation functions.

Pooling Layer
The final layer before output is the pooling layer. After the activation layer, we go through the pooling layer (MaxPool2d) to reduce the dimensions by half.

While the Dropout with 10% chance prevents co-adaptation of feature detectors.

For each layer, the output tensors will have shapes which the width and height is reduced by half. After passing through five sequential layers, number of channels increases from 3 to 512 and size reduced from 200*200 to 6*6. Finally, it will pass through the fully connected layer to finish the model for future training.

Fully-Connected layer
A Flatten layer is used to convert the many square feature maps (512 (6,6) squares) into a long vector of neurons (512*6*6 neurons in total), as the input of the FC layers. The last two FC layers convert the flattened image features to 3 values, each of them corresponding to the confidence score of the input image being belonging to one of the 3 target classes.

Loss function
We use Cross-Entropy loss here as it is commonly used for multi-class classification problems. BCEloss is not used as our problem is not a multi-label classification problem.

Optimizer
We use Adaptive Moment Estimation (Adam) with default learning rate here as Adam is one of the best and commonly used optimizers in ML. We do not use RMSprop here as Adam adds bias-correction and momentum to RMSprop and in theory Adam should be better than RMSprop.

Experiments on Fine-tuning CNN2 model

Adjusting input size of image
We also increased the input size of our image to see if it can capture features better.

Adjusting number of sequential layers
The second CNN model is an attempt to increase the complexity of our model. In order to capture more features, we decided to increase the complexity of our model. Of course, we didn't just add one layer and said it is much better. We carried out numerous experiments on numbers of layers ranging from 2 to 10.

Adjusting dropout rate
Besides, we adjusted the dropout rate to see how disabling neutrons in training affects the results.

Experiment Results Analysis on Fine-tuning CNN2 model

Adjusting input size of image
Due to the increasing complexity of our model, we increased the input size of the image from 100*100 to 200*200 to capture more features like the splitting edge of cherry blossoms. The prototype using 100*100 gives an accuracy of around 40%. After increasing the input size, the accuracy rises to around 50%. It is believed that 200*200 captures more details while keeping the training time reasonable.

Adjusting number of sequential layers (less is more!!)
We tried adjusting the number of sequential layers ranging from 2 to 10. After evaluating the accuracy of them, we found out that 5 sequential layers is the sweet spot that yielded the highest overall accuracy. From the epochs of training, we noticed that when more and more sequential layers were added, the accuracy rose very slowly. From the experience of other models, we suspected that it was due to the vanishing gradient problem. As the number of layers increased, the complexity of the model increased, which resulted in poor performance of the model.

Therefore, CNN2 model consists of one more sequential layer (sequential layer 0) compared with the first CNN model (prototype).

Adjusting dropout rate
After conducting the research on commonly used dropout rate, we found that 0.1, 0.3, 0.5 are the dropout rate used by most of the models. We tried on these values and also tried on not using dropout. We found out that disabling 10% of neutrons in training achieves the best result. With 0.1 dropout, the accuracy is around 60-70%, while others give accuracy around and below 60%.

No dropout `Accuracy of the network on 60 images: 63.33 %`
0.1 dropout `Accuracy of the network on 60 images: 70.00 %`
0.3 dropout `Accuracy of the network on 60 images: 61.67 %`
0.5 dropout `Accuracy of the network on 60 images: 55.00 %`

We suspect that due to the small dataset size and low complexity model, we lose some important features during training if the dropout rate is set too high. Dropout of 0.1 is just enough to prevent co-adaptation of feature detectors and not lose too many features. Therefore, the second CNN model used 0.1 as the dropout rate.

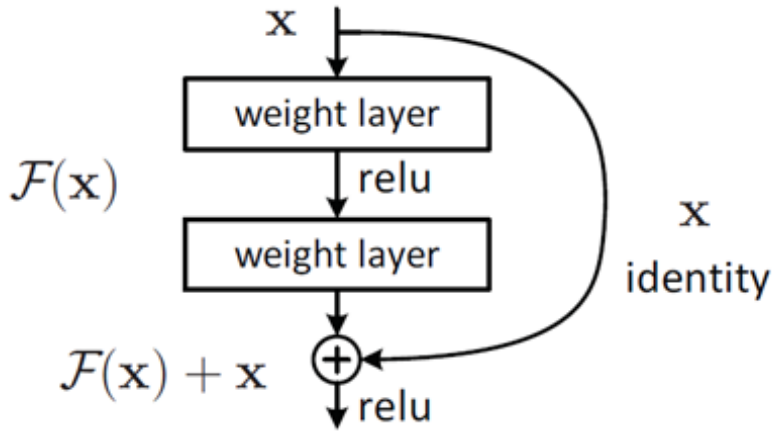
Experiments on Transfer Learning

Other pre-trained models are used for testing and analyzing results.

ResNet
The first network model we applied was ResNet34. ResNet34 is a network model that is able to solve the vanishing and exploding gradient problem. When the network model is quite complex, the update in weight will be very small. I.e. 0< wi < 1, wi*wi+1*wi+2*... Through applying the identity function, the problem of vanishing gradient becomes small.

From the experience of the ResNet34 network model, we know that a complex network model might experience a vanishing gradient problem, vice versa. Therefore, we don't want to build an overcomplicated model which might experience a vanishing gradient problem, nor a too simple model which might experience an exploding gradient problem.

```
Accuracy of the network on 32 images: 78.12 %
Accuracy of the network on 60 images: 76.67 %
```



VGG19
The second network model we applied was VGG19, another classic CNN network model. VGG makes use of a series of convolution layers of kernel size 3x3 to replace convolution layers of kernel size 11x11, 7x7. VGG derives from Alexnet, in which different kernel sizes are used to capture various features better. The good thing about VGG is that the structure is very simple, all convolution layers have a kernel size of 3x3. By adding convolution layers to make the network deeper, the performance of the network can be improved.
However, the problem of VGG is that the computational cost is high and the vanishing gradient problem is severe. What we can learn from this is that we don't really need to add convolution layers of different kernel sizes but still obtain the desired result.

```
Accuracy of the network on 32 images: 75.00 %
Accuracy of the network on 60 images: 76.67 %
```

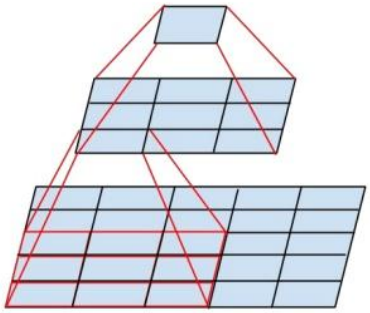
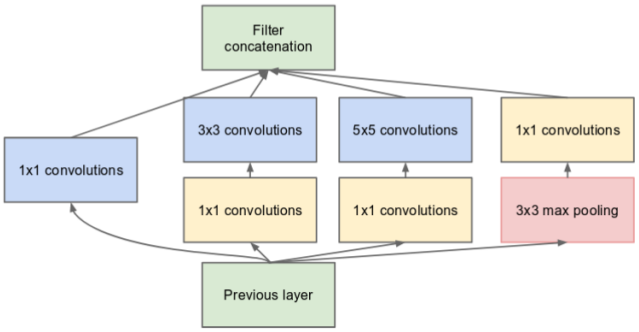


Figure 1. Mini-network replacing the 5×5 convolutions.

InceptionV3

The third network model we applied was InceptionV3. The InceptionV3 network model is the most complicated model we have ever referenced. The model aims not only at the depth but the width of the model. It concatenates layers of different kernel sizes(3x3, 5x5, 7x7) to the output layer, which can capture more features than just by using one big filter (11x11). InceptionV3 also makes use of the residual network's concept(ResNet) to deal with the vanishing gradient model. InceptionV3 is a powerful network structure, but it requires a large amount of training data to obtain a good result. The complexity of the model is too high for our task.

```
Accuracy of the network on 32 images: 56.25 %
Accuracy of the network on 60 images: 58.33 %
```



(b) Inception module with dimension reductions

Conclusion on transfer learning

After transfer learning of ResNet34, VGG19 and InceptionV3, we designed to modify our model using the VGG19 network structure. VGG19 in theory can capture better features than our original model because the model is deeper and resembles larger size filters(5x5, 7x7). Other models are too complex for our tasks and the result is even poorer if we applied them.

Experiments on Third CNN Model (CNN3)

The third CNN model is an attempt to resemble the VGG19 network. The benefit of VGG19 is shown above. It can theoretically capture more features but not increase the complexity of the model to a very high extent. As it uses 2 3x3 kernel size filters to resemble 1 5x5 filter and 3 3x3 kernel size filters to resemble 1 7x7 filter, which further improves the network depth. We expect the performance of the third CNN model will be better than the second CNN model and the first CNN model.

Experiment Results Analysis on Third CNN Model (CNN3)

Our first attempt is to add 1 more Conv2d layer after the initial Conv2d layer in the first two sequential layers, then add 2 more Conv2d layers after the initial Conv2d layer in the last three sequential layers. The model structure now becomes 2 sequential layers with kernel size of 5x5, 3 sequential layers with kernel size of 7x7. With the increased number of convolution layers, we expect that the test accuracy will be higher. However, the test accuracy only yielded around 40-50%.

Then our second attempt is to reduce the number of Conv2d layers in the third CNN model. The first two sequential layers have only 1 Conv2d layer and the last three layers have 2 Conv2d layers. With the decreased number of convolution layers, the test accuracy is higher, which yielded around 50%-60%.

The last attempt is to remove all added Conv2d layers after the initial Conv2d layer in all sequential layers, which is basically the second CNN model. The test accuracy rose back to 60%-70%.

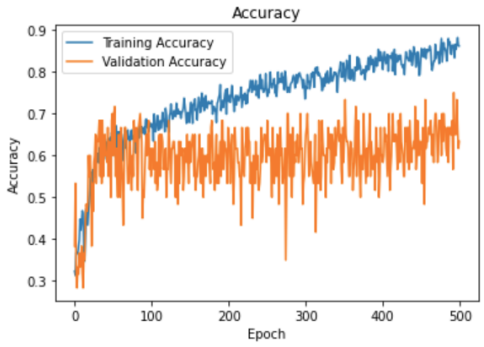
Conclusion on choosing CNN2 or CNN3

In this project, we have tried to improve our prototype by fine tuning the hyperparameters, which gives our second CNN model.

Then, we applied transfer learning and tried to further improve our model, which gives our third CNN model. However, the result is even worse than our second CNN model. Here is the analysis of why the third CNN model performs worse than our second model.

Vanishing Gradient Problem
Vanishing gradient problem is an important problem when dealing with complex models. As mentioned in the ResNet section, when the model is complex, the update in weights becomes very insignificant each time. Hence the learning rate of the model is very slow. A very large number of epochs is required to get high training accuracy (not test accuracy). For example, 100 epochs are sufficient for the second CNN model to have training accuracy of 85% and yield test accuracy of 70%, but over 500 epochs is needed to have training accuracy of 85% and only

yield test accuracy of 56.66%. It shows that the third CNN model is too complex for our project and suffered from a serious vanishing gradient problem as the update is very slow.



Overfitting Problem
We know that the third CNN model is a complex model, it requires more time on training, so we tried training the model using 700 epochs. The training accuracy yielded nearly 100% upon training completion. However, the test accuracy is only 60%. These two extreme figures showed that a serious overfitting problem occurred. The model focuses very much on the training dataset, the dev accuracy only yields around 60% throughout the entire training process. Hence, we found that the third CNN model is worse than the second CNN model in terms of overfitting.

Over-complex model
When we do research on ResNet, VGG19 and InceptionV3, we know that these models can classify over 1000 types of objects, that is very impressive. However, the supporting ground of these models are very big datasets. However, due to time constraints, we can only gather 618 images for training, dev and test set. It is a very small dataset compared to the datasets used in the pre-trained model of ResNet, VGG19 and InceptionV3. The CNN3 model we developed based on VGG19 is believed to be too complex for such a small dataset, causing low test accuracy.

In conclusion, we decided to use **the second CNN model** as it actually yields better results compared to the third CNN model.

Model Evaluation

After choosing CNN2, we train the model for 100 epochs and evaluate the model performance with our test dataset which contains 60 images. The accuracy is around 60-70%.

```
Accuracy of the network on 32 images: 75.00 %
Accuracy of the network on 60 images: 70.00 %
```

The two graphs below are the trend of loss and accuracy during the 100 epochs. It can be shown that the two graphs are converged. Although the training accuracy is still higher than the validation accuracy, the gap between is much smaller than the prototype. It is believed that the overfitting problem is relieved.

Limitations and possible improvements

Size of dataset
The size of our dataset with 618 images may not be enough, which makes the performance of our model less satisfactory. However, we have already gathered 2000+ images labelled with the three classes on Pixabay, which is almost all it can provide. After filtering mislabelled images manually, only around 600 images are correctly labelled on Pixabay. Therefore, extra sources like flickr can be added such that more data images can be used to provide more training data to the model and provide more variation to the dataset.

Mislabelled Data
Although we have tried our best to perform image filtering and checking after we have gathered the source images from the internet source, it is not guaranteed to be 100% correct. Afterall, we are not botanists and we lack the domain knowledge of flowers and plants. So there may be the existence of mislabelled data images, with an incorrect class label and hence misleading the prediction and classification of our model.

Dataset bias
Not all the species of the three types of flowers are included in the dataset images. For example, there are more than 20 species of Cherry Blossom, and our dataset may not contain all of the flower species. It is believed that some species are probably missed, making the model difficult to classify such species and worsened the performance.
Also, most of the images gathered are close-up photos of blossoms. In other words, if the photo is taken at a further distance, our model may not be able to correctly classify it. This issue arises as we cannot manually distinguish the type of flower if it was shot far away from the blossoms, so we failed to include such images into our dataset. It is believed that if a botanist with the domain knowledge does the data collection part, such images can be included in the dataset.

User experience

For the user interface, currently we are using the colab notebook as the user interface, users need to first upload all the photos to be predicted into the /images directory and then follow instructions to run the code blocks in the notebook to produce the result, which is not convenient for users in terms of the user experience. Better improvements will be building a better graphic user interface for the users to input images and deploy the model to the web application, without the need of putting the images to the directory folder and running the notebook manually.

Time limitation

Due to the limited project duration, we do not have enough time and resources to perform and try out all the possible stages of a machine learning project, and methodology of implementing models. For example, there is not enough time for us to acquire domain knowledge for better data images labelling, and the task of gathering more usable data images from other image sources, which is a time consuming task, as well as data cleaning,retraining, trying more combinations of hyperparameters and seeking for extra models for transfer learning.

Reference

Amusi (2018, August 6). 一文读懂VGG网络. Retrieved from https://zhuanlan.zhihu.com/p/41423739

Choi, Y.K. (2021). Module 3 - Classification and Object Detection of Images[jupyter notebook]. COMP3359, The University of Hong Kong, Hong Kong

Gombru, R (2018, May 23). Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. Retrieved from https://gombru.github.io/2018/05/23/cross_entropy_loss/

Inception Module. (n.d.). Retrieved from https://deepai.org/machine-learning-glossary-and-terms/inception-module

Inkawhich, N. (2017). Finetuning Torchvision Models. Retrieved from https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

Jajoo, N. (2021, March 11). Flower_Classification_Pytorch. Retrieved from https://www.kaggle.com/ach048/flower-classification-pytorch

Keras Applications. (n.d.). Retrieved from https://keras.io/api/applications/

Renewang. (2018, October 25). 10. 深度學習甜點系列：全面啟動. Retrieved from https://ithelp.ithome.com.tw/articles/10205210

ResNet and ResNetV2. (n.d.). Retrieved from https://keras.io/api/applications/resnet/

Ruder, S (2016, January 19). An overview of gradient descent optimization algorithms. Retrieved from https://ruder.io/optimizing-gradient-descent/

Simonyan, K. (2015, April 10). Retrieved from https://arxiv.org/abs/1409.1556

VGG16 and VGG19. (n.d.). Retrieved from https://keras.io/api/applications/vgg/