Machine Learning Techniques Final Project

HE-ZHE, LIN, b07902028
MING-HSUAN, TSAI, b07902064
CHI-FENG, TSAI, b07902123

1 CONVOLUTIONAL NEURAL NETWORK

1.1 Environment and Packages

- Programming environment: *Python 3.8.3*
- Packages: *numpy, torch, cv2*

1.2 Data Processing

We followed the tradition established by some famous model architectures, e.g. VGG, ResNet, and resized the images to 224×224. We chose this size only for simplicity. In fact, even if the images were not square, we could still adjust the model to handle different sizes. As for image size, while a larger size keeps more details of an image, it requires more memory and time to process. We have attempted several sizes, e.g. 192×192, 224×224, etc, and it turned out that 224×224 gave slightly better result while keeping the process time short.

1.2.1 Data Augmentation. According to personal experiences and various materials, data augmentation plays an important role in the performance of a model. Combined with our judgement and experiment results, we adopted the following methods for data augmentation.

- `transforms.CenterCrop(224)`
- `transforms.RandomVerticalFlip()`
- `transforms.RandomHorizontalFlip()`
- `transforms.RandomRotation(30)`
- `transforms.RandomPerspective()`
- `transforms.ColorJitter((0.8, 1.2), (0.8, 1.2), (0.8, 1.2))`
- `transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.256, 0.225])`

Although we mentioned previously that we resized images to 224×224, it was not the full story. To be precise, we noticed that in almost every image, the mango was placed in the center. So when we read in the images, they were initially resized to 256×256 with *cv2*, and then cropped to 224×224 at the center. This way, the peripheral objects would be erased and thus the model could focus more on the mango itself.
We have conducted an experiment which showed that compared to resizing to 224×224 directly, the idea above resulted in slightly better performance.
Note that except for center cropping and normalizing, all other operations were performed on the training data only.

1.3 Model Architecture

An visualization of our model architecture that resulted in the best performance is shown in `Fig.1`. Detailed values of hyperparameters are shown in the following table.
For the sake of brevity, we use the following templates

- `nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)`
- `nn.Linear(in_features, out_features)`

To further save space, we ignore the `in_channels` *parameter and replace it with an underscore _. The appropriate value can be deduced from the previous layer.*

Architecture and Hyperparameters	
Block	Design
Input Image	`(length, width, channel) = (224, 224, 3)`

Conv 1/2	`nn.Conv2d(_, 32, 3, 1, 1)`
	`nn.PReLU(32)`
	`nn.BatchNorm2d(32)`

Conv 3/4	`nn.Conv2d(_, 64, 3, 1, 1)`
	`nn.PReLU(64)`
	`nn.BatchNorm2d(64)`

Conv 5/6	`nn.Conv2d(_, 128, 3, 1, 1)`
	`nn.PReLU(128)`
	`nn.BatchNorm2d(128)`

Conv 7/8	`nn.Conv2d(_, 256, 3, 1, 1)`
	`nn.PReLU(256)`
	`nn.BatchNorm2d(256)`

Conv 9/10	`nn.Conv2d(_, 512, 3, 1, 1)`
	`nn.PReLU(512)`
	`nn.BatchNorm2d(512)`

Dense 1	`nn.Linear(512·7·7, 512)`
	`nn.PReLU()`
	`nn.BatchNorm1d()`

Dense 2	`nn.Linear(512, 256)`
	`nn.PReLU()`
	`nn.BatchNorm1d()`

Dense 3	`nn.Linear(256, 64)`
	`nn.PReLU()`
	`nn.BatchNorm1d()`

Dense 4	`nn.Linear(64, 3)`

The design of this architecture is based on both personal experience with CNN models and trial and error. We have attempted dozens of architectures and hyperparameters values. However, we do not have space to enumerate all of them, so we focus on some major adjustments in the following section.

1.3.1 Discussion.

- Order of BatchNorm and ReLU
  Although some well-known architectures, e.g. VGG, put `BatchNorm` before `ReLU`, we believe `BatchNorm` behind `ReLU` makes more sense. We conducted an experiment to test our idea.
  In the following table, we use `B` for `BatchNorm`, and `R` for `ReLU`.

Order	Accuracy on validation set
B → R	83.125%
R → B	**84.125%**

  Just as we normalize the images before they are fed into the first block, we believe that putting `BatchNorm` after `ReLU`, which normalizes the output of a block (which is also the input of the next block), can result in better performance.
- ReLU, LeakyReLU, PReLU
  We found no mathematical analysis that guarantees any of them is always a better choice, so we set up experiments and adopted the one that gave the best performance.

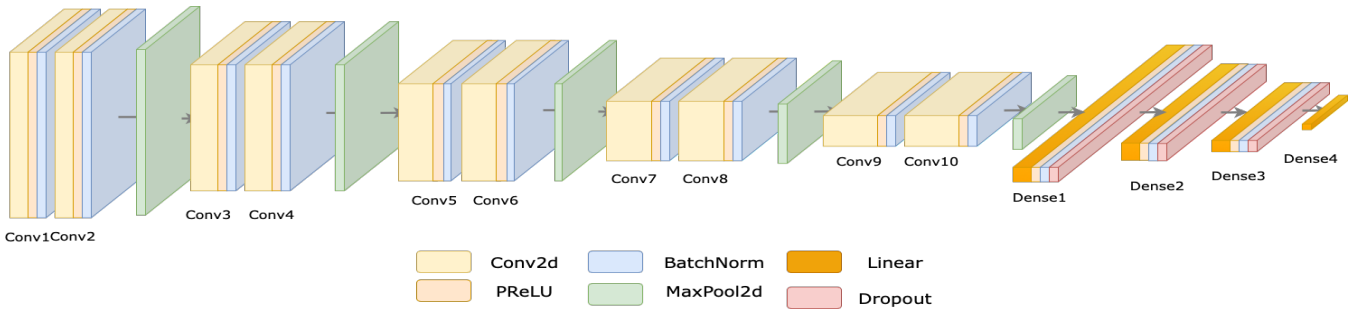


Fig. 1. Visualization of model architecture.

Activation function	Accuracy on validation set
ReLU	83.25%
LeakyReLU	83.125%
PReLU	**84.125%**

- Dropout

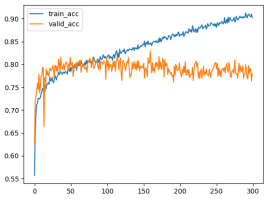


Fig. 2. Acc without dropout

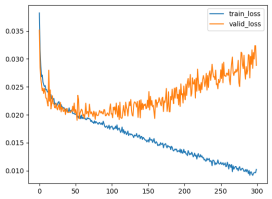


Fig. 3. Loss without dropout

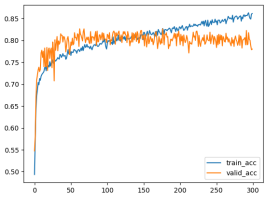


Fig. 4. Acc with dropout p=0.5

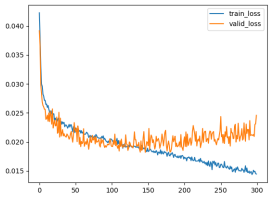


Fig. 5. Loss with dropout p=0.5

We have tested dropout from p=0.1 to 0.5, but we do not have enough space to contain all the figures. However, we see that with dropout, overfitting is less severe. Finally, we chose dropout with p=0.5 in our model.

- Pooling
  We thought that instead of using max pooling, applying another convolutional layer with `stride=2` might give the model more power and diversity. However, the result did not support our idea.

Pooling	Accuracy on validation set
`MaxPool2d`	**84.125%**
`Conv2d with stride=2`	83.5%

- Optimizer
  In a classification problem, the performance of SGD and Adam has been widely discussed. We conducted experiments and the results are shown in `Fig.6` and `Fig.7`. We see that SGD outperformed Adam in this task, especially in latter phase of training, so we adopted SGD to train our model.
- Others
  The following adjustments only resulted in negligible difference of performance.

  – Stride of convolutional layers: We have tried `stride=3`, `stride=5`, `stride=7`. It turned out, albeit not significant, `stride=3` resulted in better performance on validation set.

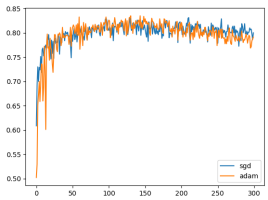


Fig. 6. Acc on val set

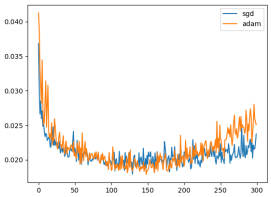


Fig. 7. Loss on val set

  – Number of fully connected layers: We have attempted different combinations of number of layers and number of neurons, but none of them gave significant improvement compared to other configurations. We could obtain similar performance with only one layer of 512 neurons.

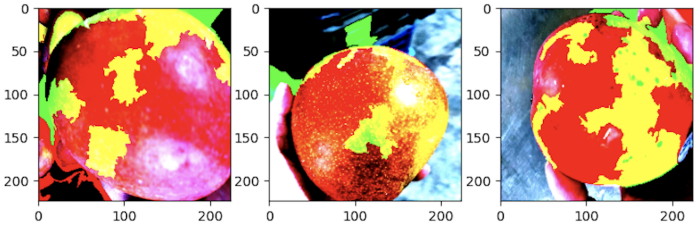
1.4 Performance

After devoting great amount of time to adjusting the model architecture and training methods, our final model achieved the following result.

Data	Accuracy
Validation set	84.125%
Testing set	79.6875%

To further improve the performance, we trained the same model for five times, but each time with different training set and validation set. We let the five models vote for the final answer, i.e. uniform blending, and achieved 80.0625% on testing set.

1.5 Visualization



To further understand what the model was doing underneath the hood, we use the technique of **Explainable AI**, and use the package *lime* to visualize how the model classified an image.
The images from left to right are of category A, B, C, respectively. The parts colored green are the parts that are helpful to the model for correct predictions. We can see that the model focused mostly on the black dots on the mangoes, as well as their edges.

2 SUPPORT VECTOR MACHINE WITH FEATURE EXTRACTION

2.1 Environment and Packages

- Programming environment: *Python 3.8.3*
- Packages: *sklearn, numpy, PIL, skimage*

2.2 Overview

`Fig.8` outlines the final version of our model based on PCA and SVM. We would describe the methods trialed in each process in the following subsections.

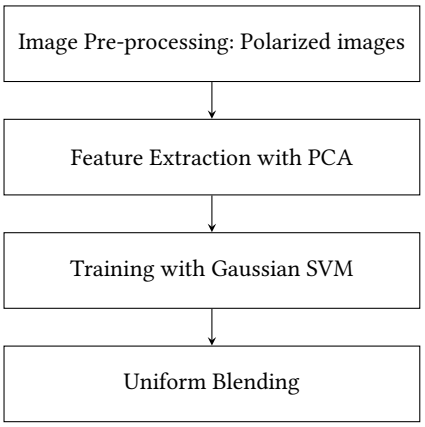


Fig. 8. The flow chart of the model based on PCA and SVM

2.3 Image Preprocessing

For simplicity, we resized the image to 192×192. After scanning the images, we found that there are lots of noises, such as hands of the photographer, or other mangoes in the baskets. We tried to remove the background at first but found it challenging, since edge detection is hard to achieve, especially when there are other mangoes overlapping with the target.
Instead, we transformed the image to a polarized one, as shown in `Fig.9` and `Fig.10`. We see that the mangoes were mostly transformed to the upper half in the polarized image, since they're mostly located in the middle of the original one. Hence, we cropped the lower-half image and flattened it to an 1D numpy array with length $96 \times 192 \times 3$.

Fig. 9. Origin Image

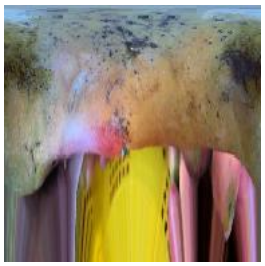


Fig. 10. Polarized Image

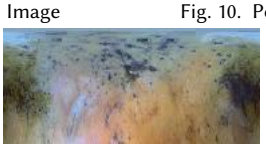


Fig. 11. Cropped Image

Below is the training result of PCA (`n_components = 200`) and Gaussian SVM(`C = 1.5, gamma = 'scale'`) using different pre-processing method.

Pre-processing	Accuracy on validation set
Original	68.25%
Polarized	**69.875%**
Cropped	64.875%

We see that polarized image performs slightly better than the original one. However, the difference is not significant. Thus, in the following sections, we use the original images to conduct experiments.

2.4 Feature Extraction

2.4.1 Principle Component Analysis. It's time-consuming to put over 10^5 dimensional data into SVM. Clearly, dimension reduction is needed. We first use PCA introduced in class to extract the features in each image.

- `n_component`
  Under Gaussian kernel SVM, we used `GridSearchCV` to find optimal $C \in \{1, 1.5, 2\}$ and $\gamma \in \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$. (The parameter is based on the experiment described in Section 2.5 under the scheme of the approach.)

num_comp	best C	best γ	Accu on validation set
50	2	10^{-5}	68.000%
100	2	10^{-5}	**69.250%**
200	2	10^{-5}	69.000%
500	2	10^{-5}	68.500%
700	2	10^{-5}	68.250%
800	2	10^{-5}	68.250%

- `whiten`
  According to personal experiences, in many practical uses of PCA, the parameter `whiten = True` helps make the input less redundant and thus more "independent" to one another. For the two choices, we search for the best `n_component` in PCA, C and γ in SVM. The result didn't agree with our expectation.

whiten	n_comp	C	γ	Accu on val set
True	200	2	5×10^{-3}	66.000%
False	100	2	10^{-5}	**69.250%**

Time complexiy analysis:
If there are n data with dimension d. PCA takes $O(d^2n + d^3)$. The good thing is, in `sklearn`, when `n_comp` is small , the randomized PCA, proposed by Halko et al only takes $O(n\tilde{d}^2 + \tilde{d}^3)$, where $\tilde{d}$ denotes the number of components after running PCA. Thus, using PCA as dimension reduction is rather efficient.

2.4.2 TSNE. TSNE is another well-known techniques when it comes to feature extraction. We've tried to decrease the data to a dimension of 2 or 3. After visualizing the data, we found that there's no clustering on the training set. Also, as a non-linear dimension decreasing method, TSNE also takes long time, so we would use PCA as our method to extract the feature.

2.5 Support Vector Machine

We first tried linear SVM for training. However, linear SVM didn't converge on training data, even if `num_of_iter` was set to 100,000. Thus, we turned to kernel SVM. We tuned the parameter to find the optimal (C, γ) with different `n_component`. Below is the experiments we've conducted with.

n_component	C	γ scale $\approx 10^{-5}$	Train acc	Valid acc
100	10^{-3}	scale	36.875%	37.000%
	10^{-2}	scale	37.428%	37.625%
	10^{-1}	scale	64.875%	63.25%
	1	scale	79.875%	69.000%
	10^{1}	scale	97.767%	67.375%
	1	10^{-6}	62.732%	63.625%
	1	3×10^{-6}	67.392%	65.625%
	1	10^{-5}	78.017%	**69.125%**
	1	3×10^{-5}	91.696%	68.125%
	1	10^{-4}	99.696%	52.875%
200	10^{-1}	scale	64.875%	63.000%
	1	scale	80.732%	69.625%
	10^{1}	scale	99.17%	67.625%
	1	10^{-6}	63.732%	63.375%
	1	3×10^{-6}	68.553%	65.875%
	1	10^{-5}	80.339%	**69.750%**
	1	3×10^{-5}	94.053%	67.875%
	1	10^{-4}	99.946%	47.500%
500	10^{-1}	scale	64.553%	62.500%
	1	scale	82.035%	**69.500%**
	10^{1}	scale	99.678%	66.500%
	1	3×10^{-6}	69.482%	66.125%
	1	10^{-5}	80.339%	69.375%
	1	3×10^{-5}	95.714%	68.250%
1000	10^{-1}	scale	64.375%	62.750%
	1	scale	83.125%	**69.500%**
	10^{1}	scale	99.875%	67.125%
	1	3×10^{-6}	70.196%	65.875%
	1	10^{-5}	83.750%	69.125%
	1	3×10^{-5}	96.446%	68.250%

One can see that the best accuracy on validation set is roughly from 66% to 69%. We found that the data that these SVMs made a mistake on are almost the same. Hence, further optimization like blending, hardly works, since the SVM's consensus was reached and no diversity was shown in the approach.

Time Complexity analysis:
According to Bottou amd Lin, the time complexity of non-linear SVM with n training data is roughly between n^2 (When n is small) to n^3 (When n is large). Thus, data augmentation is hard to be achieved.

2.6 Performance

The best performance of this approach is obtained by

- Use polarized image as data preprocessing
- Apply PCA with `n_component = 200` and `whiten = False`
- Set kernel SVM with `C = 1.5` and `gamma = 'scale'`

3 EXPERIMENTS ON RANDOM FOREST

3.1 Environment and Packages

- Programming environment: *Python 3.8.3*
- Packages: *sklearn, numpy, PIL*

3.2 Overview

We performed many experiments on the hyperparameters. We tried PCA to reduce the noise. We also tried using Random Forest Feature Importance to remove irrelevant features. Nevertheless, these techniques did not seem to improve the accuracy of our model. In the following sections, we will demonstrate the results of our experiments.

3.3 Image Preprocessing

We tried a few ways to preprocess the images.

- Resize image to $N \times N$ and use RGB value (0-255). The resulting number of feature is $3 \cdot N^2$.
- Add the horizontally flipped image to data set. This procedure will double the data set.
- Rotate the image (0°, 90°, 180°, 270°). This procedure will make data set 4-times larger.
- Use the polarized images mentioned in the previous SVM section.

To catch a glimpse the performance, we used these datasets to train a `sklearn.ensemble.RandomForestClassifier` with default parameters. We used `oob_score_` to evaluate the performance since we observed that `oob_score_` was numerically stable and close to validation error.

method	data size	# feature	`oob_score_`	training time
resize 128	6400	49152	0.6279	< 30 sec
resize 192	6400	110592	0.6249	< 1 min
resize 128 flip	12800	49152	0.6252	1 min
resize 128 rotate	25600	49152	0.6170	< 3 min
resize 128 rotate, flip	51200	49152	0.8065	4 min
polarized	6400	55296	0.6188	< 1 min

As we can see in the above table, the best `oob_score_` can be derived from `resize 128, rotate, flip`.

3.4 Tuning Hyperparameters

In this section, for those hyperparameters not explicitly mentioned, we will use its default value stated in *sklearn* documents. The data preprocessing is `resize 128` (for faster training) if not stated explicitly.

3.4.1 n_estimators. This hyperparameter represents the number of the trees in Random Forest.

`Fig.12` corresponds to the property of Random Forest. The more trees in a Random Forest, the higher the accuracy of the prediction. The increase of accuracy slows down after a threshold.

3.4.2 max_features. This hyperparameter represents the fraction of features to consider when looking for the best split.

As `Fig.13` shows, this hyperparameter does not seem to affect `oob_score_` much.

3.4.3 min_samples_leaf and min_samples_split. `min_samples_leaf` is the minimum number of samples required to be at a leaf node, whereas `min_samples_split` is the minimum number of samples required to split an internal node.

In `Fig.14` , the two hyperparameters acts similarly in a Random Forest. They have similar effect on `oob_score_` by their definitions, which corresponds to the result.

3.4.4 max_depth. This hyperparameter limit the maximum depth of a tree.

As shown in `Fig.15`, the smaller the tree depth, the worst the model perform since the tree is less powerful. If the tree has not yet reached `max_depth` but already fully grown, `max_depth` does not affect the model anymore.

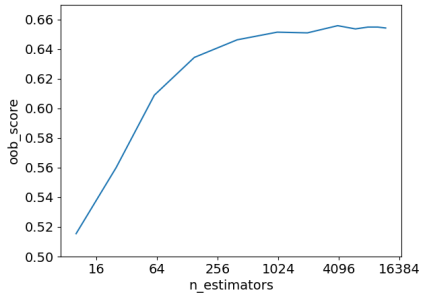


Fig. 12. The relation between n_estimators and oob_score_. The more the trees in Random Forest, the better the performance.

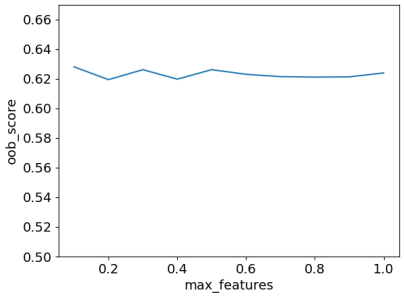


Fig. 13. The relation between max_features and oob_score_. max_features does not seem to affect the performance.

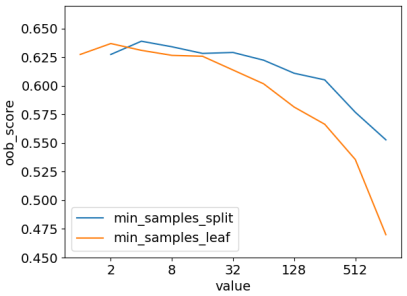


Fig. 14. min_samples_leaf and min_samples_split

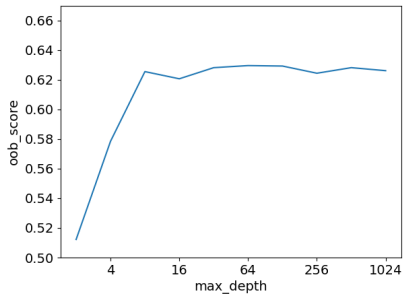


Fig. 15. The relation between max_depth and oob_score_

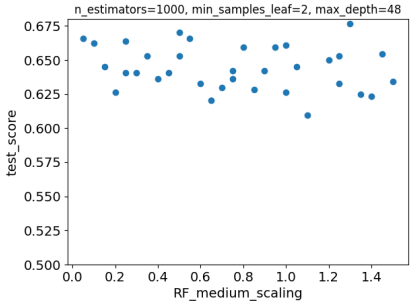


Fig. 16. The relation between scaling and test_score

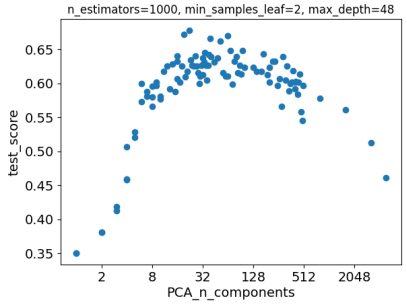


Fig. 17. The relation betweeen n_components and test_score

3.5 Feature Selection / Dimension Reduction

3.5.1 Feature Selection. We use the impurity-based feature importance of a Random Forest to remove irrelevant features. We call `sklearn.feature_selection.SelectFromModel` with a `RandomForestClassifier(n_estimators=400, max_depth=64)` and `threshold=f'{scaling}*median'`, where `scaling` is the hyperparameter we are going to tune. We use training data to train the feature selection model, transform the training data, and use the transformed training data to train the prediction model. We use RandomForestClassifier(n_estimators=1000, min_samples_leaf=2, max_depth=48) as our prediction model. We use transformed testing data for evaluation.

As `Fig.16` shows, our feature selection does not seem to improve the Random Forest in this problem. The result is averagely no better than that of the model without feature selection.

3.5.2 Dimension Reduction. We use `sklearn.decomposition.PCA` to try to reduce the noise in the data. The parameter `n_components` is the target we are going to tune. We use PCA to transform our training data, and use the transformed training data to train a RandomForestClassifier(n_estimators=1000, min_samples_leaf=2, max_depth=48). We use transformed testing data to evaluate our model.

As shown in `Fig.17`, the `test_score` is bad when we use too few or too many components. Note that the variance of this model is relatively high.

3.6 Performance

The model with best `oob_score_` is simply RandomForestClassifier(n_estimators=1000) trained by `resize 128, rotate, flip` data. This model takes nearly 30 minutes to train on a CSIE workstation.

Score Type	Value
Training Data Accuracy	1.0
OOB Score	0.8235
Testing Data Accuracy	0.61

3.7 Discussion

- Why is `oob_score_` of `resize 128, rotate, flip` outnumber that of the other preprocessing methods?

Using `resize 128, rotate, flip` data preprocessing will lead to a 8-times larger data set. Every piece of data in this data set has 7 other siblings that is similar to itself. Therefore, even though a data is not sampled for current Decision Tree, it could still be well predicted if current Decision Tree is trained by its siblings.
Let's recall that the probability of one data becoming OOB for a Decision Tree is $\frac{1}{e} \approx 0.368$. Consider `resize 128, flip` and `resize 128, rotate` data preprocessing, which leads to a twice and 4-times large data set, respectively. For each OOB data, the probability of current Decision Tree being trained by the data's siblings is quite small. Even if the Decision Tree is trained by its siblings, the similarity between the data and its siblings may not be significant enough. Therefore, the OOB score is not affected much when using these data preprocessing techniques.
Nonetheless, with `resize 128, rotate, flip` data preprocessing, each piece of data has 7 siblings in the data set. The chance of the current Decision Tree being trained by its siblings increases, which means the OOB data being correctly predicted becomes much more likely to happen. As a result, we can see significant improvement in OOB Score when using this 8-times large data set.
We should also note that the testing data accuracy in `resize 128, rotate, flip` is not improved at all. The reason is that testing data does not have any sibling in training data. To our Random Forest, testing data is a brand new data set. Hence, it will have no advantage on predicting testing data.

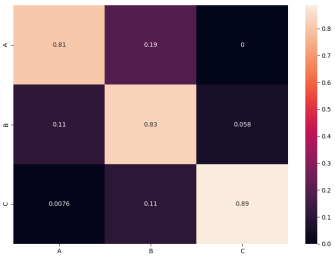


Fig. 18. CNN

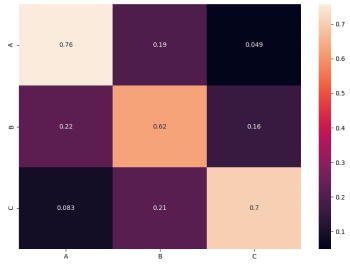


Fig. 19. SVM

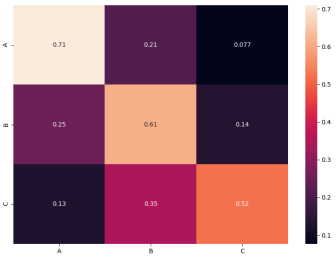


Fig. 20. Random Forest

4 COMPARISON

4.1 Confusion Matrices

`Fig.18` to `Fig.20` are the confusion matrices. The x-axis is the prediction, whereas the y-axis is the true label. The figures show that a mango is usually misclassified as its adjacent rank. For example, a level-A mango is often mistaken for a level-B one, but is rarely misjudged as a level-C one. This is especially true for CNN, where no mangoes of level-A were mistaken for level-C.

4.2 Efficiency

The training time for the three models on a CSIE workstation are listed as follows

- CNN: roughly 4 hours
- SVM: 10 minutes
- Random Forest: 4 minutes (*)

In terms of efficiency, CNN takes the longest time to converge, while Random Forest takes the least.
(*) We do not use the model of best `oob_score_`, which is 30 minutes, to evaluate the efficiency.

4.3 Scalability

Generally, big data size prevents a model from overfitting. However, the second approach isn't suitable for a large dataset, since the running time of PCA and SVM are both highly dependent on the number of data, as mentioned in Section 2.4.1 and 2.5. For CNN, a large data size usually improves the performance, as long as there is no too much noise. When training a CNN model, we usually make use of GPUs to greatly improve the training speed. Also, we can specify `batch_size` during training. While a larger batch size usually results in faster running time, a smaller batch size involves more update of parameters, and thus often leads to a better result. It is the designer's job to find the sweet spot between training time and the quality of model. Random forest also has good scalability due to its bootstrapping techniques.

4.4 Popularity

CNN for image classification is very common. There are many well-known CNN architectures, such as VGG, ResNet, AlexNet, and new architectures are being studied.
SVM and Random Forest are classical yet powerful learning models. Though it is not designed for image-based learning, there are still some cases in which these models can be applied.

4.5 Interpretability

- CNN: When a neural network gets deep, it is hard to tell exactly which function a layer or a neuron is performing. This is true for CNN as well. However, with the techniques of **Explainable AI**, we are able to visualize how an image is processed at each layer, greatly improve the interpretability of the model.
- SVM: SVM itself is easy to explain, i.e, find a hyperplane to separate data. However, the training features in SVM are only principal components of image pixels, which is meaningfulness to human beings.
- Random Forest: Tree models can be easily visualized due to its nature. Nevertheless, the features are pixels with RGB values in image classification problems. Trees are less meaningful when split by a single pixel.

4.6 Conclusion

Based on the accuracy of the three models, our final recommendation is **CNN**, as it delivers significant better performance over the other two models.
The major benefit of using CNN is that it is by nature suitable for image classification, and it can be easily extended to the situation with hundreds or thousands of categories and still produce excellent result. Also, CNN is a popular technique that is widely studied by many people. New architectures and techniques are being proposed and it has more potential to be explored.
However, CNN usually requires more advanced hardware, e.g. GPUs, to be useful. And even with the support of hardware, it can easily take hours or even days to produce the result. Besides the requirement for computing resources, special domain knowledge is also required to fine-tune a CNN model. A CNN model can easily get deep and complicated. Without a solid understanding of the model, it is hard to keep track of what is going on in each layer.

5 LOAD BALANCE

- Section 1 (CNN): b07902064
- Section 2 (SVM): b07902028
- Section 3 (Random Forest): b07902123
- Section 4, 5, 6: b07902028/b07902064/b07902123

6 REFERENCES

[1]: PyTorch official document
[2]: Experiment on order of ReLU and BatchNorm
[3]: Comparison of SGD and Adam
[4]: Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853.
[5]: Keskar, N. S., & Socher, R. (2017). Improving generalization performance by switching from adam to sgd. arXiv preprint arXiv:1712.07628.
[6]: scikit-learn API Reference
[7]: Luuk Derksen. *Visualising high-dimensional datasets using PCA and t-SNE in Python*
[8]: Sharon Morris. *Image Classification Using SVM*, 2018
[9]: Leon Bottou, Chih-Jen Lin. *Support Vector Machine Solvers*, 2007
[10]: Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836.