In this section, we would try transfer learning to train models based on AlexNet and GoogLeNet, performing a 5-class, or 10-class in cases, fruits classification, including durian, mango, papaya, kiwifruit and mangosteen.

## Data Collection

In this section, we will provide a brief overview of our data collection and processing methods, including the structure of the datasets we have created.

We have collected 15 images of each whole fruit and 15 more images of each fruit when it is cut. We created tried 4 different datasets for them. In the first dataset, there are only five classes, with all pictures are whole fruits. (closed dataset) There are also 5 classes in the second set, with all images featuring cut fruits. (open dataset) The third one contains 10 classes instead, with 5 classes for the whole fruits and the others for the opened fruits. (mixed-10 dataset) The last set contains only 5 classes, each representing a different fruit, while the images in each class include both whole and cut versions of the fruit. (mixed-5 dataset)

Since most fruits look different when cut and whole, we establish the four datasets to test whether combining different forms of the same fruit will significantly impact the performance of the model during prediction.

In addition, all pictures are resized to 224x224x3 in this step, although we would also perform picture auto-resizing during the training.

## Training

In this section, we will introduce the code used for transfer learning in this task.

Performing transfer learning in MATLAB is straightforward, as the code for training process is the same with the choice of different models.

First, we defined some hyper-parameters. This this section, we would use a learning rate of 1e-4 and train the model for 6 epochs, with a batch size of 10. As shown in the code below.

1. %% hyper-parameters >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

2. datasetPath = "Dataset";

3. % choose from "Alex" and "Google"

4. backbone = "Alex";

5. lr = 1e-4;

6. n\_epoch = 6;

7. batch\_size = 10;

Then the specified dataset is loaded, in which 70% of data is randomly picked to form the training set.

1. %% load dataset >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

2. imds = imageDatastore(datasetPath,"IncludeSubfolders",true,"LabelSource","foldernames");

3. [imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');

After the number of classes is obtained, the specified pre-trained network is loaded, with its output size configured to be the number of classes classified. The input size of the network is also acquired, which will be useful to perform the image auto-resizing in the next steps.

1. %% get classes for training set >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

2. classNames = categories(imdsTrain.Labels); % get the names of classes

3. numClasses = numel(classNames); % get the number of classes

4.

5. %% load pre-trained network >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

7. if backbone == "Google" % load GoogLeNet

8. net = imagePretrainedNetwork("googlenet",NumClasses=numClasses);

9. disp("GoogLeNet for transfer backbone.")

10. elseif backbone == "Alex" % load AlexNet

11. net = imagePretrainedNetwork("alexnet",NumClasses=numClasses);

12. disp("AlexNet for transfer backbone.")

13. end

14.

15. %% get the input size of network

16. inputSize = net.Layers(1).InputSize;

17.

18. %% open the visual analyzer

19. analyzeNetwork(net)

Data augmentation is also performed to avoid overfitting, as the pre-trained model is large while our dataset is quite small.

This operation is optional, and we would perform a test to examine the performance of the model with and without data augmentation later.

1. %% apply data augmentation to avoid overfitting

2. randomPixelRange = [-30 30];

3. imageAugmenter = imageDataAugmenter( ...

4. 'RandXReflection',true, ...

5. 'RandXTranslation',randomPixelRange, ...

6. 'RandYTranslation',randomPixelRange);

After resizing the images and applying the imageAugmenter we have defined before, the train is about to start with proper training options and loss function. We chose cross entropy loss as the model is expected to perform a classification task.

1. %% resize training images

2. augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...

3. 'DataAugmentation',imageAugmenter);

4.

5. %% auto-resize validation images

6. augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);

7.

8. %% training options

9. options = trainingOptions("sgdm", ...

10. MiniBatchSize=batch\_size, ...

11. MaxEpochs=n\_epoch, ...

12. Metrics="accuracy", ...

13. InitialLearnRate=lr, ...

14. Shuffle="every-epoch", ...

15. ValidationData=augimdsValidation, ...

16. ValidationFrequency=3, ...

17. Verbose=false, ...

18. Plots="training-progress");

19.

20. %% train net with cross entropy loss

21. net = trainnet(augimdsTrain,net,"crossentropy",options);

After the training, we collected the results and tested the model with four randomly picked images.

1. %% get four predictions and show them up

2. scores = minibatchpredict(net,augimdsValidation);

3. YPred = scores2label(scores,classNames);

4.

5. idx = randperm(numel(imdsValidation.Files),4);

6. figure

7. for i = 1:4

8. subplot(2,2,i)

9. I = readimage(imdsValidation,idx(i));

10. imshow(I)

11. label = YPred(idx(i));

12. title(string(label));

13. end

## Results Discussion and Comparison

In this section, we focus on examining the model performance with different models, datasets and training parameters. Tests examining the influences introduced by different training strategies would not be taken in this task since there have been detailed discussions on them in task 1. All tests in this task use SGDM as their training method, unless been specified.

### Comparison of AlexNet and GoogLeNet

In order to examine the difference of performance between different models, we trained both AlexNet and GooLeNet on the mix dataset with a learning rate of 1e-4 and batch size of 10 for 15 epochs. We tested the two models both with and without the data augmentation process. The results of both tests are plotted, as shown in Figure 3.1 and Figure 3.2.

In the test where the data augmentation is not applied, both AlexNet and GooLeNet levels at an accuracy of nearly 100%. However, AlexNet converges to only 95% if the data augmentation process is taken whereas GoogLeNet remains at an accuracy of 100%. Additionally, AlexNet converges faster than GoogLeNet at the beginning in both tests.

Both tests suggest that AlexNet has a faster convergence speed at the beginning, which should be attributed to its simpler and more straightforward model structure, while GoogLeNet has a better ability of generalization due to its inception modules, leading to a higher accuracy at the end.

图表, 折线图

描述已自动生成

Figure 3.1: AlexNet Accuracy Curve with Data Augmentation

图形用户界面, 图表, 折线图

描述已自动生成

Figure 3.2: GoogLeNet Accuracy Curve with Data Augmentation

### Training with Different Datasets

As previously mentioned, we created 4 datasets for this task, which are called closed, open, mixed-5 and mixed-10 respectively. After examination, we found that the models perform similarly on both the closed and open datasets. This is because both datasets consist of 5 classes, making them essentially almost equivalent in terms of their structure. Although the mixed-10 dataset contains 5 additional classes, the only difference in training on it compared to the closed and open datasets is a slightly slower convergence speed. However, both AlexNet and GoogLeNet converges much slower on the mix-5 dataset, likely due to its increased complexity. In fact, AlexNet struggles to reach an 100% accuracy on this dataset when the data augmentation is applied during further testing.

### Training with and without Data Augmentation

For this test, we chose AlexNet with a learning rate of 1e-4 and a batch size of 10, training for 10 epochs each, using the mixed-10 dataset.

The results are plotted below in Figure 3.3 and Figure 3.4. As shown in the graphs, the validation accuracy levels at 95% without a data augmentation process. However, prediction accuracy reaches nearly 100% if data augmentation is applied.

However, this difference is not as evident in other datasets, which may be due to the higher complexity of this specified dataset compared to the others. This suggests that data augmentation is more crucial in complex and varied tasks and datasets, especially when the amount of data is limited. By contrast, it is not necessary to apply data augmentation in simpler datasets or tasks, or when the amount of data is sufficient, as data augmentation itself can be time-consuming. 图表, 折线图

描述已自动生成

Figure 3.3: Accuracy curve, training with data augmentation

图表, 折线图

描述已自动生成

Figure 3.4: Accuracy curve, training without data augmentation

### Training with Different Learning Rates

The choice of learning rate would largely affect the training results.

On the one hand, the training process would be quite time-consuming, and the model may stick within some local minimum during the process if the learning rate is too small. On the other hand, a large learning rate may lead to model overfitting. Additionally, the model may struggle to converge to optimal results, as it would oscillate around the minimum.

In this task, we examined the influence of different learning rates on the performance of the model by training GoogLeNet on the close dataset with a batch size of 10 for 6 epochs each, without data augmentation. We trained the model with three different learning rates, 1e-3, 1e-4 and 1e-5 respectively. The plots are shown below in Figure 3.5, Figure 3.6 and Figure 3.7.

It can be seen that the prediction accuracy increased pretty slow when the model was training with a learning rate of 1e-5, reaching 85% at the end of the 6th epoch. On the other hand, although the validation accuracy rocketed to 100% at the beginning of training with a learning rate of 1e-3, it then dropped to 95% later, with the training accuracy remaining at a high level, indicating that it may meet some overfitting issue. Finally, the model converges to nearly 100% accuracy at the end of 6th epoch with a learning rate of 1e-4, which should be the optimal choice in this case.

图表, 折线图

描述已自动生成

Figure 3.5: Accuracy Curve with a learning rate of 1e-5

图表, 折线图

描述已自动生成

Figure 3.6: Accuracy Curve with a learning rate of 1e-4

图表, 折线图

描述已自动生成

Figure 3.7: Accuracy Curve with a learning rate of 1e-3

### Training with Different Numbers of Epochs

The number of epochs is also quite important to the training. Models may fail to achieve their best performance if the number of training epochs is not enough and may also meet overfitting issues if the number of training epochs is too large, as shown in Figure 3.7, where accuracy first increases and then decreases during the training.

### Training with Different Batch Sizes

The batch size of data is also a hyper-parameter to be controlled. On the one hand, training with a small batch size can be time-consuming, especially when using a GPU, compared to using a large batch size. Additionally, a small batch size may lead to a fluctuating learning curve, reducing training efficiency and making the model's training unstable. On the other hand, training with a large batch size can sometimes result in excessive stability, when the learning parameters of the model become stuck in some local minima and thus fail to find the optimal solution.

We trained the GoogLeNet on the mixed-5 dataset with a learning rate of 1e-4 for 5 epochs with data augmentation, with batch sizes of 1 and 20 respectively. RMSProp is used as the training strategy to make the difference more clearly. The resultant accuracy plots are shown in Figure 3.8 and Figure 3.9.

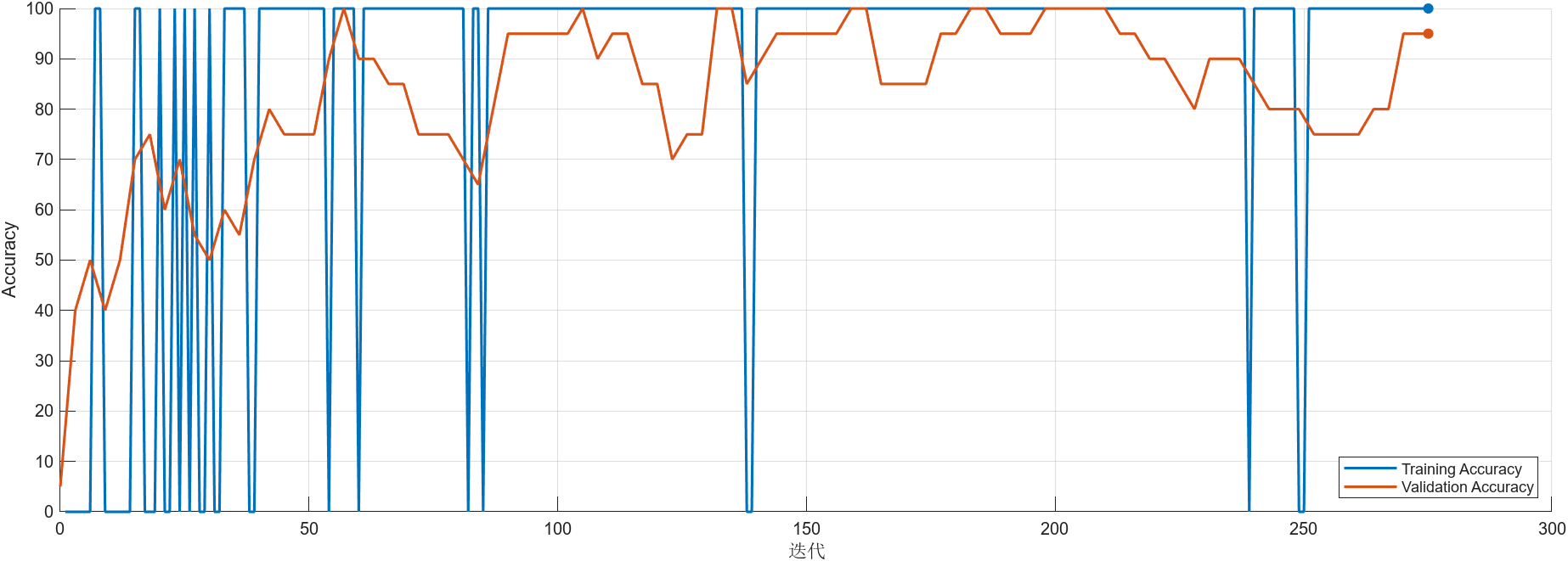


Figure 3.8: Accuracy Curve with a batch size of 1

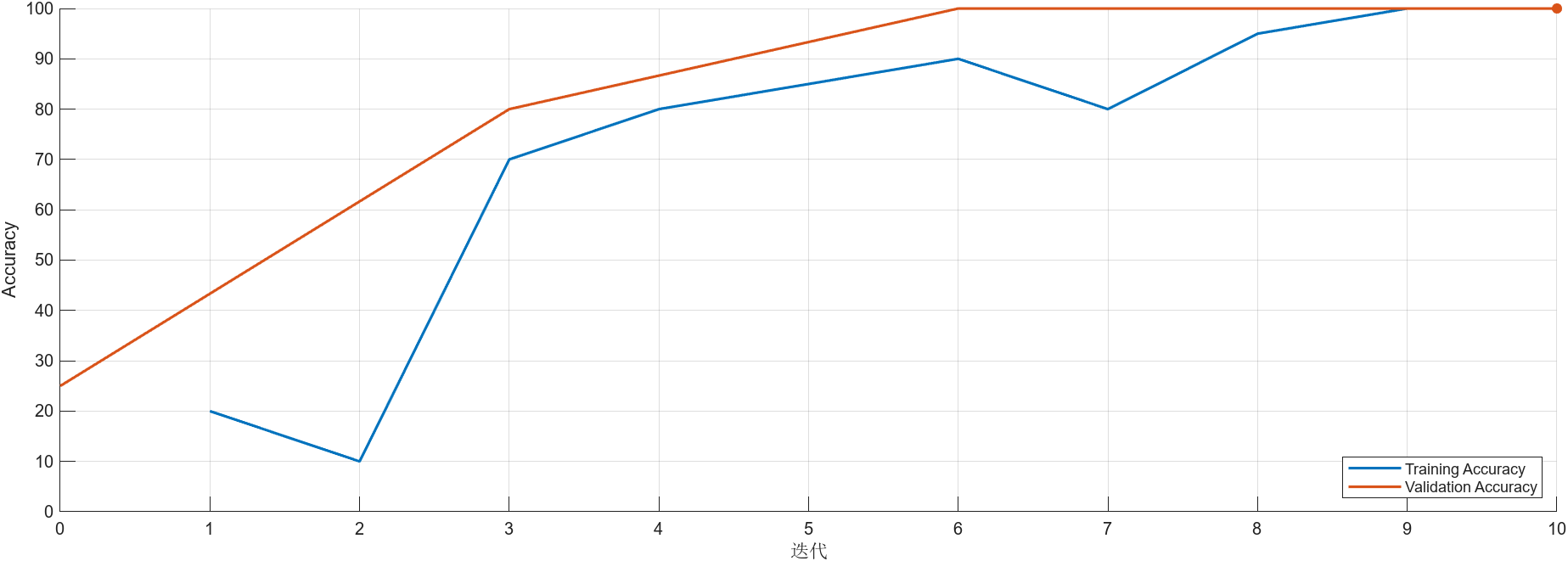


Figure 3.9: Accuracy Curve with a batch size of 20