Predicting Fruit Differentials through Videos, Pictures

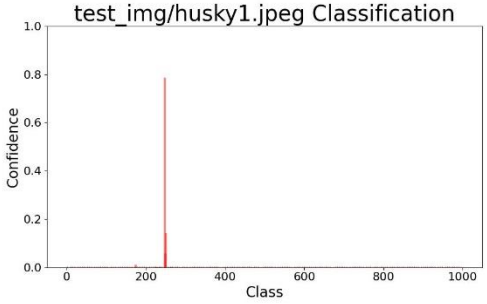
Name:Xintian Yin Number:22020583

Abstract

Machine learning has now seeped into life as a variety of different forms, for example, in the covin-9 period, China in order to prevent the spread of the epidemic, access to places in public places, set up cameras to detect the temperature of the monitoring is placed at the door, when the detection of human beings, appearing to be higher than the normal temperature of the body will notify the relevant personnel. Machine learning plays a huge role in remote identification. Training models for object recognition is also one of the main uses of the camera in the future, I chose to try to make a demo using the models that already existed, and then downloaded the dataset of 25 different categories of fruits by myself to do fruit recognition training. After completing the video to identify the fruit categories, some of the images were drawn to see the extent of machine learning.

Demo Pre-training image classification model recognition prediction

Using the existing database and model, import images to test recognition accuracy.



Part1 Train model - Fruit Recognition

Install and configure the environment, Import os to create 'output', 'checkpoint' used to, 'diagrams' three empty folders. 'output' is used to store the result files, 'checkpoint' is used to store the weight files of the trained model, and 'diagrams' is used to store the generated diagrams. The generated diagrams. Import the font package and choose Times the roman font.

Prepare the image classification dataset, the fruit classification dataset includes: apple, banana, carrot, cherries, cucumber, durian, kiwi, lemon, pitaya and other fruits in all

seasons of the year.

Check the dataset directory structure, see the dataset is divided into two train and val two parts, first of all, you have to take the dataset of the train to do data training, and then use the dataset of the val to do the test, test the data to see if it can run successfully. If it runs well on the training set, but performs mediocre on the val set, then call this effect overfitting (look up the terminology). Next import the toolkit, import the system modules time, os, import the data analysis progress bar module import numpy as np, from tqdm import **tqdm**, and go on to import the pytorch related modules and the icon matplotlib module. Get computing hardware, detect if the environment has GPU, if not, is by CPU, device cuda:0.

Perform image preprocessing & load image classification dataset

Set up two different preprocessing methods for the training set and val set respectively, for the training set of images for random scaling, cropping, image enhancement, into pytorch's Tensor mode, followed by the RGB channels for uniformity, divided by the mean value of the standard deviation, into a normal distribution to make this data easier to be processed by the neural network. In this way the training set of images, carried out to artificially increase the number. The increase in the number of different forms of cropping, rotating, scaling, panning, filling and other image effects can firstly increase the number of datasets, enrich the datasets with different degrees of variations in the images, and secondly improve the accuracy of the fruit recognition at a later stage after the increase in the number of training datasets. To val not image enhancement, re-scaled into a 256 * 256 square, then CenterCrop into a 224 * 224 map into ToTensor, followed by the RGB channel for uniformity, with the mean divided by the standard deviation, into a normal distribution to make this data more easily processed by the neural network.

```
]: print('Number of images in the training set', len(train_dataset))
   print('Number of categories', len(train_dataset.classes))
   print('Name of each category', train_dataset.classes)

Number of images in the training set 3649
Number of categories 25
Name of each category ['CherryTomatoes', 'Mangosteen', 'MomordicaCharantia', 'Nav
elOrange', 'Sandsugaroranges', 'apple', 'banana', 'carrot', 'cherries', 'cucumbe
r', 'durian', 'grape', 'hamimelon', 'kiwi', 'lemon', 'lichee', 'longan', 'mango',
'pear', 'pineapple', 'pitaya', 'pomegranate', 'strawberry', 'tomato', 'watermelo
n']

2]: print('Number of test set images', len(test_dataset))
   print('Number of categories', len(test_dataset.classes))
   print('Name of each category', test_dataset.classes)

Number of test set images 898
Number of categories 25
Name of each category ['Cherrytomatoes', 'Mangosteen', 'MomordicaCharantia', 'Nav
elOrange', 'Sandsugaroranges', 'apple', 'banana', 'carrot', 'cherries', 'cucumbe
r', 'durian', 'grape', 'hamimelon', 'kiwi', 'lemon', 'lichee', 'longan', 'mango',
'pear', 'pineapple', 'pitaya', 'pomegranate', 'strawberry', 'tomato', 'watermelo
n']
```

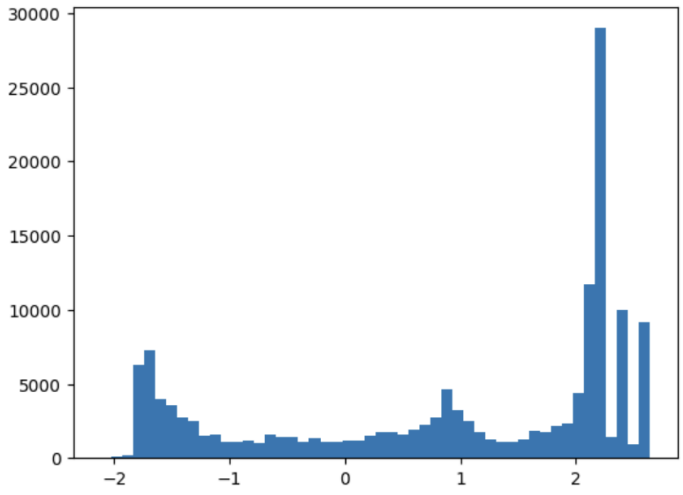
The training set is shown to have 3649 images and the test set has 898 images. The test set is 25% of the training set, with a total of 25 categories. Next, the categories are mapped to index numbers, resulting in 'CherryTomatoes': 0, 'Mangosteen': 1,

'MomordicaCharantia': 2, 'NavelOrange': 3, 'Sandsugaroranges': 4,... 'watermelon': 24, and save as npy file for subsequent queries.

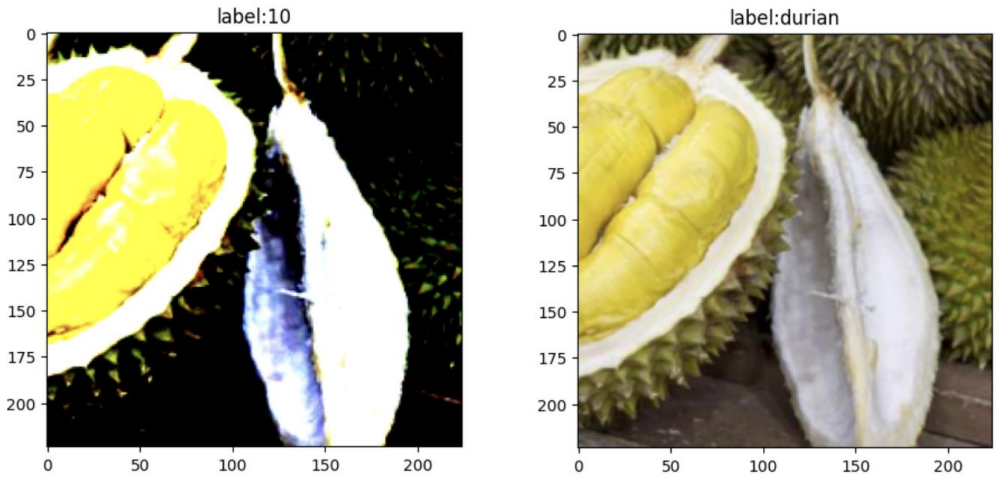
Batch_size is set to 32, the training set is set to shuffle = True to randomly disrupt, and labels are called to see if the batch is randomly disrupted.

```
In [56]: labels

Out[56]: tensor([16, 16, 10,  1, 24,  3,  4,  4, 17, 23,  3,  3, 15,  8, 19, 12,  1, 10,
                 19,  4, 18,  3, 18,  4,  5, 16, 24,  0,  8, 20, 24, 11])
```

View and visualise the batch's images and annotations. The pixel distribution has 0 as the mean, because through the normalisation by preprocessing, the pixels are no longer integers from 0 to 255, but a distribution with negative and positive numbers with 0 as the mean.



A randomly selected imshow(), which could not display a normal durian image, was extracted and modified to obtain a normal durian image.



Then migration learning training is performed, randomly initialising all the weights of the model and training all the layers from scratch. Only load the model structure, not load the pre-training weight parameters, set the random initialisation of ResNet18. Set the loss function to cross-entropy loss function, set epoch=20, which means that epoch train all the training set 20 times. get the predicted category logit scores for all images in the current batch after torch.Size([32, 25]), which means the output is a pytorch tensor or a matrix with 32 rows and 25 columns. Then take the output prediction result drink batch's labelled labels to calculate the cross-entropy loss function to derive the loss function. Then the loss function is backward derived to find the gradient and iteratively optimised to update. optimizer.zero_gard(), loss.backward(), optimizer.step(), optimizer clears the gradient, back propagates to find the gradient, and optimises to update in three steps to achieve this. Next find the corresponding category with the highest confidence, but you can see that the sequences cannot correspond to each other, which proves that the training is wrong. Next, go to the full training.

```
# Get the prediction categories for all images in the current batch
_, preds = torch.max(outputs, 1)
```

```
preds
```

```
tensor([13, 14, 19,  3, 14,  9,  3, 19, 10, 14, 19, 19, 14, 13, 12, 21,  3, 19,
        15, 14, 14, 19,  3, 14,  3, 15, 19, 13,  3, 13, 14,  3],
       device='cuda:0')
```

```
labels
```

```
tensor([23, 15, 10, 24, 15,  3,  9, 15, 17,  0,  5,  1, 24,  0, 21, 14,  6, 13,
        24, 19,  0, 10, 10,  0, 19, 10, 23,  2, 12, 22,  1,  2],
       device='cuda:0')
```

Run full training

Run through all the epochs, adjust the model to training mode before starting, and iteratively fetch individual batch and corresponding labels from the train loader.

Transfer the data to the device, feed the data to the model to perform forward prediction, and get the prediction result. Then calculate the current batch and calculate the average cross-entropy loss function value for each sample. Finally. optimizer.zero_gard(), loss.backward(), optimizer.step(), optimizer clears the gradient, backpropagates for the gradient and optimises the update.

```
# Iterate through each EPOCH
for epoch in tqdm(range(EPOCHS)):

    model.train()

    for images, labels in train_loader:  # Get a batch of the training set with
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)           # Forward Prediction, get the predicti
        loss = criterion(outputs, labels) # Compare the predictions with the ann

        optimizer.zero_grad()
        loss.backward()                   # Loss function back propagation of ne
        optimizer.step()                  # Optimisation to update neural networ
```

```
100%|██████████| 20/20 [01:46<00:00,  5.33s/it]
```

```
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in tqdm(test_loader): # Get a batch of the test set with
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)              # Forward prediction, get the predi
        _, preds = torch.max(outputs, 1)     # Obtain the category corresponding
        total += labels.size(0)
        correct += (preds == labels).sum()   # Number of correct samples predict

    print('The accuracy on the test set is {:.3f} %'.format(100 * correct / tota
```

```
00%|██████████| 29/29 [00:01<00:00, 17.24it/s]
he accuracy on the test set is 62.695 %
```

```
torch.save(model, 'checkpoint/fruit25_pytorch_xintian.pth')
```

Initial testing on the test set yielded an accuracy of 62.695% on the test set.

In the second attempt, add module lr_scheduler aiming at learning rate reduction and optimisation strategy, set up category, index number and mapping dictionary, define dataloader, add lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5), and every five epochs let the learning rate decrease to half of the original, multiplied by the gamma. Set up a couple of functions: compute more loss, accuracy, precision, recall, f1-score.

```python
def train_one_batch(images, labels):
    '''
    Run a batch of training and return the training log for the current batch.
    '''

    # Get a batch of data and annotations
    images = images.to(device)
    labels = labels.to(device)

    outputs = model(images) # Input model to perform forward prediction
    loss = criterion(outputs, labels) # Calculate the average cross-entropy loss

    # Optimising update weights
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Get the label category and prediction category of the current batch
    _, preds = torch.max(outputs, 1) # Get the prediction categories for all ima
    preds = preds.cpu().numpy()
    loss = loss.detach().cpu().numpy()
    outputs = outputs.detach().cpu().numpy()
    labels = labels.detach().cpu().numpy()

    log_train = {}
    log_train['epoch'] = epoch
    log_train['batch'] = batch_idx
    # Calculation of disaggregated assessment indicators
    log_train['train_loss'] = loss
    log_train['train_accuracy'] = accuracy_score(labels, preds)
    # log_train['train_precision'] = precision_score(labels, preds, average='mac
    # log_train['train_recall'] = recall_score(labels, preds, average='macro')
    # log_train['train_f1-score'] = f1_score(labels, preds, average='macro')

    return log_train
```

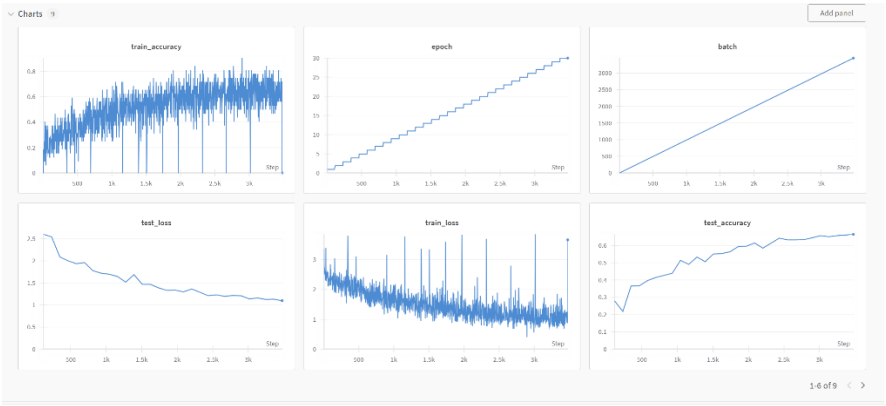
Functions: Evaluate on the whole test set

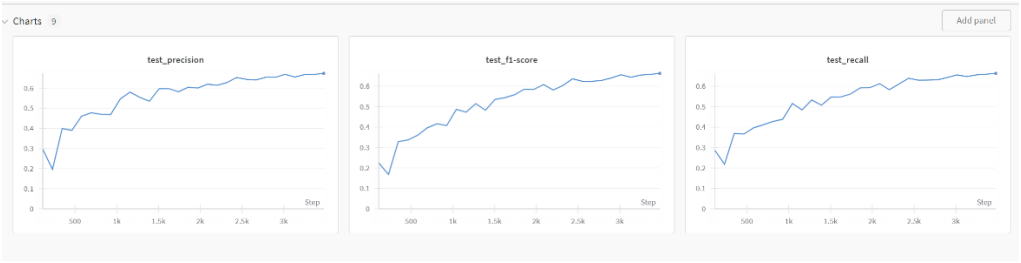
```python
def evaluate_testset():
    '''
    Evaluate on the entire test set and return a log of categorised evaluation m
    '''

    loss_list = []
    labels_list = []
    preds_list = []

    with torch.no_grad():
        for images, labels in test_loader: # Generate a batch of data and annota
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images) # Input model to perform forward prediction

            # Get label categories and prediction categories for the entire test
            _, preds = torch.max(outputs, 1) # Get the prediction categories for
            preds = preds.cpu().numpy()
            loss = criterion(outputs, labels) # From logit, calculate the averag
            loss = loss.detach().cpu().numpy()
            outputs = outputs.detach().cpu().numpy()
            labels = labels.detach().cpu().numpy()

            loss_list.append(loss)
            labels_list.extend(labels)
            preds_list.extend(preds)

    log_test = {}
    log_test['epoch'] = epoch

    # Calculation of disaggregated assessment indicators
    log_test['test_loss'] = np.mean(loss_list)
    log_test['test_accuracy'] = accuracy_score(labels_list, preds_list)
    log_test['test_precision'] = precision_score(labels_list, preds_list, averag
    log_test['test_recall'] = recall_score(labels_list, preds_list, average='mac
    log_test['test_f1-score'] = f1_score(labels_list, preds_list, average='macro

    return log_test
```

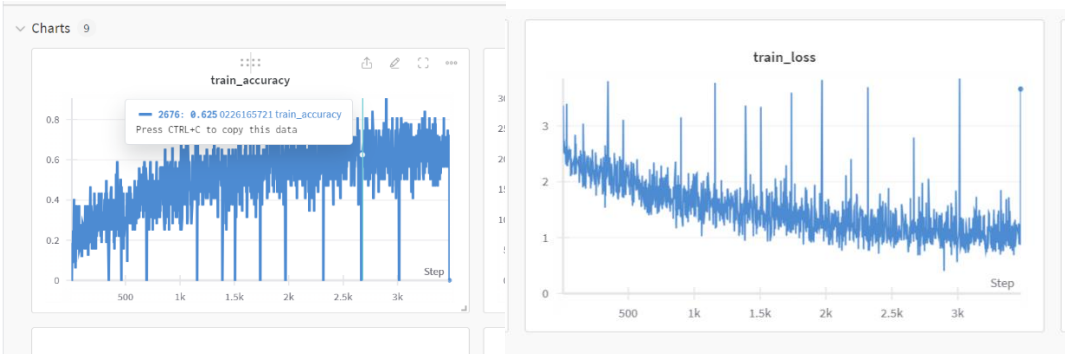
	epoch	batch	train_loss	train_accuracy
0	0	0	3.3938172	0.03125

	epoch	test_loss	test_accuracy	test_precision	test_recall	test_f1-score
0	0.0	3.697069	0.046771	0.022265	0.046261	0.01971

Before this training begins, record the logs, on the training set the loss is 3.39 and the probability of guessing correctly as the correct loss rate is 0.03125.Next evaluate the accuracy on the entire test set, which is 3.69 with an accuracy of 0.046771.Next use wandbpip install wandb, monitor and record the data for the exercise and the change in values. Run the training phase and test phase to get and save the latest best model file.







After training is complete load the best model 'best-0.675.pth' and using evaluate_testset, do a complete evaluation of the entire test set. Yields {'epoch': 30, 'test loss': 1.0977473, 'test accuracy': 0.6648106904231625, 'test_ precision': 0.6735762147736243, 'test recall'. 0.66314819409647, 'test_f1-score': 0.6627346390295513}. And training set logs, and test set logs.

```
100%|██████████| 115/115 [00:05<00:00, 21.76it/s]
Epoch 6/30
100%|██████████| 115/115 [00:05<00:00, 22.13it/s]
Epoch 7/30
100%|██████████| 115/115 [00:05<00:00, 21.11it/s]
Save the new best model checkpoint/best-0.675.pth
Epoch 8/30
100%|██████████| 115/115 [00:05<00:00, 21.97it/s]
Epoch 9/30
100%|██████████| 115/115 [00:05<00:00, 21.55it/s]
Epoch 10/30
100%|██████████| 115/115 [00:05<00:00, 21.62it/s]
Epoch 11/30
100%|██████████| 115/115 [00:05<00:00, 21.24it/s]
```

Part2 Anticipate new images and videos

```
In [280…  idx_to_labels = np.load('/home/featurize/idx_to_labels.npy', allow_pickle=True).
In [281…  idx_to_labels
```

Load idx_to_labels.npy, import the model 'best-0.675.pth' obtained in training, adjust the model to evaluation and load it into the computing device model = model.eval().to(device).

```
model = torch.load('/home/featurize/fruittestpytorch/best-0.675.pth')
model = model.eval().to(device)
```

Then pre-process the new image, re-scale it to a 256*256 square, then CenterCrop it to a 224*224 map to ToTensor, then unify the RGB channels, divide the standard deviation by the corresponding mean, representing the distribution of the natural image, and turn it into a normal distribution to make this data easier to be processed by the neural network. Load the image and read any image to come up with an image of a tomato. Then input_img = input_img.unsqueeze(0).to(device), pred_logits = model(input_img), in order to Perform forward prediction to get logit prediction scores for all categories. get logit prediction scores for 25 categories. Doing logic softmax on logit scores for these twenty five. finally make a visualisation of the image which shows the highest prediction as 'tomato'.

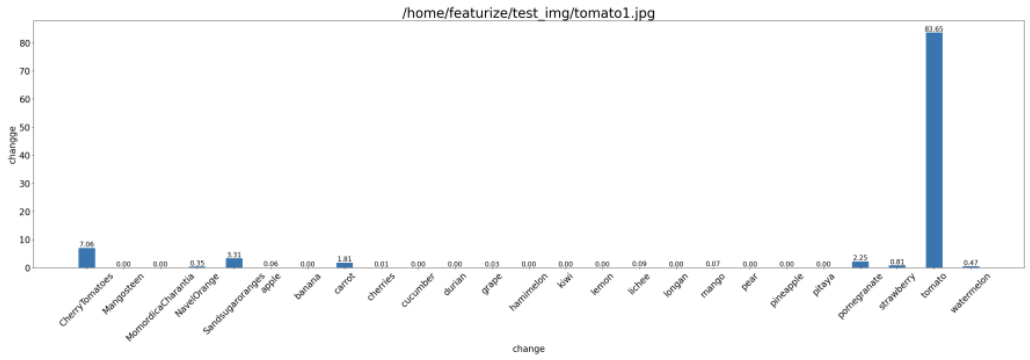
```
# Perform forward prediction to get logit prediction scores for all categories
pred_logits = model(input_img)
```

```
pred_logits
```

```
tensor([[ 2.7197, -6.5022, -5.4831, -0.2764,  1.9642, -2.0090, -5.9099,  1.360
7,
         -4.4711, -8.4817, -5.9957, -2.8411, -4.5781, -9.3885, -6.6772, -1.630
7,
         -4.8560, -1.9126, -5.2103, -6.5983, -6.0284,  1.5787,  0.5504,  5.192
5,
          0.0202]], device='cuda:0', grad_fn=<AddmmBackward0>)
```

```
pred_softmax
```

```
tensor([[7.0566e-02, 6.9754e-06, 1.9327e-05, 3.5271e-03, 3.3149e-02, 6.2364e-0
4,
         1.2612e-05, 1.8129e-02, 5.3167e-05, 9.6355e-07, 1.1576e-05, 2.7138e-0
4,
         4.7773e-05, 3.8908e-07, 5.8558e-06, 9.1042e-04, 3.6182e-05, 6.8677e-0
4,
         2.5388e-05, 6.3359e-06, 1.1203e-05, 2.2546e-02, 8.0628e-03, 8.3655e-0
1,
         4.7448e-03]], device='cuda:0', grad_fn=<SoftmaxBackward0>)
```



The top ten predictions are likely to be fruits in the image. pred_ids: array([23, 0, 4,

21, 7, 22, 24, 3, 15, 17]), check the corresponding fruits according to 'idx_to_labels.npy' derived from training. Write the image classification result on the image and bar graph.

```
n = 10
top_n = torch.topk(pred_softmax, n)
```

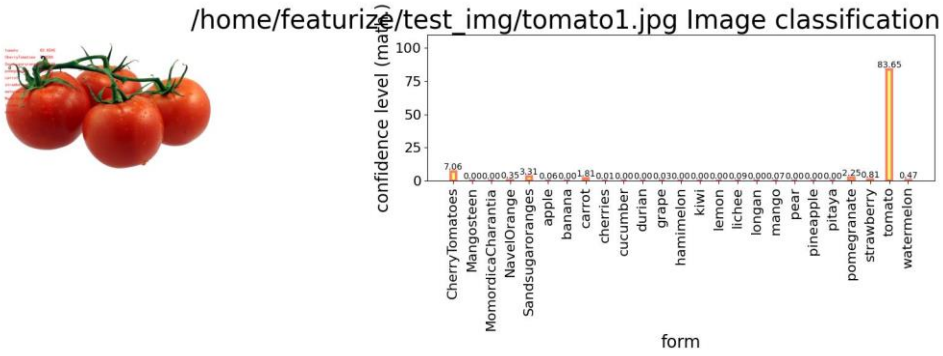
```
top_n
```

```
torch.return_types.topk(
values=tensor([[8.3655e-01, 7.0566e-02, 3.3149e-02, 2.2546e-02, 1.8129e-02, 8.0
628e-03,
         4.7448e-03, 3.5271e-03, 9.1042e-04, 6.8677e-04]], device='cuda:0',
       grad_fn=<TopkBackward0>),
indices=tensor([[23,  0,  4, 21,  7, 22, 24,  3, 15, 17]], device='cuda:0'))
```

```
# Parsing out categories
pred_ids = top_n[1].cpu().detach().numpy().squeeze()
```

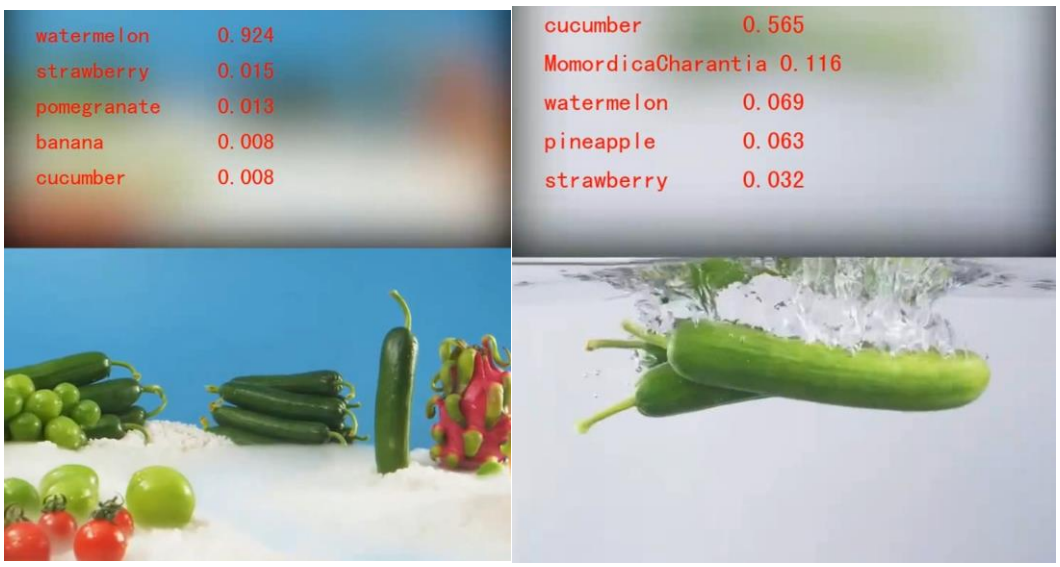
```
pred_ids
```

```
array([23,  0,  4, 21,  7, 22, 24,  3, 15, 17])
```

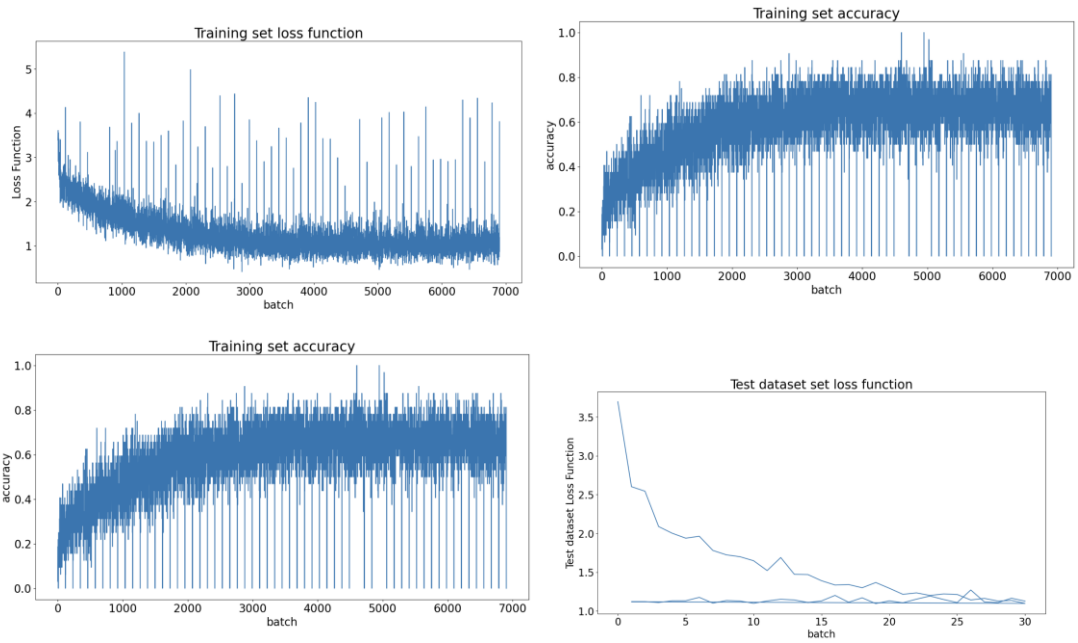


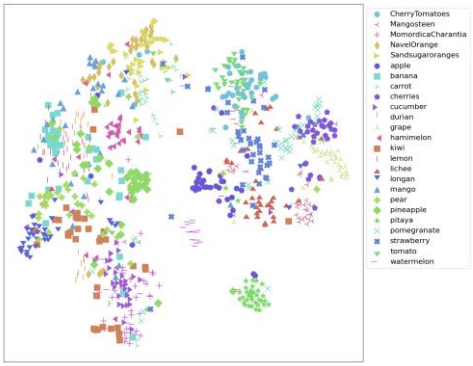
	Class	Class_ID	Confidence(%)
0	tomato	23	83.654517
1	CherryTomatoes	0	7.056604
2	Sandsugaroranges	4	3.314941
3	pomegranate	21	2.254623
4	carrot	7	1.812865
5	strawberry	22	0.806277
6	watermelon	24	0.474478
7	NavelOrange	3	0.352711
8	lichee	15	0.091042
9	mango	17	0.068677

Do the image classification as per the content in the video, first use the mmcv module, the main role of mmcv is to first disassemble the video into each frame, analyse the images of each frame, and then string all the images into a video of the same length as the original video. Derive the video result.



Visualise training logs and produce files such as test set prediction results, extract the output features of the middle layer of the image classification model obtained from Pytorch training as semantic features of the input images. Analyse different categories of semantic distance, anomalous data, fine-grained classification, and high-dimensional data structures.





Conclusion

The accuracy of the model trained with this dataset is only sixty to seventy per cent, partly due to the fact that there is too much information in the fruit pictures, with a lot of Chinese text or irrelevant influences that reduce the accuracy of the pictures. But it still has high accuracy in video and pictures. If the number of datasets can be expanded to have more than 1,000 images each, and more image forms can be converted, the accuracy of the model can be increased to more than eighty per cent.