

ML Final Project Report

110550130 劉秉驊

(please use Colab - CPU to run)

1. Run inference file:

a. /sample_dir

└ 110550130_inference.ipynb

└ 110550130_model.pt

└ /test

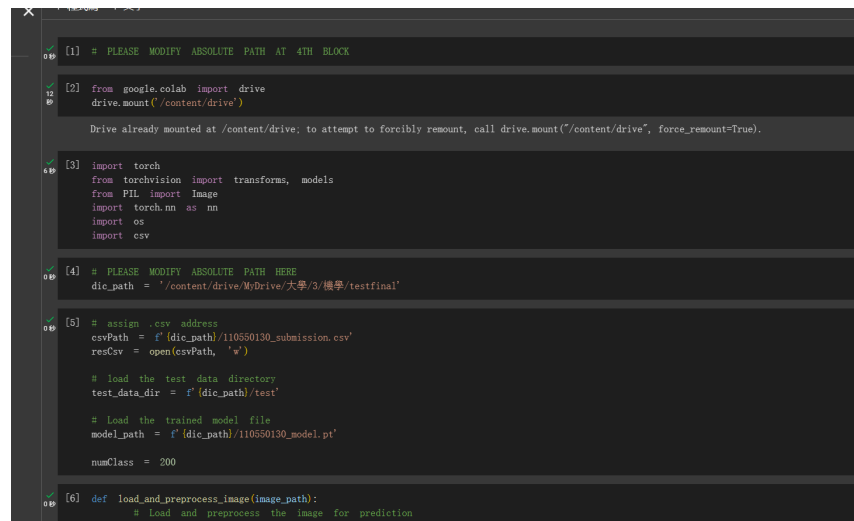
└ image.jpg...

└ 110550130_submission.csv (after running inference file)

b. Please modify the directory path (absolute path)

in 110550130_inference.ipynb

i. Please change the path at 4th block (dir_path)



```
[1] # PLEASE MODIFY ABSOLUTE PATH AT 4TH BLOCK

[2] from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[3] import torch
from torchvision import transforms, models
from PIL import Image
import torch.nn as nn
import os
import csv

[4] # PLEASE MODIFY ABSOLUTE PATH HERE
dir_path = '/content/drive/MyDrive/大學/3/機學/testfinal'

[5] # assign .csv address
csvPath = f'{dir_path}/110550130_submission.csv'
resCsv = open(csvPath, 'w')

# load the test data directory
test_data_dir = f'{dir_path}/test'

# Load the trained model file
model_path = f'{dir_path}/110550130_model.pt'

numClass = 200

[6] def load_and_preprocess_image(image_path):
    # Load and preprocess the image for prediction
    transform = transforms.Compose([
```

ii. For example:

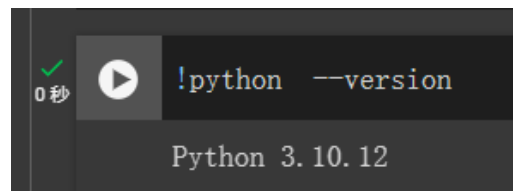
1. dir_path = '/content/drive/MyDrive/ML/sample_dir'

Report:

1. Environment details

a. Python version:

■ Python 3.10.12



```
!python --version

Python 3.10.12
```

b. Framework

■ PyTorch

```
ml - 110550130_train.ipynb

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from torch.utils.data import DataLoader, random_split
5 from torchvision import datasets, models, transforms
6 from tqdm import tqdm
7 import os
```

c. Hardware

- Colab - T4 GPU for training in training file
- Colab - CPU for predicting testing data, produce csv file in inference file

2. Implementation details

a. Model architecture

- ResNet50
- Use pre-trained weight by PyTorch, and the final fully connected layer is replaced with a new linear layer of 200 Classes.

```
ml - 110550130_train.ipynb

1 model = models.resnet50(weights = models.ResNet50_Weights.DEFAULT)
2 num_fts = model.fc.in_features
3 model.fc = nn.Linear(num_fts, numClass)
4 model = model.to(device)
```

- Use cross entropy as loss function, introduce SGD as optimizer, and learning rate scheduler as well decreasing learning rate per 7 epochs.

```
ml - 110550130_train.ipynb

6 criterion = nn.CrossEntropyLoss()
7 optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9)
8 scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

- For each epoch, keep alternating train, validation phases. Validation phase just evaluates the model.

```
ml - 110550130_train.ipynb

2 for epoch in range(num_epochs):
3     print(f'Epoch {epoch}/{num_epochs - 1}')
4     print('-' * 10)
5
6     for phase in ['train', 'val']:
7
8         model.train() if phase == 'train' else model.eval()
9
10        running_loss = 0.0
11        running_corrects = 0
12
13        for inputs, labels in tqdm(dataloaders[phase]):
14            inputs, labels = inputs.to(device), labels.to(device)
15            optimizer.zero_grad()
16
17            with torch.set_grad_enabled(phase == 'train'):
18                outputs = model(inputs)
19                _, preds = torch.max(outputs, 1)
20                loss = criterion(outputs, labels)
21
22            if phase == 'train':
23                loss.backward()
24                optimizer.step()
25
26            running_loss += loss.item() * inputs.size(0)
27            running_corrects += torch.sum(preds == labels.data)
28
29        if phase == 'train':
30            scheduler.step()
31
32        epoch_loss = running_loss / dataset_sizes[phase]
33        epoch_acc = running_corrects.double() / dataset_sizes[phase]
34
35        print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}\n')
```

b. Hyperparameters

- These are the parameters I used.

```
ml - 110550130_train.ipynb
1 # parameter
2 num_epochs = 17
3 batch_size = 16
4
5 data_dir = '/content/drive/MyDrive/大學/3/機學/final/data/train'
6 model_save_path = '/content/drive/MyDrive/大學/3/機學/final/data/myModel_e17_b16_0.85tra.pt'
7
8 learning_rate = 0.001
9
10 numClass = 200
```

- And here is data loader

```
ml - 110550130_train.ipynb
1 train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=4)
2 val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=4)
```

c. Training strategy

- For dataset, I preferred 224*224 and normalized them.

```
ml - 110550130_train.ipynb
1 data_transforms = {
2     'train': transforms.Compose([
3         transforms.RandomResizedCrop(224),
4         transforms.RandomHorizontalFlip(),
5         transforms.ToTensor(),
6         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
7     ]),
8     'val': transforms.Compose([
9         transforms.Resize(256),
10        transforms.CenterCrop(224),
11        transforms.ToTensor(),
12        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
13    ]),
14 }
```

- I also splitted 93% for training data, the rest for validation data.

```
ml - 110550130_train.ipynb
17 train_size = int(0.93*len(full_dataset))
```

- From this website [如何训练你的ResNet\(二\):Batch的大小、灾难性遗忘将如何影响学习率 - 知乎 \(zhihu.com\)](#), I couldn't set too large batch that it might result in higher loss, though bigger batch size could accelerate the speed of training.
- Therefore, set batch size like below

```
ml - 110550130_train.ipynb
1 # parameter
2 num_epochs = 17
3 batch_size = 16
4
5 data_dir = '/content/drive/MyDrive/大學/3/機學/final/data/train'
6 model_save_path = '/content/drive/MyDrive/大學/3/機學/final/data/myModel_e17_b16_0.85tra.pt'
7
8 learning_rate = 0.001
9
10 numClass = 200
```

3. Experimental results

a. Evaluation metrics

- metrix

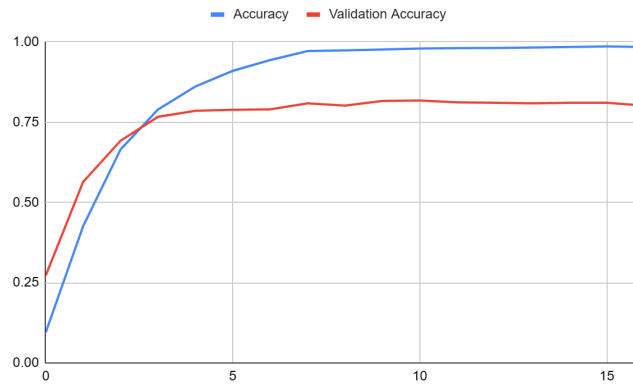
Epoch	Train Loss	Accuracy	validation Loss	Validation Accuracy
0	4.9202	0.0953	3.9268	0.2726
1	2.7857	0.42667	1.7926	0.5641
2	1.4737	0.6649	1.1381	0.6924
3	0.9157	0.7896	0.8853	0.7668
4	0.6114	0.8613	0.8059	0.7857
5	0.4242	0.91	0.7721	0.7886
6	0.2954	0.9436	0.7512	0.7901
7	0.2089	0.9718	0.7198	0.809
8	0.192	0.974	0.727	0.8017
9	0.1831	0.9764	0.7	0.8163
10	0.1693	0.9796	0.708	0.8178
11	0.1643	0.9811	0.7027	0.812
12	0.1608	0.9815	0.715	0.8105
13	0.1516	0.9828	0.7134	0.809
14	0.1469	0.9846	0.706	0.8105
15	0.146	0.9862	0.7042	0.8105
16	0.1462	0.9846	0.7317	0.8017

b. Learning curve

- X-axis is epoch. We can see both of the loss become steady from the 10th epoch.



- X-axis is epoch. We can see both of the accuracy become steady from the 10th epoch.

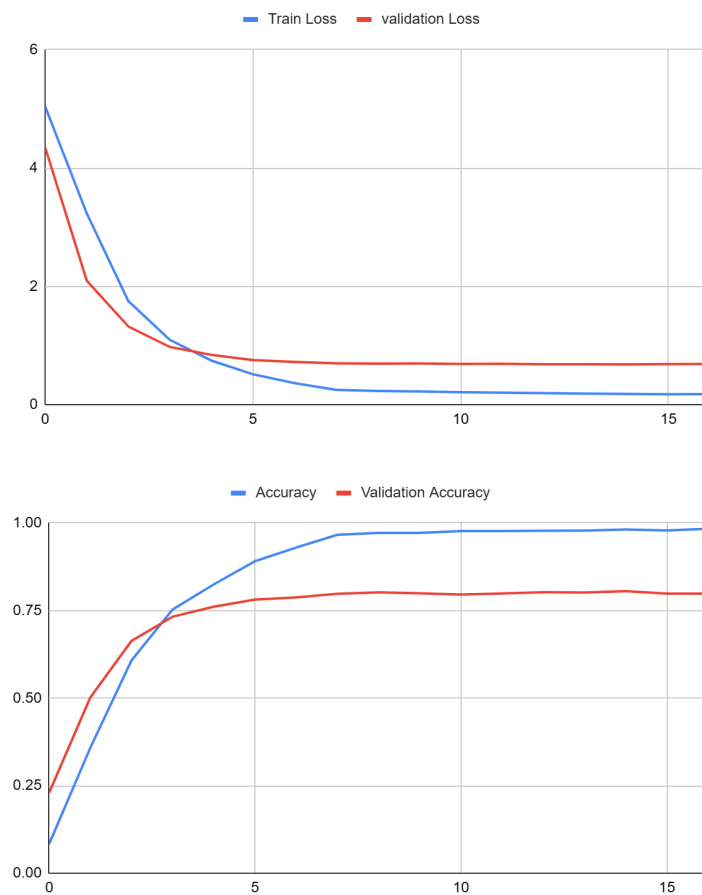


c. Ablation Study

- The main goal: make a bigger proportion of validation data distinguish new images better and avoid overfitting.
- Implement: 80% for training data, 20% for validation data, the rest is unchanged
- Evaluation metrics

Epoch	Train Loss	Accuracy	validation Loss	Validation Accuracy
0	5.0392	0.0839	4.3487	0.2303
1	3.2335	0.3582	2.0955	0.502
2	1.7503	0.6077	1.3208	0.6634
3	1.0949	0.7531	0.9742	0.7324
4	0.7438	0.8248	0.8409	0.761
5	0.5119	0.8911	0.7537	0.7814
6	0.363	0.9298	0.7201	0.7875
7	0.2496	0.9664	0.6988	0.7978
8	0.2311	0.9715	0.6932	0.8018
9	0.222	0.9716	0.696	0.7993
10	0.209	0.9771	0.6876	0.7957
11	0.201	0.9768	0.6907	0.7988
12	0.194	0.9779	0.6806	0.8023
13	0.1866	0.9784	0.682	0.8013
14	0.1796	0.9814	0.6786	0.8054
15	0.1753	0.9788	0.6834	0.7983
16	0.1778	0.9835	0.6863	0.7983

■ Learning curve



- Conclusion: As we can see, from 93% to 80% splitted training data can't even improve the validation accuracy. And just like the result above, since the 10th epoch, we've obtained a steady outcome. As for overfitting, I can't distinguish any difference.

4. Compare

a. InceptionV3

- Originally, I used InceptionV3 to recognize birds.

- With parameter below: I use imageNet pre-trained weight modified a little, and split 3% for validation data.

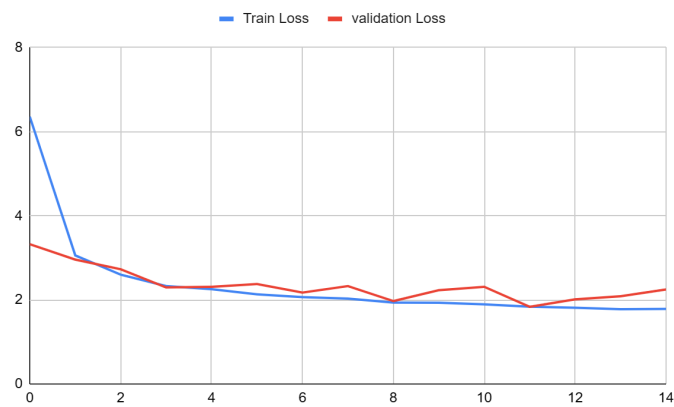
```
batch_size = 128
epoch = 15
dense = 1024
num_class = 200
dropout = 0
learning_rate = 0.01

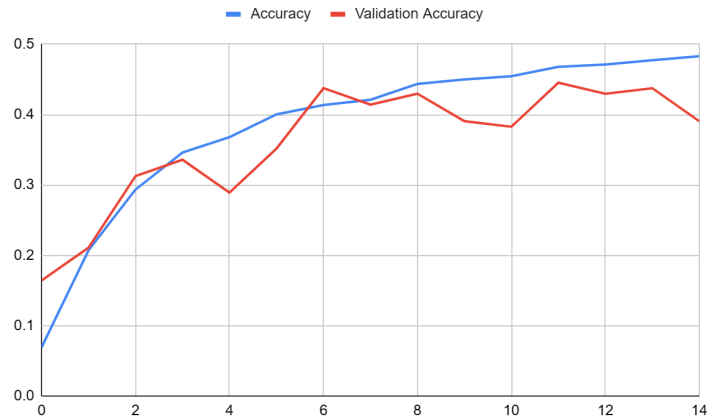
# Load the InceptionV3 model pre-trained on ImageNet data
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299, 299, 3))

# Data augmentation for training set
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.03
)
```

- And I got the result: X-axis is epoch.

Epoch	Train Loss	Accuracy	validation Loss	Validation Accuracy
0	6.3575	0.0693	3.3236	0.1641
1	3.0566	0.2073	2.9571	0.2109
2	2.6008	0.2934	2.7281	0.3125
3	2.3284	0.3462	2.2944	0.3359
4	2.2553	0.3679	2.3085	0.2891
5	2.1302	0.4003	2.3765	0.3516
6	2.0644	0.4137	2.1745	0.4375
7	2.0306	0.4211	2.3262	0.4141
8	1.9365	0.4437	1.9719	0.4297
9	1.9306	0.45	2.2306	0.3906
10	1.8936	0.4545	2.3105	0.3828
11	1.8372	0.468	1.8356	0.4453
12	1.8121	0.4712	2.0117	0.4297
13	1.7767	0.4773	2.0864	0.4375
14	1.783	0.483	2.248	0.3906





Conclusion:

- What we see:
 - We can see InceptionV3 has a slow growth at both loss and accuracy compared to ResNet50. At 14th epoch, the accuracy is approaching 0.5 slowly. We can expect how much time it takes to reach 0.75.
 - If we want to get at least 75% accuracy, from this paper [Advanced Guide to Inception v3 | Cloud TPU | Google Cloud](#) mentioned that InceptionV3 needs to run at least 140 epochs to reach enough accuracy, thus we might need to run more than 100 epochs, but it takes too much time.
- Result:
 - I quitted using InceptionV3 because it consumes too much resource and time.