Image Classification

fine-Tuning on resnet18

Environment

- 程式碼位於Google Colab

- Google Colab是什麼?
  - 以Jupyter notebook呈現
  - 操作位於雲端的虛擬機
  - 提供基本環境 (Torch, numpy等等)
  - 可以共用分享

PyTorch

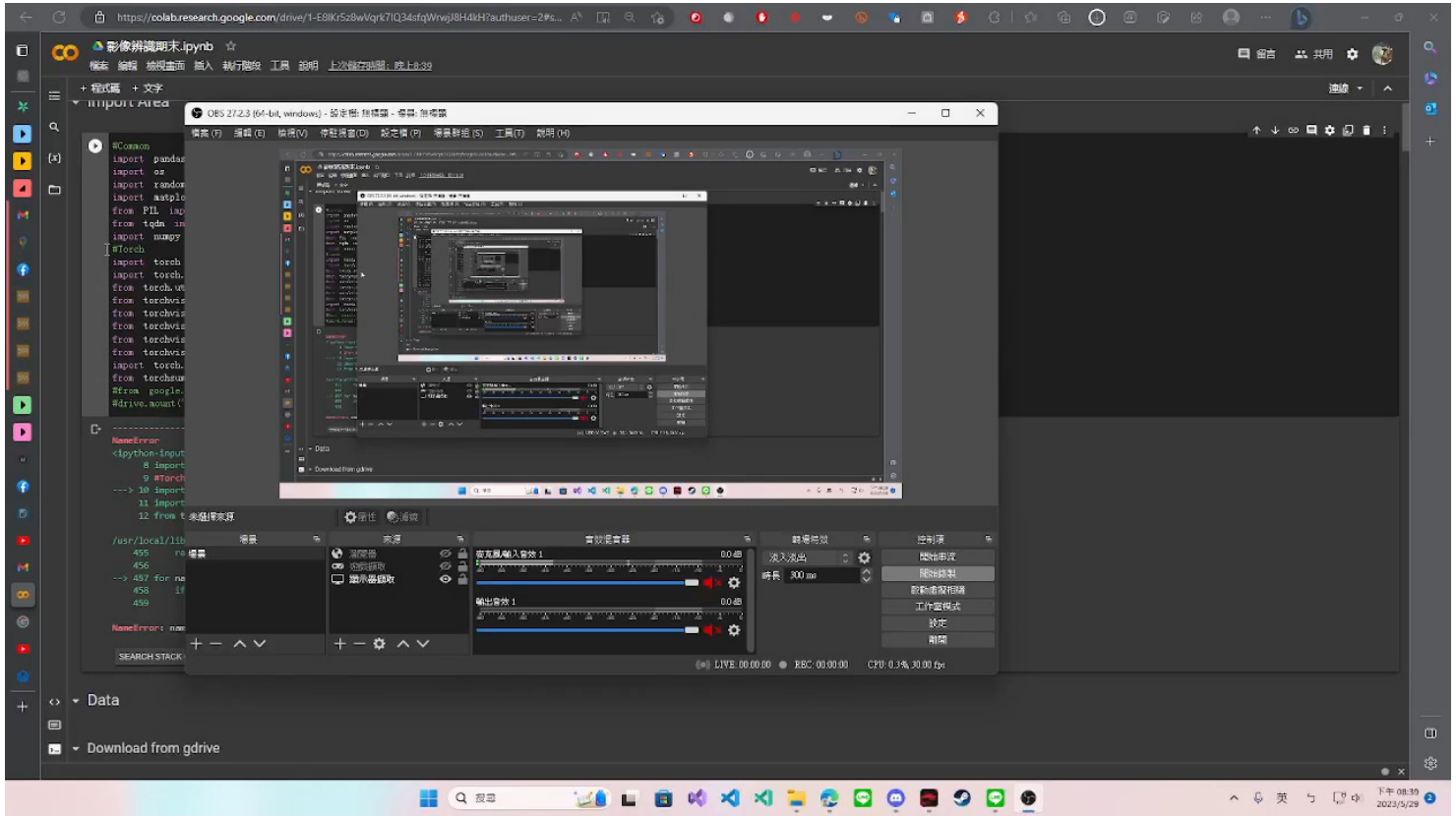
Google Colaboratory

Dataset

- Link on Kaggle: Dataset
- 內有53張撲克牌（各個花色的A~13 + Joker）
  - 每一張牌都有120~160張圖片
- 共有約8000張圖片
  - Train: 7624
  - Test: 265
  - Validation: 265

Demo

- Link



Import

```
[ ]  #Common
     import pandas as pd
     import os
     import random
     import matplotlib.pyplot as plt
     from PIL import Image
     from tqdm import tqdm
     import numpy as np
     #Torch
     import torch
     import torch.nn as nn
     from torch.utils.data import Dataset, DataLoader
     from torchvision import datasets
     from torchvision.models import resnet18
     from torchvision.transforms import ToTensor
     from torchvision.io import read_image
     from torchvision import transforms
     import torch.optim as optim
     from torchsummary import summary
     #from google.colab import drive
     #drive.mount('/content/gdrive')
```

Custom Dataset

```python
class CustomImageDataset(Dataset):
    def __init__(self, dataframe, img_dir, transform=None, target_transform=None):
        self.img_labels = dataframe
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        #Image: Path -> PIL.Image -> Tensor(Tranfrom)
        #Label: int -> Tensor(Transform)
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 1])
        image = Image.open(img_path)
        label = self.img_labels.iloc[idx, 0]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

Data Preprocess

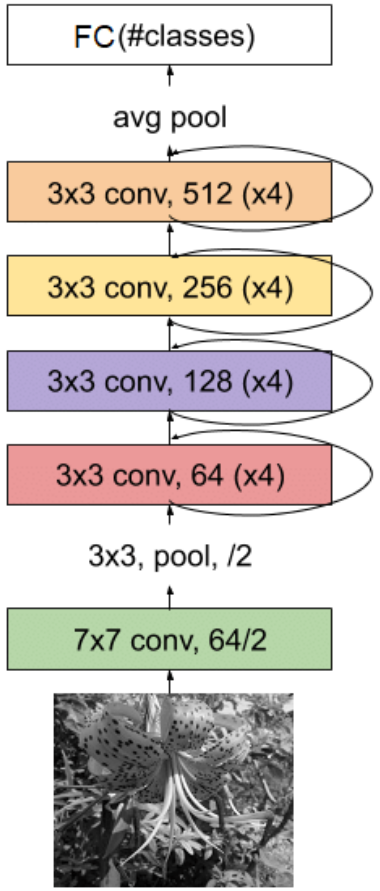
- 以下是有使用到的前處理Method
  - Resize
  - 隨機水平翻轉
  - 隨機垂直翻轉
  - ToTensor
  - Normalize

```
tform_train = transforms.Compose([
    transforms.Resize(224),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

tform_test = transforms.Compose([
    transforms.Resize(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

Model

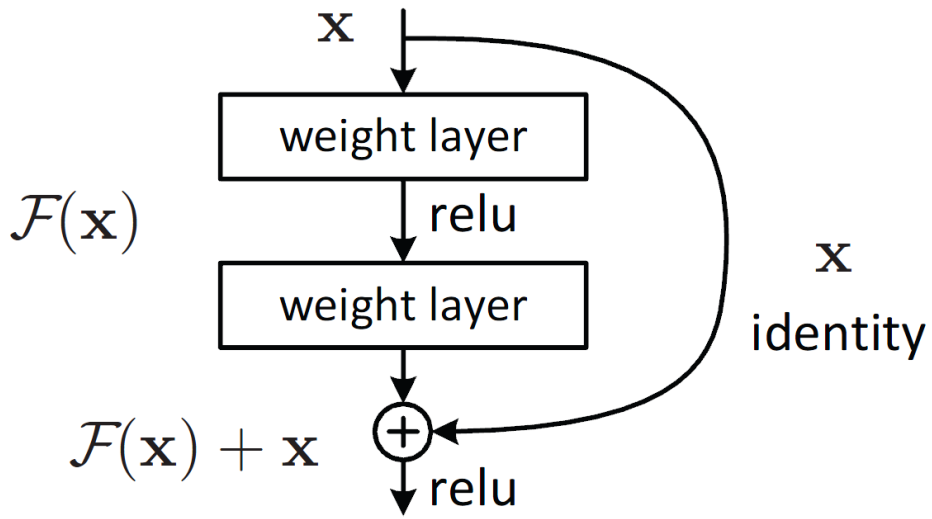
- Detail
  - 名稱：ResNet18
  - Pre-trained on ImageNet

```
model = resnet18(weights='DEFAULT')
model.fc = nn.Sequential(
    nn.Dropout(p=0.5),
    nn.Linear(512, len(class_labels))
)
model
```



Residual Network

- Residual（殘差）的好處
  - 避免梯度消失
  - 可以增加深度
  - 模型表現變好
  - 選擇較好的Layer



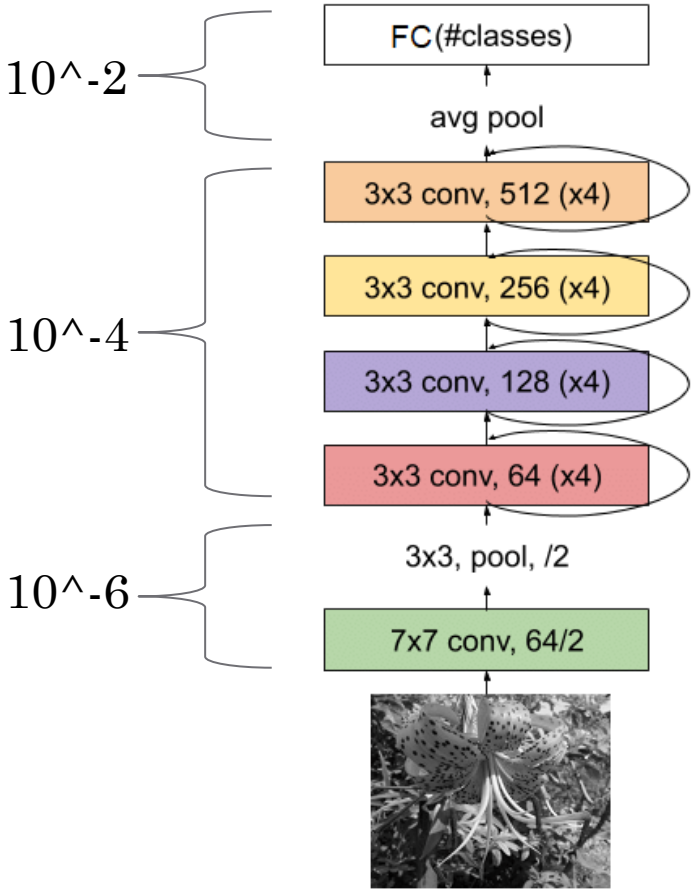
Hyper Parameter

- Epoch: 30
- Batch size: 32
- Optimizer: Adam
  - Learning rate: 10^-6, 10^-4, 10^-2
  - Schedular step: 30
- Early stop: 3

```
class Config:
  epoch = 30
  batch_size = 32
  num_workers = 0
  lr = [1e-6, 1e-4, 1e-2]
  lr_step = 30
  device = 'cuda' if torch.cuda.is_available() else 'cpu'
  patience = 3
  checkpoints = "./4.pth"
  test_model = ""
```

Fine Tuning

- 樣本數較少時使用
- 方法
  - 凍結Layer
  - 給予每個Layer不同的Learning rate
    - 底層->10^-6
    - 中間的卷積層 -> 10^-4
    - 分類層 -> 10^-2



Data Loader(Code)

```
print(len(train_ds))
train_dataLoader = DataLoader(train_ds, batch_size=Config.batch_size, shuffle=True,
                num_workers=Config.num_workers, drop_last=True)
```

```
7624
```

```
print(len(valid_ds))
valid_dataLoader = DataLoader(valid_ds, batch_size=Config.batch_size, shuffle=True,
                num_workers=Config.num_workers, drop_last=True)
```

```
265
```

Checkpoint(Code)

```python
need_to_train = False #Set true if you want to continue training
last_epoch = -1
try:
  # Using when training
  # weight_list = os.listdir(Config.checkpoints)
  # weight_list = [w.split('.')[0] for w in weight_list]
  # lastest_weight = f"{weight_list[-1]}.pth"
  # best_weight_dir = os.path.join(Config.checkpoints, lastest_weight)

  # Load checkpoint(weight, last_epoch, loss) in to model
  best_weight_dir = "./4.pth"
  checkpoint = torch.load(best_weight_dir, map_location=Config.device)
  model.load_state_dict(checkpoint['model_state_dict'])
  last_epoch = checkpoint['epoch']
  loss = checkpoint['loss']
  Config.test_model = best_weight_dir
  print(f"Load best weight path:{best_weight_dir}, epoch:{last_epoch} / {Config.epoch}, loss:{loss}")
except Exception as e:
  print(e)
  need_to_train = True
```

Fine Tuning(Code)

```
#Param1 -> bottom few layers 10^-6
#Param2 -> others 10^-4
#Param3 -> addition layer 10^-2
modelParams = []
temps = []

for param in list(model.parameters())[0:3]:
  temps.append(param)

modelParams.append(temps)

temps = []
for param in list(model.parameters())[3:-2]:
  temps.append(param)

modelParams.append(temps)

temps = []
for param in list(model.parameters())[-2:]:
  temps.append(param)

modelParams.append(temps)
```

Optimizer(Adam) Code

```
# Training Setup

# Loss -> cross entropy
criterion = nn.CrossEntropyLoss()

# Optimer -> Adam
optimizer = optim.Adam(
    [
        {'params': modelParams[0], 'lr':Config.lr[0]},
        {'params': modelParams[1], 'lr':Config.lr[1]},
        {'params': modelParams[2], 'lr':Config.lr[2]},
    ],
    lr=Config.lr[1],
)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=Config.lr_step, gamma=0.1)
```

Train Code (Train dataset)

```python
if need_to_train:
  model.to(Config.device)
  best_loss = 100 #For valid set
  count = 0

  # Iteriate
  for e in range(last_epoch + 1, Config.epoch):
    # Train
    correct_count = 0
    train_last_epoch = e
    model.train()

    with tqdm(train_dataLoader, desc=f"Epoch {e}/{Config.epoch}", total=len(train_dataLoader)) as progress_bar:
      for data, labels in progress_bar:
        #To cpu/gpu
        data = data.to(Config.device)
        labels = labels.to(Config.device)
        #Gradient descent
        optimizer.zero_grad()
        #Output
        outputs = model(data)
        #Gradient descent by loss
        train_loss = criterion(outputs, labels)
        train_loss.backward()
        #Optimzer
        optimizer.step()

        #Get prediction and count accuracy
        _, preds = torch.max(outputs, 1)
        correct_count += torch.sum(preds == labels.data)

        progress_bar.set_postfix({"loss": train_loss.item()})

    acc = correct_count.double() / (len(train_dataLoader) * Config.batch_size)

    print(f"Epoch {e}/{Config.epoch}, Train loss: {train_loss}, acc: {acc}")
    #Record
    training_loss.append(train_loss)
    training_acc.append(acc)
```

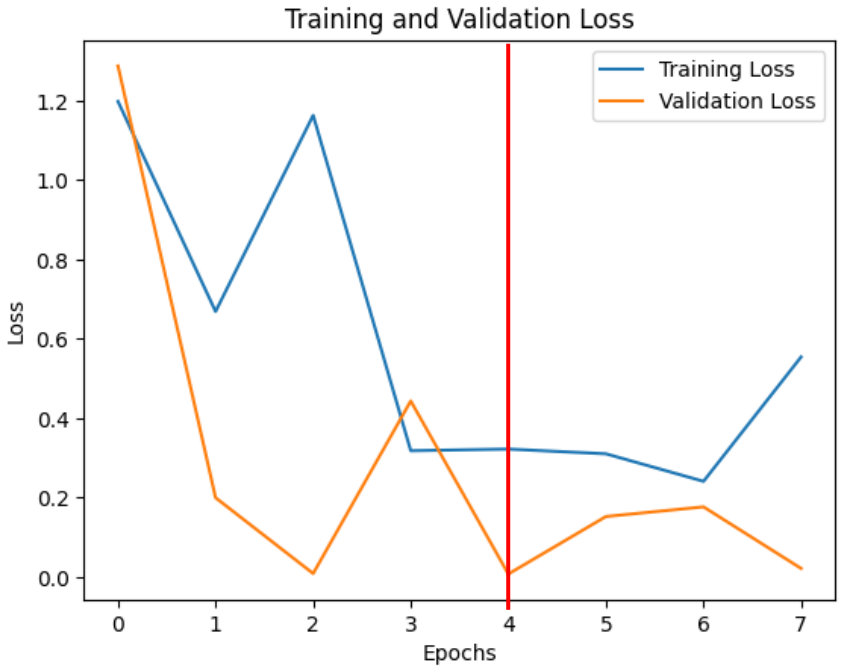
Train Code (Valid dataset)

```
#Evalution mode
correct_count = 0
torch.cuda.empty_cache()
model.eval()

with tqdm(valid_dataLoader, desc=f"Epoch {e}/{Config.epoch}", total=len(valid_dataLoader)) as progress_bar:
  for data, labels in progress_bar:
    # To cpu/gpu
    data = data.to(Config.device)
    labels = labels.to(Config.device)
    # Get output
    outputs = model(data)
    # Loss cal
    val_loss = criterion(outputs, labels)

    # Predicitons
    _, preds = torch.max(outputs, 1)
    correct_count += torch.sum(preds == labels.data)

    progress_bar.set_postfix({"loss": val_loss.item()})

acc = correct_count.double() / (len(valid_dataLoader) * Config.batch_size)
print(f"Epoch {e}/{Config.epoch}, Validation loss: {val_loss}, acc: {acc}")
# Record
validation_loss.append(val_loss)
validation_acc.append(acc)
```

Train Code (Save weight)

```
# If loss smaller than before
if val_loss < best_loss:
  count = 0
  best_loss = val_loss
  # At this point also save a snapshot of the current model
  backbone_path = os.path.join(Config.checkpoints, f"{e}.pth")
  torch.save({
  'epoch': e,
  'model_state_dict': model.state_dict(),
  'optimizer_state_dict': optimizer.state_dict(),
  'loss': val_loss,
  }, backbone_path)
  Config.test_model = backbone_path
else:
  # Early stop
  count += 1
  if count >= Config.patience:
    break

#Clean cache
del train_loss, val_loss, outputs
torch.cuda.empty_cache()

scheduler.step()
```

Loss & Accuracy



Early Stop

註：Training with dropout(p=0.5)

Test dataset(Code)

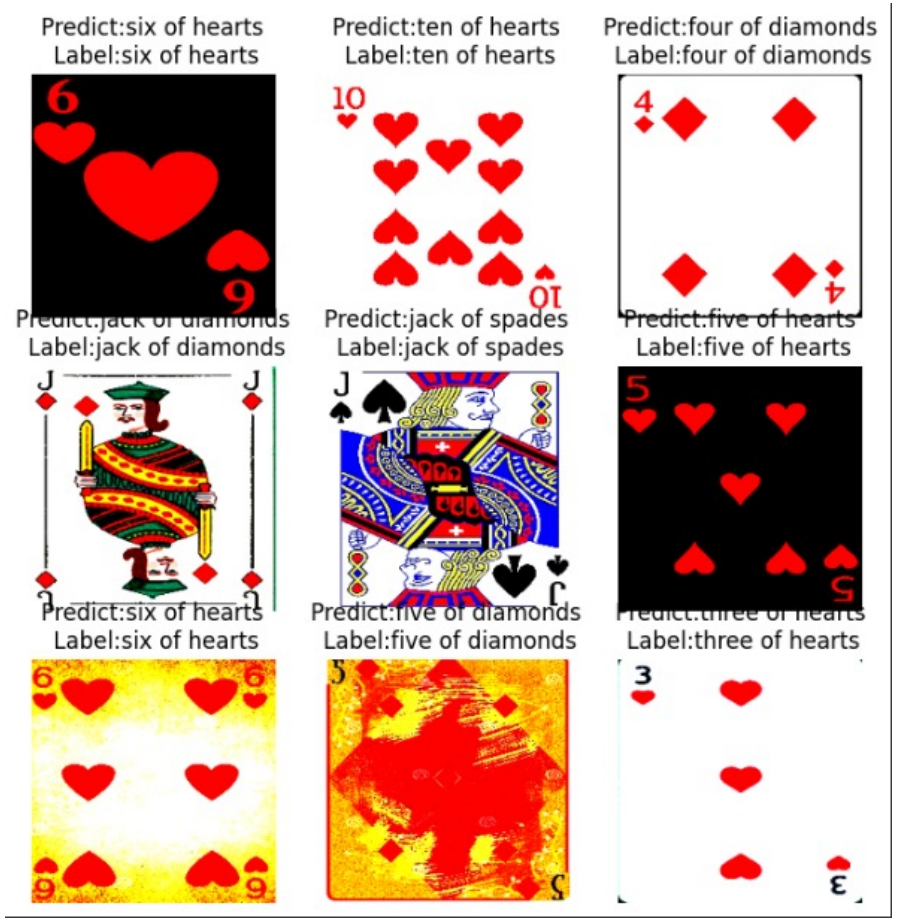
```
correct_count = 0
model.eval()
model.to(Config.device)

with tqdm(test_dataLoader, total=len(test_dataLoader)) as progress_bar:
  for data, lbl in progress_bar:
    #To cpu/gpu
    data = data.to(Config.device)
    lbl = lbl.to(Config.device)
    #Get output
    outputs = model(data)
    #Cal loss
    loss = criterion(outputs, lbl)
    #Get prediction
    _, preds = torch.max(outputs, 1)
    correct_count += torch.sum(preds == lbl.data)

print(f"Loss: {loss}, acc: {correct_count.double() / (len(test_dataLoader))}")
```

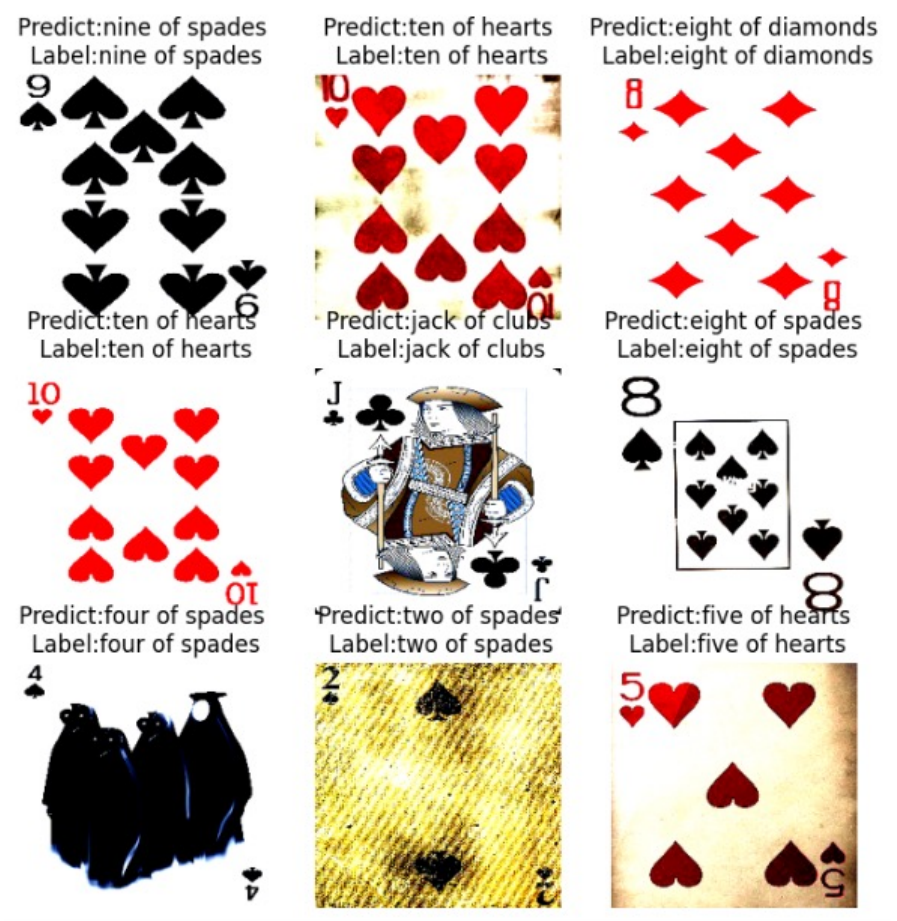
```
100%|██████████| 265/265 [00:25<00:00, 10.25it/s]Loss: 0.0010851691477000713, acc: 0.9320754716981132
```

Show Result 3x3(Code)

```
w = 10
h = 10

columns = 3
rows = 3

fig = plt.figure(figsize=(8, 8))
ax = []

for i in range(1, columns*rows +1):
  #Random fetch image in test dataset
  idx = random.randint(0, len(test_ds)-1)
  show_img, show_lbl = test_ds[idx]
  show_img, show_lbl = show_img.to(Config.device), show_lbl

  #Fit input shape
  x = show_img
  x = x.unsqueeze(0)

  #Evaluation
  model.eval()
  model = model.to(Config.device)

  #Get output without gradent descend
  with torch.no_grad():
      out = model(x)

  show_img, show_lbl, out = show_img.cpu(), show_lbl, out.cpu()

  #Show in subplot
  ax.append(fig.add_subplot(rows, columns, i))
  ax[-1].set_title(f"Predict:{class_labels[np.argmax(out)]}\n Label:{class_labels[show_lbl]}")
  plt.imshow(show_img.permute(1, 2, 0))

  plt.axis('off')

plt.show()
```

Show Result(3x3)





Try on other Image

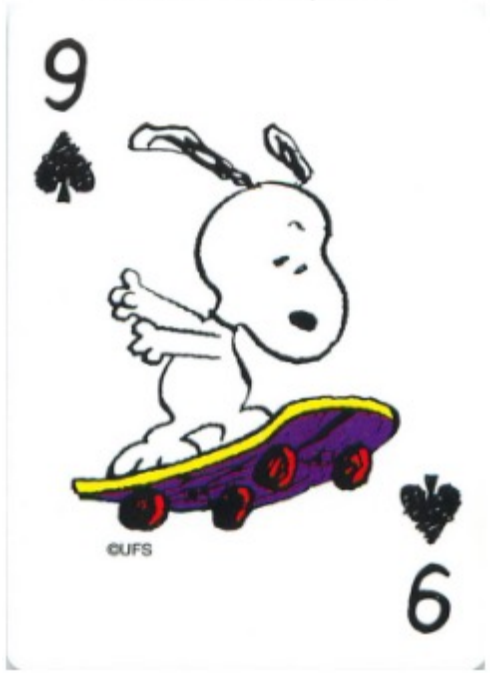
Predict:two of diamonds

Predict:ace of hearts

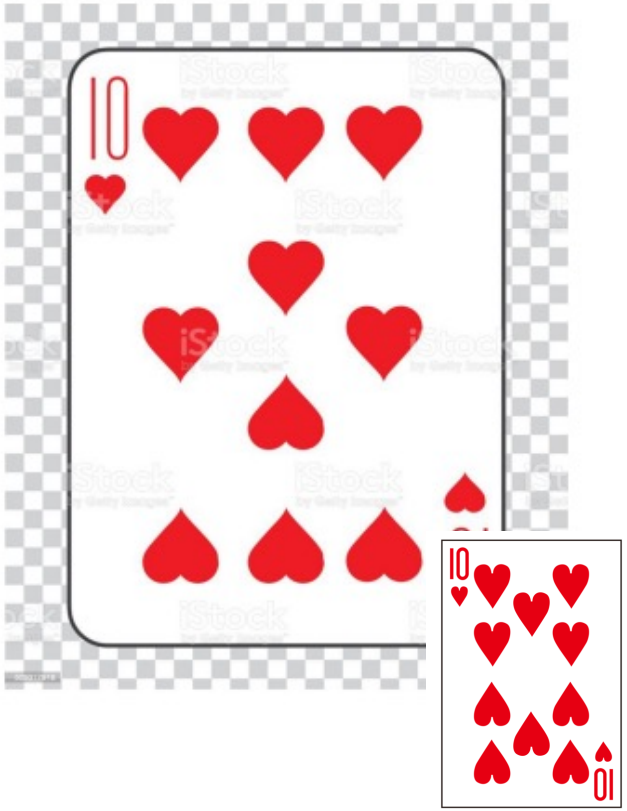
Predict:ace of spades

Special Case(Different style)

Predict:six of spades



Predict:seven of hearts

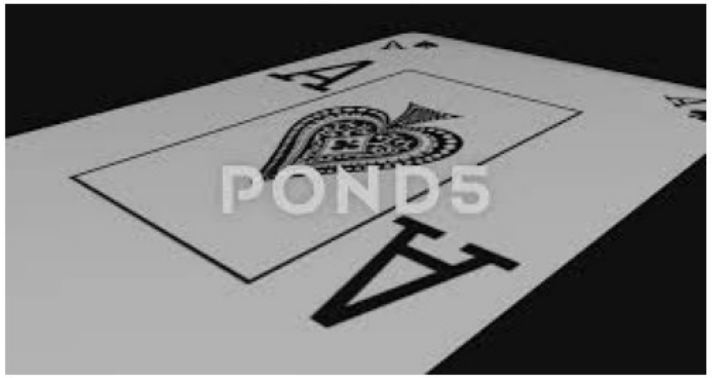


Predict:three of spades

Special Case(Different angle)

Predict:seven of spades

Predict:jack of spades



Predict:ace of hearts

Predict:jack of spades

Special Case(Different angle)

Predict:ace of spades

Predict:four of spades

Predict:four of clubs

Special Case(For Fun)

Predict:three of spades

Predict:ace of spades

Predict:four of clubs

Improvement

- On dataset
  - 裁切重點部分
  - 資料篩選
- On model
  - 結合其他模型
    - Yolo
    - OCR
  - 增加模型深度

Application

- AI牌類遊戲（真實世界）
- 撲克牌品質檢定
- 撲克牌整理
- 發牌機器（及相關統計）

Reference

- Residual network
- Pytorch document
- Transfer learning on ResNet50 (有印象但找不到)

Q&A

Author: LyraEri