Collision Avoidance (ResNet18)

Reference: NVIDIA JetBot Github Repository

0. Import modules

In [1]:

```
import torch
import torch.optim as optim # for SGD
import torch.nn.functional as F
from torch.utils.data import random_split, DataLoader

import torchvision
import torchvision.models as models # for resnet18
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder

import os # isdir, mkdir
import matplotlib.pyplot as plt
from time import strftime, localtime
```

1. Prepare the dataset

In [2]:

```
# DATASET_PATH = "./datasets/dataset_white"
DATASET_PATH = "./datasets/dataset_blue"

IMAGE_WIDTH  = 224
IMAGE_HEIGHT = 224

# The two constants below use the values specified in the reference.
NORMALIZE_MEAN = (0.485, 0.456, 0.406)
NORMALIZE_STD  = (0.229, 0.224, 0.225)

total_dataset = ImageFolder(
    DATASET_PATH,
    transforms.Compose([
        transforms.ColorJitter(0.1, 0.1, 0.1, 0.1),
        transforms.Resize((IMAGE_HEIGHT, IMAGE_WIDTH)),
        transforms.ToTensor(),
        transforms.Normalize(NORMALIZE_MEAN, NORMALIZE_STD)
    ])
)

print(f"{len(total_dataset)} images have been loaded.") # Logger
```

```
240 images have been loaded.
```

In [3]:

```
SPLIT_RATIO = (0.8, 0.1, 0.1) # train : valid : test

total_data_num = len(total_dataset)

train_data_num = int(total_data_num * SPLIT_RATIO[0])
valid_data_num = int(total_data_num * SPLIT_RATIO[1])
model_data_num = train_data_num + valid_data_num

test_data_num  = int(total_data_num * SPLIT_RATIO[2])

model_dataset, test_dataset  = random_split(total_dataset, (model_data_num, t
```

```
train_dataset, valid_dataset = random_split(model_dataset, (train_data_num, v

#-- Logger --#
print(f"Train Dataset: {len(train_dataset)} images.") # print(train_data_num)
print(f"Validation Dataset: {len(valid_dataset)} images.") # print(valid_data
print(f"Test Dataset: {len(test_dataset)} images.") # print(test_data_num)
#-- Logger --#
```

```
Train Dataset: 192 images.
Validation Dataset: 24 images.
Test Dataset: 24 images.
```

In [4]:

```
BATCH_SIZE = 8

train_loader = DataLoader(
    train_dataset,
    batch_size = BATCH_SIZE,
    shuffle = True,
    num_workers = 0
)

valid_loader = DataLoader(
    train_dataset,
    batch_size = BATCH_SIZE,
    shuffle = True,
    num_workers = 0
)
```

2. Define the model (ResNet18)

In [5]:

```
model = models.resnet18(pretrained=True)
model.fc = torch.nn.Linear(512, 2) # 2 for 'blocked' and 'free'

device = torch.device('cpu')
if torch.cuda.is_available():
    device = torch.device('cuda')
    print("This environment supports the CUDA.") # Logger
else:
    print("This environment does not support the CUDA.") # Logger
    print("The model will be running on the CPU instead.") # Logger
    # pass

model = model.to(device)

# print(model)
```

```
This environment supports the CUDA.
```

3. Train the model

In [6]:

```
if not os.path.isdir("./best_models"):
    os.mkdir("./best_models")

CURRENT_TIME = strftime('%Y%m%d_%H%M%S', localtime())
BEST_MODEL_PATH = f"./best_models/best_model_resnet18_{CURRENT_TIME}.pth"

# hyper parameters
EPOCHS = 30
LEARNING_RATE = 0.001
MOMENTUM = 0.9
```

```
L2_CONST = 1e-4

best_accuracy = 0.0 # validation accuracy

# SGD optimizer with L2 regularization
optimizer = optim.SGD(model.parameters(),
                      lr=LEARNING_RATE,
                      momentum=MOMENTUM,
                      weight_decay=L2_CONST)

accuracy_history = []

EPOCH_DIGIT = len(str(EPOCHS)) # for Logger
```

In [7]:

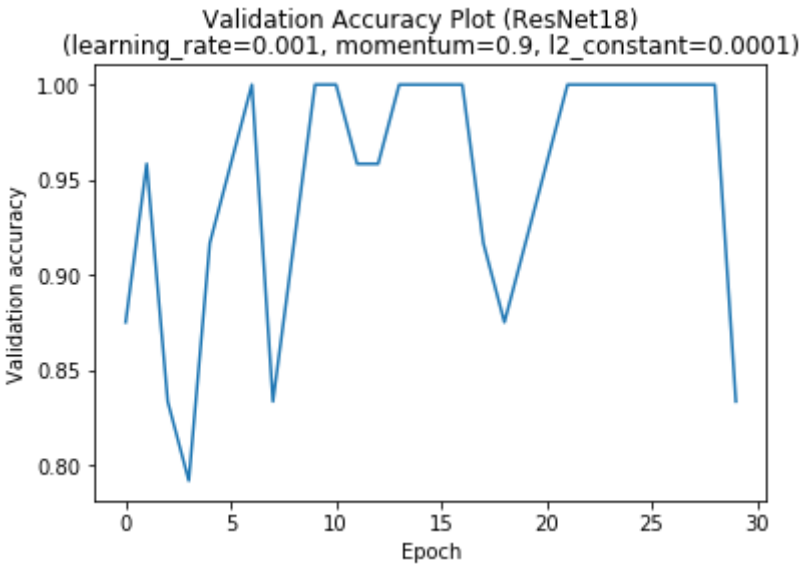
```
# model training loop
for epoch in range(EPOCHS):

    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = F.cross_entropy(outputs, labels)
        loss.backward()
        optimizer.step()

    valid_error = 0.0
    for images, labels in iter(valid_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        outputs = outputs.argmax(1)
        valid_error += float(torch.sum(torch.abs(labels - outputs)))

    valid_accuracy = 1.0 - (valid_error / valid_data_num)

    if valid_accuracy < 0:
        valid_accuracy = 0

    accuracy_history.append(valid_accuracy)

    print(f"[Epoch {epoch : >{EPOCH_DIGIT}d}] Validation accuracy: {valid_acc

    if valid_accuracy > best_accuracy:
        print("\tSave the best model") # Logger
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_accuracy = valid_accuracy

print("Training Complete!") # Logger
print(f"Best validation accuracy: {best_accuracy: .5f}") # Logger
```

```
[Epoch  0] Validation accuracy:  0.87500
        Save the best model
[Epoch  1] Validation accuracy:  0.95833
        Save the best model
[Epoch  2] Validation accuracy:  0.83333
[Epoch  3] Validation accuracy:  0.79167
[Epoch  4] Validation accuracy:  0.91667
[Epoch  5] Validation accuracy:  0.95833
[Epoch  6] Validation accuracy:  1.00000
        Save the best model
[Epoch  7] Validation accuracy:  0.83333
[Epoch  8] Validation accuracy:  0.91667
[Epoch  9] Validation accuracy:  1.00000
```

```
[Epoch 10] Validation accuracy:  1.00000
[Epoch 11] Validation accuracy:  0.95833
[Epoch 12] Validation accuracy:  0.95833
[Epoch 13] Validation accuracy:  1.00000
[Epoch 14] Validation accuracy:  1.00000
[Epoch 15] Validation accuracy:  1.00000
[Epoch 16] Validation accuracy:  1.00000
[Epoch 17] Validation accuracy:  0.91667
[Epoch 18] Validation accuracy:  0.87500
[Epoch 19] Validation accuracy:  0.91667
[Epoch 20] Validation accuracy:  0.95833
[Epoch 21] Validation accuracy:  1.00000
[Epoch 22] Validation accuracy:  1.00000
[Epoch 23] Validation accuracy:  1.00000
[Epoch 24] Validation accuracy:  1.00000
[Epoch 25] Validation accuracy:  1.00000
[Epoch 26] Validation accuracy:  1.00000
[Epoch 27] Validation accuracy:  1.00000
[Epoch 28] Validation accuracy:  1.00000
[Epoch 29] Validation accuracy:  0.83333
Training Complete!
Best validation accuracy:  1.00000
```

In [8]:

```python
if not os.path.isdir("./plots"):
    os.mkdir("./plots")

PLOT_PATH = f"./plots/validation_accuracy_plot_restnet18_{CURRENT_TIME}.png"

title = "Validation Accuracy Plot (ResNet18)"
subtitle = f"(learning_rate={LEARNING_RATE}, momentum={MOMENTUM}, l2_constant

plt.plot(accuracy_history)

plt.suptitle(title)
plt.title(subtitle)
plt.xlabel("Epoch")
plt.ylabel("Validation accuracy")

plt.savefig(PLOT_PATH)
plt.show()
```



4. Test the model

In [9]:

```python
model = models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 2)
```

```
print(f"Load model from \"{BEST_MODEL_PATH}\".") # Logger

model.load_state_dict(torch.load(BEST_MODEL_PATH))

model = model.to(device)
model = model.eval().half()
```

```
Load model from "./best_models/best_model_resnet18_20210204_221516.pth".
```

In [10]:

```
test_loader = DataLoader(
    test_dataset,
    batch_size=1,
    shuffle=True,
    num_workers=0
)
```

In [11]:

```
correct_case_count = 0

for case, sample in enumerate(iter(test_loader)):
    image, label = sample
    image = image.to(device).half()
    label = int(label)
    predict = model(image)
    predict = F.softmax(predict, dim=1)
    predict = predict.flatten()

    #-- Logger --#
    print(f"[Test Case {case}]")
    print(f"\t[Prediction] {float(predict[0]): .5f} : {float(predict[1]): .5f
    # print(f"\t[Prediction] Blocked : Free")
    print(f"\t[Real output] {label}") # 0: Blocked, 1: Free
    #-- Logger --#

    if label == 1 and float(predict[0]) < float(predict[1]):
        correct_case_count += 1
        print(f"\t[Result] Correct") # Logger
    elif label == 0 and float(predict[0]) > float(predict[1]):
        correct_case_count += 1
        print(f"\t[Result] Correct") # Logger
    else:
        print(f"\t[Result] Incorrect") # Logger
        # pass

print(f"[Total Test Accuracy] {correct_case_count/test_data_num : .5f}")
```

```
[Test Case 0]
        [Prediction]  0.99756 :  0.00250
        [Real output] 0
        [Result] Correct
[Test Case 1]
        [Prediction]  0.00004 :  1.00000
        [Real output] 1
        [Result] Correct
[Test Case 2]
        [Prediction]  0.00037 :  0.99951
        [Real output] 1
        [Result] Correct
[Test Case 3]
        [Prediction]  1.00000 :  0.00008
        [Real output] 0
        [Result] Correct
[Test Case 4]
        [Prediction]  0.99756 :  0.00252
```

```
        [Real output] 0
        [Result] Correct
[Test Case 5]
        [Prediction]  0.00010 :  1.00000
        [Real output] 1
        [Result] Correct
[Test Case 6]
        [Prediction]  0.00001 :  1.00000
        [Real output] 1
        [Result] Correct
[Test Case 7]
        [Prediction]  0.00002 :  1.00000
        [Real output] 1
        [Result] Correct
[Test Case 8]
        [Prediction]  0.00004 :  1.00000
        [Real output] 1
        [Result] Correct
[Test Case 9]
        [Prediction]  1.00000 :  0.00009
        [Real output] 0
        [Result] Correct
[Test Case 10]
        [Prediction]  0.00070 :  0.99951
        [Real output] 1
        [Result] Correct
[Test Case 11]
        [Prediction]  0.00029 :  0.99951
        [Real output] 1
        [Result] Correct
[Test Case 12]
        [Prediction]  1.00000 :  0.00013
        [Real output] 0
        [Result] Correct
[Test Case 13]
        [Prediction]  0.00002 :  1.00000
        [Real output] 1
        [Result] Correct
[Test Case 14]
        [Prediction]  0.99854 :  0.00153
        [Real output] 0
        [Result] Correct
[Test Case 15]
        [Prediction]  0.99316 :  0.00674
        [Real output] 0
        [Result] Correct
[Test Case 16]
        [Prediction]  0.00002 :  1.00000
        [Real output] 1
        [Result] Correct
[Test Case 17]
        [Prediction]  0.00015 :  1.00000
        [Real output] 1
        [Result] Correct
[Test Case 18]
        [Prediction]  1.00000 :  0.00018
        [Real output] 0
        [Result] Correct
[Test Case 19]
        [Prediction]  0.00158 :  0.99854
        [Real output] 1
        [Result] Correct
[Test Case 20]
        [Prediction]  0.00109 :  0.99902
        [Real output] 1
        [Result] Correct
[Test Case 21]
        [Prediction]  1.00000 :  0.00013
        [Real output] 0
```

```
        [Result] Correct
[Test Case 22]
        [Prediction]  0.97656 :  0.02328
        [Real output] 0
        [Result] Correct
[Test Case 23]
        [Prediction]  0.99951 :  0.00042
        [Real output] 0
        [Result] Correct
[Total Test Accuracy]  1.00000
```