Collision Avoidance (DNN)

0. Import modules

In [1]:

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

import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder

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

1. Prepare the dataset

In [2]:

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

IMAGE_WIDTH  = 32
IMAGE_HEIGHT = 32
IMAGE_CHANNEL = 1
```

In [3]:

```
total_dataset = ImageFolder(
    DATASET_PATH,
    transforms.Compose([
        transforms.Resize((IMAGE_HEIGHT, IMAGE_WIDTH)),
        transforms.Grayscale(num_output_channels=IMAGE_CHANNEL),
        transforms.ToTensor(),
        transforms.Normalize([0.449], [0.226]),
        transforms.Lambda(lambda img: torch.flatten(img)) # https://stackove
    ])
)

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

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

In [4]:

```
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
```

```python
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 [5]:

```python
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 (DNN)

In [6]:

```python
INPUT_SIZE = IMAGE_HEIGHT * IMAGE_WIDTH * IMAGE_CHANNEL

class DNN(nn.Module):

    """Custom DNN model for the Image classification."""

    __slots__ = "__model"

    def __init__(self, input_dim=INPUT_SIZE, output_dim=2, hidden_dims=(128,

        super(DNN, self).__init__()

        dims_list = (input_dim, *hidden_dims)
        model_components = []

        # hidden layers
        for i in range(1, len(dims_list)):
            current_input_dim = dims_list[i-1]
            current_output_dim = dims_list[i]
            model_components.append(nn.Linear(current_input_dim, current_outp

            if do_batch_normal == True:
                model_components.append(nn.BatchNorm1d(current_output_dim))

            model_components.append(nn.ReLU())

            if dropout > 0:
                model_components.append(nn.Dropout(dropout))

        # output layer
        output_layer = nn.Linear(dims_list[-1], output_dim)
        model_components.append(output_layer)
        model_components.append(nn.Softmax(dim=1))

        # make DNN model
        self.__model = nn.Sequential(*model_components)
```

```
    def forward(self, x):
        return self.__model(x)
```

3. Train the model

In [7]:

```
model = DNN(hidden_dims=(128, 64, 32))

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.
```

In [8]:

```
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_dnn_{CURRENT_TIME}.pth"

# hyper parameters
EPOCHS = 30
LEARNING_RATE = 0.001
MOMENTUM = 0.9
L2_CONST = 1e-4

best_accuracy = 0.0 # validation accuracy

criterion = nn.CrossEntropyLoss()

# 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 [9]:

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

    model.train()

    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
```

```python
        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)
        valid_error += float(torch.sum(torch.abs(labels - outputs.argmax(1))))

    valid_accuracy = 1.0 - float(valid_error) / float(valid_data_num)

    if valid_accuracy < 0:
        valid_accuracy = 0

    accuracy_history.append(valid_accuracy)

    print(f"[Epoch {epoch: >{EPOCH_DIGIT}d}] Accuracy: {valid_accuracy: .5f}'

    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] Accuracy:  0.00000
[Epoch  1] Accuracy:  0.58333
        Save the best model
[Epoch  2] Accuracy:  0.66667
        Save the best model
[Epoch  3] Accuracy:  0.83333
        Save the best model
[Epoch  4] Accuracy:  0.62500
[Epoch  5] Accuracy:  0.87500
        Save the best model
[Epoch  6] Accuracy:  0.62500
[Epoch  7] Accuracy:  0.75000
[Epoch  8] Accuracy:  0.87500
[Epoch  9] Accuracy:  0.95833
        Save the best model
[Epoch 10] Accuracy:  0.79167
[Epoch 11] Accuracy:  0.91667
[Epoch 12] Accuracy:  0.75000
[Epoch 13] Accuracy:  0.83333
[Epoch 14] Accuracy:  0.95833
[Epoch 15] Accuracy:  0.95833
[Epoch 16] Accuracy:  0.87500
[Epoch 17] Accuracy:  0.91667
[Epoch 18] Accuracy:  0.79167
[Epoch 19] Accuracy:  1.00000
        Save the best model
[Epoch 20] Accuracy:  1.00000
[Epoch 21] Accuracy:  0.79167
[Epoch 22] Accuracy:  0.91667
[Epoch 23] Accuracy:  0.95833
[Epoch 24] Accuracy:  0.79167
[Epoch 25] Accuracy:  1.00000
[Epoch 26] Accuracy:  1.00000
[Epoch 27] Accuracy:  0.91667
[Epoch 28] Accuracy:  0.87500
[Epoch 29] Accuracy:  0.87500
Training Complete!
Best validation accuracy:  1.00000
```

In [10]:

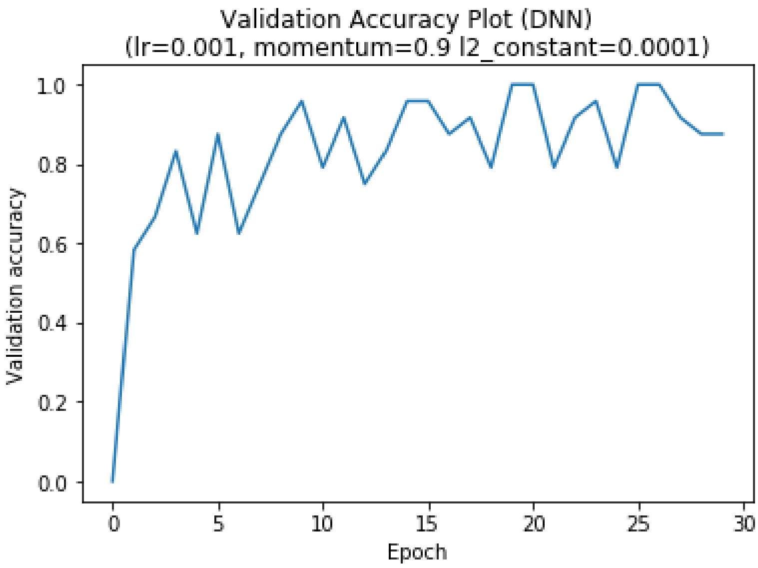
```
if not os.path.isdir("./plots"):
    os.mkdir("./plots")

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

title = "Validation Accuracy Plot (DNN)"
subtitle = f"(lr={LEARNING_RATE}, momentum={MOMENTUM} l2_constant={L2_CONST})

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 [11]:

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

In [12]:

```
correct_case_count = 0

model.eval()

for case, sample in enumerate(iter(test_loader)):
    image, label = sample
    image = image.to(device)
    label = int(label)
    predict = model(image)
    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
```

```python
    #-- 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]  0.87057 :  0.12943
	[Real output] 0
	[Result] Correct
[Test Case 1]
	[Prediction]  0.97369 :  0.02631
	[Real output] 0
	[Result] Correct
[Test Case 2]
	[Prediction]  0.31342 :  0.68658
	[Real output] 0
	[Result] Incorrect
[Test Case 3]
	[Prediction]  0.00893 :  0.99107
	[Real output] 1
	[Result] Correct
[Test Case 4]
	[Prediction]  0.68830 :  0.31170
	[Real output] 0
	[Result] Correct
[Test Case 5]
	[Prediction]  0.01415 :  0.98585
	[Real output] 1
	[Result] Correct
[Test Case 6]
	[Prediction]  0.88827 :  0.11173
	[Real output] 0
	[Result] Correct
[Test Case 7]
	[Prediction]  0.00900 :  0.99100
	[Real output] 1
	[Result] Correct
[Test Case 8]
	[Prediction]  0.01733 :  0.98267
	[Real output] 1
	[Result] Correct
[Test Case 9]
	[Prediction]  0.61169 :  0.38831
	[Real output] 0
	[Result] Correct
[Test Case 10]
	[Prediction]  0.03434 :  0.96566
	[Real output] 1
	[Result] Correct
[Test Case 11]
	[Prediction]  0.01664 :  0.98336
	[Real output] 1
	[Result] Correct
[Test Case 12]
	[Prediction]  0.03614 :  0.96386
	[Real output] 1
	[Result] Correct
[Test Case 13]
	[Prediction]  0.92288 :  0.07712
```

```
        [Real output] 0
        [Result] Correct
[Test Case 14]
        [Prediction]  0.97228 :  0.02772
        [Real output] 0
        [Result] Correct
[Test Case 15]
        [Prediction]  0.86470 :  0.13530
        [Real output] 0
        [Result] Correct
[Test Case 16]
        [Prediction]  0.00734 :  0.99266
        [Real output] 1
        [Result] Correct
[Test Case 17]
        [Prediction]  0.04791 :  0.95209
        [Real output] 1
        [Result] Correct
[Test Case 18]
        [Prediction]  0.01627 :  0.98373
        [Real output] 1
        [Result] Correct
[Test Case 19]
        [Prediction]  0.94554 :  0.05446
        [Real output] 0
        [Result] Correct
[Test Case 20]
        [Prediction]  0.94354 :  0.05646
        [Real output] 0
        [Result] Correct
[Test Case 21]
        [Prediction]  0.02641 :  0.97359
        [Real output] 1
        [Result] Correct
[Test Case 22]
        [Prediction]  0.93535 :  0.06465
        [Real output] 0
        [Result] Correct
[Test Case 23]
        [Prediction]  0.91091 :  0.08909
        [Real output] 0
        [Result] Correct
[Total Test Accuracy]  0.95833
```