Loading and Wrangling Data

Loading in data using the functions that we made:

In [67]:
```
oneBatch = False
```

In [68]:
```
# Run if you want one batch
import data_loading as dt
import os
import gdown
import logging
from zipfile import ZipFile
import os
import numpy as np
from multiprocessing import Pool
from pathlib import Path
import concurrent.futures
import cv2

idArrays, imageArrays = [], []
for num in range(13):
    print(num)
    path = f"./batch{num}.zip"

    # We import images into np.arrays
    newPath = path.replace(".zip", f"/part_{num}")
    ids, images = dt.importImages(newPath)

    # Adding to arrays
    idArrays.append(ids)
    imageArrays.append(images)


totalIds = np.concatenate(idArrays) if len(idArrays) > 1 else idArrays[0]
totalImages = (
    np.concatenate(imageArrays) if len(imageArrays) > 1 else imageArrays[0]
    )

ids = totalIds
images= totalImages
```

```
0
1
2
3
4
5
6
7
8
9
10
```

Getting annotations and getting them into the correct order:

In [69]:
```
import pandas as pd
annotations = pd.read_csv("data/annotations.csv")
annotations.head()
```

Out[69]:

	position	image
0	standing	1
1	standing	2
2	standing	3
3	standing	4
4	standing	5

In [70]:
```
position_maps= {"standing": 0,
                "takedown1": 1,
                "takedown2": 2,
                "open_guard1": 3,
                "open_guard2": 4,
                "half_guard1": 5,
                "half_guard2": 6,
                "closed_guard1": 7,
                "closed_guard2": 8,
                "5050_guard": 9,
                "mount1": 10,
                "mount2": 11,
                "back1": 12,
                "back2": 13,
                "turtle1": 14,
                "turtle2": 15,
                "side_control1" : 16,
                "side_control2" : 17}


labels = []
for id in ids:
    labels.append(position_maps[annotations[annotations['image'] == id]['position']
```

In [71]:
```
labels[0:20]
```

Out[71]:
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Converting images to tensors:

In [72]:
```
import torch
import numpy as np

# Converting into torch tensors
```

```
for i, img in enumerate(images):
    images[i] = torch.from_numpy(np.array(img))
```

Convolutional Neural Net

Creating Neural Net

In [73]:
```
# Data argumentation
from torchvision import transforms
data_transforms = transforms.Compose([
        transforms.GaussianBlur(kernel_size=(3,3), sigma=(0.1, 5)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(degrees=(0, 180))
])

# CNN
from torch import nn
import torchvision
class my_net(nn.Module):

    ## Constructor commands
    def __init__(self):
        super(my_net, self).__init__()

        ## Define architecture
        self.conv_stack = nn.Sequential(
            nn.Conv2d(3,8,3,1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.Conv2d(8,16,2,1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.Conv2d(16,32,3,1),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(5408, 200),
            nn.ReLU(),
            nn.Linear(200, 18)
        )

    ## Function to generate predictions
    def forward(self, x):
        scores = self.conv_stack(x)
        return scores
```

Training Neural Net

Train-test split:

In [74]:
```
from sklearn.model_selection import train_test_split
```

```
train_X, test_X, train_y, test_y = train_test_split(
    images, labels, test_size=0.25, random_state=42)

train_X = torch.from_numpy(train_X)
train_X = torch.movedim(train_X, source=3, destination=1)

test_X = torch.from_numpy(test_X)
test_X = torch.movedim(test_X, source=3, destination=1)
```

In [75]:

```
import torch
import torch.nn as nn
import numpy as np
from torch.utils.data import DataLoader, TensorDataset

# Hyperparameters
epochs = 300
lrate = 0.001
bsize = 32

# For reproducibility
torch.manual_seed(3)

# Cost Function
cost_fn = nn.CrossEntropyLoss()

# Initialize the model
net = my_net()

# Optimizer (Stochastic Gradient Descent)
optimizer = torch.optim.SGD(net.parameters(), lr=lrate)

# Make DataLoader
y_tensor = torch.Tensor(train_y)
train_loader = DataLoader(TensorDataset(train_X.type(torch.FloatTensor),
                                        y_tensor.type(torch.LongTensor)), batch_siz

# Training loop
track_cost = np.zeros(epochs)

for epoch in range(epochs):
    cur_cost = 0.0

    for i, (inputs, labels) in enumerate(train_loader):
        # Transform the input data using our data augmentation strategies
        inputs = data_transforms(inputs)

        # Forward, backward, and optimize
        optimizer.zero_grad()
        outputs = net(inputs)
        cost = cost_fn(outputs, labels)  # CrossEntropyLoss already applies Softmax
        cost.backward()
        optimizer.step()

        cur_cost += cost.item()

    # Store the accumulated cost at each epoch
```

```
    track_cost[epoch] = cur_cost
    print(epoch/epochs)
    #print(f"Epoch: {epoch} Cost: {cur_cost}")
```

```
0.0
0.0033333333333333335
0.006666666666666667
0.01
0.013333333333333334
0.016666666666666666
0.02
0.023333333333333334
0.026666666666666667
0.03
0.03333333333333333
0.03666666666666667
0.04
0.043333333333333335
0.04666666666666667
0.05
0.05333333333333334
0.056666666666666664
0.06
0.06333333333333334
0.06666666666666667
0.07
0.07333333333333333
0.07666666666666666
0.08
0.08333333333333333
0.08666666666666667
0.09
0.09333333333333334
0.09666666666666666
0.1
0.10333333333333333
0.10666666666666667
0.11
0.11333333333333333
0.11666666666666667
0.12
0.12333333333333334
0.12666666666666668
0.13
0.13333333333333333
0.13666666666666666
0.14
0.14333333333333334
0.14666666666666667
0.15
0.15333333333333332
0.15666666666666668
0.16
0.16333333333333333
0.16666666666666666
0.17
```

0.17333333333333334
0.17666666666666667
0.18
0.18333333333333332
0.18666666666666668
0.19
0.19333333333333333
0.19666666666666666
0.2
0.20333333333333334
0.20666666666666667
0.21
0.21333333333333335
0.21666666666666667
0.22
0.22333333333333333
0.22666666666666666
0.23
0.23333333333333334
0.23666666666666666
0.24
0.24333333333333335
0.24666666666666667
0.25
0.25333333333333335
0.25666666666666665
0.26
0.2633333333333333
0.26666666666666666
0.27
0.2733333333333333
0.27666666666666667
0.28
0.2833333333333333
0.2866666666666667
0.29
0.29333333333333333
0.2966666666666667
0.3
0.30333333333333334
0.30666666666666664
0.31
0.31333333333333335
0.31666666666666665
0.32
0.3233333333333333
0.32666666666666666
0.33
0.3333333333333333
0.33666666666666667
0.34
0.3433333333333333
0.3466666666666667
0.35
0.35333333333333333
0.3566666666666667

0.36
0.36333333333333334
0.36666666666666664
0.37
0.37333333333333335
0.37666666666666665
0.38
0.38333333333333336
0.38666666666666666
0.39
0.3933333333333333
0.39666666666666667
0.4
0.4033333333333333
0.4066666666666667
0.41
0.41333333333333333
0.4166666666666667
0.42
0.42333333333333334
0.4266666666666667
0.43
0.43333333333333335
0.43666666666666665
0.44
0.44333333333333336
0.44666666666666666
0.45
0.4533333333333333
0.45666666666666667
0.46
0.4633333333333333
0.4666666666666667
0.47
0.47333333333333333
0.4766666666666667
0.48
0.48333333333333334
0.4866666666666667
0.49
0.49333333333333335
0.49666666666666665
0.5
0.5033333333333333
0.5066666666666667
0.51
0.5133333333333333
0.5166666666666667
0.52
0.5233333333333333
0.5266666666666666
0.53
0.5333333333333333
0.5366666666666666
0.54
0.5433333333333333

0.5466666666666666
0.55
0.5533333333333333
0.5566666666666666
0.56
0.5633333333333334
0.5666666666666667
0.57
0.5733333333333334
0.5766666666666667
0.58
0.5833333333333334
0.5866666666666667
0.59
0.5933333333333334
0.5966666666666667
0.6
0.6033333333333334
0.6066666666666667
0.61
0.6133333333333333
0.6166666666666667
0.62
0.6233333333333333
0.6266666666666667
0.63
0.6333333333333333
0.6366666666666667
0.64
0.6433333333333333
0.6466666666666666
0.65
0.6533333333333333
0.6566666666666666
0.66
0.6633333333333333
0.6666666666666666
0.67
0.6733333333333333
0.6766666666666666
0.68
0.6833333333333333
0.6866666666666666
0.69
0.6933333333333334
0.6966666666666667
0.7
0.7033333333333334
0.7066666666666667
0.71
0.7133333333333334
0.7166666666666667
0.72
0.7233333333333334
0.7266666666666667
0.73

0.7333333333333333
0.7366666666666667
0.74
0.7433333333333333
0.7466666666666667
0.75
0.7533333333333333
0.7566666666666667
0.76
0.7633333333333333
0.7666666666666667
0.77
0.7733333333333333
0.7766666666666666
0.78
0.7833333333333333
0.7866666666666666
0.79
0.7933333333333333
0.7966666666666666
0.8
0.8033333333333333
0.8066666666666666
0.81
0.8133333333333334
0.8166666666666667
0.82
0.8233333333333334
0.8266666666666667
0.83
0.8333333333333334
0.8366666666666667
0.84
0.8433333333333334
0.8466666666666667
0.85
0.8533333333333334
0.8566666666666667
0.86
0.8633333333333333
0.8666666666666667
0.87
0.8733333333333333
0.8766666666666667
0.88
0.8833333333333333
0.8866666666666667
0.89
0.8933333333333333
0.8966666666666666
0.9
0.9033333333333333
0.9066666666666666
0.91
0.9133333333333333
0.9166666666666666

```
0.92
0.9233333333333333
0.9266666666666666
0.93
0.9333333333333333
0.9366666666666666
0.94
0.9433333333333334
0.9466666666666667
0.95
0.9533333333333334
0.9566666666666667
0.96
0.9633333333333334
0.9666666666666667
0.97
0.9733333333333334
0.9766666666666667
0.98
0.9833333333333333
0.9866666666666667
0.99
0.9933333333333333
0.9966666666666667
```

Calculating training accuracy:

In [76]:

```
filename = "cnn_medium"
f = open(filename + '.txt', "a")

## Initialize objects for counting correct/total
correct = 0
total = 0

# Specify no changes to the gradient in the subsequent steps (since we're not using
with torch.no_grad():
    for data in train_loader:
        # Current batch of data
        images, labels = data

        # pass each batch into the network
        outputs = net(images)

        # the class with the maximum score is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)

        # add size of the current batch
        total += labels.size(0)

        # add the number of correct predictions in the current batch
        correct += (predicted == labels).sum().item()

## Calculate and print the proportion correct
print(f"Training Accuracy is {correct/total}")
f.write(f"Training Accuracy is {correct/total}")
```

```
Training Accuracy is 0.9743041159972952
```

Out[76]:
```
39
```

Calculating testing accuracy:

In [77]:
```
## Combine X and y tensors into a TensorDataset and DataLoader
test_loader = DataLoader(TensorDataset(test_X.type(torch.FloatTensor),
                         torch.Tensor(test_y).type(torch.LongTensor)), batch_size=bs

## Initialize objects for counting correct/total
correct = 0
total = 0

# Specify no changes to the gradient in the subsequent steps (since we're not using
with torch.no_grad():
    for data in test_loader:
        # Current batch of data
        images, labels = data

        # pass each batch into the network
        outputs = net(images)

        # the class with the maximum score is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)

        # add size of the current batch
        total += labels.size(0)

        # add the number of correct predictions in the current batch
        correct += (predicted == labels).sum().item()

## Calculate and print the proportion correct
print(f"Test Accuracy is {correct/total}")
f.write(f"\nTest Accuracy is {correct/total}")

f.close()
```

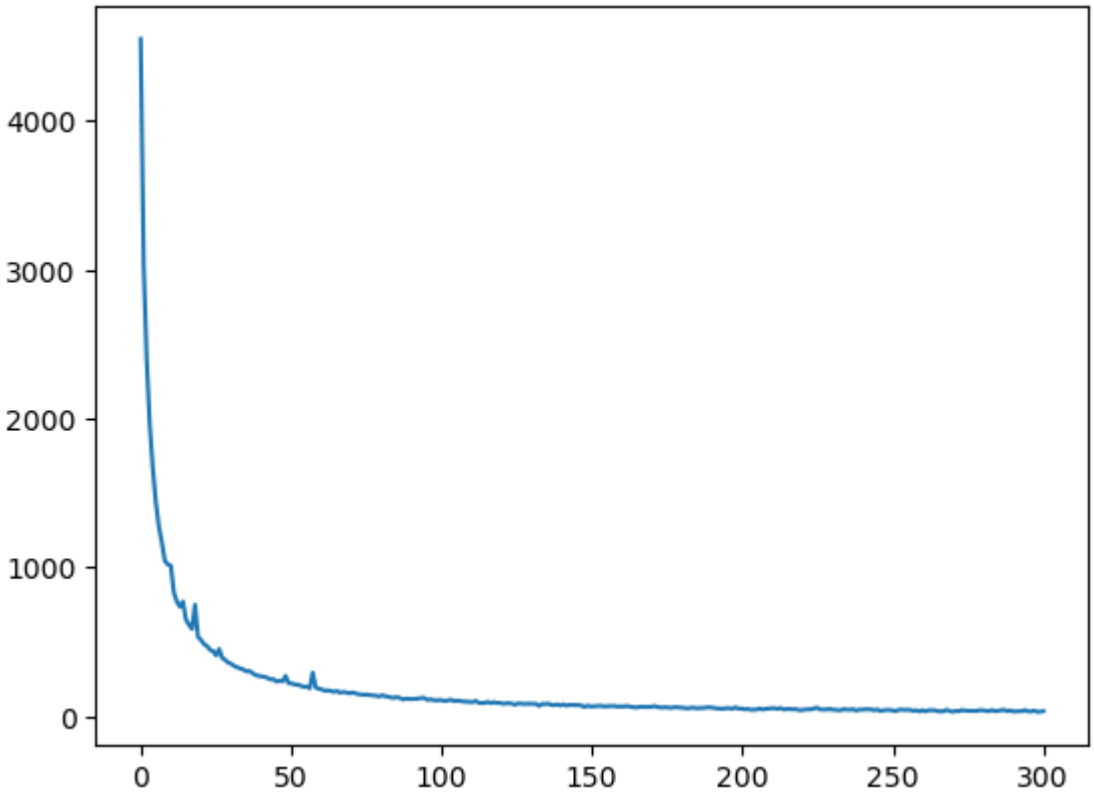
```
Test Accuracy is 0.9666777519122048
```

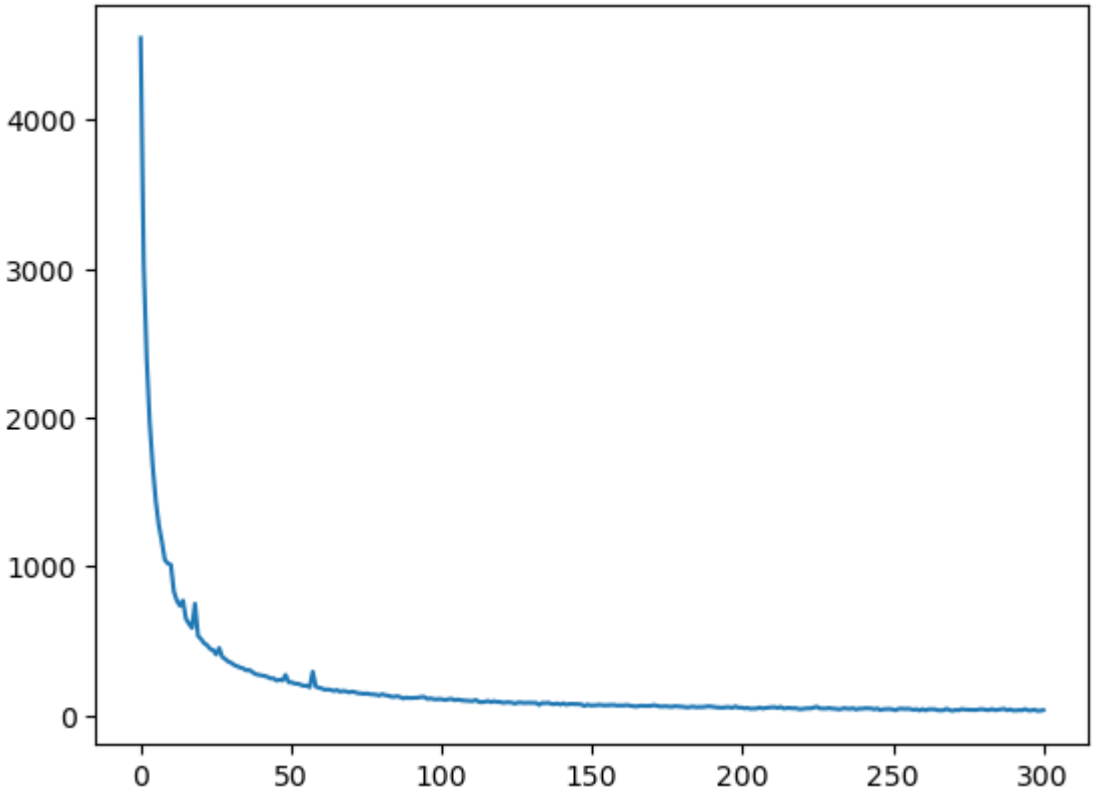
In [78]:
```
set(train_y)
```

Out[78]:
```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17}
```

In [79]:
```
# verifying the convergence of cost
import matplotlib.pyplot as plt
plt.plot(np.linspace(0, epochs, epochs), track_cost)
plt.show()

plt.plot(np.linspace(0, epochs, epochs), track_cost)
plt.savefig('cnn_medium_cost_plot.png')
```





In [80]:
```
torch.save(net.state_dict(), 'model1_weights.pth')
```