Coding_Challenge_for_Fatima_Fellowship

April 1, 2022

1 Fatima Fellowship Quick Coding Challenge (Pick 1)

Thank you for applying to the Fatima Fellowship. To help us select the Fellows and assess your ability to do machine learning research, we are asking that you complete a short coding challenge. Please pick **1 of these 5** coding challenges, whichever is most aligned with your interests.

Due date: 1 week

How to submit: Please make a copy of this colab notebook, add your code and results, and submit your colab notebook to the submission link below. If you have never used a colab notebook, check out this video.

Submission link: https://airtable.com/shrXy3QKSsO2yALd3

2 1. Deep Learning for Vision

Upside down detector: Train a model to detect if images are upside down

- Pick a dataset of natural images (we suggest looking at datasets on the Hugging Face Hub)
- Synthetically turn some of images upside down. Create a training and test set.
- Build a neural network (using Tensorflow, PyTorch, or any framework you like)
- Train it to classify image orientation until a reasonable accuracy is reached
- Upload the the model to the Hugging Face Hub, and add a link to your model below.
- Look at some of the images that were classified incorrectly. Please explain what you might do to improve your model's performance on these images in the future (you do not need to impelement these suggestions)

Submission instructions: Please write your code below and include some examples of images that were classified

3 Imports

[1]:
```
### WRITE YOUR CODE TO TRAIN THE MODEL HERE
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import torchvision
```

```
import torchvision.transforms as T
import torchvision.transforms.functional as TF

from datasets import load_dataset

import matplotlib.pyplot as plt
import numpy as np
np.random.seed(7)
import cv2 as cv
from tqdm import tqdm
```

4 Helper functions

[2]:
```
def show_image(img, title='', ctype='rgb'):
    plt.figure(figsize=(10, 10))
    img = np.transpose(img, (1, 2, 0))
    if ctype == 'bgr':
        b, g, r = cv.split(img)
        rgb_img = cv.merge([r, g, b])
        plt.imshow(rgb_img)
    elif ctype == 'hsv':
        rgb = cv.cvtColor(img, cv.COLOR_HSV2RGB)
        plt.imshow(rgb)
    elif ctype == 'gray':
        plt.imshow(img, cmap='gray')
    elif ctype == 'rgb':
        plt.imshow(img)
    else:
        raise Exception("Unknown colour type")
    plt.title(title)
    plt.show()
```

5 Load dataset

5.1 Pick a dataset of natural images

[3]:
```
dataset = load_dataset('cifar10')
```

```
Reusing dataset cifar10 (/home/cis/.cache/huggingface/datasets/cifar10/plain_tex
t/1.0.0/447d6ec4733dddd1ce3bb577c7166b986eaa4c538dcd9e805ba61f35674a9de4)
```

```
  0%|          | 0/2 [00:00<?, ?it/s]
```

5.2 Synthetically turn some of images upside down

[4]:
```
classes = ('normal', 'flipped')
batch_size = 16
transform = T.Compose(
    [T.ToTensor(),
     T.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

for split in ['train', 'test']:
    images = dataset[split]['img']
    labels = dataset[split]['label']
    for i in tqdm(range(len(images))):
        img = transform(images[i])
        if np.random.rand(1) > 0.5:
            images[i] = TF.vflip(img)
            labels[i] = 1
        else:
            images[i] = img
            labels[i] = 0
    split_dataset =  list(zip(images, labels))
    if split == 'train':
        trainloader = torch.utils.data.DataLoader(split_dataset,␣
 ↪batch_size=batch_size)
    else:
        testloader = torch.utils.data.DataLoader(split_dataset, batch_size=1)
```

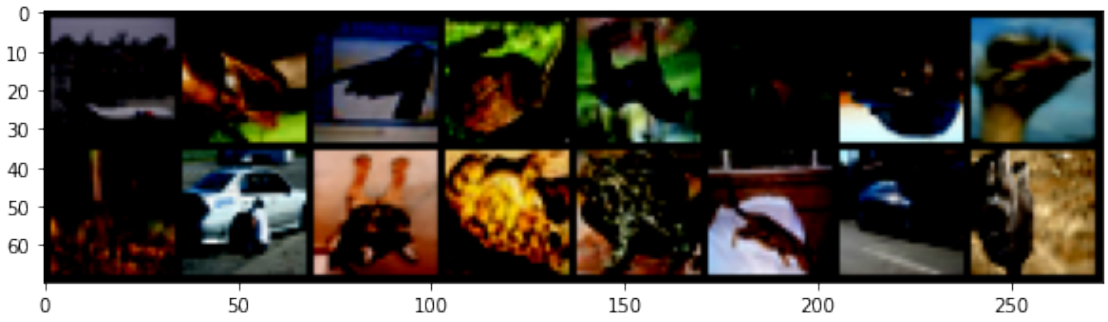
```
100%|
                  | 50000/50000 [00:09<00:00, 5336.27it/s]
100%|
                  | 10000/10000 [00:01<00:00, 5504.90it/s]
```

6 Visualization

[5]:
```
# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()
print(f'images.shape = {images.shape}')
show_image(torchvision.utils.make_grid(images))
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
images.shape = torch.Size([16, 3, 32, 32])
```



7 Build the model

[6]:
```
# Build a neural network
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 8, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(8, 16, 5)

        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

8 Training

[7]:
```
model = Model()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())
```

[8]:
```
epochs = 50
for epoch in range(epochs):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
```

```
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.5f}')
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 0.56227
[2,  2000] loss: 0.48812
[3,  2000] loss: 0.45425
[4,  2000] loss: 0.42685
[5,  2000] loss: 0.40361
[6,  2000] loss: 0.37881
[7,  2000] loss: 0.35889
[8,  2000] loss: 0.33771
[9,  2000] loss: 0.31993
[10,  2000] loss: 0.29883
[11,  2000] loss: 0.28178
[12,  2000] loss: 0.26565
[13,  2000] loss: 0.25455
[14,  2000] loss: 0.23797
[15,  2000] loss: 0.22605
[16,  2000] loss: 0.22094
[17,  2000] loss: 0.21007
[18,  2000] loss: 0.20093
[19,  2000] loss: 0.18976
[20,  2000] loss: 0.18003
[21,  2000] loss: 0.17212
[22,  2000] loss: 0.16658
[23,  2000] loss: 0.15356
[24,  2000] loss: 0.15458
[25,  2000] loss: 0.14523
[26,  2000] loss: 0.14522
[27,  2000] loss: 0.13753
```

```
[28,  2000] loss: 0.13043
[29,  2000] loss: 0.12817
[30,  2000] loss: 0.12139
[31,  2000] loss: 0.12434
[32,  2000] loss: 0.11681
[33,  2000] loss: 0.11471
[34,  2000] loss: 0.11297
[35,  2000] loss: 0.10869
[36,  2000] loss: 0.10742
[37,  2000] loss: 0.10433
[38,  2000] loss: 0.09798
[39,  2000] loss: 0.10139
[40,  2000] loss: 0.10111
[41,  2000] loss: 0.09679
[42,  2000] loss: 0.09670
[43,  2000] loss: 0.08554
[44,  2000] loss: 0.09479
[45,  2000] loss: 0.09182
[46,  2000] loss: 0.08623
[47,  2000] loss: 0.07728
[48,  2000] loss: 0.08316
[49,  2000] loss: 0.08408
[50,  2000] loss: 0.07680
Finished Training
```

[9]:
```
torch.save(model.state_dict(), './up_down_net.pth')
```

9 Test

[10]:
```
dataiter = iter(testloader)
images, labels = dataiter.next()
outputs = model(images)
_, predicted = torch.max(outputs, 1)

print('GroundTruth:', classes[labels[0]])
print('Predicted:', classes[predicted[0]])
```

```
GroundTruth: normal
Predicted: flipped
```

[11]:
```
correct = 0
total = 0
error_images = list()

with torch.no_grad():
    for i, data in enumerate(testloader):
        images, labels = data
```

```
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        if predicted != labels:
            error_images.append(i)

print(f'Accuracy of the network on the 10000 test images: {100 * correct //␣
 ↪total} %')
```

```
Accuracy of the network on the 10000 test images: 74 %
```

10 Error exploration

[12]:
```
idxes = np.random.choice(error_images, 16)
images = list()
convert_tensor = T.ToTensor()

for idx in idxes:
    images.append(convert_tensor(dataset['test']['img'][idx]))

show_image(torchvision.utils.make_grid(images))
```

Write up: * Link to the model on Hugging Face Hub: - Model link * Include some examples of misclassified images. Please explain what you might do to improve your model's performance on these images in the future (you do not need to impelement these suggestions) 1. We can increase the model complexity by implementing a deeper model 2. We can infer from the error images that the model has somehow inferred a relationship between the white pixels (sky for example) and the image orientation so to overcome this problem we can do data augmentation in a way that the data has white pixels in places other that the top or bottom (like color jitter).

[ ]:
```

```

11 2. Deep Learning for NLP

Fake news classifier: Train a text classification model to detect fake news articles!

- Download the dataset here: https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset
- Develop an NLP model for classification that uses a pretrained language model
- Finetune your model on the dataset, and generate an AUC curve of your model on the test set of your choice.
- Upload the the model to the Hugging Face Hub, and add a link to your model below.
- *Answer the following question*: Look at some of the news articles that were classified incorrectly. Please explain what you might do to improve your model's performance on these news articles in the future (you do not need to impelement these suggestions)

[13]:
```
### WRITE YOUR CODE TO TRAIN THE MODEL HERE
```

Write up: * Link to the model on Hugging Face Hub: * Include some examples of misclassified news articles. Please explain what you might do to improve your model's performance on these news articles in the future (you do not need to impelement these suggestions)

12 3. Deep RL / Robotics

RL for Classical Control: Using any of the classical control environments from OpenAI's `gym`, implement a deep NN that learns an optimal policy which maximizes the reward of the environment.

- Describe the NN you implemented and the behavior you observe from the agent as the model converges (or diverges).
- Plot the reward as a function of steps (or Epochs). Compare your results to a random agent.
- Discuss whether you think your model has learned the optimal policy and potential methods for improving it and/or where it might fail.
- (Optional) Upload the the model to the Hugging Face Hub, and add a link to your model below.

You may use any frameworks you like, but you must implement your NN on your own (no predefined/trained models like `stable_baselines`).

You may use any simulator other than `gym` *however*: * The environment has to be similar to the classical control environments (or more complex like `robosuite`). * You cannot choose a game/Atari/text based environment. The purpose of this challenge is to demonstrate an understanding of basic kinematic/dynamic systems.

[14]:
```
### WRITE YOUR CODE TO TRAIN THE MODEL HERE
```

Write up: * (Optional) link to the model on Hugging Face Hub: * Discuss whether you think your model has learned the optimal policy and potential methods for improving it and/or where it might fail.

13 4. Theory / Linear Algebra

Implement Contrastive PCA Read this paper and implement contrastive PCA in Python.

- First, please discuss what kind of dataset this would make sense to use this method on
- Implement the method in Python (do not use previous implementations of the method if they already exist)
- Then create a synthetic dataset and apply the method to the synthetic data. Compare with standard PCA.

Write up: Discuss what kind of dataset it would make sense to use Contrastive PCA

[15]:
```
### WRITE YOUR CODE HERE
```

14 5. Systems

Inference on the edge: Measure the inference times in various computationally-constrained settings

- Pick a few different speech detection models (we suggest looking at models on the Hugging Face Hub)
- Simulate different memory constraints and CPU allocations that are realistic for edge devices that might run such models, such as smart speakers or microcontrollers, and measure what is the average inference time of the models under these conditions
- How does the inference time vary with (1) choice of model (2) available system memory (3) available CPU (4) size of input?

Are there any surprising discoveries? (Note that this coding challenge is fairly open-ended, so we will be considering the amount of effort invested in discovering something interesting here).

[16]:
```
### WRITE YOUR CODE HERE
```

Write up: What surprising discoveries do you see?