```
 1 import torch
 2 from torch import nn
 3
 4 import torchvision
 5 from torchvision import datasets
 6 from torchvision.transforms import ToTensor
 7 import torch.nn.functional as F
 8 import matplotlib.pyplot as plt
 9 import numpy as np
10 import pandas as pd
11 import matplotlib.pyplot as plt
12 import zipfile
13 import os
14
15 import numpy as np
16 import pandas as pd
17 import torch.optim as optim
18 from torch.utils.data import Dataset
19 import time
20
21 class CustomDataset(Dataset):
22     def __init__(self, dataframe, transform=None):
23         self.dataframe = dataframe
24         self.transform = transform
25
26     def __len__(self):
27         return len(self.dataframe)
28
29     def __getitem__(self, idx):
30         img_values = self.dataframe.iloc[idx, 1:].
   values
31         img = img_values.reshape(28, 28).astype('
   float32')
32         label = int(self.dataframe.iloc[idx, 0])
33
34         if self.transform:
35             img = self.transform(img)
36
37         return img, label
38
39
```

```
train_data = pd.read_csv('sign_mnist_train.csv')
test_data = pd.read_csv('sign_mnist_test.csv')
print(train_data)

print(train_data.shape)

print(train_data.head())

column_names = train_data.columns

print(column_names)

# Access the first row using iloc (index location)
first_row = train_data.iloc[0]

# Extract the pixel values (excluding the label
column)
img_values = first_row[1:].values

# Reshape the pixel values into a 28x28 image (
assuming it's an image)
img = img_values.reshape(28, 28)

# Access the label (assuming the label is in the
first column)
label = first_row[0]

# Display the image and label
print(img)

print(label)

print(img.shape)

print(len(train_data))

print(len(test_data))

print(type(train_data))

classes = train_data['label'].unique()
```

```
 78
 79 print(classes)
 80
 81 device = "cuda" if torch.cuda.is_available() else "
    cpu"
 82
 83 print(device)
 84
 85 plt.imshow(img.squeeze())
 86
 87 plt.show()
 88
 89 plt.imshow(img.squeeze(), cmap="gray")
 90
 91 plt.show()
 92
 93 import torch
 94 import numpy as np
 95 import matplotlib.pyplot as plt
 96
 97 class_names = train_data['label'].unique()
 98 fig = plt.figure(figsize=(6, 8))
 99 rows = 4
100 cols = 4
101
102 for i in range(1, rows * cols + 1):
103     random_index = torch.randint(0, len(train_data
    ), size=[1]).item()
104
105     # Ensure that the random_index is within the
    valid range
106     random_index = random_index % len(train_data)
107
108     img, target = train_data.iloc[random_index, 1
    :], train_data.iloc[random_index, 0]
109     target = target - 1
110     fig.add_subplot(rows, cols, i)
111     plt.imshow(img.values.reshape(28, 28), cmap="
    gray")
112     plt.title(class_names[target])
113     plt.axis(False)
```

```
114
115 plt.tight_layout()
116 plt.show()
117
118
119
120 def accuracy(y_true, y_pred):
121   correct_classification=torch.eq(y_true,y_pred).sum
    ().item()
122   acc=(correct_classification/len(y_pred))*100
123   return acc
124
125
126 import torch
127 import torch.nn as nn
128 import torch.optim as optim
129 from torch.utils.data import DataLoader
130 from tqdm import tqdm
131
132
133 class Model(nn.Module):
134     def __init__(self, input_shape, hidden_units,
    output_shape):
135         super(Model, self).__init__()
136         self.input_shape = input_shape
137         self.layer_stack = nn.Sequential(
138             nn.Flatten(),
139             nn.Linear(in_features=input_shape,
    out_features=hidden_units),
140             nn.ReLU(),
141             nn.Linear(in_features=hidden_units,
    out_features=output_shape)
142         )
143
144     def forward(self, x):
145         return self.layer_stack(x)
146
147 print(train_data['label'].unique())
148
149 train_data['label'] = train_data['label'] - 1
150 # Instantiate the model
```

```
model = Model(input_shape=28 * 28, hidden_units=64, output_shape=len(classes))

optimizer = optim.Adam(model.parameters(), lr=0.01)


# Choose device
device = "cuda" if torch.cuda.is_available() else "cpu"

model.to(device)

# Loss function and optimizer
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
transform = ToTensor()
train_dataset = CustomDataset(dataframe=train_data, transform=transform)
train_dataLoader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# Training function
def train_step(model, data_loader, loss_fn, optimizer, accuracy, device):
    train_loss, train_acc = 0, 0
    model.train()

    for batch, (X, y) in enumerate(data_loader):
        X, y = X.to(device), y.to(device)

        optimizer.zero_grad()
        y_pred = model(X)

        loss = loss_fn(y_pred, y)
        train_loss += loss.item()
        train_acc += accuracy(y, y_pred.argmax(dim=1))

        loss.backward()
        optimizer.step()

```

```
186         if batch % 100 == 0:
187             print(f"Batch: {batch}/{len(data_loader)
    }, Loss: {loss.item():.4f}")
188 
189 
190 
191     train_loss /= len(data_loader)
192     train_acc /= len(data_loader)
193 
194     print(f"Train acc :{train_acc:.3f} | Train Loss:
    {train_loss:.3f}\n")
195 
196 # Testing function (similar to the training loop)
197 
198 def test_step(model, data_loader, loss_fn, accuracy
    , device):
199     test_loss, test_acc = 0, 0
200     model.eval()
201 
202     with torch.no_grad():
203         for X, y in data_loader:
204             X, y = X.to(device), y.to(device)
205 
206             y_pred = model(X)
207             loss = loss_fn(y_pred, y)
208             test_loss += loss.item()
209             test_acc += accuracy(y, y_pred.argmax(
    dim=1))
210 
211     test_loss /= len(data_loader)
212     test_acc /= len(data_loader)
213 
214     print(f"Test acc :{test_acc:.3f} | Test Loss:{
    test_loss:.3f}\n")
215 
216 
217 # Training loop
218 epochs = 5
219 
220 for epoch in tqdm(range(epochs)):
221     print(f"Epoch : {epoch}\n")
```

```
222     train_step(model, train_dataLoader,
    loss_function, optimizer, accuracy, device)
223
224 from cvzone.ClassificationModule import Classifier
225
226 class ClassificationModule:
227     def __init__(self, model, data_loader, device):
228         self.model = model
229         self.data_loader = data_loader
230         self.device = device
231
232     def classify(self):
233         predictions = []
234         ground_truth = []
235
236         self.model.eval()
237         self.model.to(self.device)
238
239         with torch.no_grad():
240             for X, y in self.data_loader:
241                 X, y = X.to(self.device), y.to(self.
    device)
242
243                 y_pred = self.model(X)
244                 predictions.extend(y_pred.argmax(dim
    =1).tolist())
245                 ground_truth.extend(y.tolist())
246
247         return predictions, ground_truth
248
249 # Usage
250 classifier = ClassificationModule(model,
    test_dataLoader, device)
251 predicted, actual = classifier.classify()
252
253 prediction, index = classifier.classify()
254 print(prediction, index)
255
256 import torch
257 import torchvision
258 import torch.nn as nn
```

```
259 from torchvision import transforms
260 import cv2
261 import os
262
263
264
265 # Function to classify images
266 def classify_image(img_path):
267     img = cv2.imread(img_path, 0)  # Read the image
    in grayscale
268     img = cv2.resize(img, (28, 28))  # Resize the
    image to match the MNIST dataset
269     img = transforms.ToTensor()(img).unsqueeze(0)
    # Convert image to tensor
270     model.eval()  # Set model to evaluation mode
271
272     with torch.no_grad():
273         output = model(img)
274         prediction = output.argmax(dim=1).item()
275     return prediction
276
277
278
279 import cv2
280 import time
281 from cvzone.HandTrackingModule import HandDetector
282 cap = cv2.VideoCapture(0)
283 detector = HandDetector(maxHands=1)
284 folder = "Data/A"
285 counter = 0
286 #labels = ["A", "B", "C"]
287 while True:
288     success, img = cap.read()
289     hands, img = detector.findHands(img)
290     if hands:
291         hand = hands[0]
292         x, y, w, h = hand['bbox']
293         img_crop = img[y:y + h, x:x + w]
294         if img_crop.shape[0] > 0 and img_crop.shape[
    1] > 0:
295             cv2.imshow("ImageCrop", img_crop)
```

```
296             predicted_label = classify_image(img_crop
    )
297             print(f"Predicted Label: {predicted_label
    }")
298 
299     cv2.imshow("Image", img)
300     key = cv2.waitKey(1)
301     if key == ord("q"):
302         counter += 1
303         cv2.imwrite(f'{folder}/Image_{time.time()}.
    jpg', img_crop)
304         print(counter)
305 
306 
307 
308 # Classify images in the folder
309 
310 
311 folder_path = 'C:\\Users\\91965\\PycharmProjects\\
    Signlanguagetranslation\\Data\\A'
312 
313 
314 for filename in os.listdir(folder_path):
315     if filename.endswith('.jpg') or filename.
    endswith('.png'):
316         image_path = os.path.join(folder_path,
    filename)
317         predicted_label = classify_image(image_path)
318         print(f"Image: {filename}, Predicted Label:
    {predicted_label}")
319 cap.release()
320 
321 
322 
```