```
’’’
USAGE:
python cam_test.py
’’’

import torch
import joblib
import torch.nn as nn
import numpy as np
import cv2
import argparse
import torch.nn.functional as F
import time
import cnn_models

from torchvision import models

# load label binarizer
lb = joblib.load(’../outputs/lb.pkl’)

model = cnn_models.CustomCNN()
# model = cnn_models.CustomCNN().cuda()

model.load_state_dict(torch.load(’../outputs/model.pth’))
print(model)
print(’Model loaded’)


def hand_area(img):
    hand = img[100:324, 100:324]
    hand = cv2.resize(hand, (224, 224))
    return hand


cap = cv2.VideoCapture(0)

if (cap.isOpened() == False):
    print(’Error while trying to open camera. Plese check again...’)

# get the frame width and height
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

# define codec and create VideoWriter object
out = cv2.VideoWriter(’../outputs/asl.mp4’,
                      cv2.VideoWriter_fourcc(*’mp4v’), 30, (frame_width, frame_height))

# read until end of video
while(cap.isOpened()):
    # capture each frame of the video
    ret, frame = cap.read()
    # get the hand area on the video capture screen
    cv2.rectangle(frame, (100, 100), (324, 324), (20, 34, 255), 2)
    hand = hand_area(frame)
    image = hand

    image = np.transpose(image, (2, 0, 1)).astype(np.float32)
    image = torch.tensor(image, dtype=torch.float)
    # image = torch.tensor(image, dtype=torch.float).cuda()

    image = image.unsqueeze(0)

    outputs = model(image)
    _, preds = torch.max(outputs.data, 1)
    # print(’PREDS’, preds)
    # print(f"Predicted output: {lb.classes_[preds]}")

    cv2.putText(frame, lb.classes_[preds], (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
    cv2.imshow(’image’, frame)
    out.write(frame)

    # time.sleep(0.09)

    # press ‘q‘ to exit
    if cv2.waitKey(27) & 0xFF == ord(’q’):
        break

# release VideoCapture()
cap.release()

# close all frames and video windows
cv2.destroyAllWindows()
```

cnn_models.py

```
import torch.nn as nn
import torch.nn.functional as F
import joblib

from torchvision import models

# load the binarized labels
print(’Loading label binarizer...’)
lb = joblib.load(’../outputs/lb.pkl’)

class CustomCNN(nn.Module):
    def __init__(self):
        super(CustomCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 5)
        self.conv2 = nn.Conv2d(16, 32, 5)
        self.conv3 = nn.Conv2d(32, 64, 3)
        self.conv4 = nn.Conv2d(64, 128, 5)

        self.fc1 = nn.Linear(128, 256)
        self.fc2 = nn.Linear(256, len(lb.classes_))

        self.pool = nn.MaxPool2d(2, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        bs, _, _, _ = x.shape
        x = F.adaptive_avg_pool2d(x, 1).reshape(bs, -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

create_csv.py

```
import pandas as pd
import numpy as np
import os
import joblib

from sklearn.preprocessing import LabelBinarizer
from tqdm import tqdm
from imutils import paths

# get all the image paths
image_paths = list(paths.list_images(’../input/preprocessed_image’))

# create a DataFrame
data = pd.DataFrame()

labels = []
for i, image_path in tqdm(enumerate(image_paths), total=len(image_paths)):
    label = image_path.split(os.path.sep)[-2]
    # save the relative path for mapping image to target
    data.loc[i, ’image_path’] = image_path

    labels.append(label)

labels = np.array(labels)
# one hot encode the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)

print(f"The first one hot encoded labels: {labels[0]}")
print(f"Mapping the first one hot encoded label to its category: {lb.classes_[0]}")
print(f"Total instances: {len(labels)}")

for i in range(len(labels)):
    index = np.argmax(labels[i])
    data.loc[i, ’target’] = int(index)

# shuffle the dataset
data = data.sample(frac=1).reset_index(drop=True)

# save as CSV file
data.to_csv(’../input/data.csv’, index=False)

# pickle the binarized labels
print(’Saving the binarized labels as pickled file’)
joblib.dump(lb, ’../outputs/lb.pkl’)

print(data.head(10))
```

```
’’’
USAGE:
python preprocess_image.py --num-images 1200
’’’

import pandas as pd
import os
import cv2
import random
import albumentations
import numpy as np
import argparse

from imutils import paths
from tqdm import tqdm

parser = argparse.ArgumentParser()
parser.add_argument(’-n’, ’--num-images’, default=1000, type=int,
    help=’number of images to preprocess for each category’)
args = vars(parser.parse_args())

print(f"Preprocessing {args[’num_images’]} from each category...")

# get all the image paths
image_paths = list(paths.list_images(’../input/asl_alphabet_train/asl_alphabet_train’))
dir_paths = os.listdir(’../input/asl_alphabet_train/asl_alphabet_train’)
dir_paths.sort()

root_path = ’../input/asl_alphabet_train/asl_alphabet_train’

# get 1000 images from each category
for idx, dir_path  in tqdm(enumerate(dir_paths), total=len(dir_paths)):
    all_images = os.listdir(f"{root_path}/{dir_path}")
    os.makedirs(f"../input/preprocessed_image/{dir_path}", exist_ok=True)
    for i in range(args[’num_images’]): # how many images to preprocess for each category
        # generate a random id between 0 and 2999
        rand_id = (random.randint(0, 2999))
        image = cv2.imread(f"{root_path}/{dir_path}/{all_images[rand_id]}")
        image = cv2.resize(image, (224, 224))

        cv2.imwrite(f"../input/preprocessed_image/{dir_path}/{dir_path}{i}.jpg", image)

print(’DONE’)
```

```
’’’
USAGE:
python test.py --img A_test.jpg
’’’

import torch
import joblib
import torch.nn as nn
import numpy as np
import cv2
import argparse
import albumentations
import torch.nn.functional as F
import time
import cnn_models

# construct the argument parser and parse the arguments
parser = argparse.ArgumentParser()
parser.add_argument(’-i’, ’--img’, default=’A_test.jpg’, type=str,
    help=’path for the image to test on’)
args = vars(parser.parse_args())

# load label binarizer
lb = joblib.load(’../outputs/lb.pkl’)

aug = albumentations.Compose([
                albumentations.Resize(224, 224, always_apply=True),
])

# model = models.MobineNetV2(pretrained=False, requires_grad=False)
model = cnn_models.CustomCNN()
# model = cnn_models.CustomCNN().cuda()

model.load_state_dict(torch.load(’../outputs/model.pth’))
print(model)
print(’Model loaded’)

image = cv2.imread(f"../input/asl_alphabet_test/{args[’img’]}")
image_copy = image.copy()

image = aug(image=np.array(image))[’image’]
image = np.transpose(image, (2, 0, 1)).astype(np.float32)
image = torch.tensor(image, dtype=torch.float)
# image = torch.tensor(image, dtype=torch.float).cuda()

image = image.unsqueeze(0)
print(image.shape)

start = time.time()
outputs = model(image)
_, preds = torch.max(outputs.data, 1)
print(’PREDS’, preds)
print(f"Predicted output: {lb.classes_[preds]}")
end = time.time()
print(f"{(end-start):.3f} seconds")
```

```
cv2.putText(image_copy, lb.classes_[preds], (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
cv2.imshow(’image’, image_copy)
cv2.imwrite(f"../outputs/{args[’img’]}", image_copy)
cv2.waitKey(0)
```

```
’’’
USAGE:
python train.py --epochs 10
’’’

import pandas as pd
import joblib
import numpy as np
import torch
import random
import albumentations
import matplotlib.pyplot as plt
import argparse
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision.transforms as transforms
import time
import cv2
import cnn_models

from PIL import Image
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader

# construct the argument parser and parse the arguments
parser = argparse.ArgumentParser()
parser.add_argument(’-e’, ’--epochs’, default=10, type=int,
    help=’number of epochs to train the model for’)
args = vars(parser.parse_args())

’’’ SEED Everything ’’’
def seed_everything(SEED=42):
    random.seed(SEED)
    np.random.seed(SEED)
    torch.manual_seed(SEED)
    torch.cuda.manual_seed(SEED)
    torch.cuda.manual_seed_all(SEED)
    torch.backends.cudnn.benchmark = True
SEED=42
seed_everything(SEED=SEED)
’’’ SEED Everything ’’’


# set computation device
device = (’cuda:0’ if torch.cuda.is_available() else ’cpu’)
# device =(torch.cuda.get_device_properties(0) if torch.cuda.is_available() else ’CPU’)


print(f"Computation device: {device}")

# read the data.csv file and get the image paths and labels
df = pd.read_csv(’../input/data.csv’)
X = df.image_path.values
```

```
y = df.target.values

(xtrain, xtest, ytrain, ytest) = (train_test_split(X, y,
                        test_size=0.15, random_state=42))

print(f"Training on {len(xtrain)} images")
print(f"Validationg on {len(xtest)} images")

# image dataset module
class ASLImageDataset(Dataset):
    def __init__(self, path, labels):
        self.X = path
        self.y = labels

        # apply augmentations
        self.aug = albumentations.Compose([
            albumentations.Resize(224, 224, always_apply=True),
        ])

    def __len__(self):
        return (len(self.X))

    def __getitem__(self, i):
        # image = Image.open(self.X[i])
        image = cv2.imread(self.X[i])
        image = self.aug(image=np.array(image))[’image’]
        image = np.transpose(image, (2, 0, 1)).astype(np.float32)
        label = self.y[i]

        return torch.tensor(image, dtype=torch.float), torch.tensor(label, dtype=torch.long)

train_data = ASLImageDataset(xtrain, ytrain)
test_data = ASLImageDataset(xtest, ytest)

# dataloaders
trainloader = DataLoader(train_data, batch_size=32, shuffle=True)
testloader = DataLoader(test_data, batch_size=32, shuffle=False)

# model = models.MobineNetV2(pretrained=True, requires_grad=False)
model = cnn_models.CustomCNN().to(device)
print(model)

# Find total parameters and trainable parameters
total_params = sum(p.numel() for p in model.parameters())
print(f"{total_params:,} total parameters.")
total_trainable_params = sum(
    p.numel() for p in model.parameters() if p.requires_grad)
print(f"{total_trainable_params:,} training parameters.")

# optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)
# loss function
criterion = nn.CrossEntropyLoss()

#validation function
```

```
def validate(model, dataloader):
    print(’Validating’)
    model.eval()
    running_loss = 0.0
    running_correct = 0
    with torch.no_grad():
        for i, data in tqdm(enumerate(dataloader), total=int(len(test_data)/dataloader.batch_size)):
            data, target = data[0].to(device), data[1].to(device)
            outputs = model(data)
            loss = criterion(outputs, target)

            running_loss += loss.item()
            _, preds = torch.max(outputs.data, 1)
            running_correct += (preds == target).sum().item()

        val_loss = running_loss/len(dataloader.dataset)
        val_accuracy = 100. * running_correct/len(dataloader.dataset)
        print(f’Val Loss: {val_loss:.4f}, Val Acc: {val_accuracy:.2f}’)

        return val_loss, val_accuracy

# training function
def fit(model, dataloader):
    print(’Training’)
    model.train()
    running_loss = 0.0
    running_correct = 0
    for i, data in tqdm(enumerate(dataloader), total=int(len(train_data)/dataloader.batch_size)):
        data, target = data[0].to(device), data[1].to(device)
        optimizer.zero_grad()
        outputs = model(data)
        loss = criterion(outputs, target)
        running_loss += loss.item()
        _, preds = torch.max(outputs.data, 1)
        # print(preds)
        running_correct += (preds == target).sum().item()
        loss.backward()
        optimizer.step()

    train_loss = running_loss/len(dataloader.dataset)
    train_accuracy = 100. * running_correct/len(dataloader.dataset)

    print(f"Train Loss: {train_loss:.4f}, Train Acc: {train_accuracy:.2f}")

    return train_loss, train_accuracy

train_loss , train_accuracy = [], []
val_loss , val_accuracy = [], []
start = time.time()
for epoch in range(args[’epochs’]):
    print(f"Epoch {epoch+1} of {args[’epochs’]}")
    train_epoch_loss, train_epoch_accuracy = fit(model, trainloader)
    val_epoch_loss, val_epoch_accuracy = validate(model, testloader)
    train_loss.append(train_epoch_loss)
    train_accuracy.append(train_epoch_accuracy)
    val_loss.append(val_epoch_loss)
    val_accuracy.append(val_epoch_accuracy)
end = time.time()

print(f"{(end-start)/60:.3f} minutes")

# accuracy plots
plt.figure(figsize=(10, 7))
plt.plot(train_accuracy, color=’green’, label=’train accuracy’)
plt.plot(val_accuracy, color=’blue’, label=’validataion accuracy’)
plt.xlabel(’Epochs’)
plt.ylabel(’Accuracy’)
plt.legend()
plt.savefig(’../outputs/accuracy.png’)
plt.show()

# loss plots
plt.figure(figsize=(10, 7))
plt.plot(train_loss, color=’orange’, label=’train loss’)
plt.plot(val_loss, color=’red’, label=’validataion loss’)
plt.xlabel(’Epochs’)
plt.ylabel(’Loss’)
plt.legend()
plt.savefig(’../outputs/loss.png’)
plt.show()

# save the model to disk
print(’Saving model...’)
torch.save(model.state_dict(), ’../outputs/model.pth’)
```