HW4 P1 Image Classification

If you have any questions, please contact the TAs via discord.

Check GPU Type

```
!nvidia-smi
```

```
Tue Mar 28 21:53:47 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   63C    P8    11W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Get Data

Notes: if the links are dead, you can download the data directly from Kaggle and upload it to the workspace, or you can use the Kaggle API to directly download the data into colab.

```
# Download Link
# Link (Google Drive): https://drive.google.com/file/d/1aUvXeksCbe_8gvHGRXuEp84wpcDHwAw2/view?usp=sharing

!pip install gdown --upgrade
!gdown --id '1aUvXeksCbe_8gvHGRXuEp84wpcDHwAw2' --output food11.zip
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gdown in /usr/local/lib/python3.9/dist-packages (4.6.4)
Collecting gdown
  Downloading gdown-4.7.1-py3-none-any.whl (15 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from gdown) (3.10.2)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.9/dist-packages (from gdown) (2.27.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from gdown) (4.65.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.9/dist-packages (from gdown) (4.11.2)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from gdown) (1.16.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.9/dist-packages (from beautifulsoup4->gdown) (2.4)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests[socks]->gd
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests[socks]->gdown)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests[socks]->gdown) (2
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests[socks]->gdown) (3.4)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.9/dist-packages (from requests[socks]->gdown
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.6.4
    Uninstalling gdown-4.6.4:
      Successfully uninstalled gdown-4.6.4
Successfully installed gdown-4.7.1
/usr/local/lib/python3.9/dist-packages/gdown/cli.py:126: FutureWarning: Option `--id` was deprecated in version 4.3.1 and wi
  warnings.warn(
Downloading...
From (original): https://drive.google.com/uc?id=1aUvXeksCbe_8gvHGRXuEp84wpcDHwAw2
From (redirected): https://drive.google.com/uc?id=1aUvXeksCbe_8gvHGRXuEp84wpcDHwAw2&confirm=t&uuid=e0d2a350-8c02-4e2c-9a8f-9
To: /content/food11.zip
100% 1.16G/1.16G [00:12<00:00, 92.0MB/s]
```

```
! unzip food11.zip
```

```
      inflating: test/2592.jpg
      inflating: test/0770.jpg
      inflating: test/0801.jpg
      inflating: test/2244.jpg
      inflating: test/1795.jpg
      inflating: test/2735.jpg
      inflating: test/2732.jpg
      inflating: test/2563.jpg
      inflating: test/1927.jpg
      inflating: test/2242.jpg
      inflating: test/0316.jpg
      inflating: test/0421.jpg
      inflating: test/0510.jpg
      inflating: test/1295.jpg
      inflating: test/0122.jpg
      inflating: test/1895.jpg
      inflating: test/1599.jpg
      inflating: test/0687.jpg
      inflating: test/2918.jpg
      inflating: test/1082.jpg
      inflating: test/1688.jpg
      inflating: test/1414.jpg
      inflating: test/1336.jpg
      inflating: test/0840.jpg
      inflating: test/0987.jpg
      inflating: test/1745.jpg
      inflating: test/0656.jpg
      inflating: test/1974.jpg
      inflating: test/2696.jpg
      inflating: test/1272.jpg
      inflating: test/0759.jpg
      inflating: test/1873.jpg
      inflating: test/0521.jpg
      inflating: test/1348.jpg
      inflating: test/2723.jpg
      inflating: test/2127.jpg
      inflating: test/1500.jpg
      inflating: test/1015.jpg
      inflating: test/0941.jpg
      inflating: test/0908.jpg
      inflating: test/0777.jpg
      inflating: test/1698.jpg
      inflating: test/1255.jpg
      inflating: test/1862.jpg
      inflating: test/2051.jpg
      inflating: test/2075.jpg
      inflating: test/1371.jpg
      inflating: test/2304.jpg
      inflating: test/0789.jpg
      inflating: test/2600.jpg
      inflating: test/0626.jpg
      inflating: test/1221.jpg
      inflating: test/0624.jpg
      inflating: test/1208.jpg
      inflating: test/1595.jpg
      inflating: test/2622.jpg
      inflating: test/0219.jpg
      inflating: test/0576.jpg
```

Import Packages

```
_exp_name = "sample"


# Import necessary packages.
import numpy as np
import pandas as pd
import torch
import os
import torch.nn as nn
import torchvision.transforms as transforms
from PIL import Image
# "ConcatDataset" and "Subset" are possibly useful when doing semi-supervised learning.
from torch.utils.data import ConcatDataset, DataLoader, Subset, Dataset
from torchvision.datasets import DatasetFolder, VisionDataset
# This is for the progress bar.
```

```
from tqdm.auto import tqdm
import random


myseed = 6666  # set a random seed for reproducibility
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(myseed)
torch.manual_seed(myseed)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(myseed)
```

Transforms

Torchvision provides lots of useful utilities for image preprocessing, data *wrapping* as well as data augmentation.

Please refer to PyTorch official website for details about different transforms.

```
# Normally, We don't need augmentations in testing and validation.
# All we need here is to resize the PIL image and transform it into Tensor.
test_tfm = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
])

# However, it is also possible to use augmentation in the testing phase.
# You may use train_tfm to produce a variety of images and then test using ensemble methods
train_tfm = transforms.Compose([
    # Resize the image into a fixed shape (height = width = 128)
    transforms.Resize((128, 128)),
    # You may add some transforms here.

    # ToTensor() should be the last one of the transforms.
    transforms.ToTensor(),
])
```

Datasets

The data is labelled by the name, so we load images and label while calling '**getitem**'

```
class FoodDataset(Dataset):

    def __init__(self,path,tfm=test_tfm,files = None):
        super(FoodDataset).__init__()
        self.path = path
        self.files = sorted([os.path.join(path,x) for x in os.listdir(path) if x.endswith(".jpg")])
        if files != None:
            self.files = files

        self.transform = tfm

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

    def __getitem__(self,idx):
        fname = self.files[idx]
        im = Image.open(fname)
        im = self.transform(im)

        try:
            label = int(fname.split("/")[-1].split("_")[0])
        except:
            label = -1 # test has no label

        return im,label
```

Model

```
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
        # torch.nn.MaxPool2d(kernel_size, stride, padding)
        # input 維度 [3, 128, 128]
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),  # [64, 128, 128]
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [64, 64, 64]

            nn.Conv2d(64, 128, 3, 1, 1), # [128, 64, 64]
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [128, 32, 32]

            nn.Conv2d(128, 256, 3, 1, 1), # [256, 32, 32]
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [256, 16, 16]

            nn.Conv2d(256, 512, 3, 1, 1), # [512, 16, 16]
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),       # [512, 8, 8]

            nn.Conv2d(512, 512, 3, 1, 1), # [512, 8, 8]
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),       # [512, 4, 4]
        )
        self.fc = nn.Sequential(
            nn.Linear(512*4*4, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 11)
        )

    def forward(self, x):
        out = self.cnn(x)
        out = out.view(out.size()[0], -1)
        return self.fc(out)
```

Configurations

```
# "cuda" only when GPUs are available.
device = "cuda" if torch.cuda.is_available() else "cpu"

# Initialize a model, and put it on the device specified.
model = Classifier().to(device)

# The number of batch size.
batch_size = 64

# The number of training epochs.
n_epochs = 8

# If no improvement in 'patience' epochs, early stop.
patience = 5

# For the classification task, we use cross-entropy as the measurement of performance.
criterion = nn.CrossEntropyLoss()

# Initialize optimizer, you may fine-tune some hyperparameters such as learning rate on your own.
optimizer = torch.optim.Adam(model.parameters(), lr=0.0003, weight_decay=1e-5)
```

Dataloader

```
# Construct train and valid datasets.
# The argument "loader" tells how torchvision reads the data.
train_set = FoodDataset("./train", tfm=train_tfm)
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=0, pin_memory=True)
valid_set = FoodDataset("./valid", tfm=test_tfm)
valid_loader = DataLoader(valid_set, batch_size=batch_size, shuffle=True, num_workers=0, pin_memory=True)
```

Start Training

```
# Initialize trackers, these are not parameters and should not be changed
stale = 0
best_acc = 0

for epoch in range(n_epochs):

    # ---------- Training ----------
    # Make sure the model is in train mode before training.
    model.train()

    # These are used to record information in training.
    train_loss = []
    train_accs = []

    for batch in tqdm(train_loader):

        # A batch consists of image data and corresponding labels.
        imgs, labels = batch
        #imgs = imgs.half()
        #print(imgs.shape,labels.shape)

        # Forward the data. (Make sure data and model are on the same device.)
        logits = model(imgs.to(device))

        # Calculate the cross-entropy loss.
        # We don't need to apply softmax before computing cross-entropy as it is done automatically.
        loss = criterion(logits, labels.to(device))

        # Gradients stored in the parameters in the previous step should be cleared out first.
        optimizer.zero_grad()

        # Compute the gradients for parameters.
        loss.backward()

        # Clip the gradient norms for stable training.
        grad_norm = nn.utils.clip_grad_norm_(model.parameters(), max_norm=10)

        # Update the parameters with computed gradients.
        optimizer.step()

        # Compute the accuracy for current batch.
        acc = (logits.argmax(dim=-1) == labels.to(device)).float().mean()

        # Record the loss and accuracy.
        train_loss.append(loss.item())
        train_accs.append(acc)

    train_loss = sum(train_loss) / len(train_loss)
    train_acc = sum(train_accs) / len(train_accs)

    # Print the information.
    print(f"[ Train | {epoch + 1:03d}/{n_epochs:03d} ] loss = {train_loss:.5f}, acc = {train_acc:.5f}")

    # ---------- Validation ----------
    # Make sure the model is in eval mode so that some modules like dropout are disabled and work normally.
    model.eval()

    # These are used to record information in validation.
    valid_loss = []
    valid_accs = []

    # Iterate the validation set by batches.
    for batch in tqdm(valid_loader):

        # A batch consists of image data and corresponding labels.
```

```
            imgs, labels = batch
            #imgs = imgs.half()

            # We don't need gradient in validation.
            # Using torch.no_grad() accelerates the forward process.
            with torch.no_grad():
                logits = model(imgs.to(device))

            # We can still compute the loss (but not the gradient).
            loss = criterion(logits, labels.to(device))

            # Compute the accuracy for current batch.
            acc = (logits.argmax(dim=-1) == labels.to(device)).float().mean()

            # Record the loss and accuracy.
            valid_loss.append(loss.item())
            valid_accs.append(acc)
            #break

        # The average loss and accuracy for entire validation set is the average of the recorded values.
        valid_loss = sum(valid_loss) / len(valid_loss)
        valid_acc = sum(valid_accs) / len(valid_accs)

        # Print the information.
        print(f"[ Valid | {epoch + 1:03d}/{n_epochs:03d} ] loss = {valid_loss:.5f}, acc = {valid_acc:.5f}")


        # update logs
        if valid_acc > best_acc:
            with open(f"./{_exp_name}_log.txt","a"):
                print(f"[ Valid | {epoch + 1:03d}/{n_epochs:03d} ] loss = {valid_loss:.5f}, acc = {valid_acc:.5f} -> best")
        else:
            with open(f"./{_exp_name}_log.txt","a"):
                print(f"[ Valid | {epoch + 1:03d}/{n_epochs:03d} ] loss = {valid_loss:.5f}, acc = {valid_acc:.5f}")


        # save models
        if valid_acc > best_acc:
            print(f"Best model found at epoch {epoch}, saving model")
            torch.save(model.state_dict(), f"{_exp_name}_best.ckpt") # only save best to prevent output memory exceed error
            best_acc = valid_acc
            stale = 0
        else:
            stale += 1
            if stale > patience:
                print(f"No improvment {patience} consecutive epochs, early stopping")
                break
```

```
100%                                      157/157 [01:48<00:00, 2.20it/s]
[ Train | 001/008 ] loss = 1.86637, acc = 0.34703
100%                                      57/57 [00:29<00:00, 2.02it/s]
[ Valid | 001/008 ] loss = 1.91723, acc = 0.33631
[ Valid | 001/008 ] loss = 1.91723, acc = 0.33631 -> best
Best model found at epoch 0, saving model
100%                                      157/157 [01:37<00:00, 1.86it/s]
[ Train | 002/008 ] loss = 1.52216, acc = 0.47771
100%                                      57/57 [00:28<00:00, 2.09it/s]
[ Valid | 002/008 ] loss = 1.53888, acc = 0.47872
[ Valid | 002/008 ] loss = 1.53888, acc = 0.47872 -> best
Best model found at epoch 1, saving model
100%                                      157/157 [01:38<00:00, 1.82it/s]
[ Train | 003/008 ] loss = 1.31692, acc = 0.54628
100%                                      57/57 [00:29<00:00, 2.41it/s]
[ Valid | 003/008 ] loss = 1.59201, acc = 0.47628
[ Valid | 003/008 ] loss = 1.59201, acc = 0.47628
100%                                      157/157 [01:37<00:00, 1.63it/s]
[ Train | 004/008 ] loss = 1.13490, acc = 0.60788
100%                                      57/57 [00:30<00:00, 1.66it/s]
[ Valid | 004/008 ] loss = 1.47096, acc = 0.49661
[ Valid | 004/008 ] loss = 1.47096, acc = 0.49661 -> best
Best model found at epoch 3, saving model
```

Dataloader for test

```
100%                                      57/57 [00:28<00:00, 2.13it/s]
```

```
# Construct test datasets.
# The argument "loader" tells how torchvision reads the data.
test_set = FoodDataset("./test", tfm=test_tfm)
test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=False, num_workers=0, pin_memory=True)
```

```
[ Train | 006/008 ] loss = 0.88167, acc = 0.69248
```

Testing and generate prediction CSV

```
Best model found at epoch 5, saving model
```

```
model_best = Classifier().to(device)
model_best.load_state_dict(torch.load(f"{_exp_name}_best.ckpt"))
model_best.eval()
prediction = []
with torch.no_grad():
    for data,_ in tqdm(test_loader):
        test_pred = model_best(data.to(device))
        test_label = np.argmax(test_pred.cpu().data.numpy(), axis=1)
        prediction += test_label.squeeze().tolist()
```

```
100%                                      47/47 [00:24<00:00, 2.26it/s]
[ Valid | 008/008 ] loss = 1.29389, acc = 0.59324
```

```
# create test csv
def pad4(i):
    return "0"*(4-len(str(i)))+str(i)
df = pd.DataFrame()
df["Id"] = [pad4(i) for i in range(len(test_set))]
df["Category"] = prediction
df.to_csv("submission.csv",index = False)
```

Q1. Augmentation Implementation

Implement augmentation by finishing train_tfm in the code with image size of your choice.

Your train_tfm must be capable of producing 5+ different results when given an identical image multiple times.

Your train_tfm in the report can be different from train_tfm in your training code.

```
train_tfm = transforms.Compose([
    # Resize the image into a fixed shape (height = width = 128)
    transforms.Resize((128, 128)),
    # You can add some transforms here.
    transforms.RandomRotation(degrees=(0,180)), # Random rotate the image with degree of 0 to 180
    transforms.ToTensor(),
])
```

Q2. Visual Representations Implementation

Visualize the learned visual representations of the CNN model on the validation set by implementing t-SNE (t-distributed Stochastic Neighbor Embedding) on the output of both top & mid layers (You need to submit 2 images).

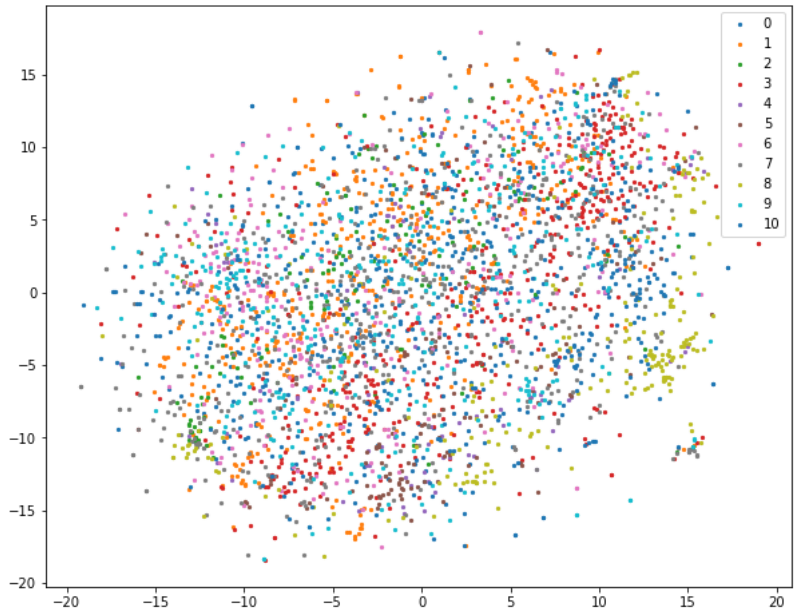
```
import torch
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from tqdm import tqdm
import matplotlib.cm as cm
import torch.nn as nn

device = 'cuda' if torch.cuda.is_available() else 'cpu'

# Load the trained model
model = Classifier().to(device)
state_dict = torch.load(f"{_exp_name}_best.ckpt")
model.load_state_dict(state_dict)
model.eval()

print(model)
```

```
Classifier(
  (cnn): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU()
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): ReLU()
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=8192, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=11, bias=True)
  )
)
```

```
# Load the vaildation set
valid_set = FoodDataset("./valid", tfm=test_tfm)
valid_loader = DataLoader(valid_set, batch_size=64, shuffle=False, num_workers=0, pin_memory=True)

# Extract the representations for the specific layer of model
index = 12 # You should find out the index of layer which is defined as "top" or 'mid' layer of your model. # top_layer=20, mid_la
features = []
labels = []
for batch in tqdm(valid_loader):
    imgs, lbls = batch
```

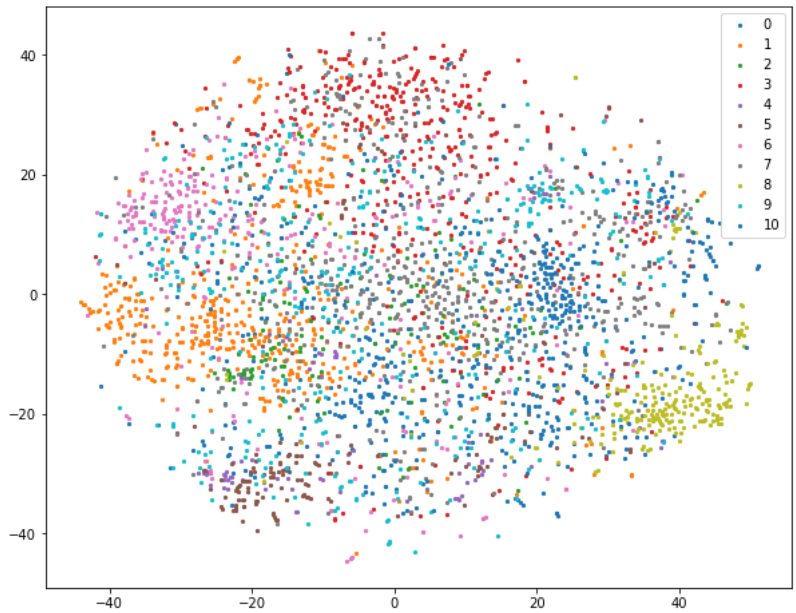
```
    with torch.no_grad():
        logits = model.cnn[:index](imgs.to(device))
        logits = logits.view(logits.size()[0], -1)
    labels.extend(lbls.cpu().numpy())
    logits = np.squeeze(logits.cpu().numpy())
    features.extend(logits)

features = np.array(features)
colors_per_class = cm.rainbow(np.linspace(0, 1, 11))

# Apply t-SNE to the features
features_tsne = TSNE(n_components=2, init='pca', random_state=42).fit_transform(features)

# Plot the t-SNE visualization
plt.figure(figsize=(10, 8))
for label in np.unique(labels):
    plt.scatter(features_tsne[labels == label, 0], features_tsne[labels == label, 1], label=label, s=5)
plt.legend()
plt.show()
```

```
100%|██████████| 57/57 [00:28<00:00,  2.02it/s]
```




```
# Load the vaildation set
valid_set = FoodDataset("./valid", tfm=test_tfm)
valid_loader = DataLoader(valid_set, batch_size=64, shuffle=False, num_workers=0, pin_memory=True)

# Extract the representations for the specific layer of model
index = 20 # You should find out the index of layer which is defined as "top" or 'mid' layer of your model. # top_layer=20, mid_la
features = []
labels = []
for batch in tqdm(valid_loader):
    imgs, lbls = batch
    with torch.no_grad():
        logits = model.cnn[:index](imgs.to(device))
        logits = logits.view(logits.size()[0], -1)
    labels.extend(lbls.cpu().numpy())
    logits = np.squeeze(logits.cpu().numpy())
    features.extend(logits)

features = np.array(features)
colors_per_class = cm.rainbow(np.linspace(0, 1, 11))

# Apply t-SNE to the features
features_tsne = TSNE(n_components=2, init='pca', random_state=42).fit_transform(features)

# Plot the t-SNE visualization
plt.figure(figsize=(10, 8))
for label in np.unique(labels):
    plt.scatter(features_tsne[labels == label, 0], features_tsne[labels == label, 1], label=label, s=5)
plt.legend()
plt.show()
```

```
100%|██████████| 57/57 [00:28<00:00,  2.02it/s]
```




```
!ls
```

```
food11.zip        sample_data     submission.csv  train
sample_best.ckpt  sample_log.txt  test            valid
```

```
!ls valid | wc -l
```

```
3643
```

```
#!column -s, -t < submission.csv | less -#2 -N -S
```

✓ 1m 5s completed at 6:36 PM